Πτυχιακή εργασία……..

Τίτλος: Πρόγραμμα επίλυσης δικτυωμάτων σε Java

Φοιτητής: Μιχαλάκης Κωνσταντίνος

Εισηγητής: Παπαδάκης Νικόλαος

Περιεχόμενα

1 Εισαγωγή

1.1 Σκοπός της πτυχιακής εργασίας

Στόχος της πτυχιακής είναι η «σύνδεση» της θεωρίας της επίλυσης ενός δικτυώματος με τη χρήση της μεθόδου των μετατοπίσεων με τον προγραμματισμό με τη χρήση της γλώσσας προγραμματισμού Java.

Το προγραμμα θα πρέπει να δίνει τις ακόλουθες δυνατότητες στον χρήστη:

- Εισάγωγη δεδομένων ενός δισδιάστατου δικτυώματος όπως: συντεταγμένες των κόμβων, στηρίξεις στους κόμβους, υλικό και διατομή των ράβδων, ασκούμενες δυνάμεις στους κόμβους κ.α.
- Επιλύσει το δικτύωματα με την μέθοδο των μετατοπίσεων των κόμβων
- Να κάνει μία βασική γραφική αναπαράσταση του δικτυώματος πριν και μέτα την επίλυση του

Για να γραφτεί ο κώδικας του προγράμματος θα χρησιμοποιήθηκε το ολοκληρωμένο περιβάλλον προγραμματισμού Eclipse (http://www.eclipse.org/) .

2 Προγραμματισμός Η/Υ

2.1 Τι είναι προγραμματισμός;

Προγραμματισμός είναι η διαδικασία μετατροπής ενός αλγόριθμου σε πρόγραμμα. Είναι δηλαδή η μετατροπή του αλγόριθμου σε μορφή τέτοια, ώστε να είναι κατανοητή από τον υπολογιστή. Πρόγραμμα είναι το σύνολο των εντολών που πρέπει να δοθούν στον υπολογιστή για να υλοποιηθεί ένας αλγόριθμος για την επίλυση ενός προβλήματος. Το πρόγραμμα δεν αποτελείται μόνο από τις εντολές αλλά και από τα δεδομένα και τις δομές δεδομένων στις οποίες ενεργεί. (Αλγόριθμοι και δομές δεδομένων είναι μία αδιάσπαστη ενότητα.)

2.1.1 Γλώσσα προγραμματισμού

Γλώσσα λέγεται μια τεχνητή γλώσσα που μπορεί να χρησιμοποιηθεί για τον έλεγχο μιας μηχανής, συνήθως ενός υπολογιστή. Οι γλώσσες προγραμματισμού (όπως άλλωστε και οι ανθρώπινες γλώσσες) ορίζονται από ένα σύνολο συντακτικών και εννοιολογικών κανόνων, που ορίζουν και τη δομή και το νόημα , αντίστοιχα, των προτάσεων της γλώσσας.

Οι γλώσσες προγραμματισμού χρησιμοποιούνται για να διευκολύνουν την οργάνωση και διαχείριση πληροφοριών, αλλά και για την ακριβή διατύπωση αλγορίθμων. Ορισμένοι ειδικοί χρησιμοποιούν τον όρο *γλώσσα προγραμματισμού* μόνο για τυπικές γλώσσες που μπορούν να εκφράσουν όλους τους πιθανούς αλγόριθμους.

Υπάρχουν χιλιάδες διαφορετικές γλώσσες προγραμματισμού και κάθε χρόνο δημιουργούνται περισσότερες.

2.1.2 Χαρακτηριστικά γλωσσών προγραμματισμού

Κάθε γλώσσα προγραμματισμού έχει το δικό της σύνολο τυπικών προδιαγραφών (ή κανόνων) που αφορούν το συντακτικό, το λεξιλόγιο και το νόημα της. Για πολλές γλώσσες που χρησιμοποιούνται ευρέως και έχουν χρησιμοποιηθεί για αρκετό χρονικό διάστημα (π.χ. C, C++, Java, Scheme), υπάρχουν ειδικοί φορείς τυποποίησης, οι οποίοι μέσα από τακτές συναντήσεις δημιουργούν, τροποποιούν ή επεκτείνουν τις τυπικές προδιαγραφές που διέπουν τη χρήση μιας γλώσσας προγραμματισμού. Άλλες γλώσσες δεν περιγράφονται σε κάποιο επίσημο πρότυπο αλλά ορίζονται μόνο με βάση κάποια υλοποίησή τους (που αποτελεί το ντε φάκτο πρότυπο), όπως η Python που περιγράφεται από την υλοποίηση CPython.

2.1.3 Κατηγοριοποίηση γλωσσών προγραμματισμού

Δεν υπάρχει απλός τρόπος να κατηγοριοποιηθούν οι γλώσσες προγραμματισμού. Αυτό συμβαίνει γιατί συνήθως κάθε γλώσσα προγραμματισμού περιέχει επιρροές από πολλές προηγούμενες γλώσσες, συνδυάζοντας θετικά στοιχεία και προσθέτοντας νέα. Χαρακτηριστικά που εμφανίζονται σε μια γλώσσα και έχουν θετική αποδοχή, συνήθως υιοθετούνται από μεταγενέστερες γλώσσες ακόμα και αν πρόκειται για γλώσσες που ανήκουν σε διαφορετική κατηγορία.

Η κατηγοριοποίηση είναι ακόμα πιο περίπλοκη για το λόγο ότι πολλές γλώσσες συνήθως ανήκουν σε παραπάνω από μια κατηγορίες. Για παράδειγμα, η Java είναι τόσο αντικειμενοστραφής όσο και παράλληλη γλώσσα, δεδομένου ότι υποστηρίζει την οργάνωση των δεδομένων και υπολογισμών σε αντικείμενα, αλλά επιτρέπει επίσης και την δημιουργία προγραμμάτων με ταυτόχρονα νήματα (threads) που εκτελούνται παράλληλα.

Δεδομένης της δυσκολίας στην κατηγοριοποίηση, οι γλώσσες προγραμματισμού μπορούν να κατηγοριοποιηθούν με διάφορους τρόπους. Οι συνηθέστεροι τρόποι είναι:

- Με βάση τον τρόπο οργάνωσης του προγράμματος
- Με βάση τον στόχο που έχει η γλώσσα
- Με βάση τον τρόπο που περιγράφουν το ζητούμενο αποτέλεσμα

Στην πρώτη περίπτωση προκύπτουν κατηγορίες όπως:

- **Διαδικαστικές γλώσσες (procedural)** όπου το πρόγραμμα είναι οργανωμένο σε διαδικασίες, που αποτελούνται από σειρές εντολών που περιγράφουν αλγόριθμους. Παραδείγματα γλωσσών που ανήκουν σε αυτήν την κατηγορία είναι η Pascal ή η C.
- **Αντικειμενοστραφείς γλώσσες (object-oriented)** όπου το πρόγραμμα είναι οργανωμένο σε αντικείμενα. Ένα αντικείμενο είναι μια μονάδα που αποτελείται από την περιγραφή κάποιων δεδομένων και την περιγραφή των αλγόριθμων που τα επεξεργάζονται. Ένα αντικείμενο είναι μια μονάδα που αποτελείται από την περιγραφή κάποιων δεδομένων και την περιγραφή των αλγόριθμων που τα επεξεργάζονται. Ένα αντικειμενοστραφών γλωσσών είναι η Java ή η C++.
- **Συναρτησιακές γλώσσες (fuctional)** όπου οι υπολογισμοί εκφράζονται ως εφαρμογές μαθηματικών συναρτήσεων, σε αντίθεση με τα άλλα είδη προγραμματισμού όπου οι υπολογισμοί εκφράζονται ως σειρές εντολών, όπου η κάθε μία αλλάζει με κάποιο τρόπο την κατάσταση του συστήματος. Θεωρητικό τους υπόβαθρο είναι ο λ-λογισμός. Χαρακτηριστικές συναρτησιακές γλώσσες είναι η Lisp, η Haskell και η OCaml.

Στην περίπτωση που οι κατηγοριοποίηση των γλωσσών γίνει με βάση το στόχο που έχει η γλώσσα, υπάρχουν οι παρακάτω κατηγορίες:

- Γλώσσες γενικής χρήσης. Σ’ αυτή την κατηγορία ταξινομούνται γλώσσες που δημιουργήθηκαν για τον προγραμματισμό γενικών εφαρμογών, καθώς και πολλές εκπαιδευτικές γλώσσες που αποδείχτηκαν χρήσιμες για την ανάπτυξη γενικών εφαρμογών, όπως η Pascal.
- Γλώσσες προγραμματισμού συστημάτων, που χρησιμοποιούνται συνήθως για το προγραμματισμό λειτουργικών συστημάτων ή οδηγών (drivers) υλικού, όπου χρειάζεται ο προγραμματιστής να έχει έλεγχο και γνώση του πως λειτουργεί το υλικό. Η πιο συχνά χρησιμοποιούμενη γλώσσα προγραμματισμού συστημάτων είναι η C.
- Γλώσσες σεναρίων (scripting). Αυτές οι γλώσσες χρησιμοποιούνται συνήθως για τη γρήγορη ανάπτυξη μικρών προγραμμάτων, σε περιπτώσεις που ο χρόνος του προγραμματιστή είναι πιο πολύτιμος από την ταχύτητα εκτέλεσης του προγράμματος, όπως για παράδειγμα συμβαίνει όταν το πρόγραμμα απλά αυτοματοποιεί απλές λειτουργίες. Παραδείγματα σεναριακών (scripting) γλωσσών είναι η Perl, η Python, η Ruby ή τα κελύφη του λειτουργικού συστήματος Unix (shells).
- Γλώσσες ειδικών εφαρμογών. Σε αυτή τη κατηγορία ανήκουν γλώσσες που αναπτύχθηκαν ειδικά για μια συγκεκριμένη εφαρμογή. Για παράδειγμα, η γλώσσα PostScript είναι σχεδιασμένη ειδικά για να περιγράφονται με λεπτομέρεια κείμενα προς εκτύπωση, ενώ η γλώσσα Matlab είναι σχεδιασμένη για την επεξεργασία πινάκων από αριθμητικά δεδομένα.
- Παράλληλες ή κατανεμημένες γλώσσες. Στη συγκεκριμένη κατηγορία ταξινομούνται γλώσσες που εκτρέπουν τη ανάπτυξη παράλληλων προγραμμάτων, όπου πολλές εντολές εκτελούνται ταυτόχρονα σε πολλούς υπολογιστές, έτσι ώστε το τελικό αποτέλεσμα να προκύψει γρηγορότερα. Οι παράλληλες γλώσσες προσφέρουν

συνήθων εύκολους τρόπους επικοινωνίας μεταξύ των νημάτων που εκτελούνται παράλληλα, καθώς και τρόπους ώστε να δημιουργούνται καινούργιες παράλληλες εκτελέσεις.
- Παραδείγματα γλωσσών που ανήκουν (και) σε αυτή την κατηγορία είναι η Java, η Erlang, η MultiLisp ή η Cilk.

Τέλος, στην περίπτωση που η κατηγοριοποίηση γίνεται με βάση τον τρόπο που περιγράφεται το ζητούμενο, υπάρχουν οι παρακάτω κατηγορίες:

- Προστακτικές γλώσσες προγραμματισμού (imperative) είναι οι γλώσσες που περιγράφουν το ζητούμενο αποτέλεσμα κατασκευαστικά, δίνοντας μια σειρά εντολών που όταν εκτελεστούν παράγουν το ζητούμενο αποτέλεσμα. Τέτοιες γλώσσες είναι η C, η Java αλλά και η OCaml.
- Δηλωτικές γλώσσες προγραμματισμού (declarative) είναι οι γλώσσες που περιγράφουν το ζητούμενο αποτέλεσμα χρησιμοποιώντας τις ιδιότητες που έχει, και όχι τον τρόπο με τον οποίο υπολογίζεται.
- Παράδειγμα δηλωτικών γλωσσών είναι η Haskell, η SQL και η Prolog.

2.2 Η γλώσσα προγραμματισμού JAVA

2.2.1 Ιστορική αναδρομή της γλώσσας JAVA

Η Java θεωρείται ότι είναι μια γλώσσα προγραμματισμού για εφαρμογές Internet. Πολλοί θεωρούν ότι η Java είναι μια γλώσσα προγραμματισμού γενικού σκοπού που μπορεί να χρησιμοποιηθεί χωρίς κανένα συσχετισμό με το Internet. Κατά την δημιουργία της, η Java δεν ήταν τίποτα από αυτά τα δύο αλλά τελικά εξελίχθηκε και στα δύο.

Η ιστορία της Java αρχίζει το 1991 όταν ο James Gosling και η ομάδα του στην εταιρεία Sun Microsystems άρχιζαν να σχεδιάζουν την πρώτη έκδοση μιας νέας γλώσσας προγραμματισμού που θα γινόταν η σημερινή Java, αν και τότε λεγόταν «Oak» πριν αλλάξει λόγο της ύπαρξης ήδη γλώσσας με αυτό το όνομα. Θα είχε στόχο να χρησιμοποιηθεί στον προγραμματισμό οικιακών συσκευών. Λόγο της μεγάλης ποικιλίας και της μεγάλης πολυπλοκότητας των οικιακών συσκευών αυτό το σχέδιο απέτυχε, όμως δεν ήταν και το οριστικό τέλος της Java.

Το 1994, ο Glosling συνειδητοποίησε ότι η γλώσσα του θα ήταν ιδανική για την ανάπτυξη ενός προγράμματος περιήγησης (Web browser) στο διαδίκτυο που θα μπορούσε να τρέχει προγράμματα (Java) μέσω του Internet. Ο Web browser δημιουργήθηκε από τους Patrick Noughton και Jonathan Payne στη Sun Microsystems και έχει εξελιχθεί στο πρόγραμμα που

σήμερα είναι γνωστό ως HotJava. Αυτή είναι η αρχή σύνδεσης της Java με το Internet. Το φθινόπωρο του 1995, η Netscape αποφάσισε να ανακοινώσει τον επόμενο Web browser της εταιρείας που είχε τη δυνατότητα να τρέχει προγράμματα Java. Οι άλλες εταιρείες που δραστηριοποιούνται με το Internet έχουν ακολουθήσει το παράδειγμα και έχουν αναπτύξει λογισμικό που εξυπηρετεί προγράμματα Java.

2.2.2 Αντικειμενοστρεφής Προγραμματισμός

Η Java είναι μια γλώσσα **αντικειμενοστραφούς προγραμματισμού (object-oriented programming)**, που εν συντομία γράφεται ΑΣΠ (και OOP στα αγγλικά). Ο ΑΣΠ είναι μια μεθοδολογία προγραμματισμού που θεωρεί ότι ένα πρόγραμμα αποτελείται από παρεμφερή αντικείμενα που αλληλεπιδρούν μεταξύ τους μέσω λειτουργιών.

Ο αντικειμενοστρεφής προγραμματισμός έχει τη δική του ορολογία. Τα αντικείμενα λέγονται, προφανέστατα, **αντικείμενα (objects)**. Οι λειτουργίες που μπορεί να πραγματοποιήσει ένα αντικείμενο λέγονται **μέθοδοι (methods).** Τα αντικείμενα του ίδιου είδους λέμε ότι έχουν το ίδιο τύπο (type) ή, πιο συχνά, ότι ανήκουν στην ίδια **κλάση (class).**

Η αντικειμενοστρεφής μεθοδολογία μπορεί να εφαρμοστεί σε οποιοδήποτε είδος προγράμματος υπολογιστή και δεν περιορίζεται μόνο σε προγράμματα προσομοίωσης. Ο αντικειμενοστρεφής προγραμματισμός χρησιμοποιεί κλάσεις και αντικείμενα ακολουθώντας συγκεκριμένες αρχές σχεδίασης. Οι παρακάτω είναι τρεις από τις κυριότερες αρχές σχεδίασης του αντικειμενοστραφούς προγραμματισμού:

- Ενθυλάκωση
- Πολυμορφισμός
- Κληρονομικότητα

- **Ενθυλάκωση (encapsulation)**: Είναι η διαδικασία της απόκρυψης (ενθυλάκωσης) όλων των λεπτομερειών σχετικά με το πώς είναι γραμμένο ένα τμήμα λογισμικού και της αναφοράς μόνον όσων χρειάζεται να γνωρίζει ο προγραμματιστής προκειμένου να μπορεί να χρησιμοποιεί το λογισμικό. Με άλλα λόγια, ενθυλάκωση είναι η διαδικασία της περιγραφής μια κλάσης ή ενός αντικειμένου δίνοντας μόνο τις απαραίτητες πληροφορίες που θα επιτρέψουν σε έναν προγραμματιστή να χρησιμοποιήσει την κλάση ή το αντικείμενο.
- **Πολυμορφισμός (polymorphism)**: Είναι η ιδιότητα κάποιου να έχει πολλές μορφές. Στο προγραμματισμό είναι ότι επιτρέπει στην ίδια εντολή προγράμματος να σημαίνει διαφορετικά πράγματα σε διαφορετικές καταστάσεις. Σε μια γλώσσα προγραμματισμού όπως η Java, πολυμορφισμός σημαίνει ότι ένα όνομα μεθόδου, που χρησιμοποιείται σαν εντολή, μπορεί να προκαλέσει διαφορετικές λειτουργίες, ανάλογα με το είδος του αντικειμένου που πραγματοποιεί τη λειτουργία.
- **Κληρονομικότητα (inheritance)**: Αναφέρεται σε έναν τρόπο οργάνωσης των κλάσεων. Σε γλώσσες προγραμματισμού όπως η Java η κληρονομικότητα χρησιμοποιείται για την

οργάνωση των κλάσεων. Αυτός ο τρόπος έχει το πλεονέκτημα ότι επιτρέπει στον προγραμματιστή να αποφεύγει την επανάληψη της ίδια ομάδας εντολών προγραμματισμού για κάθε κλάση. Γενικότερα στην Java, η κληρονομικότητα είναι ένας τρόπος οργάνωσης κλάσεων τέτοιος ώστε οι ιδιότητες να μπορούν να οριστούν μόνο μια φορά και να ισχύουν σε μια ολόκληρη ομάδα κλάσεων.

2.2.3 Applets

Υπάρχουν δύο είδη προγραμμάτων Java: τα applets και οι εφαρμογές. Μια εφαρμογή (application) είναι απλά ένα συνηθισμένο πρόγραμμα. Ένα applet είναι μια μικρή εφαρμογή ή μικροεφαρμογή. Τα applets και οι εφαρμογές είναι σχεδόν ίδια. Η διαφορά είναι ότι μια εφαρμογή ή μικροεφαρμογή προορίζεται να τρέξει στον υπολογιστή μας, όπως κάθε άλλο πρόγραμμα, ενώ ένα applet προορίζεται να σταλεί σε μια άλλη τοποθεσία μέσω Internet και να τρέξει εκεί.

2.2.4 Απλό παράδειγμα εφαρμογής Java

Ένα απλό παράδειγμα. Έχουμε το παρακάτω κομμάτι κώδικα:

```
Import java.util.*;
public class FirstProgram
{
  public static void main(String[] args)
  {

    System.out.println(“Hello out there.”);
    System.out.println(“I will add two numbers for you.”);
    System.out.println(“Enter two whole numbers on a line:”);

    int n1, n2;
    Scanner keyboard = new Scanner(System.in);
    n1 = keyboard.nextInt();
    n2 = keyboard.nextInt();
    System.out.println(“The sum of those two numbers is”);
    System.out.println(n1 + n2);
  }
}
```

Αν πληκτρολογήσουμε του ακέραιους αριθμούς (**π.χ. 10 και 20**), θα εμφανιστεί στην οθόνη του υπολογιστή:

```
Hello out there.

I will add two numbers for you.

Enter two whole numbers on a line

10 20

The sum of those two numbers is

30
```

2.2.5 Ανάλυση παραδείγματος

```
Import java.util.*;
```

Ένα **πακέτο (package)** είναι μια βιβλιοθήκη κλάσεων οι οποίες έχουν ήδη οριστεί για εμάς (μια κλάση είναι ένα τμήμα του λογισμικού το οποίο μπορεί να χρησιμοποιηθεί μέσα σε ένα πρόγραμμα). Το πρόγραμμα αυτό χρησιμοποιεί την κλάση `Scanner` και η κλάση `Scanner` ορίζεται στο πακέτο `java.util`.

```
puplic class FirstProgram
{
```

Αυτή η γραμμή του κώδικα ορίζει μια νέα ομάδα κλάσεων που την ονομάζει "FirstProgram". Το ανοικτό άγκιστρο σημαίνει ότι "εδώ είναι η αρχή της ομάδας αυτής".

```
public static void main(String[] args)
{
```

Στο παραπάνω παράδειγμα χρησιμοποιήθηκε μία μόνο κλάση. Βέβαια, ένα πρόγραμμα Java μπορεί να περιλαμβάνει πολλές κλάσεις, αλλά όταν εκτελούμε ένα πρόγραμμα Java τρέχουμε μόνο την κλάση που εμείς θεωρούμε ως πρόγραμμα. Είναι εύκολο να ξεχωρίσει κανείς αυτή τη κλάση επειδή θα περιέχει μία γραμμή κώδικα παραπλήσια με την παραπάνω.

Αυτή η γραμμή πιθανότητα (όχι απαραίτητα) θα βρίσκεται κοντά στην αρχή του αρχείου. Οι κρίσιμες λέξεις που πρέπει να ψάξει κανείς είναι `public static void main`. Το υπόλοιπο τμήμα της γραμμής μπορεί να διαφέρει λίγο σε κάποιες περιπτώσεις.

```
System.out.println("Hello out there.");
System.out.println("I will add two numbers for you.");
System.out.println("Enter two whole numbers on a line:");
```

Αυτές οι τρείς γραμμές είναι οι πρώτες ενέργειες που πραγματοποιεί το πρόγραμμα. Είναι ίδιες μεταξύ τους, με διαφορετικό αποτέλεσμα η κάθε μία.

Αρχίζουν όλες με `System.out.println` και η κάθε μία εμφανίζει σαν έξοδο το αλφαριθμητικό με τα εισαγωγικά που βρίσκεται μέσα στις παρενθέσεις. Για παράδειγμα η γραμμή:

```
System.out.println("Hello out there.");
```

Θα εμφανίσει στην οθόνη τη φράση σαν έξοδο:

```
Hello out there.
```

Πιο συγκεκριμένα το κομμάτι της εντολής " `System.out`" είναι ένα αντικείμενο που χρησιμοποιείται για να στέλνει την έξοδο στην οθόνη και η "`println`" είναι η μέθοδος (δηλαδή, η λειτουργία) που πραγματοποιεί αυτό το αντικείμενο προκειμένου να στείλει το περιεχόμενο των παρενθέσεων στην οθόνη.

Όταν λέμε ότι ένα αντικείμενο πραγματοποιεί μία λειτουργία χρησιμοποιώντας μία μέθοδο, τότε λέμε ότι **καλεί (invoke ή call)** τη μέθοδο. Η φράση ή οι φράσεις, ενδεχομένως,

μέσα στην παρένθεση λέγονται **ορίσματα (arguments)** και παρέχουν πληροφορίες που χρειάζεται η μέθοδος για να πραγματοποιήσει τη λειτουργία της.

```
int n1, n2;
```

Εδώ δηλώνονται οι μεταβλητές που θα χρησιμοποιηθούν. Τα n1 και n2 είναι τα ονόματα των μεταβλητών.

Η **μεταβλητή (variable)** είναι κάτι στο οποίο μπορεί να αποθηκευτεί ένα τμήμα δεδομένων. Η λέξη `int` υποδηλώνει ότι τα δεδομένα πρέπει να είναι ένας ακέραιος αριθμός.

```
Scanner keyboard = new Scanner(System.in);
```

Η εντολή αυτή δημιουργεί ένα αντικείμενο της κλάσης `Scanner` και του δίνει το όνομα `keyboard` (θα μπορούσε να είναι οποιοδήποτε μη δεσμευμένο όνομα). Αυτό το αντικείμενο μπορεί να χρησιμοποιηθεί από το πρόγραμμα για να δοθούν δεδομένα από το πληκτρολόγιο. Δηλαδή αυτή η γραμμή προετοιμάζει το πρόγραμμα ώστε να "κρατήσει" την επόμενη πληκτρολόγηση κάποιου δεδομένου, τον τύπο του οποίου ορίζουμε αμέσως μετά από αυτή τη γραμμή.

```
n1 = keyboard.nextInt();
n2 = keyboard.nextInt();
```

Αφού έχουν δηλωθεί οι μεταβλητές, οι γραμμές αυτές δηλώνουν στο πρόγραμμα να κρατήσει τις τιμές αρχικά για τη μεταβλητή `n1` και μετά για τη μεταβλητή `n2` τη πρώτη και δεύτερη ακέραια τιμή, αντίστοιχα, που θα δοθεί από το πληκτρολόγιο. Η γραμμές αυτές αναφέρονται σε συγκεκριμένο τύπο δεδομένων που θα κρατήσει το πρόγραμμα.

```
System.out.println(n1 + n2);
  }
}
```

Η πρώτη γραμμή καλεί το πρόγραμμα να εμφανίσει στην οθόνη το άθροισμα των μεταβλητών `n1` και `n2`. Προσοχή! Εδώ δεν έχει εισαγωγικά για να εμφανίσει τη φράση "`n1 + n2`" , αλλά να κάνει την πράξη και να εμφανίσει το αποτέλεσμα αυτής **μόνο**.

Το πρώτο άγκιστρο υποδηλώνει ότι η κύρια κλάση του προγράμματος τελειώνει εδώ. Το δεύτερο άγκιστρο υποδηλώνει ότι η ομάδα κλάσεων με τη σειρά της τελειώνει εδώ.

3 Βασικά στοιχεία της γλώσσας JAVA

Μία αναλυτική παρουσίαση βασικών στοιχείων της γλώσσας προγραμματισμού Java.

3.1 Πρωτογενείς τύποι και εκφράσεις

Όνομα Τύπου	Είδος Τιμής	Μνήμη σε Χρήση	Μέγεθος Περιοχής
byte	ακέραιος	1 byte	-128 ως 127
short	ακέραιος	2 bytes	-32768 ως 32767
int	ακέραιος	4 bytes	-2147483648 ως 2147483647
long	ακέραιος	8 bytes	-9223372036854775808 ως 9223372036854775807
float	δεκαδικός	4 bytes	$\pm 3.40282347 \times 10^{+38}$ ως $\pm 1.40239846 \times 10^{-45}$
double	δεκαδικός	8 bytes	$\pm 1.76769313486231570 \times 10^{+308}$ ως $\pm 4.94065645841246544 \times 10^{-324}$
char	ένας χαρακτήρας (unicode)	2 bytes	όλοι οι χαρακτήρες unicode
boolean	**true** ή **false**	1 bit	δεν υπάρχει

Πίνακας 3.1

3.1.1 Μεταβλητές

Οι **μεταβλητές (variables)** σε ένα πρόγραμμα χρησιμοποιούνται για την αποθήκευση (φύλαξη) δεδομένων όπως οι αριθμοί και τα γράμματα. Ο αριθμός, το γράμμα ή κάποιο άλλο στοιχείο μέσα σε μια μεταβλητή λέγεται **τιμή (value)** της μεταβλητής. Η μεταβλητή, όπως σημαίνει και το όνομα της, μπορεί να αλλάξει τιμές κατά τη διάρκεια εκτέλεσης του προγράμματος. Αφού δηλωθεί μια μεταβλητή, μπορεί να πάρει μια τιμή ως εξής:

Έστω ότι έχουμε την ακέραια μεταβλητή goals και έστω ότι θέλουμε να τις δώσουμε την τιμή “10” , θα έχουμε:

```
int goals;
goals = 10;
```

Ή αλλιώς:

```
int goals = 10;
```

Αν όμως η μεταβλητή είχε προηγούμενη τιμή π.χ. “8” τότε θα μπορούσε να γραφεί:

```
goals = goals + 2;
```

Αυτό θα μπορούσε να διαβαστεί ως εξής: «Η μεταβλητή `goals` θα αυξηθεί κατά δύο».

Όπως παρουσιάστηκε πιο πάνω, πρέπει να η μεταβλητή να δηλώνεται πριν χρησιμοποιηθεί. Δηλώνοντας τη μεταβλητή γνωστοποιείται στον υπολογιστή το όνομα της μεταβλητής, πόση μνήμη πρέπει να δεσμεύσει (με βάση τον τύπο της) για τη μεταβλητή

καθώς επίσης και τον τρόπο με τον οποίο πρόκειται το στοιχείο δεδομένων που βρίσκεται μέσα στη μεταβλητή να κωδικοποιηθεί σαν ακολουθίες μηδέν και ένα.

3.1.2 Ονόματα (Αναγνωριστικά)

Ένα όνομα σε ένα πρόγραμμα Java, όπως το όνομα μίας μεταβλητής, μίας κλάσης, μίας μεθόδου ή ενός αντικειμένου, δεν πρέπει να αρχίζει με ψηφίο και μπορεί να περιέχει μόνο γράμματα, ψηφία (0 έως 9) και το χαρακτήρα υπογράμμισης ή κάτω παύλας (underscore) (_). Τα κεφαλαία και τα μικρά γράμματα θεωρούνται διαφορετικοί χαρακτήρες (η Java είναι γλώσσα Case Sensitive). Τα ονόματα σε ένα πρόγραμμα συχνά λέγονται και **αναγνωριστικά**.

3.1.3 Εντολές εκχώρησης

Μία εντολή εκχώρησης σε μια μεταβλητή ενός πρωτογενούς τύπου στο αριστερό μέρος του συμβόλου ισότητας προκαλεί την ακόλουθη διαδικασία: Πρώτα αποτιμάται η έκφραση που βρίσκεται στα δεξιά από το σύμβολο ισότητας και στη συνέχεια η τιμή της αποδίδεται στη μεταβλητή που βρίσκεται στα αριστερά του συμβόλου ισότητας.

Η σύνταξη της είναι η εξής:

```
Μεταβλητή = Έκφραση;
```

3.1.4 Διαχωριστικά

Διαχωριστικά: ( ) { } [ ] ; , .

3.1.5 Τελεστές εκχώρησης:

=	>	<	!	~	?	:				
==	<=	>=	! =	&&	||	++	--			
+	-	*	/	&	|	^	%	<<	>>	>>>
+=	-=	*=	/ =	&=	| =	^=	%=	<<=	>>=	>>>=

Πίνακας 3.2

3.1.5.1 Συμβατότητες εκχώρησης

Το να γίνει προσπάθεια να τοποθετηθεί μια τετραγωνική τάπα σε μια στρογγυλή τρύπα θεωρείται αστείο πράγμα. Έτσι και στη Java δε γίνεται να εκχωρηθεί μια τιμή ενός τύπου σε μια μεταβλητή άλλου τύπου. Αυτό μπορεί να γίνει εφικτό, αν μετατραπεί η τιμή με κάποιο τρόπο και να γίνει τέτοια ώστε να ταιριάζει με τον τύπο της μεταβλητής που θα εκχωρηθεί.

Η τιμή ενός ακέραιου αριθμού γίνεται να αποθηκευτεί σε μια μεταβλητή που έχει τύπο αριθμού κινητής υποδιαστολής και μάλιστα αυτόματα. Ωστόσο αυτό δε μπορεί να γίνεται με όλους τους τύπους και όχι πάντα. Κάποια παραδείγματα αυτόματης μετατροπής και εκχώρησης:

```
double doubleVariable;
```

```
doubleVariable = 7;
```

ή

```
int intVariable;
intVariable = 7;
double doubleVariable;
doubleVariable = 7;
```

Γενικότερα, μπορεί να εκχωρηθεί μια τιμή οπουδήποτε τύπου της παρακάτω λίστας σε μια μεταβλητή τύπου που βρίσκεται μετά από αυτόν στη λίστα:

byte → short → int → long → float → double

Για παράδειγμα, μπορεί να εκχωρηθεί μια μεταβλητή long σε μία μεταβλητή float ή double. Όμως δε μπορεί να εκχωρηθεί σε int ή σε short ή σε byte

3.1.5.2 Μετατροπή τύπου

Στη περίπτωση λοιπόν που η μετατροπή δε γίνεται να γίνει αυτόματα, θα πρέπει ο χρήστης να χρησιμοποιήσει κάποιο **τύπο μετατροπής** έτσι ώστε να έχει το επιθυμητό αποτέλεσμα. Δηλαδή χρησιμοποιώντας κάποιο τύπο μετατροπής γίνεται να μετατραπεί μία τιμή σε μία "ισοδύναμη" τιμή του επιθυμητού τύπου.

Σύνταξη:

(*Όνομα_τύπου*) Έκφραση ;

Παράδειγμα:

```
double guess;
guess = 7.8;
int answer;
answer = (int) guess;
```

3.1.5.3 Μετατροπή τύπου ενός χαρακτήρα σε ακέραιο

Η Java μερικές φορές μεταχειρίζεται τιμές τύπου char σαν ακέραιους αλλά η εκχώρηση ακέραιων σε χαρακτήρες δεν έχει καμία σχέση με την έννοια των χαρακτήρων. Για παράδειγμα, η παρακάτω μετατροπή τύπου θα δώσει στην έξοδο την τιμή int που αντιστοιχεί στο χαρακτήρα '7':

```
char symbol;
symbol = '7';
System.out.println((int)symbol);
```

Αυτό που αναμένεται να εμφανιστεί στην έξοδο από τον παραπάνω κώδικα θα ήταν το 7. Αυτό όμως που θα εμφανιστεί τελικά θα είναι το 55. Η Java (και όλες οι άλλες γλώσσες προγραμματισμού) χρησιμοποιούν μία αυθαίρετη αρίθμηση χαρακτήρων για τη δημιουργία του ακέραιου που αντιστοιχεί στον κάθε χαρακτήρα. Δεν έχει γίνει κάποια ιδιαίτερη μελέτη για την αντιστοίχηση των ψηφίων στις τιμές που αντιπροσωπεύουν, απλά έχει μια καταγραφή όλων των χαρακτήρων και στη συνέχεια αριθμήθηκαν με τη σειρά που είχαν γραφτεί. Έτσι για το παραπάνω παράδειγμα, ο χαρακτήρας '7' έτυχε να έχει την τιμή

55. **Αυτό το σύστημα αρίθμησης είναι το γνωστό, ενδεχομένως, Unicode**. Το σύστημα **Unicode** είναι το ίδιο με το σύστημα **ASCII** για τους χαρακτήρες της Αγγλικής γλώσσας.+

3.1.5.4 Αρχικοποίηση μεταβλητών

Μία μεταβλητή η οποία δεν έχει δηλωθεί ή δεν της έχει δοθεί ακόμη μία τιμή με μία εντολή εκχώρησης (ή με κάποιο άλλον τρόπο) λέμε ότι **δεν έχει αρχικοποιηθεί (uninitialized).**

Αν η μεταβλητή είναι μια μεταβλητή κλάσης, τότε ουσιαστικά δεν έχει τιμή. Αν η μεταβλητή είναι μία μεταβλητή κάποιου πρωτογενούς τύπου, τότε μπορεί να έχει μία προεπιλεγμένη τιμή. Ωστόσο, το πρόγραμμα θα είναι πιο σαφές αν δοθεί μια τιμή στη μεταβλητή, ακόμα και αν απλά επανεκχωρείται την προεπιλεγμένη τιμή. Ο μεταγλωττιστής ορισμένες φορές μπορεί να διαμαρτυρηθεί ότι δεν έχετε αρχικοποιήσει κάποια μεταβλητή. Στις περισσότερες περιπτώσεις αυτό θα είναι όντως αλήθεια αλλά σε κάποιες περιπτώσεις ο μεταγλωττιστής κάνει λάθος. Ο μεταγλωττιστής δεν θα μεταγλωττίσει το πρόγραμμα αν δεν πειστεί ότι δεν υπάρχουν τέτοια φαινόμενα στο πρόγραμμα. Η λύση σε αυτό είναι να δίδεται τιμή σε μία μεταβλητή πριν χρησιμοποιηθεί για οτιδήποτε.

3.1.5.5 Τελεστές αύξησης και μείωσης

Οι τελεστές αυτοί (++ και -- αντίστοιχα) μπορούν να χρησιμοποιηθούν μέσα σε εκφράσεις αλλάζουν την τιμή της μεταβλητής και επιστρέφουν μία τιμή. Μέσα στις εκφράσεις μπορούν να μπούνε πριν ή μετά από την μεταβλητή. Η σημασία τους δε, είναι διαφορετική σε κάθε περίπτωση. Θεωρείται ο κώδικας:

```
int n = 3;
int m = 4;
int result;
result = n * (++m);
```

Μετά την εκτέλεση αυτού του κώδικα η τιμή της n παραμένει αμετάβλητη στο 3, η τιμή της `m` είναι 5 και η τιμή της `result` είναι 15. Επομένως, η `++m` αλλάζει την τιμή της `m` και επιστρέφει αυτήν την αλλαγμένη τιμή για να χρησιμοποιηθεί στην αριθμητική έκφραση.

Αν στο προηγούμενο παράδειγμα τοποθετηθεί ο τελεστής αύξησης μετά την μεταβλητή `m`, τότε το αποτέλεσμα θα είναι λίγο διαφορετικό. Έστω ο κώδικας:

```
int n = 3;
int m = 4;
int result;
result = n * (m++);
```

Στην περίπτωση αυτή, μετά την εκτέλεση του κώδικα η τιμή της `n` είναι 3 και η τιμή της `m` είναι 5 όπως και πριν, αλλά η τιμή της `result` είναι 12 και όχι 15.

Τι έχει συμβεί; Και οι δύο διαφορετικές εκφράσεις στους παραπάνω κώδικες, αλλάζουν την τιμή της `m` κατά ένα προς τα πάνω. Η διαφορά είναι ότι στην πρώτη περίπτωση (`++m`) η τιμή της `m` αλλάζει **αμέσως πριν** εκτελεστεί η έκφραση, δηλαδή η πράξη, όπου το `m` γίνεται 5 και το αποτέλεσμα βγαίνει 15. Στην δεύτερη περίπτωση, η τιμή της μεταβλητής `m` αλλάζει

αμέσως μετά την εκτέλεση της έκφρασης, οπότε κατά την πράξη η τιμή της `m` είναι ακόμα 4 και αφού γίνει η πράξη και η μεταβλητή `result` πάρει την τιμή 12, αλλάζει η τιμή της `m` και γίνεται 5.

Οι τελεστές αύξησης και μείωσης μπορούν να εφαρμοστούν μόνο σε μεταβλητές και όχι σε σταθερές ή σε πιο σύνθετες αριθμητικές εκφράσεις.

3.2 Η κλάση String

Τα αλφαριθμητικά χαρακτήρων, όπως για παράδειγμα το *"Enter the amount:"*, συμπεριφέρονται λίγο διαφορετικά από τις τιμές των πρωτογενών τύπων. Στη `Java`, δεν υπάρχει πρωτογενής τύπος για αλφαριθμητικά, υπάρχει όπως μια κλάση, η κλάση `String`, η οποία μπορεί να χρησιμοποιηθεί για την αποθήκευση και την επεξεργασία αλφαριθμητικών χαρακτήρων.

3.2.1 Σταθερές και μεταβλητές αλφαριθμητικών

Μία φράση μέσα σε εισαγωγικά, όπως:

```
System.out.println("Enter a whole number from 1 to 99.");
```

είναι μια σταθερή αλφαριθμητικού.

Μια τιμή τύπου String είναι ένα από αυτά τα αλφαριθμητικά μέσα στα εισαγωγικά. Δηλαδή, μια τιμή τύπου String είναι μια ακολουθία χαρακτήρων που αντιμετωπίζεται ως ένα αντικείμενο.

Δήλωση μεταβλητής String:

```
String greeting;
```

Εκχώρηση τιμής σε String:

```
greeting = "Hello!";
```

ή αλλιώς και τα δύο μαζί σε μία έκφραση:

```
String greeting = "Hello!";
```

Μόλις μία μεταβλητή String, όπως η greeting, πάρει μία τιμή, μπορεί να εμφανιστεί στην οθόνη με την εξής εντολή:

```
System.out.println(greeting);
```

Και το αποτέλεσμα στην οθόνη θα είναι:

```
Hello!
```

3.2.2 Συνένωση αλφαριθμητικών

Μπορεί να γίνει πρόσθεση δύο αλφαριθμητικών χρησιμοποιώντας τον τελεστή +. Η σύνδεση δύο αλφαριθμητικών για να ένα μεγαλύτερο λέγεται **συνένωση (concatenation).** Έτσι όταν χρησιμοποιείται με αλφαριθμητικά, το σύμβολο + λέγεται μερικές φορές και **τελεστής συνένωσης (concatenation operator).** Παράδειγμα:

```
String greeting = “Hello”;
String sentence;
sentence = greeting + “ my friend.”;
System.out.println(sentence);
```

Και το αποτέλεσμα από αυτό το κομμάτι κώδικα θα είναι:

```
Hello my friend.
```

Αξίζει σε αυτό το σημείο να διευκρινιστεί μια λεπτομέρεια. Στο παραπάνω κομμάτι κώδικα, στη γραμμή όπου γίνεται η συνένωση των αλφαριθμητικών. Η σταθερή μεταβλητή “ my friend.” πριν την λέξη “my” υπάρχει ένα κενό. Δηλαδή ένας κενός χαρακτήρας. Αν δεν είχε τοποθετηθεί εκεί ο χαρακτήρας αυτός, το αποτέλεσμα της συνένωσης θα ήταν:

```
Hellomy friend.
```

Με την κλάση `String` μπορούν να προστεθούν οποιοιδήποτε αριθμοί αντικειμένων τύπου `String`, ακόμα και με αντικείμενα οποιουδήποτε άλλου τύπου. Το αποτέλεσμα θα είναι πάντα αντικείμενο τύπου `String`. Η Java θα βρει τον τρόπο να εκφράσει οποιοδήποτε αντικείμενο σαν αλφαριθμητικό όταν αυτό συνδέεται μέσω του τελεστή +, με ένα αλφαριθμητικό. Παράδειγμα, για μια απλή περίπτωση όπως οι αριθμοί θα κάνει το προφανές:

```
String solution = “The answer is “ + 42;
```

Η τιμή της μεταβλητής `solution` που θα προκύψει θα είναι ίση με “`The answer is 42”.` Αυτό είναι τόσο φυσιολογικό που φαίνεται σαν να μην συμβαίνει κάτι ιδιαίτερο, αλλά στην πραγματικότητα απαιτεί μια μετατροπή από έναν τύπο στον άλλο. Η σταθερά 42 είναι ένας αριθμός ενώ το “42” είναι ένα αλφαριθμητικό που αποτελείται από τον χαρακτήρα 4 ακολουθούμενο από τον χαρακτήρα 2. Η Java θα πρέπει να μετατρέπει την αριθμητική σταθερή 42 στην σταθερή αλφαριθμητικού “42” και μετά να ενώνει τα δύο αλφαριθμητικά “The answer is “ και “42” δίνοντας το μεγαλύτερο αλφαριθμητικό “`The answer is 42”.`

3.2.3 Μέθοδοι κλάσης `String`

Αν και ακόμα δεν έχουν αναλυθεί η έννοια του αντικειμένου και της μεθόδου παρά μόνο έχουν απλά αναφερθεί σε προηγούμενη παράγραφο. Σε αυτό το σημείο και με την ευκαιρία της ανάλυσης της κλάσης `String` θα πρέπει να γίνει μια αναφορά στους ορισμούς αυτών. Τα **αντικείμενα (objects)** είναι οντότητες οι οποίες αποθηκεύουν δεδομένα και μπορούν να εκτελέσουν λειτουργίες. Οι λειτουργίες που μπορεί να εκτελέσει ένα αντικείμενο λέγονται **μέθοδοι (methods)**. Οι περισσότερες μέθοδοι για την κλάση `String` επιστρέφουν ή δημιουργούν, κάποια τιμή. Για παράδειγμα η μέθοδος `length()` επιστρέφει τον αριθμό των χαρακτήρων που περιέχονται σε ένα αντικείμενο `String`. Άρα

η `"Hello".length()` επιστρέφει τον ακέραιο αριθμό 5 και μπορεί να αποθηκευτεί σε μία μεταβλητή `int` ως εξής:

```
int n = "Hello".length();
```

Όπως φαίνεται στην παραπάνω γραμμή κώδικα, η σύνταξη για να γίνει κλήση μιας μεθόδου χρειάζεται η εντολή να απαρτίζεται από το αντικείμενο, μια τελεία και με το όνομα της μεθόδου ακολουθούμενο από μια ανοιχτή παρένθεση και μια κλειστή. Όταν ενεργοποιείται μια μέθοδος λέμε ότι ο κώδικας **καλεί (invoke ή call)** τη μέθοδο και το αντικείμενο πριν από την τελεία λέγεται **καλούν αντικείμενο (calling object).** Προφανώς στο παραπάνω παράδειγμα θα μπορούσε το αντικείμενο `"Hello"` να εκχωρηθεί σε μία μεταβλητή τύπου String π.χ. `String greeting = "Hello";` και να διαμορφωθεί κατάλληλα η εντολή `"Hello".length()` σε:

```
int n = greeting.length();
```

Οι πληροφορίες για την κλήση της μεθόδου δίνονται μέσα στις παρενθέσεις. Σε κάποιες περιπτώσεις, όπως στην μέθοδο length, δε χρειάζεται να δοθεί καμία πληροφορία πέρα από τα δεδομένα που βρίσκονται στο καλούν αντικείμενο, και οι παρενθέσεις είναι άδειες. Σε άλλες περιπτώσεις όμως πρέπει να παρέχονται κάποιες πληροφορίες μέσα στις παρενθέσεις, οι οποίες λέγονται **όρισμα** ή **ορίσματα (argument** ή **arguments)**.

3.2.4 Χαρακτήρες διαφυγής

Έστω ότι θέλει να εμφανισθεί στην οθόνη η φράση:

```
The word "Java" names a language, not just a drink!
```

Η παρακάτω εντολή δεν θα λειτουργήσει:

```
System.out.println("The word "Java" names a language, not just a drink!");
```

Αντίθετα, θα δώσει ένα μήνυμα σφάλματος του μεταγλωττιστή. Το πρόβλημα είναι ότι ο μεταγλωττιστής βλέπει τη φράση `"The word "` , σαν ένα απόλυτα έγκυρο αλφαριθμητικό μέσα σε εισαγωγικά και στη συνέχεια βλέπει `Java"` , το οποίο δε σημαίνει τίποτα απολύτως στη γλώσσα `Java` (αν και θα μπορούσε να υποθέσει ότι πρόκειται για ένα αλφαριθμητικό μέσα σε εισαγωγικά στο οποίο έχει παραληφθεί το λάθος στο ένα σκέλος των εισαγωγικών ή να γνωρίζει ότι υπήρχε θέληση να συμπεριληφθεί το σύμβολο ‘ ” ’ ως μέρος του αλφαριθμητικού μέσα σε εισαγωγικά, εκτός αν ενημερωθεί από τον προγραμματιστή για κάτι τέτοιο. Αυτό θα μπορούσε να πραγματοποιηθεί αν διαμορφωνόταν η εντολή, ως εξής:

```
System.out.println("The word \"Java\" names a language, not just a drink!");
```

Στον παρακάτω πίνακα Πίνακας 3.3 φαίνονται κάποιες **ακολουθίες διαφυγής (escape sequences)** ή **χαρακτήρες διαφυγής (escape characters)**, επειδή ξεφεύγουν από την συνηθισμένη σημασία ενός χαρακτήρα, όπως είναι για παράδειγμα η συνήθης σημασία των διπλών εισαγωγικών.

`\"`	Διπλά εισαγωγικά.
`\'`	Απλό εισαγωγικό.
`\\`	Ανάποδη κάθετος.

\n	Αλλαγή γραμμής. Πήγαινε στην αρχή της επόμενης γραμμής.
\r	Χαρακτήρας επιστροφής. Πήγαινε στην αρχή της τρέχουσας γραμμής.
\t	Στηλοθέτης. Πρόσθεσε κενό χώρο μέχρι το επόμενο σημείο στηλοθέτη.

Πίνακας 3.3

3.2.5 Είσοδος/Έξοδος πληκτρολογίου και οθόνης

Υπάρχουν διάφοροι τρόποι με τους οποίους ένα πρόγραμμα Java μπορεί να εκτελέσει λειτουργίες Ε/Ε (Εισόδου/Εξόδου). Εδώ θα παρουσιαστούν οι πιο απλοί τρόποι.

3.2.5.1 Έξοδος στην οθόνη

Ήδη από τις πρώτες σελίδες αυτού του κειμένου έχουν χρησιμοποιηθεί απλές εντολές εξόδου. Αυτή η ενότητα απλά θα συνοψίσει και θα εξηγήσει αυτά που ήδη έχουν χρησιμοποιηθεί προηγουμένως.

Η `System.out` είναι ένα αντικείμενο το οποίο αποτελεί μέρος της γλώσσας `Java`. Μπορεί να φαίνεται στον παρατηρητή παράξενο το γεγονός ότι το όνομα ενός αντικειμένου περιέχει μια τελεία, αλλά αυτό είναι κάτι που δε χρειάζεται να αναλυθεί προς το παρόν. Μια από τις μεθόδους του αντικειμένου `System.out` είναι η `println`. Για να χρησιμοποιηθούν εντολές εξόδου αυτής της μορφής, αρκεί μετά από την κλήση της μεθόδου να προστεθεί μέσα στις παρενθέσεις αυτό που επιθυμείται να εμφανιστεί και στο τέλος να προστεθεί το ελληνικό ερωτηματικό. Επίσης στην έξοδο μπορεί να σταλεί αλφαριθμητικά κειμένου μέσα σε διπλά εισαγωγικά όπως `“My name is Kostas!”` ή `“The cost is 50 cents.”` ή `“100”` και σχεδόν οτιδήποτε άλλο αντικείμενο ή τιμή. Επίσης μέσα στις παρενθέσεις μπορούν να γίνουν συνένωση αλφαριθμητικών όπως για παράδειγμα:

```
System.out.prinln(“Lucky number = “ + 13 + “\n” + “Secret number = “ + number);
```

Αυτή η γραμμή κώδικα, αν η μεταβλητή number έχει την τιμή 7 ,θα δώσει:

```
Lucky number = 13

Secret number = 7
```

Μια άλλη μέθοδος για έξοδο στην οθόνη είναι η `print` . Η διαφορά με την μέθοδο `println` είναι ότι η πρώτη εμφανίζει στην ίδια γραμμή αυτό που της έχει δοθεί και μένει στην ίδια γραμμή ενώ η δεύτερη εμφανίζει αυτό που της έχει δοθεί στην ίδια γραμμή και μετά αλλάζει σε μια νέα γραμμή. Παράδειγμα, έστω ότι έχουμε το κομμάτι κώδικα:

```
System.out.print(“One, two,”;
System.out.print(“ buckle my shoe.”);
System.out.println(“ Three, four,”);
System.out.println(“ shut the door.”);
```

θα δώσει την ακόλουθη έξοδο:

```
One, two, buckle my shoe. Three, four,

shut the door.
```

Αν προσέξει κανείς την έξοδο στο προηγούμενο παράδειγμα, θα παρατηρήσει πως μια νέα γραμμή αρχίζει αφού χρησιμοποιηθεί η μέθοδος `println`, αντί της `print`. Δηλαδή **μετά** την εμφάνιση στην έξοδο του τμήματος που καθορίζεται από την μέθοδο `println`. Αυτή είναι και η μοναδική διαφορά μεταξύ των μεθόδων `print` και `println`.

3.2.5.2 Είσοδος από το πληκτρολόγιο

Αυτό επιτυγχάνεται με την χρήση της κλάσης Scanner που βρίσκεται στο πακέτο `java.util` (`util` είναι συντόμευση της αγγλικής λέξης *utility* – που σημαίνει χρησιμότητα, ωφέλεια – αλλά στον κώδικα `Java` χρησιμοποιείται πάντα η απλοποιημένη μορφή `util`). Ένα πακέτο είναι απλά μια βιβλιοθήκη κλάσεων. Προκειμένου να κατασταθεί η κλάση Scanner και η κάθε κλάση του πακέτου `java.util` θα πρέπει κάπου στην αρχή του αρχείου που υπάρχει το πρόγραμμα, να προστεθεί η γραμμή που ορίζει την εισαγωγή της βιβλιοθήκης `java.util`, δηλαδή να εισαχθεί η εντολή:

```
import java.util.*;
```

Έτσι μπορούν να χρησιμοποιηθούν όλες οι κλάσεις του πακέτου `java.util`. Για να γίνει λοιπόν είσοδος από το πληκτρολόγιο πρέπει να δημιουργηθεί ένα αντικείμενο της κλάσης `Scanner`, για παράδειγμα:

```
Scanner keyboard = new Scanner(System.in);
```

όπου το αντικείμενο `keyboard` είναι οποιαδήποτε ονομασία (μη δεσμευμένη λέξη) της `Java`. Μετά την εμφάνιση αυτής της γραμμής κώδικα, μπορεί να χρησιμοποιηθεί οποιαδήποτε μέθοδος της κλάσης `Scanner` με το αντικείμενο `keyboard` για να διαβαστούν τα δεδομένα που εισάγονται από το πληκτρολόγιο. Για παράδειγμα η μέθοδος `nextInt` διαβάζει μια τιμή `int` που εισάγεται από το πληκτρολόγιο. Η κλήση της μεθόδου γράφεται ως εξής:

```
keyboard.nextInt();
```

Δηλαδή για να εκχωρηθεί σε μια μεταβλητή τύπου `int` με όνομα `n1` κάποιος ακέραιος αριθμός από το πληκτρολόγιο η εντολή θα είναι:

```
int n1 = keyboard.nextInt();
```

ή για εκχωρηθεί μια λέξη από το πληκτρολόγιο σε μια μεταβλητή τύπου `String` με όνομα `word` , η γραμμή κώδικα θα πρέπει να είναι:

```
String word = keyboard.next();
```

δηλαδή η μέθοδος `next` εισάγει μια λέξη.

3.2.6 Σχόλια (comments)

Υπάρχουν δύο τρόποι για να εισαχθούν σχόλια σε ένα πρόγραμμα `Java` (ή σε τμήμα κώδικα `Java`):

1. Οτιδήποτε μετά από τα δύο σύμβολα // μέχρι το τέλος της γραμμής είναι σχόλιο και δε λαμβάνεται υπόψη από το μεταγλωττιστή.
2. Οτιδήποτε υπάρχει μεταξύ των ζευγαριών διπλών συμβόλων /* και */ είναι σχόλιο και δε λαμβάνεται υπόψη από το μεταγλωττιστή.
3. Χρησιμοποιείται επίσης για εισαγωγή σχολίων, αλλά με διαφορετικό χρώμα από τις δύο προηγούμενες, ο συμβολισμός για αρχή /** και για κλείσιμο ο συμβολισμός */ .

3.3 Ροή Ελέγχου στη Java

Η **ροή ελέγχου (flow of control)** είναι η σειρά με την οποία ένα πρόγραμμα εκτελεί λειτουργίες. Η Java, όπως και οι περισσότερες γλώσσες προγραμματισμού, χρησιμοποιούν δύο είδη εντολών για τη ρύθμιση της ροής ελέγχου: Μία **εντολή διακλάδωσης (branching statement)** επιλέγει μία από ένα σύνολο δύο ή περισσότερων πιθανών λειτουργιών. Μια **εντολή βρόγχου (loop statement)** επαναλαμβάνει συνεχώς μία λειτουργία μέχρι να ικανοποιηθεί μία συνθήκη διακοπής. Προτού όμως γίνει αναφορά στις μορφές της ροής ελέγχου, θα γίνει μια εισαγωγή στις λογικές εκφράσεις.

3.3.1 Λογικές εκφράσεις

Μια **λογική έκφραση (boolean expression)** είναι μια έκφραση που μπορεί να είναι ή αληθής ή ψευδής. Το όνομα *boolean* προέρχεται από τον George Boole, ο οποίος ήταν Άγγλος επιστήμονας λογικής και μαθηματικών του 19^{ου} αιώνα και του οποίου η εργασία ήταν σχετική με αυτού του είδους τις εκφράσεις.

Στον παρακάτω πίνακα Πίνακας 3.4 φαίνονται οι τελεστές σύγκρισης στη `Java` :

Μαθηματικός συμβολισμός	**Ονομασία**	**Συμβολισμός Java**	**Παράδειγμα Java**
=	*Ίσο με*	`==`	`balance = 0`
≠	*Διάφορο από*	`!=`	`income != tax`
>	*Μεγαλύτερο από*	`>`	`expenses > income`
?	*Μεγαλύτερο από ή ίσο με*	`>=`	`points >= 60`
<	*Μικρότερο από*	`<`	`pressure < max`
?	*Μικρότερο από ή ίσο με*	`<=`	`expenses <= income`

Πίνακας 3.4

Μερικές παρατηρήσεις:

- Μια λογική έκφραση δε χρειάζεται να μπει μέσα σε παρενθέσεις

- Μια λογική έκφραση όμως πρέπει να περικλείεται από παρενθέσεις, όταν χρησιμοποιείται σε μία εντολή `if – else`
- Μπορούν να σχηματιστούν πιο σύνθετες εκφράσεις ενώνοντας τις απλές με το λογικό «και» (“and”) που συμβολίζεται με && ή με το λογικό «είτε» (“OR”) που συμβολίζεται με ||
- Οι παρενθέσεις σε κάθε λογική έκφραση, όταν γίνεται σύγκριση ανάμεσα σε δύο, δεν είναι απαραίτητες, αλλά συνήθως χρησιμοποιούνται ώστε να γίνεται ο κώδικας πιο ευανάγνωστος.
- Στην Java μπορεί να αντιστραφεί μια λογική έκφραση με της προσθήκη θαυμαστικού μπροστά από αυτήν. Για παράδειγμα: `!(name == “Paul”)`
- Για τον έλεγχο ισότητας ή μη αντικειμένων δεν ισχύουν οι τελεστές “==” και “`!=`” γιατί όπως αναφέρθηκε σε προηγούμενο κεφάλαιο ένα αντικείμενο είναι μέρος μια κλάσης, όπως ένα αλφαριθμητικό. Όλα τα αριθμητικά βρίσκονται στη κλάση `String`, επομένως ο τελεστής “==” δεν θα ελέγξει αν τα δύο αλφαριθμητικά είναι ίσα. Δηλαδή χρειάζεται να χρησιμοποιηθεί η μέθοδος “`.equals()` “ και όχι ο τελεστής “==” .

3.3.1.1 Κανόνες προτεραιότητας

Πίνακες αληθείας (truth tables) για τους λογικούς (`boolean`) τελεστές:

&& (and)		
Τιμή του Α	**Τιμή του Β**	**Τιμή της πράξης Α && Β**
true	*true*	*true*
true	*false*	*false*
false	*true*	*false*
false	*false*	*false*

Πίνακας 3.5

|| (OR)		
Τιμή του Α	**Τιμή του Β**	**Τιμή της πράξης Α || Β**
true	*true*	*true*
true	*false*	*true*
false	*true*	*true*
false	*false*	*false*

Πίνακας 3.6

! (not)	
Τιμή του Α	**Τιμή της πράξης !(Α)**
true	*false*

false	*true*

Πίνακας 3.7

Κανόνες προτεραιότητας Πίνακας 3.8 (preceding rules) :

Υψηλότερη προτεραιότητα
Πρώτη: οι μοναδιαίοι τελεστές +,-,++,-- και !
Δεύτερη: οι δυαδικοί αριθμητικοί τελεστές *,/,%
Τρίτη: οι δυαδικοί αριθμητικοί τελεστές +,-
Τέταρτη: οι λογικοί τελεστές <,>,<=,>=
Πέμπτη: οι λογικοί τελεστές ==,!=
Έκτη: ο λογικός τελεστής &
Έβδομη: ο λογικός τελεστής |
Όγδοη: ο λογικός τελεστής &&
Ένατη: ο λογικός τελεστής ||
Χαμηλότερη προτεραιότητα

Πίνακας 3.8

3.3.2 Εντολές διακλάδωσης

Η πιο απλό μορφή εντολής διακλάδωσης είναι αυτή που έχει δύο πιθανές εναλλακτικές λειτουργίες και αυτή είναι η εντολή διακλάδωσης `if-else`.

3.3.2.1 Η εντολή `if – else`

Η σύνταξη της εντολής είναι η εξής:

```
if (λογική_έκφραση) {
  λειτουργία 1  }
else {
  λειτουργία 2  }
```

Στο παραπάνω κομμάτι κώδικα συμβαίνει το εξής: Αν η `λογική_έκφραση` είναι αληθής (true) , τότε εκτελείται η *λειτουργία 1*. Αν η `λογική_έκφραση` είναι ψευδής (false) τότε θα εκτελεστεί η *λειτουργία 2*.

Μπορεί επίσης να σχηματιστούν σύνθετες εντολές. Δηλαδή μπορούν να χρησιμοποιηθεί *if – else* μέσα σε μια άλλη εντολή *if – else* και ούτω καθεξής. Παράδειγμα:

```
if (λογική_έκφραση1) {
   if (λογική_έκφραση2) {
      λειτουργία 1  }
  else {
      λειτουργια 2  }
else {
     λειτουργία 3 }
```

Σε αυτήν την περίπτωση, αν ισχύει η `λογική_έκφραση1` τότε θα εκτελεστεί η *λειτουργία 1* ή η *λειτουργία 2* , ανάλογα με την τιμή της `λογική_έκφραση2`. Δηλαδή αν η `λογική_έκφραση2` είναι αληθής τότε θα εκτελεστεί η *λειτουργία 1* , εάν πάλι είναι ψευδής θα εκτελεστεί η *λειτουργία 2* . Αν εξαρχής ήταν ψευδής η `λογική_έκφραση1` τότε θα εκτελούνταν η *λειτουργία 3*.

3.3.2.2 Εντολές `if – else` πολλαπλών διακλαδώσεων

Το προηγούμενο παράδειγμα αναφέρθηκε σε σύνθετες εντολές ελέγχου. Ουσιαστικά αυτές οι σύνθετες εντολές στην Java, μπορούν να με έναν τρόπο πιο ευανάγνωστο και πιο «έξυπνο». Αυτός ο τρόπος περιγράφεται από τη φράση «πολλαπλών διακλαδώσεων» και έχει την εξής μορφή:

```
if (λογική_έκφραση1) {
      λειτουργία 1  }
else if (λογική_έκφραση2) {
      λειτουργία 2  }
.
.
.
else if (λογική_έκφραση_n) {
      λειτουργία n  }
else {
      προεπιλεγμένη_λειτουργία  }
```

Οι παραπάνω λειτουργίες είναι εντολές της Java. Οι λογικές εκφράσεις ελέγχονται η μία μετά την άλλη ξεκινώντας από την κορυφή. Όταν βρεθεί η πρώτη αληθής λογικά έκφραση, τότε εκτελείτε η εντολή που ακολουθεί αμέσως μετά από αυτήν την αληθή λογική έκφραση. Η *προεπιλεγμένη_λειτουργία* εκτελείται μόνο αν καμία από τις λογικές εκφράσεις δεν είναι αληθής.

3.3.2.3 Η εντολή `switch`

Η εντολή switch είναι μια εντολή πολλαπλών διακλαδώσεων που παίρνει την απόφαση για το σκέλος προς το οποίο θα διακλαδωθεί βάσει της τιμής μίας έκφρασης ακεραίου ή χαρακτήρα. Η εντολή αρχίζει με την δεσμευμένη λέξη switch και ακολουθεί μια έκφραση μέσα σε παρενθέσεις η οποία ονομάζεται **έκφραση ελέγχου (controlling expression).** Κάτω από αυτήν την έκφραση υπάρχει μια λίστα περιπτώσεων μέσα σε αγκύλες. Η κάθε περίπτωση αποτελείται από την δεσμευμένη λέξη case, το σύμβολο της άνω τελείας και από μία λίστα εντολών, οι οποίες είναι οι λειτουργίες γι' αυτήν την περίπτωση. Η σταθερή που χρησιμοποιείται μετά την δεσμευμένη λέξη case λέγεται **ετικέτα περίπτωσης (case label)**. Σε αντίθεση με την εντολή `if – else` , εδώ δεν υπάρχει default περίπτωση (ο ρόλος της else στην εντολή `if – else`) οπότε αν καμία συνθήκη δεν είναι αληθής, τότε δεν θα εκτελεστεί καμία λειτουργία. Παρακάτω φαίνεται η δομή της εντολής switch:

```
switch (έκφραση_ελέγχου) {
      case Ετικέτα_περίπτωσης:
      εντολή:
      εντολή:
      .
      .
      .
      εντολή:
      break;
      case Ετικέτα_περίπτωσης:
      εντολή:
      εντολή:
      .
      .
      .
```

```
        εντολή:
        break;
        /* ο αριθμός των περιπτώσεων είναι απεριόριστος. Η παρακάτω περίπτωση
          default είναι προαιρετική */
        default Ετικέτα_περίπτωσης:
        εντολή:
        εντολή:
        .
        .
        .
        εντολή:
        break;
}
```

Μερικές παρατηρήσεις για την παραπάνω δομή της `switch`:

- Ο αριθμός των περιπτώσεων είναι απεριόριστος
- Η περίπτωση default είναι προαιρετική
- Η εντολή `break` η οποία είναι δεσμευμένη λέξη θα μπορούσε να μην υπάρχει. Έτσι το πρόγραμμα θα έλεγχε κάθε μία από τις επόμενες περιπτώσεις με την σειρά έως ότου δεν έχει άλλη περίπτωση ή μέχρι να βρει κάποια εντολή `break`
- Κάθε *Ετικέτα_περίπτωσης* είναι μία σταθερά του ίδιου τύπου με την *Έκφραση_ελέγχου*
- Κάθε περίπτωση έχει διαφορετική *Ετικέτα_περίπτωσης*
- Η έκφραση ελέγχου πρέπει να είναι τύπου `char`, `int`, `short` ή `byte`.

3.3.3 Εντολές βρόχου

Τα προγράμματα χρειάζεται να επαναλαμβάνουν κάποια λειτουργία ή ενέργεια. Ένα τμήμα ενός προγράμματος που επαναλαμβάνει μια εντολή ή μία ομάδα εντολών λέγεται **βρόχος (loop)**. Η εντολή (ή η ομάδα εντολών) που επαναλαμβάνεται, λέγεται **σώμα (body)** του βρόχου. Κάθε εκτέλεση του σώματος βρόχου λέγεται **επανάληψη (iteration)** του βρόχου.

3.3.3.1 Εντολή `while`

Ένας τρόπος για να φτιαχτεί ένας βρόχος στην `Java` είναι με μία **εντολή** `while`, η οποία λέγεται και **βρόχος** `while`. Μία εντολή `while` επαναλαμβάνει την λειτουργία της συνεχώς μέχρι μία λογική έκφραση ελέγχου να επιστρέψει ψευδή τιμή. Η εντολή αρχίζει με την δεσμευμένη λέξη `while` ακολουθούμενη από μία λογική έκφραση μέσα σε παρενθέσεις, η οποία είναι η λογική έκφραση ελέγχου. Το σώμα επαναλαμβάνεται εφόσον η λογική έκφραση ελέγχου παραμένει αληθής. Παρακάτω φαίνεται η δομή της εντολής `while` :

```
while (λογική_έκφραση) {
        Σώμα   }
```

Παρατηρήσεις:

- Αν και για όσο η *λογική_έκφραση* είναι αληθής, το σώμα θα εκτελείται επαναλαμβανόμενα
- Το σώμα μπορεί να μην εκτελεστεί καθόλου
- Αν υπάρχει εντολή `break` κάπου μέσα στο σώμα, τότε ο βρόχος σταματά να εκτελείται

3.3.3.2 Εντολή do-while

Η **εντολή** do-while (η οποία λέγεται και **βρόχος** do-while) μοιάζει πολύ με την εντολή while. Η βασική διαφορά είναι ότι με μία εντολή do-while το σώμα του βρόχου εκτελείται πάντοτε τουλάχιστον μία φορά. Όπως φάνηκε προηγουμένως στην εντολή while , το σώμα μπορεί να μην εκτελεστεί καθόλου. Παρακάτω φαίνεται η δομή της εντολής do-while :

```
do  {
      Σώμα  }
while (λογική_έκφραση);
```

Παρατηρήσεις:

- Το σώμα θα εκτελεστεί τουλάχιστον μία φορά
- Μετά την *λογική_έκφραση* μέσα στις παρενθέσεις υπάρχει ελληνικό ερωτηματικό
- Το σώμα βρίσκεται μέσα σε αγκύλες

3.3.3.3 Εντολή for

Η εντολή for είναι ουσιαστικά μια άλλη μορφή της εντολής while. Η δομή της εντολής for έχει ως εξής:

```
for (Αρχικοποίηση; λογική_έκφραση; Ανανέωση) {
      Σώμα  }
```

Παρατηρήσεις:

- Όπως και στην εντολή while το σώμα μπορεί να μην εκτελεστεί καμία φορά
- Το σώμα βρίσκεται μέσα σε αγκύλες
- Η τρείς εκφράσεις μέσα στην παρένθεση διαχωρίζονται από δύο και όχι από τρία ελληνικά ερωτηματικά
- Θα μπορούσε η πρώτη έκφραση με την δεύτερη έκφραση να διαχωρίζονται με κόμμα
- Η ανανέωση των βρόχων for αλλάζει πάντοτε μόνο μία μεταβλητή, ωστόσο μπορεί να χρησιμοποιηθεί οποιαδήποτε έκφραση της Java , οπότε μπορεί να χρησιμοποιηθούν παραπάνω από μία μεταβλητές σε αυτήν την έκφραση, ακόμα και διαφορετικών τύπων μεταξύ τους. Αυτό ισχύει και για τις άλλες δύο εκφράσεις.
- Για να γίνει έλεγχος στον τερματισμό του βρόχου for μπορεί να χρησιμοποιηθεί μόνο μία λογική έκφραση. Μπορεί όμως πολλές λογικές εκφράσεις να σχηματίζουν τελικά μία μεγαλύτερη λογική έκφραση.
- Μπορούν να χρησιμοποιηθούν παραπάνω από μία ανανεώσεις χρησιμοποιώντας μια λίστα στην οποία η ανανεώσεις διαχωρίζονται μεταξύ τους με κόμμα. Σε αυτή την περίπτωση ενδεχομένως να υπάρξει κάποιο κενό σώμα, αλλά παρόλα αυτά να εξακολουθεί να κάνει κάτι χρήσιμο.
- Η εντολή for ολοκληρώνει πάντοτε το σώμα βρόχου της σε κάθε επανάληψη, εκτός της περίπτωσης που στο σώμα εκτελεστεί εντολή break.

Μερικές φορές ένα πρόγραμμα μπορεί να βρεθεί σε μία τέτοια κατάσταση που η συνέχιση να μην έχει νόημα. Σε αυτές τις περιπτώσεις μπορεί να τερματιστεί το πρόγραμμα κάνοντας κλήση της μεθόδου `exit` ως εξής:

```
System.exit(0);
```

Η παραπάνω εντολή θα τερματίσει ένα πρόγραμμα αμέσως μετά την εκτέλεση της.

3.4 Ορισμός κλάσεων και μεθόδων

Ένα πρόγραμμα Java αποτελείται από αντικείμενα, διαφόρων κλάσεων, που αλληλεπιδρούν μεταξύ τους. Τα **αντικείμενα (objects)** μπορούν να αναπαραστήσουν αντικείμενα του πραγματικού κόσμου, όπως αυτοκίνητα, σπίτια, φακέλους εργαζομένων – σχεδόν τα πάντα. Μία **κλάση (class)** είναι ο ορισμός ενός είδους αντικειμένου. Είναι κάτι σαν το περίγραμμα ή το πλάνο για την κατασκευή συγκεκριμένων αντικειμένων. Παρακάτω φαίνεται το περίγραμμα μιας κλάσης:

```
Όνομα κλάσης: Automobile
Δεδομένα:
       ποσότητα καυσίμου
       ταχύτητα
       αριθμός πινακίδων
Μέθοδοι (λειτουργίες):
       increaceSpeed:
              Πώς: Πάτα το γράζι
       stop:
              Πώς: Πάτα το φρένο
```

Υλοποιήσεις της κλάσης *Automobile :*

Πρώτη υλοποίηση: **Όνομα Αντικειμένου:** patsCar

Ποσότητα καυσίμου: 40 **λίτρα**

ταχύτητα: 90 **χιλιόμετρα την ώρα**

αριθμός πινακίδων: ‘NZH 4515’

Δεύτερη υλοποίηση: **Όνομα Αντικειμένου:** suesCar

Ποσότητα καυσίμου: 56 **λίτρα**

ταχύτητα: 0 **χιλιόμετρα την ώρα**

αριθμός πινακίδων: ‘ΔΧ 1999’

Τρίτη υλοποίηση: **Όνομα Αντικειμένου:** ronsCar

Ποσότητα καυσίμου: 2 **λίτρα**

ταχύτητα: 75 **χιλιόμετρα την ώρα**

αριθμός πινακίδων: ‘351 WLF’

Ο παραπάνω τρόπος απεικόνισης του περιγράμματος μιας κλάσης, στη συγκεκριμένα περίπτωση της κλάσης *Automobile* είναι λίγο δύσχρηστος γι' αυτό και στον προγραμματισμό χρησιμοποιείται ευρύτερα ένα είδος απεικόνισης που ονομάζεται **διάγραμμα κλάσης UML (UML class diagram)** ή απλά **διάγραμμα κλάσης**. Τα αρχικά UML σημαίνουν Unified Modeling Language και στα ελληνικά Ενοποιημένη Γλώσσα Μοντελοποίησης. Το περίγραμμα της κλάσης *Automobile* σε διάγραμμα κλάσης UML φαίνεται παρακάτω Πίνακας 3.9:

Automobile
- fuel: double
- speed: double
- license: String
+ increaseSpeed (double howHardPress) : void
+ stop (double howHardPress) : void

Πίνακας 3.9

Κάθε ορισμός κλάσης που γράφεται, δημιουργείται ένα αυτόνομο αρχείο με όνομα ίδιο με αυτό της κλάσης και κατάληξη `.java` . Μία κλάση μπορεί να μεταγλωττιστεί, ακόμα και αν δεν είναι διαθέσιμο ακόμα το πρόγραμμα που θα την χρησιμοποιήσει. Η μεταγλώττιση μια κλάσης θα δημιουργήσει ένα αρχείο με όνομα σαν αυτό του αρχείου της κλάσης και κατάληξη `.class` . Αργότερα μπορεί να μεταγλωττιστεί ένα αρχείο προγράμματος με ένα τμήμα `main` το οποίο θα χρησιμοποιεί την κλάση, χωρίς να χρειάζεται να μεταγλωττιστεί ξανά ο ορισμός της κλάσης.

Παρακάτω θα παρουσιαστεί μία κλάση και ο κώδικας ενός προγράμματος που χρησιμοποιεί την κλάση και στην συνέχεια θα αναλυθούν κάποια βασικά στοιχεία τους. Ο κώδικας της κλάσης είναι ο εξής:

```
import java.util.*;
public class SpeciesFirstTry
{
	public String name;
	public int population;
	public double growthRate;
	public void readInput()
	{
		Scanner keyboard = new Scanner(System.in);
		System.out.println("What is the species' name?");
		name = keyboard.nextLine();
		System.out.println("What is the population of the species?");
		population = keyboard.nextInt();
		while (population < 0)
		{
			System.out.println("Population cannot be negative.");
			System.out.println("Reenter population:");
			population = keyboard.nextInt();
		}
		System.out.println("Enter growth rate (percent increase per
					year):");
		growthRate = keyboard.nextDouble();
	}
	public void writeOutput()
	{
		System.out.println("Name = " + name);
```

```
            System.out.println("Population = " + population);
            System.out.println("Growth rate = " + growthRate + "%");
      }
      public int populationIn10()
      {
            double populationAmount = population;
            int count = 10;
            while ((count > 0) && (populationAmount > 0))
            {
                  populationAmount = (populationAmount +
                  (growthRate/100) * populationAmount);
                  count--;
            }
            if (populationAmount > 0)
            {
                  return (int)populationAmount;
                  /** Το (int) μπροστά από την μεταβλητή είναι μια
                  *  μετατροπή τύπου.Αυτή η εντολή μετά την εκτέλεση
                  *  θα επιστρέψει το ακέραιο μέρος της μεταβλητής
                  */
            else
            return 0;
            }
      }
}
```

Και το πρόγραμμα που υλοποιεί την κλάση είναι:

```
public class SpeciesFirstTryDemo
{
      public static void main(String[] args)
      {
            SpeciesFirstTry speciesOfTheMonth = new SpeciesFirstTry();
            int futurePopulation;
            System.out.println("Enter data on the Species of the Month:");
            speciesOfTheMonth.readInput();
            speciesOfTheMonth.writeOutput();
            futurePopulation = speciesOfTheMonth.populationIn10();
            System.out.println("In ten years the population will be " +
             futurePopulation);
            speciesOfTheMonth.name = "Klingon ox";
            speciesOfTheMonth.population = 10;
            speciesOfTheMonth.growthRate = 15;
            System.out.println("The new Species of the Month:");
            speciesOfTheMonth.writeOutput();
            System.out.println("In ten years the population will be: " +
            speciesOfTheMonth.populationIn10());
      }
}
```

Η εκτέλεση του προγράμματος θα μπορούσε να εμφανίσει το εξής αποτέλεσμα:

```
Enter data on the Species of the Month:

What is the species' name?

Ferengie fur ball

What is the population of the species?

1000

Enter growth rate (percent increase per year):

-20.5
```

```
Name = Ferengie fur ball

Population = 1000

Growth rate = -20.5%

In ten years the population will be 100

The new Species of the Month:

Name = Klingon ox

Population = 10

Growth rate = 15.0%

In the years the population will be 40
```

Το όνομα της κλάσης `SpeciesFirstTry` και η κλάση έχει σχεδιαστεί για να διατηρεί στοιχεία για τα υπό εξαφάνιση ζώα. Κάθε αντικείμενο αυτής της κλάσης έχει τρία είδη δεδομένων: όνομα, μέγεθος πληθυσμού και ρυθμό αναπαραγωγής. Τα αντικείμενα έχουν τρείς μεθόδους: `readInput, writeOutput` και `populationIn10`. Και τα δεδομένα και οι μέθοδοι μερικές φορές αναφέρονται ως **μέλη (members)** του αντικειμένου επειδή ανήκουν στο αντικείμενο. Μερικές φορές αναφέρονται και ως **πεδία (fields)**. Ωστόσο, στο παρόν κείμενο τα δεδομένα θα αποκαλούνται **μεταβλητές στιγμιοτύπου (instance variables)** και οι μέθοδοι **μέθοδοι**.

3.4.1 Μεταβλητές στιγμιοτύπου

Οι τρεις πρώτες γραμμές μέσα στην κλάση ορίζουν τρείς μεταβλητές στιγμιοτύπου (τρία μέλη δεδομένα):

```
public String name;
public int population;
public double growthRate;
```

Η λέξη `public` (δημόσια) σημαίνει απλά ότι δεν υπάρχουν περιορισμοί στον τρόπο χρήσης αυτών των μεταβλητών στιγμιοτύπου. Κάθε μια από αυτές τις γραμμές κώδικα δηλώνει ένα όνομα μιας μεταβλητής στιγμιοτύπου. Αν γίνει αποδεκτό ότι ένα αντικείμενο της κλάσης είναι μια σύνθετη δομή που περιέχει μεταβλητές στιγμιοτύπου, τότε μπορεί να θεωρηθεί ότι μια μεταβλητή στιγμιοτύπου είναι μια μικρότερη μεταβλητή μέσα σε ένα αντικείμενο της κλάσης. Σε παρούσα περίπτωση, οι μεταβλητές στιγμιοτύπου λέγονται `name`, `population` και `growthRate`. Για αναφερθεί κανείς σε κάποια από αυτές τις μεταβλητές στιγμιοτύπου αρκεί να γράψει, χωρίς κενά διαστήματα, το όνομα του αντικειμένου, μία τελεία και το όνομα της μεταβλητής στιγμιοτύπου. Για παράδειγμα η:

```
speciesOfTheMonth.name
```

υπονοεί τη μεταβλητή στιγμιοτύπου `name` για το αντικείμενο `speciesOfTheMonth`.

Κάθε αντικείμενο του τύπου `SpeciesFirstTry` έχει τις δικές του τρεις μεταβλητές στιγμιοτύπου. Για παράδειγμα, αν το πρόγραμμα περιείχε και την εντολή-δήλωση:

```
SpeciesFirstTry speciesOfTheLastMonth = new SpeciesFirstTry();
```

οι `speciesOfTheMonth.name` και `speciesOfLastMonth.name` είναι δύο διαφορετικές μεταβλητές στιγμιοτύπου που πιθανώς θα έχουν διαφορετικές τιμές αλφαριθμητικών.

Η δεσμευμένη λέξη «new» όταν χρησιμοποιείται σε μια έκφραση όπως η προηγούμενη, μπορεί να θεωρηθεί ότι σκοπός της είναι να δημιουργήσει τις μεταβλητές στιγμιοτύπου του αντικειμένου. Η λέξη «new» βάζει τις μεταβλητές στιγμιοτύπου μέσα στο αντικείμενο.

3.4.2 Είδη μεθόδων

Όπως έχει αναφερθεί και προηγουμένως, όταν χρησιμοποιείται μια μέθοδος λέγεται ότι **καλείται** η μέθοδος. Στα κομμάτια κώδικα που έχουν παρουσιαστεί στο κείμενο έχουν γίνει κλήσεις μεθόδων. Υπάρχουν δύο είδη μεθόδων: Αυτές που επιστρέφουν μια τιμή και αυτές που εκτελούν κάποια διαφορετική λειτουργία από την επιστροφή μιας τιμής. Οι μέθοδοι της δεύτερης περίπτωσης ονομάζονται **μέθοδοι** `void`. Στο παράδειγμα της Ενότητας 2.4 έχει γίνει αναφορά στα βασικά στοιχεία και χαρακτηριστικά της κλήσης κάποιας μεθόδου.

3.4.2.1 Ορισμοί μεθόδων void

Στο πρόγραμμα αυτής της ενότητας πραγματοποιήθηκε η κλήση της μεθόδου `writeOutput()` με την εξής εντολή:

```
speciesOfTheMonth.writeOutput();
```

Στο σώμα της πρώτης κλάσης βρίσκεται η μέθοδος `writeOutput()` η οποία είναι η εξής:

```
public void writeOutput()
      {
            System.out.println("Name = " + name);
            System.out.println("Population = " + population);
            System.out.println("Growth rate = " + growthRate + "%");
      }
```

Παρατηρώντας την δομή του κώδικα της μεθόδου:

- η λέξη που βρίσκεται μετά το κενό δεξιά από την δεσμευμένη λέξη void η οποία ακολουθείται από ανοικτή και κλειστή παρένθεση λέγεται **επικεφαλίδα (heading)** της μεθόδου.
- μετά την επικεφαλίδα ακολουθεί μια ανοικτή αγκύλη και στο τέλος του κώδικα της μεθόδου μια κλειστή αγκύλη. Το περιεχόμενο αυτών των αγκυλών ονομάζεται **σώμα (body)** της μεθόδου.

Όλοι οι ορισμοί μεθόδων ανήκουν σε κάποια κλάση και όλοι γράφονται μέσα στο ορισμό της κλάσης στην οποία ανήκουν. Η μέθοδος `writeOutput()` βρίσκεται μέσα στον ορισμό της κλάσης `SpeciesFirstTry` . Αυτό σημαίνει ότι αυτή η μέθοδος μπορεί να χρησιμοποιηθεί μόνο με αντικείμενα της κλάσης `SpeciesFirstTry` . Η λέξη `public`

υποδηλώνει πως δεν υπάρχουν ειδικοί περιορισμοί στην χρήση της μεθόδου. Παρακάτω σε αυτό το κεφάλαιο θα αναλυθούν οι προσδιορισμοί στην χρήση των μεθόδων.

Στην επόμενη υποενότητα που αναφέρεται στις μεθόδους που επιστρέφουν μια τιμή, παρουσιάζεται η δεσμευμένη λέξη «return». Παρόλο που ο τύπος μεθόδου void δεν επιστρέφει κάποια τιμή, μπορεί να χρησιμοποιηθεί η εντολή `return` η οποία σε αυτήν την περίπτωση δεν θα ακολουθείται από καμία έκφραση δηλαδή θα έχει την μορφή:

```
return;
```

με την εκτέλεση αυτής της εντολής, η εκτέλεση της μεθόδου τύπου `void` θα τερματίζεται. Δηλαδή η εντολή `return` θα μπορούσε να χρησιμοποιηθεί για την διακοπή της εκτέλεσης της μεθόδου νωρίτερα.

3.4.2.2 Μέθοδοι που επιστρέφουν μια τιμή

Στο πρόγραμμα υπάρχει επίσης η εξής εντολή:

```
futurePopulation = speciesOfTheMonth.populationIn10();
```

όπου η μεταβλητή `futurePopulation` είναι μια μεταβλητή ακέραιου αριθμού. Με την παραπάνω εντολή καλείται η μέθοδος `populationIn10()` της οποίας ο κώδικας υπάρχει στον ορισμό της κλάσης `SpeciesFirstTry` και το αποτέλεσμα από την εκτέλεση της εντολής θα είναι η εκχώρηση της επιστρεφόμενης τιμής στην μεταβλητή `futurePopulation`. Ο κώδικας της μεθόδου είναι ο εξής:

```
public int populationIn10()
	{
		double populationAmount = population;
		int count = 10;
		while ((count > 0) && (populationAmount > 0))
		{
			populationAmount = (populationAmount +
			(growthRate/100) * populationAmount);
			count--;
		}
		if (populationAmount > 0)
		{
			return (int)populationAmount;
			/** Το (int) μπροστά από την μεταβλητή είναι μια
			*  μετατροπή τύπου.Αυτή η εντολή μετά την εκτέλεση
			*  θα επιστρέψει το ακέραιο μέρος της μεταβλητής
			*/
		else
		return 0;
		}
	}
```

Παρατηρείται πως ο ορισμός μοιάζει με τον ορισμό μιας μεθόδου τύπου `void` με τη διαφορά ότι πρέπει να προστεθεί ένα επιπλέον στοιχείο, το οποίο είναι αυτό που καθορίζει την επιστρεφόμενη τιμή. Συγκεκριμένα:

- μετά την δεσμευμένη λέξη προσδιορισμού της χρήσης της μεθόδου (στην περίπτωση αυτή, της λέξης `public`) και μετά από ένα κενό γράφεται ο τύπος της επιστρεφόμενης τιμής ακολουθούμενος από ένα ακόμα κενό.

- στην συνέχεια γράφεται όπως σε μια μέθοδο τύπου void το όνομα της μεθόδου, δηλαδή η επικεφαλίδα.
- όσον αφορά το σώμα της μεθόδου, έχει ίδια χαρακτηριστικά με το σώμα μια μεθόδου void.
- ένα άλλο διαφορετικό χαρακτηριστικό στην δομή αυτής της μεθόδου είναι η δεσμευμένη λέξη «return» ακολουθούμενη από ένα κενό και μια μεταβλητή, η οποία είναι και η επιστρεφόμενη μεταβλητή της μεθόδου.

Λίγα λόγια για την λέξη «return». Χρησιμοποιείται για να επιστρέψει την επιστρεφόμενη τιμή της μεθόδου. Συνήθως είναι η τελευταία από τις εκτελούμενες εντολές και βρίσκεται στο σώμα της μεθόδου. Αυτό όμως δεν είναι υποχρεωτικό. Δηλαδή μπορεί να βρίσκεται οπουδήποτε μέσα στο σώμα. Σημασία έχει πως με την εκτέλεση αυτής της εντολής η μέθοδος σταματάει να εκτελείται και η μέθοδος επιστρέφει την επιστρεφόμενη τιμή. Δηλαδή στην περίπτωση που υπάρχει και άλλη-άλλες εντολές μετά την εντολή `return` , αυτές **δεν** θα εκτελεστούν.

Σε κάποιες άλλες γλώσσες προγραμματισμού αυτοί οι μέθοδοι που επιστρέφουν μια τιμή λέγονται *συναρτήσεις* και μία μέθοδος που επιστρέφει μια τιμή δεν αντιστοιχεί στην μαθηματική έννοια της συνάρτησης. Ωστόσο, στη `Java` λέγονται *μέθοδοι* (που επιστρέφουν μια τιμή) και όχι *συναρτήσεις*.

3.4.3 Η παράμετρος `this`

Αν παρατηρήσει κανείς τον ορισμό της κλάσης `SpeciesFirstTry` στην αρχή αυτής της ενότητας και μετά το πρόγραμμα που χρησιμοποιεί την κλάση αυτή, θα προσέξει ότι οι μεταβλητές στιγμιοτύπου είναι γραμμένες διαφορετικά, ανάλογα με το αν βρίσκονται μέσα στον ορισμό της κλάσης ή κάπου έξω από αυτόν, όπως για παράδειγμα σε ένα πρόγραμμα που χρησιμοποιεί την κλάση. Έξω από τον ορισμό της κλάσης, η ονομασία μιας μεταβλητής στιγμιοτύπου γίνεται δίδοντας το όνομα ενός αντικειμένου της κλάσης, μία τελεία και το όνομα της μεταβλητής στιγμιοτύπου, όπως φαίνεται παρακάτω:

```
speciesOfTheMonth.name = “Klingon ox”;
```

Ωστόσο, μέσα στον ορισμό μιας μεθόδου αυτής της ίδιας κλάσης μπορεί απλά να χρησιμοποιηθεί το όνομα της μεταβλητής στιγμιοτύπου χωρίς όνομα αντικειμένου ή τελεία. Για παράδειγμα η παρακάτω γραμμή εντολής που εμφανίζεται στο σώμα της μεθόδου `readInput()` της κλάσης `SpeciesFirstTry` :

```
name = keyboard.nextLine();
```

Όπως έχει αναφερθεί προηγουμένως, κάθε μεταβλητή στιγμιοτύπου, συμπεριλαμβανομένης και της μεταβλητής στιγμιοτύπου `name` , είναι μια μεταβλητή στιγμιοτύπου κάποιου αντικειμένου. Σε περιπτώσεις σαν και αυτή, το αντικείμενο εξυπακούεται ότι βρίσκεται εκεί αλλά το όνομα του συνήθως παραλείπεται. Αυτό το εννοούμενο αντικείμενο έχει το κάπως παράξενο όνομα `this`. Έτσι η προηγούμενη εντολή εκχώρησης θα μπορούσε να γραφτεί:

```
this.name = keyboard.nextLine();
```

Η παράμετρος this αντικαθιστά, ουσιαστικά, το καλούν αντικείμενο. Είναι σαν ένα κενό που περιμένει να συμπληρωθεί από το αντικείμενο που καλεί τη μέθοδο. Εξυπακούεται ότι η this υπάρχει πάντα εκεί ακολουθούμενη από την τελεία, αλλά η Java επειδή θα χρησιμοποιούνταν πολύ συχνά, αφήνει να γίνει παράλειψη της. Έτσι συνηθίζεται να χρησιμοποιείται μόνο στις περιπτώσεις όπου χρειάζεται.

3.4.4 Τοπικές μεταβλητές

Μια μεταβλητή που δηλώνεται μέσα σε μια μέθοδο λέγεται **τοπική μεταβλητή (local variable)** επειδή η σημασία της είναι τοπική, η εμβέλεια της περιορίζεται μέσα στον ορισμό της μεθόδου. Αν υπάρχουν δύο μέθοδοι και η κάθε μία από αυτές δηλώνει μία μεταβλητή με το ίδιο όνομα, τότε πρόκειται για δύο μεταβλητές που συμβαίνει να έχουν το ίδιο όνομα. Οποιαδήποτε αλλαγή γίνεται στην μεταβλητή της μιας μεθόδου, δεν θα έχει καμία απολύτως επίδραση στην μεταβλητή, με ίδιο όνομα, της άλλης μεθόδου. Επίσης, επειδή το τμήμα main θεωρείται μια μέθοδος, όλες οι μεταβλητές που δηλώνονται μέσα σε αυτό το τμήμα θεωρούνται τοπικές μεταβλητές της μεθόδου main. Τα ίδια ισχύουν και για ένα **μπλοκ**, δηλαδή για μια σύνθετη εντολή ή για ένα σύνολο εντολών που περικλείεται από αγκύλες.

3.4.5 Παράμετροι πρωτογενούς τύπου

Έστω η μέθοδος populationIn10() από την κλάση SpeciesFirstTry που ορίστηκε στην αρχή της ενότητας. Η μέθοδος αυτή επιστρέφει τον προβλεπόμενο πληθυσμό ενός είδους στα επόμενα 10 χρόνια. Θα ήταν όμως πιο πρακτικό αν η μέθοδος υπολόγιζε για έναν συγκεκριμένο, ακέραιο, αριθμό ετών και επέστρεφε τον προβλεπόμενο πληθυσμό γι' αυτό το χρονικό διάστημα. Για να γίνει αυτό χρειάζεται, με κάποιο τρόπο να μένει ένα κενό σε μια μέθοδο το οποίο να συμπληρώνεται από κάθε κλήση της με μια διαφορετική τιμή για τον αριθμό των ετών. Αυτό που μπορεί να χρησιμοποιηθεί ως κενό μέσα σε μια μέθοδο λέγεται τυπική μεταβλητή ή απλά **παράμετρος (parameter).** Βέβαια, πρόκειται για κάτι περισσότερο από ένα απλό κενό που συμπληρώνεται από κάποια τιμή όταν καλείται η μέθοδος.

Έτσι το πρόγραμμα θα μπορούσε να διαμορφωθεί ως εξής:

```
public class SpeciesSecondTryDemo
{
       public static void main(String[] args)
       {
              SpeciesSecondTry speciesOfTheMonth = new SpeciesSecondTry();
              int futurePopulation;
              System.out.println("Enter data on the Species of the Month:");
              speciesOfTheMonth.readInput();
              speciesOfTheMonth.writeOutput();
              futurePopulation = speciesOfTheMonth.populationIn10(10);
              System.out.println("In ten years the population will be " +
               futurePopulation);
              speciesOfTheMonth.name = "Klingon ox";
              speciesOfTheMonth.population = 10;
              speciesOfTheMonth.growthRate = 15;
              System.out.println("The new Species of the Month:");
```

```
            speciesOfTheMonth.writeOutput();
            System.out.println("In ten years the population will be: " +
            speciesOfTheMonth.populationIn10(10));
      }
}
```

και η κλάση ως εξής:

```
import java.util.*;
public class SpeciesSecondTry
{
      public String name;
      public int population;
      public double growthRate;
      public void readInput()
      {
            Scanner keyboard = new Scanner(System.in);
            System.out.println("What is the species' name?");
            name = keyboard.nextLine();
            System.out.println("What is the population of the species?");
            population = keyboard.nextInt();
            while (population < 0)
            {
                  System.out.println("Population cannot be negative.");
                  System.out.println("Reenter population:");
                  population = keyboard.nextInt();
            }
            System.out.println("Enter growth rate (percent increase per
                              year):");
            growthRate = keyboard.nextDouble();
      }
      public void writeOutput()
      {
            System.out.println("Name = " + name);
            System.out.println("Population = " + population);
            System.out.println("Growth rate = " + growthRate + "%");
      }
      public int projectedPopulation(int years)
      {
            double populationAmount = population;
            int count = years;
            while ((count > 0) && (populationAmount > 0))
            {
                  populationAmount = (populationAmount +
                  (growthRate/100) * populationAmount);
                  count--;
            }
            if (populationAmount > 0)
            {
                  return (int)populationAmount;
                  /** Το (int) μπροστά από την μεταβλητή είναι μια
                  *  μετατροπή τύπου.Αυτή η εντολή μετά την εκτέλεση
                  *  θα επιστρέψει το ακέραιο μέρος της μεταβλητής
                  */
            else
            return 0;
            }
      }
}
```

Η λέξη `years` λέγεται **τυπική παράμετρος (formal parameter)** ή απλά **παράμετρος (parameter)**. Μία τυπική παράμετρος χρησιμοποιείται μέσα στον ορισμό μεθόδου σαν υποκατάστατο για μια τιμή που θα ενσωματωθεί όταν κληθεί η μέθοδος, η οποία λέγεται **όρισμα (argument)** [σε κάποια βιβλία τα ορίσματα αναφέρονται σαν **πραγματικές παράμετροι (actual parameters)**].

Όταν γίνεται κλήση μια μεθόδου, το όρισμα μπαίνει στη θέση της τυπικής παραμέτρου σε οποιοδήποτε σημείου του ορισμού μεθόδου και αν εμφανίζεται αυτή. Είναι σημαντικό να ειπωθεί ότι μόνο η τιμή του ορίσματος χρησιμοποιείται σ' αυτήν την διαδικασία αντικατάστασης. Αν το όρισμα σε μία κλήση μεθόδου είναι μια μεταβλητή, τότε το όρισμα αντικαθιστάται από την τιμή της μεταβλητής και όχι από το όνομα της μεταβλητής. Δηλαδή αν ένα κομμάτι κώδικα που χρησιμοποιεί την κλάση `SpeciesSecondTry` είχε την ακόλουθη μορφή:

```
SpeciesSecondTry mySpecies = new SpeciesSecondTry();
int yearsCount = 12;
int futurePopulation;
futurePopulation = mySpecies.projectedPopulation(yearCount);
```

σαν όρισμα θα δοθεί η τιμή 12 και όχι η μεταβλητή `yearCount`. Επειδή χρησιμοποιείται μόνο η τιμή του ορίσματος, αυτή η μέθοδος αντικατάστασης των τυπικών παραμέτρων από τα ορίσματα είναι γνωστή ως μηχανισμός **κλήσης με τιμή (call-by-value)**. Στη `Java`, αυτός είναι ο μόνος τρόπος αντικατάστασης που χρησιμοποιείται με παραμέτρους πρωτογενούς τύπου, όπως είναι ο `int`, ο `double` και ο `char`. Όπως θα φανεί παρακάτω, οι παράμετροι ενός τύπου κλάσης, χρησιμοποιούν έναν λίγο διαφορετικό μηχανισμό αντικατάστασης αλλά μέχρι στιγμής αξίζει να αναφερθούν μόνο τα ορίσματα και οι παράμετροι των πρωτογενών τύπων όπως ο `int`, ο `double` και ο `char`.

Είναι δυνατόν να υπάρχουν παραπάνω του ενός τυπικές παράμετροι μέσα στις παρενθέσεις σε μια μέθοδο. Σε αυτήν την περίπτωση θα πρέπει να υπάρχει αντιστοιχία με τα ορίσματα κατά την κλήση της μεθόδου στον τύπο, στον αριθμό και στην σειρά των ορισμάτων σε σχέση με τις τυπικές παραμέτρους.

3.4.6 Οι προσδιορισμοί public και private

Μέσα σε έναν ορισμό κλάσης πριν από κάθε δήλωση μεταβλητής στιγμιοτύπου μπορεί να προηγείται ο προσδιορισμός `public` ή `private`. Αν μια μεταβλητή στιγμιοτύπου είναι `private`, τότε δεν μπορεί να γίνει καμία αναφορά στο όνομα της πουθενά παρά μόνο μέσα στους ορισμούς των μεθόδων της ίδιας κλάσης. Αν είναι `public` , τότε δεν υπάρχει κανένας περιορισμός για την χρήση του ονόματος της μεταβλητής στιγμιοτύπου. Αν ένας ορισμός μεθόδου χαρακτηρίζεται ως `private`, τότε το όνομα της μεθόδου δεν μπορεί να κληθεί έξω από τον ορισμό της κλάσης, ενώ όταν είναι `public` δεν κανένας περιορισμός στη χρήση της μεθόδου. Συνήθως, οι μεταβλητές στιγμιοτύπου χαρακτηρίζονται ως `private` και οι μέθοδοι ως `public`.

3.4.7 Μέθοδοι προσπέλασης και μεταβολής

Μία μέθοδος `public` η οποία διαβάζει και επιστρέφει τα δεδομένα από μία ή περισσότερες μεταβλητές στιγμιοτύπου λέγεται **μέθοδος προσπέλασης**. Τα ονόματα των μεθόδων προσπέλασης συνήθως αρχίζουν με τη λέξη `get`.

Μία μέθοδος `private` η οποία αλλάζει τα δεδομένα που είναι αποθηκευμένα σε μία ή περισσότερες μεταβλητές στιγμιοτύπου λέγεται **μέθοδος μεταβολής**. Τα ονόματα των μεθόδων μεταβολής συνήθως αρχίζουν με τη λέξη `set`.

Παρακάτω φαίνεται μια κλάση με μεθόδους προσπέλασης και μεταβολής:

```
import java.util.*;
public class SpeciesFourthTry
{
       private String name;
       private int population;
       private double growthRate;
       /** Οι ορισμοί των μεθόδων readInput, writeOutput,
          projectedPopulation μπαίνουν εδώ. Είναι ίδιοι με τους ορισμούς
          στη κλάση SpeciesSecondTry που υπάρχει παραπάνω */
       public void set(String newName, int newPopulation,
                       double newGrowthRate)
       {
              name = newName;
              if (newPopulation >= 0) {
                     population = newPopulation;
              }
              else {
                     System.out.println("ERROR: Using a negative
                       population.");
                     System.exit(0);
              }
              growthRate = newGrowthRate;
       }
       public String getName();
       {
              return name;
       }
       public int getPopulation();
       {
              return population;
       }
       public double getGrowthRate();
       {
              return growthRate;
       }
}
```

3.4.8 Μεταβλητές τύπου κλάσης και Αντικείμενα

Οι μεταβλητές τύπου κλάσης προσδιορίζουν (ονομάζουν) αντικείμενα με διαφορετικό τρόπο σε σχέση με τον τρόπο που μεταβλητές πρωτογενών τύπων αποθηκεύουν τις τιμές τους. Με αυτή τη φράση εννοείται ότι η μεταβλητή τύπου κλάσης περιέχει την θέση μνήμης αυτού του αντικειμένου. Κάθε μεταβλητή, είτε είναι ενός πρωτογενούς τύπου είτε είναι ενός τύπου κλάσης, υλοποιείται σαν μία θέση μνήμης που έχει εκχωρηθεί στη μεταβλητή. Αν, όμως, η μεταβλητή είναι τύπου κλάσης, τότε ένα αντικείμενο που παίρνει το όνομα του από τη μεταβλητή αποθηκεύεται σε κάποια άλλη θέση μέσα στη μνήμη και η διεύθυνση μνήμης της θέσης στην οποία βρίσκεται το αντικείμενο είναι αυτό που αποθηκεύεται μέσα στη μεταβλητή που προσδιορίζει το αντικείμενο. Η διεύθυνση μνήμης στην οποία αποθηκεύεται ένα αντικείμενο λέγεται **αναφορά (reference)** στο αντικείμενο.

Το γεγονός ότι οι μεταβλητές τύπου κλάσης περιέχουν αναφορές μπορεί να οδηγήσει σε απρόσμενα αποτελέσματα. Οι μεταβλητές τύπου κλάσης συμπεριφέρονται πολύ διαφορετικά από τις μεταβλητές ενός πρωτογενούς τύπου. Έστω το παρακάτω κομμάτι κώδικα με το οποίο θα μπορούσε να ξεκινάει το τμήμα `main` ενός προγράμματος:

```
SpeciesFourthTry klingonSpecies, earthSpecies;
klingonSpecies = new SpeciesFourthTry();
earthSpecies = new SpeciesFourthTry();
int n, m;
n = 42;
m = n;
```

όπως θα περίμενε κανείς, υπάρχουν δύο μεταβλητές τύπου `int`, η `n` και η `m` . Και οι δύο έχουν την τιμή 42 αλλά αν αλλάξει η μία, η τιμή της άλλης θα εξακολουθεί να είναι 42. Για παράδειγμα αν το πρόγραμμα συνεχίζει με την εντολή:

```
n = 99;
System.out.println(n + " and " + m);
```

τότε η έξοδος θα ήταν:

```
99 and 42
```

μέχρι τώρα δεν υπάρχει κάτι παράξενο. Έστω όμως, ότι το πρόγραμμα συνεχίζει ως εξής:

```
klingonSpecies.set("Klingon ox", 10, 15);
earthSpecies.set("Black rhino", 11, 2);
earthSpecies = klingonSpecies;
earthSpecies.set("Elephant", 100, 12);
System.out.println("earthSpecies:");
earthSpecies.writeOutput();
System.out.println("klingonSpecies:");
klingonSpecies.writeOutput();
```

πιθανώς θα περιμέναμε ότι η `klingonSpecies` είναι το "Klingon ox" και η `earthSpecies` είναι "Elephant", αλλά η έξοδος θα μας ξαφνιάσει. Η έξοδος λοιπόν είναι η εξής:

```
earthSpecies:

Name = Elephant

Population = 100

Growth rate = 12%

klingonSpecies:

Name = Elephant

Population = 100

Growth rate = 12%
```

Αυτό που συμβαίνουν είναι ότι υπάρχουν δύο μεταβλητές, `klingonSpecies` και `earthSpecies` , αλλά μόνο ένα αντικείμενο. Και οι δύο μεταβλητές περιέχουν την ίδια αναφορά με αποτέλεσμα και οι δύο να προσδιορίζουν το ίδιο αντικείμενο.

Κάθε αντικείμενο αποθηκεύεται σε κάποια θέση στη μνήμη του υπολογιστή, η οποία έχει κάποια διεύθυνση. Οι παραπάνω δύο μεταβλητές είναι συνηθισμένες μεταβλητές, όπως είναι μια μεταβλητή `int` , απλά αποθηκεύουν διευθύνσεις μνήμης για αντικείμενα της κλάσης `SpeciesFourthTry` . Η εκτέλεση μιας εντολής εκχώρησης όπως η παρακάτω:

```
earthSpecies = klingonSpecies;
```

απλά αντιγράφει τη διεύθυνση μνήμης που περιέχει η `klingonSpecies` στη μεταβλητή `earthSpecies` και έτσι περιέχουν και οι δύο την ίδια θέση μνήμης και συνεπώς και οι δύο προσδιορίζουν το ίδιο αντικείμενο.

3.4.8.1 Ο τελεστής new

Οι μεταβλητές ενός τύπου κλάσης λειτουργούν διαφορετικά από τις μεταβλητές ενός πρωτογενούς τύπου. Μία μεταβλητή ενός πρωτογενούς τύπου περιέχει μία τιμή του τύπου αυτού ενώ μία μεταβλητή ενός τύπου κλάσης δεν περιέχει το αντικείμενο της κλάσης αυτής αλλά τη διεύθυνση μνήμης στην οποία βρίσκεται το αντικείμενο αυτό. Η δήλωση

```
SpeciesFourthTry s;
```

δημιουργεί μία μεταβλητή `s` η οποία μπορεί να αποθηκεύσει μία διεύθυνση μνήμης. Σε αυτό το σημείο, το πρόγραμμα έχει ένα μέρος για να αποθηκεύσει μία διεύθυνση μνήμης αλλά δεν έχει μέρος για να αποθηκεύσει τα δεδομένα των μεταβλητών στιγμιοτύπου ενός αντικειμένου τύπου `SpeciesFourthTry` . Για να έχει μία θέση μνήμης στην οποία να μπορεί να αποθηκεύσει τις τιμές των μεταβλητών στιγμιοτύπου, το πρόγραμμα πρέπει να χρησιμοποιεί τον τελεστή `new` . Δηλαδή εάν η παραπάνω εντολή διαμορφωνόταν ως εξής:

```
s = new SpeciesFourthTry();
```

τότε το αποτέλεσμα θα είναι να εκχωρηθεί μια θέση μνήμης σε ένα αντικείμενο τύπου `SpeciesFourthTry` και η διεύθυνση αυτής της θέσης μνήμης θα καταχωρηθεί για την μεταβλητή `s` . Θα μπορούσε να ειπωθεί πολύ χονδρικά πως ο τελεστής `new` δημιουργεί τις μεταβλητές στιγμιοτύπου του αντικειμένου.

3.4.8.2 Έλεγχος ισότητας μεταξύ μεταβλητών τύπου κλάσης

Προηγουμένως έγινε αναφορά σε κάποιο απρόβλεπτο αποτέλεσμα, όταν χρησιμοποιείται τελεστής εκχώρησης ανάμεσα σε δύο μεταβλητές τύπου κλάσης. Με τον ίδιο απρόβλεπτο τρόπο συμπεριφέρεται ο έλεγχος για ισότητα. Έστω ότι η κλάση `SpeciesFourthTry` ορίζεται με τον τρόπο που φαίνεται παραπάνω και έστω ότι υπάρχει ο εξής κώδικας προγράμματος:

```
SpeciesFourthTry klingonSpecies = new SpeciesFourthTry();
SpeciesFourthTry earthSpecies = new SpeciesFourthTry();
klingonSpecies.set(“Klingon ox”, 10, 15);
earthSpecies.set(“Klingon ox”, 10, 15);
if (klingonSpecies == earthSpecies) {
      System.out.println(“They are EQUAL.”);
}
else {
      System.out.println(“They are NOT equal.”);
}
```

Το παραπάνω κομμάτι κώδικα θα δώσει:

```
They are NOT equal.
```

Το πρόβλημα είναι ότι παρά το γεγονός ότι αυτά τα δύο είδη είναι ίσα κατά μία σαφή έννοια, μία μεταβλητή τύπου κλάσης δεν περιέχει τίποτε περισσότερο από μία διεύθυνση μνήμης. Έτσι, μέσα στη μνήμη υπάρχουν δύο αντικείμενα του τύπου `SpeciesFourthTry` που αναπαριστούν το ίδιο είδος στον πραγματικό κόσμο και τα οποία έχουν διαφορετικές διευθύνσεις μνήμης. Ο τελεστής == ελέγχει, δηλαδή, για ένα είδος ισότητας το οποίο, όμως, δεν είναι αυτό που ενδιαφέρει στη συγκεκριμένη περίπτωση.

Το συμπέρασμα αυτής της υποενότητας είναι πως για να γίνει έλεγχος ισότητας δύο μεταβλητών τύπου κλάσης, θα πρέπει να χρησιμοποιηθεί η μέθοδος `equals` η οποία θα ελέγξει τα αντικείμενα για να δει αν είναι ίσα.

3.4.9 Στατικές μέθοδοι και στατικές μεταβλητές

Οι στατικές μέθοδοι και οι στατικές μεταβλητές είναι μέθοδοι και μεταβλητές που ανήκουν σε μία κλάση ως σύνολο και δεν απαιτούν κανένα αντικείμενο. Πιο κάτω φαίνεται ο κώδικας μιας κλάσης που περιέχει στατικές μεθόδους και μεταβλητές:

```
public class CircleFirstTry
{
	public static final double PI = 3.14159;
	// Η μεταβλητή PI είναι μια στατική μεταβλητή.
	public static double area(double radius)
	{
		return (PI*radius*radius);
	}
	public static double circumference(double radius)
	{
		return (PI*(radius + radius));
	}
}
```

3.4.9.1 Στατικές μέθοδοι

Μερικές φορές χρειάζεται μία μέθοδος που δεν απαιτεί τη χρήση οποιουδήποτε είδους αντικειμένου. Για παράδειγμα, μπορεί να χρειάζεται μία μέθοδο για να υπολογιστεί το μέγιστο δύο ακεραίων ή μία μέθοδο για να υπολογιστεί η τετραγωνική ρίζα ενός αριθμού ή μία μέθοδος για να γίνει μετατροπή ενός χαρακτήρα γράμματος από μικρό σε κεφαλαίο. Καμία από αυτές τις μεθόδους δεν έχει ένα προφανές αντικείμενο στο οποίο θα έπρεπε να ανήκει. Στις περιπτώσεις αυτές, μπορεί να οριστεί η μέθοδος ως **στατική (static)**. Για να γίνει κλήση μιας στατικής μεθόδου, δεν χρειάζεται η χρήση κάποιου αντικειμένου, αλλά χρησιμοποιείται κανονικά το όνομα της κλάσης. Το παρακάτω κομμάτι κώδικα μέσα σε μία κλάση θα μπορούσε να πραγματοποιήσει κλήση μιας εκ των δύο στατικών μεθόδων της κλάσης `CircleFirstTry`:

```
double radius = 2.3;
System.out.println("A circle of radius " + radius + " inches");
System.out.println("has an area of " + CircleFirstTry.area(radius) +
```

```
                    " square inches.");
```

η εκτέλεση αυτής της εντολής θα έδινε:

```
A circle of radius 2,3 inches

has an area of 16.61901 square inches.
```

Θα μπορούσε κανείς να δημιουργήσει ένα αντικείμενο της κλάσης `CircleFirstTry` και έπειτα να το χρησιμοποιήσει για να καλέσει μία στατική μέθοδο, όμως αυτό δεν συνιστάται γιατί θα μπορούσε να προκαλέσει σύγχυση και έτσι σχεδόν πάντα χρησιμοποιείται το όνομα της κλάσης όταν καλείται μια στατική μέθοδος. Ο ορισμός μια στατικής μεθόδου γίνεται όποτε γίνονται και οι υπόλοιπες μέθοδοι, οι οποίες έχουν ήδη περιγραφεί, μόνο που απαιτείται να προστεθεί η δεσμευμένη λέξη `static` στον ορισμό της μεθόδου.

Μέσα στον ορισμό μιας στατικής μεθόδου δεν μπορεί να γίνει οτιδήποτε που να αναφέρεται σε ένα καλούν αντικείμενο, όπως για παράδειγμα να προσπελαστεί μια μεταβλητή στιγμιοτύπου. Αυτό είναι απόλυτα λογικό επειδή μια στατική μέθοδος μπορεί να κληθεί χωρίς τη χρήση καλούντος αντικειμένου και επομένως μπορεί να κληθεί όταν δεν υπάρχει μεταβλητή στιγμιοτύπου στην οποία να μπορεί να γίνει αναφορά.

3.4.9.2 Ανακατεύοντας στατικές και Μη στατικές μεθόδους

Έστω η παρακάτω κλάση:

```
import java.util.*;
public class PlayCircle
{
	public static final double PI = 3.14159;
	public double diameter;
	public void setDiameter(double newDiameter)
	{
		diameter = newDiameter;
	}
	public static double area(double radius)
	{
		return (PI*radius*radius);
	}
	public void showArea();
	{
		System.out.println("Area is " + area(diameter/2));
	}
	public static void areaDialog()
	{
		Scanner keyboard = new Scanner(System.in);
		System.out.println("Enter diameter of circle:");
		double newDiameter = keyboard.nextDouble();
		PlayCircle c = new PlayCircle();
		c.setDiameter(newDiameter);
		c.showArea();
	}
}
```

όπως φαίνεται και στον κώδικα παραπάνω, για να γίνει κλήση μιας μη στατικής μεθόδου μέσα στον ορισμό μίας στατικής μεθόδου, είναι απαραίτητο να δημιουργηθεί ένα καλούν αντικείμενο, όπως το `c` , χρησιμοποιώντας τον τελεστή `new` . Ενώ η στατική μέθοδος

`areaDialog()` θα μπορούσε να κληθεί, χρησιμοποιώντας μόνο το όνομα της κλάσης, με την εξής εντολή:

```
PlayCircle.areaDialog();
```

3.4.10 Δομητές

Όταν δημιουργείται ένα αντικείμενο μίας κλάσης, πολλές φορές υπάρχει η επιθυμία να γίνονται κάποιες συγκεκριμένες λειτουργίες αρχικοποίησης, όπως για παράδειγμα η απόδοση τιμών στις μεταβλητές στιγμιοτύπου. Ο δομητής είναι ένας ειδικός τύπος μεθόδου που έχει σχεδιαστεί για να κάνει τέτοιου είδους αρχικοποιήσεις.

Πιο συγκεκριμένα, ο **δομητής (constructor)** είναι μια μέθοδος που καλείται όταν δημιουργείται ένα αντικείμενο της κλάσης χρησιμοποιώντας τον τελεστή `new` . Οι δομητές χρησιμοποιούνται για να αρχικοποιήσουν αντικείμενα. Ένας δομητής πρέπει να έχει το ίδιο όνομα με το όνομα της κλάσης στην οποία ανήκει. Τα ορίσματα για ένα δομητή δίδονται μέσα σε παρενθέσεις, όπως συμβαίνει και για όλες τι άλλες μεθόδους.

3.4.11 Πακέτα

Ένα **πακέτο (package)** δεν είναι τίποτα άλλο από μία συλλογή κλάσεων που έχουν συγκεντρωθεί μέσα σε έναν κατάλογο και τους έχει δοθεί ένα όνομα πακέτου. Μέσα σε ένα πακέτο, οι κλάσεις μπαίνουν σε ένα διαφορετικό αρχείο το οποίο έχει το ίδιο όνομα με την κλάση, όπως ακριβώς έχει γίνει μέχρι τώρα. Η μόνη διαφορά είναι ότι κάθε αρχείο έχει την παρακάτω γραμμή στην αρχή του αρχείου:

```
package ‘Ονομα_Πακέτου
```

Πριν από αυτήν την γραμμή το μόνο που μπορεί να υπάρχει είναι κάποιες κενές γραμμές ή κάποια σχόλια, αλλά τίποτε άλλο. Επίσης, το όνομα του πακέτου γράφεται συνήθως με μικρά γράμματα και αν είναι παραπάνω από μία λέξεις, τότε συνήθως διαχωρίζονται με μία τελεία.

Οποιοδήποτε πρόγραμμα ή ορισμός κλάσης μπορεί να χρησιμοποιήσει όλες τις κλάσεις που βρίσκονται στο πακέτο, έχοντας την κατάλληλη εντολή `import` στην αρχή του αρχείου που περιέχει το πρόγραμμα ή τον ορισμό κλάσης. Αυτό συμβαίνει και για την περίπτωση που το πρόγραμμα ή ο ορισμός κλάσης δεν βρίσκονται στον ίδιο κατάλογο με τις κλάσεις του πακέτου. Για παράδειγμα, αν θέλουμε να χρησιμοποιήσουμε τις κλάσεις που βρίσκονται στο πακέτο με όνομα `general.utilities` , τότε θα πρέπει να συμπεριλάβουμε την εξής εντολή στην αρχή του αρχείου που γράφουμε:

```
import general.utilities.*;
```

Η τελεία και ο αστερίσκος στο τέλος της εντολής σημαίνουν ότι εισάγετε όλες τις κλάσεις που βρίσκονται σε αυτό το πακέτο. Βεβαίως, υπάρχει η δυνατότητα να εισαχθεί μία μόνο κλάση από το πακέτο χρησιμοποιώντας το όνομα της κλάσης στη θέση του αστερίσκου.

3.4.11.1 Ονόματα πακέτων και κατάλογοι

Ένα όνομα πακέτου δεν είναι ένα αυθαίρετο όνομα αλλά δηλώνει στο μεταγλωττιστή πού θα βρει τις κλάσεις που περιέχονται στο πακέτο. Ουσιαστικά, το όνομα πακέτου λέει στο μεταγλωττιστή το όνομα διαδρομής για τον κατάλογο που περιέχει τις κλάσεις του πακέτου.

Για να βρει τον κατάλογο για ένα πακέτο, η Java χρειάζεται δύο πράγματα: το όνομα του πακέτου και τους κατάλογους που αναφέρονται στην τιμή της μεταβλητής διαδρομής κλάσεων.

Η τιμή της **μεταβλητής διαδρομής κλάσεων (class path variable)** λέει στη Java από πού να ξεκινήσει την αναζήτηση ενός πακέτου και γι' αυτό το λόγο θα ξεκινήσουμε με αυτή. Η μεταβλητή διαδρομής κλάσεων δεν είναι μία μεταβλητή της Java αλλά είναι μέρος του λειτουργικού συστήματος και περιέχει τα ονόματα διαδρομής μίας λίστας καταλόγων. Όταν η Java ψάχνει να βρει ένα πακέτο, αρχίζει την αναζήτηση από αυτούς του καταλόγους. Ας ονομάσουμε αυτούς τους καταλόγους **βασικούς καταλόγους διαδρομής κλάσεων (class path base directions)**.

3.5 Πίνακες

Έστω ότι θέλουμε να υπολογίσουμε τον μέσο όρο από επτά τιμές θερμοκρασίας. Το παρακάτω κομμάτι κώδικα θα μπορούσε να μας δώσει το αποτέλεσμα που θέλαμε:

```
int count;
double next, sum, average;
Scanner keyboard = new Scanner(System.in);
System.out.println("Enter 7 numbers:");
sum = 0;
for (count = 0; count < 7; count++)
{
       next = keyboard.nextDouble();
       sum = sum + next;
}
average = sum/7;
```

Το παραπάνω κομμάτι κώδικα θα δώσει το αποτέλεσμα που θέλουμε.

Αν όμως το πρόβλημα είναι να υπολογίσουμε και άλλα στοιχεία γι' αυτές τις θερμοκρασίες, όπως για παράδειγμα, πόσες και ποιες είναι κάτω από τον μέσο όρο; Και αν ο αριθμός των θερμοκρασιών αναφερόταν για ένα έτος, δηλαδή είχαμε 365 τιμές θερμοκρασιών; Θα έπρεπε, αρχικά, να καταχωρήσουμε τιμές σε 365 μεταβλητές! Σε κάποιες περιπτώσεις αυτό μπορεί να είναι μεγάλο πρόβλημα. Πόσο μάλλον να δηλωθούν σε ένα πρόγραμμα τόσες πολλές μεταβλητές για ένα σχετικά μικρό και «εύκολο» πρόβλημα. Οι πίνακες μας δίνουν την δυνατότητα να δηλώσουμε ένα σύνολο σχετικών μεταξύ τους μεταβλητών με ένα σαφώς πιο κομψό τρόπο. Ένας **πίνακας (array)** είναι ουσιαστικά μία λίστα μεταβλητών, αλλά διαχειρίζεται την ονομασία των μεταβλητών αυτών με έναν ωραίο και συμπαγή τρόπο.

3.5.1 Δημιουργία και Προσπέλαση Πινάκων

Στην Java, ένας πίνακας είναι ένα ειδικός τύπος αντικειμένου τον οποίο μπορούμε να θεωρήσουμε ως ένα σύνολο μεταβλητών που είναι όλες του ίδιου τύπο. Για παράδειγμα, όσον αφορά το παράδειγμα της εισαγωγής θα μπορούσαμε να δημιουργήσουμε έναν πίνακα που να περιείχε ένα σύνολο επτά μεταβλητών τύπου double ως εξής:

```
double[] temperature = new double[7];
```

Η παραπάνω εντολή είναι σαν να δηλώνουμε επτά μεταβλητές τύπου double . Αξίζει να σημειωθεί σε αυτό το σημείο ότι η αρίθμηση των δεικτών των πινάκων για κάθε στοιχείο τους, ξεκινάει από το 0 και όχι από το 1. Αν λοιπόν, για παράδειγμα, δίναμε στις μεταβλητές κατά σειρά τις τιμές: 32 , 30 , 25.7 , 26 , 24 , 31.5 , 29 τότε θα μπορούσαμε να φανταστούμε την μορφή του πίνακα ως εξής:

0	1	2	3	4	5	6
32	30	25.7	26	24	31	29

Πίνακας 3.10

Κάθε μία από τις παραπάνω επτά μεταβλητές μπορεί να χρησιμοποιηθεί όπως οποιαδήποτε άλλη μεταβλητή double στην Java. Για παράδειγμα όλες οι παρακάτω εκφράσεις επιτρέπονται:

```
temperature[3] = 32;
temperature[6] = temperature[3] + 5;
System.out.println(temperature[6]);
```

Μέσα στις αγκύλες επιτρέπεται να υπάρχει οτιδήποτε αντιστοιχεί σε ακέραιο αριθμό ο οποίος βρίσκεται μέσα στα αποδεκτά όρια των δεικτών του πίνακα. Στο παραπάνω παράδειγμα δηλαδή, θα μπορούσε να είναι αποδεκτή η παρακάτω εντολή:

```
temperature[n + 2] //Όπου το n είναι κάποιος ακέραιος αριθμός
```

Σε αυτό το σημείο θα δοθούν οι ακριβείς ορολογίες σχετικά με τους πίνακες:

Οι μεταβλητές μέσα σε μια ακέραια έκφραση μέσα στις αγκύλες, συνήθως λέγονται **στοιχεία (elements)** πίνακα, αλλά μπορεί να συναντηθούν και με τον όρο **μεταβλητές με δείκτη (indexed ή subscripted variables)**. Η ακέραια έκφραση μέσα στις αγκύλες λέγεται **δείκτης (index ή subscript)**. Ο τύπος των στοιχείων, που στο παραπάνω παράδειγμα είναι double, λέγεται **βασικός τύπος (based type)** του πίνακα. Ο αριθμός των στοιχείων ενός πίνακα λέγεται **μήκος (length)** ή **μέγεθος (size)** του πίνακα. Αν μια τιμή δείκτη δεν είναι αποδεκτή με βάση το μέγεθος του πίνακα, τότε λέμε ότι ο δείκτης είναι **εκτός ορίων (out of bounds)** ή **μη έγκυρος (invalid)**.

Ένας πίνακας μπορεί να αρχικοποιηθεί την στιγμή που δηλώνεται. Για παράδειγμα με την εξής εντολή:

```
double[] reading = {3.3, 15.8, 9.7};
```

Σε αυτήν την περίπτωση το μήκος του πίνακα τίθεται ίσο με την ελάχιστη τιμή που μπορεί να χωρέσει αυτές τις τιμές που δίνονται. Στο παραπάνω παράδειγμα δηλαδή, το μήκος του πίνακα reading είναι ίσο με την τιμή 3. Αν δεν γίνει αρχικοποίηση από τον προγραμματιστή, τότε μπορεί να αρχικοποιηθούν σε μία προεπιλεγμένη τιμή του βασικού

τύπου. Παρόλα αυτά συνιστάται η αρχικοποίηση με κάποιον τρόπο από τον προγραμματιστή.

3.5.2 Ορίσματα για τη Μέθοδο main

Η επικεφαλίδα της μεθόδου `main` έχει ως εξής:

```
public static void main(String[] args)
```

Το τμήμα `String[] args` την κάνει να φαίνεται σαν να είναι η `args` μία παράμετρος για έναν πίνακα με βασικό τύπου `String`. Όχι μόνο φαίνεται έτσι, αλλά είναι πράγματι γεγονός ότι η μέθοδος `main` δέχεται ως όρισμα έναν πίνακα τιμών τύπου `String`. Όμως δεν έχουμε δώσει ποτέ στην main ένα όρισμα πίνακα, ή οποιοδήποτε άλλο είδος ορίσματος, όταν εκτελούσαμε τα προγράμματα μας.

Όπως έχουμε πει σε κάποια ενότητα πιο πάνω, μία κλήση της `main` είναι ένα ξεχωριστό είδος κλήση. Όταν τρέχει ένα πρόγραμμα, παρέχεται αυτόματα στη `main` ως προεπιλεγμένο όρισμα ένας προεπιλεγμένος πίνακας αλφαριθμητικών. Αλλά, υπάρχει η δυνατότητα να δοθούν παραπάνω ορίσματα αλφαριθμητικών και αυτά τα ορίσματα αλφαριθμητικών θα γίνουν αυτόματα στοιχεία του ορίσματος πίνακα που παρέχεται στη `main`. Αυτό κανονικά γίνεται τρέχοντας το πρόγραμμα από τη γραμμή εντολών του λειτουργικού συστήματος, ως εξής:

```
Java TestProgram Kostas Michalakis
```

Αυτή η εντολή θα θέσει το `args[0]` ίσο με “Kostas” και το `args[1]` ίσο με “Michalakis” και αυτά τα δύο στοιχεία μπορούν να χρησιμοποιηθούν στη μέθοδο `main`, όπως φαίνεται και στο παρακάτω παράδειγμα:

```
public class TestProgram
{
	public static void main(String[] args)
	{
		System.out.println(“Hello “ + args[0] + “ “ + args[1]);
	}
}
```

Το παραπάνω πρόγραμμα, αφού μεταγλωττιστεί, και μετά εκτελεστεί από τη γραμμή εντολών όπως παρακάτω:

```
java TestProgram Mechanical Engineer
```

ή έξοδος που θα δώσει το πρόγραμμα θα είναι:

```
Hello Mechanical Engineer
```

Επειδή το όνομα `args` είναι μια παράμετρος, μπορεί να χρησιμοποιηθεί οποιοδήποτε όνομα (εκτός από δεσμευμένη λέξη) στη θέση του `args` και η σημασία της εντολής θα μείνει αναλλοίωτη (εφόσον φυσικά αλλαχθεί το όνομα `args` οπουδήποτε εμφανίζεται μέσα στο σώμα της `main`). Ωστόσο παραδοσιακά, χρησιμοποιείται το όνομα `args` για την παράμετρο αυτή.

3.5.3 Χρήση των τελεστών = και == με Πίνακες

Οι πίνακες είναι αντικείμενα και επομένως οι τελεστές εκχώρησης = και ισότητας == συμπεριφέρονται με τους πίνακες με τον ίδιο τρόπο που συμπεριφέρονται και με τα διάφορα αντικείμενα που έχουμε δει μέχρι τώρα. Για να καταλάβει κανείς πως ισχύει αυτό με τους πίνακες, πρέπει να γνωρίζει κάποια πράγματα για το πώς αποθηκεύονται τα περιεχόμενα ενός πίνακα στην μνήμη του υπολογιστή. Το σημαντικό σε αυτό το σημείο που πρέπει να αναφερθεί, είναι πως τα περιεχόμενα όλου του πίνακα, δηλαδή τα περιεχόμενα όλων των στοιχείων του, αποθηκεύονται όλα μαζί σε ένα τμήμα της μνήμης του υπολογιστή έτσι ώστε η θέση των περιεχομένων ενός πίνακα να καθορίζεται από μία θέση μνήμης. Όπως έχει αναφερθεί σε προηγούμενη ενότητα, μία μεταβλητή για ένα αντικείμενο περιέχει στην πραγματικότητα την διεύθυνση μνήμης του αντικειμένου. Ο τελεστής εκχώρησης αντιγράφει αυτή την διεύθυνση μνήμης.

Έτσι λοιπόν, ο τελεστής ισότητας == ελέγχει αν δύο πίνακες είναι αποθηκευμένοι στο ίδιο σημείο μέσα στην μνήμη του υπολογιστή. Δηλαδή, το παρακάτω κομμάτι κώδικα:

```
int[] a = new int[3];
int[] b = new int[3];
int i;
for (i = 0; i < a.lenght; i++) {
      a[i] = i;
      for (i = 0; i < a.length; i++) {
            b[i] = i;
            if (b == a) {
                  System.out.println(“Equal by == “);
            }
            else {
                  System.out.println(“Not equal by == “);
            }
      }
}
```

το παραπάνω κομμάτι κώδικα θα δώσει την ακόλουθη έξοδο:

```
Not equal by ==
```

δηλαδή, ενώ οι παραπάνω πίνακες a και b περιέχουν τους ίδιους ακέραιους μέσα στα ίδια στοιχεία, η έξοδος δεν επαληθεύει κάτι τέτοιο. Αυτό συμβαίνει επειδή οι πίνακες a και b είναι αποθηκευμένοι σε διαφορετικά σημεία της μνήμης και ο τελεστής == κάνει έλεγχο για ίδιες θέσεις μνήμης. Για να ελεγχθούν δύο πίνακες για το αν περιέχουν τα ίδια στοιχεία, μπορεί να οριστεί μία μέθοδος `equals` για τους πίνακες, όπως ακριβώς ορίσαμε μία μέθοδο `equals` για μια κλάση.

Μία μεταβλητή πίνακα λοιπόν, περιέχει μόνο την διεύθυνση μνήμης στην οποία είναι αποθηκευμένος ο πίνακας. Και για τους πίνακες ισχύει ότι ισχύει για τις θέσεις μνήμης, τα αντικείμενα και τις μεταβλητές και ειπώθηκαν σε προηγούμενο κεφάλαιο. Το αν πρέπει οι πίνακες να θεωρούνται αντικείμενα δεν είναι απόλυτα σαφές γενικότερα στον προγραμματισμό και αυτό είναι αποτέλεσμα κυρίως μίας ακαδημαϊκής διαμάχης, όμως η στην `Java`, οι πίνακες είναι *επισήμως αντικείμενα*. Έτσι κάθε φορά που η τεκμηρίωση της `Java` αναφέρει ότι κάτι ισχύει για όλα τα αντικείμενα, ισχύει επίσης και για τους πίνακες.

3.5.4 Ταξινόμηση πινάκων

Έστω ότι έχουμε έναν πίνακα και θέλουμε να ταξινομηθούν με κάποιον τρόπο. Για παράδειγμα, να θέλουμε να ταξινομηθεί με αύξουσα ή με φθίνουσα σειρά ή μπορεί να θέλουμε να ταξινομήσουμε έναν πίνακα αλφαριθμητικών με αλφαβητική σειρά. Στην ενότητα αυτή θα εξετάσουμε έναν απλό αλγόριθμο και θα δώσουμε μία υλοποίηση σε Java αυτού του αλγορίθμου. Ας υποθέσουμε λοιπόν πως θέλουμε να ταξινομήσουμε τον παρακάτω πίνακα τιμών Πίνακας 3.11 τύπου int:

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
7	6	11	17	3	15	5	19	30	14

Πίνακας 3.11

3.5.4.1 Ταξινόμηση επιλογής

Αυτός ο αλγόριθμος λέγεται αλγόριθμος **ταξινόμησης επιλογής (selection sort)** και είναι από τους πιο ευκολονόητους αλγόριθμους ταξινόμησης και μια υλοποίηση του για τον πίνακα a φαίνεται πιο κάτω:

```
public class SelectionSort
{
	//*
	Προσυνθήκη: Κάθε στοιχείο του πίνακα έχει μία τιμή.
	Λειτουργία: Ταξινομεί τον a με σειρά αύξουσα.
	*/
	public static void sort(int[] a)
	{
		int index, indexOfNextSmallest;
		for (index = 0; index < a.length – 1; index++)
		{
			//Βάλε την σωστή τιμή στο στοιχείο a[index]
			indexOfNextSmallest = indexOfSmallest(index, a);
			interchange(index, indexOfNextSmallest, a);
			//a[0] <= a[1] <= ... <= a[index] και αυτά είναι
			//τα μικρότερα στοιχεία του αρχικού πίνακα.
			//Οι θέσεις που απομένουν περιέχουν τα υπόλοιπα
			//στοιχεία του αρχικού πίνακα.
		}
	}
	/**
	Επιστρέφει το δείκτη της μικρότερης τιμής μεταξύ των a[startIndex],
	a[startIndex + 1], ... a[a.length – 1]
	*/
	private static int indexOfSmallest(int startIndex, int[] a)
	{
		int min = a[startIndex];
		int indexOfMin = startIndex;
		int Index;
		for (Index = startIndex + 1; index < a.length; index++)
		{
			if (a[index] < min)
			{
				min = a[index];
				indexOfMin = index;
				//Η min είναι η μικρότερη από τους
				a[startIndex] μέχρι a[index]
			}
		return indexOfMin;
	}
	/**
	Προσυνθήκη: Οι i και j είναι έγκυροι δείκτες για τον πίνακα a.
	Μετασυνθήκη: Οι τιμές των a[i] και j[j] έχουν αντιμετατεθεί.
	*/
	private static void interchange(intt i, int j, int[] a)
```

```
        {
                int temp;
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;  //αρχική τιμή του a[i]
        }
}
```

Και το πρόγραμμα που υλοποιεί την παραπάνω κλάση:

```
public class SelectionSortDemo
{
        public static void main(String[] args)
        {
                int[] b = {7, 6, 11, 17, 3, 15, 5, 19, 30, 14};
                System.out.println("Array values before sorting:");
                int i;
                for (i = 0; i < b.length; i++)
                {
                        System.out.print(b[i] + " ");
                 }
                System.out.println( );
                SelectionSort.sort(b);
                System.out.println("Array values after sorting:");
                for (i = 0; i < b.length; i++)
                {
                        System.out.print(b[i] + " ");
                }
                System.out.println( );
        }
}
```

Το παραπάνω πρόγραμμα θα δώσει το εξής αποτέλεσμα:

```
Array value before sorting:

7 6 11 17 3 15 5 19 30 14

Array values after sorting:

3 5 6 7 11 14 15 17 19 30
```

Αυτό που ουσιαστικά γίνεται στο παραπάνω κομμάτι κώδικα είναι η αντιμετάθεση στοιχείων του πίνακα. Αρχικά ο αλγόριθμος εντοπίζει το μικρότερο ακέραιο στοιχείο του πίνακα καθώς και την θέση του. Στη συνέχεια τοποθετεί στην αρχή αυτό το στοιχείο και στην θέση του τοποθετεί την τιμή του πρώτου στοιχείου του πίνακα (αρχικά). Έπειτα συνεχίζει με το δεύτερο στοιχείο του πίνακα κ.ο.κ , μέχρι που στο τέλος μπορεί ενδεχομένως να μείνουν στοιχεία όπου δεν θα χρειάζεται να ανταλλάξουν θέση. Έτσι δεν χάνεται κανένα στοιχείο του πίνακα. Κάθε αλγόριθμος που χρησιμοποιεί αυτό το είδος αντιμετάθεσης τιμών λέγεται **αλγόριθμος ταξινόμησης με ανταλλαγή (interchange sorting algorithm)**. Άρα, ο αλγόριθμος ταξινόμησης επιλογής είναι ένας αλγόριθμος ταξινόμησης με ανταλλαγή.

Υπάρχουν αρκετοί αλγόριθμοι ταξινόμησης, πολλοί από τους οποίους είναι πιο αποτελεσματικοί (και πιο πολύπλοκοι) από την επιλογή ταξινόμησης.

3.5.4.2 Άλλοι Αλγόριθμοι Ταξινόμησης

Ο αλγόριθμος να ταξινόμησης επιλογής δεν είναι ο πιο αποτελεσματικός αλγόριθμος ταξινόμησης. Στην πραγματικότητα, είναι σαφώς λιγότερο αποτελεσματικός από πολλούς άλλους γνωστούς αλγόριθμους ταξινόμησης, ωστόσο είναι πολύ πιο απλό από αυτούς. Ένας πιο απλός αλγόριθμος έχει λιγότερες πιθανότητες εμφάνισης σφαλμάτων κατά τη μετατροπή του σε κώδικα. Έτσι, αν πρέπει να προγραμματίσει κανείς έναν αλγόριθμο ταξινόμησης βιαστικά, είναι ασφαλέστερο να χρησιμοποιήσει την επιλογή ταξινόμησης (ή κάποιον άλλον απλό αλγόριθμο).

Αν όμως η αποτελεσματικότητα είναι μείζονος σημασίας, τότε ίσως θα πρέπει να χρησιμοποιήσει έναν πιο σύνθετο και πιο αποτελεσματικό αλγόριθμο. Θα πρέπει να έχουμε υπόψη μας όμως ότι ένας πιο σύνθετος αλγόριθμος θα χρειαστεί περισσότερο χρόνο για τον προγραμματισμό, τον έλεγχο και την αποσφαλμάτωσή του. Σίγουρα τα λάθος αποτελέσματα είναι πάντα ανεπαρκή ανεξάρτητα από το πόσο γρήγορα μπορεί να τα δώσει το πρόγραμμα μας.

3.5.5 Πολυδιάστατοι Πίνακες

Πολλές φορές είναι πολύ χρήσιμο να έχουμε πίνακες με περισσότερες του ενός δείκτες. Όπως και στους απλούς πίνακες ή μονοδιάστατους πίνακες και εδώ η αρίθμηση ξεκινάει από το 0 για κάθε δείκτη. Οι πίνακες που έχουν δύο δείκτες λέγονται **δισδιάστατοι πίνακες (two-dimensional arrays)**. Κατά σύμβαση, θεωρούμε ότι ο πρώτος δείκτης υποδηλώνει την γραμμή και ο δεύτερος δείκτης την στήλη. Γενικότερα, ένας πίνακας που έχει πολλούς δείκτες λέγεται **πολυδιάστατος πίνακας (multi-dimensional array)** (αν για παράδειγμα έχει τρείς δείκτες λέγεται τρισδιάστατος κ.ο.κ). Έστω ότι έχουμε τον παρακάτω πίνακα Πίνακας 3.12:

	0	1	2	3
0	24	13	35	1
1	2	11	20	5

Πίνακας 3.12

Ο συμβολισμός για τον παραπάνω πίνακα στην Java έχει ως εξής:

```
int[][] table = new int[2][4];
```

Και η εντολή που θα μπορούσε να εκχωρήσει την τιμή ,για παράδειγμα, 35 όπως φαίνεται στην παραπάνω εικόνα είναι:

```
table[0][2] = 35;
```

Γενικά, ένα δισδιάστατος πίνακας ή και πολυδιάστατος στην ουσία είναι περισσότεροι του ενός μονοδιάστατοι, μαζί. Έτσι για να ξεχωρίσουμε του πίνακες μεταξύ τους, προσθέτουμε περισσότερους από έναν δείκτες. Ο συνηθέστερος τρόπος προσπέλασης πολυδιάστατων πινάκων γίνεται με τη χρήση πολλαπλών εντολών επανάληψης for, όπως φαίνεται στο παρακάτω παράδειγμα:

```
int[][] table = new int[4][3];
for (int i = 0; i < 4; i++)
{
      for (int j = 0; j < 3; j++)
```

```
        {
                table[i][j] = 1;
        }
}
```

Το παραπάνω τμήμα κώδικα δημιουργεί έναν δισδιάστατο πίνακα τύπου `int` και τον ονομάζει `table`, ο οποίος αποτελείται από 4 γραμμές και 3 στήλες και στην συνέχεια δίνει την τιμή 1 σε κάθε ένα από τα στοιχεία του.

3.6 Κληρονομικότητα

Η **κληρονομικότητα (inheritance)** μας επιτρέπει να ορίσουμε μια πολύ γενική κλάση και αργότερα να ορίσουμε πιο εξειδικευμένες κλάσεις προσθέτοντας απλά κάποια νέα στοιχεία στον παλαιότερο, πιο γενικό ορισμό κλάσης. Αυτό μας γλιτώνει από επιπλέον δουλειά, επειδή η πιο εξειδικευμένη κλάση *κληρονομεί* όλες τις ιδιότητες της γενικής κλάσης και το μόνο που πρέπει να κάνουμε εμείς είναι να προγραμματίσουμε τα νέα χαρακτηριστικά.

3.6.1 Βασικά στοιχεία κληρονομικότητας

Αυτό που συμβαίνει ουσιαστικά με την κληρονομικότητα στην `Java`, είναι το ότι ο προγραμματιστής δημιουργεί μια γενική κλάση που την χρησιμοποιεί ως πρότυπο. Έτσι μπορεί να ορίσει κάποια απορρέουσα κλάση και να κληρονομήσει από την γενική κλάση όλες τις μεταβλητές στιγμιοτύπου. Το παρακάτω παράδειγμα που θα δούμε, περιέχει μια κλάση που λέγεται `Person` . Αυτή η κλάση είναι τόσο απλή, ώστε η μόνη ιδιότητα που δίνει σε έναν άνθρωπο είναι το όνομα:

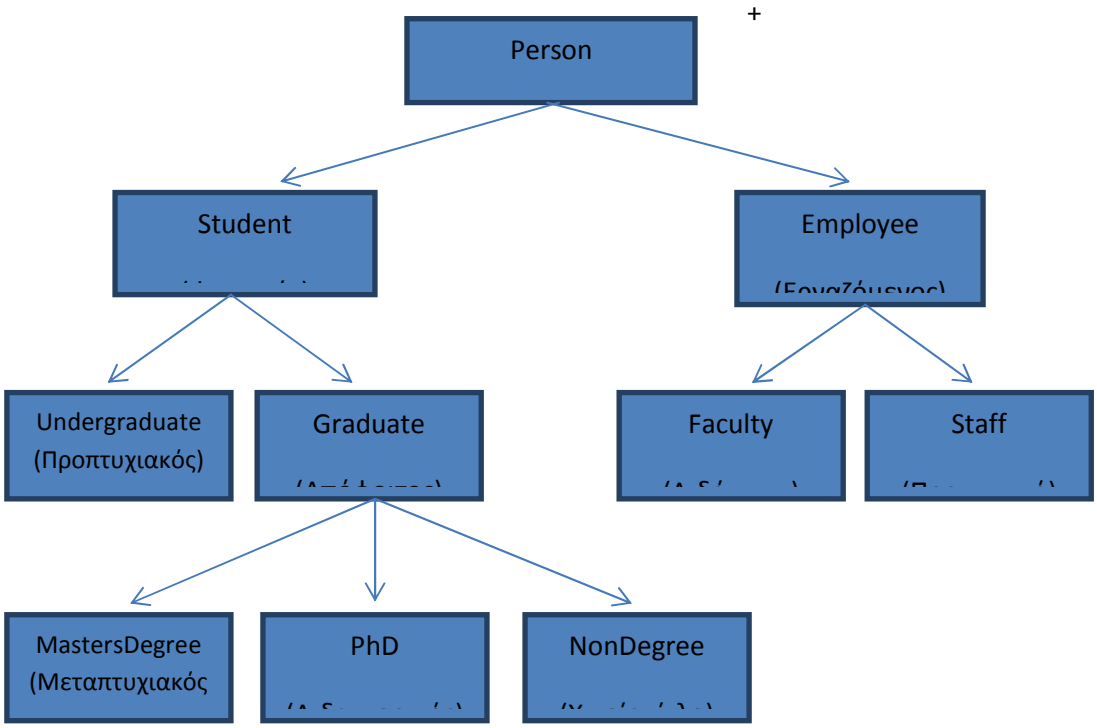
```
public class Person
{
       private String name;
       public Person()
       {
              name = “No name yet.”;
       }
       public Person(String initialName)
       {
              name = initialName;
       }
       public void setName(String newName)
       {
              name = newName;
       }
       public String getName()
       {
              return name;
       }
       public void writeOutput()
       {
              System.out.println(“Name: “ + name);
       }
       public boolean sameName(Person otherPerson)
       {
              return(this.name.equalsIgnoreCase(otherPerson.name));
       }
}
```

Οι πιο πολλές από τις μεθόδους της κλάσης `Person` είναι σαφείς από μόνες τους. Για παράδειγμα, η μέθοδος `sameName` είναι παρόμοια με τη μέθοδο `equals`, αλλά να

σημειωθεί ότι κατά τη σύγκριση ονομάτων θεωρεί ότι ένα γράμμα είτε με κεφαλαίο είτε με μικρό είναι το ίδιο.

3.6.2 Απορρέουσες Κλάσεις

Ας βασιστούμε στην παραπάνω γενική κλάση και ας υποθέσουμε πως θέλουμε να σχεδιάσουμε ένα πρόγραμμα τήρησης φακέλων πανεπιστημίου το οποίο έχει φακέλους για φοιτητές, καθηγητές, διδάσκοντες και προσωπικό. Υπάρχει μια φυσική ιεραρχία για την ομαδοποίηση αυτών των τύπων κλάσεων. Είναι όλοι φάκελοι ανθρώπων. Οι φοιτητές είναι μια υποκλάση (subclass) των ανθρώπων ενώ μία άλλη υποκλάση είναι οι εργαζόμενοι, η οποία περιλαμβάνει και τους διδάσκοντες και το προσωπικό. Οι φοιτητές χωρίζονται σε δύο μικρότερες υποκλάσεις, στους προπτυχιακούς και στους απόφοιτους. Αυτές οι υποκλάσεις μπορούν να διαιρεθούν σε ακόμα μικρότερες υποκλάσεις.



Η παραπάνω εικόνα απεικονίζει ένα τμήμα αυτής της ιεραρχικής διάταξης. Στη Java, μπορούμε να ορίσουμε μία κλάση που να περιλαμβάνει μεταβλητές στιγμιοτύπου για τις ιδιότητες που ανήκουν σε όλες τις υποκλάσεις των ανθρώπων, όπως είναι για παράδειγμα οι φοιτητές, οι διδάσκοντες και το προσωπικό. Ο ορισμός κλάσης μπορεί επίσης να περιέχει όλες τις μεθόδους που διαχειρίζονται τις μεταβλητές στιγμιοτύπου της κλάσης αυτής, όπως συμβαίνει άλλωστε με την κλάση Person που έχουμε ορίσει ήδη στην ενότητα 3.6.1 .

Παρακάτω παρουσιάζεται ο ορισμός μια κλάσης για φοιτητές:

```
public class Student extends Person
{
	private int studentNumber;
	public Student()
```

```
        {
                super();
                studentNumber = 0;   //Υποδηλώνει ότι δεν υπάρχει ακόμα
                κανένας αριθμός.
        }
        public Student(String initialName, int initialStudentNumber)
        {
                super(initialName);
                studentNumber = initialStudentNumber;
        }
        public void reset(String newName, int newStudentNumber)
        {
                setName(newName);
                studentNumber = newStudentNumber;
        }
        public int getStudentNumber()
        {
                return studentNumber;
        }
        public void setStudentNumber(int newStudentNumber)
        {
                studentNumber = newStudentNumber;
        }
        public void writeOutput()
        {
                System.out.println("Name: " + getName());
                System.out.println("Student Number: " + studentNumber);
        }
        public boolean equals(Student otherStudent)
        {
                return (this.sameName(otherStudent)
                 && (this.studentNumber == otherStudent.studentNumber));
        }
}
```

Ένας φοιτητής είναι ένας άνθρωπος και συνεπώς ορίσαμε την κλάση Student ως απορρέουσα κλάση της κλάσης Person. Μία **απορρέουσα κλάση (derived class)** είναι μία κλάση που ορίζεται με την προσθήκη μεταβλητών στιγμιοτύπου και μεθόδων σε μία υπάρχουσα κλάση. Η υπάρχουσα κλάση πάνω στην οποία στηρίζεται η απορρέουσα κλάση λέγεται **βασική κλάση (base class)**. Στο παράδειγμά μας, η βασική κλάση είναι η Person και η απορρέουσα κλάση είναι η Student.

Μία απορρέουσα κλάση ορίζεται με την δεσμευμένη λέξη extends και η σύνταξη της έχει ως εξής, όπως φαίνεται και στον παραπάνω ορισμό κλάσης:

```
public class Student extends Person
```

Όταν ορίζουμε μία απορρέουσα κλάση, δίδουμε μόνο τις επιπλέον μεταβλητές στιγμιοτύπου και τις επιπλέον μεθόδους. Στο παράδειγμα μας, η κλάση Student έχει όλες τις μεταβλητές στιγμιοτύπου και όλες τις μεθόδους της κλάσης Person, τις οποίες όμως δεν αναφέρουμε στον ορισμό της Student. Κάθε αντικείμενο της κλάσης Student έχει μία μεταβλητή στιγμιοτύπου που λέγεται name, αλλά μέσα στον ορισμό της κλάσης Student δεν καθορίζουμε τη μεταβλητή στιγμιοτύπου name. Η κλάση Student και οποιαδήποτε άλλη απορρέουσα κλάση, λέμε ότι **κληρονομεί (inherits)** τις μεταβλητές στιγμιοτύπου και τις μεθόδους της βασικής κλάσης που επεκτείνει (extends).

Έστω ότι δημιουργούμε ένα αντικείμενο της κλάσης Student ως εξής:

```
Student s = new Student();
```

Μετά την εκτέλεση αυτής της εντολής, υπάρχει μία μεταβλητή στιγμιοτύπου s.name . Επειδή η name είναι μία private μεταβλητή στιγμιοτύπου, δεν μπορούμε να γράψουμε s.name (έξω από τον ορισμό της κλάσης Person), όπως η μεταβλητή υπάρχει και μπορεί να προσπελαστεί και να μεταβληθεί. Αυτό μπορεί να πραγματοποιηθεί, χρησιμοποιώντας την κληρονομημένη μέθοδο setName , με τον εξής τρόπο:

```
s.setName("Michalakis Kostas");
```

Όπως ειπώθηκε πιο πάνω στην παρούσα ενότητα, η κλάση Student κληρονομεί τη μέθοδο setName και όλες τις άλλες μεθόδους της βασικής κλάσης Person. Επίσης, μια απορρέουσα κλάση, όπως η Student, μπορεί να προσθέσει κάποιες μεθόδους ή μεταβλητές στιγμιοτύπου σε αυτές που κληρονομεί από την βασική της κλάση. Αυτό φαίνεται παραπάνω στον ορισμό της κλάσης Student με την προσθήκη της μεταβλητής στιγμιοτύπου studentNumber και τις μεθόδους reset, getStudentNumber, setStudentNumber, writeOutput και equals, καθώς και κάποιους δομητές. Στο παρακάτω πρόγραμμα επίδειξης φαίνεται πως το αντικείμενο s της κλάσης Student μπορεί να καλέσει τη μέθοδο setName παρά το γεγονός ότι αυτή είναι μία μέθοδος της βασικής κλάσης Person. Η κλάση Student κληρονομεί τη setName από την κλάση Person.

```
public class InheritanceDemo
{
	public static void main(String[] args)
	{
		Student s = new Student();
		s.setName(Michalakis Kostas");
		s.setStudentNumber(1234);
		s.writeOutput();
	}
}
```

3.6.3 Υπερκάλυψη Ορισμών Μεθόδων

Στον ορισμό της κλάσης Student , προσθέσαμε μια μέθοδο με το όνομα writeOutput η οποία δεν έχει παραμέτρους. Όμως και η κλάση Person έχει μία μέθοδο που λέγεται writeOutput η οποία και αυτή δεν έχει παραμέτρους. Αν η μέθοδος Student κληρονομούσε τη μέθοδο writeOutput από τη βασική κλάση Person, τότε η Student θα περιείχε δύο μεθόδους με το όνομα writeOutput από τις οποίες καμία δε θα είχε παραμέτρους. Η Java έχει έναν κανόνα για την αποφυγή αυτού του προβλήματος. Αν μία απορρέουσα κλάση ορίζει μία μέθοδο που έχει το ίδιο όνομα με μία μέθοδο της βασικής κλάσης και η μέθοδος αυτή *έχει και τον ίδιο αριθμό και τους ίδιους τύπους παραμέτρων με τη μέθοδο της βασικής κλάση*, τότε λέμε ότι ο ορισμός στην απορρέουσα κλάση **υπερισχύει ή υπερκαλύπτει (overrides)** τον ορισμό της βασικής κλάσης. Με άλλα λόγια, με τα αντικείμενα της απορρέουσας κλάσης χρησιμοποιείτε ο ορισμός της μεθόδου που βρίσκεται στην απορρέουσα κλάση.

Για παράδειγμα, στο παραπάνω πρόγραμμα επίδειξης, η κλήση της μεθόδου writeOutput για το αντικείμενο s της κλάσης Student θα χρησιμοποιήσει τον ορισμό της writeOutput που βρίσκεται στην κλάση Student και όχι τον ορισμό που βρίσκεται στην κλάση Person. Όταν υπερκαλύπτουμε μια μέθοδο, μπορούμε να αλλάξουμε το σώμα του ορισμού μεθόδου με οποιονδήποτε τρόπο επιθυμούμε, αλλά δεν μπορούμε να κάνουμε καμία αλλαγή στην επικεφαλίδα της μεθόδου.

3.6.3.1 Ο προσδιορισμός final

Αν θέλουμε να καθορίσουμε ότι ένας ορισμός μεθόδου δεν μπορεί να υπερκαλυφθεί με ένα νέο ορισμό σε μία απορρέουσα κλάση, μπορούμε να το κάνουμε προσθέτοντας τον προσδιορισμό final στην επικεφαλίδα της μεθόδου, όπως στο παρακάτω παράδειγμα:

```
public final void specialMethod()
{
	...
}
```

Αν μια μέθοδος δηλώνεται ότι είναι final, ο μεταγλωττιστής γνωρίζει περισσότερα για το πώς θα χρησιμοποιηθεί, επομένως μπορεί να δημιουργήσει πιο αποτελεσματικό κώδικα για τη μέθοδο.

Επίσης, μπορεί να δηλωθεί ως final ακόμα και μία ολόκληρη κλάση. Σε αυτήν την περίπτωση δε μπορούμε να τη χρησιμοποιήσουμε ως βασική κλάση για τη δημιουργία απορρεουσών κλάσεων.

3.6.4 Υπερκάλυψη έναντι Υπερφόρτωσης

Δεν πρέπει να μπερδεύουμε την υπερκάλυψη μεθόδου με την υπερφόρτωση μεθόδου. Όταν υπερκαλύπτεται ένας ορισμός μεθόδου, ο νέος ορισμός μεθόδου που δίδεται στην απορρέουσα κλάση έχει τον ίδιο ακριβώς αριθμό και τύπο παραμέτρων. Αν, όμως, η μέθοδος στην απορρέουσα κλάση έχει διαφορετικό αριθμό παραμέτρων ή μία παράμετρο διαφορετικού τύπου από τη μέθοδο της βασικής κλάσης, τότε η απορρέουσα κλάση θα έχει και τις δύο μεθόδους, το οποίο σημαίνει ότι το όνομα της μεθόδου έχει υπερφορτωθεί.

3.6.5 Οι Δομητές στις Απορρέουσες Κλάσεις

Μία απορρέουσα κλάση, όπως η κλάση Student που είδαμε πιο πάνω σε αυτήν την ενότητα, έχει τους δικούς της δομητές. Η βασική κλάση από την οποία προέρχεται, όπως η κλάση Person, έχει επίσης τους δικούς της δομητές. Όταν ορίζουμε ένα δομητή για την απορρέουσα κλάση, η πρώτη κίνηση συνήθως είναι να καλέσουμε έναν δομητή από τη βασική κλάση. Έστω ότι ορίζουμε έναν δομητή για την κλάση Student. Ένα από τα πρώτα στοιχεία που πρέπει αν αρχικοποιηθούν είναι το όνομα του φοιτητή. Αυτή η αρχικοποίηση ονόματος κανονικά γίνεται από τους δομητές της βασικής κλάσης Person (αφού η μεταβλητή στιγμιοτύπου name εμφανίζεται στον ορισμό της Person). Για παράδειγμα, έστω ο παρακάτω ορισμός ενός δομητή για την απορρέουσα κλάση Student:

```
public Student(String initialName, int initialStudentNumber)
{
	super(initialName);
	studentNumber = initialStudentNumber;
}
```

Η γραμμή κώδικα:

```
super(initialName);
```

είναι μια κλήση προς έναν δομητή της βασικής κλάσης και στην περίπτωση μας είναι μια κλήση προς έναν δομητή της κλάσης Person. Αξίζει ιδιαίτερης προσοχής, ότι για να καλέσουμε τον δομητή της βασικής κλάσης χρησιμοποιήσαμε την δεσμευμένη λέξη super και όχι μόνο το όνομα του δομητή, δηλαδή δεν χρησιμοποιήσαμε:

```
Person(initialName);  //ΜΗ ΕΓΚΥΡΗ
```

Υπάρχουν κάποια πράγματα που πρέπει να προσέχουμε με την χρήση της super. Πρέπει πάντα να είναι η πρώτη λειτουργία που εκτελείται μέσα σε έναν ορισμό δομητή. Δεν μπορούμε να την χρησιμοποιήσουμε αργότερα μέσα στον ορισμό. Αν δεν συμπεριλάβουμε μια κλήση προς τον δομητή της βασικής κλάσης, τότε η Java θα το κάνει αυτόματα, δηλαδή θα συμπεριλάβει μία κλήση προς τον προεπιλεγμένο δομητή της βασικής κλάσης η οποία θα είναι και η πρώτη λειτουργία οποιουδήποτε δομητή για μία απορρέουσα κλάση.

3.6.6 Κλήση προς μία Υπερκαλυμμένη Μέθοδο

Όταν ορίζουμε έναν δομητή για μία απορρέουσα κλάση, μπορούμε να χρησιμοποιήσουμε την δεσμευμένη λέξη super ως όνομα για το δομητή της βασικής κλάσης. Μπορούμε όμως να χρησιμοποιήσουμε τη super και για να καλέσουμε μία μέθοδο της βασικής κλάσης η οποία είναι υπερκαλυμμένη στην απορρέουσα κλάση, αλλά αυτό γίνεται με ένα λίγο διαφορετικό τρόπο.

Για παράδειγμα, έστω η μέθοδος της απορρέουσας κλάσης Student, του παραδείγματος μας η οποία χρησιμοποιεί την παρακάτω εντολή για την εμφάνιση στην έξοδο του ονόματος της κλάσης Student :

```
System.out.println("Name :" + getName());
```

Εναλλακτικά, θα μπορούσαμε να χρησιμοποιήσουμε την μέθοδο writeOutput της κλάσης Person για την έξοδο του ονόματος, αφού η μέθοδος writeOutput της κλάσης Person θα εμφανίσει το όνομα του ανθρώπου (person). Το μόνο πρόβλημα είναι η ότι η χρήση του ονόματος writeOutput για μία μέθοδο στη κλάση Student θα εννοεί τη μέθοδο writeOutput στην κλάση Student. Αυτό που χρειαζόμαστε είναι ένας τρόπος να πούμε "writeOutput() όπως ορίζεται στη βασική κλάση" , το οποίο μπορεί να γίνει γράφοντας super.writeOutput() . Έτσι, ένας ισοδύναμος ορισμός της μεθόδου writeOutput για την κλάση Student είναι ο εξής:

```
public void writeOutput()
{
      super.writeOutput();
      System.out.println("Student Number: " + studentNumber);
}
```

Αν αντικαταστήσουμε τον ορισμό της μεθόδου writeOutput που βρίσκεται μέσα στον ορισμό της Student που είχαμε ορίσει αρχικά, με τον προηγούμενο, τότε η συμπεριφορά της κλάσης Student θα είναι ακριβώς ίδια με πριν.

3.6.7 Η Κλάση Object

Η Java έχει μία κλάση η οποία είναι κλάση πρόγονος κάθε κλάσης. Στη Java, κάθε κλάση είναι απορρέουσα κλάση μίας απορρέουσας κλάσης (μέχρι κάποιο αριθμό διαδοχικών

απορρεουσών κλάσεων) της κλάσης Object. Έτσι, κάθε αντικείμενο κάθε κλάσης είναι τύπου Object, καθώς επίσης και του τύπου της κλάσης του (και βέβαια των τύπων των κλάσεων προγόνων της). Ακόμα και οι κλάσεις που ορίζουμε εμείς οι ίδιοι είναι κλάσεις απόγονοι της κλάσης Object. Αν δεν κάνουμε την κλάση μας απορρέουσα κλάση κάποια κλάσης, τότε η Java αυτόματα θα την κάνει απορρέουσα κλάση της κλάσης Object.

Η κλάση Object μας επιτρέπει να γράψουμε κώδικα Java για μεθόδους σε μία παράμετρο (τύπου Object) η οποία μπορεί να αντικατασταθεί από ένα αντικείμενο οποιασδήποτε κλάσης. Η Κλάση Object έχει κάποιες μεθόδους τις οποίες κληρονομεί κάθε κλάση της Java. Για παράδειγμα, κάθε αντικείμενο κληρονομεί τις μεθόδους equals και toString από κάποια κλάση πρόγονο, είτε κατευθείαν από την κλάση Object είτε από μία κλάση που έχει κληρονομήσει τις μεθόδους από την κλάση Object. Ωστόσο, οι κληρονομούμενες μέθοδοι equals και toString δεν θα λειτουργήσουν σωστά για καμία σχεδόν κλάση που ορίζουμε. Θα πρέπει να υπερκαλύψουμε τους ορισμούς των μεθόδων που έχουν κληρονομηθεί με νέους και πιο κατάλληλους τρόπους.

Η κληρονομούμενη μέθοδος toString δεν έχει ορίσματα. Η μέθοδος toString υποτίθεται ότι πρέπει να επιστρέφει όλα τα δεδομένα που περιέχει ένα αντικείμενο, κωδικοποιημένα σε ένα αντικείμενο τύπου String. Όμως, η κληρονομούμενη έκδοση της toString είναι σχεδόν πάντα άχρηστη επειδή δε θα δώσει μία προσεκτική μορφοποίηση των δεδομένων. Πρέπει να υπερκαλύψουμε τον ορισμό της toString για να δώσει ένα κατάλληλο αντικείμενο τύπου String για τα δεδομένα που περιέχουν τα αντικείμενα της κλάσης που ορίζουμε. Για παράδειγμα ο παρακάτω ορισμός της toString θα μπορούσε να προστεθεί στην κλάση Student :

```
public String toString()
{
      return(“Name: “ + getName() + “\nStudent number: “ + studentNumber);
}
```

Αν αυτή η μέθοδος toString προστεθεί στην κλάση Student, θα μπορούσε να χρησιμοποιηθεί δίνοντας, μεταξύ άλλων, έξοδο με τον ακόλουθο τρόπο:

```
Student s = new Student(“Joe Student”, 2011);
System.out.println(s.toString());
```

Η έξοδος που θα έδινε θα ήταν:

```
Name: Joe Student

Student number: 2001
```

Η μέθοδος toString έχει μία πολύ σημαντική ιδιότητα. Αν δεν τη συμπεριλάβουμε μέσα σε μία κλήση της System.out.println , τότε καλείται αυτόματα. Άρα, οι δύο παρακάτω κλήσεις είναι ισοδύναμες:

```
System.out.println(s.toString());
System.out.println(s);
```

Αυτές οι δύο εντολές θα δώσουν την ακόλουθη έξοδο:

```
Name: Joe Student
Student number: 2001
Name: Joe Student
```

```
Student number: 2001
```

Για το λόγο αυτό, είναι καλή πρακτική να ορίζουμε πάντοτε μία κατάλληλη μέθοδο toString για τις κλάσεις σας.

Μία άλλη μέθοδος που κληρονομείται από την κλάση Object είναι η μέθοδος clone. Η μέθοδος αυτή δε δέχεται ορίσματα και επιστρέφει ένα αντίγραφο του καλούντος αντικειμένου. Το επιστρεφόμενο αντικείμενο υποτίθεται ότι πρέπει να έχει ακριβώς τα ίδια δεδομένα με αυτά του καλούντος αντικειμένου, αλλά είναι ένα διαφορετικό αντικείμενο (ένας κλώνος). Όπως συμβαίνει και με τις άλλες μεθόδους που έχουν κληρονομηθεί από την κλάση Object, για να λειτουργήσει σωστά η μέθοδος clone θα πρέπει να επανακαθοριστεί (δηλαδή, να υπερκαλυφθεί). Όμως, στην περίπτωση της μεθόδου clone υπάρχουν και άλλα πράγματα που θα πρέπει επίσης να κάνουμε. Μία ενδελεχής ανάλυση της μεθόδου clone είναι πέρα από το σκοπό αυτής της πτυχιακής.

3.6.8 Αφηρημένες Κλάσεις

Αν μία μέθοδος μίας κλάσης οριστεί με τον προσδιοριστή abstract και χωρίς σώμα, τότε η συνάρτηση αυτή είναι **αφηρημένη κλάση (abstract class)** δηλαδή δεν έχει υλοποίηση στη συγκεκριμένη κλάση. Για παράδειγμα:

```
abstract class document
{
	public abstract String identify();
}
```

Αν μία κλάση περιέχει, ή και κληρονομήσει, τουλάχιστον μία αφηρημένη μέθοδο τότε και αυτή πρέπει να οριστεί με τον προσδιοριστή abstract και ονομάζεται αφηρημένη (abstract). Αν μία κλάση είναι αφηρημένη, δεν μπορούμε να έχουμε αντικείμενα αυτής τη κλάσης (εκτός αν πρόκειται για αντικείμενα μία απορρέουσας κλάσης) και μπορεί να χρησιμοποιηθεί μόνο ως βασική κλάση για τη δημιουργία απορρεουσών κλάσεων. Αν μια κλάση είναι αφηρημένη, δεν είναι αφηρημένες και όλες οι μέθοδοι της. Αν θα πρέπει να δώσουμε έναν κανονικό ορισμό για μία μέθοδο, τότε θα πρέπει να το κάνουμε. Μ' αυτόν τον τρόπο, η αφηρημένη κλάση έχει όσο το δυνατόν περισσότερες λεπτομέρειες οι οποίες δε χρειάζεται να επαναλαμβάνονται σε κάθε απορρέουσα κλάση. Αν πρέπει να δοθεί μια απάντηση στην απορία «γιατί πρέπει να έχουμε αφηρημένες κλάσεις;» , τότε η απάντηση είναι «γιατί η χρήση τους απλοποιεί τη σκέψη μας» . Ορίζουμε σαν αφηρημένη μία κλάση γιατί μπορούμε κάποιες μεθόδους να τις ορίσουμε μόνο μία φορά. Για παράδειγμα, όταν έχουμε μία κλάση για τον γραφικό σχεδιασμό αντικειμένων, που θα αναλύσουμε σε επόμενη ενότητα, ενδεχομένως, να θέλουμε να χρησιμοποιήσουμε μία μέθοδο για οποιοδήποτε είδος σχήματος. Αν δεν έχει νόημα να ορίσουμε μία μέθοδο, επειδή πάντα θα υπερκαλύπτεται, τότε κάνουμε τη μέθοδο αφηρημένη (επομένως ορίζουμε και την κλάση αφηρημένη) και απαλλασσόμαστε από την υποχρέωση να γράψουμε έναν άχρηστο ορισμό μεθόδου.

Μία αφηρημένη μέθοδος εξυπηρετεί έναν σκοπό, παρά το γεγονός ότι δεν της δίδεται πλήρης ορισμός. Χρησιμεύει ως υποκατάστατο για μία μέθοδο που πρέπει να οριστεί σε όλες τις (μη αφηρημένες) απορρέουσες κλάσεις. Θα δοθούν παραδείγματα σε επόμενη ενότητα για την καλύτερη κατανόηση των κλάσεων αυτού του είδους.

3.6.9 Διασυνδέσεις

Μία **διασύνδεση (interface)** είναι η ακραία περίπτωση μίας αφηρημένης κλάσης. Στην πραγματικότητα, είναι τόσο ακραία που μία διασύνδεση δεν είναι μία κλάση. Ωστόσο, είναι ένας τύπος που μπορεί να ικανοποιηθεί από οποιανδήποτε κλάση η οποία *υλοποιεί τη διασύνδεση*.

Μία διασύνδεση καθορίζει τις επικεφαλίδες για τις μεθόδους που πρέπει να οριστούν σε οποιαδήποτε κλάση υλοποιεί τη διασύνδεση. Μία διασύνδεση περιέχει μόνο επικεφαλίδες μεθόδων και καθόλου μεταβλητές στιγμιοτύπου ή οποιονδήποτε ολοκληρωμένο ορισμό μεθόδου. Μπορεί, όμως, να περιέχει και ορισμένες σταθερές. Για να υλοποιήσει μια διασύνδεση, μία κλάση πρέπει να πρέπει να κάνει δύο πράγματα:

1. Πρέπει να συμπεριλάβει τη φράση:

```
implements Όνομα_Διασύνδεσης
```

στην αρχή του ορισμού κλάσης. Για να υλοποιήσουμε περισσότερες από μία διασυνδέσεις, παραθέτουμε όλα τα ονόματα διασυνδέσεων διαχωρίζοντάς τα με κόμμα, ως εξής:

```
implements MyInterface, YourInterface
```

2. Η κλάση πρέπει να υλοποιεί όλες τις επικεφαλίδες μεθόδων που αναφέρονται στον ορισμό της διασύνδεσης (ή στους ορισμούς των διασυνδέσεων, αν είναι πολλές).

Ακολουθεί ένα παράδειγμα διασύνδεσης:

```
public interface Writable
{
	public String toString();
	public void writeOutput();
}
```

Και παρακάτω φαίνεται η υλοποίηση της παραπάνω διασύνδεσης:

```
public class WritableUndergraduate extends Student implements Writable
{
	private int level;   //1 για πρωτοετή, 2 δευτεροετή, 3 για τριτοετή
	// ή 4 για τελειόφοιτο
	public WritableUndergraduate()
	{
		super();
		level = 1;
	}
	public WritableUndergraduate(String initialName,
		initialStudentNumber, int initialLevel)
	{
		super (initialName, initialStudentNumber);
		setLevel (initialLevel);  //Ελέγχει αν ισχύει
		// 1 <= initialLevel <= 4
	}
	public String toString()
	{
		return (“Name: “ + getName() + “\nStudent number: “ +
			“\nLevel: “ + getLevel());
	}
	public void writeOutput()
	{
		super.writeOutput();
		System.out.println(“Student Level: “ + level);
```

```
	}
	public boolean equals(WritableUndergraduate other)
	{
		return (super.equals(other) && (this.level == other.level));
	}
	public void reset(String newName, int newStudentNumber, int newLevel)
	{
		reset (newName, newStudentNumber);
		setLevel(newLevel);  //Ελέγχει αν ισχύει 1 <= initialLevel <= 4
	}
	public getLevel()
	{
		return level;
	}
	public void setLevel(int newLevel)
	{
		if ((1 < newLevel) && (newLevel <= 4)) level = newLevel; {
			level = newLevel;
		}
		else {
			System.out.println("Illegal level!");
			System.exit(0);
		}
	}
}
```

Η βασική κλάση είναι η κλάση Student που έχουμε ορίσει σε προηγούμενη ενότητα.

Για να υλοποιήσει, για παράδειγμα, τη διασύνδεση Writable ένας ορισμός κλάσης, πρέπει να περιέχει τη φράση implements Writable στην αρχή του ορισμού μεθόδου, όπως φαίνεται και στην πρώτη γραμμή της παραπάνω υλοποίησης της διασύνδεσης. Η κλάση πρέπει επίσης να υλοποιεί τις δύο μεθόδους writeOutput() και toString() .

Μία διασύνδεση είναι ένας τύπος. Αυτό μας επιτρέπει να γράψουμε μία μέθοδο με μία παράμετρο ενός τύπου διασύνδεσης, όπως για παράδειγμα μία παράμετρο τύπου Writable, και αυτή η μέθοδος θα ισχύει σε οποιαδήποτε κλάση ορίσουμε αργότερα που θα υλοποιεί τη διασύνδεση. Μία διασύνδεση έχει παρόμοια λειτουργία με μία βασική κλάση αλλά είναι σημαντικό να θυμόμαστε ότι δεν είναι μία βασική κλάση (για την ακρίβεια, δεν είναι καν κλάση). Μερικές γλώσσες προγραμματισμού επιτρέπουν σε μία κλάση να είναι απορρέουσα κλάση δύο διαφορετικών βασικών κλάσεων. Αυτό στη Java δεν επιτρέπεται. Στη Java, μία απορρέουσα κλάση μπορεί να έχει μόνο μία βασική κλάση. Εκτός, όμως, από την οποιαδήποτε βασική κλάση που μπορεί να έχει μία κλάση της Java, μπορεί επίσης να υλοποιεί οποιονδήποτε αριθμό διασυνδέσεων. Αυτό επιτρέπει στα προγράμματα της Java να προσεγγίζουν την ισχύ των πολλαπλών βασικών κλάσεων, χωρίς τις επιπλοκές που μπορούν να εμφανιστούν με τις πολλαπλές βασικές κλάσεις. Ένας ορισμός διασύνδεσης αποθηκεύεται σε ένα αρχείο Java και μπορεί να μεταγλωττιστεί όπως μεταγλωττίζεται ένας ορισμός κλάσης.

3.6.10 Δυναμική σύνδεση

Για να εξηγήσουμε την δυναμική σύνδεση ας δούμε τον παρακάτω ορισμό βασικής κλάσης:

```
public class Figure
{
	private int offset;
	public Figure()
```

```
	{
		offset = 0;
	}
	public Figure(int theOffset)
	{
		offset = theOffset;
	}
	public void setOffset(int newOffset)
	{
		offset = newOffset;
	}
	public int getOffset()
	{
		return offset;
	}
	//* Σχεδιάζει το σχήμα σε lineNumber γραμμές κάτω από την τρέχουσα
	   γραμμή.
	*/
	public void drawAt(int lineNumber)
	{
		int count;
		for (count = 0; count < lineNumber; count++)
		{
			System.out.println();
		}
		drawHere();
	}
	/** Σχεδιάζει το σχήμα στην τρέχουμε γραμμή.
	*/
	public void drawHere()
	{
		int count;
		for (count = 0; count < offset; count++)
		{
			System.out.print(‘ ‘);
		}
		System.out.println(‘*’);
	}
```

Αν και όπως έχει ειπωθεί σε προηγούμενη ενότητα ότι η σχεδίαση γραφικών θα αναλυθεί σε επόμενη ενότητα, είναι απαραίτητο να δούμε έστω και σαν παράδειγμα μια τέτοια κλάση ώστε να γίνει καλύτερα κατανοητή η έννοια της δυναμικής σύνδεσης. Παρακάτω είναι η κλάση για τον σχεδιασμό ενός ορθογωνίου. Η παρακάτω κλάση, κληρονομεί τις μεθόδους getOffset, setOffset, drawAt από την κλάση Figure :

```
public class Box extends Figure
{
	private int height;
	private int width;
	public Box()
	{
		super();
		height = 0;
		width = 0;
	}
	public Box(int theOffset, int theHeight, int theWidth)
	{
		super(theOffset);
		height = theHeight;
		width = theWidth;
	}
	public void reset (int newOffset, int newHeight, int newWidth)
	{
		setOffset(newOffset);
		height = newHeight;
		width = newWidth;
```

```
	}
	/** Σχεδιάζει το σχήμα στην τρέχουμε γραμμή.
	*/
	public void drawHere()
	{
		drawHorizontalLine();
		drawSides();
		drawHorizontalLine();
	}
	private void drawHorizontalLine()
	{
		spaces(getOffset());
		int count;
		for (count = 0; count < width; count++)
		{
			System.out.print('-');
		}
		System.out.println();
	}
	private void drawSides()
	{
		int count;
		for (count = 0; count < (height – 2); count++)
		{
			drawOneLineOfSides();
		}
	}
	private void drawOneLineOfSides()
	{
		spaces(getOffset());
		System.out.print('|');
		spaces(width – 2);
		System.out.println('|');
	}
	//Γράφει τον υποδηλούμενο αριθμό διαστημάτων.
	private static void spaces(int number)
	{
		int count;
		for (count = 0; count < number; count++)
		{
			System.out.print(' ');
		}
	}
	/** Η spaces θα μπορούσε να μην είναι static. Η κλάση θα δούλευε
		κανονικά αν δεν ήταν static. Έγινε static, επειδή δεν
		χρειάζεται καλούν αντικείμενο και έτσι είναι πιο σαφές.
	*/
```

Έτσι λοιπόν, αν κοιτάξουμε τον ορισμό της μεθόδου drawAt στη κλάση Figure παραπάνω, θα δούμε ότι περιέχει μία κλήση προς τη μέθοδο drawHere . Αν η Figure ήταν η μοναδική κλάση που είχαμε, αυτό δε θα ήταν τίποτε το ιδιαίτερο. Έχουμε όμως την κλάση Box η οποία είναι απορρέουσα κλάση της κλάσης Figure . Η κλάση Box κληρονομεί αναλλοίωτη τη μέθοδο drawAt από τη βασική της κλάση αλλά υπερκαλύπτει τον ορισμό της μεθόδου drawHere . Μπορεί με την πρώτη ματιά να μη βλέπουμε τίποτα το περίεργο, αλλά τα πράγματα δεν είναι τόσο απλά. Ας δούμε τι κάνει ο μεταγλωττιστής όταν συναντάει τον παρακάτω κώδικα:

```
Box b = new Box(1 ,4, 4);
b.drawAt(2);
```

Η μέθοδος drawAt ορίστηκε στη κλάση Figure , αλλά καλεί την μέθοδο drawHere που ορίστηκε εκ νέου στην κλάση Box . Ο κώδικας για τη drawAt μεταγλωττίστηκε με την κλάση Figure , η οποία μεταγλωττίστηκε πριν ακόμα γραφτεί η κλάση Box και η μέθοδος της

drawHere . Έτσι, ο μεταγλωττισμένος κώδικας για την drawAt χρησιμοποιεί έναν ορισμό της μεθόδου drawHere ο οποίος δεν είχε καν γραφτεί κατά τη μεταγλώττιση της drawAt . Πως μπορεί να συμβαίνει κάτι τέτοιο;

Όταν μεταγλωττίζεται ο κώδικας για την drawAt δεν εισάγεται τίποτα για την κλήση drawHere εκτός από μία επισήμανση που λέει “Use the currently applicable definition of drawHere” (“Χρησιμοποιώ τον τρέχοντα κατάλληλο ορισμό της drawHere”) . Μετά, όταν καλούμε την b.drawAt(2) , ο μεταγλωττισμένος κώδικας φτάνει κάποια στιγμή στην επισήμανση και την αντικαθιστά με μία κλήση της έκδοσης της drawHere που περιέχεται στο αντικείμενο b . Αν το b είναι τύπου Box , τότε η έκδοση της drawHere που χρησιμοποιείται θα είναι αυτή που βρίσκεται μέσα στον ορισμό της κλάσης Box.

Με τον τρόπο αυτό η κλήση μία μεθόδου που μπορεί να υπερκαλυφθεί αργότερα λέγεται **δυναμική** ή **καθυστερημένη σύνδεση (dynamic** or **late binding)** , επειδή η σημασία της κλήσης μεθόδου δεν είναι συσχετισμένη με τη θέση της κλήσης μεθόδου μέχρι να τρέξουμε το πρόγραμμα. Αν η Java δεν χρησιμοποιούσε την δυναμική σύνδεση, τότε όταν θα εκτελούσαμε την κλήση f.drawAt() , θα βλέπαμε πάντα αυτό που θα είχε σχεδιαστεί από τη μέθοδο drawAt της κλάσης Figure (το οποίο στην περίπτωσή μας είναι ένας αστερίσκος) είτε το f ήταν αντικείμενο Box είτε κάποιο άλλο αντικείμενο κάποιας άλλης απορρέουσας κλάσης.

Οι απορρέουσες κλάσεις, όπως η κλάση Box παραπάνω, κληρονομούν κάποια μέθοδο από την βασική κλάση, όπως η μέθοδος drawAt . Η μέθοδος drawAt στο παράδειγμα μας, δεν υπερκαλύπτεται στον ορισμό της κλάσης Box. Άρα, ακόμα και το κείμενο του ορισμού μεθόδου της drawAt είναι το ίδιο για το αντικείμενο Box. Αυτή που υπερκαλύπτεται (άμεσα) στο παράδειγμά μας είναι η μέθοδος drawHere , η οποία καλείται μέσα στον ορισμό της drawAt. Σε αυτή την περίπτωση λέμε ότι το όνομα μεθόδου drawAt **υπερκαλύπτεται έμμεσα (indirectly overridden)** στη κλάση Box.

3.6.11 Πολυμορφισμός

Η λέξη **πολυμορφισμός (polymorphism)** σημαίνει την ύπαρξη πολλών μορφών. Η αρχική σημασία της λέξης (στην επιστήμη των υπολογιστών) ήταν η δυνατότητα χρήσης διαφορετικών ορισμών για το ίδιο όνομα μεθόδου, ανάλογα με το περιβάλλον. Μ’ αυτήν την αρχική σημασία, μερικές λειτουργίες όπως η υπερφόρτωση θεωρούνταν πολυμορφισμός. Όμως, η σύγχρονη χρήση του όρου είναι πιο σαφής. Η τρέχουσα σημασία της λέξης αναφέρεται στο μηχανισμό της δυναμικής σύνδεσης ο οποίος καθορίζει ποια λειτουργία μεθόδου θα χρησιμοποιηθεί για ένα (άμεσα ή έμμεσα) υπερκαλυμμένο όνομα μεθόδου. Άρα, σύμφωνα με την τρέχουσα χρήση του όρου, *πολυμορφισμός* είναι μία άλλη ονομασία της δυναμικής σύνδεσης.

Ουσιαστικά η λειτουργία του πολυμορφισμού και της δυναμικής σύνδεσης είναι ίδια. Διαχωρίζουμε τις έννοιες κατά κάποιο τρόπο, επειδή η δυναμική σύνδεση θεωρούμε ότι είναι μια λειτουργία που εκτελείται από τον υπολογιστή (ή το σύστημα γενικότερα) και ο πολυμορφισμός είναι κάτι που κάνουν τα αντικείμενα.

3.7 Χειρισμός Εξαιρέσεων

Ένας τρόπος για να γράψουμε ένα πρόγραμμα είναι να υποθέσουμε πως δεν θα συμβεί τίποτα ασυνήθιστο ή τίποτα λάθος. Για παράδειγμα, αν το πρόγραμμα λαμβάνει είσοδο από μία λίστα, τότε θα μπορούσαμε να υποθέσουμε ότι η λίστα δεν είναι κενή. Στην Java υπάρχει τρόπος να επιτύχουμε μία προσέγγιση ώστε να αντιμετωπίσουμε τις ιδιαίτερες περιπτώσεις, όπου ενδεχομένως να συμβεί κάτι ασυνήθιστο ή λάθος. Ο χειρισμός εξαιρέσεων μας επιτρέπει να προσθέσουμε κώδικα αφού γράψουμε τον κώδικα του προγράμματος μας, χωρίς να περιμένουμε να συμβεί κάτι ασυνήθιστο. Έτσι, μπορούμε να διαιρέσουμε το πρόγραμμα μας ή μία μέθοδο σε ξεχωριστά τμήματα, δηλαδή, ένα τμήμα για την κανονική περίπτωση και ένα τμήμα για την ειδική περίπτωση.

3.7.1 Βασικός χειρισμός εξαιρέσεων

Στην Java, ο **χειρισμός εξαιρέσεων (exception handling)** λειτουργεί ως εξής: Είτε η ίδια η γλώσσα Java είτε ο κώδικας μας περιέχει ένα μηχανισμό που ειδοποιεί όταν συμβαίνει κάτι ασυνήθιστο. Αυτή η διαδικασία ειδοποίησης λέγεται **μεταβίβαση μίας εξαίρεσης (throwing an exception)**. Σε ένα άλλο σημείο του προγράμματός μας, σε μία ξεχωριστή κλάση ή σε μία μέθοδο ή απλά σε ένα άλλο τμήμα του κώδικα του προγράμματός μας, τοποθετούμε τον κώδικα που αντιμετωπίζει αυτήν την ειδική περίπτωση. Αυτός ο κώδικας εκτελεί την εργασία που λέγεται **χειρισμός της εξαίρεσης (handling the exception)**. Βέβαια, υπάρχουν πολλά πράγματα που χρειάζεται να εξηγήσουμε προκειμένου να μπορεί να γίνει κατανοητή η διαχείριση εξαιρέσεων στην Java.

Ο βασικός τρόπος χειρισμού εξαιρέσεων στην Java αποτελείται από το τρίπτυχο try – throw – catch . Η σύνταξη για ένα **μπλοκ** try είναι:

```
try
{
        Κώδικας_Προς_Εκτέλεση
}
```

Αυτό το μπλοκ try περιέχει τον κώδικα για το βασικό αλγόριθμο ο οποίος λέει στον υπολογιστή τι να κάνει όταν όλα κυλούν ομαλά. Λέγεται μπλοκ try επειδή δεν είμαστε 100% σίγουροι ότι όλα θα κυλήσουν ομαλά, αλλά θέλουμε να κάνουμε μία προσπάθεια.

Αν κάτι συμβαίνει λάθος, θέλουμε να μεταβιβάσουμε μία εξαίρεση (throw an exception), ο οποίος είναι ένα τρόπος για να υποδηλώσουμε ότι κάτι πήγε λάθος. Έτσι, το γενικό πλάνο όταν προσθέτουμε μία εντολή throw έχει ως εξής:

```
try
{
        Κώδικας_Προς_Εκτέλεση
        Πιθανή_Μεταβίβαση_Μίας_Εξαίρεσης
        Επιπλέον_Κώδικας
}
```

Υπάρχει μία προκαθορισμένη κλάση που λέγεται Exception. Η **εντολή** throw δημιουργεί ένα αντικείμενο της κλάσης Exception και **μεταβιβάζει (throws)**. Όταν μεταβιβάζεται μία

εξαίρεση, διακόπτεται η εκτέλεση του κώδικα που βρίσκεται μέσα στο **μπλοκ** catch. Η εκτέλεση του μπλοκ catch λέγεται **σύλληψη της εξαίρεσης (catching the exception)**. Όταν μεταβιβάζεται μία εξαίρεση, θα πρέπει τελικά να συλληφθεί από κάποιο μπλοκ catch. Παρακάτω φαίνεται ένα παράδειγμα του τρίπτυχου try – throw – catch :

```
try
{
	System.out.println(“Enter number of donuts:”);
	donutCount = keyboard.nextInt();
	System.out.println(“Enter number of glasses of milk:”);
	milkCount = keyboard.nextInt();
	if (milkCount < 1)
	{
		throw new Exception(“Exception: No Milk!”);
	}
	donutsPerGlass = donutCount/(double)milkCount;
	System.out.println(donutCount + “ donuts.”);
	System.out.println(milkCount + “ glasses of milk.”);
	System.out.println(“You have “ + donutsPerGlass + “ donuts for
			each glass of milk.”);
}

catch(Exception e)
{
	System.out.println(e.getMessage());
	System.out.println(“Go buy some milk.”);
}
```

3.7.2 Προκαθορισμένες Κλάσεις Εξαιρέσεων

Στη βιβλιοθήκη της γλώσσας προγραμματισμού Java, υπάρχει κάποιες προκαθορισμένες μέθοδοι και κλάσεις εξαιρέσεων. Συγκεκριμένα, αυτές οι μέθοδοι εξαιρέσεων, μπορούν να μεταβιβάσουν συγκεκριμένο τύπο εξαιρέσεων. Αν χρησιμοποιήσουμε μία από αυτές τις μεθόδους, μπορούμε να βάλουμε ένα μπλοκ try και αμέσως μετά ένα μπλοκ catch για να συλλάβει την εξαίρεση. Τα ονόματα των προκαθορισμένων εξαιρέσεων επιλέγονται έτσι ώστε να προσδιορίζουν και την αιτία της εξαίρεσης. Μερικά παραδείγματα προκαθορισμένων εξαιρέσεων είναι τα:

IOException

ClassNotFoundException

FileNotFoundException

Όταν συλλαμβάνουμε μία εξαίρεση εξ αυτών των προκαθορισμένων κλάσεων εξαιρέσεων, το αλφαριθμητικό που επιστρέφεται από τη μέθοδο getMessage που μας δίνει συνήθως αρκετές πληροφορίες για να προσδιορίσουμε την πηγή της εξαίρεσης.

Η προκαθορισμένη κλάση εξαίρεσης Exception είναι η βάση όλων των εξαιρέσεων. Κάθε κλάση εξαίρεσης είναι απόγονος της κλάσης Exception (δηλαδή, προέρχεται από την κλάση Exception ή προέρχεται από μία μακριά αλυσίδα απορρεουσών κλάσεων που έχει ξεκινήσει από την κλάση Exception). Μπορούμε να χρησιμοποιήσουμε την ίδια κλάση Exception αλλά το πιο πιθανό είναι ότι θα τη χρησιμοποιήσουμε για να ορίσουμε απορρέουσες κλάσεις.

3.7.3 Κλάσεις Εξαιρέσεων Ορισμένες από τον Προγραμματιστή

Μπορούμε να ορίσουμε τις δικές μας κλάσης εξαιρέσεων αλλά κάθε τέτοια κλάση πρέπει να είναι μία απορρέουσα κλάση από μία ήδη υπάρχουσα κλάση εξαίρεσης (είτε μία προκαθορισμένη κλάση είτε μία επιτυχώς ορισμένη από εμάς κλάση). Παρακάτω παρουσιάζονται κάποιοι γενικοί κανόνες:

- Αν δεν έχουμε κανένα περιορισμό να χρησιμοποιήσουμε οποιαδήποτε άλλη κλάση ως βασική κλάση, τότε χρησιμοποιούμαι ως βασική την κλάση Exception.
- Θα πρέπει να ορίσουμε τουλάχιστον δύο δομητές, όπως περιγράφεται παρακάτω στη λίστα αυτή. Η δική μας κλάση εξαίρεσης κληρονομεί τη μέθοδο getMessage. Κανονικά, δε χρειάζεται να προσθέσουμε άλλες μεθόδους, αλλά αν θέλουμε μπορούμε να το κάνουμε.
- Κάθε ορισμός δομητή θα πρέπει να αρχίζει με μία κλήση προς τον δομητή της βασικής κλάσης, όπως παρακάτω:

```
super("Trial Wave Exception thrown!");
```

- Θα πρέπει να συμπεριλάβουμε έναν προεπιλεγμένο δομητή, στην περίπτωση αυτή η κλήση προς τη super θα πρέπει να έχει ένα όρισμα αλφαριθμητικού που θα υποδηλώνει τι είδους εξαίρεση είναι. Αυτό το αλφαριθμητικό μπορεί στη συνέχεια να ανακτηθεί με μία κλήση προς τη getMessage.
- Θα πρέπει επίσης να συμπεριλάβουμε ένα δομητή που να δέχεται ένα μόνο όρισμα αλφαριθμητικού. Στην περίπτωση αυτή, το αλφαριθμητικό θα είναι ένα όρισμα στην κλήση προς την super. Με τον τρόπο αυτό, το αλφαριθμητικό μπορεί να ανακτηθεί με μία κλήση προς τη getMessage.

Ακολουθεί ένα παράδειγμα κώδικα:

```
public class TidalWaveException extends Exception
{
	public TidalWaveException()
	{
		super("Tidal Wave Exception thrown!");
		//Η super είναι μια κλήση προς το δομητή της βασικής
		// κλάσης Exception.
	}
	public TidalWaveException(String message)
	{
		super(message);
	}
}
```

3.8 Ροές & Είσοδος/Έξοδος Αρχείων

Ο όρος **Είσοδος/Έξοδο** ή **Ε/Ε** ή **I/O (Input/Output)** αναφέρεται στην είσοδο και στην έξοδο των προγραμμάτων. Η είσοδος μπορεί να ληφθεί από το πληκτρολόγιο ή από ένα αρχείο, ενώ η έξοδος μπορεί να σταλεί στην οθόνη ή σε ένα αρχείο. Στην ενότητα αυτή θα γίνει μια γενική εισαγωγή στην Είσοδο/Έξοδο (Ε/Ε) αρχείων.

3.8.1 Η Έννοια της Ροής

Η χρήση των αρχείων δε μας είναι άγνωστη στη Java, αφού χρησιμοποιούμε αρχεία για να αποθηκεύσουμε τις κλάσεις και τα προγράμματα Java που φτιάχνουμε. Μπορούμε όμως να χρησιμοποιήσουμε αρχεία και να για να αποθηκεύσουμε δεδομένα εισόδου για ένα πρόγραμμα ή για να διατηρήσουμε δεδομένα εξόδου από ένα πρόγραμμα. Στη Java, ο χειρισμός της λειτουργίας Εισόδου/Εξόδου σε αρχεία, καθώς επίσης και η απλή Ε/Ε από το πληκτρολόγιο και προς την οθόνη, γίνεται από **ροές (streams)**. Μία ροή είναι ένα αντικείμενο το οποίο είτε παραδίδει δεδομένα στον προορισμό τους, όπως για παράδειγμα σε ένα αρχείο ή στην οθόνη, είτε δέχεται δεδομένα από μία πηγή, όπως για παράδειγμα από ένα αρχείο ή από το πληκτρολόγιο και τα παραδίδει στο πρόγραμμά μας. Το αντικείμενο System.out είναι μέχρι τώρα η μοναδική ροή εξόδου που έχουμε χρησιμοποιήσει ενώ κάποια αντικείμενα της κλάσης Scanner, τα οποία έχουμε χρησιμοποιήσει για είσοδο από το πληκτρολόγιο, είναι ροές εισόδου.

3.8.2 Είσοδος/Έξοδος αρχείων κειμένου

Σε αυτό το σημείο θα δοθεί μια περιγραφή των πιο συνηθισμένων τρόπων για την εκτέλεση της λειτουργίας Εισόδου/Εξόδου αρχείων κειμένου στη Java.

3.8.2.1 Έξοδος Αρχείων Κειμένου με την Κλάση PrintWriter

Όταν γράφουμε ένα πρόγραμμα που στέλνει την έξοδο του σε ένα αρχείο κειμένου, χρησιμοποιούμε τη μέθοδο println. Η μέθοδος αυτή έχει την ίδια συμπεριφορά με την System.out.println, αλλά είναι μία μέθοδος της κλάσης PrintWriter. Η κλάση PrintWriter είναι η προτιμώμενη κλάση ροών για την εγγραφή σε ένα αρχείο κειμένου. Παρακάτω παρουσιάζεται ένα απλό πρόγραμμα που γράφει δεδομένα σε ένα αρχείο κειμένου:

```
import java.io.*;
import java.util.*;
public class TextFileOutputDemo
{
	public static void main(String[] args)
	{
		PrintWriter outputStream = null;
		try
		{
			outputStream = new PrintWriter
			(new FileOutputStream (“out.txt”));
		}
		catch(FileNotFoundException e)
		{
			System.out.println(“Error opening the file out.txt.”);
			System.exit(0);
		}
		System.out.println(“Enter three lines of text:”);
		String line = null;
		Scanner keyboard = new Scanner(System.in);
		int count;
		for (count = 1; count <= 3; count++)
		{
			line = keyboard.nextLine();
			outputStream.println(count + “ “ + line);
		}
		outputStream.close();
```

```
            System.out.println("Those lines were written to out.txt.);
        }
}
```

Ας δούμε τι έξοδο θα δώσει ο παραπάνω κώδικας και έπειτα θα τον σχολιάσουμε:

```
Enter three lines of text:
A tall three
in a short forest is like
a big fish in a small pond.
Those lines were written to out.txt.
```

Το παραπάνω πρόγραμμα, ξεκινάει με την εντολή import java.io.*; και αυτό γιατί οι ορισμοί της κλάσης PrintWriter περιέχονται στην βιβλιοθήκη java.io . Το παραπάνω πρόγραμμα, λοιπόν, δημιουργεί ένα αρχείο κειμένου που λέγεται out.txt , το οποίο μπορεί να διαβάσει κάποιος χρησιμοποιώντας έναν κειμενογράφο ή να το διαβάσει κάποιο άλλο πρόγραμμα Java χρησιμοποιώντας την κλάση BufferedReader , για την οποία θα μιλήσουμε παρακάτω. Το άνοιγμα του αρχείου γίνεται με την εξής εντολή:

```
outputStream = new PrintWriter(new FileOutputStream("out.txt"));
```

Η παραπάνω εντολή συνδέει τη ροή που λέγεται OutputStream με το αρχείο out.txt . Η δημιουργία αυτής της σύνδεσης λέγεται **άνοιγμα του αρχείου (opening the file)**. Όταν συνδέουμε ένα αρχείο με μία ροή κατ' αυτόν τον τρόπο, το πρόγραμμα μας ξεκινά πάντα με ένα άδειο αρχείο. Αν το αρχείο out.txt υπάρχει ήδη, τότε τα παλιά περιεχόμενα του θα χαθούν. Αν, όμως, το αρχείο out.txt δεν υπάρχει, τότε θα δημιουργηθεί ένα νέο, άδειο αρχείο με αυτό το όνομα.

Γιατί μία εξαίρεση FileNotFoundException; Είπαμε νωρίτερα, ότι όταν ανοίγουμε ένα αρχείο κειμένου για να γράψουμε την έξοδο στο αρχείο αυτό, ο δομητής μπορεί να μεταβιβάσει μία FileNotFoundException. Όμως, στην περίπτωση που θέλουμε να δημιουργήσουμε ένα νέο αρχείο για έξοδο, γιατί θα πρέπει να μας ενδιαφέρει ότι το αρχείο δεν έχει βρεθεί; Η απάντηση είναι ότι το όνομα της εξαίρεσης (προσοχή, μόνο το όνομα, όχι και η λειτουργία της εξαίρεσης) δεν καλύπτει αυτή την περίπτωση. Η εξαίρεση FileNotFoundException μεταβιβάζεται όταν είναι αδύνατη η δημιουργία του αρχείου, για παράδειγμα όταν το όνομα του αρχείου χρησιμοποιείται ήδη για ένα όνομα φακέλου (καταλόγου).

Αξίζει να προσέξουμε ότι, όμως άλλωστε φαίνεται και στον παραπάνω κώδικα, η μέθοδος println της κλάσης PrintWriter λειτουργεί για την εγγραφή σε ένα αρχείο με τον ίδιο τρόπο που λειτουργεί η μέθοδος System.out.println για την εμφάνιση στην οθόνη. Η κλάση PrintWriter έχει επίσης και τη μέθοδο print, η οποία συμπεριφέρεται όπως ακριβώς και η μέθοδος System.out.print, με τη διαφορά ότι η έξοδος πηγαίνει σε ένα αρχείο κειμένου.

Όταν το πρόγραμμα μας τελειώσει με την εγγραφή σε ένα αρχείο (ή με την ανάγνωση από ένα αρχείο), θα πρέπει να κλείσει τη ροή που συνδέεται με το αρχείο αυτό καλώντας τη μέθοδο close. Η σύνταξη της εντολής που πραγματοποιεί αυτή τη διαδικασία φαίνεται παρακάτω:

```
Όνομα_Ροής.close();
```

Στο παραπάνω πρόγραμμα, η εντολή που κλείνει τη ροή είναι:

```
outputStream.close();
```

Γιατί πρέπει να κλείσουμε ένα αρχείο; Αν το πρόγραμμα μας τελειώσει κανονικά, χωρίς όμως να κλείσει ένα αρχείο, τότε το σύστημα θα το κλείσει αυτόματα για εμάς. Οι λόγοι που πρέπει να κλείνουμε εμείς με μια ρητή εντολή προς τη μέθοδο close τα αρχεία είναι τουλάχιστον δύο. Ο πρώτος είναι ότι αν το πρόγραμμα μας τερματιστεί βίαια, τότε η Java ίσως να μην μπορεί να κλείσει το αρχείο για εμάς, αφήνοντας το αρχείο ανοικτό και με κανένα πρόγραμμα συνδεμένο μαζί του. Αυτό θα μπορούσε να καταστρέψει το αρχείο και γι' αυτό όσο πιο σύντομα κλείνουμε ένα αρχείο τόσο μικρότερη η πιθανότητα να συμβεί κάτι τέτοιο. Ο δεύτερος λόγος, είναι ότι αν το πρόγραμμα μας γράφει σε ένα αρχείο και αργότερα διαβάζει από το ίδιο αυτό αρχείο, τότε πρέπει να κλείσει το αρχείο αφού τελειώσει την εγγραφή σε αυτό και μετά να το ανοίξει ξανά για να το διαβάσει (η Java έχει κλάση που επιτρέπει το άνοιγμα ενός αρχείου και για εγγραφή και για ανάγνωση, αλλά δεν θα παρουσιαστεί η κλάση αυτή στο παρόν κείμενο).

Υπάρχει η δυνατότητα για την περίπτωση που υπάρχει το όνομα του αρχείου ήδη, η Java να μην σβήσει τα περιεχόμενα αυτού, αλλά να προσαρτήσει τα δεδομένα που εγγράφει στα ήδη υπάρχοντα. Αυτό γίνεται τροποποίηση της παρακάτω εντολής του κώδικα:

```
outputStream = new PrintWriter(new FileOutputStream("out.txt"));
```

με την παρακάτω εντολή:

```
outputStream = new PrintWriter(new FileOutputStream("out.txt", true));
```

σε αυτήν την περίπτωση, τώρα, αν δεν υπάρχει το αρχείο με το όνομα out.txt θα δημιουργήσει ένα νέο -κενό- αρχείο, αλλά στην περίπτωση που υπάρχει ήδη το αρχείο με αυτό το όνομα, θα προσαρτήσει τις νέες εγγραφές στο αρχείο αυτό και δεν θα διαγράψει τις υπάρχουσες.

3.8.2.2 Είσοδος Αρχείων με την Κλάση BufferedReader

Η κλάση BufferedReader είναι η προτιμώμενη κλάση ροών για την ανάγνωση από ένα αρχείο κειμένου. Όπως συμβαίνει και στην κλάση PrintWriter, η κλάση BufferedReader δεν έχει δομητή που να δέχεται ως όρισμα ένα όνομα αρχείου και γι' αυτό χρειαζόμαστε μία άλλη κλάση, στη συγκεκριμένη περίπτωση την κλάση FileReader, για να ανοίξουμε ένα αρχείο. Η κλάση FileReader θα δεχτεί ένα όνομα αρχείου ως όρισμα δομητή και θα δημιουργήσει μία ροή που θα είναι ένα αντικείμενο Reader, το οποίο μπορεί να δεχτεί ως όρισμα ο δομητής της κλάσης BufferedReader. Η κλάση Reader είναι μία αφηρημένη κλάση η οποία περιλαμβάνει όλες τις ροές που περιέχουν στο όνομά τους τη λέξη "Reader".

Η έκφραση που ακολουθεί είναι μία πιο προσεκτική μορφή όλων των παραπάνω για τη σύνδεση ενός αρχείου κειμένου με μία ροή της κλάσης BufferedReader ώστε να μπορεί το πρόγραμμά μας να διαβάζει από ένα αρχείο:

```
BufferedReader Όνομα_Ροής = new BufferedReader(
             new FileReader(Όνομα_Αρχείου));
```

Ακολουθεί ένα απλό πρόγραμμα για καλύτερη κατανόηση της λειτουργία της κλάσης BufferedReader, το οποίο διαβάζει δεδομένα από ένα αρχείο και τα εμφανίζει στην οθόνη:

```
import java.io.*;
public class TextFileInputDemo
{
	public static void main(String[] args)
	{
		try
		{
			BufferedReader inputStream = new BufferedReader(
				new  FileReader("data.txt"));
			String line = null;
			line = inputStream.readLine();
			System.out.println("The first line in data.txt is);
			System.out.println(line);
			line = inputStream.readLine();
			System.out.println("The second line in data.txt is);
			System.out.println(line);

			inputStream.close();
		}
		catch(FileNotFoundException e)
		{
			System.out.println("File data.txt was not found");
			System.out.println("or could not be opened.");
		}
		catch(IOException e)
		{
			System.out.println("Error reading from file data.txt.");
		}
	}
}
```

Αν το περιεχόμενο του αρχείου data.txt είναι:

```
1 2

buckle my shoe

3 4

shut the door
```

τότε η έξοδος που θα έδινε αυτό το πρόγραμμα θα ήταν:

```
The first line in data.txt is:

1 2

The second line in data.txt is:

buckle my shoe
```

Το αρχείο data.txt είναι ένα αρχείο που θα μπορούσε να είχε γραφτεί από ένα άνθρωπο ή από ένα πρόγραμμα Java με χρήση της κλάσης PrintWriter. Θα πρέπει να προσέξουμε ότι το πρόγραμμα συλλαμβάνει δύο είδη εξαιρέσεων: μία FileNotFoundException και μία IOException. Μία προσπάθεια ανοίγματος του αρχείου μπορεί να μεταβιβάσει μία FileNotFoundException και οποιαδήποτε από τις κλάσεις inputStream.readLine() μπορεί να μεταβιβάσει μία IOException. Επειδή η FileNotFoundException είναι ένα είδος εξαίρεσης IOException, θα μπορούσαμε να χρησιμοποιήσουμε μόνο το μπλοκ catch για την

IOException. Στην περίπτωση αυτή, όμως, θα αποκτήσουμε λιγότερες πληροφορίες για την εξαίρεση που μεταβιβάζεται. Αν χρησιμοποιήσουμε μόνο ένα μπλοκ catch και μεταβιβαστεί μία εξαίρεση, τότε δε θα ξέρουμε αν το πρόβλημα προκλήθηκε κατά το άνοιγμα του αρχείου ή κατά την ανάγνωση του αρχείου μετά το άνοιγμά του.

3.8.3 Η κλάση File

Σε αυτήν την ενότητα θα παρουσιάσουμε τη κλάση File, η οποία δεν είναι μία κλάση ροών Εισόδου/Εξόδου αλλά είναι πολύ χρήσιμη κατά την εκτέλεση λειτουργιών Ε/Ε σε αρχεία.

Η κλάση File είναι σαν μία περικλείουσα κλάση για ονόματα αρχείων. Ο δομητής της κλάσης File δέχεται ως όρισμα ένα αλφαριθμητικό και δημιουργεί ένα αντικείμενο το οποίο μπορεί να θεωρηθεί ότι είναι το αρχείο μ' αυτό το όνομα. Μπορείτε να χρησιμοποιήσουμε το αντικείμενο File και τις μεθόδους της κλάσης File για να απαντήσουμε σε ερωτήσεις όπως οι ακόλουθες: Υπάρχει αυτό το αρχείο; Έχει το πρόγραμμα μας δικαίωμα να διαβάσει αυτό το αρχείο; Έχει το πρόγραμμά μας δικαίωμα να γράψει σε αυτό το αρχείο; Παρακάτω φαίνεται ένα παράδειγμα, που περιγράφει κάποιες από τις μεθόδους της κλάσης File:

```
File fileObject = new File("Stuff.txt");
if ( ! fileObject.exist())
{
      System.out.println("There is no file named stuff.txt.");
}
else if ( ! fileObject.canRead())
{
      System.out.println("File stuff.txt is not readable.");
}
```

3.8.4 Βασική Είσοδος/Έξοδος δυαδικών αρχείων

Τα δυαδικά αρχεία αποθηκεύουν δεδομένα με την ίδια μορφή με αυτή που χρησιμοποιείται στην κύρια μνήμη του υπολογιστή. Κάθε τμήμα δεδομένων, όμως για παράδειγμα ένας ακέραιος, αποθηκεύεται σαν μια σειρά από bytes. Ένα πρόγραμμα Java διαβάζει αυτά τα bytes με το ίδιο σχεδόν τρόπο που διαβάζει και ένα τμήμα δεδομένων από την κύρια μνήμη του υπολογιστή. Αυτό το γεγονός, εξάλλου, είναι και ο λόγος που τα δυαδικά αρχεία μπορούν να είναι τόσο αποτελεσματικά στη διαχείριση τους.

Τα δυαδικά αρχεία που δημιουργούνται από ένα πρόγραμμα Java μπορούν να μετακινηθούν από έναν υπολογιστή σε έναν άλλο και να εξακολουθούν να μπορούν να διαβάζονται από ένα πρόγραμμα Java, αλλά μόνο ένα πρόγραμμα Java. Συνήθως δεν μπορούν να διαβαστούν από έναν κειμενογράφο ούτε από ένα πρόγραμμα που έχει γραφτεί σε κάποια άλλη γλώσσα προγραμματισμού.

Οι προτιμώμενες κλάσεις ροών για την επεξεργασία δυαδικών αρχείων είναι οι ObjectInputStream και ObjectOutputStream, οι οποίες έχουν μεθόδους για ανάγνωση ή εγγραφή δεδομένων κατά 1 byte τη φορά. Οι ροές αυτές μπορούν αυτόματα να μετατρέψουν αριθμούς και χαρακτήρες σε bytes, τα οποία μπορούν να αποθηκεύουν σε ένα δυαδικό αρχείο. Επίσης, δίνουν τη δυνατότητα σε ένα πρόγραμμα να γραφτεί έτσι ώστε τα δεδομένα που γράφονται σε ένα αρχείο, ή που διαβάζονται από ένα αρχείο, να μην είναι

απλώς bytes, αλλά αλφαριθμητικά ή καταχωρήσεις οποιουδήποτε από τους βασικούς τύπους δεδομένων της Java, όπως για παράδειγμα int, char, double ή ακόμα και αντικείμενα των κλάσεων που ορίζει ο προγραμματιστής.

3.8.4.1 Σύνδεση ενός Δυαδικού Αρχείου με μία Ροή για Εγγραφή

Μπορούμε να δημιουργήσουμε μία ροή της κλάσης ObjectOutputStream και να τη συνδέσουμε με ένα δυαδικό αρχείο ως εξής:

```
ObjectOutputStream Όνομα_Ροής_Εξόδου = new ObjectOutputStream
      (new FileOutputStream(Όνομα_Αρχείου));
```

Ο δομητής FileOutputStream μπορεί να μεταβιβάσει μία FileNotFoundException, η οποία είναι ένα είδος εξαίρεσης IOException. Αν η κλήση του δομητή FileOutputStream είναι επιτυχής, τότε ο δομητής ObjectOutputStream μπορεί να μεταβιβάσει μία διαφορετική εξαίρεση IOException. Ένα μόνο μπλοκ catch για εξαιρέσεις IOException καλύπτει όλες τις περιπτώσεις. Ακολουθεί ένα παράδειγμα:

```
ObjectOutputStream myOutputStream = new ObjectOutputStream
      (new FileOutputStream("myfile.dat"));
```

Μετά το άνοιγμα του αρχείου, μπορούμε να χρησιμοποιήσουμε τις μεθόδους της κλάσης ObjectOutputStream για να γράψουμε σε αρχείο.

3.8.4.2 Σύνδεση ενός Δυαδικού Αρχείου με μία Ροή για Ανάγνωση

Μπορούμε να δημιουργήσουμε μία ροή της κλάσης ObjectInputStream και να τη συνδέσουμε με ένα δυαδικό αρχείο ως εξής:

```
ObjectInputStream Όνομα_Ροής_Εισόδου = new ObjectInputStream
      (new FileInputStream(Όνομα_Αρχείου));
```

Ο δομητής FileInputStream μπορεί να μεταβιβάσει μία FileNotFoundException, η οποία είναι ένα είδος εξαίρεσης IOException. Αν η κλήση του δομητή FileInputStream είναι επιτυχής, τότε η δομητής ObjectInputStream μπορεί να μεταβιβάσει μία διαφορετική εξαίρεση IOException. Ακολουθεί ένα παράδειγμα:

```
ObjectInputStream myInputStream = new ObjectInputStream
      (new FileInputStream ("myfile.dat"));
```

3.8.5 Η κλάση EOFException

Όλες οι μέθοδοι που έχουμε εξετάσει μέχρι τώρα για την ανάγνωση από ένα δυαδικό αρχείο, θα μεταβιβάσουν μία εξαίρεση EOFException όταν επιχειρήσουν να διαβάσουν μετά από το τέλος ενός αρχείου. Πιο συγκεκριμένα, όλες οι μέθοδοι της κλάσης ObjectInputStream, μεταβιβάζουν μία εξαίρεση EOFException αν προσπαθήσουν να διαβάσουν μετά το τέλος ενός αρχείου.

Η κλάση EOFException μπορεί να χρησιμοποιηθεί για να βρούμε το τέλος ενός αρχείου όταν χρησιμοποιούμαι την κλάση ObjectInputStream. Στο παρακάτω παράδειγμα που θα δούμε, η λειτουργία της ανάγνωσης τοποθετείται μέσα σε έναν "ατέρμονα βρόχο" στον οποίο ως

λογική έκφραση χρησιμοποιείται η true. Ο βρόχος δεν είναι πραγματικά ατέρμων επειδή όταν η ανάγνωση φτάνει στο τέλος του αρχείου μεταβιβάζεται μία εξαίρεση, οπότε τερματίζεται ολόκληρο το μπλοκ try και ο έλεγχος περνάει στο μπλοκ catch. Ας δούμε το παράδειγμα και έπειτα θα τονίσουμε κάποια πράγματα που αξίζουν προσοχής:

```
import java.io.*;
public class EOFException
{
	public static void main(String[] args)
	{
		try
		{
			ObjectInputStream inputStream = new ObjectInputStream
				(new FileInputStream(“numbers.dat”));
			int n;
			System.out.println(“Reading ALL the integers”);
			System.out.println(“in the file numbers.dat.”);
		try
		{
			while (true)
			{
				n = inputStream.readInt();
				System.out.println(n);
			}
		}
		catch(EOFException e)
		{
			System.out.println(“End of reading from file.”);
		}
		inputStream.close();
		catch(FileNotFoundException e)
		{
			System.out.println(“Cannot find file number.dat.”);
		}
		catch(IOException e)
		{
			System.out.println(“Problem with input from
				file numbers.dat.”);
		}
	}
}
```

Το παραπάνω πρόγραμμα θα δώσει σαν έξοδο:

```
Reading ALL the integers

in the file numbers.dat.

1

2

3

-1

End of reading from file.
```

Το παραπάνω πρόγραμμα χρησιμοποιεί την κλάση EOFException για να εντοπίσει το τέλος ενός αρχείου, το οποίο σημαίνει ότι μπορεί να χειριστεί αρχεία που περιέχουν οποιοδήποτε είδος ακεραίων, ακόμη και αρνητικών ακεραίων.

Δεν θα μεταβιβάσουν όλες οι μέθοδοι όλων των κλάσεων μια εξαίρεση EOFException όταν επιχειρήσουν να διαβάσουν μετά από το τέλος ενός αρχείου. Αν το πρόγραμμα μας διαβάζει από ένα δυαδικό αρχείο, τότε θα πρέπει να μεταβιβάζει μία εξαίρεση EOFException (ή οποιαδήποτε άλλη εξαίρεση) όταν το πρόγραμμα επιχειρεί να διαβάσει μετά από το τέλος του αρχείου.

Οι διαφορετικές μέθοδοι ανάγνωσης αρχείων (συνήθως σε διαφορετικές κλάσεις) ελέγχουν για το τέλος ενός αρχείου με διαφορετικούς τρόπους. Κάποιες μέθοδοι μεταβιβάζουν μία εξαίρεση τύπου EOFException ενώ κάποιες άλλες επιστρέφουν μία ειδική τιμή, όπως η null. Κατά την ανάγνωση από ένα αρχείο θα πρέπει να είμαστε προσεκτικοί και να ελέγχουμε για το τέλος του αρχείου με το σωστό τρόπο για τη μέθοδο που χρησιμοποιούμε. Αν ελέγξουμε για το τέλος του αρχείου με λάθος τρόπο, τότε θα συμβεί ένα από τα επόμενα δύο ενδεχόμενα: ή το πρόγραμμα μας θα εισέλθει σε έναν ακούσιο ατέρμονα βρόχο ή δεν θα τερματιστεί ομαλά. Για τις κλάσεις που παρουσιάζονται και εξετάζονται σε αυτό το κείμενο, ισχύει ο ακόλουθος κανόνας: Αν το πρόγραμμά μας διαβάζει από ένα δυαδικό αρχείο, τότε θα μεταβιβαστεί μία εξαίρεση EOFException όταν η ανάγνωση προχωρήσει πέρα από το τέλος του αρχείου. Αν το πρόγραμμα μας διαβάζει από ένα αρχείο κειμένου, τότε θα επιστραφεί κάποια ειδική τιμή, όπως η null, όταν το πρόγραμμά μας επιχειρήσει να διαβάσει μετά από το τέλος του αρχείου και δε θα μεταβιβαστεί καμία εξαίρεση EOFException.

3.8.6 Οι Κλάσεις FileInputStream και FileOutputStream

Κάθε φορά που δημιουργήσαμε μία ροή της κλάσης ObjectInputStream ή της κλάσης ObjectOutputStream, χρησιμοποιούσαμε την κλάση ροών FileInputStream ή την FileOutputStream, αντίστοιχα. Σε κάποια από τα τελευταία παραδείγματα χρησιμοποιήσαμε τις κλάσεις FileOutputStream και FileInputStream επειδή δέχονται ως όρισμα δομητή ένα όνομα αρχείου, όπως το όρισμα “numbers.dat” στα παραπάνω παραδείγματα. Ούτε η ObjectInputStream αλλά ούτε και η ObjectOutputStream δέχονται ένα όνομα αρχείου ως όρισμα δομητή. Άρα, όταν συνδέουμε μια ροή ObjectInputStream με ένα αρχείο χρησιμοποιώντας ένα όνομα αλφαριθμητικού, θα πρέπει να το κάνουμε σε δύο βήματα. Πρώτα, δημιουργούμε ένα αντικείμενο της κλάσης FileOutputStream, με την εντολή new FileOutputStream(“numbers.dat”) , για παράδειγμα, και μετά το χρησιμοποιούμε για να δημιουργήσουμε ένα αντικείμενο της κλάσης ObjectOutputStream με την εντολή:

```
ObjectOutputStream outputStream = new ObjectOutputStream
            (new FileOutputStream(“numbers.dat”));
```

Η ίδια ακριβώς διαδικασία σε δύο στάδια, με τη βοήθεια της FileInputStream ισχύει και για τη σύνδεση μίας ροής ObjectInputStream χρησιμοποιώντας ένα όνομα αλφαριθμητικού.

3.9 Δομές Δεδομένων Συλλογών Java

Στην ενότητα «Πίνακες» είδαμε τι γίνεται σε έναν πίνακα(array), δηλαδή σε μία σταθερή δομή δεδομένων. Στην γλώσσα προγραμματισμού Java, υπάρχουν δυναμικές δομές δεδομένων των οποίων το μέγεθος μεταβάλλεται κατά την διάρκεια εκτέλεσης ενός προγράμματος. Ένας πίνακας λοιπόν, είναι η πιο στοιχειώδης δομή δεδομένων.

Οι τύποι συλλογών δεδομένων στη Java είναι οι εξής:

- List
- Set
- Map

Και οι τεχνικές υλοποίησης αυτών:

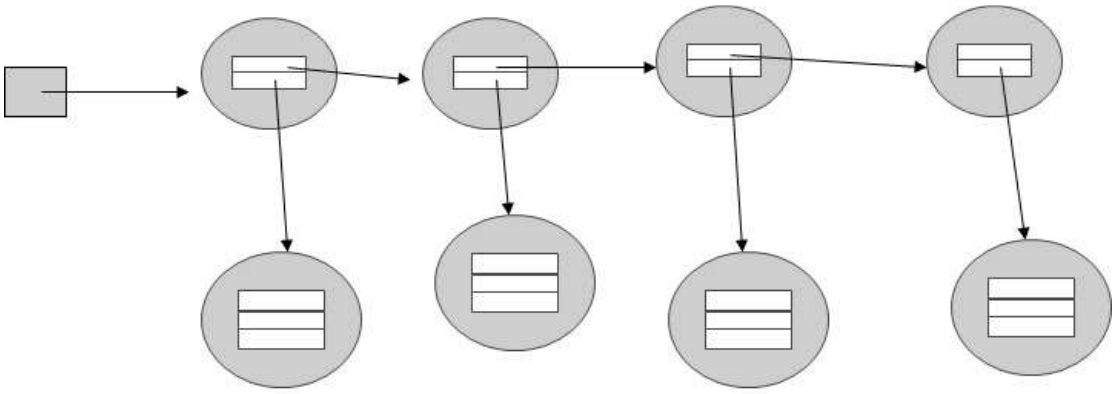
- linked
- array

Η Java χρησιμοποιεί τις παρακάτω δομές δεδομένων στις υλοποιήσεις των συλλογών:

- Συνδεδεμένη λίστα [linked list]
- Δυναμικό διάνυσμα [resizable array]
- «Ισοσκελισμένο» δένδρο [balanced tree]
- Πίνακα κατακερματισμού [hash table]

3.9.1 Συνδεδεμένη λίστα [linked list]:

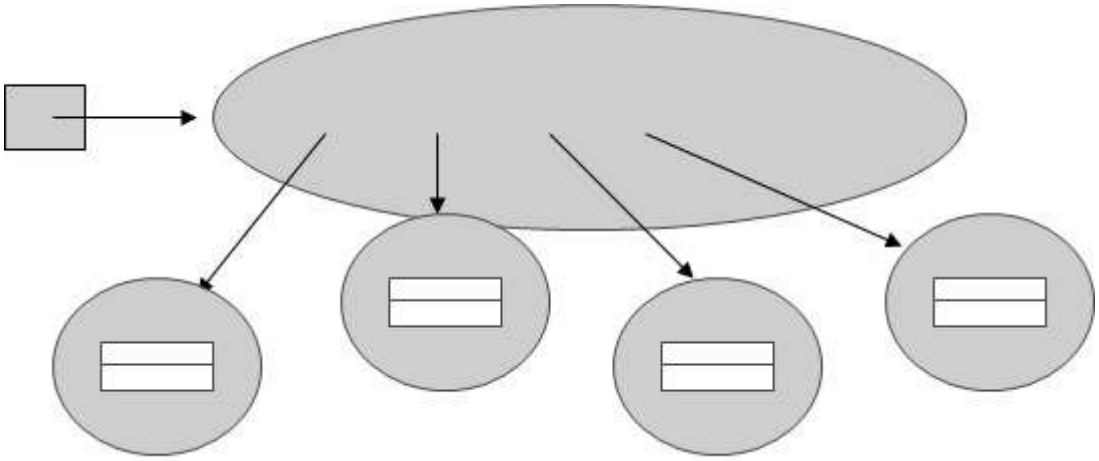
- Δομή βασισμένη αποκλειστικά σε «συνδέσμους» [links]



Εικόνα 3–1

3.9.2 Δυναμικό διάνυσμα [resizable array]:

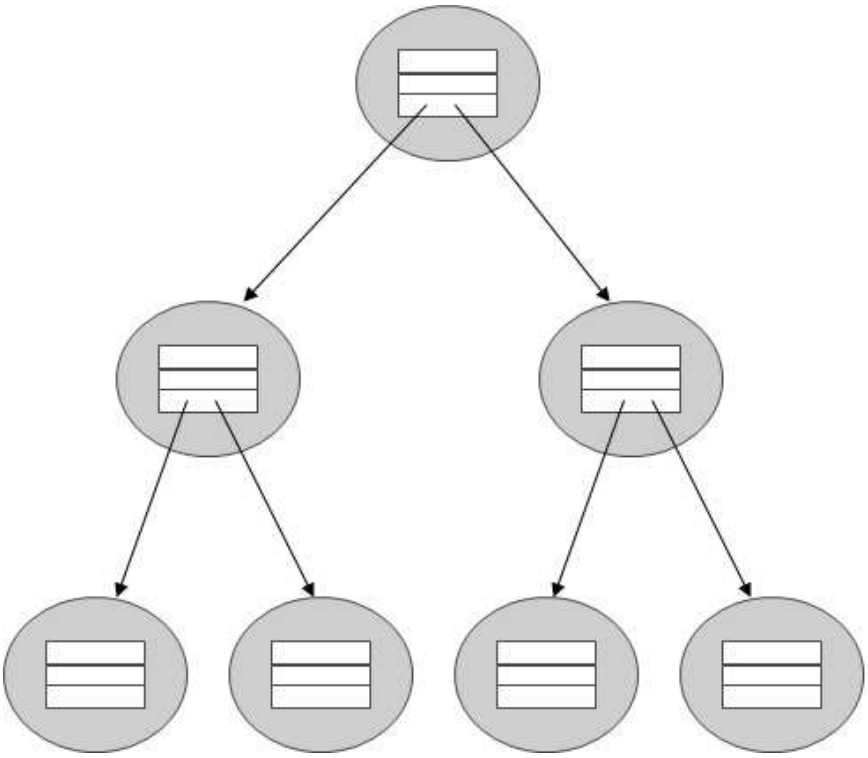
- Βασίζεται σε απλά διανύσματα
- Χρησιμοποιεί επανακαταχώρηση [reallocation]



Εικόνα 3–2

3.9.3 «Ισοσκελισμένο» δένδρο [balanced tree]:

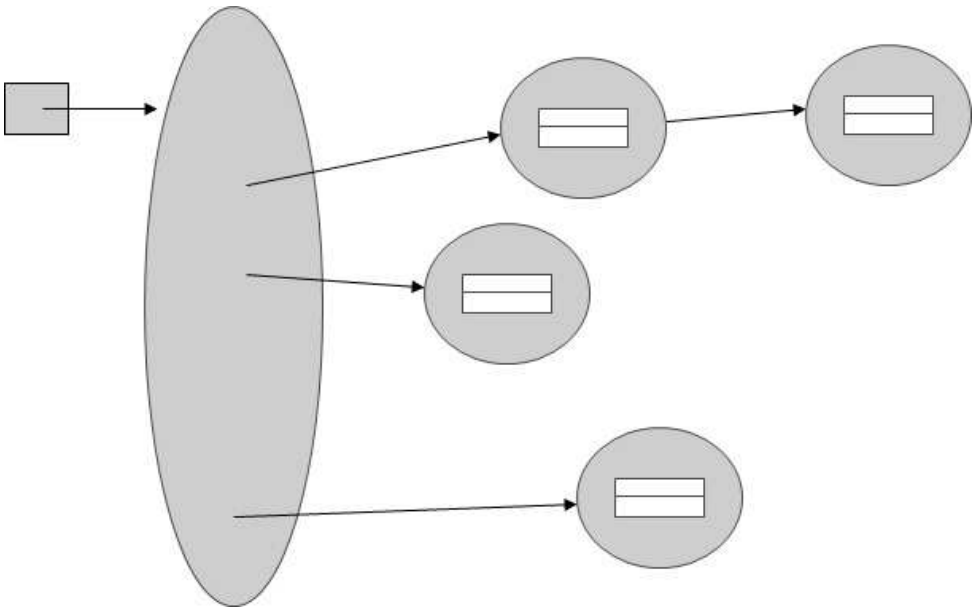
- Συνδεδεμένη δομή
- Δυαδικό (στη Java)
- Πάντοτε ισοσκελισμένο (μέσω ανακατανομής των στοιχείων του [reshuffling]



Εικόνα 3–3

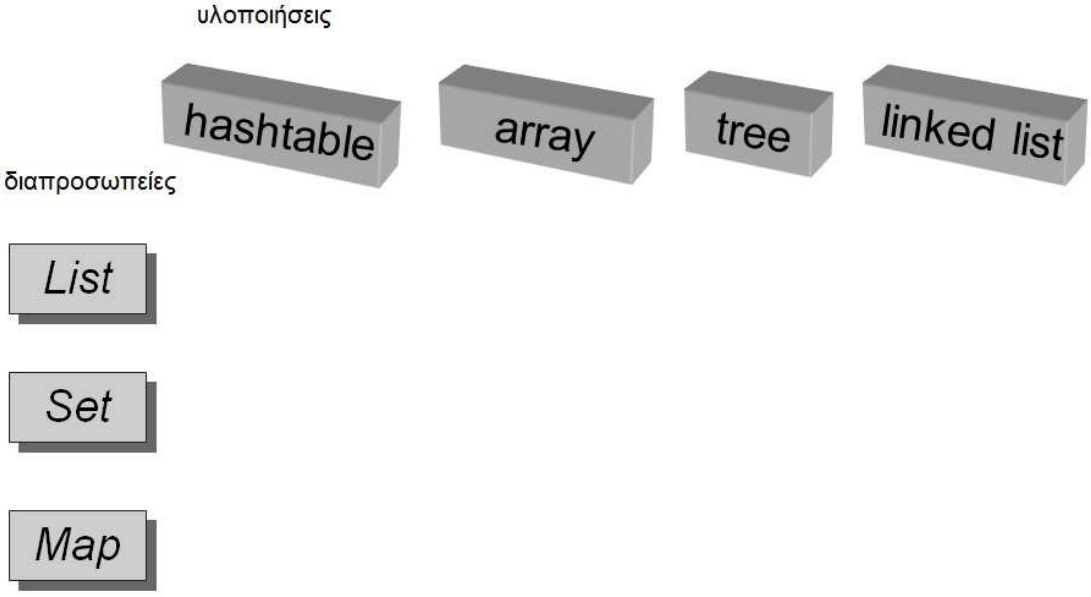
3.9.4 Πίνακας κατακερματισμού [hash table]:

- Συνδυασμός διανύσματος και συνδεδεμένης δομής
- Χρησιμοποιεί τιμές κατακερματισμού [hash values] για την τοποθέτηση των στοιχείων στην κατάλληλη θέση
- Πολύ γρήγορη ανάκτηση στοιχείων



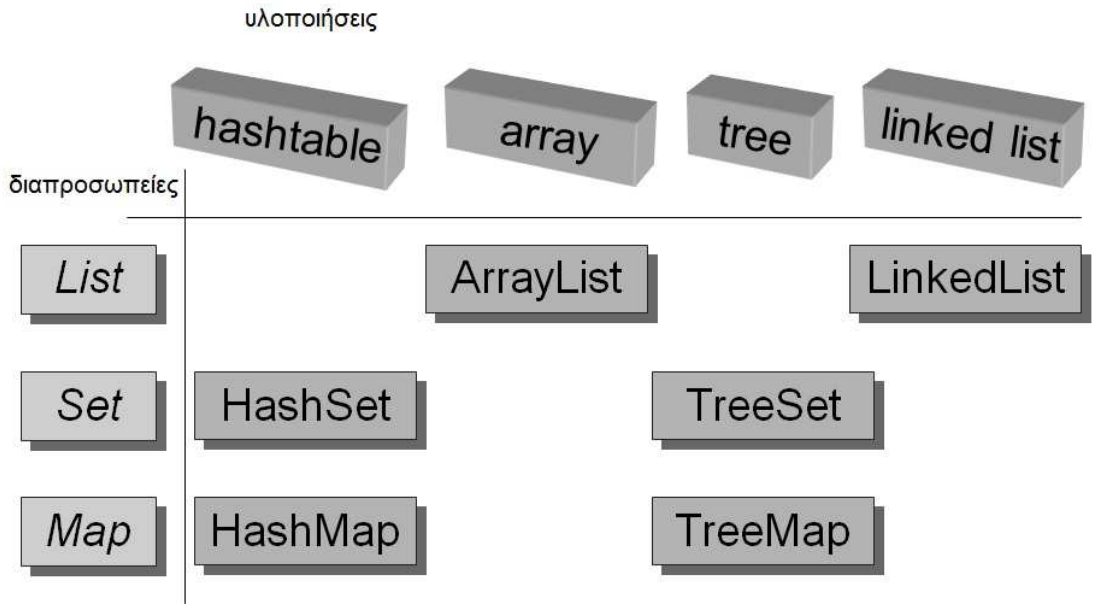
Εικόνα 3–4

3.9.5 Υλοποιήσεις:



Εικόνα 3–5

3.9.6 Υλοποιήσεις συλλογών δεδομένων Java:



Εικόνα 3–6

List:
ArrayList (σταθερός χρόνος προσπέλασης, χρησιμοποιείται στις περισσότερες περιπτώσεις)
LinkedList (γρήγορη εισαγωγή στοιχείων)
Set:
HashSet (γρήγορη δομή, χρησιμοποιείται στις περισσότερες περιπτώσεις)
TreeSet (ταξινομημένη)
Map:
HashMap (γρήγορη δομή)
TreeMap (ταξινομημένη)

Στην Java 1.1 υπήρχαν οι κλάσεις Vector και Hashtable οι οποίες στην Java 2 έχουν αντικατασταθεί από τις κλάσεις ArrayList και HashMap αντίστοιχα. Στην Java 2 πρέπει να χρησιμοποιούμε τις νέες κλάσεις συλλογής δεδομένων. Σε αυτό το κείμενο θα αναλυθούν λίγο περισσότερο η ArrayList και η HashMap επειδή στην παρούσα πτυχιακή εργασία χρησιμοποιήθηκαν ευρέως στον κώδικα του προγράμματος.

3.9.7 Η κλάση ArrayList

Η κλάση ArrayList υπάρχει στη βιβλιοθήκη java.util της Java. Αυτή η κλάση, όπως φαίνεται στις παραπάνω εικόνες, παριστάνει μία δομή δεδομένων η οποία συνδυάζει τα πλεονεκτήματα του πίνακα (array) και της συνδεδεμένης λίστας (list). Το βασικό του χαρακτηριστικό είναι η μεγάλου βαθμού δυναμικότητα – ένα αντικείμενό της κλάσης ArrayList μπορεί να περιέχει έναν μη προκαθορισμένο αριθμό στοιχείων. Τα στοιχεία αυτά μπορεί να είναι μεταβλητές οτιδήποτε τύπου, ή αντικείμενα άλλων ή ίδιας κλάσεων.

Όταν αρχικοποιείται ένα αντικείμενο της κλάσης αυτής δεν δηλώνεται προκαθορισμένο μέγεθος, αλλά αυτό προσδιορίζεται αυτόματα στη συνέχεια. η μέθοδος ArrayList.size() επιστρέφει τον αριθμό αντικειμένων που περιέχονται.

Η απόδοση της ArrayList συγκρίνοντάς την με αυτή της συνδεδεμένης λίστα και του απλού πίνακα (array) φαίνεται στον παρακάτω πίνακα Πίνακας 3.13:

	Συνδεδεμένη Λίστα	**Array (πίνακας)**	**ArrayList (δυναμικός πίνακας)**
Δεικτοδότηση (Indexing)	Θ(n)	Θ(1)	Θ(1)
Πρόσθεση/Διαγραφή στο τέλος	Θ(1)	N/A – δεν υποστηρίζεται	Θ(1)
Πρόσθεση/Διαγραφή στη μέση	Θ(1)	N/A – δεν υποστηρίζεται	Θ(n)
Χαμένος χώρος (μέση τιμή)	Θ(n)	0	Θ(n)

Πίνακας 3.13

3.9.7.1 Δομητές

Στον παρακάτω πίνακα Πίνακας 3.14 φαίνονται οι δομητές της κλάσης ArrayList:

Constructor Summary
ArrayList() Constructs an empty list with an initial capacity of ten.
ArrayList(Collection c) Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.
ArrayList(int initialCapacity) Constructs an empty list with the specified initial capacity.

Πίνακας 3.14

3.9.7.2 Μέθοδοι

Ο παρακάτω πίνακας Πίνακας 3.15 περιέχει όλες τις μεθόδους της κλάσης ArrayList :

Method Summary	
void	**add**(int index, Object element) Inserts the specified element at the specified position in this list.
boolean	**add**(Object o) Appends the specified element to the end of this list.
boolean	**addAll**(Collection c) Appends all of the elements in the specified Collection to the end of this list, in the order that they are returned by the specified Collection's Iterator.
boolean	**addAll**(int index, Collection c) Inserts all of the elements in the specified Collection into this list, starting at the specified position.
void	**clear**() Removes all of the elements from this list.
Object	**clone**() Returns a shallow copy of this ArrayList instance.
boolean	**contains**(Object elem) Returns true if this list contains the specified element.
void	**ensureCapacity**(int minCapacity) Increases the capacity of this ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.
Object	**get**(int index) Returns the element at the specified position in this list.

int	**indexOf**(Object elem) Searches for the first occurrence of the given argument, testing for equality using the equals method.
boolean	**isEmpty**() Tests if this list has no elements.
int	**lastIndexOf**(Object elem) Returns the index of the last occurrence of the specified object in this list.
Object	**remove**(int index) Removes the element at the specified position in this list.
protected void	**removeRange**(int fromIndex, int toIndex) Removes from this List all of the elements whose index is between fromIndex, inclusive and toIndex, exclusive.
Object	**set**(int index, Object element) Replaces the element at the specified position in this list with the specified element.
int	**size**() Returns the number of elements in this list.
Object[]	**toArray**() Returns an array containing all of the elements in this list in the correct order.
Object[]	**toArray**(Object[] a) Returns an array containing all of the elements in this list in the correct order; the runtime type of the returned array is that of the specified array.
void	**trimToSize**() Trims the capacity of this ArrayList instance to be the list's current size.

Πίνακας 3.15

3.9.8 Η κλάση HashMap

Και η κλάση HashMap ανήκει στην βιβλιοθήκη java.util της γλώσσας προγραμματισμού Java.

Η κλάση HashMap υλοποιεί έναν πίνακα αντιστοίχησης. Ένας τέτοιος πίνακας αντιστοιχίζει κλειδιά σε τιμές. Δεν επιτρέπονται διπλά κλειδιά, ενώ κάθε κλειδί αντιστοιχίζεται σε μία το πολύ τιμή. Μια τέτοια δομή επιτρέπει την αναζήτηση μιας συγκεκριμένης τιμής με βάση ένα μοναδικό κλειδί. Παράδειγμα ενός HashMap αποτελεί ένα λεξικό, όπου το λήμμα αποτελεί το κλειδί ενώ η ερμηνεία του, για παράδειγμα, η μετάφρασή του σε μία άλλη γλώσσα, αποτελεί την τιμή (value) στην οποία αντιστοιχίζεται το λήμμα. Παράδειγμα:

```
HashMap<String, String> dictionary = new HashMap<String, String>();
dictionary.put("boy", "αγόρι");
dictionary.put("cat", "γάτα");
String catTrans = dictionary.get("cat");
```

Η συμβολοσειρά catTrans του παραδείγματος παίρνει την τιμή "γάτα" , με την οποία έχει αντιστοιχηθεί στο λεξικό η συμβολοσειρά "cat".

3.9.8.1 Δομητές

Στον παρακάτω πίνακα Πίνακας 3.16 φαίνονται οι δομητές της κλάσης HashMap:

Constructor Summary
HashMap() Constructs an empty HashMap with the default initial capacity (16) and the default load factor (0.75).
HashMap(int initialCapacity) Constructs an empty HashMap with the specified initial capacity and the default load factor (0.75).
HashMap(int initialCapacity, float loadFactor) Constructs an empty HashMap with the specified initial capacity and load factor.
HashMap(Map<? extends K,? extends V> m)

Constructs a new HashMap with the same mappings as the specified Map.

Πίνακας 3.16

3.9.8.2 Μέθοδοι

Στον παρακάτω πίνακα Πίνακας 3.17 φαίνονται οι μέθοδοι της κλάσης HashMap:

Method Summary	
void	**clear**() Removes all of the mappings from this map.
Object	**clone**() Returns a shallow copy of this HashMap instance: the keys and values themselves are not cloned.
boolean	**containsKey**(Object key) Returns true if this map contains a mapping for the specified key.
boolean	**containsValue**(Object value) Returns true if this map maps one or more keys to the specified value.
Set<Map.Entry<K,V>>	**entrySet**() Returns a Set view of the mappings contained in this map.
V	**get**(Object key) Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
boolean	**isEmpty**() Returns true if this map contains no key-value mappings.
Set<K>	**keySet**() Returns a Set view of the keys contained in this map.

V	**put**(K key, V value) Associates the specified value with the specified key in this map.
void	**putAll**(Map<? extends K,? extends V> m) Copies all of the mappings from the specified map to this map.
V	**remove**(Object key) Removes the mapping for the specified key from this map if present.
int	**size**() Returns the number of key-value mappings in this map.
Collection<V>	**values**() Returns a Collection view of the values contained in this map.

Πίνακας 3.17

3.10 Παραθυρικά περιβάλλοντα με χρήση της swing

Μέχρι τώρα σε όλα σχεδόν τα παραδείγματα αυτού του κειμένου έχουμε χρησιμοποιήσει την απλούστερη μορφή εισόδου. Ο χρήστης εισάγει απλό κείμενο από το πληκτρολόγιο και το πρόγραμμα το εμφανίζει πάλι ως απλό κείμενο, στην οθόνη. Τα σύγχρονα προγράμματα δε χρησιμοποιούν τέτοια απλή είσοδο και έξοδο.

Τα σύγχρονα προγράμματα χρησιμοποιούν παραθυρικά περιβάλλονται με μενού και κουμπιά, τα οποία επιτρέπουν στο χρήστη να κάνει επιλογές χρησιμοποιώντας το ποντίκι. Στην ενότητα αυτή θα παρουσιαστεί η βιβλιοθήκη κλάσεων Swing, που είναι μια ειδική βιβλιοθήκη κλάσεων που μας δίνει την δυνατότητα να σχεδιάσουμε τέτοια παραθυρικά περιβάλλοντα. Η Swing είναι μια τυποποιημένη βιβλιοθήκη που είναι τοποθετημένη στη Java. Υπάρχει ολόκληρη βιβλιογραφία αποκλειστικά για τη Swing και συνεπώς δεν μπορούμε να δώσουμε μία ολοκληρωμένη περιγραφή σε αυτήν την πτυχιακή εργασία.

3.10.1 Γραφικά Περιβάλλοντα Χρήστη (GUI)

Τα παραθυρικά συστήματα που αλληλεπιδρούν με το χρήστη συνήθως λέγονται GUI (προφέρεται «γκούι». Ο όρος προέρχεται από τα αρχικά των λέξεων **graphical user**

interface (γραφικό περιβάλλον χρήστη) και είναι σχεδόν αυτονόητος. Λέγεται *γραφικό* επειδή χρησιμοποιεί γραφικά στοιχεία όπως παράθυρα, κουμπιά και μενού. Λέγεται *περιβάλλον χρήστη* επειδή είναι μέρος ενός προγράμματος που αλληλεπιδρά (διασυνδέεται) με το χρήστη. Ένα GUI παίρνει πληροφορίες από το χρήστη και τις δίνει στο πρόγραμμα για επεξεργασία. Όταν το πρόγραμμα τελειώσει με την επεξεργασία των πληροφοριών, το GUI δίνει τα αποτελέσματα στο χρήστη, συνήθως με τη μορφή κάποιου παραθύρου.

Το είδος προγραμματισμού που ασχολείται με GUI, ανήκει στην κατηγορία που ονομάζεται **προγραμματισμός καθοδηγούμενος από γεγονότα**. Δηλαδή, τα προγράμματα Swing καθώς και τα περισσότερα προγράμματα άλλων GUI χρησιμοποιούν γεγονότα και χειριστές γεγονότων. Ένα **γεγονός (event)** σε ένα γραφικό περιβάλλον χρήστη είναι ένα αντικείμενο που αναπαριστά κάποια λειτουργία (ενέργεια) όπως το πάτημα ενός πλήκτρου στο ποντίκι, το σύρσιμο του ποντικιού, το πάτημα ενός πλήκτρου στο πληκτρολόγιο, το κλικ με το ποντίκι στο κουμπί τερματισμού του παραθύρου ενός παραθύρου ή οποιαδήποτε άλλη λειτουργία που απαιτεί μία απόκριση.

Όταν το αντικείμενο δημιουργεί ένα γεγονός, τότε λέμε ότι το αντικείμενο **πυροδοτεί (fire)** το γεγονός. Στη Swing, κάθε αντικείμενο που μπορεί να πυροδοτήσει γεγονότα, όπως για παράδειγμα το κλικ με το ποντίκι πάνω σε ένα κουμπί, μπορεί να έχει ένα ή περισσότερα **αντικείμενα ακροατές (listener object)**. Εμείς, οι προγραμματιστές, είμαστε αυτοί που θα καθορίσουμε ποια αντικείμενα είναι αντικείμενα-ακροατές για ένα συγκεκριμένο αντικείμενο που μπορεί να προκαλέσει το γεγονός. Ένα αντικείμενο-ακροατής έχει μεθόδους που καθορίζουν τι θα συμβεί όταν στέλνονται σε αυτόν διάφορα γεγονότα. Οι μέθοδοι αυτές που χειρίζονται τα γεγονότα λέγονται **χειριστές γεγονότων (event handlers)** και ορίζονται (ή επανακαθορίζονται) από τον προγραμματιστή.

3.10.2 Βασικές λειτουργίες της Swing

Ένα **παράθυρο (window)** είναι ένα τμήμα της οθόνης του χρήστη που εξυπηρετεί ως μία μικρότερη οθόνη μέσα στην οθόνη. Υπάρχουν πολλά πράγματα που μπορούμε να βάλουμε μέσα σε ένα παράθυρο όταν σχεδιάζουμε ένα GUI. Παρακάτω θα παρουσιάσουμε ένα απλό κώδικα που φτιάχνει ένα παράθυρο, το οποίο περιέχει κάποια βασικά πράγματα όπως το κουμπί που κλείνει το παράθυρο, ένα πλαίσιο που μπορεί να γραφεί κείμενο, αλλαγή χρώματος παραθύρου. Ας δούμε τον κώδικα του προγράμματος και έπειτα θα επισημάνουμε κάποια πράγματα:

```
import javax.swing.*;
public class FirstSwingDemo
{
	public static final int WIDTH = 300;
	public static final int HEIGHT = 200;
	public static void main(String[] args)
	{
		JFrame myWindow = new JFrame();
		myWindow.setSize(WIDTH, HEIGHT);
		JLabel myLabel = new JLabel(“Please don’t click that button!”);
```

```
            myWindow.getContentPane().add(myLabel);
            WindowDestroyer myListener = new WindowsDestroyer();
            myWindow.addWindowListener(myListener);
            myWindow.setVisible(true);
        }
}
```

Αυτό το παράδειγμα θα μπορούσε να δώσει το εξής GUI:

Εικόνα 3–7

,

Όπως φαίνεται και στην παραπάνω εικόνα Εικόνα 3–7, το πρόγραμμα θα δημιουργήσει ένα παράθυρο το οποίο θα περιέχει ένα μήνυμα και τα κλασικά κουμπιά ελαχιστοποίησης, μεγιστοποίησης ή σμίκρυνσης και κλεισίματος. Ας μελετήσουμε βήμα-βήμα τον κώδικα:

```
import javax.swing.*;
```

Αυτή η πρώτη γραμμή του κώδικα λέει ότι το πρόγραμμα χρησιμοποιεί τη βιβλιοθήκη Swing. Ένα αρχείο που περιέχει ένα πρόγραμμα το οποία χρησιμοποιεί τη βιβλιοθήκη Swing θα πρέπει να αρχίζει μ' αυτήν τη γραμμή (πιθανώς και με κάποιες άλλες γραμμές import). Το υπόλοιπο του προγράμματος είναι ένας απλός ορισμός κλάσης με μία μόνο μέθοδο main. Η πρώτη γραμμή κώδικα της μεθόδου main δημιουργεί ένα αντικείμενο της κλάσης JFrame:

```
JFrame myWindow = new JFrame();
```

Το όνομα myWindow επιλέγεται από εμάς, τον εκάστοτε προγραμματιστή. Το αντικείμενο είναι ένα πολύ απλό παράθυρο και έχει όπως είπαμε προηγουμένως, μεταξύ άλλων, ένα περίγραμμα, μία θέση για ένα τίτλο (δε χρησιμοποιείται τίτλος σ' αυτό το πρόγραμμα) και το κουμπί τερματισμού παραθύρου που περιμένουμε να βρύμε σε οποιαδήποτε παράθυρο. Σύντομα θα δούμε ότι δε χρησιμοποιούμε απλά αντικείμενα JFrame, αλλά ορίζουμε μία απορρέουσα κλάση της κλάσης JFrame και χρησιμοποιούμε αντικείμενα της απορρέουσας κλάσης. Όπως, γι' αυτό το πρώτο πρόγραμμα επίδειξης, χρησιμοποιήσαμε την JFrame κατευθείαν. Η επόμενη γραμμή θέτει το μέγεθος του παραθύρου JFrame:

```
myWindow.setSize(WIDTH, HEIGHT);
```

Η μέθοδος setSize είναι μια μέθοδος της κλάσης JFrame και θέτει το μέγεθος του παραθύρου.

Οι μονάδες μεγέθους στη Swing θα εξεταστούν παρακάτω σε αυτήν την ενότητα. Η επόμενη εντολή δημιουργεί ένα αντικείμενο της κλάσης JLabel και δίνει όνομα στο αντικείμενο myLabel:

```
JLabel myLabel = new JLabel(“Please don’t click that button!”);
```

Το όνομα myLabel επιλέχθηκε τυχαία από εμάς. Ένα αντικείμενο της κλάσης JLabel συνήθως λέγεται **ετικέτα (Label)** και είναι ένας ειδικός τύπος κειμένου που μπορεί να προστεθεί σε ένα αντικείμενο JFrame (ή σε οποιοδήποτε άλλο συγκεκριμένο είδος αντικειμένων). Το αλφαριθμητικό για την ετικέτα δίδεται ως όρισμα στο δομητή της κλάσης JLabel, το οποίο στη περίπτωση αυτή είναι “Please don’t click that button!”.

Η επόμενη εντολή του προγράμματος προσθέτει την ετικέτα myLabel στο αντικείμενο JFrame που λέγεται myWindow:

```
myWindow.getContentPane().add(myLabel);
```

Αυτή η εντολή χρειάζεται περαιτέρω επεξήγηση και γι’ αυτό ας την εξετάσουμε ξεχωριστά. Η μέθοδος getContentPane είναι μία μέθοδος της κλάσης JFrame που δημιουργεί την **επιφάνεια** ή το **περιεχόμενο (content pane)** του αντικειμένου JFrame. Κάθε αντικείμενο JFrame έχει μία επιφάνεια. Δεν προσθέτουμε στοιχεία κατευθείαν στο αντικείμενο JFrame αλλά τα προσθέτουμε στην επιφάνειά του. Μπορούμε να θεωρήσουμε ότι η επιφάνεια του αντικειμένου JFrame είναι το «εσωτερικό του». Έτσι, η κλήση myWindow.getContentPane() είναι η επιφάνεια του αντικειμένου myWindow (δηλ. το εσωτερικό του αντικειμένου myWindow). Η ετικέτα myLabel προστίθεται στη myWindow.getContentPane() (δηλαδή, στην επιφάνεια του myWindow) χρησιμοποιώντας τη μέθοδο add. Κάθε επιφάνεια JFrame έχει μία μέθοδο που λέγεται add. Η μέθοδος add είναι ήδη ορισμένη για εμάς και δε χρειάζεται να την ορίσουμε εμείς.

Για να κλείσουμε το παράθυρο αυτό κάνουμε κλικ στο κουμπί τερματισμού παραθύρου. Όταν κάνουμε κλικ με το ποντίκι σ’ αυτό το κουμπί, το παράθυρο πυροδοτεί ένα γεγονός και το στέλνει σε ένα αντικείμενο-ακροατή, το οποίο κλείνει το παράθυρο. Στο πρόγραμμα αυτό, το αντικείμενο-ακροατής λέγεται myListener και είναι μέλος της κλάσης WindowDestroyer. Η παρακάτω γραμμή κώδικα δημιουργεί ένα νέο αντικείμενο της κλάσης WindowDestroyer και το ονομάζει myListener:

```
WindowDestroyer myListener = new WindowsDestroyer();
```

Το όνομα myListener είναι τυχαίο και ο καθένας μπορεί να βάλει ότι όνομα επιθυμεί. Η επόμενη εντολή που φαίνεται πιο κάτω:

```
myWindow.addWindowListener(myListener);
```

συσχετίζει το αντικείμενο myListener με το αντικείμενο (δηλ. το παράθυρο) myWindow, έτσι ώστε το myListener να λαμβάνει οποιοδήποτε event προκαλείται από το αντικείμενο myWindow.

Ένα αντικείμενο, όπως το myListener, που λαμβάνει γεγονότα από ένα άλλο αντικείμενο λέγεται **listener (ακροατής)**. Ένας ακροατής που δέχεται (ακούει) γεγονότα από ένα παράθυρο λέγεται **window listener (ακροατής παραθύρων)**. Η συσχέτιση ενός ακροατή με το αντικείμενο το οποίο δέχεται (ακούει) λέγεται **καταχώρηση του ακροατή (registering the listener)**.

Ένα αντικείμενο της κλάσης WindowDestroyer θα κλείνει το παράθυρο myWindow όταν το myWindow προκαλέσει ένα κατάλληλο γεγονός. Θα πρέπει να ορίσουμε την κλάση WindowDestroyer, αλλά αυτό θα το κάνουμε λίγο αργότερα στο παρόν κείμενο. Για την ώρα, ας υποθέσουμε ότι η κλάση WindowDestroyer έχει οριστεί έτσι ώστε όταν ο χρήστης κάνει κλικ στο κουμπί τερματισμού παραθύρου του myWindow, το αντικείμενο myListener θα κλείσει το παράθυρο myWindow και θα τερματίσει το πρόγραμμα.

Η τελευταία εντολή στη μέθοδο main είναι μία κλήση προς τη μέθοδο setVisible:

```
myWindow.setVisible(true);
```

Αυτή η κλήση της setVisible κάνει το παράθυρο που λέγεται myWindow να είναι ορατό στην οθόνη. Η μέθοδος setVisible είναι μία μέθοδος της κλάσης JFrame (καθώς επίσης και πολλών άλλων κλάσεων της Swing). Με το όρισμα true, όπως είδαμε στην παραπάνω εικόνα, το αντικείμενο φαίνεται (είναι, δηλαδή, ορατό). Αν το όρισμα ήταν false, τότε το αντικείμενο δε θα φαινόταν (θα ήταν, δηλαδή, αόρατο).

Το αντικείμενο myListener είναι ένα μέλος της κλάσης WindowDestroyer. Έχουμε υποθέσει ότι η κλάση WindowDestroyer έχει ήδη οριστεί, αλλά για να τρέξουμε το πρόγραμμα πρέπει πρώτα να ορίσουμε και να μεταγλωττίσουμε την κλάση WindowDestroyer. Η κλάση αυτή φαίνεται παρακάτω:

```
import java.awt.*;
import java.awt.event.*;
public class WindowDestroyer extends WindowAdapter
{
	public void windowClosing(WindowEvent e)
	{
		System.exit(0);
	}
}
```

Η παραπάνω κλάση WindowDestroyer είναι, και συνήθως θα είναι, μία απορρέουσα κλάση της WindowAdapter και κληρονομεί όλες τις μεθόδους αυτής. Αυτό άλλωστε φαίνεται και στον ορισμό της κλάσης, από τις λέξεις “extends WindowAdapter”.

Η μέθοδος που χειρίζεται αυτά τα γεγονότα λέγεται windowClosing. Ο ορισμός της μεθόδου είναι πολύ απλός και εκτελεί μόνον την εντολή:

```
System.exit(0);
```

Όπως θα εξηγήσουμε στην παρακάτω Συμβουλή Java, αυτή η εντολή τερματίζει το πρόγραμμα, άρα τερματίζει και το παράθυρο. Όσο για τις δύο πρώτες γραμμές:

```
import java.awt.*;
import java.awt.event.*;
```

αυτές ενημερώνουν τον μεταγλωττιστή για το πού βρίσκονται οι ορισμοί για την κλάση WindowAdapter και για το χειρισμό γεγονότων. Στον παρακάτω πίνακα Πίνακας 3.18 φαίνονται οι μέθοδοι της κλάσης WindowAdapter:

Method Summary	
void	**windowActivated**(WindowEvent e)

	Invoked when a window is activated.
void	**windowClosed**(WindowEvent e) Invoked when a window has been closed.
void	**windowClosing**(WindowEvent e) Invoked when a window is in the process of being closed.
void	**windowDeactivated**(WindowEvent e) Invoked when a window is de-activated.
void	**windowDeiconified**(WindowEvent e) Invoked when a window is de-iconified.
void	**windowGainedFocus**(WindowEvent e) Invoked when the Window is set to be the focused Window, which means that the Window, or one of its subcomponents, will receive keyboard events.
void	**windowIconified**(WindowEvent e) Invoked when a window is iconified.
void	**windowLostFocus**(WindowEvent e) Invoked when the Window is no longer the focused Window, which means that keyboard events will no longer be delivered to the Window or any of its subcomponents.
void	**windowOpened**(WindowEvent e) Invoked when a window has been opened.
void	**windowStateChanged**(WindowEvent e) Invoked when a window state is changed.

Πίνακας 3.18

3.10.3 Μονάδες Μεγέθους για Αντικείμενα στην Οθόνη

Στη Swing, το μέγεθος ενός αντικειμένου στην οθόνη μετριέται σε pixels. Ένα **pixel (εικονοστοιχείο)** είναι η μικρότερη μονάδα χώρου πάνω στην οποία μπορεί να γράφει η οθόνη. Μπορούμε να θεωρήσουμε ένα pixel σαν ένα μικρό τετράγωνο που μπορεί να έχει ένα χρώμα από ένα μικρό αριθμό προκαθορισμένων χρωμάτων και ότι η οθόνη μας είναι καλυμμένη από αυτά τα μικρά pixels. Όσο πιο πολλά pixels έχουμε στην οθόνη μας, τόσο πιο μεγάλη είναι η ανάλυση της οθόνης. Δηλαδή, όσο περισσότερα pixels έχουμε, τόσο μεγαλύτερη λεπτομέρεια μπορούμε να δούμε.

Το μέγεθος ενός pixel εξαρτάται από το μέγεθος και την ανάλυση της οθόνης μας. Αν και η Swing χρησιμοποιεί τα pixels σαν να ήταν μονάδες μήκους, στην πραγματικότητα δεν εκφράζουν κάποιο σταθερό μήκος. Το μήκος ενός pixel θα ποικίλλει από οθόνη σε οθόνη. Σε μία οθόνη με υψηλή ανάλυση (με πολλά pixels), ένα αντικείμενο μεγέθους 300 επί 200 θα φαίνεται πολύ μικρό ενώ σε μία οθόνη χαμηλής ανάλυσης (λίγα pixels), το ίδιο αντικείμενο θα φαίνεται πολύ μεγάλο. Όπως είδαμε στο παραπάνω πρόγραμμα, το αντικείμενό μας ήταν ένα παράθυρο με διαστάσεις 300 pixels πλάτος επί 200 pixels ύψος, αλλά το πραγματικό μέγεθος θα εξαρτηθεί από την ανάλυση της οθόνης που χρησιμοποιούμαι όταν τρέξουμε το πρόγραμμα.

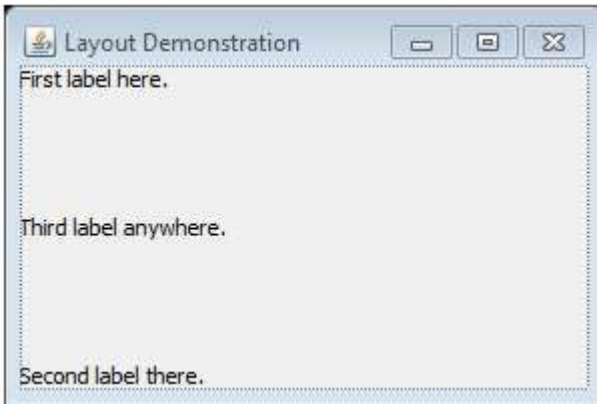
3.10.4 Διαχειριστές Διάταξης

Έχουμε δει ότι μπορούμε να προσθέσουμε ένα αντικείμενο JLabel σε ένα αντικείμενο JFrame χρησιμοποιώντας τη μέθοδο getContentPane για να πάρουμε την επιφάνεια του παραθύρου JFrame και μετά χρησιμοποιώντας τη μέθοδο add να προσθέσουμε μία ετικέτα JLabel στην επιφάνεια. Αν προσθέσουμε μία ετικέτα σε ένα κενό παράθυρο είναι προφανής, όπως είδαμε, η θέση ή το σημείο που θα τοποθετηθεί η ετικέτα. Τι γίνεται όμως όταν προσθέσουμε δύο ή περισσότερες ετικέτες; Πως θα τοποθετηθούν; Η μία πάνω στην άλλη; Η μία δίπλα στην άλλη; Ποια θα είναι πρώτη και ποια θα είναι δεύτερη, κτλ; Η διευθέτηση αυτή γίνεται από εάν ειδικό τύπο αντικειμένου που λέγεται **διαχειριστής διάταξης (layout manager)**. Ο διαχειριστής διάταξης τοποθετεί τα στοιχεία που προσθέτουμε βάσει συγκεκριμένων κανόνων, οι οποίοι είναι διαφορετικοί για διαφορετικούς διαχειριστές διάταξης. Το παράδειγμα που ακολουθεί προσθέτει τρεις ετικέτες σε ένα παράθυρο JFrame και χρησιμοποιεί ένα διαχειριστή διάταξης για να τοποθετήσει τις τρεις αυτές ετικέτες. Η παρακάτω εικόνα περιέχει ένα παράδειγμα μία κλάσης που χρησιμοποιείται για τη δημιουργία ενός παραθύρου με τρεις ετικέτες. Για την τοποθέτηση των τριών ετικετών, της μίας κάτω από άλλη σε τρεις γραμμές, χρησιμοποιείται ένας διαχειριστής διάταξης. Ας δούμε τη κλάση:

```
import javax.swing.*;
import java.awt.*;
public class BorderLayoutDemo extends JFrame
{
	public static final int WIDTH = 300;
	public static final int HEIGHT = 200;
	public static void main(String[] args)
	{
		BorderLayoutDemo gui = new BorderLayoutDemo();
		gui.setVisible(true);
	}
	public BorderLayoutDemo()
	{
```

```
            setSize(WIDTH, HEIGHT);
            addWindowListener(new WindowDestroyer());
            setTitle("Layout Demonstration");
            Container content = getContentPane();
            content.setLayout(new BorderLayout());
            JLabel label1 = new JLabel("First label here.");
            content.add(label1, BorderLayout.NORTH);
            JLabel label2 = new JLabel("Second label there.");
            content.add(label2, BorderLayout.SOUTH);
            JLabel label3 = new JLabel("Third label anywhere.");
            content.add(label3, BorderLayout.CENTER);
      }
}
```

Το GUI που προκύπτει είναι το εξής:



Εικόνα 3–8

Κατ' αρχήν, θα πρέπει να προσέξουμε ένα δευτερεύον σημείο το οποίο δεν έχουμε συναντήσει μέχρι τώρα στη Swing. Έχουμε βάλει μία μέθοδο επίδειξης main μέσα στο ορισμό κλάσης. Κανονικά, μία κλάση GUI Swing χρησιμοποιείται για την δημιουργία και την εμφάνιση ενός GUI μέσα σε μία μέθοδο main (ή κάποια άλλη μέθοδο) μίας άλλης κλάσης και όχι μέσα της κλάσης για το GUI Swing. Ωστόσο, είναι απολύτως έγκυρη και μερικές φορές πολύ βολική, η τοποθέτηση μία μεθόδου main μέσα στο ορισμό της κλάσης GUI έτσι ώστε να είναι εύκολη η εμφάνιση ενός δείγματος του GUI. Πρέπει να προσέξουμε, επίσης, ότι η μέθοδος main που δίνεται μέσα στην ίδια την κλάση είναι γραμμένη με τον ίδιο τρόπο που γράφεται μία μέθοδος main μέσα σε μία άλλη κλάση. Πιο συγκεκριμένα, πρέπει να κατασκευάσουμε ένα αντικείμενο της κλάσης, όπως στην παρακάτω εντολή της μεθόδου main στο παραπάνω πρόγραμμα:

```
BorderLayoutDemo gui = new BorderLayoutDemo();
```

Ένα διαχειριστής διάταξης προστίθεται στο GUI της παραπάνω εικόνας Εικόνα 3–8 με την ακόλουθη εντολή:

```
content.setLayout(new BorderLayout());
```

Η BorderLayout είναι μία κλάση διαχείρισης διάταξης, το οποίο σημαίνει ότι η κλήση new BorderLayout() δημιουργεί ένα αντικείμενο της κλάσης BorderLayout. Αυτό το αντικείμενο έχει την ευθύνη για τη διάταξη των συστατικών (στην περίπτωσή μας, ετικέτες) που προστίθενται στο GUI. Για να βοηθηθούμε περισσότερο, σημειώνουμε ότι η προηγούμενη κλήση της setLayout είναι ισοδύναμη με τις δύο παρακάτω εντολές:

```
BorderLayout manager = new BorderLayout();
```

```
content.setLayout(manager);
```

Προσοχή, η μέθοδος setLayout δεν καλείται από το ίδιο το παράθυρο JFrame, αλλά από τη επιφάνεια του παραθύρου JFrame, το οποίο στη συγκεκριμένη περίπτωση ονομάζεται content. Αυτό συμβαίνει επειδή στην πραγματικότητα προσθέτουμε ετικέτες στην επιφάνεια και όχι (απευθείας) στο παράθυρο JFrame. Γενικά, θα πρέπει να καλούμε τη setLayout με το ίδιο αντικείμενο που χρησιμοποιούμε για να καλέσουμε τη μέθοδο add (και μέχρι τώρα αυτό το αντικείμενο ήταν πάντοτε η επιφάνεια ενός αντικειμένου JFrame).

Ένας διαχειριστής διάταξης BorderLayout, τοποθετεί ετικέτες (ή κάποια άλλα συστατικά) στις πέντε περιοχές BorderLayout.NORTH, BorderLayout.SOUTH, BorderLayout.EAST, BorderLayout.WEST και BorderLayout.CENTER. Η διάταξη αυτών των πέντε περιοχών έχει ως εξής:

BorderLayout.NORTH		
BorderLayout.WEST	BorderLayout.CENTER	BorderLayout.EAST
BorderLayout.SOUTH		

Πίνακας 3.19

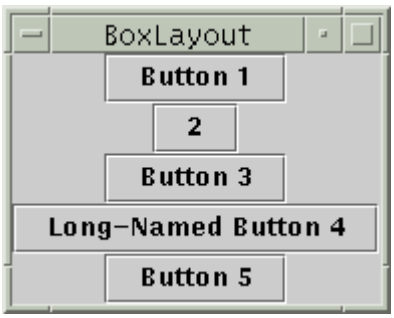
Οι παραπάνω είναι πέντε σταθερές που ορίζονται στη κλάση BorderLayout. Πρέπει να έχουμε υπόψη μας, ότι όταν χρησιμοποιούμε ένα διαχειριστή BorderLayout, δίδουμε την περιοχή ως δεύτερο όρισμα στη μέθοδο add όπως στην παρακάτω εντολή:

```
content.add(label1, BorderLayout.NORTH);
```

Δεν είναι απαραίτητο να χρησιμοποιήσουμε και τις πέντε περιοχές. Αν κάποιες περιοχές δεν χρησιμοποιηθούν, ο επιπλέον χώρος δίδεται στην περιοχή BorderLayout.CENTER.

Με την ίδια λογική και οι υπόλοιποι διαχειριστές διάταξης που φαίνονται παρακάτω, διαχειρίζονται τα συστατικά σε ένα παράθυρο, ο καθένας με διαφορετικό τρόπο:

BoxLayout:

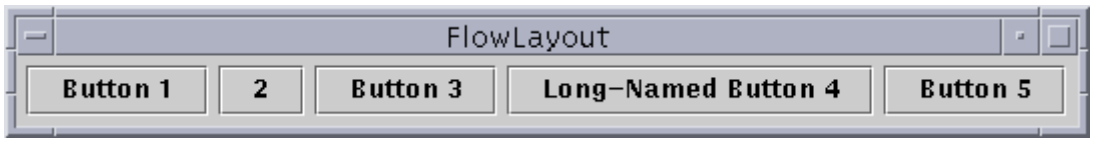


Εικόνα 3–9

CardLayout:

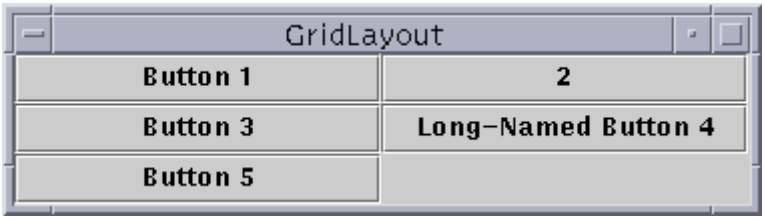
Εικόνα 3–10

FlowLayout:



Εικόνα 3–11

GridLayout:

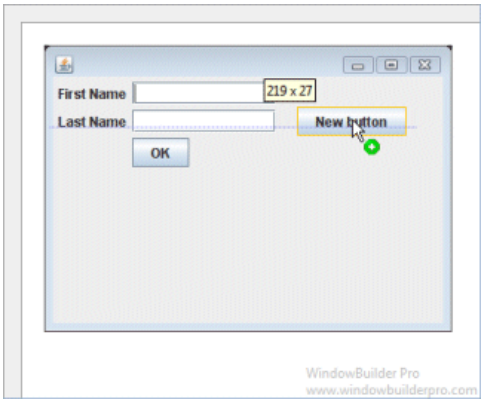


Εικόνα 3–12

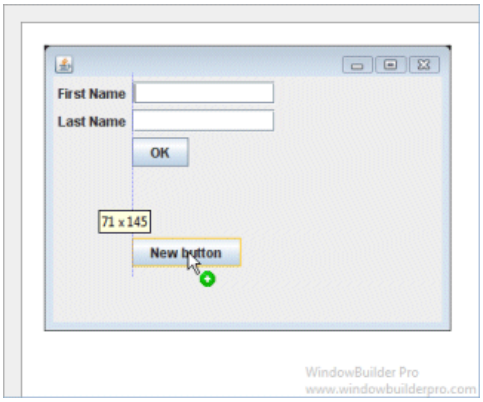
GridBagLayout:

Εικόνα 3–13

AbsoluteLayout:

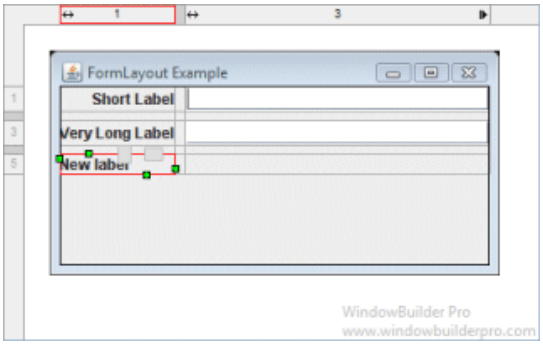


Εικόνα 3–14

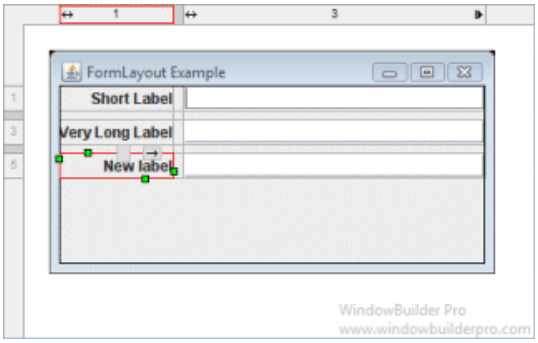


Εικόνα 3–15

JGoodies FormLayout:

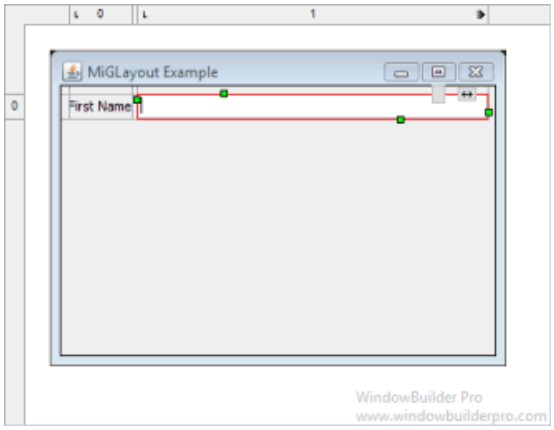


Εικόνα 3–16

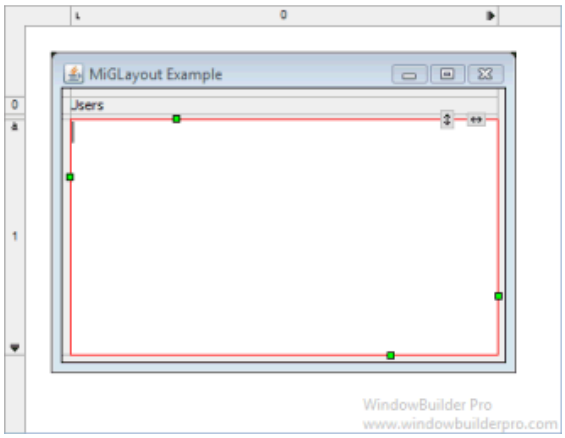


Εικόνα 3–17

MiGLayout:

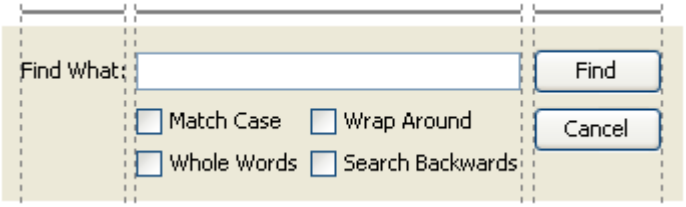


Εικόνα 3–18



Εικόνα 3–19

GroupLayout:



Εικόνα 3–20

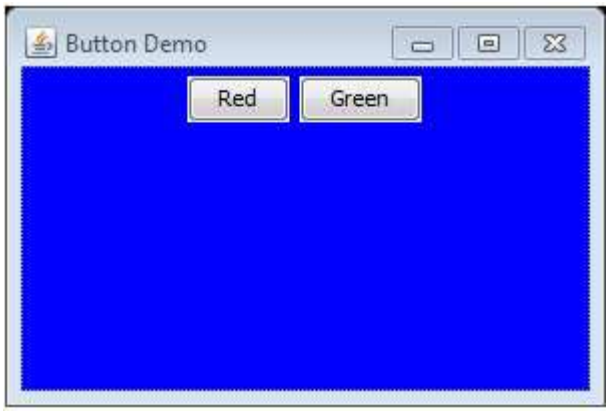
3.10.5 Κουμπιά και Ακροατές Λειτουργιών

Μέχρι τώρα τα GUI που είδαμε με την Swing δεν περιείχαν σχεδόν καμία λειτουργία. Απλά φαίνονταν και απεικόνιζαν κάποιο κείμενο. Η μοναδική λειτουργία που έκαναν ήταν αν εξαφανίζονται όταν ο χρήστης έκανε κλικ στο κουμπί τερματισμού παραθύρου. Παρακάτω θα παρουσιαστούν κάποια βασικά πράγματα για το πώς να σχεδιάσουμε GUI που πραγματοποιούν κάποιες πιο σύνθετες λειτουργίες.

Ένα κουμπί είναι απλά ένα στοιχείο μέσα σε ένα GUI και εκτελεί κάποια λειτουργία όταν ο χρήστης κάνει κλικ με το ποντίκι πάνω του. Τα κουμπιά δημιουργούνται με τρόπο παρόμοιο με αυτόν που δημιουργούνται και οι ετικέτες. Η προσθήκη κουμπιών σε ένα αντικείμενο JFrame γίνεται όπως και η προσθήκη ετικετών, αλλά υπάρχει ένα νέο χαρακτηριστικό με τα κουμπιά: Μπορούμε να συσχετίσουμε μία λειτουργία με το κουμπί ώστε όταν ο χρήστης κάνει κλικ με το ποντίκι πάνω στο κουμπί, το GUI να εκτελεί κάποια λειτουργία. Το παρακάτω παράδειγμα θα κάνει κάπως πιο ξεκάθαρα τα πράγματα:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class ButtonDemo extends JFrame implements ActionListener
{
	public static final int WIDTH = 300;
	public static final int HEIGHT = 200;
	public static void main(String[] args)
	{
		ButtonDemo buttonGui = new ButtonDemo();
		buttonGui.setVisible(true);
	}
	public ButtonDemo()
	{
		setSize(WIDTH, HEIGHT);
		addWindowListener(new WindowDestroyer());
		setTitle(“Button Demo”);
		Container contentPane = getContentPane();
		contentPane.setBackground(Color.BLUE);
		contentPane.setLayout(new FlowLayout());
		JButton stopButton = new JButton(“Red”);
		stopButton.addActionListener(this);
		contentPane.add(stopButton);
		JButton goButton = new JButton(“Green”);
		goButton.addActionListener(this);
		contentPane.add(goButton);
	}
	public void actionPerformed(ActionEvent e)
	{
		Container contentPane = getContentPane();
		if (e.getActionCommand().equals(“Red”))
		{
			contentPane.setBackground(Color.RED);
		}
		else
		{
			System.out.println(“Error in button interface.”);
		}
	}
}
```

Αυτό το πρόγραμμα θα δημιουργήσει το παρακάτω GUI:



Εικόνα 3–21

Το παραπάνω πρόγραμμα χρειάζεται πάνω από μία ενότητα για να αναλυθεί ικανοποιητικά. Σε αυτό το σημείο θα δείξουμε πως μπορούμε να προσθέσουμε κουμπιά σε ένα GUI. Όταν το παραπάνω πρόγραμμα τρέχει, εμφανίζεται το παράθυρο που φαίνεται αμέσως μετά τον κώδικα Εικόνα 3–21. Αν κάνουμε κλικ με το ποντίκι στο κουμπί “Red”, τότε το χρώμα του παραθύρου αλλάζει από μπλε σε κόκκινο. Αν κάνουμε κλικ στο κουμπί “Green”, τότε το χρώμα του παραθύρου αλλάζει σε πράσινο. Αυτές είναι όλες οι λειτουργίες που κάνει το πρόγραμμα (αλλά όπως μπορούμε να διαπιστώσουμε, σταδιακά μαθαίνουμε πώς να κατασκευάζουμε όλο και πιο σύνθετα παραθυρικά περιβάλλοντα). Για να τερματίσουμε το παράθυρο και να εξαφανιστεί, κάνουμε κλικ στο κουμπί τερματισμού παραθύρου.

Τα περισσότερα από αυτά που περιέχονται στο παραπάνω πρόγραμμα μας είναι ήδη γνωστά. Η κλάση ButtonDemo είναι απορρέουσα κλάση της κλάσης JFrame, επομένως είναι ένα παραθυρικό περιβάλλον παρόμοιο με αυτά που έχουμε ήδη δει σ’ αυτήν την ενότητα. Ένας ακροατής παραθύρου της κλάσης WindowDestroyer προστίθεται με τη μέθοδο addWindowListener, όπως και σε προηγούμενα παραδείγματα. Το μέγεθος τίθεται, και παίρνουμε την επιφάνεια, με τη μέθοδο getContentPane, επίσης όπως σε προηγούμενα παραδείγματα. Χρησιμοποιούμε ένα διαχειριστή διάταξης, όπως ακριβώς κάναμε και στο προηγούμενο παράδειγμα, αν κι αυτή τη φορά χρησιμοποιούμε την κλάση διαχειριστών FlowLayout.

Αυτό που είναι νέο στο παραπάνω πρόγραμμα είναι η χρήση αντικειμένων κουμπιών της κλάσης JButton και ένα νέο είδος κλάσης ακροατών. Ένα αντικείμενο κουμπί δημιουργείται με τον ίδιο τρόπο που δημιουργείται και οποιοδήποτε άλλο αντικείμενο, χρησιμοποιώντας όμως την κλάση JButton. Για παράδειγμα η παρακάτω γραμμή κώδικα από το παραπάνω πρόγραμμα δημιουργεί ένα κουμπί:

```
JButton stopButton = new JButton(“Red”);
```

Το όρισμα δομητή, που εδώ είναι το “Red”, είναι ένα αλφαριθμητικό που θα φαίνεται πάνω στο κουμπί, όταν αυτό θα εμφανίζεται. Αν κοιτάξει κανείς το GUI της παραπάνω εικόνας, θα δει ότι τα δύο κουμπιά έχουν ονόματα “Red” και “Green”. Το κουμπί προστίθεται στην επιφάνεια με την παρακάτω εντολή:

```
contentPane.add(stopButton);
```

Δεν υπάρχει δεύτερο όρισμα στη μέθοδο add επειδή χρησιμοποιούμε ένα διαχειριστή FlowLayout. Αν είχαμε χρησιμοποιήσει την κλάση BorderLayout, τότε θα είχαμε χρησιμοποιήσει και δεύτερο όρισμα, π.χ. το BorderLayout.NORTH.

Το πάτημα ενός κουμπιού με το ποντίκι (ή η ενεργοποίηση κάποιων άλλων στοιχείων μέσα σε ένα GUI) δημιουργεί ένα αντικείμενο που λέγεται γεγονός και στέλνει το αντικείμενο σε ένα άλλο αντικείμενο (ή αντικείμενα) το οποίο λέγεται ακροατής. Αυτή η διαδικασία λέγεται πυροδότηση ή πρόκληση του γεγονότος και στη συνέχεια ο ακροατής εκτελεί κάποια λειτουργία. Όταν λέμε ότι το γεγονός «αποστέλλεται» στο αντικείμενο ακροατής καλείται με το αντικείμενο γεγονός ως όρισμα. Αυτή η κλήση γίνεται αυτόματα. Ο ορισμός κλάσης ενός GUI Swing κανονικά δε θα περιέχει μία κλήση προς αυτήν τη μέθοδο. Θα πρέπει, ωστόσο, ο ορισμός αυτός να κάνει δύο πράγματα: Πρώτον, για κάθε κουμπί, πρέπει να καθορίζει ποιο αντικείμενο (ή ποια αντικείμενα) είναι ακροατής που θα αποκρίνεται στα γεγονότα που προκαλούνται από το κουμπί. Αυτό το στάδιο λέγεται **καταχώρηση (registering)** του ακροατή. Δεύτερον, πρέπει να ορίζει τη μέθοδο (ή τις μεθόδους) που θα καλούνται όταν το γεγονός αποστέλλεται στον ακροατή. Αυτές οι μέθοδοι θα οριστούν από εμάς, αλλά υπό φυσιολογικές συνθήκες δε θα χρειαστεί ποτέ να γράψουμε μία κλήση προς τις μεθόδους αυτές, επειδή οι κλήσεις αυτές θα γίνονται αυτόματα.

Η παρακάτω εντολή από το πρόγραμμα καταχωρεί την παράμετρο this ως ακροατή που θα λαμβάνει γεγονότα από το κουμπί που λέγεται stopButton:

```
stopButton.addActionListener(this);
```

Μία παρόμοια εντολή καταχωρεί και αυτή την παράμετρο this ως ακροατή που θα λαμβάνει γεγονότα από το κουμπί goButton. Επειδή το όρισμα είναι η παράμετρος this, η εντολή σημαίνει ότι η this (δηλαδή η ίδια η κλάση ButtonDemo) είναι η κλάση ακροατής. Σε αυτό το σημείο, αξίζει να θυμηθούμε πως μέσα στον ορισμό μία κλάσης ένα αντικείμενο της κλάσης αυτής λέγεται this. Επομένως, η κλάση ButtonDemo είναι η ίδια η κλάση ακροατής για τα κουμπιά που βρίσκονται μέσα στη κλάση ButtonDemo. Για να είμαστε πιο ακριβείς, κάθε αντικείμενο της κλάσης ButtonDemo είναι ο ακροατής για τα κουμπιά του αντικειμένου αυτού.

Διαφορετικά είδη στοιχείων απαιτούν διαφορετικά είδη κλάσεων ακροατή για το χειρισμό των γεγονότων που προκαλούν. Ένα κουμπί προκαλεί γεγονότα τα οποία λέγονται **γεγονότα λειτουργιών (action events)** και χειρίζονται από ακροατές που λέγονται **ακροατές λειτουργιών (action listeners)**.

Ένας ακροατής λειτουργιών είναι ένα αντικείμενο τύπου ActionListener. Η ActionListener δεν είναι μία κλάση, αλλά μία ιδιότητα την οποία μπορούμε να δώσουμε σε οποιαδήποτε κλάση ορίζουμε. Για να κάνουμε μία κλάση μέσα σε μία ActionListener χρειαζόμαστε δύο πράγματα:

1. Να προσθέσουμε τη φράση implements ActionListener στην αρχή του ορισμού κλάσης, συνήθως στο τέλος της πρώτης γραμμής.
2. Να ορίσουμε μία μέθοδο που να λέγεται actionPerformed.

Για να μπορεί να γίνει μία κλάση, λοιπόν, ακροατής λειτουργιών πρέπει, μεταξύ άλλων, να έχει μία μέθοδο που να λέγεται actionPerformed, η οποία θα έχει μία παράμετρο τύπου ActionEvent. Αυτή είναι η μοναδική μέθοδος που απαιτείται από τη διασύνδεση ActionListener.

Η σύνταξη της μεθόδου actionPerformed έχει ως εξής:

```
public void actionPerformed(ActionEvent e)
{
        Κώδικας_για_τις_Λειτουργίες_που_Εκτελούνται
}
```

Ο *Κώδικας_για_τις_Λειτουργίες_που_Εκτελούνται* είναι συνήθως μία εντολή διακλάδωσης που εξαρτάται από κάποια ιδιότητα e. Συχνά, η εντολή διακλάδωσης εξαρτάται από την κλήση e.getActionCommand(). Αν το e.getActionCommand() είναι ένα αλφαριθμητικό που λέγεται **εντολή λειτουργίας (action command).** Η εντολή λειτουργίας είναι το αλφαριθμητικό που είναι γραμμένο πάνω στο κουμπί, εκτός αν καθορίσουμε μία διαφορετική λειτουργία.

3.10.6 Περιέχουσες Κλάσεις

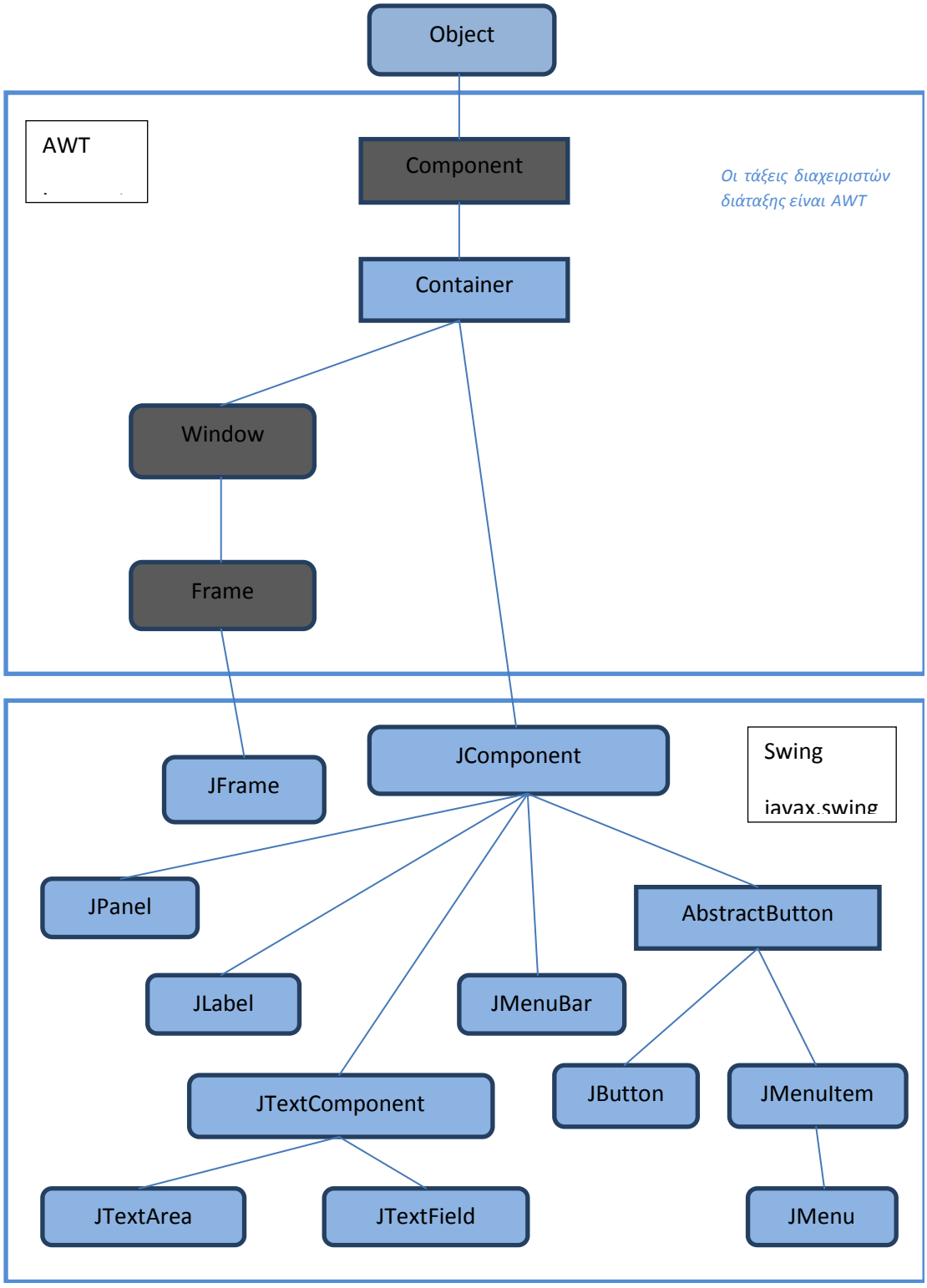
Όταν χρησιμοποιούμε την Swing για να φτιάξουμε ένα GUI όπως τα παράθυρα που έχουμε ορίσει, δημιουργούμε νέες κλάσεις από κάποιες ήδη υπάρχουσες κλάσεις. Υπάρχουν δύο βασικοί τρόποι για να κατασκευάσουμε νέες κλάσεις GUI από παλιότερες κλάσεις. Ο ένας τρόπος είναι η κληρονομικότητα. Για παράδειγμα, για να κατασκευάσουμε ένα παραθυρικό περιβάλλον, συνήθως χρησιμοποιούμε την κλάση JFrame της Swing και κάνουμε το παράθυρο μας μία απορρέουσα κλάση της κλάσης JFrame. Ο δεύτερος τρόπος είναι να χρησιμοποιήσουμε μία από τις κλάσεις της Swing ως **περιέχουσα (container)** κλάση και να προσθέσουμε στοιχεία στην κλάση αυτή.

3.10.6.1 Η Κλάση JPanel

Ένα GUI συνήθως είναι οργανωμένο με έναν ιεραρχικό τρόπο, με περιέχουσες κλάσεις μέσα σε άλλες περιέχουσες κλάσεις. Στην ενότητα αυτή, θα παρουσιάσουμε μία νέα κλάση που διευκολύνει αυτήν την οργάνωση. Η κλάση JPanel είναι μία πολύ απλά περιέχουσα κλάση που κάνει λίγα παραπάνω πράγματα από τα αντικείμενα ομάδες. Είναι μία από τις απλούστερες περιέχουσες κλάσεις, την οποία όμως θα χρησιμοποιήσουμε συχνά. Ένα αντικείμενο JPanel είναι ανάλογο με τα άγκιστρα που χρησιμοποιούμε για τη ομαδοποίηση πολλών απλούστερων εντολών Java σε μία μεγαλύτερη σύνθετη εντολή Java. Ομαδοποιεί μικρότερα αντικείμενα, όπως κουμπιά και ετικέτες, σε ένα μεγαλύτερο στοιχείο (το αντικείμενο JPanel) και μπορούμε μετά να βάλουμε αυτό το αντικείμενο JPanel στην επιφάνεια ενός αντικειμένου JFrame. Επομένως, μία από τις κύριες λειτουργίες των αντικειμένων JPanel είναι η υποδιαίρεση ενός αντικειμένου JFrame σε διάφορες περιοχές. Αυτά τα αντικείμενα JPanel συχνά λέγονται απλώς **πλαίσια (panels)**.

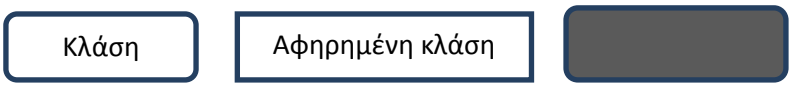
3.10.6.2 Η Κλάση Container

Υπάρχει μία προκαθορισμένη κλάση που λέγεται Container. Σε οποιαδήποτε κλάση απόγονο της κλάσης Container μπορούν να προστεθούν στοιχεία (ή για την ακρίβεια, στα αντικείμενα της κλάσης να προστεθούν στοιχεία). Η κλάση JFrame είναι μία κλάση απόγονος της κλάσης Container, επομένως οποιαδήποτε κλάση απόγονος της κλάσης JFrame μπορεί να χρησιμοποιηθεί ως περιέχουσα κλάση για ετικέτες, κουμπιά, πλαίσια ή άλλα στοιχεία. Στο παρακάτω σχήμα Εικόνα 3–22 φαίνεται ένα γενικό πλάνο για την ιεραρχία των κλάσεων:



Εικόνα 3–22

Αν υπάρχει γραμμή ανάμεσα σε δύο κλάσεις, τότε η χαμηλότερη κλάση είναι απορρέουσα κλάση της υψηλότερης κλάσης.



Η σκίαση υποδηλώνει πως η κλάση αυτή δεν χρησιμοποιείται στο παρόν κείμενο, αλλά συμπεριλαμβάνεται μόνο ως αναφορά.

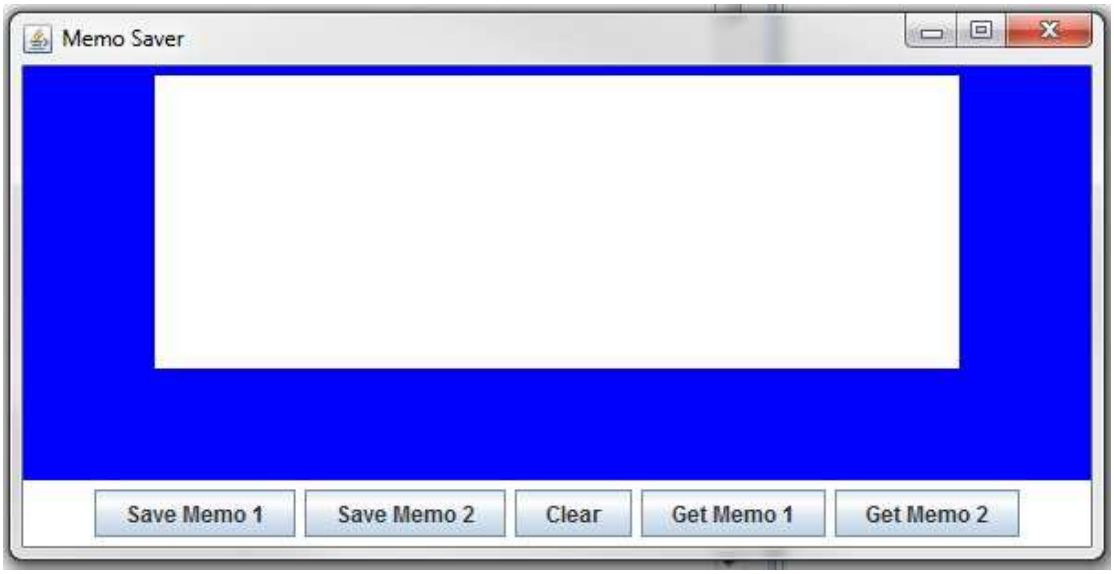
3.10.7 Είσοδος/Έξοδος κειμένου για γραφικά περιβάλλοντα (GUI)

Στο πρόγραμμα (GUI) που είδαμε στην ενότητα 3.10.4, μέσα στο παράθυρο υπάρχει κάποιο κείμενο. Ούτε ο χρήστης ούτε και το πρόγραμμα μπορούν να αλλάξουν το κείμενο. Παρακάτω θα δούμε ένα πρόγραμμα, που δημιουργεί ένα GUI με μία περιοχή κειμένου στην οποία ο χρήστης μπορεί να πληκτρολογήσει ότι κείμενο θέλει. Στην συνέχεια, μπορεί το κείμενο αυτό να αποθηκευτεί σαν ένα σημείωμα (memo) το οποίο μπορεί να ανακληθεί αργότερα. Σ' αυτό το απλό παράδειγμα, ο χρήστης επιτρέπεται να έχει μόνο δύο σημειώματα, αλλά είναι αρκετό για να δούμε πως δημιουργείται και πως χρησιμοποιείται μία περιοχή κειμένου. Ας δούμε τον κώδικα του προγράμματος:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class MemoSaver extends JFrame implements ActionListener
{
	public static final int WIDTH = 600;
	public static final int HEIGHT = 300;
	public static final int LINES = 10;
	public static final int CHAR_PER_LINE = 40;
	private JTextArea theText;
	private String memo1 = "No Memo 1.";
	private String memo2 = "No Memo 2.";
	public MemoSaver()
	{
		setSize(WIDTH, HEIGHT);
		addWindowListener(new WindowDestroyer());
		setTitle("Memo Saver");
		Container contentPane = getContentPane();
		contentPane.setLayout(new BorderLayout());
		JPanel buttonPanel = new JPanel();
		buttonPanel.setBackground(Color.WHITE);
		buttonPanel.setLayout(new FlowLayout());
		JButton memo1Button = new JButton("Save Memo 1");
		memo1Button.addActionListener(this);
		buttonPanel.add(memo1Button);
		JButton memo2Button = new JButton("Save Memo 2");
		memo2Button.addActionListener(this);
		buttonPanel.add(memo2Button);
		JButton clearButton = new JButton("Clear");
		clearButton.addActionListener(this);
		buttonPanel.add(clearButton);
		JButton get1Button = new JButton("Get Memo 1");
		get1Button.addActionListener(this);
		buttonPanel.add(get1Button);
		JButton get2Button = new JButton("Get Memo 2");
		get2Button.addActionListener(this);
		buttonPanel.add(get2Button);
		contentPane.add(buttonPanel, BorderLayout.SOUTH);
		JPanel textPanel = new JPanel();
```

```
            textPanel.setBackground(Color.BLUE);
            theText = new JTextArea(LINES, CHAR_PER_LINE);
            theText.setBackground(Color.WHITE);
            textPanel.add(theText);
            contentPane.add(textPanel, BorderLayout.CENTER);
      }
      public void actionPerformed(ActionEvent e)
      {
            String actionCommand = e.getActionCommand();
            if (actionCommand.equals("Save Memo 1"))
            {
                  memo1 = theText.getText();
            }
            else if (actionCommand.equals("Save Memo 2"))
            {
                  memo2 = theText.getText();
            }
            else if (actionCommand.equals("Clear"))
            {
                  theText.setText("");
            }
            else if (actionCommand.equals("Get Memo 1"))
            {
                  theText.setText(memo1);
            }
            else if (actionCommand.equals("Get Memo 2"))
            {
                  theText.setText(memo2);
            }
            else
            {
                  theText.setText("Error in memo interface");
            }
      }
      public static void main(String[] args)
      {
            MemoSaver guiMemo = new MemoSaver();
            guiMemo.setVisible(true);
      }
}
```

Η εκτέλεση του παραπάνω προγράμματος θα δημιουργήσει το παρακάτω GUI:



Εικόνα 3–23

Ας αναλύσουμε το παραπάνω πρόγραμμα. Η περιοχή στο κέντρο, με το άσπρο χρώμα, είναι ένα αντικείμενο της κλάσης JTextArea. Μπορούμε να γράψουμε οποιοδήποτε είδος κειμένου μέσα σε αυτό το αντικείμενο JTextArea. Αν κάνουμε κλικ στο κουμπί "Save Memo 1", τότε το κείμενο αποθηκεύεται σαν σημείωμα 1. Αν κάνουμε στο κουμπί "Save Memo 2", τότε το κείμενο αποθηκεύεται σαν σημείωμα 2. Οποιοδήποτε από τα δύο αυτά σημειώματα μπορεί να επανέλθει στην περιοχή κειμένου πατώντας ένα από τα κουμπιά "Get Memo 1" ή "Get Memo 2", αντίστοιχα. Η περιοχή κειμένου μπορεί να καθαριστεί πατώντας το κουμπί "Clear".

Τα κουμπιά που δημιουργούνται στο παραπάνω πρόγραμμα τοποθετούνται σε ένα πλαίσιο JPanel και στη συνέχεια το πλαίσιο αυτό τοποθετείται μέσα στο αντικείμενο JFrame. Η κλάση JTextArea διαμορφώνεται χρησιμοποιώντας τον παρακάτω κώδικα, ο οποίος βρίσκεται μέσα στον ορισμό δομητή:

```
JPanel textPanel = new JPanel();
textPanel.setBackground(Color.BLUE);
theText = new JTextArea(LINES, CHAR_PER_LINE);
theText.setBackground(Color.WHITE);
textPanel.add(theText);
contentPane.add(textPanel, BorderLayout.CENTER);
```

Το αντικείμενο theText είναι μέλος της κλάσης JTextArea. Ας προσέξουμε τα ορίσματα στο δομητή της κλάσης JTextArea. Τα ορίσματα LINES και CHAR_PER_LINE, τα οποία είναι ορισμένες σταθερές για το 10 και το 40, αντίστοιχα, σημαίνουν ότι η περιοχή κειμένου θα είναι 10 γραμμές από πάνω προς τα κάτω και κάθε γραμμή θα χωρά μέχρι 40 χαρακτήρες. Αν πληκτρολογήσουμε περισσότερο κείμενο από αυτό που χωρά σε μία περιοχή κειμένου με το μέγεθος που καθορίζεται από τα δύο ορίσματα του δομητή, τα αποτελέσματα μπορεί να είναι απρόβλεπτα. Αυτό μπορεί να αποφευχθεί με χρήση της μεθόδου setLineWrap για την οποία δεν θα αναφέρουμε κάτι σε αυτό το κείμενο).

Τα δύο σημειώματα αποθηκεύονται σε δύο μεταβλητές στιγμιοτύπου memo1 και memo2, που είναι και οι δύο τύπου String. Σε αυτό το σημείο να θυμηθούμε πως η μέθοδος getActionCommand επιστρέφει την ετικέτα του κουμπιού που έχει πατηθεί. Άρα ο μέθοδος actionPerformed δηλώνει π.χ. ότι αν γίνει κλικ στο κουμπί με την ετικέτα "Save Memo 1", τότε θα εκτελεστεί η παρακάτω εντολή:

```
memo1 = theText.getText();
```

Η μέθοδος getText() επιστρέφει το κείμενο που είναι γραμμένο μέσα στο αντικείμενο theText της κλάσης JTextArea. Με απλά λόγια, επιστρέφει ότι έχουν πληκτρολογήσει μέσα στην περιοχή κειμένου. Το ίδιο ισχύει, αντίστοιχα, και για το δεύτερο σημείωμα.

Η μέθοδος setText της κλάσης JTextArea αλλάζει το κείμενο που βρίσκεται μέσα στην περιοχή κειμένου με αυτό που της δίνεται ως όρισμα (δηλαδή, με το όρισμα της setText). Τα δύο εισαγωγικά που δεν έχουν τίποτα ανάμεσα τους και δίνονται ως όρισμα στη μέθοδο setText στη δεύτερη γραμμή του παραπάνω κώδικα υποδηλώνουν ότι πρόκειται για το κενό αλφαριθμητικό και κατά συνέπεια οδηγούν σε μία κενή περιοχή κειμένου.

Η κλάση JTextField, η οποία δεν χρησιμοποιείται στο παραπάνω πρόγραμμα, μοιάζει πολύ με την JTextArea με τη διαφορά ότι η πρώτη εμφανίζει μόνο μία γραμμή κειμένου. Είναι

χρήσιμη για περιβάλλοντα στο οποία ο χρήστης δίνει ως είσοδο λίγους μόνο χαρακτήρες, όπως ένα μεμονωμένο αριθμό, το όνομα ενός αρχείου ή το όνομα ενός ατόμου.

3.10.8 Είσοδος/Έξοδος Αριθμών με ένα GUI

Μπορούμε να χρησιμοποιήσουμε ένα GUI σχεδιασμένο με τη Swing για την είσοδο ή έξοδο αριθμών σε πεδία κειμένου. Το GUI μας επιτρέπει να μετατρέψουμε το κείμενο εισόδου σε αριθμούς ή αντίστοιχα να τοποθετήσουμε αριθμούς, αφού τους μετατρέψουμε σε αλφαριθμητικά, σε πεδία κειμένου.

3.10.8.1 Είσοδος Αριθμών με ένα GUI

Μπορούμε να χρησιμοποιήσουμε τη Swing για να σχεδιάσουμε ένα GUI ώστε η είσοδος να αποτελείται από έναν αριθμό που πληκτρολογείται μέσα σε ένα πεδίο κειμένου (ή σε μία περιοχή κειμένου). Η παρακάτω σύνταξη θα επιστρέψει τον αριθμό τύπου int που πληκτρολογήθηκε στο πεδίο κειμένου (υποθέτοντας ότι δεν έχει πληκτρολογηθεί τίποτα άλλο εκτός από τον αριθμό και κενά διαστήματα).

```
Σύνταξη:
Integer.parseInt(Όνομα_Του_Πεδίου_Κειμένου.getText().trim());
```

Η παρακάτω εντολή, αποθηκεύει μία τιμή τύπου int σε μία μεταβλητή n:

```
int n = Integer.parseInt(inputOutputField.getText().trim());
```

Μπορούμε να κάνουμε το ίδιο πράγμα για αριθμούς τύπου double, float και long. Απλά χρησιμοποιούμε την κλάση Double, Float ή Long, αντίστοιχα, στη θέση της κλάσης Integer και τη μέθοδο parseDouble, parseFloat ή parseLong, αντίστοιχα, στη θέση της μεθόδου parseInt.

3.10.8.2 Έξοδος Αριθμών με ένα GUI

Μπορούμε να χρησιμοποιήσουμε τη Swing για να σχεδιάσουμε ένα GUI ώστε η έξοδος να αποτελείται από έναν αριθμό που πληκτρολογείται μέσα σε ένα πεδίο κειμένου (ή σε μία περιοχή κειμένου). Η παρακάτω σύνταξη θα πάρει έναν αριθμό που βρίσκεται μέσα σε μία μεταβλητή και θα εμφανίσει τον αριθμό μέσα στο πεδίο κειμένου. Οι λεπτομέρειες είναι ίδιες είτε χρησιμοποιούμε ένα αντικείμενο JTextArea είτε ένα αντικείμενο JTextField:

```
Σύνταξη:
Όνομα_Του_Πεδίου_Κειμένου.setText(Περικλείουσα_Κλάση.toString(Μεταβλητή));
```

Παράδειγμα:

```
`
```

Μπορούμε να κάνουμε το ίδιο για αριθμούς τύπου float και long. Απλά χρησιμοποιούμε την κλάση Float ή Long στη θέση της κλάσης Integer ή Double.

4 Επίλυση Δικτυωμάτων

4.1 Γενικά

Για την σύνταξη της παρούσαν ενότητας αντλήθηκαν δεδομένα από τις ακόλουθες πηγές:

- Introduction to Finite Elements in Engineering – Tirupathi R. Chandrupatla , Ashok D. Belegundu
- http://www.colorado.edu/engineering/cas/courses.d/IFEM.d/
- Structural Analysis IV - Dr. C. Caprani

Όλα τα παρακάτω δεδομένα έχουν παρθεί κυρίως από τις παραπάνω πηγές και είναι μεταφρασμένα από εμένα.

4.2 Μέθοδος των Μετατοπίσεων

4.2.1 Εισαγωγή

Ο πλέον διαδεδομένος τρόπος διατύπωσης της μεθόδου των πεπερασμένων στοιχείων για γραμμικά στατικά προβλήματα βασίζεται στη μέθοδο των μετατοπίσεων. Άλλοι τρόποι κάνουν χρήση της ισορροπίας δυνάμεων, ή άλλων υβριδικών ή και μικτών μεθόδων. Τα βασικά στάδια της μεθόδου είναι:

1. Η προσομοίωση (διακριτοποίηση) της κατασκευής με ένα σύνολο στοιχείων που συνδέονται σε συνοριακούς κόμβους.

2. Ο προσδιορισμός των γενικευμένων (άγνωστων) μετατοπίσεων που θα καθορίσουν πλήρως την απόκριση της κατασκευής.

3. Η διατύπωση των εξισώσεων ισορροπίας που αντιστοιχούν στις άγνωστες κομβικές μετατοπίσεις και η επίλυσή τους.

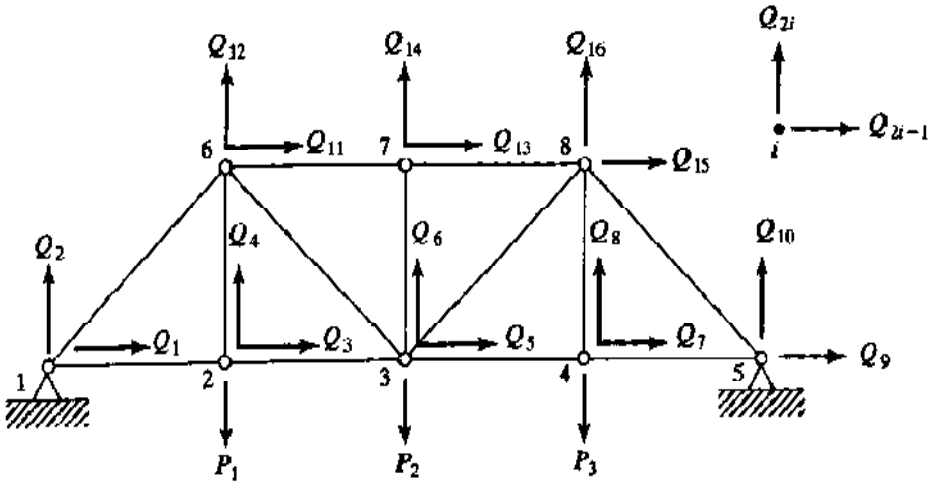
4. Ο υπολογισμός των εσωτερικών κατανομών των τάσεων των στοιχείων, για δεδομένες μετατοπίσεις στους κόμβους.

5. Η ερμηνεία των αποτελεσμάτων της ανάλυσης, (μετατοπίσεις και τάσεις), με βάση τις δεδομένες παραδοχές του προβλήματος.

Κάθε φοιτητής μηχανικής, κατά τη διάρκεια των σπουδών του, παρακολούθησε το μάθημα της στατικής ανάλυσης. Ανέλυσε δικτυώματα χρησιμοποιώντας την μέθοδο των κόμβων ή την μέθοδο των τομών. Αυτές οι μέθοδοι, ενώ απεικονίζουν τις βασικές αρχές της στατικής, γίνεται κουραστικό όταν εφαρμόζονται σε μεγάλης κλίμακας υπερστατικούς φορείς. Ακόμη, οι μετατοπίσεις των κόμβων δεν είναι εύκολα υπολογίσιμες. Η μέθοδος των πεπερασμένων στοιχείων από την άλλη είναι εφαρμόσιμη σε ισοστατικές ή αόριστες δομές ιδίως.

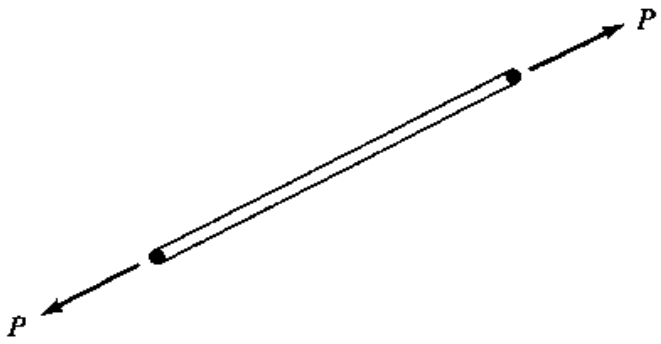
4.2.2 Ανάλυση της μεθόδου

Στην Εικόνα 4–1 είναι ένα δισδιάστατο δικτύωμα:



Εικόνα 4–1 Δισδιάστατο δικτύωμα

Το δικτύωμα έχει δύο στηρίξεις. Μια δομή δικτυώματος απαρτίζεται αποκλειστικά από μέλη των δύο δυνάμεων. Δηλαδή, κάθε στοιχείο του δικτυώματος βρίσκεται σε άμεσο εφελκυσμό ή θλίψη (Εικόνα 4–2).

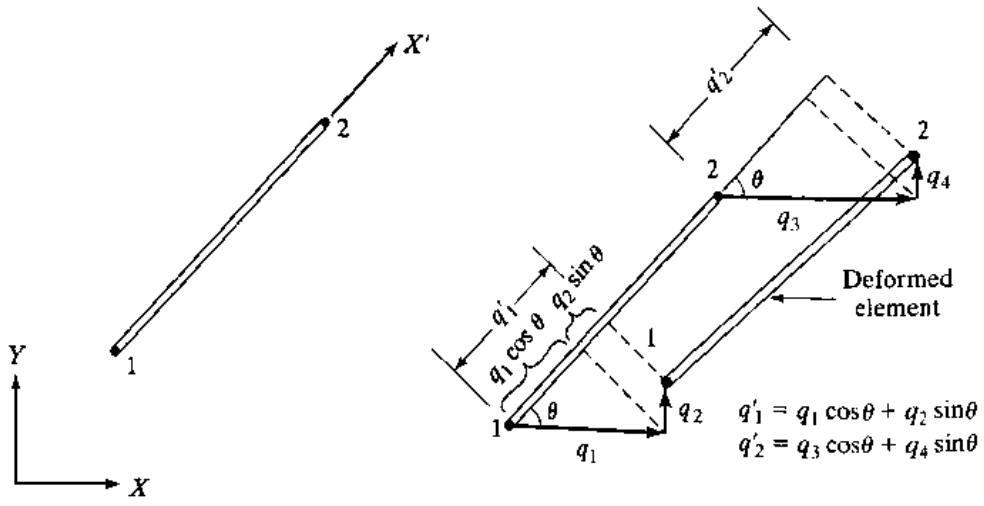


Εικόνα 4–2 Μέλος δύο δυνάμεων

Σε ένα δικτύωμα, απαιτείται αυτά όλα τα φορτία και οι αντιδράσεις να εφαρμόζονται στις αρθρώσεις και όλα τα μέλη να συνδέονται μεταξύ τους στο τέλος με αυτές.

4.2.3 Τοπικό και Γενικό σύστημα αναφοράς

Η κύρια διαφορά μεταξύ των μονοδιάστατων στοιχείων (Εικόνα 4–3α) και των στοιχείων των δικτυωμάτων (Εικόνα 4–3β) είναι ότι τα στοιχεία ενός δικτυώματος έχουν διάφορους προσανατολισμούς. Για να εξηγηθούν αυτές οι διαφορετικές κατευθύνσεις, εισάγεται το τοπικό και το γενικό σύστημα αναφοράς (συντεταγμένων) ως ακολούθως:



(α) (β)

Εικόνα 4–3 Δικτύωμα δύο διαστάσεων (α) τοπικό σύστημα αναφοράς και (β) γενικό σύστημα αναφοράς.

Στην παραπάνω εικόνα, φαίνεται ένα μέλος του δικτυώματος σε τοπικό σύστημα αναφοράς και σε γενικό σύστημα αναφοράς. Στο τοπικό σύστημα αναφοράς οι δύο κόμβοι του μέλους αριθμούνται με τους αριθμούς 1 και 2. Το τοπικό σύστημα αναφοράς ορίζεται από τον άξονα x' καθώς αυτός έχει διεύθυνση από το κόμβο 1 στον κόμβο 2. Όλοι οι συμβολισμοί που αναφέρονται στο τοπικό σύστημα αναφοράς γράφονται τονισμένοι (π.χ. q'). Το γενικό σύστημα αναφοράς είναι σταθερό και ανεξάρτητο από τον προσανατολισμό των μελών του δικτυώματος. Να σημειωθεί εδώ, πως στο σύστημα x,y,z ο άξονας z έχει διεύθυνση κάθετη στο χαρτί και κατεύθυνση προς τα έξω όπως κοιτάζουμε αυτό.

Στο γενικό σύστημα αναφοράς, κάθε κόμβος έχει δύο βαθμούς ελευθερίας (dofs). Αν παρατηρήσει κανείς την Εικόνα 4–1, θα προσέξει πως σε κάθε κόμβο ορίζονται δύο βαθμοί ελευθερίας Q . Ο καθένας από αυτούς έχει έναν δείκτη. Ο συστηματικός υπολογισμός αυτών των δεικτών καθορίζεται ως εξής: Ας πούμε ότι γενικός αριθμός κάθε κόμβου είναι το γράμμα *j* . Οι δείκτες των βαθμών ελευθερίας του κάθε κόμβου, έχουν τιμή ίση με (*2j – 1*) για τον οριζόντιο και (*2j*) για τον κατακόρυφο. Κατ' επέκταση οι μετακινήσεις κατά το γενικό σύστημα αναφοράς για τον κόμβο *j* , συμβολίζονται με Q_{2j-1} και Q_{2j} .

Οπότε, q'_1 και q'_2 είναι οι μετακινήσεις των κόμβων 1 και 2 , αντίστοιχα, με βάση το τοπικό σύστημα αναφοράς. Έτσι, το διάνυσμα της μετατόπισης του μέλους στο τοπικό σύστημα αναφοράς συμβολίζεται ως εξής:

$$q' = [q'_1 , q'_2]^T \qquad (2.1)$$

Το διάνυσμα της μετατόπισης του μέλους στο γενικό σύστημα αναφοράς είναι ένα διάνυσμα (4 X 1) και συμβολίζεται ως εξής:

$$q = [q_1, q_2 , q_3 , q_4]^T \qquad (2.2)$$

Η σχέση μεταξύ του q' και q εξελίσσεται ως εξής: Στην Εικόνα 4–3b , φαίνεται ότι το q'_1 ισούται με το άθροισμα των προβολών των q_1 και q_2 στον άξονα x'. Έτσι,

$$q'_1 = q_1 \cos\theta + q_2 \sin\theta \qquad (2.3a)$$

Ομοίως,

$$q'_2 = q_3 \cos\theta + q_4 \sin\theta \qquad (2.3b)$$

Σε αυτό το σημείο, οι συντελεστές διεύθυνσης l και m περιγράφονται από τις σχέσεις

$l = \cos\theta$ και $m = \cos\varphi$ (= $\sin\theta$). Αυτά τα συνημίτονα κατεύθυνσης, είναι τα συνημίτονα των γωνιών που σχηματίζει ο άξονας x' στο τοπικό σύστημα αναφοράς με τους άξονες x,y αντίστοιχα, του γενικού συστήματος αναφοράς. Οι εξισώσεις 2.3a και 2.3b μπορούν να γραφτούν σε μητρωϊκή μορφή ως εξής:

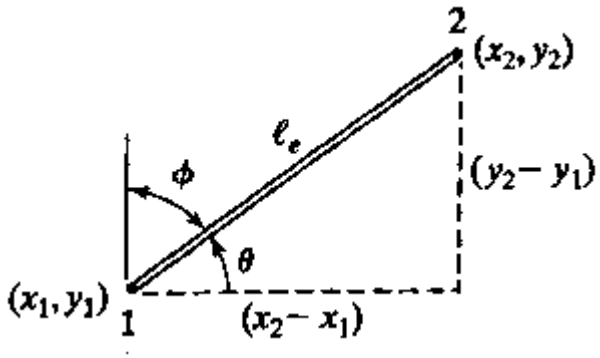
$$q' = Lq \qquad (2.4)$$

Όταν το μητρώο μετατόπισης L δίδεται από:

$$L = \begin{bmatrix} l & m & 0 & 0 \\ 0 & 0 & l & m \end{bmatrix} \qquad (2.5)$$

4.2.4 Υπολογισμός των l και m

Θα δοθεί ένας απλός τρόπος υπολογισμού των συνημίτονων κατεύθυνσης l και m από τις συντεταγμένες των κόμβων. Παρατηρώντας την Εικόνα 4–4, διακρίνεται πως οι συντεταγμένες (x_1 , y_1) και (x_2 , y_2) είναι οι θέσεις των κόμβων 1 και 2 , αντίστοιχα.



Εικόνα 4–4

Έτσι έχουμε:

$$l = \cos\theta = \frac{x_2 - x_1}{l_e} \qquad (2.6a)$$

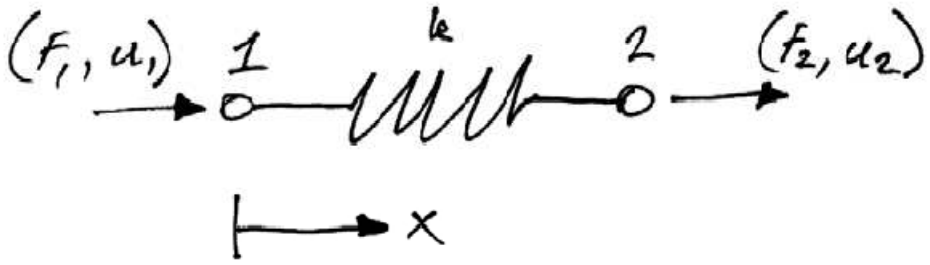
και $$m = \cos\varphi = \frac{y_2 - y_1}{l_e} \qquad (2.6b)$$

όπου το μήκος l_e ισούται με:

$$l_e = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \qquad (2.7)$$

4.2.5 Μητρώο δυσκαμψίας μονοδιάστατων στοιχείων

Για να γίνει καλύτερη η κατανόηση της έννοιας του μητρώου δυσκαμψίας, θα γίνει όσο το δυνατόν μια απλή, βασική προσέγγιση. Για να επιτευχθεί αυτό, στην παρακάτω Εικόνα 4–5 έχει αντικατασταθεί μια ράβδος, με ένα ελατήριο όπου αυτό συνδέεται στα άκρα του με δύο κόμβους, τον κόμβο 1 και τον κόμβο 2.



Εικόνα 4–5: σύστημα κόμβων ελατηρίου – Απλό στοιχείο

Θεωρείται ότι μπορούν να μετακινηθούν μόνο ως προς τον άξονα x. Έτσι ο κάθε κόμβος έχει μόνο έναν βαθμό ελευθερίας (dof). Ο κάθε κόμβος δηλαδή, μπορεί να έχει κάποια μετατόπιση *u* και επίσης σε κάθε κόμβο μπορεί να ασκείται μια δύναμη *F* από το ελατήριο, πάντα κατά την διεύθυνση του x άξονα. Στο σχήμα τα βέλη δείχνουν τις θετικές φορές των διανυσμάτων της δύναμης και της μετατόπισης κατά τον άξονα x. Η μητρωϊκή ανάλυση απαιτεί αυστηρό καθορισμό αυτών των συμβάσεων φοράς.

Χρησιμοποιώντας την βασική σχέση που λέει πως η δύναμη που ασκείται σε κάθε κόμβο ισούται με το γινόμενο της σταθεράς του ελατηρίου επί την μετατόπιση του κόμβου, μπορεί να υπολογισθεί η δύναμη στον κόμβο 1 ως:

$$F_1 = k(\mu\varepsilon\tau\alpha\tau\acute{o}\pi\iota\sigma\eta\ \tau o\upsilon\ \kappa\acute{o}\mu\beta o\upsilon\ 1)$$

Έτσι:

$$F_1 = k(u_1 - u_2) = ku_1 - ku_2 \qquad (2.5.1)$$

Ομοίως για τον κόμβο 2 :

$$F_2 = k(u_2 - u_1) = -ku_1 + ku_2 \qquad (2.5.2)$$

Μπορούν να γραφτούν οι εξισώσεις 2.5.1 και 2.5.2 σε μητρωϊκή μορφή και να υπολογιστεί το μητρώο δυσκαμψίας για τον έναν βαθμό ελευθερίας του μέλους:

$$\begin{Bmatrix} F_1 \\ F_2 \end{Bmatrix} = \begin{bmatrix} k & -k \\ -k & k \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} \qquad (2.5.3)$$

Εφαρμόζοντας μητρωϊκό συμβολισμό, προκύπτει:

$$\{F^e\} = [k]\{u^e\} \qquad (2.5.4)$$

Όπου:

{F^e} το διάνυσμα της δύναμης του μέλους.

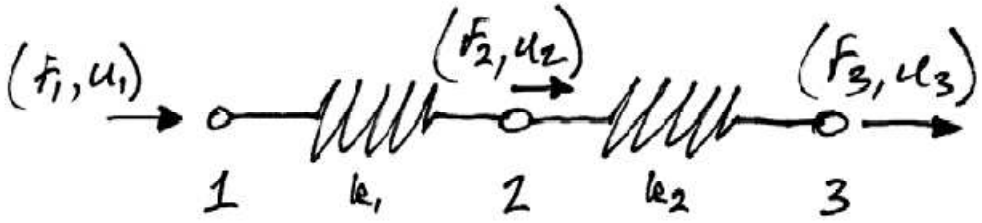
[k] το μητρώο δυσκαμψίας.

{u^e} το διάνυσμα της μετατόπισης.

Πρέπει να είναι σαφές ότι το μητρώο δυσκαμψίας είναι ζωτικής σημασίας – συνδέει τις δυνάμεις που ασκούνται στους κόμβους με τις κομβικές μετατοπίσεις και δείχνει το πώς συμπεριφέρεται το μέλος (στοιχείο) όταν βρίσκεται υπό φορτίο-ία. Επίσης, ο υπολογισμός του μητρώου δυσκαμψίας για διαφορετικούς τύπους ράβδων είναι ίσως το πιο δύσκολο μέρος στη μέθοδο δυσκαμψίας, αλλά αυτό δεν είναι σημαντικό μειονέκτημα, μιας και χρησιμοποιούνται λίγα είδη ράβδων.

Οι πραγματικές κατασκευές αποτελούνται από συνέχειες στοιχείων που ενώνονται μεταξύ τους. Εδώ θα εξηγηθεί το πώς συνδέεται το επιμέρους μητρώο δυσκαμψίας κάθε στοιχείου για να σχηματιστεί το μητρώο δυσκαμψίας όλης της κατασκευής.

Στην παρακάτω Εικόνα 4–6 φαίνεται μια απλή ακολουθία στοιχείων:



Εικόνα 4–6: Συνδυασμός δύο απλών στοιχείων.

Αποτελείται από 3 κόμβους και 2 στοιχεία, που θα ληφθούν ως ελατήρια όπως το προηγούμενο παράδειγμα. Να σημειωθεί ότι αυτά τα ξεχωριστά μέλη, έχουν διαφορετική δυσκαμψία k_1 και k_2 όπως φαίνεται στο σχήμα.

Η σχέση των δυνάμεων με τις κομβικές μετατοπίσεις για κάθε μέλος μπορεί να γραφτεί ως εξής:

Για το πρώτο στοιχείο:

$$\begin{Bmatrix} F_1 \\ F_2 \end{Bmatrix} = \begin{bmatrix} k_1 & -k_1 \\ -k_1 & k_1 \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} \qquad (2.5.5)$$

Ομοίως για το δεύτερο:

$$\begin{Bmatrix} F_2 \\ F_3 \end{Bmatrix} = \begin{bmatrix} k_2 & -k_2 \\ -k_2 & k_2 \end{bmatrix} \begin{Bmatrix} u_2 \\ u_3 \end{Bmatrix} \quad (2.5.6)$$

Αν προσαρμοστούν αυτές οι σχέσεις στο σύνολο της κατασκευής, προκύπτει:

Για το πρώτο στοιχείο:

$$\begin{Bmatrix} F_1 \\ F_2 \\ F_3 \end{Bmatrix} = \begin{bmatrix} k_1 & -k_1 & 0 \\ -k_1 & k_1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \end{Bmatrix} \quad (2.5.7)$$

Ομοίως για το δεύτερο:

$$\begin{Bmatrix} F_1 \\ F_2 \\ F_3 \end{Bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & k_2 & -k_2 \\ 0 & -k_2 & k_2 \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \end{Bmatrix} \quad (2.5.8)$$

Οι παραπάνω εξισώσεις 2.5.7 και 2.5.8 θα μπορούσαν να γραφτούν ως εξής:

$$\begin{Bmatrix} F_1 \\ F_2 \\ F_3 \end{Bmatrix} = \begin{bmatrix} k_1 & -k_1 & 0 \\ -k_1 & k_1 + k_2 & -k_2 \\ 0 & -k_2 & k_2 \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \end{Bmatrix} \quad (2.5.9)$$

Και έτσι, από την εξίσωση αυτή περιγράφεται η σχέση των δυνάμεων και των μετατοπίσεων για όλη την κατασκευή. Επίσης, η εξίσωση δείχνει την συνολική δυσκαμψία σε κάθε κόμβο. Στον κόμβο 2 για παράδειγμα όπου συνδέονται τα δύο μέλη η δυσκαμψία είναι $k_1 + k_2$.

Η εξίσωση 2.5.9 μπορεί να ξαναγραφτεί ως εξής:

$$\{F\} = [K]\{u\} \quad (2.5.10)$$

Όπου:

$\{F\}$ το τελικό διάνυσμα των δυνάμεων της κατασκευής.

$[K]$ το συνολικό μητρώο δυσκαμψίας της κατασκευής.

$\{u\}$ το τελικό διάνυσμα μετατοπίσεων της κατασκευής.

Η ακαμψία *k* ορίζεται από τον λόγο του γινομένου της επιφάνειας διατομής *A* του στοιχείου και του μέτρο ελαστικότητας Ε του υλικού του στοιχείου, δια το μήκος *L* του στοιχείου. Δηλαδή:

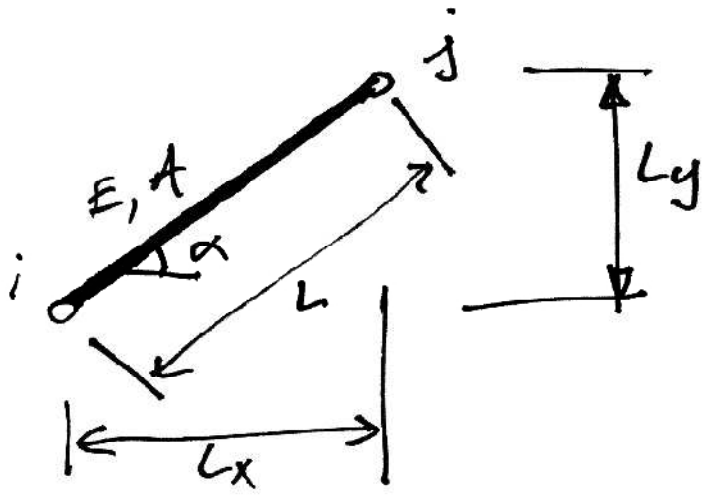
$$k_e = \frac{A_e E_e}{L_e} \qquad (2.5.11)$$

Με βάση την παραπάνω σχέση, η εξίσωση 2.5.5 θα μπορούσε να ξαναγραφτεί ως εξής:

$$\begin{Bmatrix} F_1 \\ F_2 \end{Bmatrix} = \frac{A_1 E_1}{L_1} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix}$$

4.2.6 Μητρώο δυσκαμψίας μελών δικτυώματος

Σε αυτό το σημείο θα γίνει μια σημαντική παρατήρηση: *Κάθε μέλος ενός δικτυώματος είναι ένα μονοδιάστατο μέλος καθώς παρατηρείται στο τοπικό σύστημα συντεταγμένων.* Όπως φαίνεται στην επόμενη Εικόνα 4–7:



Εικόνα 4–7: Στοιχείο στο σύστημα συντεταγμένος

Αυτή η παρατήρηση επιτρέπει να χρησιμοποιηθεί ότι παρουσιάστηκε στο κεφάλαιο 4.2.5. για τα μονοδιάστατα προβλήματα.

Με βάση την προηγούμενη ενότητα 4.2.5. όπου εξετάστηκαν τα μονοδιάστατα στοιχεία, για την ράβδο του παραπάνω σχήματος (Εικόνα 4–7: Στοιχείο στο σύστημα συντεταγμένος) το μητρώο δυσκαμψίας στο τοπικό σύστημα συντεταγμένων ισούται με:

$$[k] = \begin{bmatrix} k & -k \\ -k & k \end{bmatrix} = k \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

Επομένως, με βάση την εξίσωση 2.5.11 :

$$k = \frac{E_e A_e}{l_e}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \qquad (2.8)$$

όπου Ae το εμβαδόν διατομής του μέλους και Ee το μέτρο ελαστικότητας του υλικού.

Αυτό ισχύει για το τοπικό σύστημα αναφοράς και για την περίπτωση που το στοιχείο μπορεί να μετακινηθεί μόνο στο άξονα x'. Αν διαμορφωθεί αυτή η σχέση κατάλληλα, ούτως ώστε να φαίνονται και οι τιμές του μητρώου για τις μετακινήσεις στον τοπικό y' άξονα , στον οποίο είναι μηδέν μιας και το στοιχείο δεν φορτίζεται κατά τον άξονα y' άρα δεν μπορεί να μετακινηθεί κατά αυτόν θα γραφόταν:

$$[k] = \frac{E_e A_e}{l_e}\begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}\begin{matrix} \leftarrow x_i \\ \leftarrow y_i \\ \leftarrow x_j \\ \leftarrow y_j \end{matrix} \qquad (2.8.1)$$

Το ζήτημα τώρα, είναι το πώς θα εξελιχθεί αυτό το μητρώο ώστε να ισχύει για το γενικό σύστημα αναφοράς. Αυτό μπορεί να γίνει εφικτό εξετάζοντας την ενέργεια παραμορφώσεως στο μέλος. Συγκεκριμένα, η ενέργεια παραμορφώσεως του μέλους στο τοπικό σύστημα αναφοράς δίδεται από τη σχέση:

$$U_e = \frac{1}{2} q'^T k' q' \qquad (2.9)$$

Αντικαθιστώντας για $q' = Lq$ καθώς και $q'^T = L^T q^T$ στην Εξίσωση 2.9 , προκύπτει:

$$U_e = \frac{1}{2} q^T [L^T k' L] q \qquad (2.10)$$

Έτσι η ενέργεια παραμόρφωσης στο γενικό σύστημα συντεταγμένων μπορεί να γραφτεί τώρα:

$$U_e = \frac{1}{2} q^T k q \qquad (2.11)$$

όπου το k είναι το μητρώο δυσκαμψίας του μέλους στο γενικό σύστημα αναφοράς. Από την προηγούμενη εξίσωση, μπορεί να οριστεί η δυσκαμψία του μέλους στο γενικό σύστημα αναφοράς ως:

$$k = L^T k' L \qquad (2.12)$$

Αντικαθιστώντας για *L* από την Εξίσωση 2.5 και το k' από την Εξίσωση 2.8, προκύπτει:

$$k = \begin{bmatrix} l & 0 \\ m & 0 \\ 0 & l \\ 0 & m \end{bmatrix} \frac{E_e A_e}{l_e} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} l & m & 0 & 0 \\ 0 & 0 & l & m \end{bmatrix}$$

Ο πολλαπλασιασμός των παραπάνω πινάκων θα δώσει:

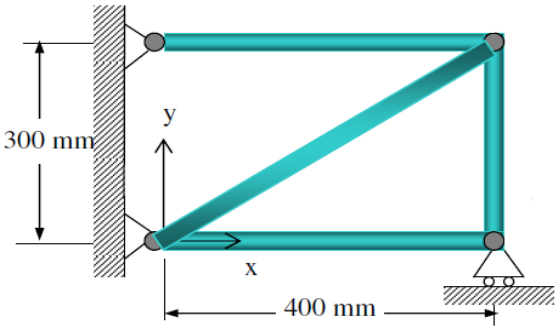
$$k = \frac{E_e A_e}{l_e} \begin{bmatrix} l^2 & lm & -l^2 & -lm \\ lm & m^2 & -lm & -m^2 \\ -l^2 & -lm & l^2 & lm \\ -lm & -m^2 & lm & m^2 \end{bmatrix} \qquad (2.13)$$

Έτσι, σχηματίζονται όλα τα μητρώα των μελών του δικτυώματος και με βάση αυτά διαμορφώνεται στο τέλος το τελικό μητρώο δυσκαμψίας για όλο το δικτύωμα. Στο Παράδειγμα 2.1 θα γίνει καλύτερη κατανόηση του τρόπου διαμόρφωσης του μητρώου δυσκαμψίας κάθε ράβδου στο γενικό σύστημα αναφοράς.

Παράδειγμα 2.1 Να εξεταστεί το παρακάτω δικτύωμα Εικόνα 4–8 το οποίο αποτελείται από τέσσερις ράβδους και τέσσερις κόμβους όπως φαίνεται στο Σχήμα 2.1 .Δίνονται ότι

E = 70 Gpa και A_e = 200 mm^2 και είναι ίδια για όλα τα μέλη. Να βρεθεί:

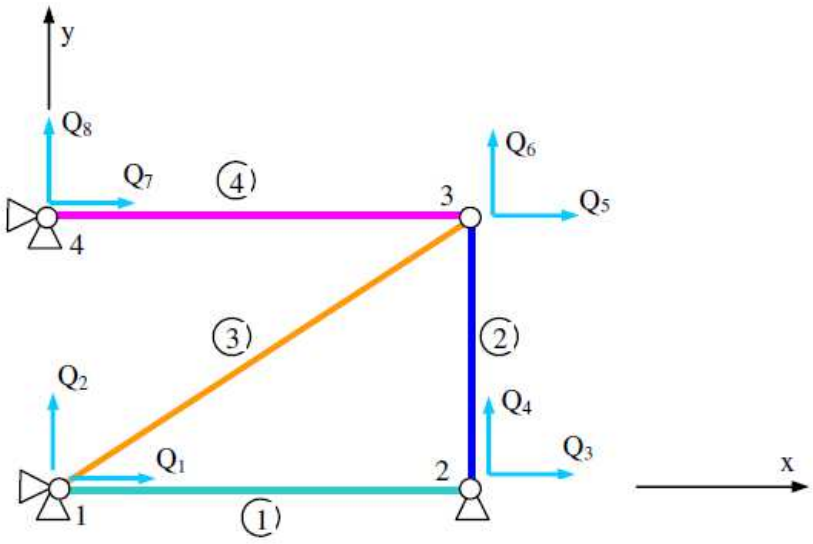
- Το μητρώο δυσκαμψίας για κάθε ράβδο στο γενικό σύστημα αναφοράς.



Εικόνα 4–8: παράδειγμα 1

Λύση:

το διάγραμμα ελευθέρου σώματος παρουσιάζεται στην Εικόνα 4–9:



Εικόνα 4–9: βαθμοι΄ελευθερίας παραδείγματος 1

Συνιστάται η δημιουργία κάποιων πινάκων με τα απαραίτητα δεδομένα. Όπως ο πίνακας με τις συντεταγμένες των κόμβων και ο πίνακας με τον αριθμό των συνδεδεμένων κόμβων σε κάθε ράβδο:

Κόμβος	x (mm)	Y (mm)
1	0	0
2	400	0
3	400	300
4	0	300

Ράβδος	1(αρχή)	2 (τέλος)
1	1	2
2	3	2
3	1	3
4	4	3

Παρατηρείται στον δεύτερο πίνακα, πως θα μπορούσε για κάποια ράβδο να οριστεί ανάποδα ο αρχικός και ο τελικός κόμβος, όπως π.χ. στην ράβδο 2 , θα μπορούσε κάλλιστα να οριστεί σαν αρχικός ο κόμβος 2 και σαν τελικός ο κόμβος 3. Αυτό δεν επηρεάζει στο αποτέλεσμα του τελικού –γενικού- μητρώου.

Με βάση τους παραπάνω πίνακες μπορούν να υπολογιστούν τα συνημίτονα κατεύθυνσης l και m για κάθε ράβδο. Με βάση τις εξισώσεις 2.6a , 2.6b και 2.7 προκύπτει ο παρακάτω πίνακας με τα αποτελέσματα:

Ράβδος	L_e	l	m
1	400	1.0	0.0
2	300	0.0	-1.0
3	500	0.8	0.6
4	400	1.0	0.0

με βάση την εξίσωση 2.13 υπολογίζονται τα μητρώα κάθε ράβδου:

$$[k^1] = \frac{7\times10^4\times200}{400}\begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad [k^2] = \frac{7\times10^4\times200}{300}\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}$$

$$[k^3] = \frac{7\times10^4\times200}{500}\begin{bmatrix} 0.64 & 0.48 & -0.64 & -0.48 \\ 0.48 & 0.36 & -0.48 & -0.36 \\ -0.64 & -0.48 & 0.64 & 0.48 \\ -0.48 & -0.36 & 0.48 & 0.36 \end{bmatrix}$$

$$[k^4] = \frac{7\times10^4\times200}{400}\begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

4.2.7 Γενικό μητρώο δυσκαμψίας δικτυώματος

Σε αυτήν την ενότητα θα γίνει η περιγραφή για το πώς φτιάχνεται το γενικό μητρώο δυσκαμψίας ενός δικτυώματος. Οι εικόνες στην παρούσα ενότητα έχουν αντληθεί από το βιβλίο Structural Analysis IV του Dr. C. Caprani.

Έστω ότι δίδεται ένα δικτύωμα και ότι η ράβδος *ij* του δικτυώματος συνδέει τον κόμβο *i* και τον κόμβο *j* . Το μητρώο δυσκαμψίας της ράβδου θα είναι:

	Node *i*	**Node *j***
Node *i*	$k11_{ij}$	$k12_{ij}$
Node *j*	$k21_{ij}$	$k22_{ij}$

Εικόνα 4–10

Οι παραπάνω τιμές του μητρώου, θα πρέπει να τοποθετηθούν στις αντίστοιχες θέσεις του γενικού μητρώου του δικτυώματος ως εξής:

	…	Node *i*	…	Node *j*	…
…	…	…	…	…	…
Node *i*	…	$k11_{ij}$	…	$k12_{ij}$	…
…	…	…	…	…	…
Node *j*	…	$k21_{ij}$	…	$k22_{ij}$	…
…	…	…	…	…	…

Εικόνα 4–11

Αν τώρα εξεταστεί μια άλλη ράβδος του υποτιθέμενου δικτυώματος, η ράβδος *jl* που συνδέει τους κόμβους *j* και *l* . Το μητρώο ακαμψίας της ράβδου αυτής θα είναι:

	Node *j*	Node *l*
Node *j*	$k11_{jl}$	$k12_{jl}$
Node *l*	$k21_{jl}$	$k22_{jl}$

Εικόνα 4–12

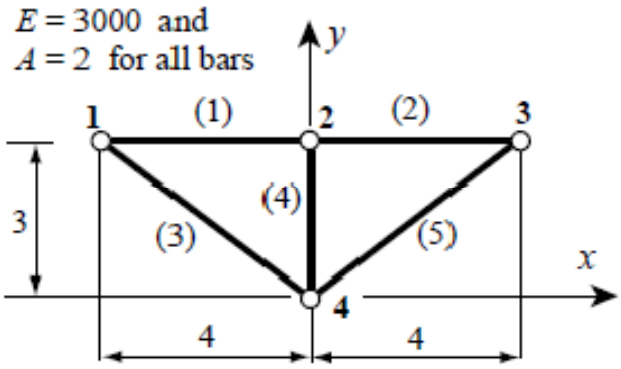
τώρα το γενικό μητρώο δυσκαμψίας θα διαμορφωθεί ως εξής:

	…	Node *i*	…	Node *j*	…	Node *l*	…
…	…	…	…	…	…	…	…
Node *i*	…	$k11_{ij}$	…	$k12_{ij}$	…	…	…
…	…	…	…	…	…	…	…
Node *j*	…	$k21_{ij}$	…	$k22_{ij} + k11_{jl}$	…	$k12_{jl}$	…
…	…	…	…	…	…	…	…
Node *l*	…		…	$k21_{lj}$	…	$k22_{jl}$	…
…	…	…	…	…	…	…	…

Εικόνα 4–13

Ακολουθεί ένα παράδειγμα για καλύτερη κατανόηση της ακολουθίας.

Παράδειγμα 2.2. Δίδεται το δικτύωμα στην Εικόνα 4–14 το οποίο αποτελείται από 5 ράβδους και 4 κόμβους. Να βρεθεί το γενικό μητρώο δυσκαμψίας του δικτυώματος.



Εικόνα 4–14

Σε αυτή τη φάση του παραδείγματος, όταν δεν έχουν υπολογιστεί τα επιμέρους μητρώα δυσκαμψία για κάθε ράβδο, το γενικό μητρώο δυσκαμψίας *K* έχει αυτήν την μορφή:

$$\mathbf{K} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{matrix}$$

Εικόνα 4–15

Το μέγεθος του γενικού μητρώου δυσκαμψίας, ορίζεται από τον αριθμό των κόμβων και τον αριθμό των βαθμών ελευθερίας. Στην προκειμένη περίπτωση είναι 8X8 , αφού το δικτύωμα του παραδείγματος έχει 4 κόμβους και επειδή είναι δισδιάστατο, δύο βαθμούς ελευθερίας ο κάθε κόμβος.

Με βάση την εξίσωση 2.13 για την ράβδο 1 το μητρώο δυσκαμψίας θα είναι:

$$\begin{bmatrix} 1500 & 0 & -1500 & 0 \\ 0 & 0 & 0 & 0 \\ -1500 & 0 & 1500 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix}$$

Element Freedom Table (EFT)

Εικόνα 4–16

Και συμπληρώνοντας, το γενικό μητρώο δυσκαμψίας παίρνει την εξής προσωρινή μορφή:

$$\begin{bmatrix} 1500 & 0 & -1500 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1500 & 0 & 1500 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ \\ \\ \\ \end{matrix}$$

Εικόνα 4–17

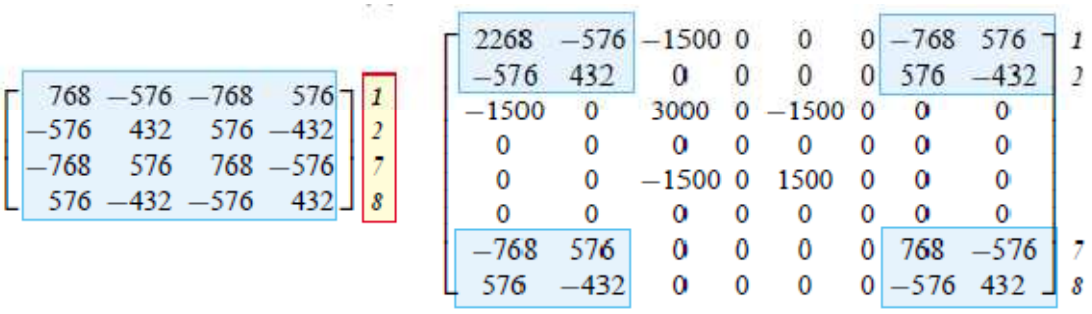
Συνεχίζοντας με την ράβδο 2 ομοίως θα προκύψει:

$$\begin{bmatrix} 1500 & 0 & -1500 & 0 \\ 0 & 0 & 0 & 0 \\ -1500 & 0 & 1500 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} 3 \\ 4 \\ 5 \\ 6 \end{matrix}$$

$$\begin{bmatrix} 1500 & 0 & -1500 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1500 & 0 & 3000 & 0 & -1500 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1500 & 0 & 1500 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} \\ \\ 3 \\ 4 \\ 5 \\ 6 \\ \\ \end{matrix}$$

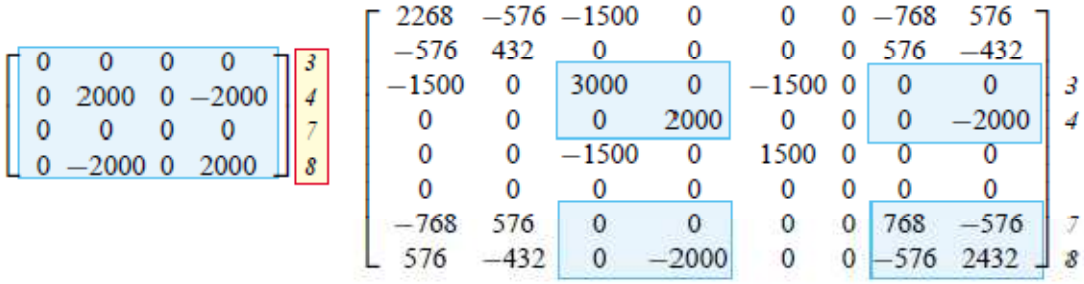
Εικόνα 4–18

Με την ράβδο 3:



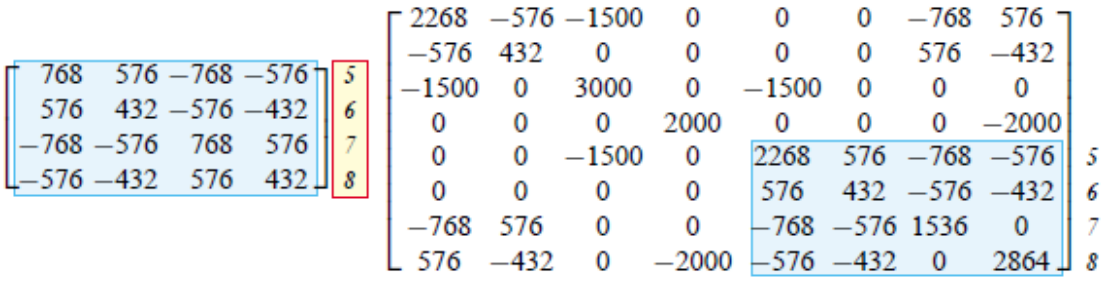
Εικόνα 4–19

Με την ράβδο 4:



Εικόνα 4–20

Και την ράβδο 5:



Εικόνα 4–21

Και τελικά προκύπτει το γενικό μητρώο δυσκαμψίας του δικτυώματος:

$$\mathbf{K} = \begin{bmatrix} 2268 & -576 & -1500 & 0 & 0 & 0 & -768 & 576 \\ -576 & 432 & 0 & 0 & 0 & 0 & 576 & -432 \\ -1500 & 0 & 3000 & 0 & -1500 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2000 & 0 & 0 & 0 & -2000 \\ 0 & 0 & -1500 & 0 & 2268 & 576 & -768 & -576 \\ 0 & 0 & 0 & 0 & 576 & 432 & -576 & -432 \\ -768 & 576 & 0 & 0 & -768 & -576 & 1536 & 0 \\ 576 & -432 & 0 & -2000 & -576 & -432 & 0 & 2864 \end{bmatrix}$$

Εικόνα 4–22

4.2.8 Μετατοπίσεις των κόμβων

Το μητρώο δυσκαμψίας του δικτυώματος [K] που υπολογίζεται πιο πάνω, χρειάζεται να διαμορφωθεί με βάση τους περιορισμούς ($Q_1 = Q_2 = Q_4 = Q_7 = Q_8 = 0$). Σε αυτό το σημείο θα χρησιμοποιηθεί μία προσέγγιση. Οι γραμμές και οι στήλες αντιστοιχούν στους βαθμούς ελευθερίας 1,2,4,7 και 8, που αναφέρονται στις σταθερές στηρίξεις, και έτσι διαγράφονται από το τελικό μητρώο δυσκαμψίας [K] πιο πάνω. Οι υπολογισμοί μας δίνουν:

$$\frac{7 \times 10^3}{3} \begin{bmatrix} 15.0 & 0 & 0 \\ 0 & 22.68 & 5.76 \\ 0 & 5.76 & 24.32 \end{bmatrix} \begin{Bmatrix} Q_3 \\ Q_5 \\ Q_6 \end{Bmatrix} = 10^3 \begin{Bmatrix} 20 \\ 0 \\ -25 \end{Bmatrix}$$

Λύνοντας το παραπάνω σύστημα προκύπτουν οι εξής μετατοπίσεις:

$Q_3 = 0.571\ mm$ $Q_5 = 0.119\ mm$ $Q_6 = -0.469\ mm$

Ο τελικός πίνακας των μετατοπίσεων για όλη την κατασκευή μπορεί τώρα να γραφτεί ως εξής:

$$\{Q\} = [0 \quad 0 \quad 0.571 \quad 0 \quad 0.119 \quad -0.469 \quad 0 \quad 0]^T mm$$

4.2.9 Υπολογισμός Τάσεων

Η τάση σε κάθε ράβδο μπορεί τώρα να υπολογιστεί με βάση τους κόμβους που συνδέονται στην ράβδο και τους πίνακες μετατόπισης τους, ως εξής:

Στο παραπάνω σχήμα, η ράβδος 1 συνδέει τον κόμβο 1 με τον κόμβο 2. Συνεπώς, ο πίνακας μετατόπισης της ράβδου 1 είναι:

$$\{q\} = [0 \quad 0 \quad 0.571 \quad 0]^T$$

Η εξίσωση που δίνει την τάση είναι η εξής:

$$\{\sigma\} = \frac{E_e}{L_e}[-l \quad -m \quad l \quad m]\begin{Bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{Bmatrix}$$

Έτσι με βάση αυτά που έχουμε υπολογίσει μέχρι τώρα, προκύπτουν τα αποτελέσματα:

$\sigma_{;} = 99.925 \ \frac{N}{mm^2} \ or \ MPa \quad \sigma_2 = -109.43 \ {}^{N}/_{mm^2} \ or \ MPa$

$\sigma_3 = -26.068 \ MPa \quad \sigma_4 = 20.825 \ MPa$

4.2.10 Αντιδράσεις στις στηρίξεις

Το τελικό βήμα σε ένα πρόβλημα και γενικότερα στην μέθοδο και στο κομμάτι των υπολογισμών είναι να βρούμε, δηλαδή, να υπολογίσουμε τις αντιδράσεις στις στηρίξεις. Χρειαζόμαστε τις αντιδράσεις κατά τις διευθύνσεις που έχουμε περιορισμούς δηλαδή στις 1,2,4,7 και 8, που απευθύνονται στις σταθερές υποστηρίξεις. Αυτές μπορούν να υπολογιστούν αντικαταστήσουμε για {Q} στην αρχική εξίσωση των αντιδράσεων:

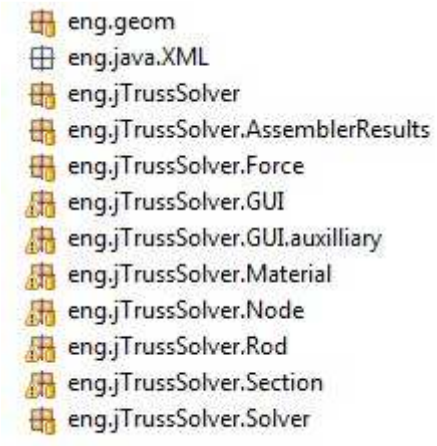
$$\begin{bmatrix} R_1 \\ R_2 \\ R_3 \\ R_4 \\ R_5 \end{bmatrix} = \frac{7 \times 10^3}{3}\begin{bmatrix} 22.68 & 5.76 & -15.0 & 0 & -7.68 & -5.76 & 0 & 0 \\ 5.76 & 4.32 & 0 & 0 & -5.76 & -4.32 & 0 & 0 \\ 0 & 0 & 0 & 20.0 & 0 & -20.0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -15.0 & 0 & 15.0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 0 \\ 0.571 \\ 0 \\ 0.119 \\ -0.469 \\ 0 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} -15814 \\ 3128 \\ 21887 \\ -4165 \\ 0 \end{bmatrix} N$$

Και έτσι υπολογίζουμε τις αντιδράσεις κατά τους βαθμούς περιορισμού στις στηρίξεις της κατασκευής.

5 Πρόγραμμα Επίλυσης Δικτυωμάτων (Κώδικας)

Σε αυτό το κεφάλαιο θα παρουσιαστεί ο κώδικας του προγράμματος, μία προς μία κλάση. Μία γενική όψη των πακέτων τους προγράμματος είναι η εξής:



Εικόνα 5–1: πακέτα του προγράμματος

Το καθένα από τα παραπάνω πακέτα, περιέχει κλάσεις που είναι απαραίτητες για την λειτουργία του προγράμματος. Για παράδειγμα το πακέτο `eng.jTrussSolver.Node` το οποίο περιέχει κάποιες κλάσεις που βοηθάνε στη συλλογή δεδομένων για τους κόμβους του δικτυώματος που είναι προς επίλυση. Στις επόμενες ενότητες θα αναλυθεί κάθε πακέτο, καθώς και οι κλάσεις που περιέχονται σε καθένα από αυτά.

5.1 eng.jTrussSolver

Αυτό το πακέτο περιέχει τις εξής κλάσεις:

- **main.TrussSolver.jav**a

Από αυτή τη κλάση το πρόγραμμα ξεκινάει τη λειτουργία του. Όπως είδαμε σε όλα τα παραδείγματα που παρουσιάστηκαν στις προηγούμενες ενότητες, κάθε κομμάτι κώδικα είχε και ένα μικρότερο κομμάτι που το λέγαμε «πρόγραμμα» και το παρουσιάζαμε ως το «το πρόγραμμα που υλοποιεί τον κώδικα». Το κομμάτι αυτό στην παρούσα εργασία είναι το εξής:

```
package eng.jTrussSolver;
import eng.jTrussSolver.GUI.frmMain;

public class mainTrussSolver {

	/**
	 * @param args
	 */
	public static void main(String[] args) {

		try
```

```
		{
			frmMain frame = new frmMain();
			frame.setVisible(true);
		} catch (Exception e) {
			e.printStackTrace();
		}
	}
}
```

όπου η `frmMain` είναι ένα αντικείμενο της κλάσης `JFrame` που έχουμε δημιουργήσει. Ουσιαστικά είναι το GUI που ο χρήστης εισάγει τα δεδομένα του δικτυώματος σχετικά με τους κόμβους, τις ράβδους κτλ. Περισσότερα για αυτή τη κλάση θα δούμε σε επόμενη υποενότητα, όπου θα αναλυθεί το πακέτο `eng.jTrussSolver.GUI`.

Το κομμάτι αυτό του κώδικα λοιπόν δημιουργεί ένα αντικείμενο της κλάσης `frmMain` το οποίο ονομάζει `frame` και το εμφανίζει, δηλαδή το κάνει ορατό, με την εντολή:

```
frame.setVisible(true);
```

5.2 eng.TrussSolver.GUI

5.2.1 Επισκόπηση πακέτου

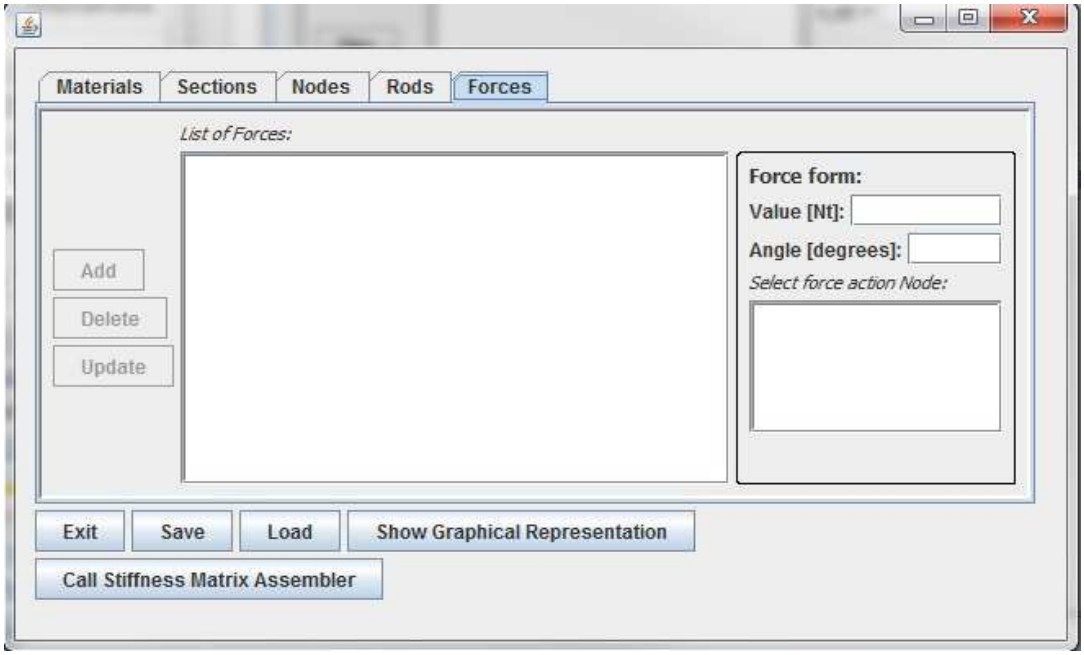
Το πακέτο αυτό αποτελείται από τις εξής κλάσεις:

- **frmMain.java**
- **Truss2dPanel.java**
- **Truss2dPanelAfterAssembler.java**

Η πρώτη κλάση `frmMain.java` περιέχει, όπως ειπώθηκε στην προηγούμενη υποενότητα, με την βασική φόρμα εισαγωγής των δεδομένων του δικτυώματος, ώστε να είναι δυνατή η επίλυση του από το πρόγραμμα. Η δεύτερη κλάση `Truss2dPanel.java` αφορά την γραφική αναπαράσταση του δικτυώματος πριν από την επίλυση του, δηλαδή χωρίς να απεικονίζεται η τελική του μορφή από την επίδραση φορτίων κτλ. Και η τρίτη κλάση `Truss2dPanelAfterAssembler.java` όπως θα μπορούσαμε να διακρίνουμε από το όνομα, αφορά την γραφική αναπαράσταση του δικτυώματος μας, μετά την επίλυση του, δηλαδή, τις τυχόν μετακινήσεις των κόμβων, ράβδων κτλ.

5.2.2 frmMain.java

Αυτή η κλάση είναι η βασικότερη του προγράμματος. Από εδώ ξεκινάνε σχεδόν όλες οι λειτουργίες του κάθε button που θα δούμε στο GUI παρακάτω. Η κλάση αυτή, δημιουργεί τη φόρμα με στοιχεία του δικτυώματος που πρέπει να εισάγει ο χρήστης όπως τα στοιχεία των κόμβων, των ράβδων, των δυνάμεων που ασκούνται σε κάθε κόμβο, των υλικών των ράβδων κτλ. Η γενική όψη του GUI αυτού φαίνεται στην παρακάτω εικόνα Εικόνα 5–2:



Εικόνα 5–2:

Ο κώδικας αυτής της κλάσης είναι σχετικά μεγάλος, μιάς και όπως είπαμε είναι η βασικότερη κλάση όλων, όπως επίσης περιέχει και σχεδόν όλες τις λειτουργίες που χρειάζεται το πρόγραμμα για να λειτουργήσει:

Στον παραπάνω κώδικα φαίνονται πάρα πολλά πράγματα, όπως είσοδος βιβλιοθηκών της Java, δηλώσεις μεταβλητών, ορισμός διαφόρων αντικειμένων στο GUI (JButton, JList, JTextField κτλ), διάφορες μέθοδοι που είναι απαραίτητες για την ομαλή και σωστή κατ' επέκταση λειτουργία του προγράμματος (όπως οι μέθοδοι που ανανεώνουν τις JList με τα δεδομένα που έχει καταχωρήσει ο χρήστης), ακροατές (Listeners) που εκτελούν συγκεκριμένες λειτουργίες όταν ο χρήστης κάνει κλικ σε κάποιο JButton (π.χ. για το JButton "Call Stiffness Matrix Assembler" όπου θα πρέπει να δημιουργηθεί ένα αντικείμενο της κλάσης που θα επιλύσει το δικτύωμα την οποία θα δούμε παρακάτω) και τέλος κάποιες μέθοδοι που εισάγουν κάποιο έτοιμο project στο πρόγραμμα (έχει οριστεί να αποθηκεύονται και να φορτώνονται, αντίστοιχα, αρχεία XML γι'αυτό το σκοπό).

Αντικείμενο της κλάσης αυτής, λοιπόν, είναι αυτό που δημιουργεί το αρχικό κομμάτι κώδικα που παρουσιάστηκε στην προηγούμενη υποενότητα και έτσι ο χρήστης είναι σε θέση να εισάγει ή να φορτώσει (Load) δεδομένα στο GUI ώστε να καταστεί δυνατόν να γίνει επίλυση και ενδεχομένως γραφική αναπαράσταση αυτού.

frmMain	
Attributes	**Methods**
SMA: StiffnessMatixAssemblr	- RefreshComboBoxNodes(): void
tr2d: Truss2dProblemDef	- RefreshComboBoxNodes_ForcesTAB(): void

SC: SecurityConditions	- RefreshComboBoxSections(): void
	- RefreshComboBoxMaterials(): void
	- RefreshComboBoxRods(): void
	- RefreshComboBoxForces(): void
	- RefreshChoiceSections(): void
	- RefreshChoiceMaterials() : void
	- RefreshChoiceNodes() : void
	- ResetAllSectionFields(): void
	- ResetAllMaterialFields(): void
	- RefreshChoiceMaterials() : void
	- ResetAllRodFields(): void
	- ResetAllNodeFields(): void
	- ResetAllForcesFields(): void
	- ResetAllChoicesOfRods(): void
	- LoadXMLdatafile(String): void
	- listNodes(Document): void

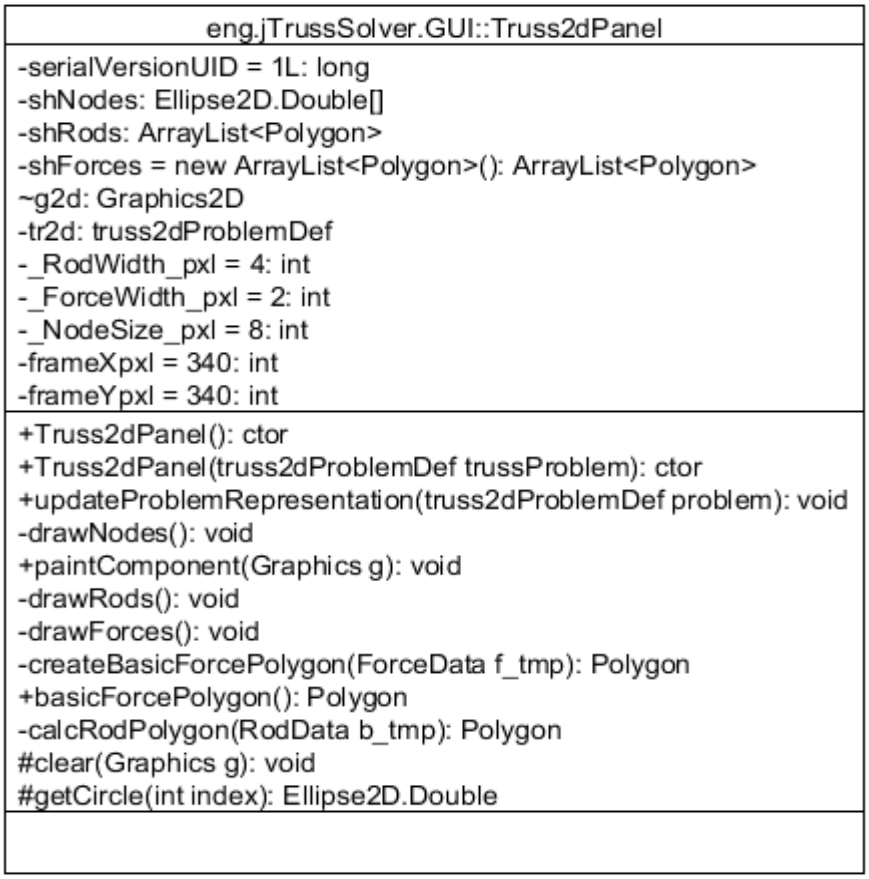
Πίνακας 5.1

5.2.3 Truss2dPanel.java

Ο κώδικας αυτής της κλάσης δημιουργεί ένα GUI με την γραφική αναπαράσταση της αρχικής μορφής του δικτυώματος, δηλαδή, αυτή πριν της επίδρασης των δυνάμεων στους κόμβους και στις ράβδους αυτού. Αυτή μέθοδος καλείται κάνοντας κλικ στο JButton "Show Graphical Representation" που ορίζεται στη κλάση `frmMain` που παρουσιάσαμε προηγουμένως μέσω ενός ακροατή που έχουμε ορίσει σε αυτό. Ο ακροατής δημιουργεί ένα αντικείμενο της κλάσης `Truss2dPanel` και δίνει σαν ορίσματα όλα τα δεδομένα που είναι απαραίτητα για την γραφική αναπαράσταση του δικτυώματος (στην αρχική του κατάσταση). Η εντολή που δημιουργεί το αντικείμενο βρίσκεται στη κλάση `frmMain` και είναι η εξής:

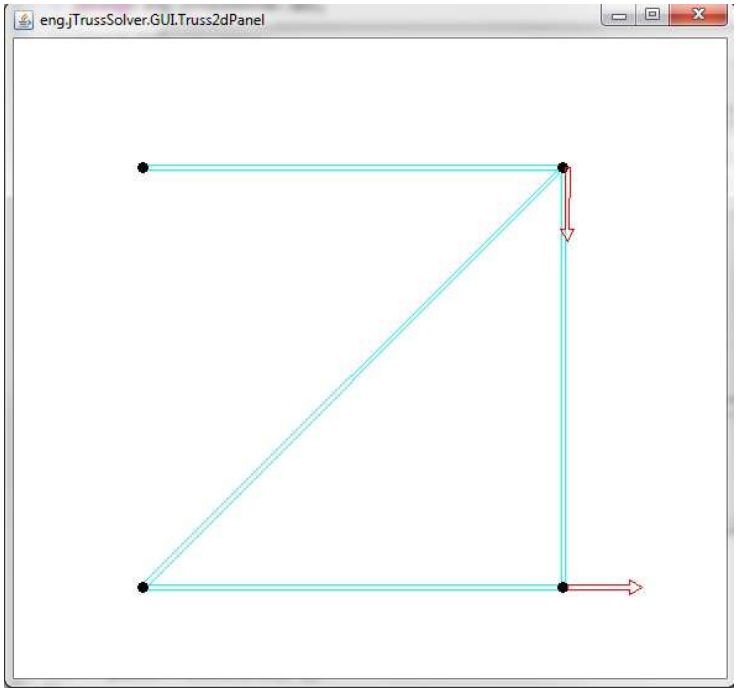
```
WindowsUtilities.openInJFrame(new Truss2dPanel(tr2d), 700, 600);
```

όπου η κλάση WindowsUtilities είναι μια κλάση είναι βοηθητική, ορίζεται στο πακέτο `eng.jTrussSolver.GUI.auxilliary` που θα παρουσιάσουμε σε επόμενη υποενότητα. Η μέθοδος `openInJFrame(new Truss2dPanel(tr2d), 700, 600)`είναι μια μέθοδος που ορίζεται στην κλάση WindowsUtilities και είναι αυτή που δημιουργεί το JFrame που χρειάζεται για την σχεδίαση του δικτυώματος, δηλαδή, την γραφική αναπαράσταση της αρχικής του μορφής. Το όρισμα `tr2d` είναι ένα αντικείμενο της κλάσης `truss2dProblemDef` που θα παρουσιαστεί σε επόμενη ενότητα. Οι αριθμοί που δύνονται σαν ορίσματα, δηλαδή, το *700* και το *600* είναι το ύψος και το πλάτος του παραθύρου που θα δημιουργηθεί (*σε pixels*).



Εικόνα 5–3

Σε αυτή τη κλάση χρησιμοποιούνται κάποιες ArrayList, όπως έχουμε δει σε προηγούμενα κεφάλαια, οι οποίες βοηθούν στην αποθήκευση των σχημάτων πριν την τελική σχεδίαση τους. Μία γραφική αναπαράσταση της αρχικής μορφής ενός δικτυώματος, ενδεικτικά, είναι η εξής:



Εικόνα 5–4: Παράθυρο με την γραφική αναπαράσταση του προβλήματος

με μαύρο χρώμα φαίνονται οι κόμβοι, με γαλάζιο οι ράβδοι και με κόκκινα βέλη διακρίνονται οι δυνάμεις που επιδρούν στους κόμβους.

Οι δύο συναρτήσεις με το όνομα Truss2dPanel() είναι οι διαθέσιμοι constructor της κλάσης. Η updateProblemRepresentation χρησιμοποιείται για να ανανεωθούν τα δεδομένα του πρόβληματος. Η drawNodes() χρησιμοποιείται για τον σχεδιασμό των κόμβων. Η paintComponent(Graphics) σχεδιάζει το σχήμα-σχήματα. Η drawRods() χρησιμοποιείται για τον σχεδιασμό των ράβδων. Η drawForces() χρησιμοποιείται για τον σχεδιασμό των δυνάμεων. Η drawNodesText() χρησιμοποιείται για την εισαγωγή κειμένου δίπλα από κάθε κόμβο, στο οποίο φαίνονται κάποια στοιχεία για τον εκάστοτε κόμβο όπως οι συντεταγμένες του. Η createBasicForcePolygon(ForceData) χρησιμοποιείται για την σχεδίαση του πολυγώνου του βέλους το οποίο απεικονίζει μία δύναμη. Η calcRodPolygon(RodData) χρησιμοποιείται για την σχεδίαση του πολυγώνου που αναπαριστά μία ράβδο. Η clear(Graphics) χρησιμοποιείται για τον καθαρισμό του παραθύρου με τα γραφικά. Η getCircle(int) επιστρέφει έναν κύκλο ο οποίος αναπαριστά γραφικά έναν κόμβο.

5.2.4 Truss2dPanelAfterAssembler.java

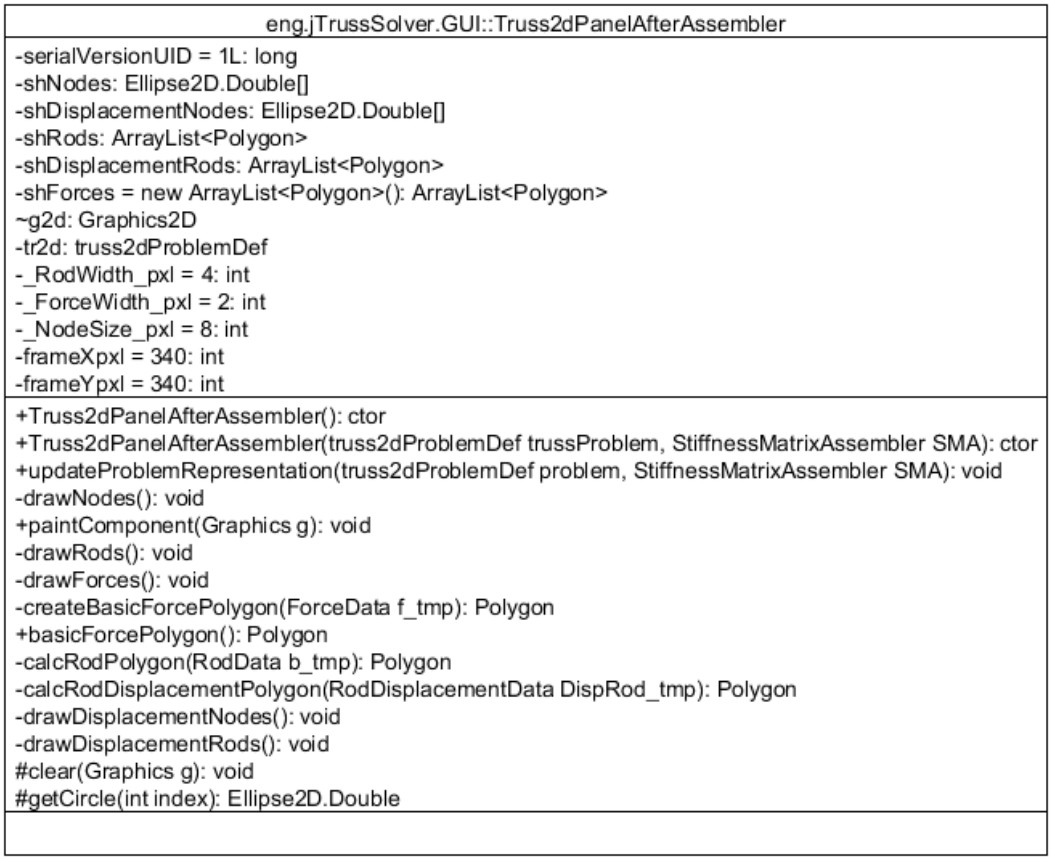
Αυτή η κλάση είναι παρόμοια με την προηγούμενη. Όπως φαίνεται και στο από τ'ονομά της έχει μια βασική διαφορά. Αυτή είναι ότι αναπαριστά γραφικά την κατάσταση του δικτυώματος μετά τους υπολογισμούς, κοινώς, λαμβάνει υπόψην του τις επιδράσεις, δηλαδή, τις μετακινήσεις των κόμβων και κατ' επέκταση των ράβδων εξαιτίας των δυνάμεων που ασκούνται στους κόμβους. Η εμφάνιση του παραθύρου γίνεται αμέσως μετά την επίλυση του δικτυώματος, δηλαδή, αμέσως μετά το πάτημα του JButton "Call Stiffness Matrix Assembler". Επίσης, επιλέχθηκε να αναπαρίσταται γραφικά και η αρχική

κατάσταση του ολοκληρώματος για καλύτερη κατανόηση από τον χρήστη, των επιδράσεων των δυνάμεων που ασκούνται.

Η δημιουργία του αντικειμένου της κλάσης αυτής, γίνεται από τον ακροατή του JButton “Call Stiffness Matrix Assembler” ο οποίς βρίσκεται στη κλάση `frmMain` , αφού αμέσως πρίν δημιουργηθεί ένα αντικείμενο της κλάσης `StiffnessMatrixAssembler` (την οποία θα παρουσιάσουμε σε επόμενη υποενότητα). Ο ακροατής που περιέχει την δημιουργία του αντικειμένου της κλάσης αυτής, έχει τον εξής κώδικα:

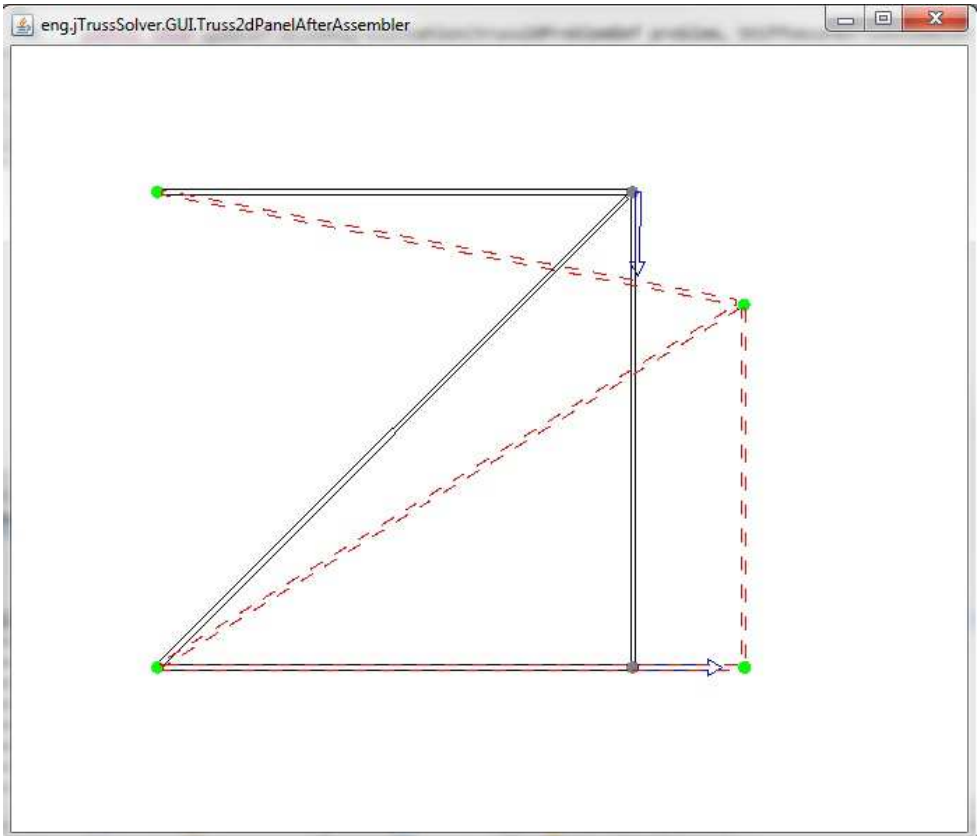
```
btnCallStiffnessMatrix.addActionListener(new ActionListener() {
	public void actionPerformed(ActionEvent arg0) {
		SMA = new StiffnessMatrixAssembler(tr2d);
		WindowsUtilities.openInJFrame(new
Truss2dPanelAfterAssembler(tr2d,SMA), 700, 600);
}
});
```

η μεταβλητή SMA έχει οριστεί προηγουμένως ότι είναι η μεταβλητή που δηλώνει αντικείμενο της κλάσης `StiffnessMatrixAssembler` η οποία όπως θα δούμε επιλύει το δικτύωμα. Αμέσως μετά την επίλυση του δικτυώματος, όπως είδαμε και προηγουμένως, με παρόμοιο τρόπο εμφανίζεται το παράθυρο με την νέα γραφική αναπαράσταση. Παρατηρούμε πως έχει ένα παραπάνω όρισμα αυτή τη φορά, το αντικείμενο της κλάσης όπου επιλύθηκε το δικτύωμα. Σε αυτό υπάρχουν όλα τα απαραίτητα δεδομένα που χρειάζεται η κλάση `Truss2dPanelAfterAssembler` ώστε να σχεδιάσει το νέο (μαζί με το αρχικό) σχήμα του δικτυώματος.



Εικόνα 5–5:

Σε αρκετά σημεία ο παραπάνω κώδικας είναι ίδιος με αυτόν της κλάσης `Truss2dPanel`. Το κύριο χαρακτηριστικό είναι κάποιες πρόσθετες μέθοδοι που αφορούν την σχεδίαση του νέου σχήματος (μορφής) του δικτυώματος με τις μετατοπίσεις των κόμβων και των ράβδων. Ενδεικτικά, στην παρακάτω εικόνα Εικόνα 5–6Εικόνα 5–5 φαίνεται πως θα μπορούσε να ήταν το αποτέλεσμα του δικτυώματος που είδαμε προηγουμένως:



Εικόνα 5–6

τα χρώματα έχουν αλλάξει, με σκοπό την καλύτερη κατανόηση από τον χρήστη των αλλαγών που έχουν γίνει. ‘Ετσι, με μαύρο χρώμα πλέον διακρίνονται οι ράβδοι στην αρχική κατάσταση, με γκρί οι κόμβοι στην αρχική τους θέση και με μπλέ οι δυνάμεις. Με κόκκινο χρώμα και διακεκομμένες γραμμές φαίνονται οι ράβδοι στην νέα τους κατάσταση και με πράσινο χρώμα οι νέες θέσεις των κόμβων μετά τους υπολογισμούς. Επίσης, παρατηρούμε πως σε δύο κόμβουν δεν υπάρχει κάποα μετακινήση και είναι επόμενο, καθώς σε αυτούς δεν ασκείται κάποια δύναμη. Να σημειωθεί σε αυτό το σημείο πως στον κώδικα της κλάσης αυτής, έχει υπολογισθεί και προβλεφθεί κατάλληλος συντελεστής κλίμακας, ώστε σε περίπτωση που οι μετακινήσεις είναι πολύ μικρές να διαμορφώνεται έτσι η γραφική αναπαράσταση, ώστε να γίνονται αντιληπτές από τον χρήστη.

Πιο συγκεκριμμένα οι δύο συναρτήσεις Truss2dPanelAfterAssembler() είναι οι διαθέσιμοι constructors της κλάσης. Η calcRodDisplacementPolygon(RodDisplacementData) χρησιμοποιείται για την σχεδίαση των πολυγώνων που χρησιμοποιούνται για την γραφική αναπαράσταση των μετακινούμενων πλέον ράβδων. Η drawDisplacementNodes() χρησιμοποιείται για την σχεδίαση των μετακινούμενων κόμβων και η drawDisplacementRods() για την σχεδίαση των μετακινούμενων ράβδων. Όλες οι υπόλοιπες συναρτήσεις είναι παρόμοιες με αυτές της κλάσης Truss2dPanel.java .

5.3 eng.jTrussSolver.GUI.auxilliary

5.3.1 Επισκόπηση πακέτου

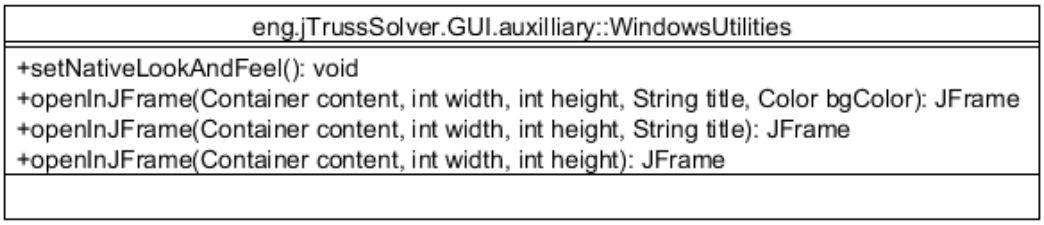
Αυτό το πακέτο περιέχει κάποιες βοηθητικές κλάσεις για τα GUI. Οι κλάσεις αυτές είναι οι εξής:

- **WindowsUtilities.java**
- **ExitListener.java**
- **SecurityConditions.java**

Η πρώτη και η δεύτερη κλάση (WindowsUtilities & ExitListener αντίστοιχα) έχουν να κάνουν με τις κλάσεις που αφορούν την γραφική αναπαράσταση πριν και μετά της επίλυση του δικτυώματος. Η μεν πρώτη για την δημιουργία του GUI (παραθύρου γραφικής σχεδίασης) και τις παραμέτρους του και δεύτερη για την λειτουργία του κουμπιού δεξιά πάνω στο GUI , όπου ορίζει ότι με το πάτημα του, το παράθυρο κλείνει. Η τρίτη κλάση έχει να κάνει με την εισαγωγή δεδομένων, δηλαδή με τη κλάση frmMain και ο κώδικάς της περιέχει διάφορες μεθόδους, που ελέγχουν και κατά συνθήκη (περίπτωση) εμφανίζουν διάφορα μηνύματα στον χρήστη. Έτσι αποφεύγεται «κρασάρισμα» του προγράμματος που θα μπορούσε να δημιουργηθεί από εσφαλμένη εισαγωγή δεδομένων, όπως να είχε εισαχθεί ένα αλφαριθμιτικό σε ένα πεδίο που θα έπρεπε να εισαχθεί κάποιος αριθμός. Έτσι δίδεται η δυνατότητα στο χρήστη να διορθώσει τα κατά περίπτωση πεδία.

5.3.2 WindowsUtilities.java

Αυτή η κλάση δημιουργεί ένα αντικείμενο το οποίο είναι ουσιαστικά το παράθυρο με όλα τα στοιχεία του, όπως ο τίτλος κτλ.

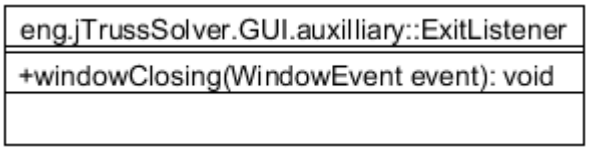


Εικόνα 5–7

όπως παρατηρούμε περιέχει διάφορες μεθόδους που εχουν ορίσματα, το περιεχόμενο, το ύψος του παραθύρου και το πλάτος καθώς και τον τίτλο του. Επίσης, πριν από κάθε μέθοδο υπάρχουν τα κατάλληλα σχόλια, τα οποία περιγράφουν την λειτουργία της μεθόδου που ακολουθεί.

5.3.3 ExitListener.java

Αυτό που περιγράφει η κλάση ExitListener.java, είναι ότι σε περίπτωση που γίνει κλικ στο κουμπί που βρίσκεται δεξιά πάνω στο παράθυρο (X) τότε το παράθυρο θα κλείσει.



Εικόνα 5–8

5.3.4 SecurityConditions.java

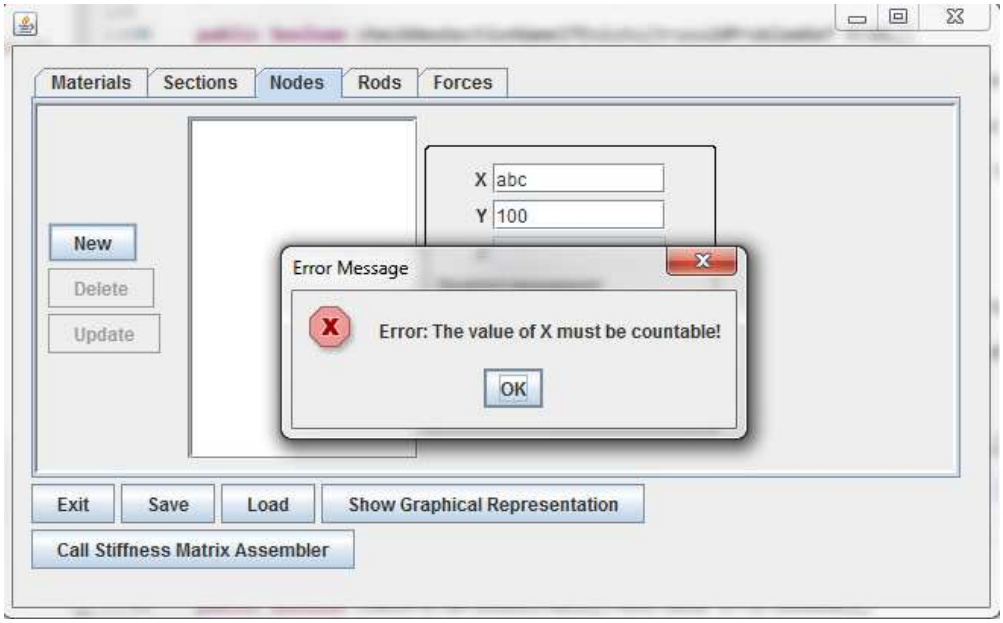
Όπως προαναφέρθηκε στην εισαγωγή τους παρούσας υποενότητας, αυτή η κλάση, έχει ως σκοπό να περιέχει κάποιες μεθόδους, που είναι απαραίτητες για τον έλεγχο από την κλάση `frmMain` κάποιων δεδομένων που εισάγει ο χρήστης. Είναι μια βοηθητική κλάση της κλάσης `frmMain`.

eng.jTrussSolver.GUI.auxilliary::SecurityConditions
- checkMaterialFieldsIfNull(JTextField, JTextField, JTextField):boolean
- checkMaterialNameIfExists(truss2dProblemDef, String, JTextField):boolean
- checkNewMaterialNameIfExists(truss2dProblemDef, JTextField):boolean
- checkMaterialFieldsIfAccepted(JTextField, JTextField) :boolean
- checkSectionFieldsIfNull(JTextField, JTextField, JTextField) :boolean
- checkSectionNameIfExists(truss2dProblemDef, String, JTextField) :boolean
- checkNewSectionNameIfExists(truss2dProblemDef, JTextField) :boolean
- checkSectionFieldsIfAccepted(JTextField, JTextField) :boolean
- checkNodeFieldsIfNull(JTextField, JTextField) :boolean
- checkNodeIfAlreadyExists(truss2dProblemDef, JTextField, JTextField) :boolean
- checkNodeFieldsIfAccepted(JTextField, JTextField) :boolean
- checkUpdatedNodeIfAlreadyExists(truss2dProblemDef, JTextField, JTextField, JList<String>):boolean
- checkRodFieldsAndChoiceIfNull(truss2dProblemDef, JTextField, Choice, Choice, Choice, Choice) :boolean
- checkNewRodNameIfExists(truss2dProblemDef, JTextField) :boolean
- checkRodSelectedNodesIfEquals(Choice, Choice) :boolean
- checkSelectedNodesIfAlreadyConnected(truss2dProblemDef, Choice, Choice) :boolean
- checkRodNewNameIfAlreadyExists(truss2dProblemDef, String, JTextField) :boolean
- checkRodNewNodesIfAlreadyConnected(truss2dProblemDef, int, Choice, Choice) :boolean
- checkForceFieldsIfNull(JTextField, JTextField) :boolean
- checkForceFieldsIfNumbers(JTextField, JTextField) :boolean
- CheckForceIfAlreadyExists(truss2dProblemDef, JTextField, JTextField, JList<String>):boolean
- CheckUpdatedForceIfAlreadyExists(truss2dProblemDef, JTextField, JTextField, JList<String>, JList<String>):boolean
- checkIfConnectedRodsOnNode(truss2dProblemDef, int) :boolean
- checkIfActionForcesOnNode(truss2dProblemDef, int) :boolean
- checkIfSectionIsRegToRod(truss2dProblemDef, String) :boolean
- checkIfMaterialIsRegToRod(truss2dProblemDef, String) :boolean

Πίνακας 5.2

όπως φαίνεται Πίνακας 5.2, περιέχει πολλές απλές boolean μεθόδους που όλες απλά ελέγχουν τα δεδομένα που εισάγονται από τον χρήστη και επιστρέφουν true ή false.

Ένα παράδειγμα μίας λάθος εισαγωγής, π.χ. αλφαριθμητικού αντί για αριθμό, θα εμφάνιζε ένα μήνυμα σαν το παρακάτω στον χρήστη Εικόνα 5–9:



Εικόνα 5–9

Η συνάρτηση checkMaterialFieldsIfNull(JTextField, JTextField, JTextField) χρησιμοποιείται για να ελεγχθεί εάν υπάρχει κάποιο κενό πεδίο στην καρτέλα με τα υλικά, πριν από την εισαγωγή ή τροποποίηση κάποιας καταχώρησης. Η checkMaterialNameIfExists(truss2dProblemDef, String, JTextField) χρησιμοποιείται για τον έλεγχο πριν την εισαγωγή κάποιας καταχώρησης υλικού του ονόματος το οποίο είναι προς καταχώρηση. Εάν το όνομα υπάρχει ήδη, εμφανίζει το αντίστοιχο μήνυμα σφάλματος. Η checkNewMaterialNameIfExists(truss2dProblemDef, JTextField) ελέγχει σε περίπτωση τροποποίησης κάποιας καταχώρησης, εάν το νέο όνομα που τυχόν έχει δοθεί υπάρχει ήδη. Η checkMaterialFieldsIfAccepted(JTextField, JTextField) ελέγχει εάν όλα τα πεδία που έχουν συμπληρωθεί για κάποιο υλικό είναι αποδεκτά, δηλαδή δεν υπάρχουν συντακτικά λάθη.

Η checkSectionFieldsIfNull(JTextField, JTextField, JTextField) ελέγχει εάν όλα τα πεδία της υπό τροποποίηση ή εισαγωγή διατομής έχουν συμπληρωθεί. Η checkSectionNameIfExists(truss2dProblemDef, String, JTextField) ελέγχει εάν το όνομα διατομής σε μία νέα καταχώρηση υπάρχει ήδη και αν ναι, τότε εμφανίζει το αντίστοιχο μήνυμα σφάλματος. Η checkNewSectionNameIfExists(truss2dProblemDef, JTextField) ελέγχει εάν το νέο όνομα που τυχόν έχει δοθεί σε μία υπάρχουσα καταχώρηση υλικού υπάρχη ήδη και αν ναι, τότε εμφανίζει το αντίστοιχο μήνυμα σφάλματος. Η checkSectionFieldsIfAccepted(JTextField, JTextField) ελέγχει εάν όλα τα πεδία σε μία εισαγωγή ή τροποποίηση υλικού είναι συντακτικά σωστά.

Η checkNodeFieldsIfNull(JTextField, JTextField) ελέγχει εάν έχουν συμπληρωθεί όλα τα πεδία σε μία νέα καταχώρηση ή τροποποίηση κόμβου. Η checkNodeIfAlreadyExists(truss2dProblemDef, JTextField, JTextField) ελέγχει εάν ο κόμβος που πρόκειται να εισαχθεί υπάρχει ήδη. Η checkNodeFieldsIfAccepted(JTextField, JTextField) ελέγχει εάν όλα τα πεδία που έχουν εισαχθεί σε μία νέα καταχώρηση ή τροποποίηση κόμβου είναι συντακτικά αποδεκτά. Η checkUpdatedNodeIfAlreadyExists(truss2dProblemDef, JTextField, JTextField, JList<String>) ελέγχει εάν τα στοιχεία ενός κόμβου που επιχειρείται να τροποποιηθεί υπάρχουν ήδη.

Η checkRodFieldsAndChoiceIfNull(truss2dProblemDef, JTextField, Choice, Choice, Choice, Choice) ελέγχει εάν όλα τα πεδία σε μία νέα καταχώρηση ή τροποποίηση ράβδου έχουν συμπληρωθεί. Η checkNewRodNameIfExists(truss2dProblemDef, JTextField) ελέγχει εάν το όνομα μιας ράβδου που βρίσκεται υπο εισαγωγή υπάρχει ήδη. Η checkRodSelectedNodesIfEquals(Choice, Choice) ελέγχει εάν ο αρχικός και ο τελικός κόμβος που έχουν επιλεχθεί σε μία νέα καταχώρηση ράβδου είναι ίδιοι. Η checkSelectedNodesIfAlreadyConnected(truss2dProblemDef, Choice, Choice) ελέγχει εάν οι κόμβοι που είναι υπό σύνδεση με κάποια ράβδο είναι ήδη συνδεμένοι με κάποια άλλη ράβδο. Η checkRodNewNameIfAlreadyExists(truss2dProblemDef, String, JTextField) ελέγχει εάν το νέο όνομα που έχει δωθεί σε μία ράβδο που βρίσκεται υπό τροποποίηση, υπάρχει ήδη. Η checkRodNewNodesIfAlreadyConnected(truss2dProblemDef, int, Choice, Choice) ελέγχει εάν οι κόμβοι που έχουν επιλεγχθεί σε μία τροποποίηση μίας καταχωρημένης ράβδου, είναι ήδη συνδεμένοι από μία άλλη ράβδο.

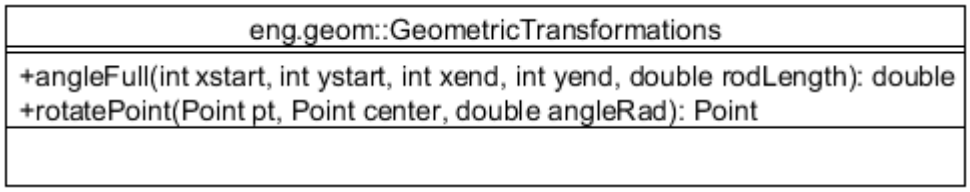
Η checkForceFieldsIfNull(JTextField, JTextField) χρησιμοποιείται για τον έλεγχο σε μία καταχώρηση δύναμης των απαραίτητων πεδίων, εάν έχουν συμπληρωθεί. Η checkForceFieldsIfNumbers(JTextField, JTextField) ελέγχει εάν τα αριθμητικά πεδία σε μία καταχώρηση δύναμης είναι αριθμοί. Η CheckForceIfAlreadyExists(truss2dProblemDef, JTextField, JTextField, JList<String>) ελέγχει εάν η δύναμη υπάρχει ήδη στις καταχωρήσεις. Η CheckUpdatedForceIfAlreadyExists(truss2dProblemDef, JTextField, JTextField, JList<String>, JList<String>) ελέγχει εάν η δύναμη που είναι υπό τροποποίηση υπάρχει ήδη στις καταχωρήσεις.

Η checkIfConnectedRodsOnNode(truss2dProblemDef, int) χρησιμοποιείται για τον έλεγχο κατά την διαγραφή ενός κόμβου, εάν υπάρχουν συνδεμένες ράβδοι σε αυτόν. Η checkIfActionForcesOnNode(truss2dProblemDef, int) ελέγχει κατά την διαγραφή ενός κόμβου εάν υπάρχουν ασκούμενες σε αυτόν δυνάμεις. Η checkIfSectionIsRegToRod(truss2dProblemDef, String) ελέγχει εάν η διατομή που είναι υπό διαγραφή, είναι καταχωρημένη σε κάποια ράβδο. Η checkIfMaterialIsRegToRod(truss2dProblemDef, String) ελέγχει εάν το υλικό που είναι υπο διαγραφή είναι καταχωρημένο σε κάποια ράβδο.

Όλες οι παραπάνω συναρτήσεις είναι λογικού τύπου και επιστρέφουν κατά περίπτωση true ή false.

5.4 eng.geom

Σε αυτό το πακέτο περιέχεται μία κλάση, η `GeometricTransformations.java`, της οποίας ο κώδικας είναι βοηθητικός της γραφικής σχεδίασης.



Εικόνα 5–10

όπως διακρίνουμε Εικόνα 5–10, περιέχει δύο μεθόδους. Η angleFull(int, int, int, int, double) επιστρέφει έναν δεκαδικό που στην περίπτωσή μας είναι μία γωνία. Πιο συγκεκριμμένα, είναι η γωνία του υπό σχεδίαση τμήματος ράβδου. Η rotatePoint(Point, Point, double) αφού περιστρέψει, ένα σημείο γύρω από ένα άλλο κατά μία συγκεκριμένη γωνία. Το σημείο (pt) ορίζεται από μία τετμημένη X και μία τεταγμένη Y.

5.5 eng.jTrussSolver.Material

5.5.1 Επισκόπηση πακέτου

Το πακέτο αυτό περιέχει δύο κλάσεις που έχουν να κάνουν με τις πληροφορίες για τα υλικά (Materials):

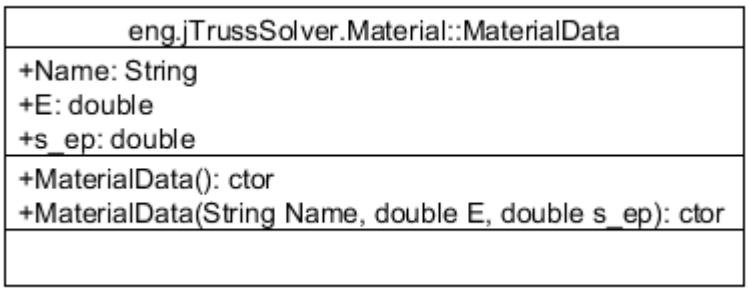
- **MaterialData.java**
- **MaterialsHashMap.java**

Η MaterialData.java έχει να κάνει με τα στοιχεία που συγκρατεί για κάθε υλικό και η MaterialsHashMap.java με τη δομή δεδομένων συλλογής όλων των υλικών.

5.5.2 MaterialData.java

Κάθε αντικείμενο αυτής της κλάσης, περιγράφει ένα υλικό. Τα δεδομένα που αποθηκεύονται για κάθε υλικό είναι τα εξής:

- ✓ Name (όνομα)
- ✓ E (μέτρο ελαστικότητας)
- ✓ s_ep (επιτρεπόμενη τάση)

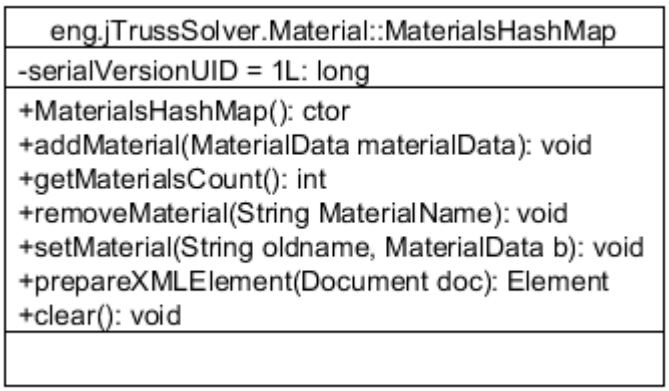


Εικόνα 5–11

όπως φαίνεται και από τα ορίσματα Εικόνα 5–11, το όνομα είναι αλφαριθμητικό, το μέτρο ελαστικότητας είναι ακέραιος αριθμός καθώς και η επιτρεπόμενη τάση. Η συναρτήσεις MaterialData() είναι οι διαθέσιμοι constructor της κλάσης.

5.5.3 MaterialsHashMap.java

Σε αυτήν την κλάση, διαμορφώνεται η δομή δεδομένων συλλογής για τα υλικά. Χρησιμοποιήσαμε HashMap σε αυτό το κομμάτι κώδικα, όπως και στις άλλες συλλογές (δυνάμεις, κόμβοι, ράβδοι) που θα δούμε παρακάτω.



Εικόνα 5–12

χρησιμοποιήθηκε σαν key το όνομα του υλικού και σαν στοιχείο (element) το αντίστοιχο αντικείμενο της κλάσης `MaterialData`.

Η MaterialsHashMap() είναι ο διαθέσιμος contructor της κλάσης. Η addMaterial(MaterialData)χρησιμοποιείται για την εισαγωγή νέου υλικού. Η getMaterialsCount() επιστρέφει τον αριθμό των καταχωρήσεων. Η removeMaterial(String) χρησιμοποιείται για την διαγραφή κάποιας καταχώρησης υλικού. Η setMaterial(String, MaterialData) χρησιμοποιείται για την τροποιποίηση κάποιας καταχώρησης υλικού. Η prepareXMLElement(Document) χρησιμοποιείται για την προετοιμασία του αρχείου xml που επιχειρηθεί για αποθήκευση του project. Η clear() χρησιμοποιείται για τον καθαρισμό όλης της συλλογής δεδομένων για τα υλικά.

5.6 eng.jTrussSolver.Node

5.6.1 Επισκόπηση πακέτου

Σε αυτό το πακέτο ορίζονται τα δεδομένα και η δομή δεδομένων συλλογής για τους κόμβους. Περιέχει δύο κλάσεις, όπως και το πακέτο με τα υλικά, οι οποίες είναι οι εξής:

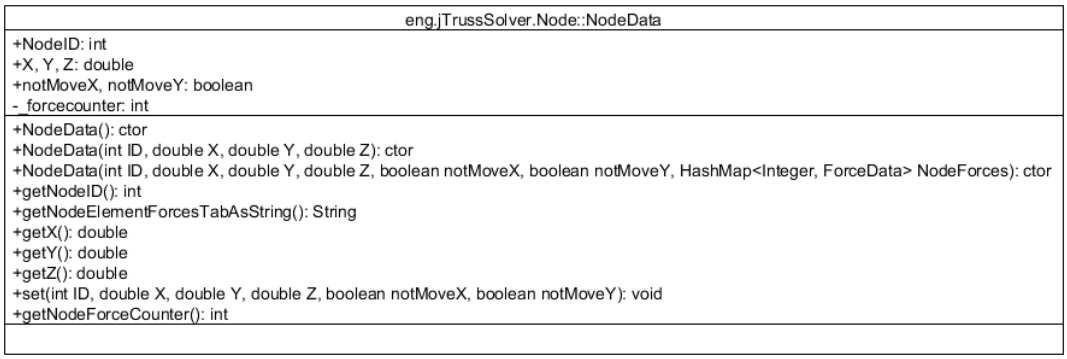
- **NodeData.java**
- **NodeHashMap.java**

Η NodeData.java έχει να κάνει με τα στοιχεία που συγκρατεί για κάθε κόμβο και η NodeHashMap.java με τη δομή δεδομένων συλλογής όλων των κόμβων.

5.6.2 NodeData.java

Κάθε αντικείμενο αυτής της κλάσης, αποθηκεύει τα χαρακτηριστικά για κάθε κόμβο. Τα δεδομένα που συγκρατεί για κάθε κόμβο είναι τα εξής:

- ✓ ID (αναγνωριστικό αριθμός, ακέραιος)
- ✓ X (συντεταγμένη X, δεκαδικός)
- ✓ Y (συντεταγμένη Ψ, δεκαδικός)
- ✓ Z (συντεταγμένη Z –στη περίπτωση μας είναι 0-)
- ✓ notMoveX (περιορισμός κίνησης κατά τον άξονα X, λογική τιμή)
- ✓ notMoveY (περιορισμός κίνησης κατά τον άξονα Ψ, λογική τιμή)



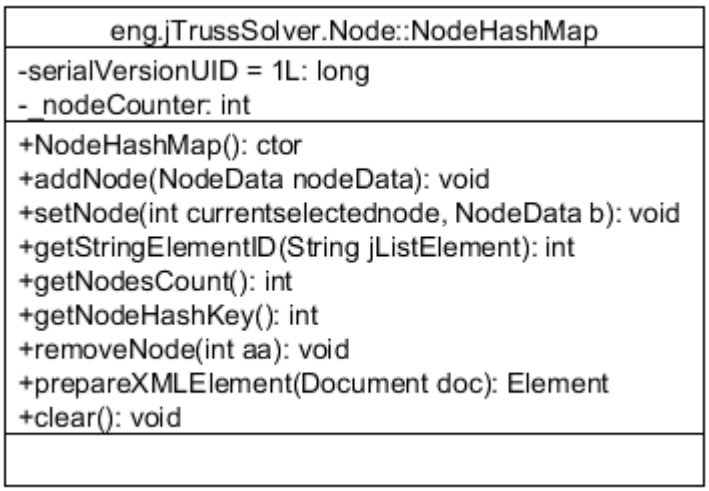
Εικόνα 5–13

Σαν ID λαμβάνεται υπόψην ο μετρητής των κόμβων που εισάγονται. Έτσι ο αριθμός αυτός θα είναι πάντα αυξανόμενος. Όσον αφορά τις παραμέτρους notMoveX και notMoveY εξαρτώνται από τις στηρίξεις στους κόμβους. Έτσι, αν για παράδειγμα ένας κόμβος έχει σαν στήριξη άρθρωση, τότε οι λογικές μεταβλητές notMoveX, notMoveY θα έχουν τιμή true. Αυτές λοιπόν οι μεταβλητές αφορούν τους **περιορισμούς**. Ο χρήστης καθορίζει αυτές τις μεταβλητές μέσω box επιλογής (check box). Οπότε αν ένα check box π.χ. για την κίνηση στον άξονα των X είναι επιλεγμένο τότε αυτόματα η τιμή που θα πάρει η μεταβλητή notMoveX θα είναι true.

Οι συναρτήσεις NodeData() είναι οι διαθέσιμοι constructors της κλάσης. Η getNodeID() χρησιμοποιείται για να επιστρέφει το ID ενός κόμβου. Η getNodeElementForcesTabAsString() χρησιμοποιείται για να δημιουργεί και να επιστρέφει μία καταχώρηση στην λίστα με τους κόμβους που βρίσκεται στην καρτέλα με τις δυνάμεις. Οι συναρτήσεις getX() , getY() και getZ() χρησιμοποιούνται για την επιστροφή των συντεταγμένων X, Y, Z αντίστοιχα, κάποιου κόμβου. Η set(int, double, double, double, boolean, boolean) χρησιμοποιείται για την τροποποίηση μιας καταχώρησης κόμβου.

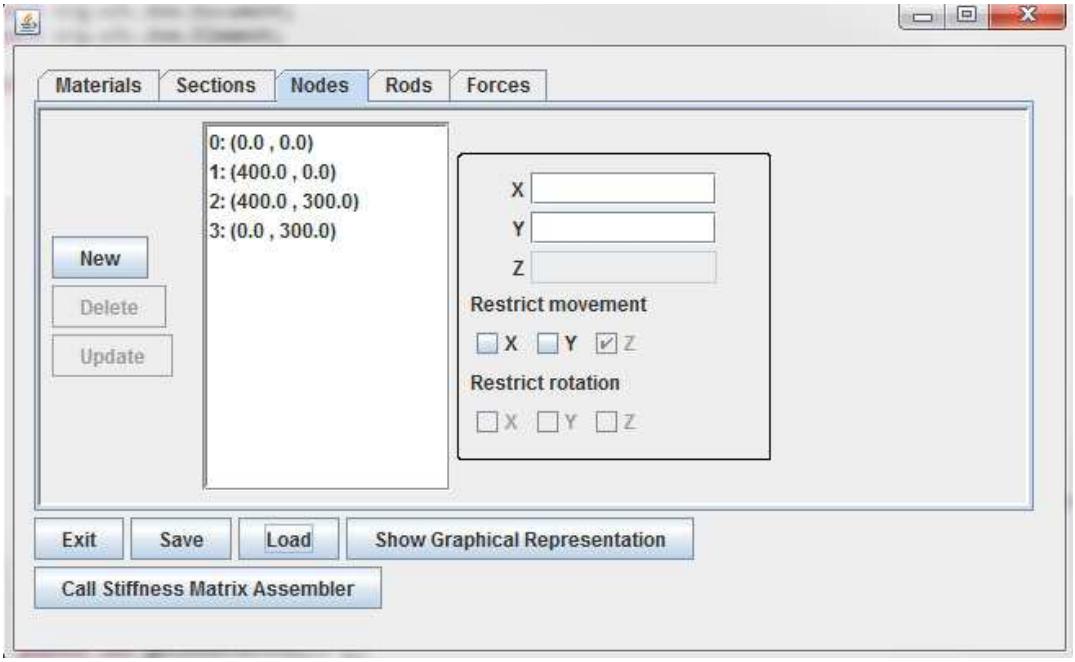
5.6.3 NodeHashMap.java

Αυτή η κλάση καθορίζει την δομή δεδομένων συλλογής για τους κόμβους. Όπως ειπώθηκε στην υποενότητα με τα υλικά (5.5.3), χρησιμοποιήθηκε σαν δομή δεδομένων συλλογής το HashMap. Έτσι με την ίδια λογική με τα υλικά που είδαμε προηγουμένως, έτσι και εδώ σαν key ορίζουμε το ID του κόμβου και σαν στοιχείο (element) ένα αντικείμενο της κλάσης `NodeData`.



Εικόνα 5–14

παρόμοιος τρόπος την αντίστοιχη κλάση για τα υλικά, παρατηρούμε (Εικόνα 5–14) κάποιες βασικές μεθόδους που κάνουν κάποιες κύριες λειτουργίες όπως προσθήκη, διαμόρφωση ή διαγραφή στοιχείων από το HashMap με τους κόμβους. Σε αυτό το σημείο, αξίζει να σημειωθεί πως στο GUI με τα δεδομένα,δηλαδή, στην κλάση frmMain, στην αντίστοιχη JList στο κάθε element που αφορά έναν κάμβο περιγράφεται ως εξής(Εικόνα 5–15):



Εικόνα 5–15

όπως φαίνεται στο πλαίσιο βρίσκεται η JList με τους κόμβους. Αυτή αφορά το GUI και όχι το HashMap. Βέβαια αντλεί τα δεδομένα από το HashMap με τους κόμβους. Όπως φαίνεται λοιπόν στην παραπάνω εικόνα(Εικόνα 5–15), κάθε element της JList, περιέχει το ID του κόμβου και τις συντεταγμένες (X,Y) του. Για αυτό το λόγο λοιπόν, στην κλάση `NodeHashMap` έχει οριστεί μία μέθοδος (`public int getStringElementID(String jListElement)`) που ανάλογα με τον ποιο κόμβο έχει επιλεγμένο ο χρήστης, αναγνωρίζει μέσω του ID που ξεκινάει το κάθε element, ποιον κόμβο έχει επιλεγμένο ο χρήστης. Έτσι και η αποφυγή σφάλματος επιτυχάνεται αλλά και η παροχή κάποιων πληροφοριών στον χρήστη που ενδεχομένως να τον βοηθήσουν σε κάποιες περιπτώσεις.

Η συνάρτηση NodeHashMap() είναι ο διαθέσιμος constructor της κλάσης. Η addNode(NodeData) χρησιμοποιείται για την προσθήκη μιάς καταχώρησης κόμβου. Η setNode(int, NodeData) χρησιμοποιείται για την τροποποίηση μιάς υπάρχουν καταχώρησης κόμβου. Η getStringElementID(String) χρησιμοποιείται για την επιστροφή του ID μιάς καταχώρησης από την λίστα των κόμβων που βρίσκεται στην καρτέλα με τους κόμβους. Η getNodesCount() επιστρέφει τον αριθμό των καταχωρήσεων των κόμβων. Η getNodeHashKey() επιστρέφει την τιμή του αριθμητή των κόμβων. Η removeNode(int) χρησιμοποιείται για την διαγραφή ενός κόμβου. Η prepareXMLElement(Document, HashMap<Integer, NodeData>) χρησιμποιείται για την προετοιμασία των κόμβων του project προς αποθήκευση σε έγγραφο xml. Η clear() χρησιμοποιείται για τον καθαρισμό των καταχωρήσεων των κόμβων.

5.7 eng.jTrussSolver.Section

5.7.1 Επισκόπηση πακέτου

Στο αυτό το πακέτο συλλέγονται τα δεδομένα σχετικά με τις διατομές των ράβδων. Έτσι έχουμε μία κλάση που ορίζει τα αντικείμενα με τα απαραίτητα χαρακτηριστικά για κάθε διατομή και μία άλλη κλάση που συλλέγει αυτά τα αντικείμενα σε ένα HashMap:

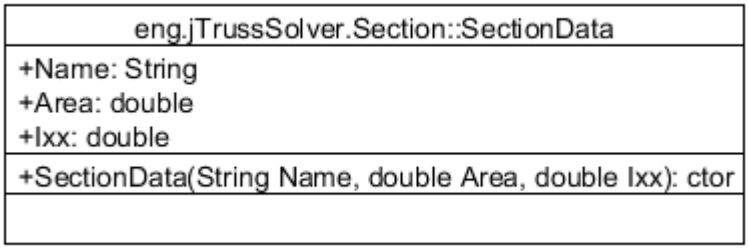
- **SectionData.java**
- **SectionHashMap.java**

Η SectionData.java έχει να κάνει με τα στοιχεία που συγκρατεί για κάθε διατομή και η SectionHashMap.java με τη δομή δεδομένων συλλογής όλων των διατομών.

5.7.2 SectionData.java

Όπως είπαμε στην εισαγωγή της υποενότητας(5.7.1), με αυτή την κλάση, κρατάμε αντικείμενα με τα βασικά χαρακτηριστικά για κάθε διατομή ράβδου. Έτσι κρατάμε τα εξής χαρακτηριστικά:

- ✓ Name (όνομα, αλφαριθμητικό)
- ✓ Area (επιφάνεια, δεκαδικός αριθμός)
- ✓ Ixx (ροπή αδράνειας, δεκαδικός αριθμός)

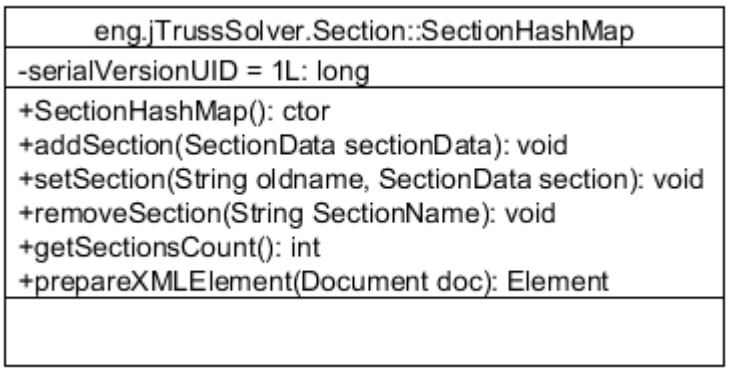


Εικόνα 5–16

Η συνάρτηση SectionData() είναι ο διαθέσιμος constructor της κλάσης.

5.7.3 SectionHashMap.java

Στο HashMap που χρησιμοποιήθηκε για την δομή δεδομένων συλλογής που αφορά τις διατομές των ράβδων, χρησιμοποιήθηκε σαν key το όνομα της διατομής και σαν στοιχείο το αντικείμενο της κλάσης `SectionData` που περιγράφει την διατομή.



Εικόνα 5–17

Η SectionHashMap() είναι ο διαθέσιμος constructor για την κλάση. Η addSection(SectionData) χρησιμοποιείται για την εισαγωγή μίας νέας καταχώρησης διατομής. Η setSection(String, SectionData) χρησιμοποιείται για την τροποποίηση μιάς διατομής. Η removeSection(String) χρησιμοποιείται για την διαγραφή της καταχώρησης μιάς διατομής. Η getSectionsCount() χρησιμοποιείται για να επιστρέφει τον αριθμό των καταχωρήσεων των διατομών. Η prepareXMLElement(Document) προετοιμάζει τις καταχωρήσεις των διατομών ώστε να αποθηκευτούν σε αρχείο xml.

5.8 eng.jTrussSolver.Force

5.8.1 Επισκόπηση πακέτου

Σε αυτό το πακέτο περιέχονται δύο κλάσεις. Στην μία ορίζεται η κλάση που θα ορίζει το αντικείμενο μίας δύναμης και στην άλλη κλάση, ορίζεται η δομή δεδομένων όπου θα αποθηκεύονται τα αντικείμενα με τις δυνάμεις. Οι κλάσεις είναι οι εξής:

- **ForceData.java**
- **ForceHashMap.java**

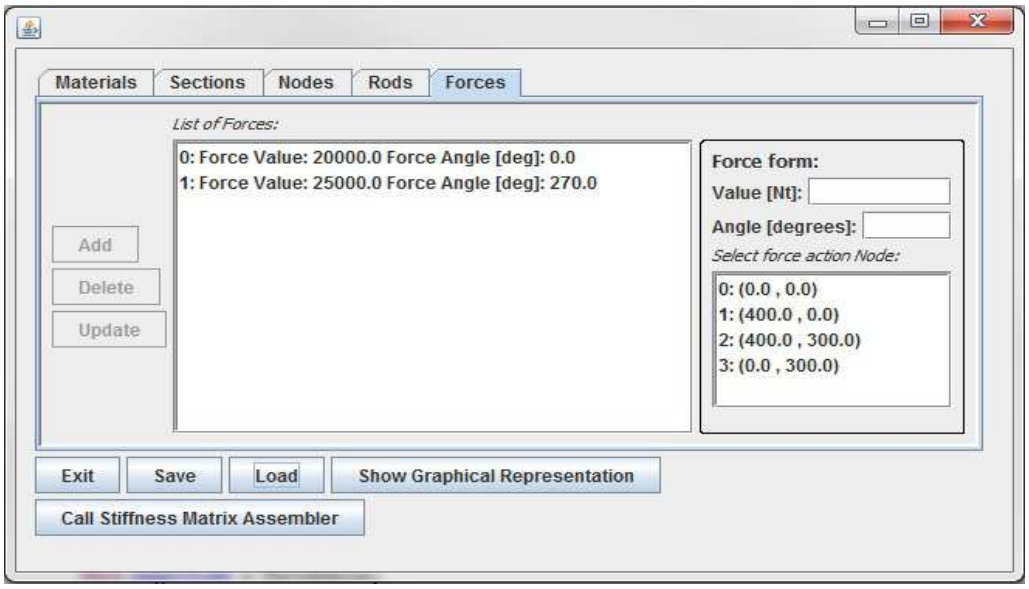
Η ForceData.java έχει να κάνει με τα στοιχεία που συγκρατεί για κάθε δύναμη και η ForceHashMap.java με τη δομή δεδομένων συλλογής όλων των δυνάμεων.

5.8.2 ForceData.java

Στην κλάση αυτή, όπως ειπώθηκε, ορίζονται τα αντικείμενα των δυνάμεων. Έχει οριστεί κάθε αντικείμενο να κρατάει τα εξής δεδομένα για κάθε δύναμη:

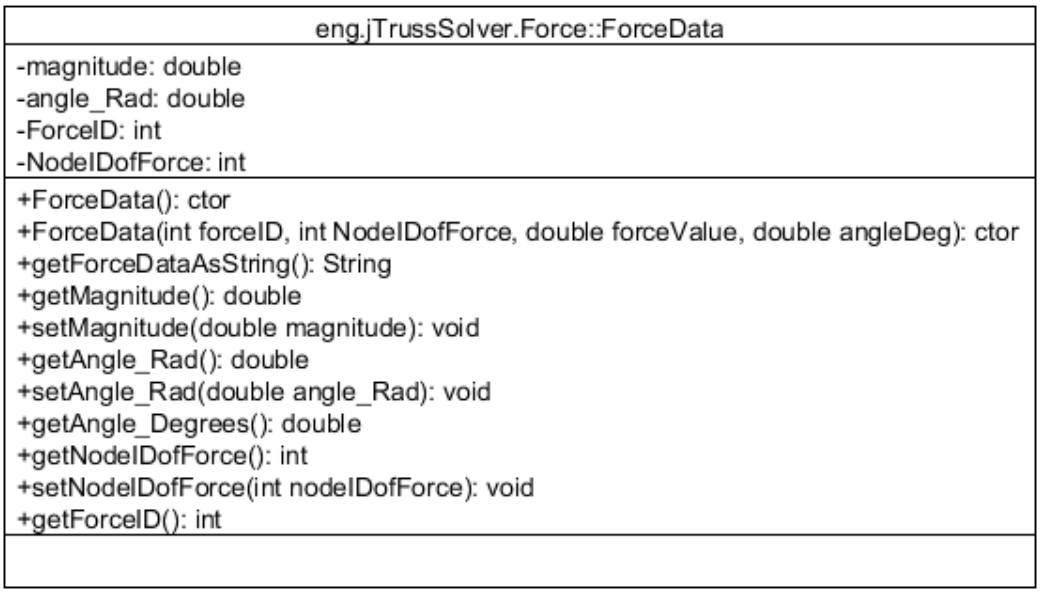
- ✓ forceID (αναγνωριστικό, ακέραιος αριθμός)
- ✓ NodeIDofForce (αναγνωριστικό του κόμβου, ακέραιος αριθμός)
- ✓ forceValue (ένταση της δύναμης, δεκαδικός αριθμός)
- ✓ angleDeg (γωνία της δύναμης σε ακτίνια, δεκαδικός αριθμός)

Ίδια λογική με τις προηγούμενες κλάσεις για τα υπόλοιπα δεδομένα, όμως εδώ αξίζει να παρατηρήσουμε κάποια επιπλέον κομμάτια του κώδικα. Πρώτα απ' όλα μπορεί εμείς να εισάγουμε τα δεδομένα της γωνίας σε μοίρες, αλλά πρέπει να μετατραπεί σε ακτίνια, ώστε να γίνουν σωστά οι πράξεις. Έτσι λοπόν η εξίσωση (angleDeg /180 *Math.PI) όπου angleDeg η γωνία σε μοίρες. Επίσης, ως NodeIDofForce ορίζεται το ID του κόμβου που ασκείται η δύναμη και το GUI που επιλέγει ο χρήστης τον κόμβο φαίνεται στην παρακάτω εικόνα:



Εικόνα 5–18

Στα δεξιά διακρίνονται τα πεδία όπου ο χρήστης εισάγει το μέτρο και την γωνία σε μοίρες της δύναμη και ακριβώς από κάτω στο μικρό λευκό πλαίσιο, διακρίνονται οι καταχωρημένοι κόμβοι εκείνη την χρονική στιγμή. Έτσι, αφού ο χρήστης συμπληρώσει σωστά τα πεδία του μέτρου και της γωνίας, έπειτα αν επιλέξει τον κόμβο όπου εφαρμόζεται η δύναμη, το JButton αριστερά με όνομα "Add" θα μπορεί να πατηθεί και τελικά ο χρήστης θα μπορεί να προσθέσει την δύναμη.

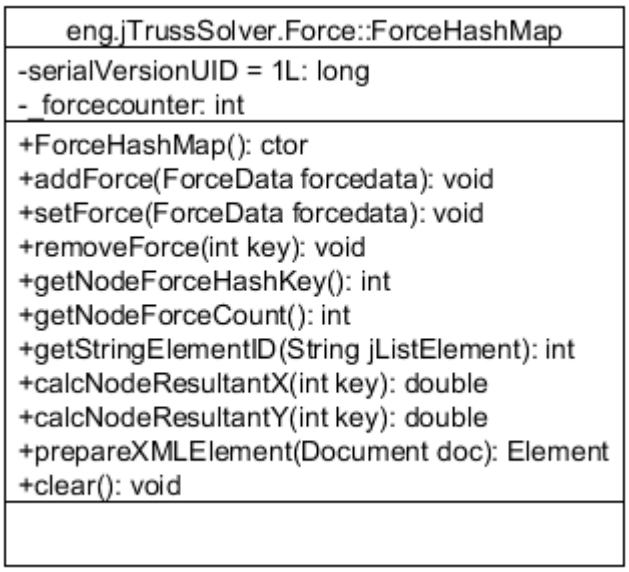


Εικόνα 5–19

Οι συναρτήσεις ForceData() είναι οι διαθέσιμοι constructors της κλάσης. Η getForceDataAsString() χρησιμοποιείται για την επιστροφή των στοιχείων μίας δύναμης σαν κείμενο για την συμπλήρωση της λίστας με τις δυνάμεις στην καρτέλα με τις δυνάμεις. Η getMagnitude() χρησιμοποιείται για την επιστροφή της έντασης μίας δύναμης. Η setMagnitude(double) χρησιμοποιείται για τον ορισμό της έντασης μίας δύναμης. Η getAngle_Rad() χρησιμοποιείται για την επιστροφή της γωνίας της δύναμης σε μοίρες. Η setAngle_Rad(double) χρησιμοποιείται για τον ορισμό της γωνίας μίας δύναμης σε μοίρες. Η getAngle_Degrees() χρησιμοποιείται για την επιστροφή της γωνίας μίας δύναμης σε ακτίνια. Η getNodeIDofForce() χρησιμοποιείται για την επιστροφή του ID του κόμβου στον οποίο ασκείται μία δύναμη. Η setNodeIDofForce(int) χρησιμοποιείται για την καταχώρηση του ID του κόμβου στον οποίο ασκείται μία δύναμη. Η getForceID() χρησιμοποιείται για την επιστροφή του ID μίας δύναμης.

5.8.3 ForceHashMap.java

Σε αυτή την κλάση ορίζεται η δομή δεδομένων συλογής των δυνάμεων. Έτσι, σαν key στο HashMap έχει οριστεί το forceID και σαν στοιχεία (elements) τα αντικείμενα της κλάσης `ForceData`.



Εικόνα 5–20

Και σε αυτό το κομμάτι κώδικα υπάρχει κατάλληλη μέθοδος, ώστε επιλέγοντας ο χρήστης από την JList με τις δυνάμεις, το πρόγραμμα, μέσω του forceID που είναι ο πρώτος ακέραιος στο element της JList στο GUI με τις δυνάμεις, να αναγνωρίζει ποια δύναμη έχει επιλεγμένη ο χρήστης. Επίσης εδώ υπάρχουν δύο μέθοδοι, οι οποίες υπολογίζουν και επιστρέφουν τη συνισταμένη σε κάθε άξονα (Χ και Ψ), αυτές οι μέθοδοι χρησιμοποιούνται στη κλάση με τους υπολογισμούς.

Η ForceHashMap() είναι ο διαθέσιμος contructor της κλάσης. Η addForce(ForceData) χρησιμποιείται για την καταχώρηση μίας δύναμης. Η setForce(ForceData) χρησιμποιείται για την τροποποίηση μίας καταχώρησης μίας δύναμης. Η removeForce(int) χρησιμοποιείται για την διαγραφή μίας καταχώρησης μίας δύναμης. Η getNodeForceHashKey() χρησιμοποιείται για την επιστροφή του αριθμητή των δυνάμεων. Η getNodeForceCount() χρησιμοποιείται για την επιστροφή του αριθμού των καταχωρημένων δυνάμεων. Η getStringElementID(String) χρησιμοποιείται για την επιστροφή του ID της δύναμης από ένα στοιχείο της λίστας με τις δυνάμεις στην καρτέλα των δυνάμεων. Η calcNodeResultantX(int) χρησιμοποιείται για τον υπολογισμό της συνισταμένης Χ των δυνάμεων που ασκούνται σε έναν κόμβο. Η calcNodeResultantY(int) χρησιμοποιείται για τον υπολογισμό της συνισταμένης Υ των δυνάμεων που ασκούνται σε έναν κόμβο. Η prepareXMLElement(Document) χρησιμοποιείται για την προετοιμασία των καταχωρήσεων των δυνάμεων για αποθήκευση σε αρχείο XML. Η clear() χρησιμοποιείται για την εκκαθάριση των καταχωρήσεων των δυνάμεων στη συλλογή με τις δυνάμεις, καθώς και τον μηδενισμό του αριθμητή αυτών.

5.9 eng.jTrussSolver.Rod

5.9.1 Επισκόπηση πακέτου

Σε αυτό το πακέτο ορίζονται οι ράβδοι. Η ράβδοι συνθέτουν όλες τις συλλογές δεδομένων, εκτός από αυτήν με τις δυνάμεις, ώστε να οριστούν τα αντικείμενα για την κάθε ράβδο. Με λίγα λόγια αν δεν έχουμε ορίσει κάποια καταχώρηση σε κάποια συλλογή από τις τρείς (κόμβων, υλικών ή διατομών) δεν είναι δυνατόν να καταχωρήσουμε κάποιο στοιχείο ράβδου. Ποιο συγκεκριμμένα, αυτό το πακέτο περιέχει δύο κλάσεις. Οι οποίες είναι οι εξής:

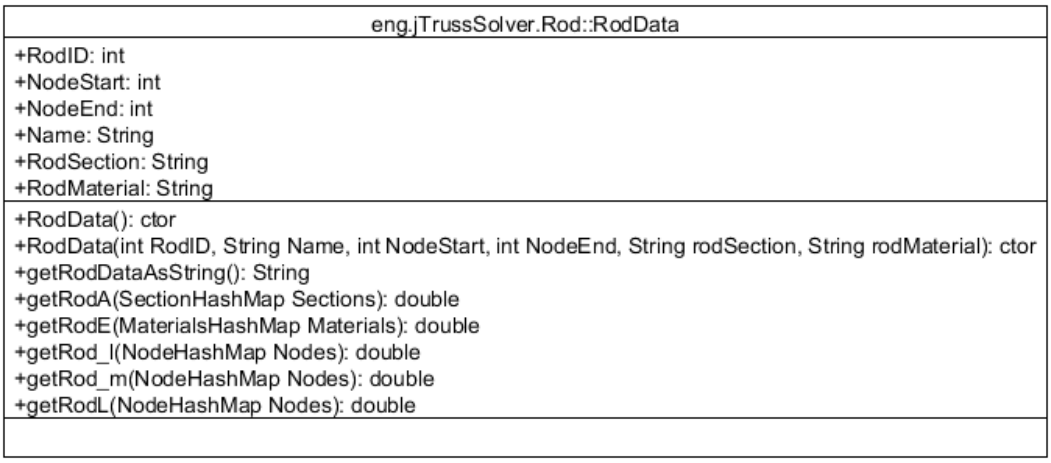
- **RodData.java**
- **RodHashMap.java**

Η RodData.java έχει να κάνει με τα στοιχεία που συγκρατεί για κάθε ράβδο και η RodHashMap.java με τη δομή δεδομένων συλλογής όλων των ράβδων.

5.9.2 RodData.java

Σε αυτήν την κλάση, έχει οριστεί το κάθε αντικείμενο να ορίζεται από αυτά τα χαρακτηριστικά που είναι απαραίτητα για τους υπολογισμούς σχετικά με τις ράβδους. Έτσι κάθε αντικείμενο αυτής της κλάσης, συγκρατεί τα ακόλουθα στοιχεία για την ράβδο:

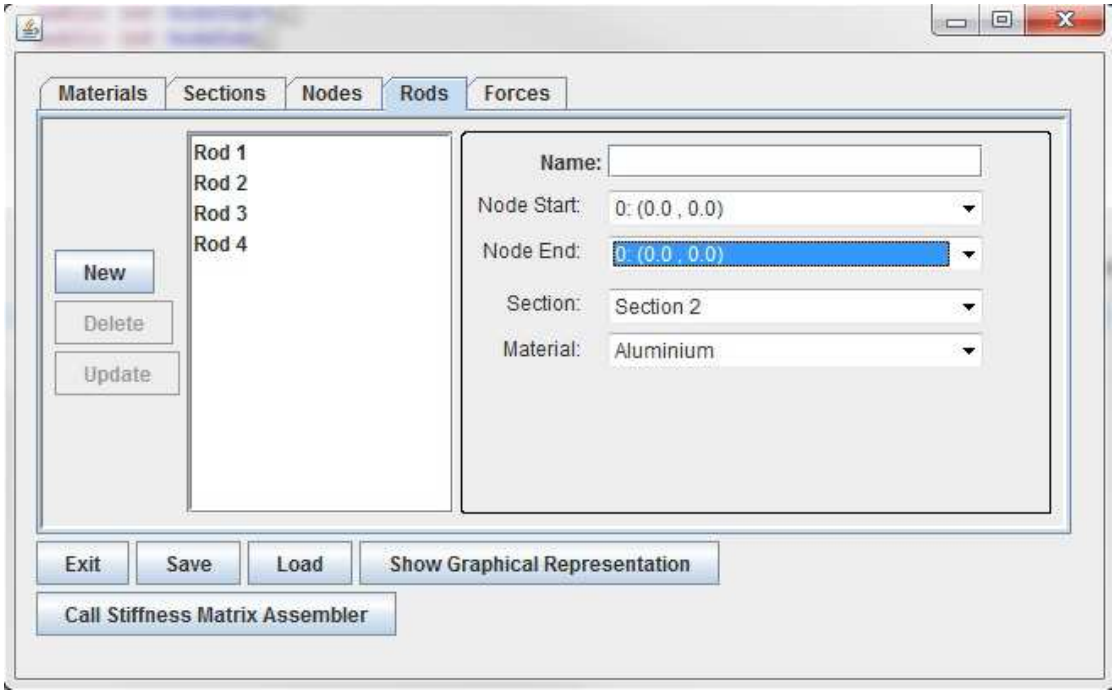
- ✓ RodID (αναγνωριστικό, ακέραιος αριθμός)
- ✓ Name (όνομα, αλφαριθμητικό)
- ✓ NodeStart (αναγνωριστικό κόμβου αρχής, ακέραιος αριθμός)
- ✓ NodeEnd (αναγνωριστικό κόμβου τέλους, ακέραιος αριθμός)
- ✓ rodSection (αναγνωριστικό διατομής, ακέραιος αριθμός)
- ✓ rodMaterial (αναγνωριστικό υλικού, ακέραιος αριθμός)



Εικόνα 5–21

Με την ίδια λογική όπως και στις προηγούμενες περιπτώσεις. Εδώ επιπλέον παρατηρούμε κάποιες μεθόδους που υπολογίζουν και επιστρέφουν διάφορα γεωμετρικά στοιχεία για τις ράβδους. Αυτό διευκολύνει στους υπολογισμούς και στην καλύτερη οργάνωση των

κλάσεων, αλλά γίνεται ευκολότερη και συντομότερη η διόρθωση κάποιου λάθους, ενδεχομένως, από την πλευρά του προγραμματιστή. Παρακάτω (Εικόνα 5–22) φαίνεται η όψη από το GUI με τις ράβδους στην κλάση `frmMain`:



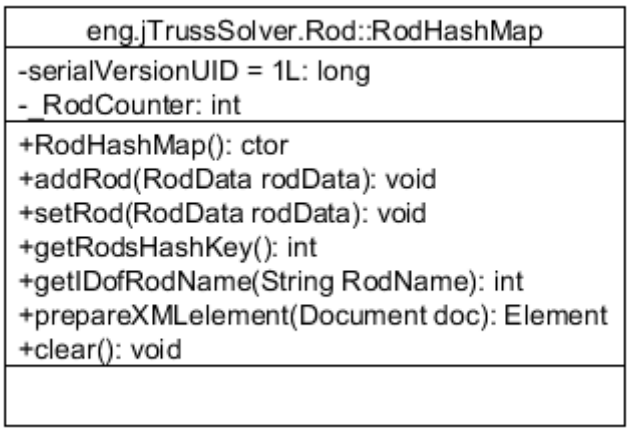
Εικόνα 5–22

Όταν ο χρήστης επιλέξει κάποιο στοιχείο από την λίστα (π.χ. Rod 1,Rod 2 κτλ) τότε το πεδίο με το όνομα στα δεξιά πάνω καθώς και τα choice list από κάτω από αυτό το πεδίο, συμπληρώνουν και επιλέγουν, αντίστοιχα, τα δεδομένα του επιλεγμένου στοιχείου (ράβδου) και ταυτόχρονα ενεργοποιούνται και είναι διαθέσιμα για χρήση τα JButtons (Delete, Update).

Οι συναρτήσεις RodData() είναι οι διαθέσιμοι constructors της κλάσης. Η getRodDataAsString() χρησιμοποιείται για την δημιουργία και επιστροφή των στοιχείων μίας ράβδου σε μορφή κειμένου. Η getRodA(SectionHashMap) χρησιμοποιείται για την επιστροφή της διατομής μίας ράβδου. Η getRodE(MaterialsHashMap) χρησιμοποιείται για την επιστροφή του μέτρου ελαστικότητας μίας ράβδου. Η getRod_I(NodeHashMap) χρησιμοποιείται για τον υπολογισμό και την επιστροφή της τετμημένης μίας ράβδου. Η getRod_m(NodeHashMap) χρησιμοποιείται για τον υπολογισμό και την επιστροφή της τεταγμένης μίας ράβδου. Η getRodL(NodeHashMap) χρησιμοποιείται για τον υπολογισμό και την επιστροφή του μήκους μίας ράβδου.

5.9.3 RodHashMap.java

Σε αυτή την κλάση ορίζεται η δομή δεδομένων συλλογής των ράβδων. Ορίζεται δηλαδή το HashMap με τη συλλογή των ράβδων που έχουν σαν key το RodID της κάθε ράβδου και σαν στοιχεία αντικείμενα της κλάσης `RodData` για την κάθε ράβδο.



Εικόνα 5–23

Η συνάρτηση RodHashMap() είναι ο διαθέσιμος constructor της κλάσης. Η addRod(RodData) χρησιμοποιείται για την καταχώρηση μίας ράβδου. Η setRod(RodData) χρησιμοποιείται για την τροποποίηση μίας καταχωρημένης ράβδου. Η getRodsHashKey() χρησιμοποιείται την επιστροφή του αριθμητή από την συλλογή των ράβδων. Η getIDofRodName(String) χρησιμοποιείται για την επιστροφή του ID μίας καταχώρησης ράβδου από τη λίστα στην καρτέλα με της ράβδους. Η prepareXMLelement(Document, HashMap<Integer, RodData>) χρησιμοποιείται για την προετοιμασία των καταχωρήσεων των ράβδων για αποθήκευση σε αρχείο XML. Η clear() χρησιμοποιείται για την εκκαθάριση της συλλογής με τις ράβδους καθώς και για τον μηδενισμό του αριθμητή αυτών.

5.10 eng.jTrussSolver.Solver

5.10.1 Επισκόπηση πακέτου

Αυτό το πακέτο περιέχει δύο σημαντικές κλάσεις για το πρόγραμμα και είναι οι εξής:

- **truss2dProblemDef.java**
- **StiffnessMatrixAssembler.java**

Η κλάση truss2dProblemDef.java είναι κομβικής σημασίας μιάς και τα αντικείμενα των συλλογών δεδομένων του δικτυώματος, συμπεριλαμβάνονται στο αντικείμενο αυτής της κλάσης. Όλες οι κλάσεις που πραγματοποιούν υπολογισμούς, αντλούν τα δεδομένα των στοιχείων του δικτυώματος από αυτήν την κλάση. Η κλάση StiffnessMatrixAssembler.java είναι εξίσου σημαντική, μιάς και σε αυτήν πραγματοποιούνται όλες οι πράξεις που πρέπει να γίνουν για την επίλυση του δικτυώματος.

5.10.2 truss2dProblemDef.java

Στη κλάση αυτή ορίζονται όλα τα αντικείμενα των συλλογών των δεδομένων.

truss2dProblemDef.java
- truss2dProblemDef()
- truss2dProblemDef(NodeHashMap, RodHashMap, SectionHashMap, MaterialsHashMap, ForceHashMap, NodeDisplacementHashMap, RodDisplacementHashMap)
- getRodDisplacements():RodDisplacementHashMap
- getNodeDisplacements():NodeDisplacementHashMap
- getMaterials():MaterialsHashMap
- getSections():SectionHashMap
- getNodes():NodeHashMap
- getForces():ForceHashMap
- getRods():RodHashMap
- CreateDataXML(String):void
- createDataXMLJAXB(String):void
- clearProblem():void

Πίνακας 5.3

Οι συναρτήσεις truss2dProblemDef() είναι οι διαθέσιμοι constructors της κλάσης. Η getRodDisplacements() χρησιμοποιείται για την επιστροφή της συλλογής με τις ράβδους που έχουν μετακινηθεί. Η getNodeDisplacements() χρησιμοποιείται για την επιστροφή της συλλογής με τους κόμβους που έχουν μετακινηθεί. Η getMaterials() χρησιμοποιείται για την επιστροφή της συλλογής με τα υλικά. Η getSections() χρησιμοποιείται για την επιστροφή της συλλογής με τις διατομές. Η getNodes() χρησιμοποιείται για την επιστροφή της συλλογής με τους κόμβους. Η getForces() χρησιμοποιείται για την επιστροφή της συλλογής με τις δυνάμεις. Η getRods() χρησιμοποιείται για την επιστροφή της συλλογής με τις ράβδους στην αρχική τους θέση και κατάσταση. Η CreateDataXML(String) χρησιμοποιείται για την αποθήκευση του project σε αρχείο XML. Η createDataXMLJAXB(String) χρησιμοποιείται για την αποθήκευση του project με την χρήση της βιβλιοθήκης JAXB. Η clearProblem() χρησιμοποιείται για τον καθαρισμό και τον μηδενισμό των μετρητών-αριθμητών όλων των συλλογών του project.

5.10.3 StiffnessMatrixAssembler.java

Σε αυτήν την κλάση γίνεται η επίλυση του δικτυώματος. Δημιουργούνται όλα τα αποτελέσματα που είναι προς έξοδο.

Εικόνα 5–24

Η StiffnessMatrixAssembler(truss2dProblemDef) είναι ο διαθέσιμος constructor της κλάσης. Η calcReactions(truss2dProblemDef, int[], double[][], double[][]) χρησιμοποιείται για τον υπολογισμό των αντιδράσεων στις στηρίξεις του δικτυώματος. Η calcStresses(truss2dProblemDef, Matrix) χρησιμοποιείται για τον υπολογισμό των τάσεων στις ράβδους. Η getDisplacementMatrix_q(HashMap<Integer, NodeData>, RodData, Matrix) χρησιμοποιείται για τον υπολογισμό και την επιστροφή του πίνακα με τις μετακινήσεις των κόμβων. Η calcStructureDisplacement(int[], double[][]) χρησιμοποιείται για τον υπολογισμό και την επιστροφή ενός matrix που αναπαριστά τους βαθμούς ελευθερίας (με 1 είναι ελεύθερη η μετακίνηση). Η calcRestricts(int[]) χρησιμοποιείται για τον υπολογισμό των αντιδράσεων στις στηρίξεις. Η printRodStiffnessMatrix(RodData, NodeData, NodeData, SectionData, MaterialData) χρησιμοποιείται για την εμφάνιση των αποτελεσμάτων των μητρώων δυσκαμψίας για κάθε ράβδο. Η calcStiffnessMatrixRod(RodData, NodeData, NodeData, SectionData, MaterialData) χρησιμοποείται για τον υπολογισμό των μητρώων δυσκαμψίας για κάθε ράβδο του δικτυώματος. Η calcMasterK(RodData, NodeData, NodeData, SectionData, MaterialData, double[][]) χρησιμοποιείται για τον υπολογισμό του τελικού μητρώου δυσκαμψίας στο τοπικό σύστημα αναφοράς. Η calcFinalStiffnessMatrix(truss2dProblemDef, double[][]) χρησιμοποιείται για τον υπολογισμό του τελικού μητρώου δυσκαμψίας στο γενικό σύστημα αναφοράς. Η calcRestrictArrays(int[], int) χρησιμοποιείται για την δημιουργία ενός πίνακα ακεραίων που αναπαριστά τους δείκτες όπου δεν έχουμε περιορισμό στην κίνηση. Η calcFinalForcesMatrix(truss2dProblemDef, double[], int, int[], int[]) χρησιμοποιείται για τον υπολογισμό ενός πίνακα δεκαδικών που αναπαριστά τις δυνάμεις στους βαθμούς ελευθερίας (αυτός ο πίνακας χρησιμοποιείται στους υπολογισμούς). Η calcDisplacementsMatrix(truss2dProblemDef, double[][], double[]) χρησιμοποιείται για τον υπολογισμό και τη δημιουργία ενός matrix όπου αναπαρίσταται οι μετακινήσεις των

κόμβων (και αυτό το matrix επίσης χρησιμοποιείται στους υπολογισμούς). Η FillNodeDisplacementHashMap() χρησιμοποιείται για την συμπλήρωση της συλλογής με τους μετακινούμενους κόμβους (έχει δημιουργηθεί μια νέα συλλογή για τους κόμβους αυτούς). Η FillRodDisplacementHashMap() χρησιμοποιείται για την συμπλήρωση της συλλογής με τις μετακινούμενες ράβδους.

6 Συμπεράσματα

6.1 Προσωπική ανασκόπηση/σκέψεις

Παρόλο που οι επαφή μου με τους υπολογιστές είναι πολύ καλή, μέσα από την πτυχιακή έμαθα πως η χρήση του υπολογιστή είναι πολύ διαφορετική όταν την βλέπεις από την σκοπιά του προγραμματιστή. Κατά την επιλογή του θέματος οι γνώσεις μου για την Java και τον προγραμματισμό σταματούσαν στο λύκειο και σε κάποια ψευδογλώσσα που μάθαμε εκεί. Αυτό με βοήθησε στο να είμαι αισιόδοξος στην πορεία της παρούσας πτυχιακής. Όσον αφορά το κομμάτι της μηχανικής και πάλι απείχα αρκετά από την μέθοδο που χρησιμοποιήθηκε καθώς οι γνώσεις μου σταματούσαν σε ότι διδάκτηκα από το μάθημα της τεχνικής μηχανικής και αντοχής των υλικών.

Με βάση τα παραπάνω, οι δυσκολίες που συνάντησα κατά την διάρκεια της σύνταξης του προγράμματος ήταν αρκετές. Αυτές δεν περιορίστηκαν στις γνώσεις μου όσον αφορά την μέθοδο επίλυσης ή στο κομμάτι του προγραμματισμού, αλλά και σε ψυχολογικό επίπεδο.

Σε όλες αυτές τις δυσκολίες καθοριστικός παράγοντας στο να ξεπεραστούν ήταν ο εισηγητής καθηγητής μου **κ.Παπαδάκης Νικόλαος** καθώς και η οικογένεια μου που με στήριξε και μου έδωσε την απαραίτητη ψυχολογική ώθηση να προχωρήσω και να φέρω εις πέρας την πτυχιακή εργασία. Θέλω να τους ευχαριστήσω γι' όλα.

6.2 Αποτέλεσμα πτυχιακής

Ο κώδικας που γράφτηκε έχει περιθώρια εξέλιξης, κάποια πιο άμεσα και κάποια πιο περίπλοκα. Έγινε προσπάθεια να μπορεί να πραγματοποιεί κάποιες βασικές λειτουργίες όπως η ορθή εισαγωγή των δεδομένων του δικτυώματος και η γραφική αναπαράσταση του. Επίσης δόθηκε προσοχή στην αποφυγή εισαγωγής λάθος δεδομένων και συντακτικά και λογικά. Επίσης έγινε προσπάθεια να είναι το GUI φιλικό προς τον χρήστη.

Εκτελώντας κανείς το πρόγραμμα αντικρίζει ένα GUI το οποίο περιέχει καρτέλες (tabs) και κάποια κουμπιά (buttons). Κάθε καρτέλα αναφέρεται σε κάποια ομάδα δεδομένων όπως υλικά (materials), διατομές (sections), κόμβοι (nodes), ράβδοι (rods) και δυνάμεις (forces). Οι καρτέλες αυτές του δίνουν την δυνατότητα να εισάγει όλα τα απαραίτητα δεδομένα που χρειάζονται να την επίλυση του δικτυώματος. Τα κουμπιά εκτελούν όλες τις βασικές λειτουργίες όπως αποθήκευση ή φόρτωση κάποιου project, ανοίγμα της γραφικής αναπαράστασης του δικτυώματος, επίλυση του δικτυώματος ή έξοδος από το πρόγραμμα. Επίσης υπάρχουν και σε κάθε καρτέλα αντίστοιχα κουμπιά που εκτελούν κάποιες λειτουργίες που αφορούν την διαχείριση των αντίστοιχων δεδομένων όπως προσθήκη, διαγραφή ή τροποποίηση κάποιας καταχώρησης.

Το πρόγραμμα μπορεί να αποθηκεύει το project σε αρχείο xml καθώς και να φορτώνει κάποιο έτοιμο project από αρχείο xml. Έχει προβλεφθεί να εμφανίζει αντίστοιχα μηνύματα σε περίπτωση που εισαχθεί κάποιο λάθος στοιχείο π.χ. εισαγωγή κειμένου σε αριθμητικό πεδίο ή ακόμα και κάποιο λογικό λάθος, όπως, σύνδεση δύο ήδη συνδεμένων κόμβων με

κάποια νέα ράβδο. Στην τελευταία περίπτωση θα εμφανίσει κάποιο μήνυμα σφάλματος και αυτομάτως δεν θα πραγματοποιηθεί η λειτουργία την προσθήκης της ράβδου.

Πατώντας το κουμπί επίλυσης του δικτυώματος που έχει περιγραφεί, το πρόγραμμα το επιλύει και στη συνέχεια εμφανίζει ένα παράθυρο με την γραφική αναπαράσταση της μορφής του τελικού δικτυώματος. Με τον όρο «επίλυση» νοείται η εύρεση των επιμέρους μητρώων δυσκαμψίας των ράβδων αλλά και του τελικού μητρώου δυσκαμψίας του δικτυώματος, ο υπολογισμός των μετατοπίσεων των κόμβων, των τάσεων στις ράβδους του δικτυώματος και τέλος ο υπολογισμός των αντιδράσεων στις στηρίξεις.

Συγκεντρωτικά, το πρόγραμμα που έχει γραφτεί έχει τα ακολουθα χαρακτηριστικά

- Επιλύει δισδιάστατα δικτυώματα, χρησιμοποιώντας την μέθοδο των μετατοπίσεων των κόμβων.
- Πρακτικά δεν υπάρχει περιορισμός στο μέγεθος του δικτυώματος. Περιοριζόμαστε μόνο από την μνήμη του υπολογιστή.
- Μπορεί να ελέγχει τα δεδομένα που εισάγονται από τον χρήστη για κάθε στοιχείο του GUI όπως υλικά, ράβδοι, κόμβοι, δυνάμεις και διατομές. Σε κάθε περίπτωση έχει προβλεφθεί να εμφανίζει αντίστοιχο μήνυμα σφάλματος και να δίνει στον χρήστη την αντίστοιχη πληροφορία για το σφάλμα που προέκυψε.
- Μπορεί να αναπαριστά γραφικά την μορφή του δικτυώματος που έχει εισαχθεί από τον χρήστη, πριν και μετά την επίλυση του.
- Επίσης μπορεί να αποθηκεύει και να φορτώνει έτοιμα project που είναι σε μορφή XML.
- Ο χρήστης επιπλέον έχει την δυνατότητα να τροποποιεί κάθε στοιχείο που έχει εισάγει καθώς και να διαγράφει κάποιο από αυτά.
- Μπορούν να εισαχθούν φορτία υπό γωνία.

6.3 Περιορισμοί

Το παρόν πρόγραμμα έχει περιθώρια εξέλιξης, πράγμα που σημαίνει ότι στην παρούσα μορφή έχει αδυναμίες και κατ' επέκταση περιορισμούς.

Βασικοί περιορισμοί είναι ότι :

- Τέλος, βασικός περιορισμός είναι ότι περιορίζεται στις δύο διαστάσεις, (αν και έχει γίνει πρόβλεψη στην εισαγωγή δεδομένων για τρισδιάστατα προβλήματα)
- δεν μπορούν να εισαχθούν σε αυτό στηρίξεις υπο γωνία, όπως επίσης και κατανεμημένα φορτία στις ράβδους.
- Όσον αφορά την γραφική αναπαράσταση, δεν είναι τόσο «ευέλικτο», δηλαδή δεν δίνει πολλές δυνατότητες στον χρήστη(πχ. Zoom in, zoom out).
- Επίσης, δεν δίνει επιλογές στον χρήστη για την εισαγωγή δεδομένων όσον αφορά στις μονάδες.
- Η έξοδος των αποτελεσμάτων θα μπορούσε να ήταν αρκετά καλύτερη και πιο σαφής, πιο κατατοπιστική για τον χρήστη.

6.4 Περαιτέρω δουλειά – Επόμενα βήματα

Κάποιες άμεσες αλλαγές που θα μπορούσαν να γίνουν είναι η βελτίωση της γραφικής αναπαράστασης με προσθήκη απλών δεδομένων σε αυτήν, πρό και μετά υπολογισμών. Επιπλέον θα μπορούσε να βελτιωθεί σημαντικά το GUI με την προσθήκη παραμέτρων στις μονάδες όπου ο χρήστης εισάγει τα δεδομένα ή ακόμα και στην καλύτερη έξοδο των αποτελεσμάτων.

Κάποιες πιο περίπλοκες αλλαγές που θα βελτίωναν το πρόγραμμα ήταν η βελτίωση του κώδικα επίλυσης του δικτυώματος και της μορφής των εξαγώμενων αποτελεσμάτων. Η αναδιάρθρωση των κλάσεων έτσι ώστε να είναι πιο «ελαφρύ» και πιο κατανοητό σε κάποιον προγραμματιστή που θέλει να διαβάσει τον κώδικα. Επίσης θα μπορούσαν να εισαχθούν κάποιες πηγές δεδομένων, όπως, μία βιβλιοθήκη με υλικά, όπου ο χρήστης θα ήταν σε θέση να επιλέγει και να εισάγει.

7 Βιβλιογραφία - Πηγές

- http://docs.oracle.com/
- Java - Μία Εισαγωγή στην Επίλυση Προβλημάτων & στον Προγραμματισμό (Walter Savitch) – Εκδόσεις Τζιόλα
- XML Documentation - http://www.roseindia.net/xml/
- XML Documentation - http://www.javaprogrammingforums.com
- http://www.dmst.aueb.gr/dds/isdi/
- www.wikipedia.org
- Java2D - http://www.apl.jhu.edu/~hall/java/Java2D-Tutorial.html
- http://stackoverflow.com
- http://www.semfe.gr/
- http://mindprod.com/jgloss/hashmap.html
- http://www.java-forum.org/
- http://download.java.net
- http://www.softlab.ntua.gr
- http://web.itu.edu.tr/~mecit/uum508e_resources.htm
- http://www.colorado.edu/engineering/cas/courses.d/IFEM.d/
- Introduction to Finite Elements in Engineering – Tirupathi R. Chandrupatla , Ashok D. Belegundu
- http://www.eclipse.org/documentation/

8 Παράρτημα (SOURCE CODE)

8.1 Eng.TrussSolver.Rod

8.1.1 RodData

```
package eng.jTrussSolver.Rod;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlType;
import eng.jTrussSolver.Material.MaterialsHashMap;
import eng.jTrussSolver.Node.NodeHashMap;
import eng.jTrussSolver.Section.SectionHashMap;

@XmlType(name = "Rod", propOrder = { "RodID", "NodeStart","NodeEnd","Name",
"RodSection", "beamMaterial" })
@XmlAccessorType(XmlAccessType.FIELD)
public class RodData {
	@XmlElement
	public int RodID;
	@XmlElement
	public int NodeStart;
	@XmlElement
	public int NodeEnd;
	@XmlElement
	public String Name;
	@XmlElement
	public String RodSection;
	@XmlElement
	public String RodMaterial;



	public RodData ()
	{}

	public RodData (int RodID,String Name,int NodeStart,int NodeEnd,String
rodSection,String rodMaterial) {

		this.RodID = RodID;
		this.Name = Name;
		this.NodeStart = NodeStart;
		this.NodeEnd = NodeEnd;
		this.RodSection = rodSection;
		this.RodMaterial = rodMaterial;
	}

	public String getRodDataAsString()
	{
		String str="";
		str += "ID:"+ RodID+"\t";
		str += "Name:"+ Name+"\t";
		str += "N.Start:"+ NodeStart+"\t";
		str += "N.End:"+ NodeEnd+"\t";
		str += "Section:"+ RodSection+"\t";
		str += "Material:"+ RodMaterial;
		return str;

	}


	public double getRodA(SectionHashMap Sections) {
		return Sections.get(this.RodSection).Area;
	}
```

```
	public double getRodE(MaterialsHashMap Materials) {
		return  Materials.get(this.RodMaterial).E;
	}

	/**
	 * Calculate Beam cos with horizontal axis
	 * @param Nodes
	 * @return cos
	 */
	public double getRod_l(NodeHashMap Nodes) {

		double temp = 0;
		int N1 = this.NodeStart;
		int N2 = this.NodeEnd;
		double X1 = Nodes.get(N1).X;
		double X2 =Nodes.get(N2).X;

		temp = (X2-X1)/this.getRodL(Nodes);

		return temp;
	}

	public double getRod_m(NodeHashMap Nodes) {

		int N1 = this.NodeStart;
		int N2 = this.NodeEnd;
		double Y1 = Nodes.get(N1).Y;
		double Y2 = Nodes.get(N2).Y;
		return (Y2-Y1)/this.getRodL(Nodes);
	}

	/**
	 * calculates the beam length
	 * @param Nodes
	 * @return
	 */
	public double getRodL(NodeHashMap Nodes) {

		int N1 = this.NodeStart;
		int N2 = this.NodeEnd;
		double X1 = Nodes.get(N1).X;
		double Y1 = Nodes.get(N1).Y;
		double X2 = Nodes.get(N2).X;
		double Y2 = Nodes.get(N2).Y;
		return  Math.sqrt(  (Math.pow((X2-X1),2))+(Math.pow((Y2-Y1),2))
);
	}

}
```

8.1.2 RodHashMap.java

```
package eng.jTrussSolver.Rod;

import java.util.HashMap;
import javax.swing.JOptionPane;
import org.w3c.dom.Document;
import org.w3c.dom.Element;

public class RodHashMap extends HashMap<Integer, RodData> {

	/**
	 *
	 */
	private static final long serialVersionUID = 1L;
```

```
	private int _RodCounter ;

	public RodHashMap()
	{
		_RodCounter=0;
	}



	/**
	 * adds rods at the end of rods list
	 * @param Rod
	 */
	public void addRod(RodData rodData) {
		this.put(_RodCounter, rodData);
		_RodCounter =_RodCounter+1;
	}

	public void setRod(RodData rodData) {
		this.remove(rodData.RodID);
		this.put(rodData.RodID, rodData);
	}

	public int getRodsHashKey() {
		return _RodCounter;
	}

	public int getIDofRodName(String RodName) {
		int RodID = 0;
		for (Integer Key : this.keySet()) {
			String HashKeyName = this.get(Key).Name;
			if (HashKeyName.equals(RodName)) {
				RodID = Key;
			}
		}
		return RodID;
	}

	/**
	 * Code For Rods Data
	 * @param doc
	 * @return
	 */
	public Element prepareXMLelement(Document doc) {
		int i;
		Element B = doc.createElement("Rods");

		for (i=0; i<this.size(); i++) {

			Element RodElement = doc.createElement("Rod");
			B.appendChild(RodElement);

			Element g = doc.createElement("RodID");

	g.appendChild(doc.createTextNode(String.valueOf((this.get(i).RodID))))
;
			RodElement.appendChild(g);

			Element el = doc.createElement("Name");
			el.appendChild(doc.createTextNode(this.get(i).Name));
			RodElement.appendChild(el);

			Element sn = doc.createElement("Start_Node");

	sn.appendChild(doc.createTextNode(String.valueOf(this.get(i).NodeStart
)));
			RodElement.appendChild(sn);
```

```
                    Element en = doc.createElement("End_Node");

      en.appendChild(doc.createTextNode(String.valueOf(this.get(i).NodeEnd))
);
                    RodElement.appendChild(en);

                    Element m = doc.createElement("RodMaterial");

      m.appendChild(doc.createTextNode(String.valueOf(this.get(i).RodMateria
l)));
                    RodElement.appendChild(m);

                    Element s = doc.createElement("RodSection");

      s.appendChild(doc.createTextNode(String.valueOf(this.get(i).RodSection
)));
                    RodElement.appendChild(s);

             }
             return B;
      }

      /**
       * Clears() Hashmap and also resets counter.
       */
      public void clear()
      {
             super.clear();
             this._RodCounter =0;
      }

}
```

8.2 Eng.trussSolver.GUI

8.2.1 frmMain.java

```
package eng.jTrussSolver.GUI;

import java.awt.Choice;
import java.awt.Color;
import java.awt.Label;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.io.File;
import java.util.HashMap;
import javax.swing.DefaultListModel;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.JTabbedPane;
import javax.swing.JTextField;
import javax.swing.border.BevelBorder;
import javax.swing.border.EmptyBorder;
import javax.swing.border.LineBorder;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
```

```
import net.miginfocom.swing.MigLayout;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import eng.jTrussSolver.Force.ForceData;
import eng.jTrussSolver.GUI.auxilliary.WindowsUtilities;
import eng.jTrussSolver.Material.MaterialData;
import eng.jTrussSolver.Node.NodeData;
import eng.jTrussSolver.Rod.RodData;
import eng.jTrussSolver.Section.SectionData;
import eng.jTrussSolver.Solver.StiffnessMatrixAssembler;
import eng.jTrussSolver.Solver.truss2dProblemDef;
import eng.jTrussSolver.GUI.auxilliary.SecurityConditions;
import javax.swing.ListSelectionModel;
import java.awt.Font;
import javax.swing.JScrollPane;


public class frmMain extends JFrame {

	private static final long serialVersionUID = 1L;
	private static final String JAXB_XML = "./Njaxb.xml";
	private static String TrussProblem_XML = "./DataFinal.xml";
	private JPanel contentPane;
	private truss2dProblemDef tr2d;
	private StiffnessMatrixAssembler SMA;
	private JList<String> lstOfNodes;
	private SecurityConditions SC;
	DefaultListModel<String> myModel;

	 /** Create the frame.*/

	private JFrame forClosingFrame;
	private JTextField tFxNode;
	private JTextField tFyNode;
	private JTextField tFzNode;
	private int currentSelected;
	private String nameField;
	private double xcoordField, ycoordField, zcoordField;
	private JPanel pnlNodes;
	private JPanel pnlNodes_NodesData;
	private JTabbedPane tabbedPane;
	private JPanel pnlRods;
	private JLabel lblXNode;
	private JLabel lblYNode;
	private JLabel lblZNode;
	private JButton btnNewRod;
	private JButton btnDeleteRod;
	private JButton btnUpdateRod;
	private JList<String> lstOfRods;
	private Choice chcNodeStart;
	private Label lblNodeStart;
	private Choice chcNodeEnd;
	private Label lblNodeEnd;
	private JPanel pnlRods_RodsData;
	private Choice chcSection;
	private Label lblSection;
	private Choice chcMaterial;
	private Label lblMaterial;
	private JPanel pnlSection;
	private JButton btnNewSection;
	private JButton btnDeleteSection;
	private JButton btnUpdateSection;
	private JButton btnLoadSection;
	private JButton btnSaveSection;
	private JPanel pnlSectionData;
	private JList<String> lstOfSection;
```

```
	private JTextField tFNameOfSection;
	private JLabel lblNameOfSection;
	private JTextField tFAreaOfSection;
	private JLabel lblAreaOfSection;
	private JLabel lblIxx;
	private JTextField tFIxxOfSection;
	private JPanel pnlMaterial;
	private JButton btnNewMaterial;
	private JButton btnDeleteMaterial;
	private JButton btnUpdateMaterial;
	private JList<String> lstOfMaterials;
	private JPanel pnlMaterialData;
	private JLabel lblNameOfMaterial;
	private JTextField tFNameOfMaterial;
	private JLabel lblEOfMaterial;
	private JTextField tFEOfMaterial;
	private JLabel lblS_epOfMaterial;
	private JTextField tFs_epOfMaterial;
	private JTextField tFNameOfRod;
	private JLabel lblNameOfRods;
	public int numberofnodes;
	public Document doc;
	public Document RodsDoc;
	public Document NodesDoc;
	public Document MaterialsDoc;
	private JButton btnShowGraphicalRepresentation;
	private JButton btnCallStiffnessMatrix;
	private JLabel lblRestrictMovement;
	private JCheckBox chckbxMoveX;
	private JCheckBox chckbxMoveY;
	private JCheckBox chckbxMoveZ;
	private JLabel lblRestrictRotation;
	private JCheckBox chckbxRotationX;
	private JCheckBox chckbxRotationY;
	private JCheckBox chckbxRotationZ;
	private JPanel pnlForces;
	private JButton btnAddForce;
	private JButton btnUpdateForce;
	private JButton btnDeleteForce;
	private JPanel pnlForces_Forcedata;
	private JList<String> lstOfForces;
	private JLabel lblValue;
	private JTextField tFForceValue;
	private JLabel lblAngle;
	private JTextField tFForceAngle;
	private JList<String> lstOfNode_Forces;
	private JLabel lblNewLabel;
	private JLabel lblAddForce;
	private JLabel lblAllForces;

	private int ListOfForcesSelection;
	private JScrollPane jScPnNodes_Forces;
	private JScrollPane jScPnForces;
	private JScrollPane jScPnRods;
	private JScrollPane jScPnNodes;
	private JScrollPane jScPnSections;
	private JScrollPane jScPnMaterials;

	public frmMain() {

		this.tr2d = new truss2dProblemDef();
		this.SC = new SecurityConditions();

		currentSelected = -1;


		forClosingFrame = this;
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
            setBounds(100, 100, 659, 407);
            contentPane = new JPanel();
            contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
            setContentPane(contentPane);
            contentPane.setLayout(new      MigLayout("",      "[612.00px]",
"[272.00px][][]"));

            tabbedPane = new JTabbedPane(JTabbedPane.TOP);
            contentPane.add(tabbedPane, "cell 0 0,alignx left,aligny top");

            pnlMaterial = new JPanel();
            pnlMaterial.setBorder(new      BevelBorder(BevelBorder.LOWERED,
null, null, null, null));
            tabbedPane.addTab("Materials", null, pnlMaterial, null);
            pnlMaterial.setLayout(new MigLayout("", "[][75.00,grow][grow]",
"[grow]"));

            btnNewMaterial = new JButton("New");
            btnNewMaterial.addActionListener(new ActionListener() {
            pnlMaterial.add(btnNewMaterial, "flowy,cell 0 0");

            btnDeleteMaterial = new JButton("Delete");
            btnDeleteMaterial.setEnabled(false);
            btnDeleteMaterial.addActionListener(new ActionListener() {
            pnlMaterial.add(btnDeleteMaterial, "cell 0 0");

            /**
             * Update material arraylist Data of selected material.
             */
            btnUpdateMaterial = new JButton("Update");
            btnUpdateMaterial.setEnabled(false);
            /**
             * Update material Method
             */
            btnUpdateMaterial.addActionListener(new ActionListener() {
            pnlMaterial.add(btnUpdateMaterial, "cell 0 0");

            jScPnMaterials = new JScrollPane();
            pnlMaterial.add(jScPnMaterials, "cell 1 0,grow");

            lstOfMaterials = new JList<String>();
            jScPnMaterials.setViewportView(lstOfMaterials);

      lstOfMaterials.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
            lstOfMaterials.setBorder(new   BevelBorder(BevelBorder.LOWERED,
null, null, null, null));
            lstOfMaterials.addListSelectionListener(new
ListSelectionListener() {

            pnlMaterialData = new JPanel();
            pnlMaterialData.setBorder(new LineBorder(new Color(0, 0, 0), 1,
true));
            pnlMaterial.add(pnlMaterialData, "cell 2 0,grow");
            pnlMaterialData.setLayout(new MigLayout("", "[]", "[][][]"));

            lblNameOfMaterial = new JLabel("Name:");
            pnlMaterialData.add(lblNameOfMaterial,  "flowx,cell  0  0,alignx
right");

            tFNameOfMaterial = new JTextField();
            pnlMaterialData.add(tFNameOfMaterial, "cell 0 0");
            tFNameOfMaterial.setColumns(10);

            lblEOfMaterial = new JLabel("E =");
            pnlMaterialData.add(lblEOfMaterial,   "flowx,cell   0   1,alignx
right");

            tFEOfMaterial = new JTextField();
```

```
            pnlMaterialData.add(tFEOfMaterial, "cell 0 1");
            tFEOfMaterial.setColumns(10);

            lblS_epOfMaterial = new JLabel("s_ep =");
            pnlMaterialData.add(lblS_epOfMaterial,  "flowx,cell  0  2,alignx
right");

            tFs_epOfMaterial = new JTextField();
            pnlMaterialData.add(tFs_epOfMaterial, "cell 0 2");
            tFs_epOfMaterial.setColumns(10);

            pnlSection = new JPanel();
            pnlSection.setBorder(new BevelBorder(BevelBorder.LOWERED, null,
null, null, null));
            tabbedPane.addTab("Sections", null, pnlSection, null);
            pnlSection.setLayout(new MigLayout("", "[][178.00,grow][grow]",
"[80.00,grow]"));

            btnNewSection = new JButton("New");
            btnNewSection.addActionListener(new ActionListener() {
            pnlSection.add(btnNewSection, "flowy,cell 0 0");

            btnDeleteSection = new JButton("Delete");
            btnDeleteSection.setEnabled(false);
            pnlSection.add(btnDeleteSection, "cell 0 0");
            btnDeleteSection.addActionListener(new ActionListener() {

            btnUpdateSection = new JButton("Update");
            btnUpdateSection.setEnabled(false);
            btnUpdateSection.addActionListener(new ActionListener() {
            pnlSection.add(btnUpdateSection, "cell 0 0");

            jScPnSections = new JScrollPane();
            pnlSection.add(jScPnSections, "cell 1 0,grow");


            lstOfSection = new JList<String>();
            jScPnSections.setViewportView(lstOfSection);
            lstOfSection.setBorder(new     BevelBorder(BevelBorder.LOWERED,
null, null, null, null));

            lstOfSection.addListSelectionListener(new
ListSelectionListener() {

            pnlSectionData = new JPanel();
            pnlSectionData.setBorder(new LineBorder(new Color(0, 0, 0), 1,
true));
            pnlSection.add(pnlSectionData, "cell 2 0,grow");
            pnlSectionData.setLayout(new                      MigLayout("",
"[31px][86px][27px][86px]", "[32.00px][][][]"));

            lblNameOfSection = new JLabel("Name:");
            pnlSectionData.add(lblNameOfSection,   "flowy,cell   0   0,alignx
right,aligny center");

            tFNameOfSection = new JTextField();
            pnlSectionData.add(tFNameOfSection,   "flowy,cell   1   0,alignx
left,aligny top");
            tFNameOfSection.setColumns(10);

            lblAreaOfSection = new JLabel("Area[mm2]:");
            pnlSectionData.add(lblAreaOfSection,    "cell    0    1,alignx
right,aligny center");

            tFAreaOfSection = new JTextField();
            pnlSectionData.add(tFAreaOfSection,    "cell    1    1,alignx
left,aligny top");
            tFAreaOfSection.setColumns(10);
```

```
            lblIxx = new JLabel("Ixx =");
            pnlSectionData.add(lblIxx,   "cell   0   2,alignx   right,aligny
center");

            tFIxxOfSection = new JTextField();
            pnlSectionData.add(tFIxxOfSection, "cell 1 2");
            tFIxxOfSection.setColumns(10);

            pnlNodes = new JPanel();
            tabbedPane.addTab("Nodes", null, pnlNodes, null);
            pnlNodes.setBorder(new  BevelBorder(BevelBorder.LOWERED,  null,
null, null, null));
            pnlNodes.setLayout(new MigLayout("", "[88.00][152.00][192.00]",
"[grow]"));

            final JButton btnNewNode = new JButton("New");
            btnNewNode.setEnabled(true);
            pnlNodes.add(btnNewNode, "flowy,cell 0 0");

            final JButton btnDeleteNode = new JButton("Delete");
            btnDeleteNode.setEnabled(false);
            pnlNodes.add(btnDeleteNode, "cell 0 0");

            final JButton btnUpdateNode = new JButton("Update");
            btnUpdateNode.setEnabled(false);
            pnlNodes.add(btnUpdateNode, "cell 0 0");

            jScPnNodes = new JScrollPane();
            pnlNodes.add(jScPnNodes, "cell 1 0,grow");
            lstOfNodes = new JList<String>();
            jScPnNodes.setViewportView(lstOfNodes);
            lstOfNodes.setBackground(Color.WHITE);
            lstOfNodes.setBorder(new BevelBorder(BevelBorder.LOWERED, null,
null, null, null));

                        lstOfNodes.addListSelectionListener(new
ListSelectionListener() {


            pnlNodes_NodesData = new JPanel();
            pnlNodes_NodesData.setBorder(new LineBorder(new Color(0, 0, 0),
1, true));
            pnlNodes.add(pnlNodes_NodesData, "cell 2 0,growx");
            pnlNodes_NodesData.setLayout(new    MigLayout("",    "[][][]",
"[][][][][][][][][]"));

            lblXNode = new JLabel("X");
            pnlNodes_NodesData.add(lblXNode, "cell 0 1,alignx trailing");

            tFxNode = new JTextField();
            pnlNodes_NodesData.add(tFxNode, "cell 1 1 2 1");
            tFxNode.setToolTipText("X");
            tFxNode.setColumns(10);

            lblYNode = new JLabel("Y");
            pnlNodes_NodesData.add(lblYNode, "cell 0 2,alignx trailing");

            tFyNode = new JTextField();
            pnlNodes_NodesData.add(tFyNode, "cell 1 2 2 1");
            tFyNode.setToolTipText("Y");
            tFyNode.setColumns(10);

            lblZNode = new JLabel("Z");
            pnlNodes_NodesData.add(lblZNode, "cell 0 3,alignx trailing");

            tFzNode = new JTextField();
            tFzNode.setEnabled(false);
```

```
            tFzNode.setEditable(false);
            pnlNodes_NodesData.add(tFzNode, "cell 1 3 2 1");
            tFzNode.setToolTipText("Z");
            tFzNode.setColumns(10);

            lblRestrictMovement = new JLabel("Restrict movement");
            pnlNodes_NodesData.add(lblRestrictMovement, "cell 0 4 3 1");

            chckbxMoveX = new JCheckBox("X");
            pnlNodes_NodesData.add(chckbxMoveX, "cell 0 5");
            chckbxMoveX.setToolTipText("Limitation for X axis");

            chckbxMoveY = new JCheckBox("Y");
            pnlNodes_NodesData.add(chckbxMoveY, "cell 1 5");
            chckbxMoveY.setToolTipText("Limitation for Y axis");

            chckbxMoveZ = new JCheckBox("Z");
            pnlNodes_NodesData.add(chckbxMoveZ, "cell 2 5");
            chckbxMoveZ.setToolTipText("Limitation for Z axis");
            chckbxMoveZ.setEnabled(false);
            chckbxMoveZ.setSelected(true);

            lblRestrictRotation = new JLabel("Restrict rotation");
            pnlNodes_NodesData.add(lblRestrictRotation, "cell 0 6 3 1");

            chckbxRotationX = new JCheckBox("X");
            chckbxRotationX.setEnabled(false);
            chckbxRotationX.setToolTipText("Limitation for X axis");
            pnlNodes_NodesData.add(chckbxRotationX, "cell 0 7");

            chckbxRotationY = new JCheckBox("Y");
            chckbxRotationY.setEnabled(false);
            chckbxRotationY.setToolTipText("Limitation for Y axis");
            pnlNodes_NodesData.add(chckbxRotationY, "cell 1 7");

            chckbxRotationZ = new JCheckBox("Z");
            chckbxRotationZ.setEnabled(false);
            pnlNodes_NodesData.add(chckbxRotationZ, "cell 2 7");

            /**
             * List Selection Listener for TAB Materials
             */

            /**
             * Listener for update data (coords & name) of Node.
             */
            btnUpdateNode.addActionListener(new ActionListener() {
            btnDeleteNode.addActionListener(new ActionListener() {

            btnNewNode.addActionListener(new ActionListener() {

            pnlRods = new JPanel();
            pnlRods.setBorder(new   BevelBorder(BevelBorder.LOWERED,   null,
null, null, null));
            tabbedPane.addTab("Rods", null, pnlRods, null);
            pnlRods.setLayout(new     MigLayout("",     "[][158.00][grow]",
"[grow]"));

            btnDeleteRod = new JButton("Delete");
            btnDeleteRod.setEnabled(false);
            btnDeleteRod.addActionListener(new ActionListener() {

            btnNewRod = new JButton("New");
            btnNewRod.addActionListener(new ActionListener() {

            jScPnRods = new JScrollPane();
            pnlRods.add(jScPnRods, "cell 1 0,grow");
```

```
                        lstOfRods = new JList<String>();
                        jScPnRods.setViewportView(lstOfRods);
                        lstOfRods.setBorder(new
BevelBorder(BevelBorder.LOWERED, null, null, null, null));


      lstOfRods.addListSelectionListener(new ListSelectionListener() {


      lstOfRods.addKeyListener(new KeyListener() {

            pnlRods_RodsData = new JPanel();
            pnlRods_RodsData.setBorder(new  LineBorder(new  Color(0,  0,  0),
1, true));
            pnlRods.add(pnlRods_RodsData, "cell 2 0,grow");
            pnlRods_RodsData.setLayout(new                    MigLayout("",
"[68px][28px][63px,grow][28px][28px]", "[][22px][][][][]"));

            lblNameOfRods = new JLabel("Name:");
            pnlRods_RodsData.add(lblNameOfRods, "cell 0 0,alignx right");

            tFNameOfRod = new JTextField();
            pnlRods_RodsData.add(tFNameOfRod, "cell 1 0 3 1,growx");
            tFNameOfRod.setColumns(10);

            lblNodeStart = new Label("Node Start:");
            pnlRods_RodsData.add(lblNodeStart, "cell 0 1,alignx right");

            chcNodeStart = new Choice();
            pnlRods_RodsData.add(chcNodeStart,  "cell  1  1  3  1,growx,aligny
bottom");

            lblNodeEnd = new Label("Node End:");
            pnlRods_RodsData.add(lblNodeEnd, "cell 0 2,alignx right");

            chcNodeEnd = new Choice();
            pnlRods_RodsData.add(chcNodeEnd,  "cell  1  2  3  1,growx,aligny
bottom");

            lblSection = new Label("Section:");
            pnlRods_RodsData.add(lblSection, "cell 0 4,alignx right");

            chcSection = new Choice();
            pnlRods_RodsData.add(chcSection,  "cell  1  4  3  1,growx,aligny
bottom");

            lblMaterial = new Label("Material:");
            pnlRods_RodsData.add(lblMaterial, "cell 0 5,alignx right");

            chcMaterial = new Choice();
            pnlRods_RodsData.add(chcMaterial, "cell 1 5 3 1,growx");
            pnlRods.add(btnNewRod, "flowy,cell 0 0");
            pnlRods.add(btnDeleteRod, "cell 0 0");

            btnUpdateRod = new JButton("Update");
            btnUpdateRod.setEnabled(false);
            btnUpdateRod.addActionListener(new ActionListener() {

            pnlRods.add(btnUpdateRod, "cell 0 0");

            pnlForces = new JPanel();
            pnlForces.setBorder(new  BevelBorder(BevelBorder.LOWERED,  null,
null, null, null));
            tabbedPane.addTab("Forces", null, pnlForces, null);
            pnlForces.setLayout(new                           MigLayout("",
"[][350.00][186.00,grow]", "[][228.00,grow]"));

            btnAddForce = new JButton("Add");
```

```
            btnAddForce.setEnabled(false);
            btnAddForce.addActionListener(new ActionListener() {

            lblAllForces = new JLabel("List of Forces:");
            lblAllForces.setFont(new Font("Tahoma", Font.ITALIC, 11));
            pnlForces.add(lblAllForces, "cell 1 0");
            pnlForces.add(btnAddForce, "flowy,cell 0 1");

            btnDeleteForce = new JButton("Delete");
            btnDeleteForce.setEnabled(false);

            btnDeleteForce.addActionListener(new ActionListener() {

            pnlForces.add(btnDeleteForce, "cell 0 1");

            btnUpdateForce = new JButton("Update");
            btnUpdateForce.setEnabled(false);

            btnUpdateForce.addActionListener(new ActionListener() {

            pnlForces.add(btnUpdateForce, "cell 0 1");

            jScPnForces = new JScrollPane();
            pnlForces.add(jScPnForces, "cell 1 1,grow");

                        lstOfForces = new JList<String>();
                        jScPnForces.setViewportView(lstOfForces);
                        lstOfForces.addListSelectionListener(new
ListSelectionListener() {

            lstOfForces.setBorder(new      BevelBorder(BevelBorder.LOWERED,
null, null, null, null));

            pnlForces_Forcedata = new JPanel();
            pnlForces_Forcedata.setBorder(new  LineBorder(new  Color(0,  0,
0), 1, true));
            pnlForces.add(pnlForces_Forcedata, "cell 2 1,grow");
            pnlForces_Forcedata.setLayout(new                MigLayout("",
"[215.00,grow]", "[][][][][][]"));

            lblAddForce = new JLabel("Force form:");
            lblAddForce.setBackground(Color.WHITE);
            lblAddForce.setFont(new Font("Tahoma", Font.BOLD, 12));
            pnlForces_Forcedata.add(lblAddForce, "cell 0 0,alignx left");

            lblValue = new JLabel("Value [Nt]:");
            pnlForces_Forcedata.add(lblValue, "flowx,cell 0 1");

            tFForceValue = new JTextField();
            pnlForces_Forcedata.add(tFForceValue, "cell 0 1");
            tFForceValue.setColumns(10);

            lblAngle = new JLabel("Angle [degrees]:");
            pnlForces_Forcedata.add(lblAngle, "flowx,cell 0 2");

            tFForceAngle = new JTextField();
            pnlForces_Forcedata.add(tFForceAngle, "cell 0 2");
            tFForceAngle.setColumns(10);

            lblNewLabel = new JLabel("Select force action Node: ");
            lblNewLabel.setFont(new Font("Tahoma", Font.ITALIC, 11));
            pnlForces_Forcedata.add(lblNewLabel, "cell 0 3");

            jScPnNodes_Forces = new JScrollPane();
            pnlForces_Forcedata.add(jScPnNodes_Forces, "cell 0 4,grow");

            lstOfNode_Forces = new JList<String>();
            jScPnNodes_Forces.setViewportView(lstOfNode_Forces);
```

```
            lstOfNode_Forces.setVisibleRowCount(5);
            lstOfNode_Forces.setBorder(new  BevelBorder(BevelBorder.LOWERED,
null, null, null, null));
            lstOfNode_Forces.addListSelectionListener(new
ListSelectionListener() {

            JButton btnExit = new JButton("Exit");
            contentPane.add(btnExit, "flowx,cell 0 1");

            btnSaveSection = new JButton("Save");
            contentPane.add(btnSaveSection, "cell 0 1");

            btnLoadSection = new JButton("Load");
            btnLoadSection.addActionListener(new ActionListener() {
            contentPane.add(btnLoadSection, "cell 0 1");



            btnCallStiffnessMatrix  =  new  JButton("Call  Stiffness  Matrix
Assembler");
            btnCallStiffnessMatrix.addActionListener(new ActionListener() {
            contentPane.add(btnCallStiffnessMatrix, "cell 0 2");

            btnShowGraphicalRepresentation  =  new  JButton("Show  Graphical
Representation");
            btnShowGraphicalRepresentation.addActionListener(new
ActionListener() {
            contentPane.add(btnShowGraphicalRepresentation, "cell 0 1");

            btnSaveSection.addActionListener(new ActionListener() {
            btnExit.addActionListener(new ActionListener() {

      }


      private void RefreshComboBoxNodes() {

      private void RefreshComboBoxNodes_ForcesTAB() {


      private void RefreshComboBoxSections() {

      private void RefreshComboBoxMaterials() {

      private void RefreshComboBoxRods() {

      private void RefreshComboBoxForces() {


      private void RefreshChoiceSections() {

      private void RefreshChoiceMaterials() {

      private void RefreshChoiceNodes() {

      private void ResetAllSectionFields() {

      private void ResetAllMaterialFields() {

      private void ResetAllRodFields() {

      private void ResetAllNodeFields() {

      private void ResetAllForcesFields() {

      private void ResetAllChoicesOfRods() {

       * Function for loading and parsing the XML file.
```

```
	private void LoadXMLdatafile(String xmlfilename)
	 * Function for loading Data into ArrayLists
	private void listNodes(Document DataFinal) {



}
```

8.2.2 Truss2dPanel.java

```
package eng.jTrussSolver.GUI;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Point;
import java.awt.Polygon;
import java.awt.geom.Ellipse2D;
import java.util.ArrayList;
import java.util.HashMap;
import javax.swing.JPanel;
import eng.geom.GeometricTransformations;
import eng.jTrussSolver.Force.ForceData;
import eng.jTrussSolver.Rod.RodData;
import eng.jTrussSolver.Solver.truss2dProblemDef;


public class Truss2dPanel extends JPanel {
	/**
	 *
	 */
	private static final long serialVersionUID = 1L;

	private Ellipse2D.Double[] shNodes;
	private ArrayList<Polygon> shRods;
	private ArrayList<Polygon> shForces = new ArrayList<Polygon>();
    Graphics2D g2d;
	private truss2dProblemDef tr2d;
	private final int _RodWidth_pxl = 4;
	private final int _ForceWidth_pxl = 2;
	private final int _NodeSize_pxl = 8;
	private final int frameXpxl = 340;
	private final int frameYpxl = 340;


	/** constructor
	 *
	 */
	public Truss2dPanel() {

		int nmax=10;
		shNodes = new Ellipse2D.Double[nmax];


		for(int i =0;i<nmax;i++)
		{
			double x,y;
			x=y=i*10.0;
			shNodes[i]=new Ellipse2D.Double(x, y,3,3);
		}

	}
```

```
	/** constructor
	 *
	 */
	public Truss2dPanel(truss2dProblemDef trussProblem) {
		updateProblemRepresentation(trussProblem);
	}


	public void updateProblemRepresentation(truss2dProblemDef problem)
	{
		this.tr2d = problem;
		drawNodes();
		drawRods();
		drawForces();

	}
	/***
	 * Drawing of nodes
	 */
	private void drawNodes()
	{
		int nmax=this.tr2d.getNodes().getNodesCount();
		shNodes = new Ellipse2D.Double[nmax];
		double findXmax, findXmin;
		double findYmin, findYmax;
		findXmin = tr2d.getNodes().get(nmax-1).X;
		findXmax = tr2d.getNodes().get(nmax-1).X;
		findYmax = tr2d.getNodes().get(nmax-1).Y;
		findYmin = tr2d.getNodes().get(nmax-1).Y;
		for(int i =0;i<nmax;i++)
		{// use this for to identify the data.
			if (tr2d.getNodes().get(i).X < findXmin ) {
				findXmin = tr2d.getNodes().get(i).X;
			}
			if (tr2d.getNodes().get(i).X > findXmax ) {
				findXmax = tr2d.getNodes().get(i).X;
			}
			if (tr2d.getNodes().get(i).Y < findYmin ) {
				findYmin = tr2d.getNodes().get(i).Y;
			}
			if (tr2d.getNodes().get(i).Y > findYmax ) {
				findYmax = tr2d.getNodes().get(i).Y;
			}
		}

		for(int i =0;i<nmax;i++)
		{
			double x,y;
			x=(tr2d.getNodes().get(i).X-         findXmin)/(findXmax-
findXmin)* frameXpxl +100;
			y=        450        -        ((tr2d.getNodes().get(i).Y-
findYmin)/(findYmax-findYmin)* frameYpxl + 10);
			shNodes[i]=new   Ellipse2D.Double(x,   y,   _NodeSize_pxl,
_NodeSize_pxl);

		}


	}

	public void paintComponent(Graphics g) {
		clear(g);
		g2d = (Graphics2D) g;

		for (int i=0;i<shRods.size();i++)
		{
			g2d.setColor(Color.cyan);
```

```
                g2d.drawPolygon(shRods.get(i));

            }
            for (int i=0;i<shNodes.length;i++)
            {
                g2d.setColor(Color.BLACK);
                g2d.fill(shNodes[i]);
                g2d.draw(shNodes[i]);

            }
            for (int i=0; i<shForces.size(); i++) {
                g2d.setColor(Color.RED);
                g2d.drawPolygon(shForces.get(i));
            }
        }
        /**
         * Function responsible for drawing the Rod Polygons.
         */
        private void drawRods()
        {
            shRods= new ArrayList<Polygon>();
            HashMap<Integer, RodData> rods=this.tr2d.getRods();
            System.out.println("Rods: " + rods.size());
            RodData b_tmp = null;

            for(int i=0; i<rods.size();i++)
            {
                b_tmp = rods.get(i);
                System.out.println(b_tmp.getRodDataAsString());
                shRods.add( calcRodPolygon(b_tmp));
            }


        }

        /**
         * Draws forces.
         */
        private void drawForces() {

            shForces.clear();
            HashMap<Integer, ForceData> forces = this.tr2d.getForces();
            System.out.println("\n" + "forces: " + forces.size());
            ForceData f_tmp = null;

            for(int i=0; i<forces.size();i++)
            {
                f_tmp = forces.get(i);
                System.out.println(f_tmp.getForceDataAsString());
                shForces.add( createBasicForcePolygon(f_tmp));
            }

        }

        /**
         * Draw a basic Arrow pointing at 0 degrees, starting from 0,0
         * then rotate it and
         * finally translate it.
         * @param f_tmp
         * @return
         */
        private Polygon createBasicForcePolygon(ForceData f_tmp) {

            Polygon shForcePolygon = basicForcePolygon();

            //rotate each point.
            double angle = f_tmp.getAngle_Rad();
            Point tmpPoint = new Point();
```

```
            for(int i=0; i< shForcePolygon.npoints; i++)
            {

                  tmpPoint = GeometricTransformations.rotatePoint(
                                    new        Point(shForcePolygon.xpoints[i],
shForcePolygon.ypoints[i]),
                                    //TODO: center point fixed
                                    new Point(8,4),
                                    angle);
                  shForcePolygon.xpoints[i] = tmpPoint.x;
                  shForcePolygon.ypoints[i] = tmpPoint.y;

            }
            // translate each point.

      shForcePolygon.translate((int)((shNodes[f_tmp.getNodeIDofForce()].x)),
(int)((shNodes[f_tmp.getNodeIDofForce()].y)));
            return shForcePolygon;
      }

      /**
       *  Generates  an  horizontal  arrow,  which  is  used  for  force
representation
       * @return
       */
      public Polygon basicForcePolygon() {
            Polygon shForcePolygon = new Polygon();
            int x1=0, x2=0, x3=0, x4=0, x5=0, x6=0, x7=0,
                        y1=0, y2=0, y3=0, y4=0, y5=0, y6=0, y7=0;
            int xstart;
            int ystart;

            xstart = + this._NodeSize_pxl;
            ystart =+ this._NodeSize_pxl/2;

            x1 = xstart;
            y1 = ystart - this._ForceWidth_pxl;

            x2 = xstart;
            y2 = ystart  + this._ForceWidth_pxl;

            x3 = x2 + 50;
            y3 = y2;

            x4 = x3;
            y4 = y3 + (2*this._ForceWidth_pxl);

            x5 = xstart + 50 + 10;
            y5 = ystart;

            x6 = x4;
            y6 = y3 - 4*this._ForceWidth_pxl;

            x7 = x4;
            y7 = y1;

            shForcePolygon.addPoint(x1, y1);
            shForcePolygon.addPoint(x2, y2);
            shForcePolygon.addPoint(x3, y3);
            shForcePolygon.addPoint(x4, y4);
            shForcePolygon.addPoint(x5, y5);
            shForcePolygon.addPoint(x6, y6);
            shForcePolygon.addPoint(x7, y7);
            return shForcePolygon;
      }


      /**
```

```
        * Calculates Rod polygon
        * @param b_tmp
        * @param RodWidthInPxl : Rod width in Pixels
        */
       private Polygon calcRodPolygon(RodData b_tmp) {

             Polygon shRodPolygon=new Polygon();
             int x1, x2, x3, x4, y1, y2, y3, y4;

             //TODo change from shape Nodes. this is unsafe!!!!
             int xstart = (int)((shNodes[b_tmp.NodeStart].x));
             int ystart = (int)((shNodes[b_tmp.NodeStart].y));
             int xend = (int)((shNodes[b_tmp.NodeEnd].x));
             int yend = (int)((shNodes[b_tmp.NodeEnd].y));

             double    rodLength    =    Math.sqrt(Math.pow(ystart-yend,2)+
Math.pow(xstart-xend,2));
             double   angle   =   GeometricTransformations.angleFull(xstart,
ystart, xend, yend, rodLength);
             System.out.print("angle[Deg] = " + angle*180/Math.PI + "\n");
             x1= (int)(_NodeSize_pxl/2);
             y1=-(int)(_RodWidth_pxl/2);

             x2= (int)(_NodeSize_pxl/2);
             y2= (int)(_RodWidth_pxl/2);

             x3= (int)(rodLength- _NodeSize_pxl/2);
             y3= (int)(_RodWidth_pxl/2);

             x4= (int)(rodLength - _NodeSize_pxl/2);
             y4=-(int)(_RodWidth_pxl/2);

             shRodPolygon.addPoint(x1, y1);
             shRodPolygon.addPoint(x2, y2);
             shRodPolygon.addPoint(x3, y3);
             shRodPolygon.addPoint(x4, y4);

             // rotate
             Point tmpPoint = new Point();
             for(int i=0; i< shRodPolygon.npoints; i++)
             {

                   tmpPoint = GeometricTransformations.rotatePoint(
                                new          Point(shRodPolygon.xpoints[i],
shRodPolygon.ypoints[i]),
                                new Point(0,0),
                                angle);
                   shRodPolygon.xpoints[i] = tmpPoint.x;
                   shRodPolygon.ypoints[i] = tmpPoint.y;

             }
             // translate each point.
             shRodPolygon.translate(xstart  +  this._NodeSize_pxl/2,  ystart+
this._NodeSize_pxl/2);

             //translate
             return shRodPolygon;
       }

       protected void clear(Graphics g) {
             super.paintComponent(g);
       }

       protected Ellipse2D.Double getCircle(int index) {
             return (shNodes[index]);
       }
```

```
}
```

8.2.3 Truss2dPanelAfterAssembler.java

```
package eng.jTrussSolver.GUI;

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Point;
import java.awt.Polygon;
import java.awt.Shape;
import java.awt.Stroke;
import java.awt.font.LineMetrics;
import java.awt.geom.Ellipse2D;
import java.lang.reflect.Array;
import java.util.ArrayList;
import java.util.HashMap;
import javax.sound.sampled.Line;
import javax.swing.JPanel;
import org.w3c.dom.stylesheets.LinkStyle;
import org.w3c.dom.stylesheets.StyleSheet;

import eng.geom.GeometricTransformations;
import eng.jTrussSolver.AssemblerResults.RodDisplacementData;
import eng.jTrussSolver.Force.ForceData;
import eng.jTrussSolver.Rod.RodData;
import eng.jTrussSolver.Solver.StiffnessMatrixAssembler;
import eng.jTrussSolver.Solver.truss2dProblemDef;


public class Truss2dPanelAfterAssembler extends JPanel {
	/**
	 *
	 */
	private static final long serialVersionUID = 1L;
	private Ellipse2D.Double[] shNodes;
	private Ellipse2D.Double[] shDisplacementNodes;
	private ArrayList<Polygon> shRods;
	private ArrayList<Polygon> shDisplacementRods;
	private ArrayList<Polygon> shForces = new ArrayList<Polygon>();
    Graphics2D g2d;
	private truss2dProblemDef tr2d;
	private final int _RodWidth_pxl = 4;
	private final int _ForceWidth_pxl = 2;
	private final int _NodeSize_pxl = 8;
	private final int frameXpxl = 340;
	private final int frameYpxl = 340;


	/** constructor
	 *
	 */
	public Truss2dPanelAfterAssembler() {

		int nmax=10;
		shNodes = new Ellipse2D.Double[nmax];


		for(int i =0;i<nmax;i++)
		{
			double x,y;
			x=y=i*10.0;
```

```
                  shNodes[i]=new Ellipse2D.Double(x, y,3,3);
            }

      }

      /** constructor
       *
       */
      public   Truss2dPanelAfterAssembler(truss2dProblemDef   trussProblem,
StiffnessMatrixAssembler SMA) {
            updateProblemRepresentation(trussProblem,SMA);
      }


      public  void  updateProblemRepresentation(truss2dProblemDef  problem,
StiffnessMatrixAssembler SMA)
      {
            this.tr2d = problem;
            drawNodes();
            drawRods();
            drawForces();
            drawDisplacementNodes();
            drawDisplacementRods();

      }


      /***
       * Drawing of nodes
       */
      private void drawNodes()
      {
            int nmax=this.tr2d.getNodes().getNodesCount();
            shNodes = new Ellipse2D.Double[nmax];
            double findXmax, findXmin;
            double findYmin, findYmax;
            findXmin = tr2d.getNodes().get(nmax-1).X;
            findXmax = tr2d.getNodes().get(nmax-1).X;
            findYmax = tr2d.getNodes().get(nmax-1).Y;
            findYmin = tr2d.getNodes().get(nmax-1).Y;
            for(int i =0;i<nmax;i++)
            {// use this for to identify the data.
                  if (tr2d.getNodes().get(i).X < findXmin ) {
                        findXmin = tr2d.getNodes().get(i).X;
                  }
                  if (tr2d.getNodes().get(i).X > findXmax ) {
                        findXmax = tr2d.getNodes().get(i).X;
                  }
                  if (tr2d.getNodes().get(i).Y < findYmin ) {
                        findYmin = tr2d.getNodes().get(i).Y;
                  }
                  if (tr2d.getNodes().get(i).Y > findYmax ) {
                        findYmax = tr2d.getNodes().get(i).Y;
                  }
            }

            for(int i =0;i<nmax;i++)
            {
                  double x,y;
                  x=(tr2d.getNodes().get(i).X-         findXmin)/(findXmax-
findXmin)* frameXpxl +100;
                  y=        450        -        ((tr2d.getNodes().get(i).Y-
findYmin)/(findYmax-findYmin)* frameYpxl + 10);
                  shNodes[i]=new   Ellipse2D.Double(x,   y,   _NodeSize_pxl,
_NodeSize_pxl);

            }
```

```
	}

	public void paintComponent(Graphics g) {
		clear(g);
		g2d = (Graphics2D) g;

		for (int i=0;i<shRods.size();i++)
		{
			g2d.setColor(Color.BLACK);
			g2d.drawPolygon(shRods.get(i));

		}
		for (int i=0;i<shNodes.length;i++)
		{
			g2d.setColor(Color.GRAY);
			g2d.fill(shNodes[i]);
			g2d.draw(shNodes[i]);

		}
		for (int i=0; i<shForces.size(); i++) {
			g2d.setColor(Color.BLUE);
			g2d.drawPolygon(shForces.get(i));
		}
		for (int i=0; i<shDisplacementNodes.length;i++)
		{
			g2d.setColor(Color.GREEN);
			g2d.fill(shDisplacementNodes[i]);
			g2d.draw(shDisplacementNodes[i]);

		}
		for (int i=0; i<shDisplacementRods.size(); i++)
		{

			//
http://docs.oracle.com/javase/tutorial/2d/geometry/strokeandfill.html
			float dash1[] = {10.0f};
			BasicStroke dashed =  new BasicStroke(1.0f,
			BasicStroke.CAP_BUTT,
			BasicStroke.JOIN_MITER,
			10.0f, dash1, 0.0f);
			g2d.setColor(Color.RED);
			g2d.setStroke(dashed);
			g2d.drawPolygon(shDisplacementRods.get(i));

		}
	}

	/**
	 * Function responsible for drawing the Rod Polygons.
	 */
	private void drawRods()
	{
		shRods= new ArrayList<Polygon>();
		HashMap<Integer, RodData> rods=this.tr2d.getRods();
		System.out.println("Rods: " + rods.size());
		RodData b_tmp = null;

		for(int i=0; i<rods.size();i++)
		{
			b_tmp = rods.get(i);
			System.out.println(b_tmp.getRodDataAsString());
			shRods.add( calcRodPolygon(b_tmp));
		}
	}

	/**
	 * Draws forces.
```

```
	 */
	private void drawForces() {

		shForces.clear();
		HashMap<Integer, ForceData> forces = this.tr2d.getForces();
		System.out.println("\n" + "forces: " + forces.size());
		ForceData f_tmp = null;

		for(int i=0; i<forces.size();i++)
		{
			f_tmp = forces.get(i);
			System.out.println(f_tmp.getForceDataAsString());
			shForces.add( createBasicForcePolygon(f_tmp));
		}

	}

	/**
	 * Draw a basic Arrow pointing at 0 degrees, starting from 0,0
	 * then rotate it and
	 * finally translate it.
	 * @param f_tmp
	 * @return
	 */
	private Polygon createBasicForcePolygon(ForceData f_tmp) {

		Polygon shForcePolygon = basicForcePolygon();

		//rotate each point.
		double angle = f_tmp.getAngle_Rad();
		Point tmpPoint = new Point();
		for(int i=0; i< shForcePolygon.npoints; i++)
		{

			tmpPoint = GeometricTransformations.rotatePoint(
					new	Point(shForcePolygon.xpoints[i],
shForcePolygon.ypoints[i]),
					//TODO: center point fixed
					new Point(8,4),
					angle);
			shForcePolygon.xpoints[i] = tmpPoint.x;
			shForcePolygon.ypoints[i] = tmpPoint.y;

		}
		// translate each point.

	shForcePolygon.translate((int)((shNodes[f_tmp.getNodeIDofForce()].x)),
(int)((shNodes[f_tmp.getNodeIDofForce()].y)));
		return shForcePolygon;
	}

	/**
	 *  Generates  an  horizontal  arrow,  which  is  used  for  force
representation
	 * @return
	 */
	public Polygon basicForcePolygon() {
		Polygon shForcePolygon = new Polygon();
		int x1=0, x2=0, x3=0, x4=0, x5=0, x6=0, x7=0,
				y1=0, y2=0, y3=0, y4=0, y5=0, y6=0, y7=0;
		int xstart;
		int ystart;

		xstart = + this._NodeSize_pxl;
		ystart =+ this._NodeSize_pxl/2;

		x1 = xstart;
		y1 = ystart - this._ForceWidth_pxl;
```

```
            x2 = xstart;
            y2 = ystart  + this._ForceWidth_pxl;

            x3 = x2 + 50;
            y3 = y2;

            x4 = x3;
            y4 = y3 + (2*this._ForceWidth_pxl);

            x5 = xstart + 50 + 10;
            y5 = ystart;

            x6 = x4;
            y6 = y3 - 4*this._ForceWidth_pxl;

            x7 = x4;
            y7 = y1;

            shForcePolygon.addPoint(x1, y1);
            shForcePolygon.addPoint(x2, y2);
            shForcePolygon.addPoint(x3, y3);
            shForcePolygon.addPoint(x4, y4);
            shForcePolygon.addPoint(x5, y5);
            shForcePolygon.addPoint(x6, y6);
            shForcePolygon.addPoint(x7, y7);
            return shForcePolygon;
      }


      /**
       * Calculates Rod polygon
       * @param b_tmp
       * @param RodWidthInPxl : Rod width in Pixels
       */
      private Polygon calcRodPolygon(RodData b_tmp) {

            Polygon shRodPolygon=new Polygon();
            int x1, x2, x3, x4, y1, y2, y3, y4;

            //TODo change from shape Nodes. this is unsafe!!!!
            int xstart = (int)((shNodes[b_tmp.NodeStart].x));
            int ystart = (int)((shNodes[b_tmp.NodeStart].y));
            int xend = (int)((shNodes[b_tmp.NodeEnd].x));
            int yend = (int)((shNodes[b_tmp.NodeEnd].y));

            double    rodLength    =    Math.sqrt(Math.pow(ystart-yend,2)+
Math.pow(xstart-xend,2));
            double   angle   =   GeometricTransformations.angleFull(xstart,
ystart, xend, yend, rodLength);
            System.out.print("angle[Deg] = " + angle*180/Math.PI + "\n");
            x1= (int)(_NodeSize_pxl/2);
            y1=-(int)(_RodWidth_pxl/2);

            x2= (int)(_NodeSize_pxl/2);
            y2= (int)(_RodWidth_pxl/2);

            x3= (int)(rodLength- _NodeSize_pxl/2);
            y3= (int)(_RodWidth_pxl/2);

            x4= (int)(rodLength - _NodeSize_pxl/2);
            y4=-(int)(_RodWidth_pxl/2);

            shRodPolygon.addPoint(x1, y1);
            shRodPolygon.addPoint(x2, y2);
            shRodPolygon.addPoint(x3, y3);
            shRodPolygon.addPoint(x4, y4);
```

```
            // rotate
            Point tmpPoint = new Point();
            for(int i=0; i< shRodPolygon.npoints; i++)
            {

                  tmpPoint = GeometricTransformations.rotatePoint(
                                    new          Point(shRodPolygon.xpoints[i],
shRodPolygon.ypoints[i]),
                                    new Point(0,0),
                                    angle);
                  shRodPolygon.xpoints[i] = tmpPoint.x;
                  shRodPolygon.ypoints[i] = tmpPoint.y;

            }
            // translate each point.
            shRodPolygon.translate(xstart  +  this._NodeSize_pxl/2,  ystart+
this._NodeSize_pxl/2);

            //translate
            return shRodPolygon;
      }

      private     Polygon     calcRodDisplacementPolygon(RodDisplacementData
DispRod_tmp) {

            Polygon shRodPolygon = new Polygon();
            int x1, x2, x3, x4, y1, y2, y3, y4;

            //TODo change from shape Nodes. this is unsafe!!!!
            int                           xstart                           =
(int)((shDisplacementNodes[DispRod_tmp.NodeStart].x));
            int                           ystart                           =
(int)((shDisplacementNodes[DispRod_tmp.NodeStart].y));
            int xend = (int)((shDisplacementNodes[DispRod_tmp.NodeEnd].x));
            int yend = (int)((shDisplacementNodes[DispRod_tmp.NodeEnd].y));

            double    rodLength    =    Math.sqrt(Math.pow(ystart-yend,2)+
Math.pow(xstart-xend,2));
            double   angle   =   GeometricTransformations.angleFull(xstart,
ystart, xend, yend, rodLength);
            System.out.print("angle[Deg] = " + angle*180/Math.PI + "\n");
            x1= (int)(_NodeSize_pxl/2);
            y1=-(int)(_RodWidth_pxl/2);

            x2= (int)(_NodeSize_pxl/2);
            y2= (int)(_RodWidth_pxl/2);

            x3= (int)(rodLength- _NodeSize_pxl/2);
            y3= (int)(_RodWidth_pxl/2);

            x4= (int)(rodLength - _NodeSize_pxl/2);
            y4=-(int)(_RodWidth_pxl/2);

            shRodPolygon.addPoint(x1, y1);
            shRodPolygon.addPoint(x2, y2);
            shRodPolygon.addPoint(x3, y3);
            shRodPolygon.addPoint(x4, y4);

            // rotate
            Point tmpPoint = new Point();
            for(int i=0; i< shRodPolygon.npoints; i++)
            {

                  tmpPoint = GeometricTransformations.rotatePoint(
                                    new          Point(shRodPolygon.xpoints[i],
shRodPolygon.ypoints[i]),
                                    new Point(0,0),
                                    angle);
```

```
                    shRodPolygon.xpoints[i] = tmpPoint.x;
                    shRodPolygon.ypoints[i] = tmpPoint.y;

             }
             // translate each point.
             shRodPolygon.translate(xstart  +  this._NodeSize_pxl/2,  ystart+
this._NodeSize_pxl/2);

             //translate
             return shRodPolygon;
      }

      private void drawDisplacementNodes() {


             int                              nmax                              =
this.tr2d.getNodeDisplacements().getNodeDisplacementCount();
             shDisplacementNodes = new Ellipse2D.Double[nmax];
             double findXmax, findXmin;
             double findYmin, findYmax;
             findXmin = tr2d.getNodeDisplacements().get(nmax-1).newX;
             findXmax = tr2d.getNodeDisplacements().get(nmax-1).newY;
             findYmax = tr2d.getNodeDisplacements().get(nmax-1).newY;
             findYmin = tr2d.getNodeDisplacements().get(nmax-1).newY;
             for(int i =0;i<nmax;i++)
             {// use this for to identify the data.
                    if (tr2d.getNodeDisplacements().get(i).newX < findXmin )
{
                           findXmin                                             =
tr2d.getNodeDisplacements().get(i).newX;
                    }
                    if (tr2d.getNodeDisplacements().get(i).newX > findXmax )
{
                           findXmax                                             =
tr2d.getNodeDisplacements().get(i).newX;
                    }
                    if (tr2d.getNodeDisplacements().get(i).newY < findYmin )
{
                           findYmin                                             =
tr2d.getNodeDisplacements().get(i).newY;
                    }
                    if (tr2d.getNodeDisplacements().get(i).newY > findYmax )
{
                           findYmax                                             =
tr2d.getNodeDisplacements().get(i).newY;
                    }
             }

             for(int i =0; i<nmax;i++)

             {
                    double x,y;
                    if              (tr2d.getNodes().get(i).X             !=
tr2d.getNodeDisplacements().get(i).newX) {
                           x   =   ((tr2d.getNodeDisplacements().get(i).newX-
findXmin)/(findXmax-findXmin)* frameXpxl) + 100 + (findXmax-findXmin)*0.2;
                    }
                    else {
                           x = shNodes[i].x;
                    }

                    if              (tr2d.getNodes().get(i).Y             !=
tr2d.getNodeDisplacements().get(i).newY) {
                           y              =              450              -
((tr2d.getNodeDisplacements().get(i).newY-     findYmin)/(findYmax-findYmin)*
frameYpxl) + 10 + (findYmax-findYmin)*0.2;
                    }
                    else {
```

```
                        y = shNodes[i].y;
                    }

                    shDisplacementNodes[i]  =  new  Ellipse2D.Double(x,  y,
_NodeSize_pxl, _NodeSize_pxl);

              }

       }

       private void drawDisplacementRods() {

              shDisplacementRods = new ArrayList<Polygon>();
              HashMap<Integer,   RodDisplacementData>   RodsDisplacement   =
this.tr2d.getRodDisplacements();
              System.out.println("Displacements       Rods:       "       +
RodsDisplacement.size());
              RodDisplacementData DispRod_tmp = null;

              for(int i=0; i<RodsDisplacement.size();i++)
              {
                     DispRod_tmp = RodsDisplacement.get(i);
                     System.out.println("\n"                                +
DispRod_tmp.getRodDisplacementAsString());

       shDisplacementRods.add(calcRodDisplacementPolygon(DispRod_tmp));
              }
       }

       protected void clear(Graphics g) {
              super.paintComponent(g);
       }

       protected Ellipse2D.Double getCircle(int index) {
              return (shNodes[index]);
       }


}
```

8.3 eng.jTrussSolver.GUI.auxilliary

8.3.1 WindowsUtilities.java

```
package eng.jTrussSolver.GUI.auxilliary;

import javax.swing.*;
import java.awt.*;

public class WindowsUtilities {

       /**
        * Tell system to use native look and feel, as in previous releases.
Metal
        * (Java) LAF is the default otherwise.
        */

       public static void setNativeLookAndFeel() {
              try {
```

```
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
            } catch (Exception e) {
                  System.out.println("Error setting native LAF: " + e);
            }
      }

      /**
       * A simplified way to see a JPanel or other Container. Pops up a
JFrame
       * with specified Container as the content pane.
       */

      public static JFrame openInJFrame(Container content, int width, int
height,
                  String title, Color bgColor) {
            JFrame frame = new JFrame(title);
            frame.setBackground(bgColor);
            content.setBackground(bgColor);
            frame.setSize(width, height);
            frame.setContentPane(content);
            // frame.addWindowListener(new ExitListener());
            frame.setVisible(true);
            return (frame);
      }

      /**
       * Uses Color.white as the background color.
       */
      public static JFrame openInJFrame(Container content, int width, int
height,
                  String title) {
            return    (openInJFrame(content,    width,    height,    title,
Color.white));
      }

      /**
       * Uses Color.white as the background color, and the name of the
Container's
       * class as the JFrame title.
       */
      public static JFrame openInJFrame(Container content, int width, int
height) {
            return (openInJFrame(content, width, height, content.getClass()
                        .getName(), Color.white));
      }
}
```

8.3.2 ExitListener.java

```
package eng.jTrussSolver.GUI.auxilliary;

import java.awt.event.*;
/** A listener that you attach to the top-level Frame or JFrame of
 *  your application, so quitting the frame exits the application.
 */

public class ExitListener extends WindowAdapter {
  public void windowClosing(WindowEvent event) {
    System.exit(1);
  }
}
```

8.3.3 SecurityConditions.java

```
package eng.jTrussSolver.GUI.auxilliary;

import java.awt.Choice;
import javax.swing.JList;
import javax.swing.JOptionPane;
import javax.swing.JTextField;
import eng.jTrussSolver.Force.ForceData;
import eng.jTrussSolver.Rod.RodData;
import eng.jTrussSolver.Solver.truss2dProblemDef;


public class SecurityConditions {

	public boolean checkMaterialFieldsIfNull(JTextField tFNameOfMaterial,
JTextField tFEOfMaterial, JTextField tFs_epOfMaterial) {
		boolean allow = true;
		String NameFields = tFNameOfMaterial.getText();
		String ModulusField = tFEOfMaterial.getText();
		String sepFields = tFs_epOfMaterial.getText();

		if (NameFields.equals("") || ModulusField.equals("") ||
sepFields.equals("")) {
			allow = false;
			JOptionPane.showMessageDialog(null,
					"Error: All fields must be completed.", "Error
Message",
					JOptionPane.ERROR_MESSAGE);
		}

		return allow;
	}

	public boolean checkMaterialNameIfExists(truss2dProblemDef tr2d,
String OldMaterialName, JTextField tFNameOfMaterial) {
		boolean allow = true;
		if (OldMaterialName.equals(tFNameOfMaterial.getText())) {
			allow = true;
		}
		else {
			for (String MaterialName : tr2d.getMaterials().keySet())
{
				if
(MaterialName.equals(tFNameOfMaterial.getText())){
					JOptionPane.showMessageDialog(null,
					   "Error: This name of Material already
exists.Please enter a different name for this Material!", "Error Message",
					   JOptionPane.ERROR_MESSAGE);
					allow = false;
					break;
				 }
			}
		}
		return allow;
	}

	public boolean checkNewMaterialNameIfExists(truss2dProblemDef tr2d,
JTextField tFNameOfMaterial) {
		boolean allow = true;
		for (String MaterialName : tr2d.getMaterials().keySet()) {
			if (MaterialName.equals(tFNameOfMaterial.getText())){
				JOptionPane.showMessageDialog(null,
					"Error: This name of Material already
exists.Please enter a different name for this Material!", "Error Message",
					JOptionPane.ERROR_MESSAGE);
```

```
                        allow = false;
                        break;
                    }
            }
            return allow;
      }

      public boolean checkMaterialFieldsIfAccepted(JTextField tFEOfMaterial,
JTextField tFs_epOfMaterial) {
            boolean allow = true;
            String ModulusField = tFEOfMaterial.getText();
            String sepFields = tFs_epOfMaterial.getText();

            try {
                  double ModulusValue = Double.parseDouble(ModulusField);
            }
            catch (NumberFormatException e) {
                  JOptionPane.showMessageDialog(null,
                              "Error: The value Modulus of elasticity must
be countable!", "Error Message",
                              JOptionPane.ERROR_MESSAGE);
                  allow = false;
            }
            // H if prostethike gia na emfanizetai ena parathuro tin fora
(stin periptosi poy einai kai oi 2 times den einai arithmoi).
            if (allow == true) {
                  try {
                        double sepValue = Double.parseDouble(sepFields);
                  }
                  catch (NumberFormatException e) {
                        JOptionPane.showMessageDialog(null,
                                    "Error: The value of Allowable stress
must be countable!", "Error Message",
                                    JOptionPane.ERROR_MESSAGE);
                        allow = false;
                  }
            }
            return allow;
      }

      public boolean checkSectionFieldsIfNull(JTextField tFNameOfSection,
JTextField tFAreaOfSection, JTextField tFIxxOfSection) {
            boolean allow = true;

            if          (tFNameOfSection.getText().equals("")          ||
tFAreaOfSection.getText().equals("") || tFIxxOfSection.getText().equals(""))
{
                  allow = false;
                  JOptionPane.showMessageDialog(null,
                              "Error: All fields must be completed.", "Error
Message",
                              JOptionPane.ERROR_MESSAGE);
            }

            return allow;
      }

      public boolean checkSectionNameIfExists(truss2dProblemDef tr2d, String
OldSectionName, JTextField tFNameOfSection) {
            boolean allow = true;
            if (tFNameOfSection.getText().equals(OldSectionName)) {
                  allow = true;
            }
            else {
                  for (String SectionName : tr2d.getSections().keySet()) {
                        if
(SectionName.equals(tFNameOfSection.getText())){
                              JOptionPane.showMessageDialog(null,
```

```
                                    "Error:  This  name  of  Section  already
exists.Please enter a different name for this Section!", "Error Message",
                                    JOptionPane.ERROR_MESSAGE);
                                allow = false;
                                break;
                            }
                    }
             }
             return allow;
      }

      public boolean checkNewSectionNameIfExists(truss2dProblemDef tr2d,
                    JTextField tFNameOfSection) {
             boolean allow = true;
             for (String SectionName : tr2d.getSections().keySet()) {
                    if (SectionName.equals(tFNameOfSection.getText())){
                           JOptionPane.showMessageDialog(null,
                              "Error:   This   name   of   Section   already
exists.Please enter a different name for this Section!", "Error Message",
                              JOptionPane.ERROR_MESSAGE);
                           allow = false;
                           break;
                    }
             }
             return allow;
      }

      public         boolean         checkSectionFieldsIfAccepted(JTextField
tFAreaOfSection, JTextField tFIxxOfSection) {
             boolean allow = true;

             try {
                    double                     AreaValue                    =
Double.parseDouble(tFAreaOfSection.getText());
             }
             catch (NumberFormatException e) {
                    JOptionPane.showMessageDialog(null,
                              "Error: The value of Area must be countable!",
"Error Message",
                              JOptionPane.ERROR_MESSAGE);
                    allow = false;
             }

             if (allow == true) {
                    try {
                           double                  IxxValue                 =
Double.parseDouble(tFIxxOfSection.getText());
                    }
                    catch (NumberFormatException e) {
                           JOptionPane.showMessageDialog(null,
                                    "Error:  The value of Moment of Inertia
must be countable!", "Error Message",
                                    JOptionPane.ERROR_MESSAGE);
                           allow = false;
                    }
             }
             return allow;
      }

      public  boolean  checkNodeFieldsIfNull(JTextField  tFxNode,  JTextField
tFyNode) {
             boolean allow = true;

             if               (tFxNode.getText().equals("")              ||
tFyNode.getText().equals("")) {
                    allow = false;
                    JOptionPane.showMessageDialog(null,
```

```
                              "Error: All fields must be completed.", "Error
Message",
                              JOptionPane.ERROR_MESSAGE);
              }

              return allow;
       }

       public   boolean   checkNodeIfAlreadyExists(truss2dProblemDef   tr2d,
JTextField tFxNode,
                     JTextField tFyNode) {
              boolean allow = true;
              double X = Double.parseDouble(tFxNode.getText());
              double Y = Double.parseDouble(tFyNode.getText());
              for (Integer NodeKey : tr2d.getNodes().keySet()) {
                     if    ((tr2d.getNodes().get(NodeKey).X    ==    X)    &&
(tr2d.getNodes().get(NodeKey).Y == Y)) {
                            allow = false;
                            JOptionPane.showMessageDialog(null,
                                   "Error:  This  Node  already  exists.",
"Error Message",
                                   JOptionPane.ERROR_MESSAGE);
                            break;
                     }
              }
              return allow;
       }

       public boolean checkNodeFieldsIfAccepted(JTextField tFxNode,
                     JTextField tFyNode) {
              boolean allow = true;
              try {
                     double X = Double.parseDouble(tFxNode.getText());
              }
              catch (NumberFormatException e) {
                     JOptionPane.showMessageDialog(null,
                            "Error:  The  value  of  X  must  be  countable!",
"Error Message",
                            JOptionPane.ERROR_MESSAGE);
                     allow = false;
              }

              if (allow == true) {
                     try {
                            double Y = Double.parseDouble(tFyNode.getText());
                     }
                     catch (NumberFormatException e) {
                            JOptionPane.showMessageDialog(null,
                                   "Error:   The   value   of   Y   must   be
countable!", "Error Message",
                                   JOptionPane.ERROR_MESSAGE);
                            allow = false;
                     }
              }
              return allow;
       }

       public boolean checkUpdatedNodeIfAlreadyExists(truss2dProblemDef tr2d,
                     JTextField  tFxNode,  JTextField  tFyNode,  JList<String>
lstOfNodes) {
              boolean allow = true;
              int                          UpdatedNode                          =
tr2d.getNodes().getStringElementID(lstOfNodes.getSelectedValue());
              double UpdatedNodeX = Double.parseDouble(tFxNode.getText());
              double UpdatedNodeY = Double.parseDouble(tFyNode.getText());
              for (Integer key : tr2d.getNodes().keySet()) {
                     if (tr2d.getNodes().get(key).NodeID != UpdatedNode) {
                            if (tr2d.getNodes().get(key).X == UpdatedNodeX &&
```

```
                                        tr2d.getNodes().get(key).Y        ==
UpdatedNodeY) {
                                allow = false;
                                JOptionPane.showMessageDialog(null,
                                          "Error:    This    Node    already
exists.", "Error Message",
                                          JOptionPane.ERROR_MESSAGE);
                                break;
                          }

                    }
              }
              return allow;
      }

      public  boolean  checkRodFieldsAndChoiceIfNull(truss2dProblemDef  tr2d,
JTextField tFNameOfRod, Choice chcMaterial,
                    Choice    chcSection,    Choice    chcNodeStart,    Choice
chcNodeEnd) {
              boolean allow = true;
              if (tFNameOfRod.getText().equals("")) {
                    allow = false;
                    JOptionPane.showMessageDialog(null,
                              "Error:  Please  enter  a  name  for  this  Rod.",
"Error Message",
                              JOptionPane.ERROR_MESSAGE);
              }
              if (allow != false && tr2d.getNodes().getNodesCount() == 0) {
                    allow = false;
                    JOptionPane.showMessageDialog(null,
                              "Error: The list with Nodes is blank!", "Error
Message",
                              JOptionPane.ERROR_MESSAGE);
              }
              if (allow != false && tr2d.getSections().getSectionsCount() ==
0) {
                    allow = false;
                    JOptionPane.showMessageDialog(null,
                              "Error:  The  list  with  Sections  is  blank!",
"Error Message",
                              JOptionPane.ERROR_MESSAGE);
              }
              if  (allow  !=  false  &&  tr2d.getMaterials().getMaterialsCount()
== 0) {
                    allow = false;
                    JOptionPane.showMessageDialog(null,
                              "Error:  The  list  with  Materials  is  blank!",
"Error Message",
                              JOptionPane.ERROR_MESSAGE);
              }
              return allow;
      }

      public    boolean    checkNewRodNameIfExists(truss2dProblemDef    tr2d,
JTextField tFNameOfNewRod) {
              boolean allow = true;
              for (Integer RodID : tr2d.getRods().keySet()) {
                    String RodName = tr2d.getRods().get(RodID).Name;
                     if (RodName.equals(tFNameOfNewRod.getText())){
                           JOptionPane.showMessageDialog(null,
                              "Error:  This  Rod's  name  already  exists.It's
recommended to enter a different name for this Rod!", "Error Message",
                              JOptionPane.ERROR_MESSAGE);
                           allow = false;
                           break;
                     }
              }
              return allow;
```

```
	}

	public boolean checkRodSelectedNodesIfEquals(Choice chcNodeStart,
			Choice chcNodeEnd) {
		boolean allow = true;
		if
(chcNodeStart.getSelectedItem().equals(chcNodeEnd.getSelectedItem())) {
			allow = false;
			JOptionPane.showMessageDialog(null,
					"Error: The selected start and end Node must
not be the same!", "Error Message",
					JOptionPane.ERROR_MESSAGE);
		}
		return allow;
	}

	public boolean checkSelectedNodesIfAlreadyConnected(truss2dProblemDef
tr2d,
			Choice chcNodeStart, Choice chcNodeEnd) {
		boolean allow = true;
		int StartNode =
tr2d.getNodes().getStringElementID(chcNodeStart.getSelectedItem());
		int EndNode =
tr2d.getNodes().getStringElementID(chcNodeEnd.getSelectedItem());
		for (Integer key : tr2d.getRods().keySet()) {
			int tmpRodStart = tr2d.getRods().get(key).NodeStart;
			int tmpRodEnd = tr2d.getRods().get(key).NodeEnd;
			if ((tmpRodStart == StartNode && tmpRodEnd == EndNode)
|| (tmpRodStart == EndNode && tmpRodEnd == StartNode)) {
				allow = false;
				JOptionPane.showMessageDialog(null,
						"Error: The selected Nodes are already
connected with a Rod", "Error Message",
						JOptionPane.ERROR_MESSAGE);
				break;
			}
		}
		return allow;
	}

	public boolean checkRodNewNameIfAlreadyExists(truss2dProblemDef tr2d,
			String oldRodName, JTextField tFNameOfRod) {
		boolean allow = true;
		if (oldRodName.equals(tFNameOfRod.getText())) {
			allow = true;
		}
		else {
			for (Integer RodID : tr2d.getRods().keySet()) {
				String RodName = tr2d.getRods().get(RodID).Name;
				 if (RodName.equals(tFNameOfRod.getText())){
					JOptionPane.showMessageDialog(null,
						"Error: This Rod's name already
exists.It's recommended to enter a different name for this Rod!", "Error
Message",
						JOptionPane.ERROR_MESSAGE);
					allow = false;
					break;
				 }
			}

		}
		return allow;
	}

	public boolean checkRodNewNodesIfAlreadyConnected(truss2dProblemDef
tr2d,
			int rodID, Choice chcNodeStart, Choice chcNodeEnd) {
		boolean allow = true;
```

```
            int                         StartNodeID                         =
tr2d.getNodes().getStringElementID(chcNodeStart.getSelectedItem());
            int                          EndNodeID                          =
tr2d.getNodes().getStringElementID(chcNodeEnd.getSelectedItem());
            if  (StartNodeID  ==  tr2d.getRods().get(rodID).NodeStart  &&
EndNodeID == tr2d.getRods().get(rodID).NodeEnd
                        ||                StartNodeID                ==
tr2d.getRods().get(rodID).NodeEnd         &&         EndNodeID         ==
tr2d.getRods().get(rodID).NodeStart) {
                  allow = true;
            }
            else {
                  for (Integer RodID : tr2d.getRods().keySet()) {
                        int               tmpNodeStart               =
tr2d.getRods().get(RodID).NodeStart;
                        int                tmpNodeEnd                =
tr2d.getRods().get(RodID).NodeEnd;
                         if (tmpNodeStart == StartNodeID && tmpNodeEnd ==
EndNodeID ||
                                    tmpNodeStart   ==   EndNodeID   &&
tmpNodeEnd == StartNodeID){
                              JOptionPane.showMessageDialog(null,
                                 "Error: The selected Nodes are already
connected with a Rod", "Error Message",
                                    JOptionPane.ERROR_MESSAGE);
                              allow = false;
                              break;
                         }
                  }
            }

            return allow;
      }


      public boolean checkForceFieldsIfNull(JTextField tFForceValue,
                  JTextField tFForceAngle) {
            boolean allow = true;
            if            (tFForceValue.getText().equals("")            ||
tFForceAngle.getText().equals("")) {
                  allow = false;
                  JOptionPane.showMessageDialog(null,
                              "Error: All fields must be completed.", "Error
Message",
                              JOptionPane.ERROR_MESSAGE);
            }
            return allow;
      }

      public boolean checkForceFieldsIfNumbers(JTextField tFForceValue,
                  JTextField tFForceAngle) {
            boolean allow = true;
            try {
                  double                    ForceValue                    =
Double.parseDouble(tFForceValue.getText());
            }
            catch (NumberFormatException e) {
                  JOptionPane.showMessageDialog(null,
                              "Error: The Force value must be countable!",
"Error Message",
                              JOptionPane.ERROR_MESSAGE);
                  allow = false;
            }
            if (allow != false) {
                  try {
                        double                ForceAngle                =
Double.parseDouble(tFForceAngle.getText());
                  }
```

```
                    catch (NumberFormatException e) {
                          JOptionPane.showMessageDialog(null,
                                "Error:  The  angle  of  Force  must  be
countable!", "Error Message",
                                JOptionPane.ERROR_MESSAGE);
                          allow = false;
                    }
              }

              return allow;
       }

       public boolean CheckForceIfAlreadyExists(truss2dProblemDef tr2d,
                    JTextField   tFForceValue,   JTextField   tFForceAngle,
JList<String> lstOfNode_Forces) {
              boolean allow = true;
              double ForceValue = Double.parseDouble(tFForceValue.getText());
              double ForceAngle = Double.parseDouble(tFForceAngle.getText());
              int                        NodeIDofForce                       =
tr2d.getNodes().getStringElementID(lstOfNode_Forces.getSelectedValue());
              for (Integer key : tr2d.getForces().keySet()) {
                    if     (tr2d.getForces().get(key).getMagnitude()     ==
ForceValue &&

       tr2d.getForces().get(key).getAngle_Degrees() == ForceAngle
                                 &&            NodeIDofForce             ==
tr2d.getForces().get(key).getNodeIDofForce()) {
                          int answer = JOptionPane.showConfirmDialog(null,
                                "Error: This Force already exists.Do you
want to sum?", "Error Message",
                                JOptionPane.OK_CANCEL_OPTION);
                          if (answer == 2) {
                                allow = false;
                                break;
                          }
                          if (answer == 0) {
                                allow = false;
                                ForceData            EditForce            =
tr2d.getForces().get(key);
                                double            newForceValue            =
EditForce.getMagnitude() + ForceValue;
                                ForceData  newForce  =  new  ForceData(key,
NodeIDofForce, newForceValue, ForceAngle);
                                tr2d.getForces().put(key, newForce);
                                break;
                          }
                    }
              }
              return allow;
       }

       public    boolean    CheckUpdatedForceIfAlreadyExists(truss2dProblemDef
tr2d,
                    JTextField   tFForceValue,   JTextField   tFForceAngle,
JList<String> lstOfNode_Forces,
                    JList<String> lstOfForces) {
              boolean allow = true;
              double ForceValue = Double.parseDouble(tFForceValue.getText());
              double ForceAngle = Double.parseDouble(tFForceAngle.getText());
              int                        NodeIDofForce                       =
tr2d.getNodes().getStringElementID(lstOfNode_Forces.getSelectedValue());
              int                          ForceID                           =
tr2d.getForces().getStringElementID(lstOfForces.getSelectedValue());
              if     (tr2d.getForces().get(ForceID).getNodeIDofForce()     !=
NodeIDofForce) {
                    for (Integer key : tr2d.getForces().keySet()) {
                          if (key != ForceID) {
```

```
                                    if
(tr2d.getForces().get(key).getMagnitude() == ForceValue &&

      tr2d.getForces().get(key).getAngle_Degrees() == ForceAngle
                                                &&      NodeIDofForce      ==
tr2d.getForces().get(key).getNodeIDofForce()) {
                                          int             answer             =
JOptionPane.showConfirmDialog(null,
                                                   "Error: This Force already
exists.Do you want to sum?", "Error Message",

JOptionPane.OK_CANCEL_OPTION);
                                          if (answer == 2) {
                                                allow = false;
                                                break;
                                          }
                                          if (answer == 0) {
                                                allow = false;
                                                ForceData     EditForce     =
tr2d.getForces().get(key);
                                                double     newForceValue     =
EditForce.getMagnitude() + ForceValue;
                                                ForceData   newForce   =   new
ForceData(key, NodeIDofForce, newForceValue, ForceAngle);
                                                tr2d.getForces().put(key,
newForce);

      tr2d.getForces().removeForce(ForceID);
                                                break;
                                          }
                                    }
                              }
                        }
            }
            else {
                  allow = true;
            }
            return allow;
      }

}
```

8.4 eng.geom

```
package eng.geom;

import java.awt.Point;

public class GeometricTransformations {

      /**
       * @param xstart
       * @param ystart
       * @param xend
       * @param yend
       * @param rodLength
       * @return
       */
      public static double angleFull(int xstart, int ystart, int xend, int
yend,
                  double rodLength) {
            double angle ;
            if(xend != xstart)
            {
                  angle=-Math.atan2(yend-ystart, xend-xstart);
            } else
            {
                  angle = -Math.PI/2;
```

```
                    if(yend<ystart)
                    {
                          angle = -angle;
                    }
              }
              return angle;
       }

       /**
        * rotation of point around another point by a certain amount of
degrees
        * @param pt :  point to be rotated
        * @param center  : pivot point
        * @param angleDeg : degrees
        * @return pt
        */
       public static Point rotatePoint(Point pt, Point center, double
angleRad)
       {
              double cosAngle = Math.cos( angleRad);
              double sinAngle = Math.sin(-angleRad);
              int dx=(pt.x-center.x);
              int dy=(pt.y-center.y);
              pt.x = center.x + (int) (dx*cosAngle - dy*sinAngle);
              pt.y = center.y + (int) (dx*sinAngle + dy*cosAngle);
              return pt;
       }
}
```

8.5 eng.jTrussSolver.Material

8.5.1 MaterialData.java

```
package eng.jTrussSolver.Material;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlType;

@XmlType(name = "Material", propOrder = { "Name","E","s_ep"})
@XmlAccessorType(XmlAccessType.FIELD)
public class MaterialData {

       @XmlElement
       public String Name;
       @XmlElement
       public double E;
       @XmlElement
       public double s_ep;

       public MaterialData ()
       {}

       public MaterialData (String Name,double E,double s_ep) {

              this.Name = Name;
              this.E = E;
              this.s_ep = s_ep;

       }
```

```
}
```

8.5.2 MaterialsHashMap.java

```
package eng.jTrussSolver.Material;

import java.util.HashMap;
import java.util.Map;
import javax.swing.JOptionPane;
import org.w3c.dom.Document;
import org.w3c.dom.Element;

public class MaterialsHashMap extends HashMap<String, MaterialData>{

	/**
	 *
	 */
	private static final long serialVersionUID = 1L;

	public MaterialsHashMap()
	{
	}

	public void addMaterial(MaterialData materialData) {
		this.put(materialData.Name, materialData);
	}

	public int getMaterialsCount() {
		return this.size();
	}

	/**
	 * removes the material
	 * @param aa
	 * TODO: to change for HashMap compatibility
	 */
	public void removeMaterial(String MaterialName) {
		this.remove(MaterialName);

	}

	/**
	 * Sets material to index
	 * @param currentSelectedMaterial
	 * @param b
	 */
	public void setMaterial(String oldname, MaterialData b) {
		this.remove(oldname);
		this.put(b.Name,b);
	}


	/**
	 * Code For Materials Data
	 * @param doc
	 * @return
	 */
	public Element prepareXMLElement(Document doc) {

		Element M = doc.createElement("Materials");


		for (Map.Entry<String, MaterialData> entr: this.entrySet())
		{
```

```
                    Element materialElement = doc.createElement("Material");
                    M.appendChild(materialElement);

                    Element el = doc.createElement("Name");

      el.appendChild(doc.createTextNode(entr.getValue().Name));
                    materialElement.appendChild(el);

                    Element c = doc.createElement("E");

      c.appendChild(doc.createTextNode(String.valueOf(entr.getValue().E)));
                    materialElement.appendChild(c);

                    Element sep = doc.createElement("S_ep");

      sep.appendChild(doc.createTextNode(String.valueOf(entr.getValue().s_ep
)));
                    materialElement.appendChild(sep);
              }
              return M;
      }

      /**
       * Clears() Hashmap and also resets counter.
       */
      public void clear()
      {
              super.clear();
      }
}
```

8.6 eng.jTrussSolver.Node

8.6.1 NodeData.java

```
package eng.jTrussSolver.Node;

import java.util.HashMap;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlType;
import eng.jTrussSolver.Force.ForceData;

@XmlType(name = "Node", propOrder = { "NodeID","X","Y", "Z", "notMoveX",
"notMoveY" })
@XmlAccessorType(XmlAccessType.FIELD)
public class NodeData {
      public int NodeID;
      public double X,Y,Z ;
      public boolean notMoveX , notMoveY;
      private int _forcecounter;

      public NodeData() {
              this.NodeID = 0;
              this.X = 0;
              this.Y = 0;
              this.Z = 0;
              this.notMoveX = false;
              this.notMoveY = false;
              this._forcecounter = 0;
      }
      /**
       * Constructor gia NodeData
```

```
	 * @param ID: Node Data [int]
	 * @param X :
	 * @param Y :
	 * @param Z :
	 */
	@Deprecated
	public NodeData(int ID,double X,double Y,double Z) {
		this.NodeID = ID;
		this.X = X;
		this.Y = Y;
		this.Z = Z;
		this.notMoveX = false;
		this.notMoveY = false;
	}

	/**
	 * Constructor gia NodeData
	 * @param ID: Node Data [int]
	 * @param X :
	 * @param Y :
	 * @param Z :
	 */
	public NodeData(int ID,double X,double Y,double Z,
			boolean notMoveX,boolean notMoveY,
			HashMap<Integer,ForceData>	NodeForces) {
		this.NodeID = ID;
		this.X = X;
		this.Y = Y;
		this.Z = Z;
		this.notMoveX = notMoveX;
		this.notMoveY = notMoveY;
	}

	/**
	 * Returns the NodeID
	 * @return [int] Node Identification number
	 */
	public int getNodeID() {

		return this.NodeID;
	}

	public String getNodeElementForcesTabAsString() {

		String str="";
		str += NodeID;
		str += ": ("+ X + " , ";
		str +=  Y + ")";

		return str;

	}

	public double getX() {

		return this.X;
	}

	public double getY() {

		return this.Y;
	}

    public double getZ() {

		return this.Z;
	}
```

```
	public void set(int ID, double X, double Y, double Z,boolean
notMoveX,boolean notMoveY)

	{
		this.NodeID = ID;
		this.X = X;
		this.Y = Y;
		this.Z = Z;
		this.notMoveX = notMoveX;
		this.notMoveY = notMoveY;
	}


	public int getNodeForceCounter() {
		return this._forcecounter;

	}


}
```

8.6.2 NodeHashMap.java

```
package eng.jTrussSolver.Node;

import java.text.NumberFormat;
import java.text.ParseException;
import java.util.HashMap;
import org.w3c.dom.Document;
import org.w3c.dom.Element;

public class NodeHashMap extends HashMap<Integer, NodeData> {

	/**
	 *
	 */
	private static final long serialVersionUID = 1L;

	private int _nodeCounter;
	/**
	 * Default Constructor
	 */
	public NodeHashMap()
	{
		_nodeCounter=0;
	}



	/**
	 * Add Node
	 * @param nodeData
	 */
	public void addNode(NodeData nodeData) {
		this.put(_nodeCounter,nodeData);
		_nodeCounter = _nodeCounter + 1 ;
	}

	/**
	 *
	 * @param currentselectednode
	 * @param b
	 */
	public void setNode(int currentselectednode, NodeData b) {
		this.remove(b.getNodeID());
		this.put(b.getNodeID(), b);
```

```
	}

	/**
	 *     http://stackoverflow.com/questions/2338790/get-int-from-string-
also-containing-letters-in-java
	 * Teleytaio comment (answered Feb 26 '10 at 0:57).
	 * @param jListElement
	 * @return The first Integer(ID) of JList Element(String) Value.
	 */
	public int getStringElementID(String jListElement) {
		int ID = 0;
		try {
			ID                                                    =
((Number)NumberFormat.getInstance().parse(jListElement)).intValue();
		} catch (ParseException e) {
			e.printStackTrace();
		}
		return ID;
	}

	/**
	 * returns count of nodes.
	 * @return
	 */
	public int getNodesCount() {

		return this.size();
	}

	public int getNodeHashKey() {
		return this._nodeCounter;
	}


	public void removeNode(int aa) {
		this.remove(aa);
	}

	/**
	 * Code For Nodes Data
	 * @param doc
	 * @return
	 */
	public Element prepareXMLElement(Document doc) {
		int i;
		Element N = doc.createElement("Nodes");

		for (i=0; i<this.size(); i++) {

			Element nodeElement = doc.createElement("Node");
			N.appendChild(nodeElement);

			Element g = doc.createElement("NodeID");

	g.appendChild(doc.createTextNode(String.valueOf((this.get(i).NodeID)))
);
			nodeElement.appendChild(g);

			Element c = doc.createElement("Coords");

			Element cx = doc.createElement("X");

	cx.appendChild(doc.createTextNode(String.valueOf(this.get(i).X)));
			nodeElement.appendChild(c).appendChild(cx);

			Element cy = doc.createElement("Y");
```

```
	cy.appendChild(doc.createTextNode(String.valueOf(this.get(i).Y)));
			nodeElement.appendChild(c).appendChild(cy);

			Element cz = doc.createElement("Z");

	cz.appendChild(doc.createTextNode(String.valueOf(this.get(i).Z)));
			nodeElement.appendChild(c).appendChild(cz);

			Element d =doc.createElement("Restrict_Moves");

			Element dx = doc.createElement("notx");

	dx.appendChild(doc.createTextNode(String.valueOf(this.get(i).notMoveX)
));
			nodeElement.appendChild(d).appendChild(dx);

			Element dy = doc.createElement("noty");

	dy.appendChild(doc.createTextNode(String.valueOf(this.get(i).notMoveY)
));
			nodeElement.appendChild(d).appendChild(dy);

		}
		return N;
	}
	/**
	 * Clears() Hashmap and also resets counter.
	 */
	public void clear()
	{
		super.clear();
		this._nodeCounter=0;
	}
}
```

8.7 eng.jTrussSolver.Section

8.7.1 SectionData.java

```
package eng.jTrussSolver.Section;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlType;


@XmlType(name = "Section", propOrder = { "SectionID", "Name","Area","Ixx"})
@XmlAccessorType(XmlAccessType.FIELD)
public class SectionData {
	@XmlElement
	public String Name;
	@XmlElement
	public double Area;
	@XmlElement
	public double Ixx;

	public SectionData (String Name,double Area,double Ixx) {
		this.Name = Name;
		this.Area = Area;
		this.Ixx = Ixx;
	}

}
```

8.7.2 SectionHashMap.java

```
package eng.jTrussSolver.Section;

import java.util.HashMap;
import java.util.Map;
import javax.swing.JOptionPane;
import org.w3c.dom.Document;
import org.w3c.dom.Element;

public class SectionHashMap extends HashMap<String, SectionData> {

	/**
	 *
	 */
	private static final long serialVersionUID = 1L;

	public SectionHashMap ()
	{
	}

	public void addSection(SectionData sectionData) {
		this.put(sectionData.Name,sectionData);

	}

	/**
	 * Sets section to index
	 * @param currentselectedSection
	 * @param b
	 */
	public void setSection(String oldname, SectionData section) {
		this.remove(oldname);
		this.put(section.Name, section);
	}

	/**
	 * Removes Section
	 * @param currentselectedSection
	 */
	public void removeSection(String SectionName) {
		this.remove(SectionName);
	}


	/**
	 * Returns number of Sections
	 * @return
	 */
	public int getSectionsCount() {
		return this.size();
	}

	/**
	 * Code For Sections Data
	 * @param doc
	 * @return
	 */
	public Element prepareXMLElement(Document doc) {
		Element S = doc.createElement("Sections");

		for (Map.Entry<String, SectionData> entr: this.entrySet())
		{

			Element sectionElement = doc.createElement("Section");
			S.appendChild(sectionElement);
```

```
                    Element g = doc.createElement("Name");
                    g.appendChild(doc.createTextNode(entr.getValue().Name));

                    sectionElement.appendChild(g);

                    Element c = doc.createElement("Area");

      c.appendChild(doc.createTextNode(String.valueOf(entr.getValue().Area))
);
                    sectionElement.appendChild(c);

                    Element I = doc.createElement("Ixx");

      I.appendChild(doc.createTextNode(String.valueOf(entr.getValue().Ixx)))
;
                    sectionElement.appendChild(I);
              }
              return S;
      }

}
```

8.8 eng.jTrussSolver.Force

8.8.1 ForceData.java

```
package eng.jTrussSolver.Force;

public class ForceData {



      private double magnitude;
      private double angle_Rad;
      private int ForceID;
      private int NodeIDofForce;



      public ForceData() {

            this.angle_Rad = 0;
            this.magnitude = 0;
            this.ForceID = 0;
            this.NodeIDofForce = 0;

      }



      public ForceData(int forceID,int NodeIDofForce,
                    double forceValue, double angleDeg) {

            this.angle_Rad = angleDeg /180 *Math.PI;
            this.ForceID = forceID;
            this.NodeIDofForce = NodeIDofForce;
            this.magnitude = forceValue;

      }


      public String getForceDataAsString() {

            String str="";
```

```
		str += ForceID + ":" +"\t";
		str += " Force Value: " + magnitude + "\t";
		str += " Force Angle [deg]: " + angle_Rad/Math.PI * 180;

		return str;

	}

	public double getMagnitude() {
		return magnitude;
	}



	public void setMagnitude(double magnitude) {
		this.magnitude = magnitude;
	}



	public double getAngle_Rad() {
		return angle_Rad;
	}


	public void setAngle_Rad(double angle_Rad) {
		this.angle_Rad = angle_Rad;
	}

	public double getAngle_Degrees() {
		double a = this.angle_Rad*180/Math.PI;
		return a;
	}

	public int getNodeIDofForce() {
		return NodeIDofForce;
	}



	public void setNodeIDofForce(int nodeIDofForce) {
		NodeIDofForce = nodeIDofForce;
	}



	public int getForceID() {
		return ForceID;
	}


}
```

8.8.2 ForceHashMap.java

```
package eng.jTrussSolver.Force;

import java.text.NumberFormat;
import java.text.ParseException;
import java.util.HashMap;
import org.w3c.dom.Document;
import org.w3c.dom.Element;

public class ForceHashMap extends HashMap<Integer, ForceData> {

	/**
```

```
	 *
	 */
	private static final long serialVersionUID = 1L;
	private int _forcecounter;

	public ForceHashMap()
	{

		_forcecounter = 0;
	}

	public void addForce(ForceData forcedata) {

		this.put(this._forcecounter, forcedata);
		this._forcecounter = _forcecounter + 1 ;
	}


	public void setForce(ForceData forcedata) {

		this.remove(forcedata.getForceID());
		this.put(forcedata.getForceID(), forcedata);

	}

	public void removeForce(int key) {
		this.remove(key);
	}

	public int getNodeForceHashKey() {
		return _forcecounter;
	}

	public int getNodeForceCount() {

		return this.size();
	}

	public int getStringElementID(String jListElement) {
		int ID = 0;
		try {
			ID                                                              =
((Number)NumberFormat.getInstance().parse(jListElement)).intValue();
		} catch (ParseException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
		return ID;
	}

	public double calcNodeResultantX(int key) {
		double NodeResultantX = 0;
		for (int i=0; i<this.size(); i++ ) {
			if (this.get(i).getNodeIDofForce()== key ) {
				NodeResultantX      =      NodeResultantX      +
(this.get(i).getMagnitude()* Math.cos(this.get(i).getAngle_Rad()));
			}
		}
		return NodeResultantX;
	}

	public double calcNodeResultantY(int key) {
		double NodeResultantY = 0;
		for (int i=0; i<this.size(); i++ ) {
			if (this.get(i).getNodeIDofForce() == key ) {
				NodeResultantY      =      NodeResultantY      +
(this.get(i).getMagnitude() * Math.sin(this.get(i).getAngle_Rad()));
			}
```

```
		}
		return NodeResultantY;
	}

	/**
	 * Code for Forces
	 * @param doc
	 * @return
	 */
	public Element prepareXMLElement(Document doc) {
		int i;
		Element F = doc.createElement("Forces");


		for (i=0; i<this.size(); i++) {

			Element forceElement = doc.createElement("Force");
			F.appendChild(forceElement);

			Element g = doc.createElement("ForceID");

	g.appendChild(doc.createTextNode(String.valueOf((this.get(i).getForceI
D()))));
			forceElement.appendChild(g);

			Element el = doc.createElement("actionNode");

	el.appendChild(doc.createTextNode(String.valueOf(this.get(i).getNodeID
ofForce())));
			forceElement.appendChild(el);

			Element c = doc.createElement("ForceValue");

	c.appendChild(doc.createTextNode(String.valueOf(this.get(i).getMagnitu
de())));
			forceElement.appendChild(c);

			Element an = doc.createElement("ForceAngle");

	an.appendChild(doc.createTextNode(String.valueOf(this.get(i).getAngle_
Rad())));
			forceElement.appendChild(an);
		}
		return F;
	}

	/**
	 * Clears() Hashmap and also resets counter.
	 */
	public void clear()
	{
		super.clear();
		this._forcecounter=0;
	}

}
```

8.9 eng.jTrussSolver.Solver

8.9.1 truss2dProblemDef.java

```
package eng.jTrussSolver.Solver;

import java.io.FileWriter;
import java.io.IOException;
```

```
import java.io.Writer;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.Marshaller;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlElementWrapper;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.apache.crimson.tree.XmlDocument;
import org.w3c.dom.Document;
import org.w3c.dom.Element;

import eng.jTrussSolver.Force.ForceHashMap;
import eng.jTrussSolver.Material.MaterialsHashMap;
import eng.jTrussSolver.AssemblerResults.NodeDisplacementHashMap;
import eng.jTrussSolver.AssemblerResults.RodDisplacementHashMap;
import eng.jTrussSolver.Node.NodeHashMap;
import eng.jTrussSolver.Rod.RodHashMap;
import eng.jTrussSolver.Section.SectionHashMap;


//http://java.dzone.com/articles/xml-bindings-jaxb-and-jax-rs
@XmlRootElement
public class truss2dProblemDef {
	@XmlElement
	@XmlElementWrapper(name = "Nodes")
	private NodeHashMap nodes;
	@XmlElement
	@XmlElementWrapper(name = "RodList")
	private RodHashMap rods;
	@XmlElement
	@XmlElementWrapper(name = "SectionList")
	private SectionHashMap sections;
	@XmlElement
	@XmlElementWrapper(name = "MaterialList")
	private MaterialsHashMap materials;
	@XmlElement
	@XmlElementWrapper(name = "ForcesList")
	private ForceHashMap forces;
	private NodeDisplacementHashMap nodeDisplacements;
	private RodDisplacementHashMap rodDisplacements;



	/**
	 * Constructor.
	 * instantiates objects and,
	 * sets nodes, rods, materials and Section counters to Zero.
	 */
	public truss2dProblemDef()
	{
		this.rods = new RodHashMap();
		this.nodes = new NodeHashMap();
		this.sections = new SectionHashMap();
		this.materials = new MaterialsHashMap();
		this.forces = new ForceHashMap();
		this.nodeDisplacements = new NodeDisplacementHashMap();
		this.rodDisplacements = new RodDisplacementHashMap();

	}

	/**
	 * Constructor.
	 * assigns values to objects and,
	 * sets nodes, rods, materials and Section counters to Zero.
	 */
	public truss2dProblemDef(NodeHashMap nodes,
```

```
                    RodHashMap rods,
                    SectionHashMap sections,
                    MaterialsHashMap materials,
                    ForceHashMap  forces,
                    NodeDisplacementHashMap nodeDisplacements,
                    RodDisplacementHashMap rodDisplacements )
      {
            this.rods = rods;
            this.nodes = nodes;
            this.sections = sections;
            this.materials = materials;
            this.forces = forces;
            this.nodeDisplacements = nodeDisplacements;
            this.rodDisplacements = rodDisplacements;

            //TODO: The counters here are not correctly set.
            //  The  counters  should  be  equal  at  least  with  the  highest
number
            // of the highest NODEID in the problem definition.
            //_nodeCounter=0;
            //_RodCounter=0;
            //_forcecounter = 0;
      }

      public RodDisplacementHashMap getRodDisplacements() {
            return this.rodDisplacements;

      }

      public NodeDisplacementHashMap getNodeDisplacements() {
            return this.nodeDisplacements;
      }

      public MaterialsHashMap getMaterials()
      {
            return this.materials;
      }


      /**
       * returns the Section Collection
       * @return ArrayList<SectionData>
       */
      public SectionHashMap getSections() {
            return this.sections;
      }

      /**
       * returns the Node Collection
       * @return NodeHashMap
       */
      public NodeHashMap getNodes() {
            return nodes;
      }



      public ForceHashMap getForces() {
            return this.forces;
      }


      /**
       * returns the Rod Collection
       * @return
       */
      public RodHashMap getRods() {
            return rods;
```

```
	}


	/**
	 * Function for creating XML file - all data.
	 */
	public void CreateDataXML(String xmlFilename) {

		DocumentBuilderFactory dbf =
DocumentBuilderFactory.newInstance();
		DocumentBuilder db=null;
		XmlDocument xmlData;
		try {
			db = dbf.newDocumentBuilder();
		} catch (ParserConfigurationException e) {
			e.printStackTrace();
		}
		Document doc = db.newDocument();

		Element root = doc.createElement("FinalData_List");
		doc.appendChild(root);
		Element N = this.getNodes().prepareXMLElement(doc);
		Element S = this.getSections().prepareXMLElement(doc);
		Element M = this.getMaterials().prepareXMLElement(doc);
		Element F = this.getForces().prepareXMLElement(doc);
		Element rodXML = this.getRods().prepareXMLelement(doc);
		// Append to root element
		root.appendChild(N);
		root.appendChild(S);
		root.appendChild(M);
		root.appendChild(F);
		root.appendChild(rodXML);

		xmlData = (XmlDocument) doc;

		try {
			xmlData.write(new FileWriter(xmlFilename), "UTF-8");
		}
		catch (IOException e) {
			e.printStackTrace();
		}

	}


	/**
	 * Creates Xml data file with the use of JAXB
	 * @param filename
	 */

	public void createDataXMLJAXB(String filename)
	{
		truss2dProblemDef nikos = new truss2dProblemDef(nodes, rods,
sections, materials, forces, nodeDisplacements, rodDisplacements);
		// create JAXB context and instantiate marshaller
		JAXBContext context;
		Writer w = null;
		try {
			context =
JAXBContext.newInstance(truss2dProblemDef.class);
			Marshaller m = context.createMarshaller();
			m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,
Boolean.TRUE);
			m.marshal(nikos, System.out);

			w = new FileWriter(filename);
			m.marshal(nikos, w);
		} catch (Exception e1) {
```

```
                  e1.printStackTrace();
            } finally {
                  try {
                        w.close();
                  } catch (Exception e) {
                  }
            }
      }

      public void clearProblem() {

            this.rods.clear();
            this.nodes.clear();
            this.materials.clear();
            this.sections.clear();
            this.forces.clear();
            this.nodeDisplacements.clear();
            this.rodDisplacements.clear();
      }
}
```

8.9.2 StiffnessMatrixAssembler.java

```
package eng.jTrussSolver.Solver;

import java.util.HashMap;
import Jama.Matrix;
import eng.jTrussSolver.Material.MaterialData;
import eng.jTrussSolver.Material.MaterialsHashMap;
import eng.jTrussSolver.Node.NodeData;
import eng.jTrussSolver.AssemblerResults.NodeDisplacementData;
import eng.jTrussSolver.AssemblerResults.RodDisplacementData;
import eng.jTrussSolver.Node.NodeHashMap;
import eng.jTrussSolver.Rod.RodData;
import eng.jTrussSolver.Rod.RodHashMap;
import eng.jTrussSolver.Section.SectionData;
import eng.jTrussSolver.Section.SectionHashMap;

public class StiffnessMatrixAssembler {

      private double[][] StiffnessMatrix;
      private double[][] FinalStiffnessMatrix;
      private truss2dProblemDef tr2d;
      private double C , S , C2 , S2; // C:"Cos" , S:"Sin" , C2:"Cos^2" ,
S2:"Sin^2"
      private double L , A, E; // L: "Length"  &  A: "Area" & E: "Young's
Modulus"
      private int MasterKsize;
      private int finaltempsize = 0;
      private int[] nodesArray;
      private int[] totalDOFS;
      private double[] ForcesMatrix;
      private double[] FinalForcesMatrix;
      private Matrix mtrxDisplacementsMatrix;
      private Matrix mtrxStructureDisplacement;
      private Matrix mtrxStressOfElements;
      private Matrix mtrxReactions;

      /**
       *  Exo  prosarmosei  ta  dedomena  sto  Example  4.1  tou  arxeioy
fea_chapter4.pdf
       *
       * Monades: x y z - mm , L-mm , A-m^2 , E-Pa
       *
```

```
        */

       public                      StiffnessMatrixAssembler(truss2dProblemDef
StiffnessMatrixProblemData)
       {
              this.tr2d = StiffnessMatrixProblemData;
              NodeHashMap _nodes = this.tr2d.getNodes();
              RodHashMap _rods = this.tr2d.getRods();
              SectionHashMap _sections = this.tr2d.getSections();
              MaterialsHashMap _materials = this.tr2d.getMaterials();

              MasterKsize = _nodes.size();
              StiffnessMatrix = new double[MasterKsize*2][MasterKsize*2];
              //rodsSize = rods.size();
              nodesArray = new int[_nodes.getNodesCount()];
              totalDOFS = new int[2*_nodes.getNodesCount()];

              for (int i=0; i<_nodes.getNodesCount(); i++ ) {
                     nodesArray[i] = tr2d.getNodes().get(i).NodeID;
              }

              for (int i=0; i<_rods.size(); i++) {
                     printRodStiffnessMatrix(_rods.get(i),
_nodes.get(_rods.get(i).NodeStart),
                                   _nodes.get(_rods.get(i).NodeEnd),
_sections.get(_rods.get(i).RodSection),
                                   _materials.get(_rods.get(i).RodMaterial));
              }
              System.out.print("Master Stiffness Matrix of Plane Truss is:" +
"\n");
              for (int i=0; i<StiffnessMatrix.length; i++) {
                     for (int j=0; j<StiffnessMatrix.length; j++) {
                            System.out.print(Math.round(StiffnessMatrix[i][j])
+ "\t");
                     }
                     System.out.print("\n");
              }
              System.out.print("\n");
              /* Call the method calcFinalStiffnessMatrix() to calculate
               * the Final Stiffness Matrix of structure. That is modified
               * and take into account the supports at nodes.
               * This final matrix is used in the calculations.
               */
              totalDOFS = calcRestricts(totalDOFS);
              FinalStiffnessMatrix                                          =
calcFinalStiffnessMatrix(tr2d,StiffnessMatrix);
              for (int i=0; i<FinalStiffnessMatrix.length; i++) {
                     for (int j=0; j<FinalStiffnessMatrix.length; j++) {

       System.out.print(Math.round(FinalStiffnessMatrix[i][j]) + "\t");
                     }
                     System.out.print("\n");
              }
              ForcesMatrix = new double[StiffnessMatrix.length];
              FinalForcesMatrix = new double[FinalStiffnessMatrix.length];
              FinalForcesMatrix                                             =
calcFinalForcesMatrix(tr2d,ForcesMatrix,FinalForcesMatrix.length,nodesArray,
totalDOFS);
              System.out.print("\n");
              System.out.println("Final Force Matrix:");
              for (int i=0; i<FinalForcesMatrix.length; i++) {
                     System.out.print(Math.round(FinalForcesMatrix[i])      +
"\t");
              }
              System.out.print("\n");

              /**
               * @author Kostas
```

```
             * @Matrix mtrxDisplacementsMatrix: For nodes that can be
moved.
             */
            mtrxDisplacementsMatrix                                    =
calcDisplacementsMatrix(tr2d,FinalStiffnessMatrix,FinalForcesMatrix);
            System.out.println("\n" + "Displacement Matrix:");
            double[][] Displacements = mtrxDisplacementsMatrix.getArray();
            for (int i=0; i<Displacements.length; i++ ) {
                  for (int j=0; j<Displacements[i].length; j++ ) {
                        System.out.print(Displacements[i][j] + "\t");
                  }
            }
            System.out.print("\n");

            /**
             * @author Kostas
             * @Matrix mtrxStructureDisplacement: Nodal displacement vector
for the entire structure.
             */
            mtrxStructureDisplacement                                  =
calcStructureDisplacement(totalDOFS,Displacements);
            double[][]              StructureDisplacements             =
mtrxStructureDisplacement.getArray();
            System.out.println("\n" + "Structure Displacements:");
            for (int i=0; i<StructureDisplacements.length; i++ ) {
                  for (int j=0; j<StructureDisplacements[i].length; j++ )
{
                        System.out.print(StructureDisplacements[i][j]    +
"\t");
                  }
            }
            System.out.print("\n");


            calcStresses(tr2d,mtrxStructureDisplacement);

            System.out.print("\n" + "Stresses table (MPa): " + "\n" );
            double[][] stresses = mtrxStressOfElements.getArray();
            for (int i=0; i<stresses.length; i++ ) {
                  for (int j=0; j<stresses[i].length; j++ ) {
                        System.out.print(stresses[i][j] + "\t");
                  }
            }
            System.out.print("\n");

            mtrxReactions                                              =
calcReactions(tr2d,totalDOFS,StiffnessMatrix,StructureDisplacements);
            double[][] Reactions = mtrxReactions.getArray();
            System.out.print("\n" + "Reactions (Nt):" + "\n");
            for (int i=0; i<Reactions.length; i++ ) {
//                for (int j=0; j<Reactions.length; j++ ) {
                        System.out.print(Reactions[i][0] + "\t");
//                }

            }
            System.out.print("\n");

            FillNodeDisplacementHashMap();
            System.out.println("NodeDisplacementHashMap= ");
            for (int i=0; i<tr2d.getNodeDisplacements().size(); i++ ) {

      System.out.println(tr2d.getNodeDisplacements().get(i).getNodeDisplacem
entDataAsString());
            }
            System.out.println("\n");

            FillRodDisplacementHashMap();
            System.out.println("RodDisplacementHashMap= ");
```

```
            for (int i=0; i<tr2d.getRodDisplacements().size(); i++ ) {

      System.out.println(tr2d.getRodDisplacements().get(i).getRodDisplacemen
tAsString());
            }
            System.out.println("\n");

      }


      private    Matrix    calcReactions(truss2dProblemDef    tr2d,    int[]
dofsRestrOrFree,        double[][]        stiffnessMatrix,        double[][]
structureDisplacements) {

            double[] Reactions;
            int k = 0;
            for (int i=0; i<dofsRestrOrFree.length; i++ ) {
                  if (dofsRestrOrFree[i] == 0) {
                        k++ ;
                  }
            }

            int[] NotMoveIndex = new int[k];
            int index = -1;
            for (int i=0; i<dofsRestrOrFree.length; i++) {
                  if (dofsRestrOrFree[i] == 0) {
                        index++;
                        NotMoveIndex[index] = i;
                  }
            }

            double[][]          ReactionsStiffness          =          new
double[NotMoveIndex.length][stiffnessMatrix.length];
            for (int i=0; i<NotMoveIndex.length; i++ ) {
                  for (int j=0; j<stiffnessMatrix.length; j++ ) {
                        ReactionsStiffness[i][j]                          =
stiffnessMatrix[NotMoveIndex[i]][j];
                  }
            }

            Reactions = new double[NotMoveIndex.length];
            for (int i=0; i<Reactions.length; i++ ) {
                  double r = 0;
                  for (int j=0; j<stiffnessMatrix.length; j++ ) {
                        r                                                 +=
ReactionsStiffness[i][j]*structureDisplacements[0][j];
                  }
                  Reactions[i] = r;
            }

            Matrix temp = new Matrix(Reactions, Reactions.length);

            return temp;

      }

      /**
       * @author Kostas
       * @Matrix mtrxStressOfElements: Stresses of Elements
       * Equation: σ = (Ee/le)*[L]*{q}
       */
      public void calcStresses(truss2dProblemDef tr2d,
                  Matrix mtrxStructureDisplacement) {

            double[] Stresses = new double[tr2d.getRods().size()];

            RodData tmpRod=null;
            for (int i=0; i<Stresses.length; i++ ) {
```

```
                  tmpRod = tr2d.getRods().get(i);
                  double L = tmpRod.getRodL(tr2d.getNodes());
                  double l = tmpRod.getRod_l(tr2d.getNodes());
                  double m = tmpRod.getRod_m(tr2d.getNodes());
                  double E = tmpRod.getRodE(tr2d.getMaterials());
                  double[] TransformationL = {-l,-m,l,m};
                  double[]                        q                        =
getDisplacementMatrix_q(tr2d.getNodes(),tr2d.getRods().get(i),
                              mtrxStructureDisplacement);

                  double stress = 0;
                  for (int j=0; j<TransformationL.length; j++ ) {
                        /*
                         * stress is in N/mm^2 or else in MPa
                         */
                        stress = stress + (E/L) * (TransformationL[j]) *
(q[j] * 0.000001);
                  }

                  Stresses[i] = stress;
            }

            mtrxStressOfElements = new Matrix(Stresses, 1);

      }

      private  double[]  getDisplacementMatrix_q(HashMap<Integer,  NodeData>
Nodes,
                  RodData rodData, Matrix mtrxStructureDisplacement) {

            mtrxStructureDisplacement.transpose();
            double[][]                  Displacements                  =
mtrxStructureDisplacement.getArray();
            double[] temp = new double[4];
            int N1 = rodData.NodeStart;
            int N2 = rodData.NodeEnd;
            int N1fixID = Nodes.get(N1).NodeID+1;
            int N2fixID = Nodes.get(N2).NodeID+1;
            int nodeN1start = 2*(N1fixID)-2;
            int nodeN1end = 2*(N1fixID)-1;
            int nodeN2start = 2*(N2fixID)-2;
            int nodeN2end = 2*(N2fixID)-1;

            int[] indexes = {nodeN1start,nodeN1end,nodeN2start,nodeN2end};

            for (int i=0; i<temp.length; i++ ) {
                  temp[i] = Displacements[0][indexes[i]];
            }

            return temp;
      }


      private Matrix calcStructureDisplacement(int[] totalDOFS,
                  double[][] tempdisplacements) {

            this.totalDOFS = totalDOFS;
            double[]        tempStructureDisplacements        =        new
double[totalDOFS.length];
            int k = 0;
            for (int i=0; i<totalDOFS.length; i++ ) {
                  for (int j=0; j<totalDOFS[i]; j++ ) {
                        if (totalDOFS[i] == 1) {
                              tempStructureDisplacements[i]             =
tempdisplacements[k][j];
                              k ++ ;
                        }
                        else {
```

```
                                    tempStructureDisplacements[i] = 0 ;
                              }
                        }
                  }
                  Matrix temp = new Matrix(tempStructureDisplacements,1);

                  return temp;
            }

            private int[] calcRestricts(int[] totalDOFS) {
                  for (int i=0; i<(tr2d.getNodes().size()); i++) {

                        int N1fixedID = (tr2d.getNodes().get(i).NodeID) + 1;
                        int N1xstart = (2*N1fixedID)-2;
                        int N1xend = (2*N1fixedID)-1;

                        if (tr2d.getNodes().get(i).notMoveX == false) {
                              totalDOFS[N1xstart] = 1;
                              finaltempsize = finaltempsize + 1;
                        }
                        else {
                              totalDOFS[N1xstart] = 0;
                        }

                        if (tr2d.getNodes().get(i).notMoveY == false) {
                              totalDOFS[N1xend] = 1;
                              finaltempsize = finaltempsize + 1;
                        }
                        else {
                              totalDOFS[N1xend] = 0;
                        }
                  }
                  return totalDOFS;
            }

            /**
             * Prints the stiffnes Matrix of a Rod
             * Public function that allows debugging
             */
            public void printRodStiffnessMatrix(RodData B1, NodeData N1, NodeData
N2, SectionData S1,
                        MaterialData M1)
            {
                  double[][] rodMatrix = calcStiffnessMatrixRod(B1, N1, N2, S1,
M1);
                  /*Code for printing*/

                  System.out.print("\n"+"Rod name: " + B1.Name + "\t" +
                  ", L=" + L + " , "+ "A=" + A + " , "+"E=" + E + " \t " +
                  "Start_Node: "+ N1.NodeID + " , " + "End_Node: "+ N2.NodeID +
"\n");
                  System.out.print("Stiffness  Matrix  of  this  Element  is:"  +
"\n");
                  for (int i=0; i<rodMatrix.length; i++) {
                        for (int j=0; j<rodMatrix.length; j++) {
                              System.out.print(Math.round(rodMatrix[i][j])  +  "
");
                        }
                        System.out.print("\n");
                  }
                  System.out.print("\n");

                  double[][] MasterKtemp = calcMasterK(B1,N1,N2,S1,M1,rodMatrix);

                  for (int i=0; i<StiffnessMatrix.length; i++) {
                        for (int j=0; j<StiffnessMatrix.length; j++) {
                              StiffnessMatrix[i][j] += MasterKtemp[i][j];
                        }
```

```
            }

      }

      /**
       *
       */
      private double[][] calcStiffnessMatrixRod(RodData B1, NodeData N1,
NodeData N2,
                  SectionData S1, MaterialData M1)
      {

            /*Ftiaxno enan pinaka double 4x4 gia to kathe element*/
            double[][] rodStiffnessMatrix = new double[4][4];
            /* L= tetragoniki.riza( ((x2-x1)^2) + ((y2-y1)^2) )   */
            L   =   Math.sqrt(   (Math.pow((N2.X-N1.X),2))+(Math.pow((N2.Y-
N1.Y),2)) );

            C = (N2.X-N1.X)/L;
            S = (N2.Y-N1.Y)/L;
            C2 = Math.pow(C, 2);
            S2 = Math.pow(S, 2);

            /* Ftiaxno 4 arrays. Prostheto kathe mia stin antistixi grammi
toy
               pinaka toy mitrwou. */
            double firstArray[] =  {(C2) , (C*S) , (-C2) , (-C*S)};
            rodStiffnessMatrix[0] = firstArray;
            double secondArray[] = {(C*S) , (S2) , (-C*S) , (-S2)};
            rodStiffnessMatrix[1] = secondArray;
            double thirdArray[] =  {(-C2) , (-C*S) , (C2) , (C*S)};
            rodStiffnessMatrix[2] = thirdArray;
            double fourthArray[] = {(-C*S) , (-S2) , (C*S) , (S2)};
            rodStiffnessMatrix[3] = fourthArray;
            /* A: i epifaneia tis ravdou kai E: to metro elastikotitas tou
ylikou */
            A = S1.Area;
            E = M1.E;
            /* ftiaxno to teliko mitrwo tis ravdou.
             * Diladi ton parapano pinaka pollaplasiasmeno me to ginomeno
(E*A)/L .
             * */
            double[][] k = new double[4][4];
            for (int i=0; i<rodStiffnessMatrix.length; i++) {
                  for (int j=0; j<rodStiffnessMatrix.length; j++) {
                        k[i][j]  =  (  ((E*A)/L)*(rodStiffnessMatrix[i][j])
);
                  }

            }

            return k;
      }
      /**Function that allows to calculate temporary Stiffness matrix
       * before merge with master Stiffness matrix of overall truss.
       *
       * @param B1 : Rod
       * @param N1 : Start node of rod B1
       * @param N2 : End node of rod B1
       * @param S1 : Section of rod B1
       * @param M1 : Meterial of rod B1
       * @param rodMatrix : The rod Stiffeness Matrix
       * @return K : K is the temporary table 8X8. Returned for merge with
       *             Master Stiffness Matrix.
       */
      private double[][] calcMasterK(RodData B1, NodeData N1, NodeData N2,
                  SectionData S1, MaterialData M1, double[][] rodMatrix)
      {
```

```
            /*
             * To megethos tou telikou mitrwou tou dyktiomatos kathorizetai
apo
             *  ton  arithmo  twn  nodes.  To  kathe  node  exei  2  vathmous
eleutherias.
             *  Ara  to  megethos  tou  telikou  pinaka  tha  einai  enas
tetragwnikos
             * pinakas me pleura 2*arithmos_nodes .
             *  @param  MasterKsize  :  to  megethos  tis  arraylist  me  ta
nodes,diladi
             *                      o arithmos twn nodes.
             */
            double[][] K = new double[MasterKsize*2][MasterKsize*2];
            /*Write zeros in temponary table K*/
            for (int i=0; i<K.length; i++) {
                  for (int j=0; j<K.length; j++) {
                        K[i][j] += 0;
                  }
            }

            int node1xstart,node1xend,node1ystart,node1yend;
            int node2xstart,node2xend,node2ystart,node2yend;

            /*Kano mia metatropi poy einai anagkaia gia to gemisma tou
             * prosorinou mitrwou. Me ayto ton tropo apofeugo tis diklides
             * pou isos tha eprepe na valo,sxetika me tin proti thesi tou
             * pinaka. Sygkekrimena i parakato metavliti node1xstart otan
             *  eprepe  na  parei  metavliti  to  proto  stoixeio  tou  telikou
mitrwou
             * edine timi -1 , epeidi to 0 pou itan to ID tou protou komvou
             * vasei tis nodes(arraylist) pol/smeno epi 2 kai meiomeno kata
             *  ena edine -1 .Etsi i allagi ayti me kaluptei apo oles tis
apopseis.
             * */
            int N1fixID = N1.NodeID+1;
            int N2fixID = N2.NodeID+1;

            /*Prosarmozw tin thesi tou kathe stoixeiou ston pinaka 8x8
             *  Etsi,gia to kathe node ID kratao 2 theseis se kathe stili
kai
             * grammi ston pinaka K[8X8] .
             *  Aytes oi metavlites,ousiastika einai oi deiktes metatropis
apo
             * ton pinaka 4X4 tou mitrwou tou kathe rod ston pinaka 8X8 tou
             * mitrwou tou diktuomatos.*/
            node1xstart = 2*(N1fixID)-2;
            node1xend = 2*(N1fixID)-1;
            node1ystart = 2*(N1fixID)-2;
            node1yend = 2*(N1fixID)-1;
            node2xstart = 2*(N2fixID)-2;
            node2xend = 2*(N2fixID)-1;
            node2ystart = 2*(N2fixID)-2;
            node2yend = 2*(N2fixID)-1;

            /*ftiaxno  2  listes  array,i  mia  einai  gia  ta  stoixeia  kathe
grammis kai i
             *  alli  gia  ta  stoixeia  tis  kathe  stilis.Autes  oi  stiles
anaferontai se
             * kathe rod pou kalei tin methodo.*/
            int                    dofsRestrOrFree[]                    =
{node1xstart,node1xend,node2xstart,node2xend};
            int yarray[] = {node1ystart,node1yend,node2ystart,node2yend};

            /*Me  ayti  ti  dipli  'for'gemizo  kathe  mia  grammi  toy  pinaka
K[8X8] .
             *Gemizo  gia  kathe  grammi,mono  ta  stoixeia  tis  stilis  pou
xreiazetai
             *kai ta ksero,afou exo kratisei gia kathe Node to ID tou
```

```
              *kai to exo metatrepsei se theseis stoixeiwn tou pinaka 8X8 .
              *Etsi gia kathe Node ID , kratao 4 krisimes times synolika
ston pinaka
              *8X8 , diladi 2 grammes kai 2 stiles. Etsi,eisago 4 stoixeia
se kathe
              *grammi kai alla 4 se kathe stili.Synolika diladi (2*4) +
(2*4) , osa
              *telika kai ta stoixeia kathe mitrwou gia kathe ravdo.*/
             for (int xi=0; xi<4; xi++) {
                   for (int yi=0; yi<4; yi++) {
                         K[dofsRestrOrFree[xi]][yarray[yi]]             +=
rodMatrix[xi][yi];
                   }
             }

             return K;
      }
      /**
       *  Fuction that allows to calculate final Stiffeness Matrix of
structure.
       * Specifically, take into account the support at nodes,and modify the
       * temporary Stiffness matrix.
       * @param stiffnessMatrix2
       * @param tr2d2
       */
      private     double[][]     calcFinalStiffnessMatrix(truss2dProblemDef
temptr2d, double[][] tempstiffnessMatrix) {

             this.tr2d = temptr2d;

             /*
              *  Ftiaxno ena array me megethos ton arithmo ton pithanon
metatopiseon.
              *  To array ousiastika periexei 0 kai 1 , opou 0 simainei
eleutheria kinisis
              * kai 1 simainei periorismos kinisis.
              * Episis mesa apo auto to kommati metrao kai posoi periorismoi
uparxoyn sto
              *  provlima me tin metavliti finaltempsize , pou sto telos
einai kai to
              *  megethos tou telikou mitrwou,autou diladi poy tha lavei
meros stoys
              * upologismous.
              */

             /*
              * Emfanizetai to array. for debugging.
              */
             System.out.print("\n"+ "Dof's table: 0='Restrict' , 1='Free'" +
"\n");
             for (int i=0; i<totalDOFS.length; i++) {

      System.out.print(String.valueOf(totalDOFS[i])+"\t");
             }
             System.out.print("\n");
             System.out.print("\n");
             System.out.print("\n" + "finaltemp: "+ finaltempsize + "\n" );

             /*
              * Edo kratao tous deiktes tou pinaka opou yparxei periorismos
              * se ena neo array, to opoio exei megethos oso kai o arithmos
              * ton periorismwn kai kat'epektasi, oso kai to megethos toy
              * telikoy mitrwou pou thelo na diamorfosw.
              *  H metavliti "k" apla xrisimopoieitai gia tin allagi tis
thesis
              *  tou array pou tha topothetithoun oi deiktes poy me
endiaferoun.
```

```
             * Den me apasxolei na kseperasei to megethos tou array pou
molis
             * dimiourgisa, afou ayto sigoyra tha einai oso kai oi "1" pou
             * exei to array poy kratisa tous periorismous.
             */
            double             tempfinal[][]             =             new
double[finaltempsize][finaltempsize];
            int icoord[] = new int[finaltempsize];
            icoord = calcRestrictArrays(totalDOFS,finaltempsize);

            /*
             * Print for debugging.
             */
            for (int i=0; i<icoord.length; i++) {
                  System.out.print(String.valueOf(icoord[i])+"\t");
            }
            System.out.print("\n");
            System.out.print("\n");

            /*
             * Kai etsi me aytin tin sintomi dipli "for" oi times apo
             * toys deiktes poy exoyn kratithei proigoumenos,topothetountai
             * ston teliko pinaka tou mitrwou pou thelo na dimourgisw, o
opoios
             * exei megethos, ton arithmo ton periorismon tis kataskeyis.
             */
            for (int i=0; i<icoord.length; i++) {
                  for (int j=0; j<icoord.length; j++) {
                        tempfinal[i][j]                                    =
tempstiffnessMatrix[icoord[i]][icoord[j]];
                  }
            }

            return tempfinal;
      }

      private     int[]     calcRestrictArrays(int[]     dofsRestrOrFree,int
finaltempsize) {
            int icoord[] = new int[finaltempsize];
            int k = 0;
            for (int i=0; i<dofsRestrOrFree.length; i++ ) {
                  if (dofsRestrOrFree[i] == 1) {
                        icoord[k] = i;
                        k += 1;
                  }
            }
            return icoord;
      }

      private        double[]        calcFinalForcesMatrix(truss2dProblemDef
temptr2d,double[]    ForcesMatrix,int    MatrixSize,int[]    nodesarray,int[]
dofsRestrOrFree) {
            this.tr2d = temptr2d;
            for (int i=0; i<nodesarray.length; i++ ) {
                  int N1fixedID = (tr2d.getNodes().get(i).NodeID) + 1;
                  int N1xstart = (2*N1fixedID)-2;
                  int N1xend = (2*N1fixedID)-1;
                  ForcesMatrix[N1xstart]                                   =
tr2d.getForces().calcNodeResultantX(nodesarray[i]);
                  ForcesMatrix[N1xend]                                     =
tr2d.getForces().calcNodeResultantY(nodesarray[i]);
            }
            int icoord[] = new int[MatrixSize];
            icoord = calcRestrictArrays(dofsRestrOrFree,MatrixSize);
            for (int i=0; i<MatrixSize; i++ ) {
                  FinalForcesMatrix[i] = ForcesMatrix[icoord[i]];
            }
```

```
            return FinalForcesMatrix;
      }

      private        Matrix        calcDisplacementsMatrix(truss2dProblemDef
temptr2d,double[][]  tempFinalStiffnessMatrix,double[]  tempFinalForcesMatrix)
{


            Matrix    mtrxForces    =    new    Matrix(tempFinalForcesMatrix,
FinalForcesMatrix.length);
            Matrix mtrxStiffness = new Matrix(tempFinalStiffnessMatrix);

            Matrix mtrxDesplacements = mtrxStiffness.solve(mtrxForces);

            return mtrxDesplacements;
      }

      private void FillNodeDisplacementHashMap() {

            tr2d.getNodeDisplacements().clear();

            int size = tr2d.getNodes().size();
            double[][]              StructureDisplacements              =
mtrxStructureDisplacement.getArray();
            int k = 0;
            for (int i=0; i<2*size; i=i+2 ) {

                  int nodeID = tr2d.getNodes().get(k).NodeID;
                  double oldX = tr2d.getNodes().get(k).X;
                  double oldY = tr2d.getNodes().get(k).Y;
                  double newX = oldX + StructureDisplacements[0][i];
                  double newY = oldY + StructureDisplacements[0][i+1];
                  NodeDisplacementData         nodeDD         =         new
NodeDisplacementData(nodeID, newX, newY);
                  tr2d.getNodeDisplacements().addNodeDisplacement(nodeDD);

                  k++;
            }
      }

      private void FillRodDisplacementHashMap() {

            tr2d.getRodDisplacements().clear();

            int size = tr2d.getRods().size();

            for (int i=0; i<size; i++ ) {
                  int RodID = tr2d.getRods().get(i).RodID;
                  int NodeStart = tr2d.getRods().get(i).NodeStart;
                  int NodeEnd = tr2d.getRods().get(i).NodeEnd;
                  RodDisplacementData         rodDD         =         new
RodDisplacementData(RodID, NodeStart, NodeEnd);
                  tr2d.getRodDisplacements().addDisplacementRod(rodDD);
            }
      }
}
```