ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΠΟΛΥΜΕΣΩΝ

Περιβάλλον σχεδιασμού σε vector γραφικά SVG

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

του

ΔΙΟΝΥΣΗ ΚΛΑΔΗ

Επιβλέπων : Δρ. Μαλάμος Αθανάσιος
Αναπληρωτής Καθηγητής τμήματος Ε.Π.Π.

Ηράκλειο, Ιούλιος 2010

Ευχαριστίες

Σε αυτό το σημείο θα ήθελα να ευχαριστήσω όλους όσοι με βοήθησαν και με στήριξαν όλα αυτά τα χρόνια, ώστε να κάνω το όνειρο μου πραγματικότητα και να ασχοληθώ με τον μαγικό κόσμο των υπολογιστών.

Ιδιαίτερα, θέλω να πω ένα μεγάλο ευχαριστώ στον κύριο Μαλάμο Αθανάσιο, καθηγητή και επιβλέπων της πτυχιακής μου, για την ευκαιρία που μου έδωσε να συνεργαστώ μαζί του και με όλα τα παιδιά στο Εργαστήριο Πολυμέσων του τμήματος Εφαρμοσμένης Πληροφορικής και Πολυμέσων του ΤΕΙ Κρήτης.

Περίληψη

Ο σκοπός αυτής της πτυχιακής εργασίας ήταν ο σχεδιασμός και η υλοποίηση μίας σχεδιαστικής εφαρμογής, η οποία θα χρησιμοποιηθεί για σκοπούς εσωτερικού σχεδιασμού (interior design). Η σχεδιαστική αυτή εφαρμογή βασίζεται στο πρότυπο διανυσματικών γραφικών (vector graphics) SVG (Scalable Vector Graphics). Μπορεί, δηλαδή, να δημιουργήσει και να επεξεργαστεί αρχεία που ακολουθούν το παραπάνω πρότυπο.

Συγκεκριμένα, δίνεται η δυνατότητα στο χρήστη να σχεδιάσει την κάτοψη ενός δωματίου, τοποθετώντας τοίχους, πόρτες, παράθυρα και έπιπλα. Έπειτα, το σχέδιο αυτό μπορεί να αποθηκευτεί σε μορφή SVG, έτσι ώστε να είναι ευκολότερη η μεταφορά, επεξεργασία και μετατροπή αυτού του σχεδίου και σε άλλες μορφές, όπως για παράδειγμα είναι η X3D για την τρισδιάστατη απεικόνιση του δωματίου.

Λέξεις κλειδιά: SVG, Scalable Vector Graphics, XML, vector graphics

Abstract

The purpose of this thesis was to design and implement a design application, which will be used for interior design purposes. This design application is based on the SVG (Scalable Vector Graphics) vector graphics model. That is to say, to create and edit files that follow the above model.

In particular, this application enables the user to design the layout of a room (from the top view) by placing walls, doors, windows and furniture. Then, the design can be saved in SVG format so that it's easier to transport, edit and convert it to other formats, such as the X3D model in order to show the design in three dimensions (3D)

Keywords: SVG, Scalable Vector Graphics, XML, vector graphics

Περιεχόμενα

Κατάλογος εικόνων

1 Εισαγωγή

1.1 Γενική Περιγραφή

Αυτή η πτυχιακή ασχολείται με το σχεδιασμό και την υλοποίηση μίας σχεδιαστικής εφαρμογής, η οποία διαχειρίζεται αρχεία τύπου SVG. Μέσω της εφαρμογής, ο χρήστης έχει τη δυνατότητα να σχεδιάσει σχήματα και κείμενο, τα οποία στη συνέχεια μπορούν να αποθηκευτούν με βάση το πρότυπο SVG.

Εξαιτίας της μικρής ηλικίας του προτύπου SVG, δεν υπάρχουν διαθέσιμες σχεδιαστικές εφαρμογές που να πληρούν τις απαιτήσεις μας. Για αυτό το λόγο έγινε η υλοποίηση της εφαρμογής, η οποία καλύπτει ακριβώς τις ανάγκες μας.

1.2 Στόχος της εφαρμογής

Το γενικότερο πεδίο με το οποίο θα ασχοληθεί η παρούσα πτυχιακή εργασία είναι η σχεδίαση εσωτερικών χώρων. Στόχος της εφαρμογής είναι η εκμετάλλευση των δυνατοτήτων που προσφέρει το SVG σαν πρότυπο, έτσι ώστε να μπορεί ο χρήστης να σχεδιάζει την κάτοψη ενός χώρου (μαζί με κάποια βασικά έπιπλα) και έπειτα να την αποθηκεύει σε μία μορφή εύκολα προσβάσιμη.

1.3 Εργαλεία υλοποίησης

1.3.1 NetBeans

Το προγραμματιστικό εργαλείο NetBeans IDE της Sun Microsystems χρησιμοποιήθηκε για την υλοποίηση της εφαρμογής σε Java™ γλώσσα προγραμματισμού. Με αυτό, η ανάπτυξη της εφαρμογής ήταν πιο εύκολη, αφού παρέχει πλήρη έλεγχο σε όλα τα στάδια του σχεδιασμού και της υλοποίησης.

1.3.2 *Java™*

Η εφαρμογή έχει αναπτυχθεί πλήρως σε γλώσσα προγραμματισμού Java™, μία γλώσσα που μπορεί να λειτουργήσει σε διαφορετικές πλατφόρμες και ενσωματώνει πολλές δυνατότητες. Η ανάπτυξη του γραφικού περιβάλλοντος της εφαρμογής, καθώς και όλες οι παρεχόμενες λειτουργίες της είναι γραμμένες σε Java™.

1.3.3 SVG

Το SVG είναι ένα νέο πρότυπο για διανυσματικά γραφικά. Βασίζεται στην γλώσσα XML για την περιγραφή των αντικειμένων που απαρτίζουν μία εικόνα φτιαγμένη σε SVG. Μερικά από τα πλεονεκτήματα αυτού του προτύπου είναι το μικρό μέγεθος των παραγόμενων εικόνων, η εύκολη διαχείριση τους και ενσωμάτωση τους σε ιστοσελίδες κλπ. Στο επόμενο κεφάλαιο γίνεται μία σύντομη περιγραφή των δυνατοτήτων του SVG.

Για τις πλήρεις προδιαγραφές και για περισσότερες πληροφορίες, επισκεφθείτε την ιστοσελίδα **http://www.w3.org/Graphics/SVG/**

2 Θεωρητικό Υπόβαθρο

2.1 Εισαγωγή

Στη σημερινή εποχή, τα γραφικά που έχουν παραχθεί από υπολογιστές έχουν γίνει αναπόσπαστο κομμάτι της ζωής μας. Εικόνες και φωτογραφίες που έχουν κατασκευαστεί ή επεξεργαστεί με τη βοήθεια ενός ηλεκτρονικού υπολογιστή βρίσκονται στην τηλεόραση, τις εφημερίδες, πινακίδες, και σε όποιο άλλο μέρος μπορεί να φανταστεί κανείς.

Γενικά, τα δισδιάστατα γραφικά χρησιμοποιούνται ευρέως σε τομείς που αναπτύχθηκαν κυρίως πάνω στις παραδοσιακές τεχνικές εκτύπωσης και σχεδίασης, όπως είναι η τυπογραφία, η χαρτογραφία, το τεχνικό σχέδιο, οι διαφημίσεις κλπ. Σε αυτές τις εφαρμογές, η δισδιάστατη εικόνα δεν είναι απλά η αναπαράσταση ενός πραγματικού αντικειμένου, αλλά ένα ανεξάρτητο τεχνούργημα με προστιθέμενη σημασιολογική αξία [4].

Χωρίς τα γραφικά, οι υπολογιστές δεν θα είχαν εξαπλωθεί τόσο, όσο σήμερα. Όλοι οι άνθρωποι έχουν συνηθίσει να δουλεύουν τους υπολογιστές βλέποντας εικονίδια και γραφικές αναπαραστάσεις αντικειμένων, αντί να γράφουν εντολές και κείμενο [1].

Σε γενικές γραμμές, με τον όρο γραφικά υπολογιστών εννοούμε οτιδήποτε δεν είναι ήχος ή κείμενο [1]. Είναι δηλαδή όλα τα υπόλοιπα οπτικά ερεθίσματα που μεταφέρονται στον άνθρωπο μέσω της οθόνης του ηλεκτρονικού υπολογιστή.

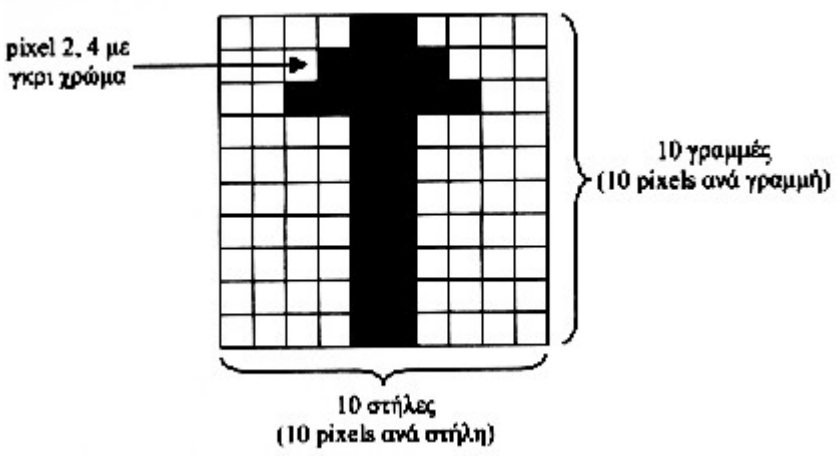
2.2 Μορφές γραφικών υπολογιστή

Τα γραφικά στους υπολογιστές μπορούν να έχουν πολλές και διάφορες μορφές. Κάνοντας μία γενική κατηγοριοποίηση, μπορούμε να χωρίσουμε στα γραφικά σε **δισδιάστατα** (**2D**) και **τρισδιάστατα** (**3D**). Όπως φανερώνουν οι ονομασίες, τα πρώτα είναι γραφικά τα οποία αποτελούνται από **δύο διαστάσεις** (**ύψος** και **πλάτος**), ενώ τα δεύτερα αποτελούνται από **τρεις διαστάσεις** (**ύψος**, **πλάτος**, **βάθος**). Επειδή η συγκεκριμένη πτυχιακή εργασία πραγματεύεται διανυσματικά γραφικά δύο διαστάσεων, δεν θα γίνει περαιτέρω ανάλυση των γραφικών τριών διαστάσεων.

Οι δύο κυριότερες μορφές δισδιάστατων γραφικών είναι τα **χαρτογραφικά** (**bitmap graphics**) και τα **διανυσματικά γραφικά** (**vector graphics**). Η κάθε μορφή χρησιμοποιείται για διαφορετικούς σκοπούς στους ηλεκτρονικούς υπολογιστές, αφού κάθε μία έχει διαφορετικά πλεονεκτήματα από την άλλη, αλλά και πολύ διαφορετικό τρόπο λειτουργίας. Σε γενικές γραμμές, οι χαρτογραφικές εικόνες εκφράζουν τα αντικείμενα που παρουσιάζουν με μικροσκοπικές κουκκίδες που ονομάζονται εικονοστοιχεία. Αντιθέτως, οι διανυσματικές εικόνες χρησιμοποιούν μαθηματικές εξισώσεις για να απεικονίσουν τα διάφορα αντικείμενα και χρώματα. Στα επόμενα κεφάλαια παρουσιάζονται αναλυτικά και οι δύο μορφές γραφικών.

2.3 Bitmap γραφικά

Τα γραφικά bitmap, όπως και οι οθόνες των ηλεκτρονικών υπολογιστών, αποτελούνται από **πλέγματα μικροσκοπικών κουκκίδων**, διατεταγμένων στο επίπεδο. Αυτές οι μικροσκοπικές κουκκίδες ονομάζονται **pixels** (η λέξη προέρχεται από τη φράση picture elements) ή εικονοστοιχεία. Ένα εικονοστοιχείο αποτελεί το ελάχιστο στοιχείο μίας εικόνας [2] [5], όπως φαίνεται και στην παρακάτω εικόνα.



Εικόνα 1: Bitmap εικόνα μεγέθους 10x10 pixels

Η εικόνα που βλέπουμε παραπάνω έχει διαιρεθεί σε 10 στήλες και 10 γραμμές, όπως ακριβώς συμβαίνει και με ένα πίνακα. Έτσι, χωρίζεται σε μικρά κομματάκια (τα

εικονοστοιχεία που είπαμε και πιο πάνω). Οι bitmap εικόνες δημιουργούνται χρωματίζοντας τα εικονοστοιχεία από τα οποία αποτελούνται. Δηλαδή, κάθε εικονοστοιχείο εκπέμπει φως σε καθορισμένες εντάσεις (εάν μιλάμε για μία ασπρόμαυρη εικόνα) ή αποχρώσεις (εάν μιλάμε για έγχρωμη εικόνα) [5] [6]. Όσο πιο κοντά βρίσκονται μεταξύ τους τα εικονοστοιχεία, τόσο πιο καλή ποιότητα θα έχει η εικόνα.

Τα χαρτογραφικά γραφικά δημιουργούνται είτε με τη χρήση **κατάλληλου λογισμικού** (Microsoft Paint, Adobe Photoshop κλπ) είτε με **ψηφιοποίηση** μίας αναλογικής εικόνας χρησιμοποιώντας το κατάλληλο υλικό (όπως είναι ο σαρωτής, η ψηφιακή φωτογραφική μηχανή ή κάμερα). Σχεδόν όλα τα λογισμικά, ακόμη και οι σύγχρονοι επεξεργαστές κειμένου, υποστηρίζουν τη δημιουργία και επεξεργασία bitmap σχεδίων. Αποτελεί δηλαδή την πιο διαδεδομένη μορφή αποθήκευσης εικόνας [5] [6].

2.3.1 *Ανάλυση εικόνας (image resolution)*

Το μέγεθος που δείχνει από πόσα εικονοστοιχεία αποτελείται μία ψηφιακή εικόνα στη μονάδα του μήκους λέγεται "ανάλυση εικόνας" [6]. Η ανάλυση εικόνας μετριέται σε ppi (pixels per inch – εικονοστοιχεία ανά ίντσα) ή αλλιώς dpi (dots per inch – κουκκίδες ανά ίντσα). Το dpi χρησιμοποιείται κυρίως στις εκτυπώσεις στο χαρτί, ενώ το ppi αναφέρεται στην παρουσίαση εικόνας σε οθόνη υπολογιστή. Για αυτό το λόγο είναι προτιμότερο να χρησιμοποιείται ο όρος ppi, αφού ασχολούμαστε με την παρουσίαση γραφικών σε οθόνη.

Όσο μεγαλύτερη είναι η ανάλυση μίας εικόνας, τόσο περισσότερα εικονοστοιχεία τη συνθέτουν και τόσο μεγαλύτερη είναι και η ποιότητα της εικόνας (αν και αυτό δεν είναι απόλυτο, αφού η ποιότητα απεικόνισης εξαρτάται σε μεγάλο βαθμό από την ανάλυση εξόδου). Το μέγεθος των αρχείων εικόνας αυξάνεται καθώς αυξάνεται η ανάλυση της [5] [6].

Η ανάλυση, όμως, μίας εικόνας δεν λέει τίποτα απολύτως για το συνολικό αριθμό εικονοστοιχείων από τα οποία αποτελείται η εικόνα. Για να υπολογίσει κάποιος το συνολικό πλήθος, πρέπει προφανώς να γνωρίζει το ακριβές αρχικό μέγεθος της εικόνας. Για αυτόν ακριβώς τον λόγο, στην ψηφιακή αναπαράσταση το μέγεθος μίας εικόνας δεν είναι σημαντικό, γιατί μπορεί να μεταβάλλεται σε σχέση και με άλλους παράγοντες (π.χ. ανάλυση εξόδου οθόνης) [6].

2.3.2 *Βάθος χρώματος*

Κάθε εικονοστοιχείο περιέχει μία πληροφορία χρώματος (δηλαδή το χρώμα που έχει) που πρέπει να αναπαραχθεί κατά την παρουσίαση της εικόνας σε κάποιο απεικονιστικό μέσο. Η πληροφορία αυτή αποθηκεύεται μέσα στο αρχείο τη εικόνας, κάνοντας χρήση κάποιο πλήθος δυαδικών ψηφίων (bits).

Άρα, το βάθος χρώματος αναφέρεται στον αριθμό των δυαδικών ψηφίων που χρησιμοποιούνται για να κωδικοποιήσουμε το εκάστοτε χρώμα της εικόνας. Όσο μεγαλύτερος είναι ο αριθμός δυαδικών ψηφίων που χρησιμοποιούμε, τόσο περισσότερες είναι οι χρωματικές αποχρώσεις που μπορεί να χρησιμοποιήσει η εικόνα. Για παράδειγμα, με βάθος χρώματος 1bit (δηλαδή τιμές 0 ή 1, αφού κάθε bit μπορεί να πάρει μονάχα αυτές τις δύο τιμές) μπορούμε να απεικονίσουμε $2^1 = 2$ χρώματα: λευκό ή μαύρο (ή ότι χρώμα επιτρέπει το κάθε απεικονιστικό μέσο). Οι εικόνες που ακολουθούν το βάθος χρώματος του παραδείγματος, δηλαδή 1bit, ονομάζονται και διτονικές [5] [6].

Οι πιο συνηθισμένες τιμές βάθους χρώματος στην εποχή μας είναι οι 8, 16 και 24 bit. Ας δούμε παρακάτω τι δυνατότητες μας δίνει το κάθε ένα.

- **8 bit χρώμα**

  Σε αυτή την περίπτωση χρησιμοποιούνται 8 bit για να περιγράψουν αποχρώσεις, άρα έχουμε $2^8 = 256$ χρώματα. Συνήθως, 8 bit χρησιμοποιούνται για να αποδοθούν κλίμακες του γκρι (grayscale), αλλά από σύστημα σε σύστημα αυτό μπορεί να διαφέρει. Στην παρακάτω εικόνα βλέπουμε δύο ίδιες εικόνες, και οι δύο με 8 bit βάθος χρώματος. Η αριστερή χρησιμοποιεί τα 8 bit για να αποδώσει 256 τόνους του γκρι, ενώ ή δεξιά εικόνα τα χρησιμοποιεί για να αποδώσει 256 διαφορετικά χρώματα.

Εικόνα 2: Εικόνα σε 8 bit βάθος χρώματος. [8]

- **16 bit χρώμα**

  Εδώ έχουμε 16 bit για την περιγραφή χρωματικών τιμών, δηλαδή $2^{16} = 65536$ διαφορετικά χρώματα. Η ονομασία με την οποία αναφερόμαστε στη συγκεκριμένη χρωματική παλέτα είναι High Color.

- **24 bit χρώμα**

  Εδώ χρησιμοποιούνται 24 bit για να περιγραφούν χρώματα. Έχουμε δηλαδή 224 = 16.777.216 = 16,7 Μ χρώματα. Η ονομασία με την οποία αναφερόμαστε στη συγκεκριμένη χρωματική παλέτα είναι Πραγματικό Χρώμα (True Color), αφού πρακτικά έχουμε στη διάθεση μας ένα μεγάλο πλήθος χρωμάτων (όλες οι αποχρώσεις που μπορεί να διακρίνει το ανθρώπινο μάτι) για να συνθέσουμε αληθοφανής εικόνες. Στην πραγματικότητα, η παλέτα αποδίδει πολύ περισσότερες αποχρώσεις, αφού το μάτι μπορεί να διακρίνει περίπου 300 – 350 χιλιάδες διαφορετικές αποχρώσεις. Στην εικόνα που ακολουθεί, βλέπουμε το ίδιο φυτό με πριν, αλλά τώρα έχει καλύτερη ποιότητα, αφού το βάθος χρώματος επιτρέπει να αποδοθούν τα πραγματικά χρώματα με ακρίβεια.

Εικόνα 3: Εικόνα με 24 bit βάθος χρώματος [8]

Δυστυχώς, αυτές οι παλέτες έχουν ένα μικρό μειονέκτημα: δεν μπορούν να αποδώσουν διαφάνεια. Για το λόγο αυτό υπάρχει ακόμα μία χρωματική παλέτα που χρησιμοποιεί **32 bit** για την απόδοση χρωμάτων. Τα 8 επιπλέον δυαδικά ψηφία που ενσωματώνει χρησιμοποιούνται για την απόδωση διαφάνειας (**alpha channel**). Επίσης, υπάρχουν και παλέτες με πολλά περισσότερα δυαδικά ψηφία (π.χ. 48 ή 96 bit) αλλά χρησιμοποιούνται σε τεχνικές επεξεργασίας εικόνας σε επαγγελματικό επίπεδο [6].

Τέλος, να πούμε πως, όπως και με την ανάλυση τη εικόνας, έτσι και το βάθος χρώματος επιρεάζει άμεσα το μέγεθος του τελικού αρχείου που περιέχει την εικόνα. Επομένως, όσο καλύτερη παλέτα χρησιμοποιούμε (δηλαδή με περισσότερα δυαδικά ψηφία) τόσο μεγαλύτερο θα είναι το τελικό μέγεθος.

2.3.3 Ανάλυση εξόδου [6]

Όπως αναφέρθηκε και παραπάνω, το πως θα παρουσιαστεί μία χαρτογραφική εικόνα στην οθόνη του υπολογιστή δεν εξαρτάται μόνο από την ανάλυση της εικόνας, αλλά και σε πολύ μεγάλο βαθμό από την ανάλυση εξόδου της συσκευής εξόδου (π.χ. οθόνη) του υπολογιστή.

Με τον όρο ανάλυση εξόδου αναφερόμαστε στο μέγεθος που μας δείχνει πόσα εικονοστοιχεία παρουσιάζει η οθόνη σε κάθε διάσταση της, δηλαδή πλάτος και ύψος. Συγκεκριμένα, η ανάλυση εξόδου εκφράζεται σαν γινόμενο δύο αριθμών, οι οποίοι εκφράζουν τα εικονοστοιχεία που παρουσιάζονται κατά πλάτος και τα εικονοστοιχεία που παρουσιάζονται κατά ύψος. Δηλαδή, μία ανάλυση εξόδου 800x600 μας δείχνει πως η οθόνη κατά τη λειτουργία της χρησιμοποιεί 800 εικονοστοιχεία στην οριζόντια διάσταση της (πλάτος) και 600 εικονοστοιχεία στην κατακόρυφη διάσταση της (ύψος).

Το πόσα εικονοστοιχεία παρουσιάζονται κάθε φορά σε κάθε διάσταση της οθόνης, δηλαδή ποιά ανάλυση εξόδου χρησιμοποιεί κάθε φορά η οθόνη καθορίζεται από το υποσύστημα εικόνας του υπολογιστή (κάρτα γραφικών και οθόνη). Πιο συγκεκριμένα, ο χρήστης μπορεί να επιλέξει την ανάλυση στην οποία θα λειτουργεί η οθόνη του. Οι πιο συνηθισμένες τιμές ανάλυσης εξόδους παρουσιάζονται στον παρακάτω πίνακα.

Ανάλυση Εξόδου	Pixels κατά πλάτος	Pixels κατά ύψος	Χαρακτηρισμός
640 x 480	640	480	VGA
800 x 600	800	600	SVGA
1024 x 768	1024	768	SVGA
1280 x 1024	1280	1024	SVGA
1600 x 1200	1600	1200	UXGA

Πίνακας 1: Συνήθεις τιμές ανάλυσης εξόδους οθόνης

Τέλος, η σχέση της ανάλυσης εικόνας με την ανάλυση εξόδου της οθόνης καθορίζει τον τρόπο και την ποιότητα παρουσίασης της ψηφιακής εικόνας σε μία οθόνη.

2.3.4 Είδη χαρτογραφικών εικόνων [6]

Σε γενικές γραμμές υπάρχουν διάφορα είδη χαρτογραφικών εικόνων, τα οποία είτε έχουν διαφορετικό πλήθος χρωμάτων το καθένα είτε αποθηκεύουν την χρωματική πληροφορία με διαφορετικό τρόπο. Παρακάτω παρουσιάζονται διάφορα είδη:

- **Διτονική (Bitonal)**

Εικόνα 4: Διτονική εικόνα με βάθος χρώματος 1 bit [8]

Μία διτονική εικόνα έχει βάθος χρώματος 1 bit και επομένως χρησιμοποιεί δύο χρωματικούς τόνους (π.χ. άσπρο και μάυρο) για κάθε εικονοστοιχείο.

- **Μονόχρωμη (Grayscale)**

Εικόνα 5: Μονόχρωμη εικόνα, με βάθος χρώματος 8 bit [8]

Πρόκειται για εικόνες που δημιουργούνται από τόνους του γκρι (από απόλυτο μαύρο μέχρι απόλυτο λευκό) και έχουν βάθος χρώματος 8 bit (δηλαδή εμφανίζουν 256 τόνους του γκρι).

- **Έγχρωμη (Colour image)**

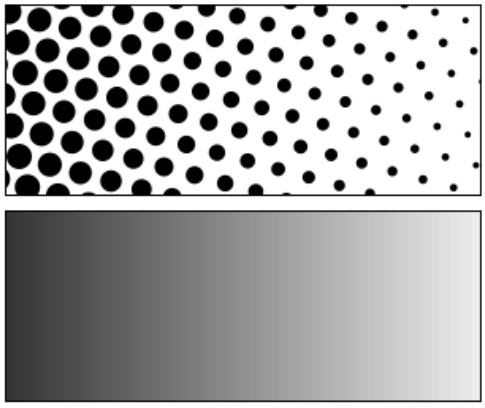
Εικόνα 6: Έγχρωμη εικόνα, με βάθος χρώματος 24 bit [8]

Στην έγχρωμη χαρτογραφική εικόνα η πληροφορία χρώματος για κάθε εικονοστοιχείο αναλύεται σε τρεις συνιστώσες, δηλαδή πληροφορία για κάθε ένα από τα τρία πρωτεύοντα χρώματα του μοντέλου RGB. Χρησιμοποιώντας 8 bit για κάθε χρώμα (κόκκινο, πράσινο, μπλε) έχουμε βάθος χρώματος 24 bit. Επομένως, μία τέτοια εικόνα μπορεί να απεικονίσει 16,7 εκατομμύρια διαφορετικά χρώματα.

- **<u>Δεικτοδοτούμενου χρώματος (Indexed colour)</u>**

  Αυτού του είδους εικόνες χρησιμοποιούν 8 bit για να περιγράψουν τις αποχρώσεις, δηλαδή 256 χρώματα. Το σημαντικό εδώ, όμως, είναι πως η πληροφορία για κάθε εικονοστοιχείο δεν εκφράζει το χρώμα αυτό καθεαυτό αλλά παραπέμπει (είναι δηλαδή δείκτης) σε θέσεις ενός πίνακα που περιέχει τα 256 χρώματα. Με αυτό τον τρόπο μπορούμε να έχουμε χρωματική ποικιλία με μικρότερο μέγεθος αρχείου συγκριτικά με άλλες μορφές εικόνας.

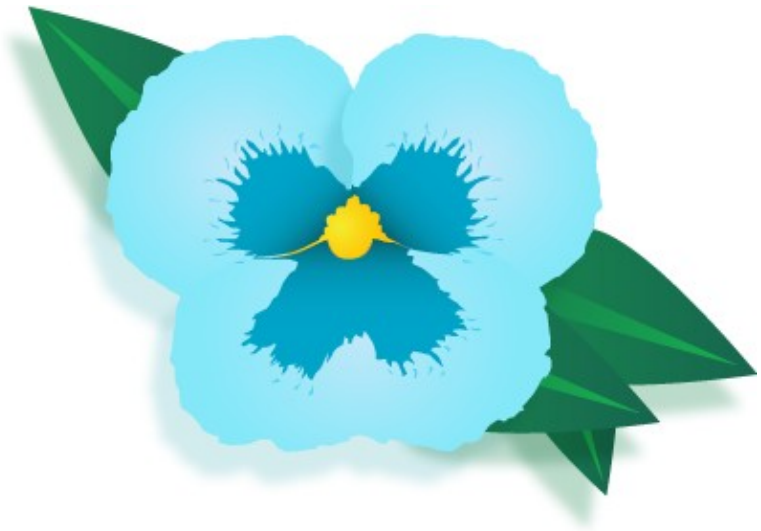
- **<u>Συνεχούς τόνου (Continuous tone) & Halftone</u>**



Εικόνα 7: Πάνω: οι κουκκίδες halftone. Κάτω: πως θα έβλεπε το ανθρώπινο μάτι αυτή τη διάταξη των κουκκίδων από απόσταση [9]

Οι χαρτογραφικές εικόνες που ακολουθούν αυτή τη μορφή παράγονται από τις συσκευές εξόδου (οθόνες και εκτυπωτές) δημιουργώντας κουκκίδες (γκρι ή έγχρωμες) σε τέτοιες πυκνότητες και τέτοιες αποχρώσεις ώστε να αναπαριστούν ικανοποιητικά την εικόνα. Κάποιες συσκευές μπορούν να προβάλλουν εικονοστοιχεία όπου η αλλαγή του γκρι ή του χρώματος είναι συνεχής. Οι εικόνες αυτές χαρακτηρίζονται εικόνες συνεχούς τόνου. Αντίθετα ορισμένες συσκευές παράγουν κουκκίδες μόνο ενός τόνου. Σε αυτή την περίπτωση οι διάφορες αποχρώσεις του γκρι ή του χρώματος αποδίδονται ρυθμίζοντας την πυκνότητα ή το σχήμα των κουκκίδων. Οι εικόνες αυτού του είδους ονομάζονται halftone images.

2.4 Διανυσματικά (vector) γραφικά



Εικόνα 8: Εικόνα με διανυσματικά γραφικά (ευγενική προσφορά της Karuna51 από το deviantArt.com) [10]

Όπως είπαμε και πιο πάνω, τα χαρτογραφικά αρχεία μπορεί να γίνουν αρκετά μεγάλα, σπαταλώντας έτσι πολύτιμο χώρο στον ηλεκτρονικό υπολογιστή. Ανάλογα το σκοπό για τον οποίο θέλουμε να φτιάξουμε ένα γραφικό, πολλές φορές χρησιμοποιούμε τα διανυσματικά γραφικά, τα οποία είναι συνήθως μικρότερα σε μέγεθος. Με τον όρο διανυσματικά γραφικά εννοούμε τη χρήση απλών γεωμετρικών σχημάτων, όπως είναι τα σημεία, οι γραμμές, οι καμπύλες, σχήματα και πολύγωνα, τα οποία βασίζονται σε μαθηματικές εξισώσεις για να συνθέσουν και να παρουσιάσουν εικόνες [2].

Τα αρχεία που περιέχουν διανυσματικά γραφικά κρατούν αποθηκευμένες τις γραμμές, τα σχήματα και τα χρώματα που συνθέτουν μία εικόνα σαν **μαθηματικές φόρμουλες**. Ένα πρόγραμμα διανυσματικών γραφικών χρησιμοποιεί αυτές τις μαθηματικές φόρμουλες για να κατασκευάσει την εικόνα στην οθόνη, δίνοντας την καλύτερη δυνατή ποιότητα σε σχέση με την ανάλυση της οθόνης [2].

Οι μαθηματικές αυτές φόρμουλες προσδιορίζουν το που θα τοποθετηθούν οι "τελείες" που συνθέτουν την εικόνα έτσι ώστε να πάρουμε το καλύτερο ποιοτικό αποτέλεσμα όταν εμφανίζουμε την εικόνα. Αφού οι φόρμουλες μπορούν να παράξουν μία εικόνα που μπορεί να έχει οποιοδήποτε μέγεθος και οποιαδήποτε λεπτομέρεια, η ποιότητα της περιορίζεται μόνο από την ανάλυση της οθόνης. Επίσης, το μέγεθος του αρχείο που περιέχει τα διανυσματικά δεδομένα παραμένει σταθερό, όσο μεγάλη και λεπτομερής κι αν είναι η εικόνα [2].

2.5 Επεξεργασία διανυσματικών γραφικών

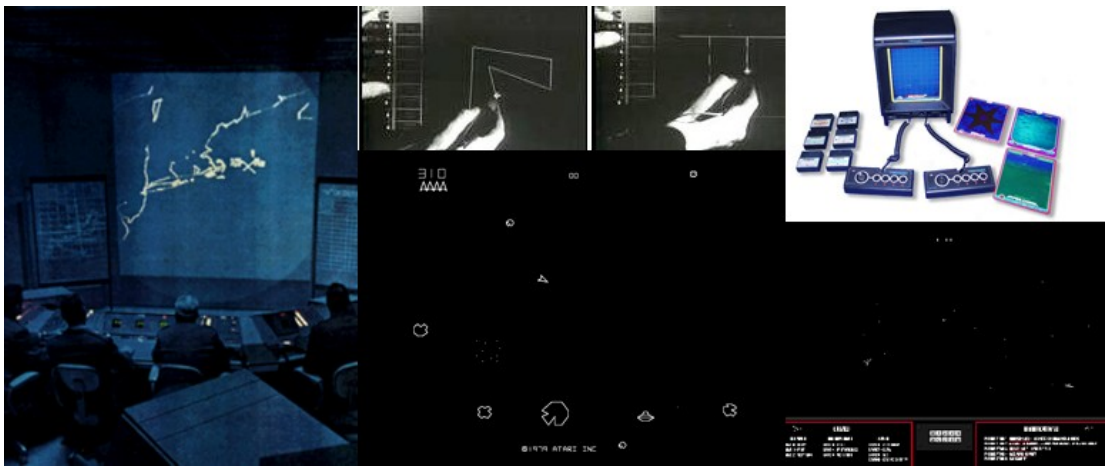
Μία εικόνα σχεδιασμένη με διανυσματικά γραφικά μπορεί να δεχτεί αλλαγές με την επεξεργασία των αντικειμένων στην οθόνη (χρησιμοποιώντας κατάλληλο λογισμικό). Έπειτα, οι αλλαγές αυτές αποθηκεύονται σαν μετατροπές στην μαθηματική φόρμουλα. Επίσης, μαθηματικοί τελεστές μπορούν να χρησιμοποιηθούν, έτσι ώστε να παραμορφωθεί η εικόνα (ή μέρος αυτής) [2].

2.6 Πρότυπα διανυσματικών γραφικών

Σε γενικές γραμμές υπάρχουν αρκετά πρότυπα διανυσματικών γραφικών, άλλα είναι ανοικτά και άλλα κλειστά. Παραδείγματα κλειστών προτύπων είναι το **AI** του προγράμματος Adobe Illustrator, όπως επίσης και το **SWF** της Adobe και πάλι (χρησιμοποιείται κυρίως στο Flash). Ένα άλλο πρότυπο είναι αυτό που χρησιμοποιεί η Microsoft είναι το **VML** [2].

Τέλος, το ανοικτό πρότυπο που δημιουργήθηκε για να εκπληρώσει τις ανάγκες για μία ευπροσάρμοστη, με δυνατότητες scripting και για όλες τις χρήσεις διανυσματική μορφή είναι το **SVG** (**Scalable Vector Graphics**). Το SVG δημιουργήθηκε και αναπτύχθηκε από το **World Wide Web Consortium** (**W3C**) [2].

2.7 Εφαρμογές διανυσματικών γραφικών [2]



Εικόνα 9: Διάφορες εφαρμογές των διανυσματικών γραφικών (πηγή εικόνων: Wikipedia)

Μία από τις πρώτες εφαρμογές των διανυσματικών γραφικών έγινε στο σύστημα αεροπορικής άμυνας **SAGE** των Ηνωμένων Πολιτειών της Αμερικής. Τα διανυσματικά γραφικά αποσύρθηκαν μόνο από την υπηρεσία ελέγχου εναέριας κυκλοφορίας των

Ηνωμένων Πολιτειών της Αμερικής το 1999, αλλά είναι πολύ πιθανό να χρησιμοποιούνται ακόμα για στρατιωτικούς λόγους και ειδικά συστήματα.

Επίσης, τα διανυσματικά γραφικά χρησιμοποιήθηκαν και στο σύστημα **TX-2** στο εργαστήριο **Linkoln** του **MIT** (Massachusetts Institute of Technology, Ινστιτούτο Τεχνολογίας Μασαχουσέτης), από τον πρωτοπόρο στα γραφικά υπολογιστών **Ivan Sutherland** για την λειτουργία του προγράμματος **Sketchpad** που αυτός έφτιαξε το 1963.

Τα διανυσματικά γραφικά έχουν χρησιμοποιηθεί και σε πολλά παιχνίδια. Υπήρχε μία οικιακή παιχνιδομηχανή με το όνομα **Vectrex** που χρησιμοποιούσε τέτοια γραφικά. Επίσης, διάφορα γνωστά παιχνίδια, όπως το **Asteroids** και το **Space Wars** έκαναν χρήση διανυσματικών γραφικών.

Τέλος, διανυσματικά γραφικά χρησιμοποιούνται σε **laser light show**, όπου δύο καθρέπτες X-Y κινούνται με μεγάλη ταχύτητα, ώστε να σχηματίσουν γρήγορα σχήματα και κείμενο σε μία οθόνη προβολής.

2.7 Εισαγωγή στο SVG

Το πρότυπο **Scalable Vector Graphics** (SVG) είναι μία οικογένεια προδιαγραφών (με βάση το πρότυπο **XML**) για την περιγραφή δισδιάστατων διανυσματικών γραφικών (και στατικών και δυναμικών). Είναι ένα ανοικτό πρότυπο, δημιουργημένο από το **W3C** το 1999 [3].

Οι εικόνες SVG και οι δράσεις που μπορούν αυτές να εκτελέσουν ορίζονται μέσα σε αρχεία XML. Αυτό σημαίνει πως μπορεί να γίνει αναζήτηση, καταγραφή και (εάν είναι απαραίτητο) συμπίεση σε αυτά. Επίσης, μπορούν να ενσωματωθούν scripts. Επίσης, επειδή οι εικόνες SVG είναι στην ουσία αρχεία XML, μπορούν να δημιουργηθούν με οποιοδήποτε πρόγραμμα επεξεργασίας κειμένου (π.χ. Notepad, Notepad++ κλπ.). Ειδικά σχεδιαστικά προγράμματα που να υποστηρίζουν το πρότυπο SVG υπάρχουν [3].

Όλοι οι σύγχρονοι browsers υποστηρίζουν εγγενώς το πρότυπο SVG έτσι ώστε να μπορούν να εμφανίζουν τις ανάλογες εικόνες. Λαμπρή εξαίρεση αποτελεί ο Internet Explorer, ο οποίος δεν υποστηρίζει καθόλου το πρότυπο SVG. Αυτό, βέβαια θα αλλάξει με την έλευση της έκδοσης 9...[3]

Το SVG υποστηρίζει τρεις τύπους γραφικών αντικειμένων:

1. Διανυσματικά γραφικά
2. Γραφικά raster
3. Κείμενο

Από το 2001 και έπειτα, η προδιαγραφή του SVG αναβαθμίστηκε στην έκδοση 1.1, η οποία είναι και η τρέχουσα έκδοση, και στην έκδοση 1.2 (υπό ανάπτυξη). Η πρόταση για το **SVG Mobile** εισήγαγε δύο απλοποιημένα προφίλ του SVG 1.1, τα **SVG Basic** και **SVG Tiny**. Τα εν λόγω προφίλ προορίζονται για συσκευές που έχουν μειωμένη επεξεργαστική ισχύ και περιορισμένες δυνατότητες απεικόνισης. Το SVG Tiny έγινε αργότερα αυτόνομη πρόταση και είναι η βάση για το SVG 1.2 [3].

2.8 Η λειτουργικότητα του SVG [3]

Το πρότυπο SVG 1.1 ορίζει 14 σημαντικούς τομείς λειτουργικότητας:

- **Paths**

  Είναι περιγράμματα απλών ή περίπλοκων σχημάτων τα οποία είναι καμπύλα ή ευθύγραμμα. Αυτά τα σχήματα μπορούν να γεμίσουν με χρώμα ή να παραμείνουν διάφανα στο εσωτερικό τους. Ορίζονται με πολύ σύντομες εντολές. Για παράδειγμα, το **M**, προέρχεται από τη φράση **move to** (**μετακίνηση σε**) και ακολουθείται από ένα ζευγάρι συντεταγμένων (**X,Y**). Άλλα γράμματα – εντολές είναι τα **L, C, S, Q, T, A**, που χρησιμοποιούνται για το σχεδιασμό γραμμών, Bezier γραμμών και ελλειπτικών γραμμών. Τέλος, η εντολή **Z** χρησιμεύει στο κλείσιμο του path. Σε όλες τις περιπτώσεις, οι εντολές με κεφαλαίο γράμμα (π.χ. **M**) ακολουθούνται από απόλυτες συντεταγμένες, ενώ οι εντολές με μικρό γράμμα (π.χ. **m**) ακολουθούνται από σχετικές συντεταγμένες.

- **Βασικά σχήματα**

  Είναι σχήματα που αποτελούνται από ευθύγραμμα τμήματα, πολύγωνα, κύκλοι και ελλείψεις. Επίσης, υπάρχουν ορθογώνια και ορθογώνια με στρογγυλεμένες άκρες.

- **Κείμενο**

  Οι χαρακτήρες **unicode** που περιλαμβάνονται σε ένα αρχείο SVG εκφράζονται σαν **δεδομένα χαρακτήρων XML**. Διάφορα οπτικά εφέ μπορούν να εφαρμοστούν σε ένα κείμενο. Επίσης, το SVG υποστηρίζει και διαχειρίζεται αυτόματα κείμενο **δύο κατευθύνσεων** (π.χ. ελληνικά και αραβικά ταυτόχρονα), **κάθετο κείμενο** και **κείμενο που ακολουθεί κάποια καμπύλη** (π.χ. το κείμενο σε μία σφραγίδα του κράτους).

- **Ζωγραφική (painting)**

  Όλα τα σχήματα που υποστηρίζει το SVG μπορούν να γεμίσουν στο εσωτερικό τους με **χρώμα**, όπως επίσης μπορούν να δεχτούν **περίγραμμα**. Όλα αυτά μπορεί να είναι με σκέτο **χρώμα**, με **ντεγκραντέ** (**gradient**) ή με κάποιο **μοτίβο**. Τα γεμίσματα μπορούν να είναι αδιαφανή ή να έχουν διάφορες διαβαθμίσεις διαφάνειας.

- **Χρώμα**

  Όπως αναφέρθηκε και παραπάνω, σε όλα τα σχήματα μπορούν να εφαρμοστούν χρώματα. Τα χρώματα ορίζονται με τέσσερις διαφορετικούς τρόπους: όπως το πρότυπο **CSS2** (δηλαδή **blue**, **white** κλπ), στο **δεκαεξαδικό** (**#ff03a1**), στο **δεκαδικό** (**rgb(255, 34, 123)**) και σαν **ποσοστά** (**rgb(3%, 45%, 1%)**)

- **Gradients και μοτίβα (patterns)**

  Τα σχήματα, όπως είπαμε και πιο πάνω, εκτός από χρώμα, μπορούν να γεμίσουν με **gradient** ή επαναλαμβανόμενα **μοτίβα**. Τα gradients μπορεί να είναι **γραμμικά** (**linear**) ή **κυκλικά** (**radial**) και δέχονται οποιοδήποτε πλήθος χρωμάτων και επαναλήψεων. Επίσης, gradients που επηρρεάζουν τη διαφάνεια ενός σχήματος επιτρέπονται. Τα μοτίβα βασίζονται σε **raster** ή **vector** γραφικά αντικείμενα, τα οποία επαναλαμβάνονται ως προς τον **άξονα X** ή/και τον **άξονα Y**. Και τα gradients και τα μοτίβα μπορούν να γίνουν **animated** και **scripted**.

- **Clipping, Masking και Compositing**

  Όλα τα γραφικά αντικείμενα, συμπεριλαμβανομένου τα κείμενα, τα βασικά σχήματα όπως και **οποιοσδήποτε συνδυασμός** τους, μπορούν να χρησιμοποιηθούν σαν όρια για να οριστούν οι **"εσωτερικές"** και **"εξωτερικές"** περιοχές που θα χρωματιστούν (με χρώμα, gradient ή μοτίβο).

- **Φίλτρα Εφέ (filter effects)**

  Τα φίλτρα εφέ είναι μία σειρά από **γραφικές λειτουργίες**, οι οποίες εφαρμόζονται σε όποιο σχήμα SVG επιθυμούμε για να παράγουμε **raster** εικόνες με συγκεκριμένες οπτικές ιδιότητες. Για παράδειγμα, μπορούμε να θολώσουμε μία εικόνα (blur) ή να την παραμορφώσουμε. Ονομαστικά, τα φίλτρα που υποστηρίζει το SVG είναι: Blend, Color Matrix, Component transfer, Composite, Convolve matrix, Diffuse lighting, Displacement map, Flood, Gaussian blur, Image, Merge, Morphology, Offset, Specular lighting, Tile, Turbulence.

- **Διαδραστικότητα (interactivity)**

  Οι εικόνες SVG μπορούν να δια δράσουν με το χρήστη μέσω διάφορων τρόπων. Εκτός από τους υπερ-συνδέσμους (αναλύονται παρακάτω), οποιοδήποτε μέρος του SVG μπορεί να ρυθμιστεί ώστε να παράγει **γεγονότα** (**events**) που αναπαριστούν αλλαγές στην εστίαση, κλικ του ποντικιού, όπως επίσης και άλλα γεγονότα από πολλές πηγές (όπως το πληκτρολόγιο κλπ). Οι **διαχειριστές γεγονότων** (**event handlers**) έχουν τη δυνατότητα να **ξεκινούν**, να **σταματούν** και να **αλλάζουν** animations, ή να **ενεργοποιούν** άλλα **scripts** σαν απόκριση σε κάθε γεγονός.

- **Σύνδεσιμότητα (linking)**

  Για τη δημιουργία εσωτερικών ή εξωτερικών συνδέσμων στο SVG χρησιμοποιείται η γλώσσα **XLink** (**XML Linking Language**), η οποία βασίζεται φυσικά στην XML. Υπάρχουν δύο είδη συνδέσμων: οι **απλοί σύνδεσμοι** και οι **σύνθετοι σύνδεσμοι**. Το πρότυπο SVG χρησιμοποιεί τους απλούς συνδέσμους, αλλά το πρότυπο SVG 1.2 που είναι υπό ανάπτυξη προτείνει τη χρήση σύνθετων συνδέσμων XLink. Οι απλοί σύνδεσμοι δημιουργούν **υπερ-συνδέσμους μονής κατεύθυνσης** από ένα στοιχείο προς ένα

άλλο με τη χρήση **URI** (**Uniform Resource Identifier**).

- **Scripting**

  Όλες οι πτυχές του SVG (σχήματα, κείμενα κλπ) μπορούν να προσεγγιστούν και να αλλαχθούν με τη χρήση scripts όπως γίνεται και στην γλώσσα HTML. Η προεπιλεγμένη γλώσσα είναι η **ECMAScript**, η οποία είναι κοντινός **συγγενής** της **JavaScript**, και περιέχει αντικείμενα **DOM** (**Document Object Model**) για όλα τα στοιχεία και τις ιδιότητες του SVG. Τα scripts μπορούν να εκτελεσθούν σαν απόκριση σε γεγονότα του ποντικιού, του πληκτρολογίου κλπ.

- **Σχεδιοκίνηση (animation)**

  Το περιεχόμενο SVG μπορεί να γίνει **animated** με τη χρήση στοιχείων animation όπως είναι τα **<animate>**, **<animateMotion>** και **<animateColor>**. Επίσης, σχεδιοκίνηση μπορεί να δημιουργηθεί χρησιμοποιώντας και "πειράζοντας" αντικείμενα DOM μέσω scripts. Τα animations του SVG είναι **πλήρως συμβατά** με την τρέχουσα έκδοση, όπως επίσης και με όλες τις μελλοντικές εκδόσεις, της **Synchronized Multimedia Integration Language** (**SMIL**). Τα animations μπορούν να είναι **συνεχόμενα**, **επαναλαμβανόμενα** και να **αντιδρούν** σε γεγονότα.

- **Γραμματοσειρές**

  Όπως συμβαίνει με την HTML και τα CSS, έτσι και στο SVG, το κείμενο μπορεί να αναφερθεί και να χρησιμοποιήσει **εξωτερικές γραμματοσειρές** (π.χ. γραμματοσειρές συστήματος). Εάν η γραμματοσειρά στη οποία αναφέρεται το SVG δεν υπάρχει στο σύστημα που προσπαθεί να εμφανίσει την εικόνα, τότε το κείμενο μπορεί να μην παρουσιαστεί έτσι όπως πρέπει. Για να εξαλειφθεί αυτός ο περιορισμός, το κείμενο μπορεί να χρησιμοποιήσει ένα **SVG Font** (**γραμματοσειρά SVG**), όπου η μορφή των χαρακτήρων της απαραίτητης γραμματοσειράς ορίζεται σαν SVG σχήμα και ενσωματώνεται μέσα στο ίδιο αρχείο. Έτσι, το κείμενο μπορεί να χρησιμοποιήσει αυτή τη γραμματοσειρά αντί την αυθεντική που μπορεί να απουσιάζει από το σύστημα.

- **Metadata**

  Το SVG μπορεί να ενσωματώσει **metadata** (**μετα-δεδομένα**) για την καλύτερη περιγραφή της εικόνας. Αυτό γίνεται με τη χρήση του στοιχείου **<metadata>**, όπου το αρχείο περιγράφεται με ιδιότητες που ακολουθούν τις προδιαγραφές **Dublin Core**.

2.9 Ορισμοί του SVG [7]

Το πρότυπο SVG 1.1 ορίζει τα παρακάτω αντικείμενα και λειτουργίες:

- **Βασικά σχήματα**

  Τυπικά σχήματα τα οποία είναι προκαθορισμένα στο SVG για διευκόλυνση σε κοινές γραφικές λειτουργίες. Συγκεκριμένα έχουμε: **rect, circle, ellipse, line, polyline, polygon**.

- **Καμβάς (canvas)**

  Είναι μία επιφάνεια, πάνω στην οποία ζωγραφίζονται τα γραφικά στοιχεία. Μπορεί να έχει φυσική υπόσταση, όπως είναι το χαρτί ή μία κορνίζα, ή μία αφηρημένη επιφάνεια, όπως είναι μία κατοχυρωμένη περιοχή της μνήμης του υπολογιστή.

- **Μονοπάτι απόκρυψης (clipping path)**

  Είναι ο συνδυασμός κειμένου, μονοπατιών (path) και βασικών σχημάτων τα οποία σχηματίζουν το περίγραμμα μίας μάσκας 1 bit, όπου ότι βρίσκεται εντός του περιγράμματος μπορεί να εμφανιστεί, αλλά οτιδήποτε βρίσκεται εκτός του περιγράμματος απορρίπτεται.

- **Στοιχείο υποδοχέας (container element)**

  Είναι ένα στοιχείο, το οποίο μπορεί να γραφικά στοιχεία ή/και άλλα στοιχεία υποδοχείς. Συγκεκριμένα έχουμε: **svg, g, defs, symbol, clipPath, mask, pattern, marker, a, switch**.

- **Τρέχων εσωτερικό τμήμα αρχείου SVG (current innermost SVG document fragment)**

  Είναι το υπο-δέντρο του εγγράφου XML που ξεκινά με το αμέσως προηγούμενο **svg** στοιχείο ενός δεδομένου στοιχείου SVG.

- **Τρέχων τμήμα αρχείου SVG (current SVG document fragment)**

  Είναι το υπο-δέντρο του εγγράφου XML που ξεκινά με το άκρως εξωτερικό στοιχείο **svg** ενός δεδομένου στοιχείου SVG, με την προϋπόθεση ότι όλα τα στοιχεία υποδοχείς μεταξύ του εξωτερικού svg και του τρέχοντος στοιχείου είναι στοιχεία στη γλώσσα SVG.

- **Τρέχων πίνακας μετασχηματισμού (current transformation matrix (CTM))**

  Οι πίνακες μετασχηματισμού ορίζουν τη μαθηματική χαρτογράφηση από ένα

σύστημα συντεταγμένων σε ένα άλλο, χρησιμοποιώντας πίνακες 3x3 με βάση την εξίσωση **[x' y' 1] = [x y 1] * πίνακας**. Ο τρέχων πίνακας μετασχηματισμού ορίζει τη χαρτογράφηση από το σύστημα συντεταγμένων του χρήστη στο σύστημα συντεταγμένων της οθόνης.

- **Γέμισμα (fill)**

  Είναι η λειτουργία του γεμίσματος του εσωτερικού ενός σχήματος ή το εσωτερικό των χαρακτήρων σε ένα κείμενο.

- **Γραμματοσειρά (font)**

  Μία γραμματοσειρά είναι μία οργανωμένη συλλογή από λογότυπους, μέσα στην οποία όλοι οι λογότυποι μοιράζονται μία κοινή εμφάνιση, έτσι ώστε όταν οι χαρακτήρες εμφανίζονται όλοι μαζί, το αποτέλεσμα να είναι εξαιρετικά ευανάγνωστο, να αποπνέει ένα συγκεκριμένο καλλιτεχνικό στυλ και να παρέχει μεταξύ των χαρακτήρων συνεπή ευθυγράμμιση και απόσταση.

- **Λογότυποι (glyphs)**

  Ένας λογότυπος αναπαριστά μία μονάδα περιεχομένου μέσα από μία γραμματοσειρά. Συχνά, υπάρχει μία αντιστοιχία 1 προς 1 μεταξύ των χαρακτήρων που θα εμφανιστούν και των αντίστοιχων λογότυπων (δηλαδή, ο χαρακτήρας Α εμφανίζεται χρησιμοποιώντας ένα μόνο λογότυπο), αλλά μερικές φορές πολλαπλοί λογότυποι χρησιμοποιούνται για να εμφανιστεί ένας χαρακτήρας ή πολλαπλοί χαρακτήρες εμφανίζονται χρησιμοποιώντας ένα μόνο λογότυπο. Τυπικά, ένας λογότυπος ορίζεται από ένα ή περισσότερα σχήματα.

- **Στοιχείο γραφικών (graphics element)**

  Είναι ένας από τους τύπους στοιχείων που έχει τη δυνατότητα να ζωγραφίσει γραφικά πάνω σε ένα συγκεκριμένο καμβά. Συγκεκριμένα έχουμε: **path, text, rext, circle, ellipse, line, polyline, polygon, image, use**.

- **Στοιχείο αναφοράς σε γραφικά (graphics referencing element)**

  Είναι ένα στοιχείο γραφικών, το οποίο χρησιμοποιεί μία αναφορά σε ένα διαφορετικό έγγραφο ή στοιχείο σαν πηγή του γραφικού του περιεχομένου. Συγκεκριμένα έχουμε: **use, image**.

- **Αναφορά τοπικού URI (local URI reference)**

  Είναι ένα URI (**Uniform Resource Identifier**) που δεν περιλαμβάνει **<absoluteURI>** ή **<relativeURI>**, με αποτέλεσμα να αποτελεί αναφορά σε ένα στοιχείο μέσα στο τρέχων έγγραφο.

- **Μάσκα (mask)**

  Είναι ένα στοιχείο υποδοχέας που μπορεί να περιέχει γραφικά στοιχεία ή άλλα στοιχεία υποδοχείς, τα οποία ορίζουν ένα σετ από γραφικά που σαν ήμι-διάφανη μάσκα, έτσι ώστε να συντεθούν αντικείμενα του προσκήνιου με το τρέχων φόντο.

- **Αναφορά μη τοπικού URI (non-local URI reference)**

  Είναι ένα URI (**Uniform Resource Identifier**) που περιλαμβάνει **<absoluteURI>** ή **<relativeURI>**, με αποτέλεσμα να αποτελεί αναφορά σε ένα διαφορετικό έγγραφο ή ένα στοιχείο σε διαφορετικό έγγραφο.

- **Βαφή (paint)**

  Η βαφή είναι ένας τρόπος ώστε να μπουν χρωματικές τιμές μέσα στον καμβά. Μία βαφή μπορεί να αποτελείται ταυτόχρονα από χρωματικές τιμές και τιμές άλφα (alpha values – διαφάνεια), έτσι ώστε να ελέγχεται η ανάμιξη των χρωμάτων πάνω στον καμβά. Το SVG υποστηρίζει τρεις τύπους βαφής: **color, gradients, patterns**.

- **Χαρακτηριστικό παρουσίασης (presentation attribute)**

  Είναι ένα XML χαρακτηριστικό ενός στοιχείου SVG που ορίζει μία τιμή για μία συγκεκριμένη ιδιότητα του στοιχείου.

- **Ιδιότητα (property)**

  Είναι μία παράμετρος που βοηθά στο να ορίσουμε πως ένα έγγραφο θα εμφανιστεί. Οι ιδιότητες εκχωρούνται σε στοιχεία στη γλώσσα SVG είτε μέσω των χαρακτηριστικών παρουσίασης των στοιχείων είτε χρησιμοποιώντας μία γλώσσα διαμόρφωσης, όπως είναι η CSS.

- **Σχήμα (shape)**

  Είναι ένα γραφικό στοιχείο, το οποίο ορίζεται από ένα συνδυασμό γραμμών και καμπυλών. Συγκεκριμένα έχουμε: **path, rect, circle, ellipse, line, polyline, polygon**.

- **Περίγραμμα (stroke)**

  Είναι η διαδικασία της σχεδίασης του περιγράμματος σε ένα σχήμα ή χαρακτήρα (όταν μιλάμε για κείμενο).

- **Καμβάς SVG (SVG canvas)**

  Είναι ο καμβάς πάνω στον οποίο εμφανίζεται το SVG περιεχόμενο.

- **Τμήμα αρχείου SVG (SVG document fragment)**

  Είναι το υπό-δέντρο του εγγράφου XML, το οποίο ξεκινά με ένα στοιχείο **svg**. Ένα τμήμα αρχείου SVG μπορεί να αποτελείται από ένα ξεχωριστό SVG έγγραφο ή ένα τμήμα κάποιου γονικού εγγράφου XML, το οποίο περικλείεται μέσα σε ένα στοιχείο **svg**. Όταν ένα στοιχείο svg είναι απόγονος κάποιου άλλου στοιχείου svg, τότε υπάρχουν δύο τμήματα αρχείου SVG, ένα για κάθε στοιχείο svg (δηλαδή, το ένα τμήμα αρχείου SVG περικλείεται μέσα στο άλλο τμήμα).

- **Τμήμα θέασης SVG (SVG viewport)**

  Είναι το τμήμα θέασης μέσα στον SVG καμβά το οποίο ορίζει μία ορθογώνια περιοχή, μέσα στην οποία εμφανίζεται το SVG περιεχόμενο.

- **Στοιχείο κειμένου (text content element)**

  Είναι ένα από τα στοιχεία στο SVG που μπορούν να ορίσουν μία σειρά κειμένου, η οποία θα εμφανιστεί πάνω στον καμβά. Συγκεκριμένα έχουμε: **text, tspan, tref, textPath, altGlyph**.

- **Μετασχηματισμός (transformation)**

  Είναι μία αλλαγή του τρέχοντα πίνακα μετασχηματισμού, η οποία γίνεται με το να παρέχουμε ένα συμπληρωματικό μετασχηματισμό, στη μορφή ομαδοποιημένων απλούστερων μετασχηματισμών (όπως είναι η κλίμακα, η περιστροφή ή η μετακίνηση) ή/και ενός ή περισσότερων πινάκων μετασχηματισμού.

- **Πίνακας μετασχηματισμού (tranformation matrix)**

  Οι πίνακες μετασχηματισμού ορίζουν τη μαθηματική χαρτογράφηση από ένα σύστημα συντεταγμένων σε ένα άλλο, χρησιμοποιώντας πίνακες 3x3 με βάση την εξίσωση **[x' y' 1] = [x y 1] * πίνακας**.

- **Αναφορά URI (URI reference)**

  Είναι ένα URI (**Uniform Resource Identifier**) που ενεργεί σαν αναφορά σε ένα αρχείο ή σε ένα στοιχείο μέσα σε ένα αρχείο.

- **User agent**

  Ο γενικός ορισμός του user agent είναι ότι πρόκειται για μία εφαρμογή η οποία ανακτά και εμφανίζει διαδικτυακό περιεχόμενο, συμπεριλαμβανομένου κείμενα, γραφικά, ήχους, βίντεο, εικόνες και διάφορους άλλους τύπους περιεχομένου. Ένα user agent μπορεί να απαιτεί πρόσθετα user agents, τα οποία διαχειρίζονται κάποιους τύπους περιεχομένου. Για παράδειγμα, ένας φυλλομετρητής (browser) μπορεί να καλέσει ένα ξεχωριστό πρόγραμμα ή πρόσθετο εργαλείο για να

προβάλλει ήχο ή βίντεο. Ένα user agent μπορεί να έχει (ή μπορεί και να μην έχει) τη δυνατότητα να προβάλλει περιεχόμενο SVG. Ένα SVG user agent προβάλλει περιεχόμενο SVG πάντα.

- **Σύστημα συντεταγμένων χρήστη (user coordinate system)**

  Σε γενικές γραμμές, ένα σύστημα συντεταγμένων ορίζει τοποθεσίες και αποστάσεις πάνω στον τρέχων καμβά. Το τρέχων σύστημα συντεταγμένων χρήστη είναι το σύστημα συντεταγμένων που είναι αυτή τη στιγμή ενεργό και το οποίο χρησιμοποιείται για να ορίσουμε πως οι συντεταγμένες και τα μήκη τοποθετούνται και υπολογίζονται, αντίστοιχα, πάνω στον τρέχων καμβά.

- **Χώρος χρήστη (user space)**

  Είναι συνώνυμο του συστήματος συντεταγμένων χρήστη (user coordinate system).

- **Μονάδες χρήστη (user units)**

  Μία τιμή συντεταγμένων ή μήκους, εκφρασμένη σε μονάδες χρήστη αναπαριστά μία τιμή συντεταγμένων ή μήκους στο τρέχων σύστημα συντεταγμένων χρήστη. Δηλαδή, 10 μονάδες χρήστη αναπαριστούν 10 μονάδες του συστήματος συντεταγμένων του χρήστη.

- **Τμήμα θέασης (viewport)**

  Είναι μία ορθογώνια περιοχή μέσα στον τρέχοντα καμβά μέσα στην οποία τα γραφικά στοιχεία θα προβληθούν.

- **Σύστημα συντεταγμένων τμήματος θέασης (viewport coordinate system)**

  Γενικά, ένα σύστημα συντεταγμένων ορίζει τις θέσεις και τις αποστάσεις πάνω στον τρέχων καμβά. Το σύστημα συντεταγμένων του τμήματος θέασης είναι το σύστημα συντεταγμένων, το οποίο είναι ενεργό κατά την έναρξη της επεξεργασίας ενός στοιχείου **svg,** και πριν την επεξεργασία του προαιρετικού χαρακτηριστικού viewBox. Στην περίπτωση που ένα τμήμα εγγράφου SVG, το οποίο είναι ενσωματομένο μέσα στο γονικό έγγραφο που χρησιμοποιεί CSS για τη διαχείριση του layout του, το σύστημα συντεταγμένων του τμήματος θέασης θα έχει τον ίδιο προσανατολισμό και ίδια μήκη με αυτά που ορίζονται μέσα στο CSS. Η αρχή των αξόνων του θα είναι στο πάνω αριστερό μέρος του τμήματος θέασης.

- **Χώρος τμήματος θέασης (viewport space)**

  Είναι συνώνυμο του συστήματος συντεταγμένων τμήματος θέασης (viewport coordinate system).

- **Μονάδες τμήματος θέασης (viewport units)**

  Μία τιμή συντεταγμένων ή μήκους, εκφρασμένη σε μονάδες τμήματος θέασης αναπαριστά μία τιμή συντεταγμένων ή μήκους στο σύστημα συντεταγμένων του τμήματος θέασης. Δηλαδή, 10 μονάδες χρήστη αναπαριστούν 10 μονάδες του συστήματος συντεταγμένων του τμήματος θέασης.

2.10 Παραδείγματα κώδικα SVG

Σε αυτό το κεφάλαιο γίνεται μία σύντομη παρουσίαση του πως δομείται ένα έγγραφο SVG και τι αποτέλεσμα δίνει ο εκάστοτε κώδικας, μέσω κάποιων παραδειγμάτων. Όπως βλέπουμε, το SVG ακολουθεί τη λογική και τη δομή ενός οποιουδήποτε αρχείου XML.

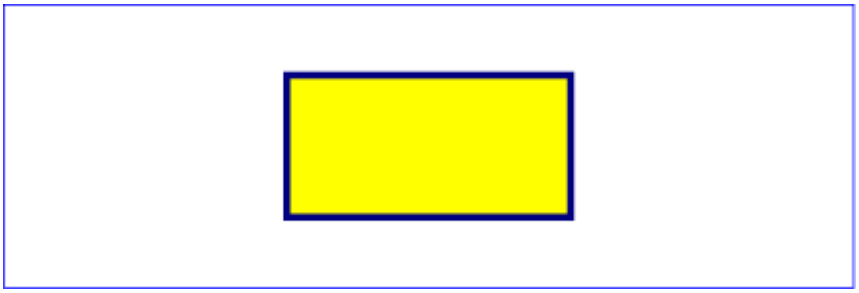
Έχουμε δηλαδή τις ετικέτες (tags), τα χαρακτηριστικά και τις ιδιότητες τους. Όλα τα αρχεία SVG για να προβάλλουν κάποιο περιεχόμενο, θα πρέπει αυτό το περιεχόμενο να περιέχεται μέσα στο tag **<svg>**. Έπειτα, χρησιμοποιώντας τις κατάλληλες εντολές, μπορούμε να συνθέτουμε διανυσματικές εικόνες ή σχεδιοκινήσεις.

2.10.1 Δημιουργία ορθογωνίου

Γράφοντας μέσα σε ένα αρχείο svg τον παρακάτω κώδικα παίρνουμε ένα ορθογώνιο, σαν και αυτό που φαίνεται στην εικόνα που ακολουθεί.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <desc>Example rect01 - rectangle with sharp corners</desc>
  <!-- Show outline of canvas using 'rect' element -->
  <rect x="1" y="1" width="1198" height="398"
        fill="none" stroke="blue" stroke-width="2"/>
  <rect x="400" y="100" width="400" height="200"
        fill="yellow" stroke="navy" stroke-width="10"  />
</svg>
```

Κώδικας για τη δημιουργία ενός ορθογώνιου



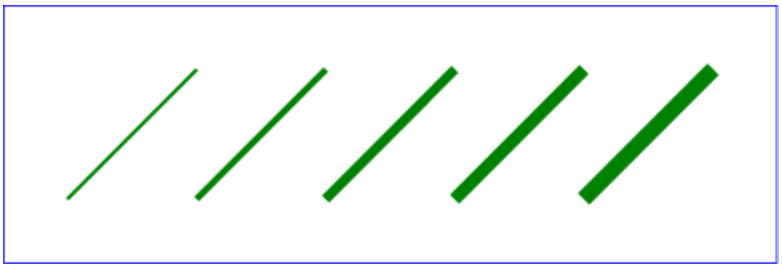
Εικόνα 10: Ορθογώνιο δημιουργημένο με την εντολή <rect>

2.10.2 Δημιουργία γραμμών

Γράφοντας μέσα σε ένα αρχείο svg τον παρακάτω κώδικα παίρνουμε πέντε γραμμές, με διαφορετικό πάχος κάθε μία. Παρακάτω, ακολουθεί η αντίστοιχη εικόνα.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <desc>Example line01 - lines expressed in user coordinates</desc>
  <!-- Show outline of canvas using 'rect' element -->
  <rect x="1" y="1" width="1198" height="398"
        fill="none" stroke="blue" stroke-width="2" />
  <g stroke="green" >
    <line x1="100" y1="300" x2="300" y2="100"
            stroke-width="5"  />
    <line x1="300" y1="300" x2="500" y2="100"
            stroke-width="10"  />
    <line x1="500" y1="300" x2="700" y2="100"
            stroke-width="15"  />
    <line x1="700" y1="300" x2="900" y2="100"
            stroke-width="20"  />
    <line x1="900" y1="300" x2="1100" y2="100"
            stroke-width="25"  />
  </g>
</svg>
```

Κώδικας για τη δημιουργία γραμμών



Εικόνα 11: Γραμμές που φτιάχτηκαν με την εντολή <line>

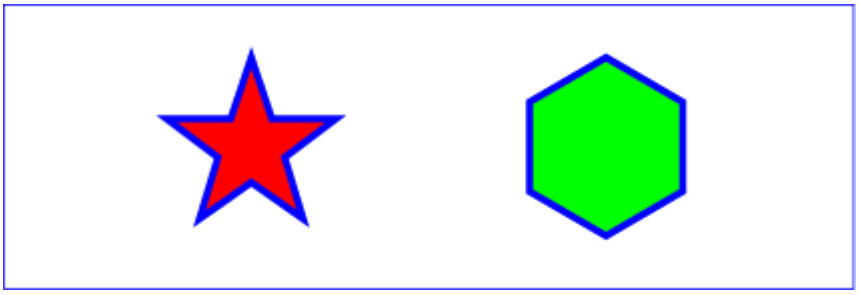
2.10.3 Δημιουργία πολύγωνων

Γράφοντας μέσα σε ένα αρχείο svg τον παρακάτω κώδικα παίρνουμε δύο διαφορετικά πολύγωνα (ένα αστέρι και ένα εξάγωνο) γεμισμένα με χρώμα. Παρακάτω, ακολουθεί η αντίστοιχη εικόνα.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
```

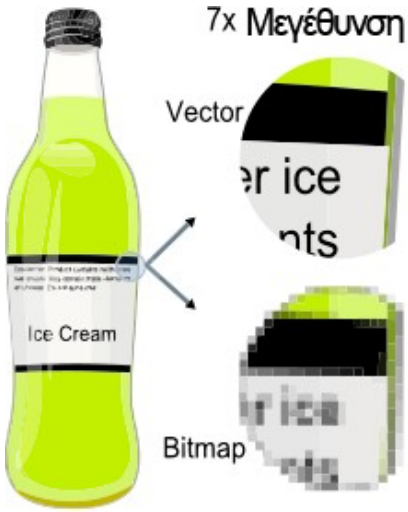
```
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <desc>Example polygon01 - star and hexagon</desc>
  <!-- Show outline of canvas using 'rect' element -->
  <rect x="1" y="1" width="1198" height="398"
        fill="none" stroke="blue" stroke-width="2" />
  <polygon fill="red" stroke="blue" stroke-width="10"
           points="350,75  379,161 469,161 397,215
                   423,301 350,250 277,301 303,215
                   231,161 321,161" />
  <polygon fill="lime" stroke="blue" stroke-width="10"
           points="850,75  958,137.5 958,262.5
                   850,325 742,262.6 742,137.5" />
</svg>
```

Κώδικας για τη δημιουργία πολύγωνων



Εικόνα 12: Πολύγωνα που κατασκευάστηκαν με την εντολή <polygon>

2.11 Χαρτογραφική Εικόνα εναντίον Διανυσματικής Εικόνας [6]



Εικόνα 13: Μία από τις διαφορές μεταξύ χαρτογραφικής και διανυσματικής εικόνας [2]

Όπως είδαμε στα προηγούμενα κεφάλαια, τα γραφικά στους υπολογιστές χωρίζονται σε δύο κατηγορίες: τα χαρτογραφικά γραφικά και τα διανυσματικά γραφικά. Παρακάτω παρουσιάζονται οι διαφορές των δύο αυτών ειδών γραφικών.

- **Φωτορεαλιστικότητα εικόνων**

  Βασικά πλεονεκτήματα των χαρτογραφικών εικόνων είναι πως αποδίδουν μεγάλη ποκιλία χρωμάτων επιτυγχάνοντας έτσι την πιστότερη φωτορεαλιστική απόδοση της εικόνας. Ακόμη μπορεί κανείς αν θέλει να επεξεργαστεί και το κάθε εικονοστοιχείο ξεχωριστά. Αντίθετα, οι διανυσματικές εικόνες αποδίδουν περιορισμένο αριθμό χρωμάτων και δεν μπορούν να παραστήσουν ικανοποιητικά εικόνες συνεχούς τόνου με πληθώρα χρωμάτων.

- **Ανεξαρτησία από την ανάλυση**

  Οι διανυσματικού τύπου εικόνες χρησιμοποιούν μαθηματικές συναρτήσεις για να περιγράψουν το περιεχόμενο τους και δεν έχουν μία καθορισμένη ανάλυση. Θα προβληθούν με τον ίδιο τρόπο ανεξάρτητα από την ανάλυση εξόδου της οθόνης του συστήματος σε αντίθεση με τις χαρτογραφικές εικόνες όπου η ποιότητα παρουσίασης τους εξαρτάται από την ανάλυση εξόδου.

Είδος Εικόνας	Συνηθισμένες χρήσεις	Πλεονεκτήματα
Χαρτογραφικές	Εικόνες συνεχούς τόνου Εκτεταμένη χρήση σε ιστοσελίδες	Υψηλός βαθμός φωτορεαλισμού στην απόδοση της εικόνας
Διανυσματικές	Σε περιπτώσεις εικόνων με λίγα σχετικά χρώματα που χρειάζεται να παρουσιαστούν σωστά σε διάφορες αναλύσεις Προγράμματα 3D και CAD	Ανεξάρτητες από την ανάλυση Ομαλή αναπαράσταση καμπυλών Μικρό μέγεθος αρχείου

Πίνακας 2: Διαφορές χαρτογραφικών και διανυσματικών εικόνων

2.12 Σχεδίαση εσωτερικών χώρων

Όπως είπαμε και στο πρώτο κεφάλαιο, το πεδίο εφαρμογών στο οποίο απευθύνεται η παρούσα πτυχιακή εργασία είναι αυτό της σχεδίασης εσωτερικών χώρων. Σε αυτή την ενότητα θα ρίξουμε μία γρήγορη ματιά στο συγκεκριμένο πεδίο.

Ο κλάδος της εσωτερικής σχεδίασης χώρων είναι ένα πολύπλευρο επάγγελμα, στο οποίο δημιουργικές και τεχνικές λύσεις συνδυάζονται και εφαρμόζονται σε ένα οικοδόμημα με σκοπό τη δημιουργία ενός εσωτερικού περιβάλλοντος. Η διαδικασία της σχεδίασης εσωτερικών χώρων ακολουθεί μία συστηματική και συντονισμένη μεθοδολογία, η οποία περιλαμβάνει την έρευνα, την ανάλυση και την ενσωμάτωση της γνώσης μέσα στη

δημιουργική διαδικασία, με στόχο να εκπληρωθούν οι ανάγκες και οι οικονομικοί στόχοι του πελάτη, έτσι ώστε να δημιουργηθεί ένας εσωτερικός χώρος που ακολουθεί τους στόχους του αρχικού σχεδιασμού [11].

Πολλοί άνθρωποι χρησιμοποιούν τους όρους "σχεδίαση εσωτερικών χώρων" και "διακόσμηση εσωτερικών χώρων" για να αναφερθούν στο ίδιο πράγμα. Αυτά τα δύο επαγγέλματα, όμως, διαφέρουν σε διάφορους σημαντικούς τομείς [12].

Η σχεδίαση εσωτερικών χώρων είναι η τέχνη και η επιστήμη του να καταλάβεις τις συνήθειες των ανθρώπων, έτσι ώστε να δημιουργηθούν χρηστικοί χώροι μέσα σε ένα κτίριο. Η διακόσμηση είναι η επίπλωση ή/και το στόλισμα του χώρου με στιλάτα και όμορφα πράγματα. Σε γενικές γραμμές, οι σχεδιαστές εσωτερικών χώρων μπορούν να διακοσμήσουν, σε αντίθεση με τους διακοσμητές, οι οποίοι δεν μπορούν να σχεδιάσουν [12].

Οι σχεδιαστές εσωτερικών χώρων εφαρμόζουν δημιουργικές και τεχνικές λύσεις μέσα σε ένα οικοδόμημα, οι οποίες είναι λειτουργικές, ελκυστικές και επωφελείς για την ποιότητα ζωής και την κουλτούρα των κατοίκων [12].

2.13 Σχεδίαση με τη βοήθεια υπολογιστή (CAD)

Για να εφαρμόσει τις γνώσεις του, ένας σχεδιαστής εσωτερικών χώρων, χρειάζεται και τα κατάλληλα εργαλεία. Αυτά τα εργαλεία τα προσφέρει ο τομέας της **σχεδίασης με τη βοήθεια υπολογιστή** (**Computer Aided Design – CAD**).

Με τον όρο "σχεδίαση με τη βοήθεια υπολογιστή" εννοούμε τη χρήση της τεχνολογίας υπολογιστών για τη σχεδίαση αντικειμένων, πραγματικών ή φανταστικών. Στην CAD, όμως, συχνά χρησιμοποιούνται περισσότερα πράγματα από απλά σχήματα. Όπως και στα τεχνικά εγχειρίδια και τα μηχανολογικά σχέδια, στο τελικό αποτέλεσμα μίας διαδικασίας CAD ενσωματώνονται και συμβολικές πληροφορίες, όπως είναι για παράδειγμα υλικά, διαδικασίες, διαστάσεις και ανοχές, σύμφωνα με τις εκάστοτε συμβάσεις της κάθε εφαρμογής [13].

Η διαδικασία σχεδίασης με τη βοήθεια υπολογιστή μπορεί να χρησιμοποιηθεί για τη σχεδίαση καμπύλων ή σχεδίων σε δύο διαστάσεις (2D) ή καμπυλών, επιφανειών και στερεών σωμάτων σε τρεις διαστάσεις (3D) [13].

Η σχεδίαση με τη βοήθεια υπολογιστή είναι μία σημαντική βιομηχανική τέχνη, η οποία χρησιμοποιείται εκτενώς σε πολλές εφαρμογές, οι οποίες περιλαμβάνουν βιομηχανίες αυτοκίνησης, ναυπηγικής και αεροδιαστημικής, βιομηχανικό και αρχιτεκτονικό σχέδιο και πολλά άλλα. Η διαδικασία της σχεδίασης με τη βοήθεια υπολογιστή χρησιμοποιείται επίσης και για την παραγωγή ειδικών εφφέ σε ταινίες, διαφημίσεις και τεχνικά εγχειρίδια. Η σύγχρονη πανταχού παρουσία και η δύναμη των υπολογιστών σημαίνει πως ακόμα και τα μπουκαλάκια των αρωμάτων και οι διάφορες συσκευασίες σχεδιάζονται με τη χρήση τεχνικών που οι μηχανικοί του 1960 δεν μπορούσαν ούτε να φανταστούν. Εξαιτίας της σημαντικής οικονομικής σημασίας, η σχεδίαση με τη βοήθεια υπολογιστή έχει αναδειχτεί σε κινητήρια δύναμη όσον αφορά την έρευνα στην υπολογιστική γεωμετρία, τα γραφικά υπολογιστών (είτε μιλάμε για υλικό, είτε μιλάμε για λογισμικό) και τη διακριτή διαφορική γεωμετρία.

3 Υλοποίηση Εφαρμογής

3.1 Οι εργαλειοθήκες [16]

Όταν αναφερόμαστε σε μία **εργαλειοθήκη γραφικής διεπαφής χρήστη** (widget toolkit, widget library, GUI toolkit), αναφερόμαστε σε μία συλλογή **εργαλείων** (widgets) με τα οποία σχεδιάζουμε τη γραφική διεπαφή κάποιας εφαρμογής. Η εργαλειοθήκη, αυτή καθεαυτή, είναι ένα κομμάτι λογισμικού, το οποίο είναι παροχή του λειτουργικού συστήματος, του συστήματος παραθύρων (windowing system) ή του διαχειριστή παραθύρων (window manager). Αυτό το κομμάτι λογισμικού παρέχει ένα **API** (application programming interface) στα προγράμματα, με σκοπό τα τελευταία να χρησιμοποιούν τα εργαλεία της εργαλειοθήκης. Κάθε εργαλείο διευκολύνει μία συγκεκριμένη διάδραση μεταξύ ανθρώπου και υπολογιστή, και εμφανίζεται σαν ένα εμφανές τμήμα της γραφικής διεπαφής χρήστη (GUI) του υπολογιστή.

Τα εργαλεία που παρέχει μία εργαλειοθήκη υπακούν σε μία ενοποιημένη προδιαγραφή σχεδιασμού, συμπεριλαμβανόμενης της αισθητικής, έτσι ώστε να δίνουν μία αίσθηση μίας συνολικής συνοχής ανάμεσα στα διάφορα μέρη της εφαρμογής, και ανάμεσα στις διάφορες εφαρμογές που βρίσκονται υπό την αιγίδα του GUI.

Οι εργαλειοθήκες περιέχουν, επίσης, κατάλληλο λογισμικό, ώστε να διευκολύνεται η δημιουργία διαχειριστών παραθύρων, μιας και τα παράθυρα θεωρούνται και αυτά εργαλεία που περιέχονται στην εργαλειοθήκη. Μερικά εργαλεία υποστηρίζουν τη διάδραση με το χρήστη, όπως είναι οι ετικέτες (labels) τα κουμπιά (buttons) και τα κουτιά μαρκαρίσματος (check boxes). Κάποια άλλα εργαλεία λειτουργούν σαν υποδοχείς, οι οποίοι ομαδοποιούν τα εργαλεία που προστίθενται σε αυτούς (π.χ. παράθυρα (windows), πλαίσια (panels) κλπ).

Η γραφική διεπαφή χρήστη ενός προγράμματος είναι συνήθως κατασκευασμένη με **επικαλυπτόμενο** τρόπο (cascading), με τα εργαλεία να προστίθενται κατευθείαν πάνω σε ήδη υπάρχοντα εργαλεία. Σε πολλές υλοποιήσεις, τα παράθυρα της εφαρμογής προστίθενται

κατευθείαν πάνω στην επιφάνεια εργασίας από τον διαχειριστή παραθύρων, και μπορούν να στοιβάζονται σε επίπεδα το ένα πάνω στο άλλο με διάφορους τρόπους. Κάθε παράθυρο σχετίζεται με μία συγκεκριμένη εφαρμογή, η οποία ελέγχει τα εργαλεία που προστίθενται στον καμβά της.

Οι περισσότερες εμπορικές εργαλειοθήκες τον **αντικειμενοστραφή προγραμματισμό** σαν μοντέλο διάδρασης με το χρήστη. Η εργαλειοθήκη χειρίζεται **γεγονότα** (events), όπως είναι για παράδειγμα όταν ο χρήστης κάνει με το ποντίκι του κλικ σε ένα κουμπί. Μόλις ανιχνευθεί ένα γεγονός, αυτό μεταφέρεται αμέσως στην εφαρμογή όπου γίνεται και ο χειρισμός του.

Ο σχεδιασμός τέτοιων εργαλειοθηκών έχει επικριθεί για την προώθηση ενός υπέρ-απλουστευμένου μοντέλου δράσης-αποτελέσματος (event-action), οδηγώντας τους προγραμματιστές στο να δημιουργούν εφαρμογές που είναι επιρρεπής σε λάθη, δύσκολες στην επέκταση και περιλαμβάνουν υπερβολικά πολύπλοκο πηγαίο κώδικα.

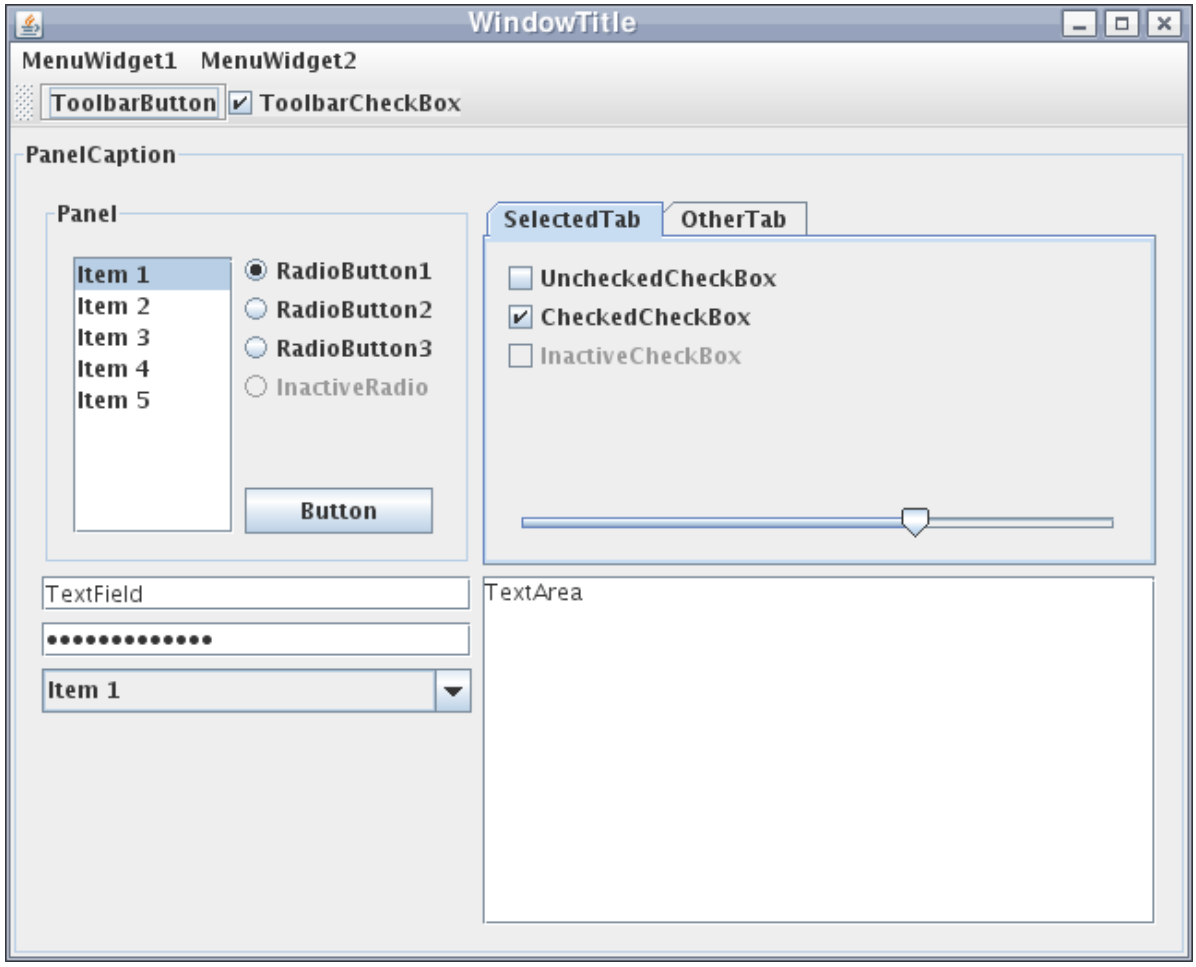
Η **εμφάνιση** (look and feel) των εργαλείων μπορεί να ενσωματωθεί στον κώδικα της εργαλειοθήκης, μερικές όμως εργαλειοθήκες αποσυνδέουν την εμφάνιση από τον ορισμό των εργαλείων, με αποτέλεσμα αυτά να μπορούν να δεχτούν **θέματα** (themes, αλλαγή της εμφάνισης τους).

Μερικές εργαλειοθήκες είναι οι ακόλουθες: wxWidgets, Juce, FLTK, FOX toolkit, GTK+, IUP, JX Application Framework, Microsoft Foundation Classes, Motif, Object Windows Library & OWLNext, Qt, Standard Widget Toolkit, **Swing**, Tk, Ultimate+++, Visual Component Library, Xforms.

3.2 Η εργαλειοθήκη Swing [15]

Το **Swing** είναι η εργαλειοθήκη που χρησιμοποιείται κατά κόρων στην γλώσσα προγραμματισμού Java™. Είναι μέρος του **Java Foundation Classes** της Sun Microsystems, το οποίο είναι ένα API (application programming interface) για τα προγράμματα Java™.

Το Swing αναπτύχθηκε για να παρέχει ένα πιο εξελιγμένο σετ στοιχείων μίας γραφικής διεπαφής χρήστη (GUI components) από το παλαιότερο Abstract Window Toolkit (AWT). Το Swing παρέχει μία έμφυτη εμφάνιση (look and feel) που **μιμείται** την εμφάνιση διάφορων πλατφορμών. Επίσης, υποστηρίζει **θέματα**, έτσι ώστε η εμφάνιση της εφαρμογής να μην σχετίζεται με την υποκείμενη πλατφόρμα.



Εικόνα 14: Παράδειγμα εφαρμογής που κάνει χρήση του Swing για την πλατφόρμα X Window System

3.3 Η αρχιτεκτονική του Swing [15]

Το Swing είναι μία εργαλειοθήκη ανεξάρτητη από τις πλατφόρμες, που ακολουθεί την αρχιτεκτονική Model – View – Controller (βλ. επόμενη ενότητα) για την Java™. Το Swing έχει τα παρακάτω πλεονεκτήματα:

- **Ανεξαρτησία από πλατφόρμες**

  Το Swing είναι ανεξάρτητο από οποιαδήποτε πλατφόρμα και ως προς την έκφραση του (Java) και ως προς την υλοποίηση του.

- **Επεκτάσιμο**

  Το Swing είναι μία **υψηλά διαχωρισμένη** αρχιτεκτονική, η οποία επιτρέπει την ενσωμάτωση διάφορων εξατομικευμένων υλοποιήσεων των διεπαφών του framework. Δηλαδή, οι χρήστες μπορούν να παρέχουν τις δικές τους εξατομικευμένες υλοποιήσεις κάποιων στοιχείων, έτσι ώστε να παρακάμψουν την

προκαθορισμένη υλοποίηση. Γενικά, οι χρήστες του Swing μπορούν να το επεκτείνουν με το να επεκτείνουν τις υπάρχουσες κλάσεις και/ή να παρέχουν εναλλακτικές υλοποιήσεις των βασικών στοιχείων.

Το Swing είναι ένα framework που βασίζεται σε **στοιχεία** (components). Η διάκριση μεταξύ των αντικειμένων και των στοιχείων του είναι ένα πολύ λεπτό σημείο. Συνοπτικά, ένα στοιχείο είναι ένα καλοφτιαγμένο αντικείμενο με ένα πολύ συγκεκριμένο μοτίβο συμπεριφοράς. Τα αντικείμενα του Swing παράγουν γεγονότα ασύγχρονα, έχουν ενσωματωμένες ιδιότητες και ανταποκρίνονται σε ένα πολύ συγκεκριμένο σετ εντολών (συγκεκριμένες για κάθε στοιχείο). Συγκεκριμένα, τα στοιχεία του Swing είναι στοιχεία **Java Beans** που εναρμονίζονται με τις προδιαγραφές Java Beans Component Architecture.

- **Προσαρμόσιμο**

  Με δεδομένο το προγραμματιστικό μοντέλο απεικόνισης του Swing, υπάρχει η δυνατότητα για ακριβή έλεγχο στις λεπτομέρειες απεικόνισης ενός στοιχείου του Swing. Σαν γενικό μοτίβο, η οπτική αναπαράσταση ενός στοιχείου του Swing είναι μία σύνθεση κοινών σχημάτων, όπως είναι ένα περίγραμμα, μία εσοχή ή κάποια "διακόσμηση". Τυπικά, οι χρήστες προγραμματιστικά θα προσαρμόσουν ένα κοινό στοιχείο του Swing (π.χ. ένα JTable) με το να εκχωρούν συγκεκριμένα περιγράμματα, χρώματα, υπόβαθρα, διαφάνειες κλπ, σαν ιδιότητες αυτού του στοιχείου. Μετά, το στοιχείο του πυρήνα του Swing θα χρησιμοποιήσει αυτές τις ιδιότητες για να καθορίσει την κατάλληλη απεικόνιση, ώστε να "ζωγραφίζει" τα διάφορα αντικείμενα.

- **Παραμετροποιήσιμο**

  Η μεγάλη εξάρτηση του Swing με τους μηχανισμούς εκτέλεσης (runtime mechanisms) και έμμεσης σύνθεσης (indirect composition) του επιτρέπει να ανταποκρίνεται στη διάρκεια εκτέλεσης (runtime) σε θεμελιώδεις αλλαγές στις ρυθμίσεις του. Για παράδειγμα, μία εφαρμογή που χρησιμοποιεί αντικείμενα του Swing μπορεί κατά τη διάρκεια της εκτέλεσης της να αλλάξει την εμφάνιση της (look and feel).

- **Ελαφρύ**

  Η δυνατότητα παραμετροποίησης του Swing είναι αποτέλεσμα της επιλογής να μην γίνει χρήση των αντικειμένων GUI του μητρικού λειτουργικού συστήματος για την εμφάνιση του. Το Swing "ζωγραφίζει" τα αντικείμενα του προγραμματιστικά μέσα από τη χρήση του API Java 2D, αντί να καλέσει την εργαλειοθήκη του χρήστη. Έτσι, ένα στοιχείο του Swing δεν έχει ένα αντίστοιχο στοιχείο στην εργαλειοθήκη του λειτουργικού συστήματος, και είναι ελεύθερο να απεικονίσει τον εαυτό του με οποιοδήποτε δυνατό τρόπο με τα παρεχόμενα API γραφικών.

Παρόλα αυτά, κάθε στοιχείο του Swing, στον πυρήνα του, βασίζεται σε ένα

υποδοχέα του AWT (Abstract Window Toolkit). Αυτό γίνεται γιατί το JComponent (του Swing) επεκτείνει το Container (του AWT). Αυτό επιτρέπει στο Swing να ενσωματωθεί στη δομή διαχείρισης της γραφικής διεπαφής χρήστη του λειτουργικού συστήματος, με αποτέλεσμα να έχει πρόσβαση στις ζωτικής σημασίας χαρτογραφήσεις μεταξύ συσκευής/οθόνης και δράσεων του χρήστη (π.χ. πάτημα κουμπιών ή κινήσεις του ποντικιού). Το Swing απλά μεταφέρει τη δική του (αποσυνδεδεμένη από το λειτουργικό σύστημα) σημασιολογία (semantics) στα υποκείμενα (συνδεδεμένα με το λειτουργικό σύστημα) στοιχεία. Έτσι, για παράδειγμα, κάθε στοιχείο του Swing ζωγραφίζει την παρουσίαση του στη συσκευή γραφικών σαν απόκριση σε μία κλήση στη συνάρτηση component.paint(), η οποία ορίζεται στο Container (του AWT). Αλλά, σε αντίθεση με τα στοιχεία του AWT, τα οποία εξουσιοδοτούν την εμφάνιση τους στα "βαριά" και συνδεδεμένα με το λειτουργικό σύστημα στοιχεία, τα στοιχεία του Swing είναι αυτά καθεαυτά υπεύθυνα για την απεικόνιση τους.

- **"Χαλαρά συνδεδεμένο"**

  Η βιβλιοθήκη Swing κάνει ευρεία χρήση του σχεδιαστικού μοτίβου Model – View – Controller για εφαρμογές. Αυτό, εννοιολογικά αποσυνδέει τα δεδομένα που θα εμφανιστούν με τα στοιχεία διεπαφής χρήστη τα οποία θα εμφανίσουν τα δεδομένα. Εξαιτίας αυτού, τα περισσότερα στοιχεία του Swing έχουν συναφή μοντέλα (που προσδιορίζονται με βάση τους όρους των διεπαφών/interfaces της Java). Έτσι, οι προγραμματιστές μπορούν να χρησιμοποιήσουν διάφορες έτοιμες υλοποιήσεις ή να παρέχουν τις δικές τους. Το framework παρέχει έτοιμες υλοποιήσεις μοντέλων διεπαφών για όλα τα πρωτεύοντα στοιχεία του.

Η τυπική χρήση της δομής του Swing δεν απαιτεί τη δημιουργία εξατομικευμένων μοντέλων, μιας και η δομή παρέχει ένα σετ από έτοιμες υλοποιήσεις που συνδέονται διάφανα με την αντίστοιχη θυγατρική JComponent κλάση μέσα στη βιβλιοθήκη Swing. Γενικά, μόνο περίπλοκα στοιχεία, όπως είναι οι πίνακες, τα δέντρα και μερικές φορές και οι λίστες, μπορεί να απαιτούν εξατομικευμένες υλοποιήσεις μοντέλων για κάποια συγκεκριμένη δομή μίας εφαρμογής.

3.4 Η σχέση του Swing με το AWT [15]

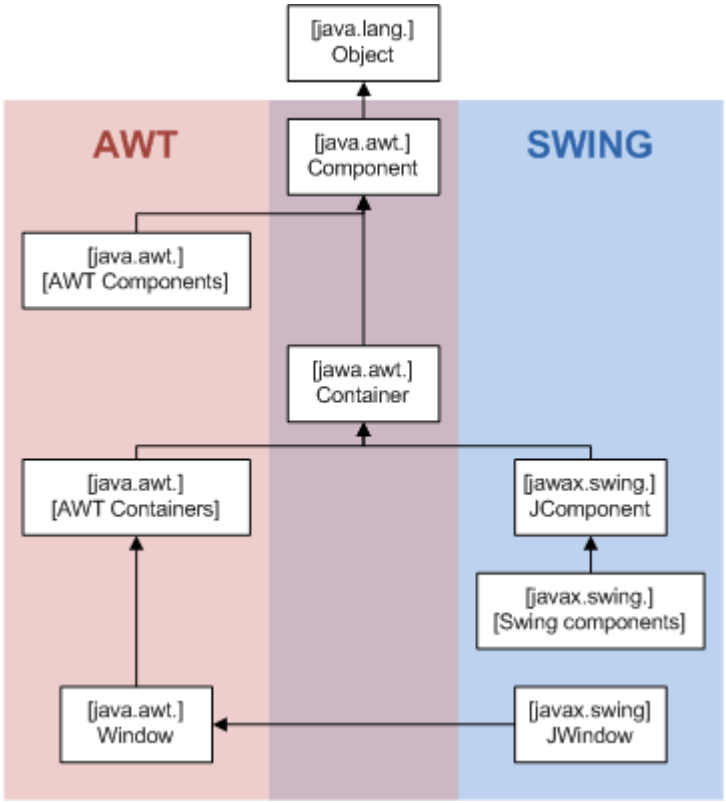
Από τις αρχικές εκδόσεις της Java™, ένα μέρος του Abstract Window Toolkit (AWT) παρείχε APIs, που ήταν ανεξάρτητα από πλατφόρμες, για τα στοιχεία διεπαφής χρήστη. Στο AWT κάθε στοιχείο απεικονίζεται και ελέγχεται από ένα τοπικό στοιχείο, που είναι συγκεκριμένο για το υποκείμενο σύστημα παραθύρων (windowing system).

Σε αντίθεση με το AWT, τα στοιχεία του Swing συχνά περιγράφονται σαν "ελαφριά" επειδή δεν απαιτούν την κατοχύρωση τοπικών πόρων στην εργαλειοθήκη παραθύρων του λειτουργικού συστήματος. Τα στοιχεία του AWT ονομάζονται "βαριά".

Μεγάλο μέρος του API του Swing είναι μία συμπληρωματική επέκταση του AWT,

παρά ένας άμεσος αντικαταστάτης. Στην πραγματικότητα, κάθε ελαφριά διεπαφή του Swing υπάρχει, τελικά, μέσα σε ένα βαρύ στοιχείο του AWT, γιατί όλα τα κορυφαία στοιχεία (top-level components) του Swing (JApplet, JDialog, JFrame και JWindow) επεκτείνουν ένα κορυφαίο AWT υποδοχέα. Παρόλα αυτά, η ταυτόχρονη χρήση ελαφριών και βαριών στοιχείων μέσα στο ίδιο παράθυρο γενικά αποφεύγεται, λόγω διάφορων ασυμβατοτήτων στη διάταξη Z (Z-order).

Η κύρια λειτουργικότητα απεικόνισης που χρησιμοποιεί το Swing για να ζωγραφίσει τα ελαφριά στοιχεία του παρέχεται από το Java2D, ένα άλλο κομμάτι του JFC.



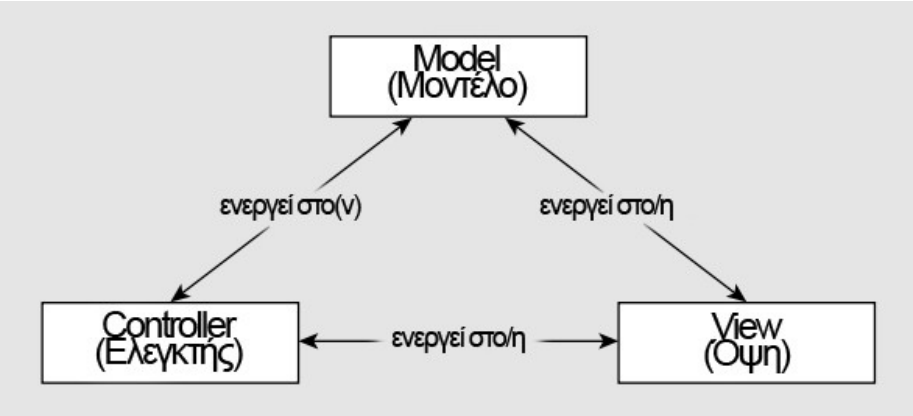
Εικόνα 15: Η ιεραρχία των κλάσεων των AWT και Swing

3.5 Αρχιτεκτονική MVC [14]

Τα γραφικά αντικείμενα που χρησιμοποιούνται στην εφαρμογή της πτυχιακής εργασίας (όπως είναι τα κουμπιά και τα μενού) πηγάζουν από τη βιβλιοθήκη Swing, η οποία βρίσκεται μέσα στη Java™. Η σχεδίαση των κλάσεων των αντικειμένων της βιβλιοθήκης Swing βασίζεται στην αρχιτεκτονική **Model – View – Controller** (**MVC**). Γνωρίζοντας πως δουλεύει η αρχιτεκτονική MVC, μπορεί κάποιος να αλλάξει τη μορφή ή τη λειτουργικότητα ενός αντικειμένου του Swing.

Η αρχιτεκτονική MVC δεν είναι καινούρια και δεν προέρχεται από τη γλώσσα προγραμματισμού Java™. Η ιδέα για την αρχιτεκτονική MVC αναδύθηκε αρκετό καιρό πριν, στα πλαίσια της γλώσσας προγραμματισμού SmallTalk. Η αρχιτεκτονική MVC είναι ένας εξιδανικευμένος τρόπος σχεδίασης ενός στοιχείου σε τρία ξεχωριστά τμήματα:

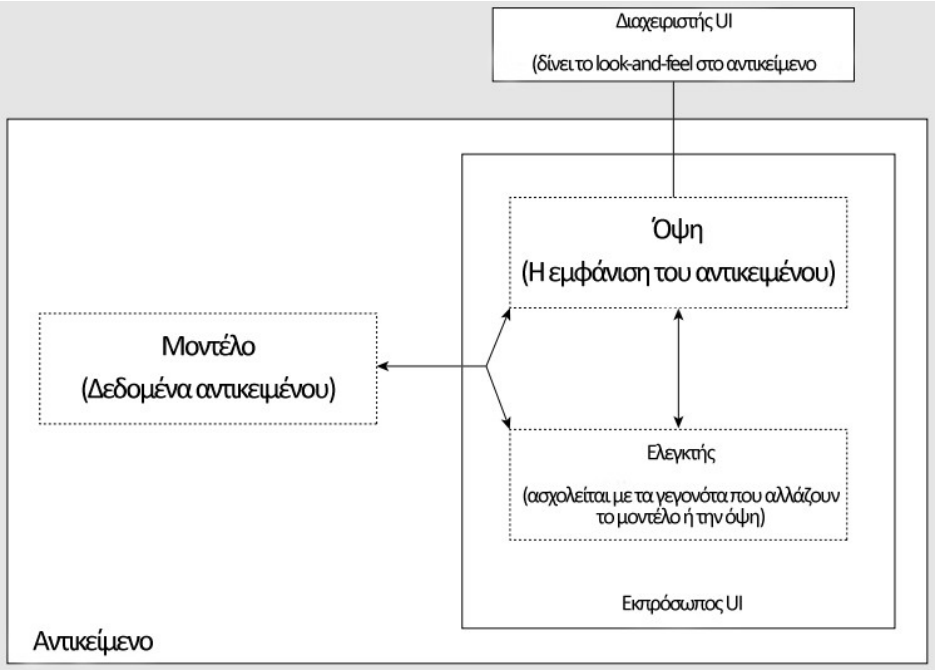
1. **Model** (Μοντέλο): Εδώ αποθηκεύονται όλα τα δεδομένα που σχετίζονται με τα αντικείμενα (τα σκίτσα στην περίπτωση της εφαρμογής της πτυχιακής).
2. **View** (Όψη): Εδώ αναπαριστώνται οπτικά τα δεδομένα των αντικεμένων (σκίτσων) από το μοντέλο.
3. **Controller** (Ελεγκτής): Εδώ έχουμε τη διαχείριση των δράσεων του χρήστη με τα διάφορα αντικείμενα της εφαρμογής. Ο controller ενημερώνει / αλλάζει τα δεδομένα του μοντέλου ή/και της όψης ανάλογα με τις δράσεις του χρήστη.



Εικόνα 16: Γενικό διάγραμμα λειτουργίας της αρχιτεκτονικής MVC

Στο πλαίσιο της αντικειμενοστρέφειας, κάθε ένα από τα τρία λογικά μέρη ενός αντικειμένου, δηλαδή το μοντέλο, η όψη και ο ελεγκτής, ιδανικά αναπαριστώνται από ένα διαφορετικό τύπο κλάσης. Πρακτικά, αυτό αποδεικνύεται δύσκολο στην υλοποίηση εξαιτίας της εξάρτησης μεταξύ της όψης και του ελεγκτή. Από τη στιγμή που ο χρήστης αλληλεπιδρά με την "φυσική" αναπαράσταση του αντικειμένου, η λειτουργία του ελεγκτή είναι εξαρτημένη σε μεγάλο βαθμό από την υλοποίηση της όψης.

Για αυτό το λόγο, η όψη και ο ελεγκτής τυπικά αναπαριστώνται από ένα μοναδικό σύνθετο αντικείμενο, το οποίο αντιστοιχεί σε μία όψη με ενσωματωμένο ελεγκτή. Σε αυτή την περίπτωση, η έννοια του MVC εκφυλίζεται στην αρχιτεκτονική Έγγραφο/Όψη (Document/View). Η εταιρία Sun ονομάζει αυτή την αρχιτεκτονική **Separable Model Architecture** (**Αρχιτεκτονική Διαχωρισμένου Μοντέλου**). Στην παρακάτω εικόνα φαίνεται η εν λόγω αρχιτεκτονική.



Εικόνα 17: Η Αρχιτεκτονική Διαχωρισμένου Μοντέλου

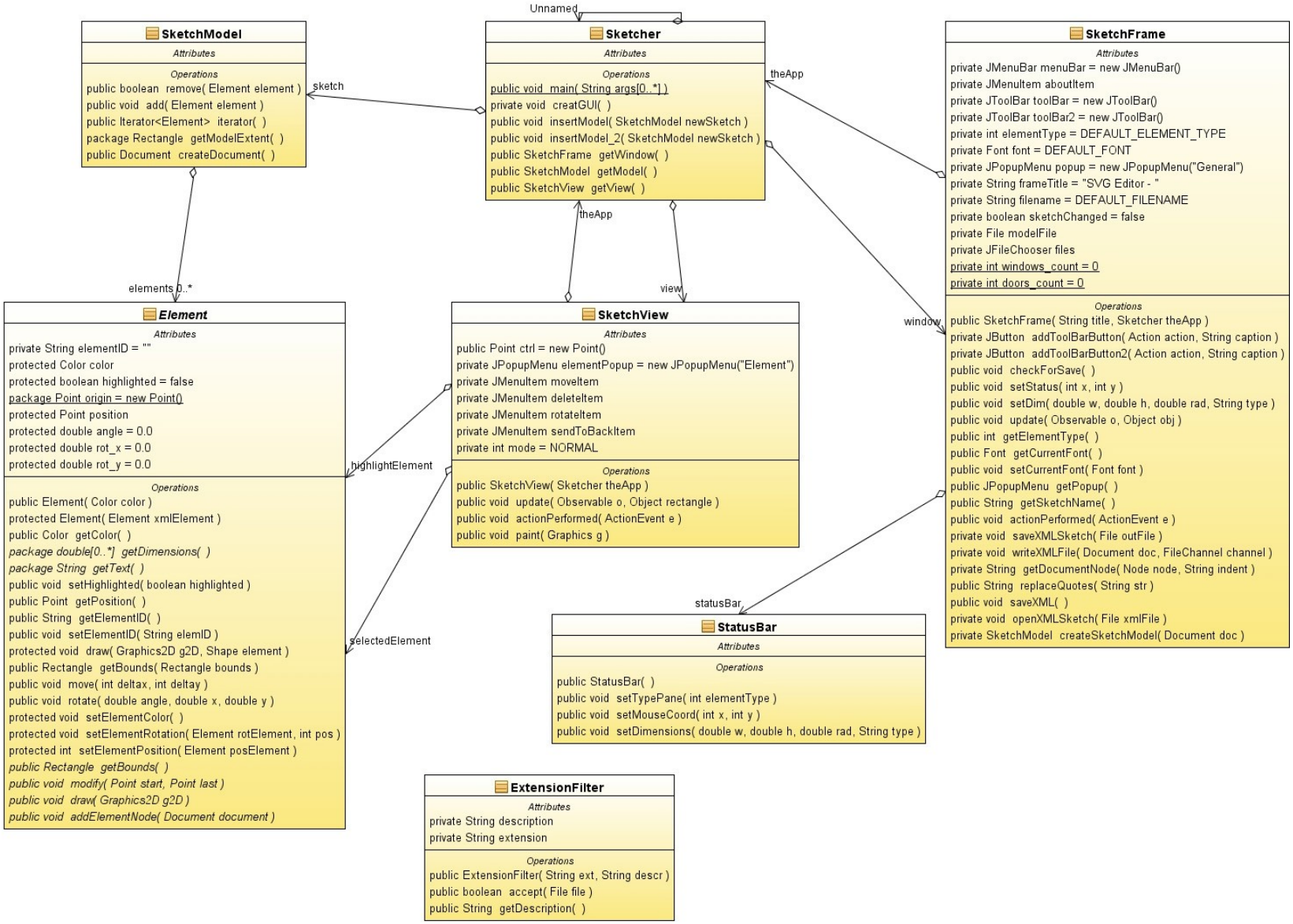
Τα αντικείμενα της βιβλιοθήκης Swing δίνουν τη δυνατότητα για αλλαγή του look-and-feel (η εμφάνιση ενός αντικειμένου) με το να γίνονται ευθύνη ενός ανεξάρτητου αντικειμένου που ονομάζεται **εκπρόσωπος UI** (**UI Delegate**) την οπτική αναπαράσταση και τη διεπαφή προς τον χρήστη. Ο εκπρόσωπος UI είναι ο συνδυασμός της όψης και του ελεγκτή της αρχιτεκτονικής MVC. Με αυτό τον τρόπο, ένας διαφορετικός εκπρόσωπος UI μπορεί να δώσει διαφορετική εμφάνιση σε ένα αντικείμενο.

3.6 Εκτέλεση εφαρμογής

Για να εκτελεστεί η εφαρμογή, ο χρήστης πρέπει να κάνει διπλό κλικ στο αρχείο svgEditor_final.jar. Τότε, καλείται η κλάση **Sketcher**. Με αυτή την κλάση δημιουργείται το παράθυρο της εφαρμογής, μέσω της κλάσης **SketchFrame**. Η SketchFrame δημιουργεί όλα τα μενού, τα κουμπιά και τις μπάρες που απαιτούνται για να λειτουργήσει σωστά η εφαρμογή. Επίσης, σε αυτή την κλάση υλοποιούνται οι μέθοδοι που αποθηκεύουν και φορτώνουν ένα σκίτσο.

3.7 Διάγραμμα κλάσεων εφαρμογής

Στην επόμενη σελίδα βρίσκεται το διάγραμμα κλάσεων της εφαρμογής. Δηλαδή, για διευκόλυνση του αναγνώστη της πτυχιακής εργασίας, έχει τοποθετηθεί ένα διάγραμμα, στο οποίο εμφανίζονται οι κλάσεις που απαρτίζουν την εφαρμογή, καθώς και οι σχέσεις μεταξύ των κλάσεων.


Unnamed
SketchModel
Attributes
Operations
public boolean remove(Element element)
public void add(Element element)
public Iterator<Element> iterator( )
package Rectangle getModelExtent( )
public Document createDocument( )
sketch
Sketcher
Attributes
Operations
public void main(String args[0..*])
private void creatGUI( )
public void insertModel(SketchModel newSketch)
public void insertModel_2(SketchModel newSketch)
public SketchFrame getWindow( )
public SketchModel getModel( )
public SketchView getView( )
theApp
SketchFrame
Attributes
private JMenuBar menuBar = new JMenuBar()
private JMenuItem aboutItem
private JToolBar toolBar = new JToolBar()
private JToolBar toolBar2 = new JToolBar()
private int elementType = DEFAULT_ELEMENT_TYPE
private Font font = DEFAULT_FONT
private JPopupMenu popup = new JPopupMenu("General")
private String frameTitle = "SVG Editor - "
private String filename = DEFAULT_FILENAME
private boolean sketchChanged = false
private File modelFile
private JFileChooser files
private int windows_count = 0
private int doors_count = 0
Operations
public SketchFrame(String title, Sketcher theApp)
private JButton addToolBarButton(Action action, String caption)
private JButton addToolBarButton2(Action action, String caption)
public void checkForSave( )
public void setStatus(int x, int y)
public void setDim(double w, double h, double rad, String type)
public void update(Observable o, Object obj)
public int getElementType( )
public Font getCurrentFont( )
public void setCurrentFont(Font font)
public JPopupMenu getPopup( )
public String getSketchName( )
public void actionPerformed(ActionEvent e)
private void saveXMLSketch(File outFile)
private void writeXMLFile(Document doc, FileChannel channel)
private String getDocumentNode(Node node, String indent)
public String replaceQuotes(String str)
public void saveXML( )
private void openXMLSketch(File xmlFile)
private SketchModel createSketchModel(Document doc)
elements 0..*
Element
Attributes
private String elementID = ""
protected Color color
protected boolean highlighted = false
package Point origin = new Point()
protected Point position
protected double angle = 0.0
protected double rot_x = 0.0
protected double rot_y = 0.0
Operations
public Element(Color color)
protected Element(Element xmlElement)
public Color getColor( )
package double[0..*] getDimensions( )
package String getText( )
public void setHighlighted(boolean highlighted)
public Point getPosition( )
public String getElementID( )
public void setElementID(String elemID)
protected void draw(Graphics2D g2D, Shape element)
public Rectangle getBounds(Rectangle bounds)
public void move(int deltax, int deltay)
public void rotate(double angle, double x, double y)
protected void setElementColor( )
protected void setElementRotation(Element rotElement, int pos)
protected int setElementPosition(Element posElement)
public Rectangle getBounds( )
public void modify(Point start, Point last)
public void draw(Graphics2D g2D)
public void addElementNode(Document document)
view
SketchView
Attributes
public Point ctrl = new Point()
private JPopupMenu elementPopup = new JPopupMenu("Element")
private JMenuItem moveItem
private JMenuItem deleteItem
private JMenuItem rotateItem
private JMenuItem sendToBackItem
private int mode = NORMAL
Operations
public SketchView(Sketcher theApp)
public void update(Observable o, Object rectangle)
public void actionPerformed(ActionEvent e)
public void paint(Graphics g)
window
highlightElement
selectedElement
statusBar
StatusBar
Attributes
Operations
public StatusBar( )
public void setTypePane(int elementType)
public void setMouseCoord(int x, int y)
public void setDimensions(double w, double h, double rad, String type)
ExtensionFilter
Attributes
private String description
private String extension
Operations
public ExtensionFilter(String ext, String descr)
public boolean accept(File file)
public String getDescription( )

3.8 Υλοποίηση σχεδιασμού πάνω στη φόρμα

Γενικά, για να σχεδιαστεί ένα σχήμα στο κεντρικό παράθυρο θα πρέπει αρχικά ο χρήστης να πατήσει το αριστερό κουμπί του ποντικιού, οπότε και αποθηκεύεται η θέση του κέρσορα στη μνήμη για να ξέρουμε το πρώτο σημείο του σχήματος. Έπειτα, σύροντας το ποντίκι ζωγραφίζει το επιθυμητό σχήμα, βλέποντας διαδοχικά ένα προσωρινό σχήμα για λόγους αναφοράς (**rubber banding**). Μόλις αφήσει το κουμπί του ποντικιού, το τελικό σχήμα εμφανίζεται.

Για να γίνει πιο κατανοητή η διαδικασία αυτή ας δώσουμε ένα παράδειγμα. Για να σχεδιαστεί μία ευθεία γραμμή στο κεντρικό παράθυρο, ο χρήστης κάνει κλικ στο σημείο απ' όπου θέλει να ξεκινά η γραμμή. Έπειτα, κρατώντας πατημένο το αριστερό κουμπί και σέρνοντας το ποντίκι σχεδιάζει τη γραμμή. Όταν το σχήμα γίνει έτσι όπως το θέλει, αφήνει το κουμπί του ποντικιού ελεύθερο, οπότε και σχεδιάζεται η τελική γραμμή.

Το **rubber-banding** που αναφέρθηκε παραπάνω είναι η διαδικασία κατά την οποία, όταν ο χρήστης σχεδιάζει ένα σχήμα (και όσο κρατάει πατημένο το κουμπί του ποντικιού) βλέπει το σχήμα να εξελίσσεται πάνω στο κεντρικό παράθυρο. Επειδή, όμως, αυτή η διαδικασία επαναλαμβάνεται πολλές φορές κατά το σύρσιμο του ποντικιού, έχουμε και μεγάλο φόρτο εργασίας. Έτσι, είναι καλύτερο να ζωγραφίζεται κάθε φορά μόνο το τμήμα που αλλάζει (δηλαδή το σχήμα που σχεδιάζει ο χρήστης).

Για να γίνει αυτό χρησιμοποιούμε την κατάσταση λειτουργίας **XOR** (**XOR Mode**). Με αυτό τον τρόπο, εάν ζωγραφίσουμε ένα σχήμα σε μία κενή περιοχή του κεντρικού παραθύρου, αυτό θα εμφανιστεί κανονικά. Εάν, τώρα, σχεδιάσουμε το ίδιο σχήμα στην ίδια θέση, τότε αυτό θα εξαφανιστεί. Την τρίτη φορά που θα σχεδιάσουμε το σχήμα, αυτό θα εμφανιστεί κανονικά και πάλι. Με λίγα λόγια, με το XOR mode μπορούμε να σβήνουμε κάθε φορά μόνο το προσωρινό σχήμα (αυτό που χρησιμοποιείται για το rubber-banding) και να το ξανά - ζωγραφίζουμε στη νέα του θέση.

Όλα αυτά, όμως, δεν θα ήταν δυνατόν να γίνουν αν δεν υπήρχε η κατάλληλη υποδομή για να ζωγραφίζουμε εύκολα και γρήγορα δισδιάστατα γραφικά με τη βοήθεια της Java™. Αυτή η υποδομή ονομάζεται **Java2D** και περιλαμβάνει όλα τα εργαλεία – αντικείμενα για τη δουλεία που θέλουμε. Στην περίπτωση μας θα γίνει χρήση του αντικειμένου **Graphics2D**, το οποίο αναφέρεται και ως **graphics context.** Με τη βοήθεια του θα ζωγραφίζουμε όλα τα σχήματα πάνω στο κεντρικό παράθυρο.

Για την υλοποίηση του σχεδιασμού πάνω στη κεντρική φόρμα της εφαρμογής χρησιμοποιήθηκε η κλάση **SketchView**. Αυτή η κλάση διαχειρίζεται οτιδήποτε εμφανίζεται οπτικά πάνω στη φόρμα. Για το λόγο αυτό, όλοι οι mouse handlers πρέπει να βρίσκονται εδώ. Τα events που χρειάζονται είναι τα **mousePressed**, **mouseDragged** και **mouseReleased**.

Ξεκινώντας την υλοποίηση, δημιουργήθηκε μία υπο-κλάση μέσα στη SketchView για τη διαχείρηση των mouse events, η **MouseHandler**. Αυτή η κλάση επεκτείνει (extends) την **MouseInputAdapter**, δηλαδή κληρονομεί όλες τις μεθόδους που αφορούν τη διαχείριση των event που δημιουργούνται από τα κουμπιά του ποντικιού, όπως επίσης και τις μεθόδους που διαχειρίζονται τα events που παράγει η κίνηση του ποντικιού.

Αφού δημιουργήσουμε την παραπάνω κλάση, ορίζουμε έναν MouseHandler μέσα στον constructor της κλάσης SketchView και τον προσθέτουμε σαν MouseListener και

MouseMotionListener. Έπειτα, προσθέτουμε στην κλάση που μόλις δημιουργήσαμε τις μεθόδους για τα events που είπαμε στην προηγούμενη παράγραφο.

3.8.1. mousePressed

Δουλειά αυτής της μεθόδου είναι να αποθηκεύσει τη θέση που έχει ο κέρσορας του ποντικιού, έτσι ώστε αυτή να χρησιμοποιηθεί ως η αρχική θέση του σχήματος που θα σχεδιάσει ο χρήστης. Αφού γίνει αυτό, ενεργοποιείται το XOR mode, έτσι ώστε να μπορέσουμε να ζωγραφίσουμε αποδοτικά το σχήμα. Τέλος, καλείται μία μέθοδος (η setStatus) της εφαρμογής, η οποία εμφανίζει στο χρήστη πληροφορίες σχετικά με το σχήμα που ζωγραφίζει (διαστάσεις, θέση κλπ).

Σημείωση: η μεταβλητή **button1Down** χρησιμοποιείται για να ξέρει η μέθοδος mouseDragged πότε το αριστερό κουμπί του ποντικιού είναι πατημένο, έτσι ώστε να αρχίσει να σχεδιάζει το σχήμα. Επίσης, η μεταβλητή **g2D** είναι ένα αντικείμενο τύπου Graphics2D, δηλαδή είναι το graphics context.

```
  public void mousePressed(MouseEvent e)  {
...
  else if((button1Down = (e.getButton() == MouseEvent.BUTTON1))) {
    start = e.getPoint();                  // Save the cursor position in start
    g2D = (Graphics2D)getGraphics();               // Get graphics context
    g2D.setXORMode(getBackground());               // Set XOR mode
   }
   theApp.getWindow().setStatus(e.getX(), e.getY());
}
```

Κώδικας της μεθόδου mousePressed

3.8.2. mouseDragged

Μέσω αυτής της μεθόδου γίνεται η σχεδίαση του προσωρινού σχήματος για τη λειτουργία **rubber-banding**. Αρχικά, αποθηκεύουμε σε μία μεταβλητή τη θέση που έχει ο κέρσορας του ποντικιού αυτή τη στιγμή. Έπειτα, εφόσον είμαστε σε κανονική λειτουργία, και όχι σε φάση μετασχηματισμού του σχήματος (π.χ. μετακίνηση), και το αριστερό κουμπί του ποντικιού είναι πατημένο ξεκινάμε τη διαδικασία rubber-banding.

Εάν δεν υπάρχει κάποιο προσωρινό σχήμα, τότε δημιουργούμε ένα (μέσω της μεθόδου **createElement**), αλλιώς το σβήνουμε από το κεντρικό παράθυρο και το αλλάζουμε (μέσω της μεθόδου **modify**) για να συνάδει με τα νέα δεδομένα. Το σβήσιμο γίνεται με το να ξανά-ζωγραφίσουμε πάνω στο ήδη σχεδιασμένο στο κεντρικό παράθυρο σχήμα.

Τέλος, αφού τελειώσουμε με τις παραπάνω διαδικασίες, ζωγραφίζουμε στο κεντρικό παράθυρο στο νέο προσωρινό σχήμα.

```
public void mouseDragged(MouseEvent e) {
    last = e.getPoint();                          // Save cursor position
    if(button1Down && (theApp.getWindow().getElementType() != TEXT)&& (mode == NORMAL)) {
      if(tempElement == null) {                   // Is there an element?
        tempElement = createElement(start, last);     // No, so create one
      } else {
        tempElement.draw(g2D);                    // Yes - draw to erase it
        tempElement.modify(start, last);           // Now modify it
      }
      tempElement.draw(g2D);                      // and draw it
    }
```

Κώδικας της μεθόδου mouseDragged

3.8.3. mouseReleased

Με τη βοήθεια της μεθόδου mouseReleased γίνεται η σχεδίαση του τελικού σχήματος. Αναλυτικότερα, εφόσον βρισκόμαστε σε κανονική λειτουργία και όχι σε φάση μετασχηματισμού του σχήματος (π.χ. μετακίνηση) και το κουμπί που αφήνει ο χρήστης είναι το αριστερό του ποντικιού τότε δίνουμε το προσωρινό σχήμα που έχει δημιουργηθεί στο μοντέλο της εφαρμογής (στην περιοχή με τα δεδομένα δηλαδή).

Επίσης, "αδειάζουμε" όποιες μεταβλητές χρησιμοποιήσαμε πιο πριν και απελευθερώνουμε το graphics context. Τέλος, εμφανίζονται στον χρήστη οι τελικές πληροφορίες για το σχήμα που μόλις σχεδίασε (μέγεθος, θέση κλπ).

```
  public void mouseReleased(MouseEvent e) {
    ...
   else if((e.getButton()==MouseEvent.BUTTON1) && (theApp.getWindow().getElementType() != TEXT) && mode == NORMAL)
{
      button1Down = false;                    // Reset the button 1 flag
      if(tempElement != null) {
     theApp.getModel().add(tempElement);       // Add element to the model
        tempElement = null;                   // No temporary element now stored
      }
    }
      ...
    if(g2D != null) {
      g2D.dispose();                          // Release graphic context resource
      g2D = null;                             // Set it to null
    }
    start = last = null;                      // Remove the points
    selectedElement = tempElement = null;      // Reset elements
    theApp.getWindow().setStatus(e.getX(), e.getY());
  }
```

Κώδικας της μεθόδου mouseReleased

3.9 Υλοποίηση σχημάτων

Για την υλοποίηση των διάφορων σχημάτων που είναι διαθέσιμα στο χρήστη, δημιουργήθηκε μία κλάση (η **Element**) η οποία περιέχει όλες τις απαραίτητες μεθόδους για τη δημιουργία και διαχείριση των σχημάτων. Επίσης, ορίζει τις απαραίτητες πληροφορίες για την αποθήκευση των σχημάτων κατά το πρότυπο SVG.

Η Element είναι abstract κλάση, δηλαδή παρέχει κάποιες μεθόδους στις κλάσεις οι οποίες την κάνουν implement. Έτσι, μαζί με την κλάση Element, έχουν δηλωθεί και οι κλασεις για τα σχήματα, οι οποίες κάνουν implement την κλάση Element. Όλες οι κλάσεις κληρονομούν τις εξής μεθόδους από την Element:

- **getDimensions**: Abstract μέθοδος χωρίς κώδικα, η οποία (ανάλογα με την κλάση σχήματος που τη χρησιμοποιεί) παρέχει τις απαραίτητες πληροφορίες για τις διαστάσεις του σχήματος. Ο κώδικας της υλοποιείται μέσα σε κάθε κλάση που τη χρησιμοποιεί και εξηγείται παρακάτω. Η μέθοδος έχει τιμή επιστροφής ένα array από double αριθμούς.

- **getText**: Abstract μέθοδος χωρίς κώδικα, η οποία επιστρέφει ένα String με το κείμενο που έδωσε ο χρήστης (σχήμα κειμένου). Ο κώδικας της μεθόδου υλοποιείται μέσα σε κάθε κλάση που τη χρησιμοποιεί. Επειδή αυτή η μέθοδος είναι χρήσιμη μόνο για την κλάση του κειμένου, όλες οι υπόλοιπες κλάσεις επιστρέφουν τη τιμή **null**.

- **setHighlighted**: Με αυτή την μέθοδο δηλώνουμε πότε θέλουμε να αλλάξει το χρώμα του σχήματος, επειδή ο δείκτης του ποντικιού βρίσκεται πάνω από το σχήμα. Παίρνει σαν όρισμα μία **boolean** τιμή, ανάλογα με το αν θέλουμε να αλλάξει το χρώμα. Η μέθοδος δεν έχει τιμή επιστροφής.

```
public void setHighlighted(boolean highlighted) {
  this.highlighted = highlighted;
}
```

Κώδικας της μεθόδου setHighlighted

- **getPosition**: Αυτή η μέθοδος δίνει τη θέση που βρίσκεται το επιλεγμένο σχήμα πάνω στη περιοχή σχεδίασης. Έχει τιμή επιστροφής ένα **Point** με τις τιμές **x, y** της θέσης.

```
public Point getPosition() {
  return position;
}
```

Κώδικας της μεθόδου getPosition

- **draw**: Αυτή η μέθοδος ζωγραφίζει ένα σχήμα στη περιοχή σχεδιάσης. Δέχεται σαν ορίσματα ένα **Graphics2D** context (το οποίο μας δίνει τη δυνατότητα να ζωγραφίζουμε στη περιοχή σχεδίασης) και ένα **Shape** (το σχήμα που θέλουμε να ζωγραφιστεί).

  Στην αρχή θέτουμε το χρώμα του σχήματος (είτε magenta είτε μαύρο, ανάλογα με το αν είναι επιλεγμένο ή όχι). Έπειτα αποθηκεύουμε τον τρέχοντα μετασχηματισμό. Μετά μετακινούμε το σχήμα στη θέση του και το περιστρέφουμε. Τέλος, ζωγραφίζουμε το σχήμα και επαναφέρουμε τον μετασχηματισμό.

```
protected void draw(Graphics2D g2D, Shape element) {
  g2D.setPaint(highlighted ? Color.MAGENTA : color);
  AffineTransform old = g2D.getTransform()
  g2D.translate(position.x, position.y);
  g2D.rotate(angle);
  g2D.draw(element);
  g2D.setTransform(old);
}
```

Κώδικας της μεθόδου draw

- **getBounds**: Η μέθοδος αυτή επιστρέφει ένα ορθογώνιο (**java.awt.Rectangle**), το οποίο περιβάλλει το σχήμα. Αυτό είναι τα όρια του σχήματος. Σαν όρισμα, η μέθοδος, δέχεται ένα java.awt.Rectangle.

  Στην αρχή δημιουργείται ένας μετασχηματισμός, ο οποίος εφαρμόζεται μετά για να επιστραφεί το κατάλληλο ορθογώνιο.

```
public java.awt.Rectangle getBounds(java.awt.Rectangle bounds) {
  AffineTransform at = AffineTransform.getTranslateInstance(position.x, position.y);
  at.rotate(angle);
  return at.createTransformedShape(bounds).getBounds();
}
```

Κώδικας της μεθόδου getBounds

- **move**: Αυτή η μέθοδος χρησιμοποιείται για να μετακινήσουμε ένα σχήμα πάνω στην περιοχή σχεδίασης. Δέχεται σαν ορίσματα δύο ακέραιες τιμές, τις **deltax** και **deltay**, οι οποίες υποδηλώνουν τη μετατόπιση του σχήματος κατά deltax pixels στον οριζόντιο άξονα και κατά deltay pixels στον κάθετο άξονα. Η μέθοδος δεν έχει τιμή επιστροφής.

```
public void move(int deltax, int deltay) {
  position.x += deltax;
  position.y += deltay;
}
```

Κώδικας της μεθόδου move

- **rotate**: Αυτή η μέθοδος χρησιμοποιείται για να περιστρέψουμε ένα σχήμα. Δέχεται σαν όρισμα μία double τιμή (**angle**), η οποία αντιπροσωπεύει τη περιστροφή κατά angle. Η μέθοδος δεν έχει τιμή επιστροφής.

```
public void rotate(double angle) {
  this.angle += angle;
}
```

Κώδικας της μεθόδου rotate

- **setElementRotation**: Αυτή η κλάση χρησιμοποιείται για να περιστραφεί το αντικείμενο, όταν αυτό φορτώνεται από ένα αποθηκευμένο αρχείο SVG. Δέχεται σαν ορίσματα ένα **org.w3c.dom.Element rotElement**, το οποίο μας δίνει πρόσβαση στο αρχείο SVG, και ένα **integer αριθμό**, ο οποίος μας δείχνει από πιο χαρακτήρα του string που θα διαβαστεί από το SVG και μετά υπάρχει η πληροφορία για την περιστροφή του σχήματος.Η μέθοδος δεν έχει τιμή επιστροφής.

  Αφού διαβαστεί το string του σχήματος από το αρχείο SVG και βρεθεί η θέση που βρίσκεται η πληροφορία για την περιστροφή, διαβάζουμε διαδοχικά κάθε χαρακτήρα του string έτσι ώστε να ανακτήσουμε αυτή την πληροφορία. Αφού διαβαστεί, τη μεταφέρουμε στη μεταβλητή **angle**.

```
protected  void setElementRotation(org.w3c.dom.Element rotElement, int pos)
{
   angle = 0;
   String tmpStr;
   int flag = 0, start = 0;
   tmpStr = String.valueOf(rotElement.getAttribute("transform"));
   tmpStr = tmpStr.substring(pos);
   pos = 0;

   while(flag == 0)
   {
      if(tmpStr.charAt(pos) != ')')
      {
         pos++;
      }
      else
      {
         flag = 1;
      }
   }

   String tmpStr2 = "";
   tmpStr2 = tmpStr.substring(start, pos);
   angle = Double.parseDouble(tmpStr2);
   angle = (angle * 1.7388) / 100;
}
```

Κώδικας της μεθόδου setElementRotation

- **setElementPosition**: Αυτή η μέθοδος χρησιμοποιείται για να μετακινηθεί το σχήμα στην κατάλληλη θέση, όταν αυτό φορτώνεται από ένα αποθηκευμένο αρχείο SVG. Δέχεται σαν όρισμα ένα **org.w3c.dom.Element** και έχει σαν τιμή επιστροφής τη θέση μέσα στο string που διαβάστηκε από το αρχείο SVG, έτσι ώστε να χρησιμοποιηθεί από τη μέθοδο setElementRotation για να διαβάσει την πληροφορία για την περιστροφή.

  Αφού διαβαστεί το string του σχήματος από το SVG, βρίσκουμε διαδοχικά τη μετατόπιση στον οριζόντιο και τον κατακόρυφο άξονα. Έπειτα, ορίζουμε τη μεταβλητή **position** έτσι ώστε να δείχνει τη θέση στην οποία θέλουμε να μετακινηθεί το σχήμα. Τέλος, επιστρέφουμε την κατάλληλη θέση για την περιστροφή.

```
protected int setElementPosition(org.w3c.dom.Element posElement) {
   position = new Point();
   String tmpStr, x = "", y = "";
   int flag = 0, pos = 0, start = 0;
   tmpStr = String.valueOf(posElement.getAttribute("transform"));
   tmpStr = tmpStr.substring(10);
   while(flag == 0)
   {
      if(tmpStr.charAt(pos) != ',')
      {
        pos++;
      }
      else
      {
         flag = 1;
      }
   }
   x = tmpStr.substring(start, pos);
   pos++;
   start = pos;
   flag = 0;
   while(flag == 0)
   {
      if(tmpStr.charAt(pos) != ')')
      {
        pos++;
      }
      else
      {
         flag = 1;
      }
   }
   y = tmpStr.substring(start, pos);
   position.setLocation(Double.parseDouble(x), Double.parseDouble(y));
   return (pos + 19);
}
```

Κώδικας της μεθόδου setElementPosition

- **getBounds**: Abstract μέθοδος, χωρίς ορίσματα, που χρησιμοποιείται από την κάθε κλάση σχήματος για να δίνει στην **Element.getBounds** το java.awt.Rectangle όρισμα.

- **modify**: Abstract μέθοδος, χωρίς ορίσματα και τιμή επιστροφής, που χρησιμοποιείται από τις κλάσεις σχημάτων για να τροποποιούν τα χαρακτηριστικά των σχημάτων κατά τη σχεδίαση.

- **draw**: Abstract μέθοδος που χρησιμοποιείται για από τις κλάσεις σχημάτων για να καλέσουν την **Element.draw** για να ζωγραφίζουν τα σχήματα. Έχει σαν όρισμα ένα **Graphics2D** context και δεν έχει τιμή επιστροφής.

- **AddElementNode**: Abstract μέθοδος που χρησιμοποιείται από τις κλάσεις σχημάτων για να μετατρέπουν τα σχήματα τους σε SVG. Δέχεται σαν όρισμα ένα **Document** που αντιστοιχεί στο SVG αρχείο και δεν έχει τιμή επιστροφής.

3.9.1 Γραμμή

Για το σχήμα της γραμμής έχει δημιουργηθεί η κλάση **Line**, η οποία κάνει extend την κλάση Element. Μία γραμμή είναι τύπου **Line2D.Double**. Μέσα στη Line υπάρχουν δύο constructors.

- Ο πρώτος δέχεται σαν ορίσματα **δύο** τιμές τύπου **Point** (τα σημεία που ξεκινά και τελειώνει αντίστοιχα η γραμμή) και μία τιμή τύπου **Color** (σε περίπτωση που θέλουμε να έχουμε διαφορετικά χρώματα για κάθε σχήμα). Το χρώμα που χρησιμοποιείται στην εφαρμογή είναι αποκλειστικά το μαύρο.

  Αυτός ο constructor χρησιμοποιείται κάθε φορά που ο χρήστης σχεδιάζει μία νέα γραμμή στην περιοχή σχεδίασης.

```
public Line(Point start, Point end, Color color) {
  super(color);
  position = start;

  line = new Line2D.Double(origin, new Point(end.x - position.x, end.y - position.y));
}
```

Κώδικας του constructor της κλάσης Line

- Ο δεύτερος constructor χρησιμοποιείται για να δημιουργηθεί ένα αντικείμενο γραμμής όταν φορτώνεται από ένα αρχείο SVG. Δέχεται σαν όρισμα μία τιμή **org.w3c.dom.Element** η οποία αναφέρεται στο αρχείο SVG. Από το αρχείο, διαβάζονται οι τιμές του τελικού σημείου της γραμμής (το πρώτο σημείο δεν έχει νόημα να το διαβάσουμε, γιατί όλα τα σχήματα σχεδιάζονται πρώτα στο σημείο 0,0

και μετά μεταφέρονται στη σωστή θέση), και δημιουργείται η γραμμή.

```
public Line(org.w3c.dom.Element xmlElement) {
  super(xmlElement);
  org.w3c.dom.Node node = xmlElement.getFirstChild();
  String x2, y2;
  x2 = ((org.w3c.dom.Element)node).getAttribute("x2");
  y2 = ((org.w3c.dom.Element)node).getAttribute("y2");
  line=new Line2D.Double(origin.x, origin.y, Double.parseDouble(x2),Double.parseDouble(y2));
 }
```

Κώδικας του constructor της κλάσης Line

Μετά από τους constructors, υπάρχουν οι μέθοδοι της κλάσης που διαχειρίζονται διάφορα θέματα. Αναλυτικά:

- **draw**: Μέθοδος η οποία καλεί τη μέθοδο **Element.draw** για να ζωγραφιστεί το σχήμα της γραμμής. Δέχεται σαν όρισμα ένα **Graphics2D** context και δεν έχει τιμή επιστροφής.

```
  public void draw(Graphics2D g2D) {
    draw(g2D, line);
  }
```

Κώδικας της μεθόδου draw της κλάσης Line

- **getBounds**: Μέθοδος που χρησιμοποιείται για να πέρνουμε τα όρια μέσα στα οποία είναι σχεδιασμένη η γραμμή. Δεν έχει ορίσματα και επιστρέφει ένα **java.awt.Rectangle** που περιβάλλει τη γραμμή. Το java.awt.Rectangle το παίρνει καλώντας την Element.getBounds.

```
  public java.awt.Rectangle getBounds() {
    return getBounds(line.getBounds());
  }
```

Κώδικας της μεθόδου getBounds της κλάσης Line

- **modify**: Με τη μέθοδο αυτή, ενημερώνεται το σχήμα της γραμμής για τις αλλαγές που πρέπει να γίνουν όσο ο χρήστης σχεδιάζει τη γραμμή με το ποντίκι. Δέχεται σαν ορίσματα δύο **Point** τιμές, μία που δείχνει το αρχικό σημείο της γραμμής (δεν χρησιμοποιείται) και μία που δείχνει το τελικό σημείο (αυτό που βρίσκεται εκείνη τη στιγμή κάτω από το δείκτη του ποντικιού). Η μέθοδος δεν έχει τιμή επιστροφής.

  Ορίζουμε πως το τελικό σημείο της γραμμής θα είναι το σημείο που δείχνει το ποντίκι (αν και το μετατοπίζουμε κατά **-position** διότι όλα τα σχήματα σχεδιάζονται αρχικά στο σημείο 0.0 και μετά μετατοπίζονται στη σωστή θέση).

  Μετά, επειδή το ορθογώνιο που περιβάλλει τη γραμμή, όταν αυτή είναι κάθετη ή οριζόντια, δεν έχει διαστάσεις, με αποτέλεσμα να μην μπορεί ο χρήστη να επιλέξει τη γραμμή με το ποντίκι, μεριμνούμε ώστε όταν βρίσκεται η γραμμή σε μία από τις δύο καταστάσεις να μετατοπίζεται το τελικό σημείο ελάχιστα.

```
public void modify(Point start, Point last) {
  line.x2 = last.x - position.x;
  line.y2 = last.y - position.y;
  if(line.y2 == line.y1)
  {
     line.y2 = line.y2 + 1;
  }
  if(line.x2 == line.x1)
  {
     line.x2 = line.x2 + 1;
  }
}
```

Κώδικας της μεθόδου modify της κλάσης Line

- **addElementNode**: Με αυτή την κλάση αποθηκεύουμε το σχήμα της γραμμής με βάση το πρότυπο SVG. Δέχεται σαν όρισμα μία τιμή τύπου **Document** η οποία δείχνει στο αρχείο SVG στο οποίο θα αποθηκευτεί το σκίτσο. Η μέθοδος δεν έχει τιμή επιστροφής.

  Αρχικά, δημιουργούμε ένα tag **<g></g>** με το οποίο ομαδοποιούμε τα δεδομένα που ακολουθούν, έτσι ώστε να μπορούμε να τα διαχειριστούμε εύκολα.

  Έπειτα, δημιουργούμε την ιδιότητα **transform** του <g> και της δίνουμε τις κατάλληλες τιμές (**translate** και **rotate**) έτσι ώστε να μετακινείται και να περιστρέφεται το σχήμα όπως πρέπει.

  Μετά, κατασκευάζουμε το tag **<line/>** που συμβολίζει τη γραμμή στο SVG πρότυπο, και του προσθέτουμε τις ιδιότητες που απαιτούνται (**x1, y1, x2, y2, stroke-width, stroke**).

  Τέλος, προσθέτουμε το <line> που φτιάξαμε στο <g> και τοποθετούμε το <g> μέσα στο αρχείο.

```
public void addElementNode(Document doc) {
  org.w3c.dom.Element gElement = doc.createElement("g");
  Attr rotAttr = doc.createAttribute("transform");
  rotAttr.setValue("translate(" + position.getX() + "," +
                position.getY() + ") rotate(" +
                ((angle*100)/(1.7388)) + ")");
  gElement.setAttributeNode(rotAttr);
  org.w3c.dom.Element lineElement = doc.createElement("line");
  Attr attr1 = doc.createAttribute("x1");
  attr1.setValue("0");
  lineElement.setAttributeNode(attr1);
  Attr attr2 = doc.createAttribute("y1");
  attr2.setValue("0");
  lineElement.setAttributeNode(attr2);
  Attr attr3 = doc.createAttribute("x2");
  attr3.setValue(String.valueOf(line.x2 /*+ position.x*/));
  lineElement.setAttributeNode(attr3);
  Attr attr4 = doc.createAttribute("y2");
```

```
    attr4.setValue(String.valueOf(line.y2 /*+ position.y*/));
    lineElement.setAttributeNode(attr4);
    Attr attr5 = doc.createAttribute("stroke-width");
    attr5.setValue("1");
    lineElement.setAttributeNode(attr5);
    Attr attr6 = doc.createAttribute("stroke");
    attr6.setValue("black");
    lineElement.setAttributeNode(attr6);
    gElement.appendChild(lineElement);
    doc.getDocumentElement().appendChild(gElement);
  }
```

Κώδικας της μεθόδου addElementNode της κλάσης Line

- **toString**: Μέθοδος που χρησιμοποιείται για να παίρνουμε τι είδους σχήμα είναι αυτό. Δεν έχει ορίσματα και επιστρέφει μία τιμή **String**.

```
  @Override
  public String toString()
  {
     return "line";
  }
```

Κώδικας της μεθόδου toString της κλάσης Line

- **getDimensions**: Αυτή η μέθοδος μας δίνει το μήκος της γραμμής που έχει σχεδιάσει ο χρήστης. Δεν έχει ορίσματα και επιστρέφει ένα **array τύπου double**.

  Αφού δηλώσουμε τις απαραίτητες μεταβλητές, υπολογίζουμε το μήκος της γραμμής με βάση τον τύπο $\sqrt{(x2-x1)^2+(y2-y1)^2}$, όπου (x1, y1) το αρχικό σημείο της γραμμής και (x2, y2) το τελικό σημείο. Μετά, βάζουμε την τιμή αυτή στην πρώτη θέση του array, μηδενίζουμε τις υπόλοιπες θέσεις και επιστρέφουμε το array.

```
  @Override
  public double[] getDimensions()
  {
     double ar[] = new double[3];

     double len;
     len = Math.sqrt(Math.pow(line.x2 - line.x1, 2) +
                Math.pow(line.y2 - line.y1, 2));
     ar[0] = len;
     ar[1] = 0;
     ar[2] = 0;
     return ar;
  }
```

Κώδικας της μεθόδου getDimensions της κλάσης Line

- **getText**: Αυτή η μέθοδος **δεν** χρησιμεύει στην κλάση Line, οπότε επιστρέφουμε **null**.

```
public String getText()
{
   return "null";
}
```

Κώδικας της μεθόδου getText της κλάσης Line

3.9.2 Ορθογώνιο

Για το σχήμα του ορθογώνιου έχει δημιουργηθεί η κλάση **Rectangle**, η οποία κάνει extend την κλάση Element. Ένα ορθογώνιο είναι τύπου **Rectangle2D.Double**. Μέσα στη Rectangle υπάρχουν δύο constructors.

- Ο πρώτος δέχεται σαν ορίσματα **δύο** τιμές τύπου **Point** (το πάνω αριστερό σημείο και το κάτω δεξί σημείο του ορθογωνίου) και μία τιμή τύπου **Color** (σε περίπτωση που θέλουμε να έχουμε διαφορετικά χρώματα για κάθε σχήμα). Το χρώμα που χρησιμοποιείται στην εφαρμογή είναι αποκλειστικά το μαύρο.

  Αυτός ο constructor χρησιμοποιείται κάθε φορά που ο χρήστης σχεδιάζει ένα νέο ορθογώνιο στην περιοχή σχεδίασης.

```
public Rectangle(Point start, Point end, Color color) {
  super(color);
position = new Point(Math.min(start.x, end.x),
                Math.min(start.y, end.y));
rectangle = new Rectangle2D.Double(origin.x,
                            origin.y,
                            Math.abs(start.x - end.x),
                            Math.abs(start.y - end.y));
}
```

Κώδικας του constructor της κλάσης Rectangle

- Ο δεύτερος constructor χρησιμοποιείται για να δημιουργηθεί ένα αντικείμενο ορθογωνίου όταν φορτώνεται από ένα αρχείο SVG. Δέχεται σαν όρισμα μία τιμή **org.w3c.dom.Element** η οποία αναφέρεται στο αρχείο SVG. Από το αρχείο, διαβάζονται οι τιμές του ύψους και του πλάτους του ορθογωνίου, και δημιουργείται το ορθογώνιο.

```
public Rectangle(org.w3c.dom.Element xmlElement) {
  super(xmlElement);

org.w3c.dom.Node node = xmlElement.getFirstChild();

String width, height;
width = ((org.w3c.dom.Element)node).getAttribute("width");
height = ((org.w3c.dom.Element)node).getAttribute("height");

  rectangle = new Rectangle2D.Double(origin.x, origin.y,
                            Double.parseDouble(width),
                            Double.parseDouble(height));
```

```
}
```

Κώδικας του constructor της κλάσης Rectangle

Μετά από τους constructors, υπάρχουν οι μέθοδοι της κλάσης που διαχειρίζονται διάφορα θέματα. Αναλυτικά:

- **draw**: Μέθοδος η οποία καλεί τη μέθοδο **Element.draw** για να ζωγραφιστεί το σχήμα του ορθογωνίου. Δέχεται σαν όρισμα ένα **Graphics2D** context και δεν έχει τιμή επιστροφής.

```
public void draw(Graphics2D g2D) {
  draw(g2D, rectangle);
}
```

Κώδικας της μεθόδου draw της κλάσης Rectangle

- **getBounds**: Μέθοδος που χρησιμοποιείται για να πέρνουμε τα όρια μέσα στα οποία είναι σχεδιασμένο το ορθογώνιο. Δεν έχει ορίσματα και επιστρέφει ένα **java.awt.Rectangle** που περιβάλλει το ορθογώνιο. Το java.awt.Rectangle το παίρνει καλώντας την Element.getBounds.

```
public java.awt.Rectangle getBounds() {
  return getBounds(rectangle.getBounds());
}
```

Κώδικας της μεθόδου getBounds της κλάσης Rectangle

- **modify**: Με τη μέθοδο αυτή, ενημερώνεται το σχήμα του ορθογωνίου για τις αλλαγές που πρέπει να γίνουν όσο ο χρήστης σχεδιάζει το ορθογώνιο με το ποντίκι. Δέχεται σαν ορίσματα δύο **Point** τιμές, μία που δείχνει το αρχικό σημείο του ορθογωνίου και μία που δείχνει το τελικό σημείο (αυτό που βρίσκεται εκείνη τη στιγμή κάτω από το δείκτη του ποντικιού). Η μέθοδος δεν έχει τιμή επιστροφής.

  Σαν θέση του ορθογωνίου στον οριζόντιο άξονα παίρνουμε τη **μικρότερη τιμή** από τις **start.x** και **last.x**. Το ίδιο κάνουμε και για τον κατακόρυφο άξονα.

  Σαν πλάτος του ορθογωνίου ορίζουμε τη απόλυτη τιμή της αφαίρεσης του **last.x** από το **start.x**. Σαν ύψος του ορθογωνίου ορίζουμε την απόλυτη τιμή της αφαίρεσης του **last.y** από το **start.y**. Παίρνουμε τις απόλυτες τιμές γιατί η πράξη μπορεί να μας δώσει αρνητική τιμή, πράγμα που δεν γίνεται, αφού μιλάμε για πλάτος και ύψος.

```
public void modify(Point start, Point last) {
  position.x = Math.min(start.x, last.x);
  position.y = Math.min(start.y, last.y);
  rectangle.width = Math.abs(start.x - last.x);
  rectangle.height = Math.abs(start.y - last.y);
}
```

Κώδικας της μεθόδου modify της κλάσης Rectangle

- **addElementNode**: Με αυτή την κλάση αποθηκεύουμε το σχήμα του ορθογωνίου με βάση το πρότυπο SVG. Δέχεται σαν όρισμα μία τιμή τύπου **Document** η οποία δείχνει στο αρχείο SVG στο οποίο θα αποθηκευτεί το σκίτσο. Η μέθοδος δεν έχει τιμή επιστροφής.

  Αρχικά, δημιουργούμε ένα tag **<g></g>** με το οποίο ομαδοποιούμε τα δεδομένα που ακολουθούν, έτσι ώστε να μπορούμε να τα διαχειριστούμε εύκολα.

  Έπειτα, δημιουργούμε την ιδιότητα **transform** του <g> και της δίνουμε τις κατάλληλες τιμές (**translate** και **rotate**) έτσι ώστε να μετακινείται και να περιστρέφεται το σχήμα όπως πρέπει.

  Μετά, κατασκευάζουμε το tag **<rect/>** που συμβολίζει το ορθογώνιο στο SVG πρότυπο, και του προσθέτουμε τις ιδιότητες που απαιτούνται (**x, y, width, height, stroke-width, stroke, fill**).

  Τέλος, προσθέτουμε το <rect> που φτιάξαμε στο <g> και τοποθετούμε το <g> μέσα στο αρχείο.

```
public void addElementNode(Document doc) {
  //Create rotation attribute for rect
  org.w3c.dom.Element gElement = doc.createElement("g");
  Attr rotAttr = doc.createAttribute("transform");
  rotAttr.setValue("translate(" + position.getX() + "," +
                position.getY() + ") rotate(" +
                ((angle*100)/(1.7388)) + ")");
  gElement.setAttributeNode(rotAttr);
  org.w3c.dom.Element rectElement = doc.createElement("rect");
  Attr attr1 = doc.createAttribute("x");
  attr1.setValue("0");
  rectElement.setAttributeNode(attr1);
  Attr attr2 = doc.createAttribute("y");
  attr2.setValue("0");
  rectElement.setAttributeNode(attr2);
  Attr attr3 = doc.createAttribute("width");
  attr3.setValue(String.valueOf(rectangle.width));
  rectElement.setAttributeNode(attr3);
  Attr attr4 = doc.createAttribute("height");
  attr4.setValue(String.valueOf(rectangle.height));
  rectElement.setAttributeNode(attr4);
  Attr attr5 = doc.createAttribute("stroke");
  attr5.setValue("black");
  rectElement.setAttributeNode(attr5);
  Attr attr6 = doc.createAttribute("stroke-width");
  attr6.setValue("1");
  rectElement.setAttributeNode(attr6);
  Attr attr7 = doc.createAttribute("fill");
  attr7.setValue("none");
  rectElement.setAttributeNode(attr7);
  gElement.appendChild(rectElement);
  doc.getDocumentElement().appendChild(gElement);
}
```

Κώδικας της μεθόδου addElementNode της κλάσης Rectangle

- **toString**: Μέθοδος που χρησιμοποιείται για να παίρνουμε τι είδους σχήμα είναι αυτό. Δεν έχει ορίσματα και επιστρέφει μία τιμή **String**.

```
@Override
public String toString()
{
   return "rectangle";
}
```

Κώδικας της μεθόδου toString της κλάσης Rectangle

- **getDimensions**: Αυτή η μέθοδος μας δίνει το πλάτος και το ύψος του ορθογωνίου που έχει σχεδιάσει ο χρήστης. Δεν έχει ορίσματα και επιστρέφει ένα **array τύπου double**.

  Αφού δηλώσουμε τις απαραίτητες μεταβλητές, τοποθετούμε το πλάτος του ορθογωνίου στην πρώτη θέση του array και το ύψος στη δεύτερη θέση (τα παίρνουμε με τις μεθόδους **get.Width** και **getHeight**. Τέλος, επιστρέφουμε το array.

```
@Override
public double[] getDimensions()
{
   double[] ar = new double[3];
   ar[0] = rectangle.getWidth();
   ar[1] = rectangle.getHeight();
   ar[2] = 0;
   return ar;
}
```

Κώδικας της μεθόδου getDimensions της κλάσης Rectangle

- **getText**: Αυτή η μέθοδος **δεν** χρησιμεύει στην κλάση Rectangle, οπότε επιστρέφουμε **null**.

```
public String getText()
{
   return "null";
}
```

Κώδικας της μεθόδου getText της κλάσης Rectangle

3.9.3 Κύκλος

Για το σχήμα του κύκλου έχει δημιουργηθεί η κλάση **Circle**, η οποία κάνει extend την κλάση Element. Ένας κύκλος είναι τύπου **Ellipse2D.Double**. Μέσα στη Circle υπάρχουν δύο constructors.

- Ο πρώτος δέχεται σαν ορίσματα **δύο** τιμές τύπου **Point** (το κέντρο του κύκλου και ένα σημείο της περιφέρειας του) και μία τιμή τύπου **Color** (σε περίπτωση που θέλουμε να έχουμε διαφορετικά χρώματα για κάθε σχήμα). Το χρώμα που χρησιμοποιείται στην εφαρμογή είναι αποκλειστικά το μαύρο.

Αυτός ο constructor χρησιμοποιείται κάθε φορά που ο χρήστης σχεδιάζει ένα νέο κύκλο στην περιοχή σχεδίασης.

```
public Circle(Point center, Point circum, Color color) {
  super(color);
  double radius = center.distance(circum);
  position = new Point(center.x - (int)radius,
                center.y - (int)radius);
  circle = new Ellipse2D.Double(origin.x, origin.y,
                        2.*radius, 2.*radius );
}
```

Κώδικας του constructor της κλάσης Circle

- Ο δεύτερος constructor χρησιμοποιείται για να δημιουργηθεί ένα αντικείμενο κύκλου όταν φορτώνεται από ένα αρχείο SVG. Δέχεται σαν όρισμα μία τιμή **org.w3c.dom.Element** η οποία αναφέρεται στο αρχείο SVG. Από το αρχείο, διαβάζονται η τιμή της ακτίνας του κύκλου, και δημιουργείται ο κύκλος.

```
public Circle(org.w3c.dom.Element xmlElement) {
  super(xmlElement);
org.w3c.dom.Node node = xmlElement.getFirstChild();
String cr;
cr = ((org.w3c.dom.Element)node).getAttribute("r");
  double radius = Double.parseDouble(cr);
  circle = new Ellipse2D.Double(origin.x - radius,
            origin.y – radius, 2.*radius, 2.*radius);
}
```

Κώδικας του constructor της κλάσης Circle

Μετά από τους constructors, υπάρχουν οι μέθοδοι της κλάσης που διαχειρίζονται διάφορα θέματα. Αναλυτικά:

- **draw**: Μέθοδος η οποία καλεί τη μέθοδο **Element.draw** για να ζωγραφιστεί το σχήμα του κύκλου. Δέχεται σαν όρισμα ένα **Graphics2D** context και δεν έχει τιμή επιστροφής.

```
public void draw(Graphics2D g2D) {
  draw(g2D, circle);
}
```

Κώδικας της μεθόδου draw της κλάσης Circle

- **getBounds**: Μέθοδος που χρησιμοποιείται για να πέρνουμε τα όρια μέσα στα οποία είναι σχεδιασμένος ο κύκλος. Δεν έχει ορίσματα και επιστρέφει ένα **java.awt.Rectangle** που περιβάλλει τον κύκλο. Το java.awt.Rectangle το παίρνει καλώντας την Element.getBounds.

```
public java.awt.Rectangle getBounds() {
  return getBounds(circle.getBounds());
}
```

Κώδικας της μεθόδου getBounds της κλάσης Circle

- **modify**: Με τη μέθοδο αυτή, ενημερώνεται το σχήμα του κύκλου για τις αλλαγές που πρέπει να γίνουν όσο ο χρήστης σχεδιάζει τον κύκλο με το ποντίκι. Δέχεται σαν ορίσματα δύο **Point** τιμές, μία που δείχνει το κέντρο του κύκλου και μία που δείχνει ένα σημείο της περιφέρειας (αυτό που βρίσκεται εκείνη τη στιγμή κάτω από το δείκτη του ποντικιού). Η μέθοδος δεν έχει τιμή επιστροφής.

  Σαν ακτίνα του κύκλου πέρνουμε την απόσταση του σημείου της περιφέρειας από το κέντρο.

  Η θέση ορίζεται να είναι η θέση που βρίσκεται το κέντρο μείων την ακτίνα. Τέλος, ορίζουμε τις διαστάσεις του κύκλου.

```
public void modify(Point center, Point circum) {
  double radius = center.distance(circum);
  position.x = center.x - (int)radius;
  position.y = center.y - (int)radius;
  circle.width = circle.height = 2*radius;
}
```

Κώδικας της μεθόδου modify της κλάσης Circle

- **addElementNode**: Με αυτή την κλάση αποθηκεύουμε το σχήμα του κύκλου με βάση το πρότυπο SVG. Δέχεται σαν όρισμα μία τιμή τύπου **Document** η οποία δείχνει στο αρχείο SVG στο οποίο θα αποθηκευτεί το σκίτσο. Η μέθοδος δεν έχει τιμή επιστροφής.

  Αρχικά, δημιουργούμε ένα tag **<g></g>** με το οποίο ομαδοποιούμε τα δεδομένα που ακολουθούν, έτσι ώστε να μπορούμε να τα διαχειριστούμε εύκολα.

  Έπειτα, δημιουργούμε την ιδιότητα **transform** του <g> και της δίνουμε τις κατάλληλες τιμές (**translate** και **rotate**) έτσι ώστε να μετακινείται και να περιστρέφεται το σχήμα όπως πρέπει.

  Μετά, κατασκευάζουμε το tag **<circle/>** που συμβολίζει τον κύκλο στο SVG πρότυπο, και του προσθέτουμε τις ιδιότητες που απαιτούνται (**cx, cy, r, stroke, stroke-width, fill**).

  Τέλος, προσθέτουμε το <circle> που φτιάξαμε στο <g> και τοποθετούμε το <g> μέσα στο αρχείο.

```
public void addElementNode(Document doc) {
  org.w3c.dom.Element gElement = doc.createElement("g");
  Attr rotAttr = doc.createAttribute("transform");
  Double xx, yy;
  double help = circle.width/2.0;
```

```
    xx = position.getX() + help;
    yy = position.getY() + help;
    rotAttr.setValue("translate(" + xx + "," + yy +
          ") rotate(" + ((angle*100)/(1.7388)) + ")");
    gElement.setAttributeNode(rotAttr);
    org.w3c.dom.Element circleElement = doc.createElement("circle");
    Attr attr1 = doc.createAttribute("cx");
    attr1.setValue("0");
    circleElement.setAttributeNode(attr1);
    Attr attr2 = doc.createAttribute("cy");
    attr2.setValue("0");
    circleElement.setAttributeNode(attr2);
    Attr attr3 = doc.createAttribute("r");
    attr3.setValue(String.valueOf(circle.width/2.0));
    circleElement.setAttributeNode(attr3);
    Attr attr4 = doc.createAttribute("fill");
    attr4.setValue("none");
    circleElement.setAttributeNode(attr4);
    Attr attr5 = doc.createAttribute("stroke");
    attr5.setValue("black");
    circleElement.setAttributeNode(attr5);
    Attr attr6 = doc.createAttribute("stroke-width");
    attr6.setValue("1");
    circleElement.setAttributeNode(attr6);
    gElement.appendChild(circleElement);
    doc.getDocumentElement().appendChild(gElement);
  }
```

Κώδικας της μεθόδου addElementNode της κλάσης Circle

- **toString**: Μέθοδος που χρησιμοποιείται για να παίρνουμε τι είδους σχήμα είναι αυτό. Δεν έχει ορίσματα και επιστρέφει μία τιμή **String**.

```
  @Override
  public String toString()
  {
     return "circle";
  }
```

Κώδικας της μεθόδου toString της κλάσης Circle

- **getDimensions**: Αυτή η μέθοδος μας δίνει την ακτίνα του κύκλου που έχει σχεδιάσει ο χρήστης. Δεν έχει ορίσματα και επιστρέφει ένα **array τύπου double**.

  Αφού δηλώσουμε τις απαραίτητες μεταβλητές, μηδενίζουμε τις δύο πρώτες θέσεις του array. Μετά, υπολογίζουμε την ακτίνα και την τοποθετούμε στην τρίτη θέση του array. Τέλος, επιστρέφουμε το array.

```
  @Override
  public double[] getDimensions()
  {
     double[] ar = new double[3];
     ar[0] = 0;
     ar[1] = 0;
     ar[2] = circle.width / 2.0;
```

```
    return ar;
}
```

Κώδικας της μεθόδου getDimensions της κλάσης Circle

- **getText**: Αυτή η μέθοδος **δεν** χρησιμεύει στην κλάση Circle, οπότε επιστρέφουμε **null**.

```
public String getText()
{
    return "null";
}
```

Κώδικας της μεθόδου getText της κλάσης Circle

3.9.4 Καμπύλη

Για το σχήμα της καμπύλης έχει δημιουργηθεί η κλάση **Curve**, η οποία κάνει extend την κλάση Element. Μία καμπύλη είναι τύπου **QuadCurve2D.** Μέσα στη Curve υπάρχουν δύο constructors.

- Ο πρώτος δέχεται σαν ορίσματα **τρεις** τιμές τύπου **Point** (το αρχικό σημείο, το σημείο ελέγχου και το τελικό σημείο) και μία τιμή τύπου **Color** (σε περίπτωση που θέλουμε να έχουμε διαφορετικά χρώματα για κάθε σχήμα). Το χρώμα που χρησιμοποιείται στην εφαρμογή είναι αποκλειστικά το μαύρο. **Το σημείο ελέγχου** είναι ένα σημείο που βρίσκεται στο μέσο της απόστασης μεταξύ του αρχικού και του τελικού σημείου. Όσο πιο ψηλά βρίσκεται, τόσο πιο απότομη γίνεται η καμπύλη.

  Αυτός ο constructor χρησιμοποιείται κάθε φορά που ο χρήστης σχεδιάζει μία νέα καμπύλη στην περιοχή σχεδίασης.

```
public Curve(Point start, Point ctrl, Point end, Color color) {
  super(color);
  position = start;
 curve = new QuadCurve2D.Float(origin.x, origin.y, ctrl.x,
                        ctrl.y, end.x - position.x,
                        end.y - position.y);
}
```

Κώδικας του constructor της κλάσης Curve

- Ο δεύτερος constructor χρησιμοποιείται για να δημιουργηθεί ένα αντικείμενο καμπύλης όταν φορτώνεται από ένα αρχείο SVG. Δέχεται σαν όρισμα μία τιμή **org.w3c.dom.Element** η οποία αναφέρεται στο αρχείο SVG. Από το αρχείο, διαβάζονται, με διαδοχικές for, τα δεδομένα που αφορούν το σημείο ελέγχου και το τελικό σημείο της καμπύλης, από την ιδιότητα "**d**". Αυτό γίνεται γιατί στο SVG η καμπύλη είναι ένα **path** το οποίο το διαμορφώνουμε με την προηγούμενη ιδιότητα.

```
public Curve(org.w3c.dom.Element xmlElement) {
```

```
super(xmlElement);
  org.w3c.dom.Node node = xmlElement.getFirstChild();
  String d;
  d = ((org.w3c.dom.Element)node).getAttribute("d");
  d = d.substring(6);
  double x2=-4545.0, y2=-4545.0, ctrl_x=-4545.0, ctrl_y=-4545.0;
  String tempstr = new String();
  for(int i = 0; i < d.length(); i++)
  {
     if(d.substring(i, i+1).compareTo(",") != 0)
     {
        tempstr = tempstr + d.substring(i, i+1);
     }
     else
     {
        ctrl_x = Double.parseDouble(tempstr);
        ctrl_x += 20;
        d = d.substring(i+1);
        tempstr = "";
        break;
     }
  }
  for(int i = 0; i < d.length(); i++)
  {
     if(d.substring(i, i+1).compareTo(" ") != 0)
     {
        tempstr = tempstr + d.substring(i, i+1);
     }
     else
     {
        ctrl_y = Double.parseDouble(tempstr);
        ctrl_y -= 10;
        d = d.substring(i+1);
        tempstr = "";
        break;
     }
  }
  for(int i = 0; i < d.length(); i++)
  {
     if(d.substring(i, i+1).compareTo(" ") == 0)
     {
        d = d.substring(i+1);
        tempstr = "";
        break;
     }
  }
  for(int i = 0; i < d.length(); i++)
  {
     if(d.substring(i, i+1).compareTo(",") != 0)
     {
        tempstr = tempstr + d.substring(i, i+1);
     }
     else
     {
        x2 = Double.parseDouble(tempstr);
        d = d.substring(i+1);
        y2 = Double.parseDouble(d);
```

```
            tempstr = "";
            break;
        }
    }
  curve = new QuadCurve2D.Double(origin.x, origin.y, ctrl_x, ctrl_y, x2, y2);
}
```

Κώδικας του constructor της κλάσης Curve

Μετά από τους constructors, υπάρχουν οι μέθοδοι της κλάσης που διαχειρίζονται διάφορα θέματα. Αναλυτικά:

- **draw**: Μέθοδος η οποία καλεί τη μέθοδο **Element.draw** για να ζωγραφιστεί το σχήμα της καμπύλης. Δέχεται σαν όρισμα ένα **Graphics2D** context και δεν έχει τιμή επιστροφής.

```
public void draw(Graphics2D g2D) {
  draw(g2D, curve);
}
```

Κώδικας της μεθόδου draw της κλάσης Curve

- **getBounds**: Μέθοδος που χρησιμοποιείται για να πέρνουμε τα όρια μέσα στα οποία είναι σχεδιασμένη η καμπύλη. Δεν έχει ορίσματα και επιστρέφει ένα **java.awt.Rectangle** που περιβάλλει την καμπύλη. Το java.awt.Rectangle το παίρνει καλώντας την Element.getBounds.

```
public java.awt.Rectangle getBounds() {
  return getBounds(curve.getBounds());
}
```

Κώδικας της μεθόδου getBounds της κλάσης Curve

- **modify**: Με τη μέθοδο αυτή, ενημερώνεται το σχήμα της καμπύλης για τις αλλαγές που πρέπει να γίνουν όσο ο χρήστης σχεδιάζει την καμπύλη με το ποντίκι. Δέχεται σαν ορίσματα δύο **Point** τιμές, μία που δείχνει το αρχικό σημείο της καμπύλης (δεν χρησιμοποιείται) και μία που δείχνει το επόμενο σημείο (αυτό που βρίσκεται εκείνη τη στιγμή κάτω από το δείκτη του ποντικιού). Η μέθοδος δεν έχει τιμή επιστροφής.

  Σαν αρχικό σημείο ορίζουμε το position και σαν τελικό το next.

  Μετά, υπολογίζουμε την απόσταση αυτών των δύο σημείων με βάση τον τύπο $\sqrt{(x2-x1)^2+(y2-y1)^2}$, όπου (x1, y1) το αρχικό σημείο και (x2, y2) το τελικό. Το αποτέλεσμα το αποθηκεύουμε στη μεταβλητή d.

  Έπειτα, ορίζουμε το x2 να είναι η απόσταση του πρώτου σημείου της καμπύλης στον οριζόντιο άξονα προσθέτοντας την απόσταση d.

  Μετά, ορίζουμε το σημείο ελέγχου να βρίσκεται στη μέση της απόστασης του αρχικού σημείου της καμπύλης με το τελικό (στον οριζόντιο άξονα) και να έχει ύψος μειωμένο κατά τη μισή απόσταση των ίδιων σημείων.

Κατόπιν, περιστρέφουμε κατά -angle την καμπύλη για να επανέλθει στην οριζόντια θέση της και, ανάλογα με το που βρίσκεται ο δείκτης του ποντικιού, υπολογίζουμε τη γωνία που πρέπει να έχει η καμπύλη (χρησιμοποιώντας το atan) και την αποθηκεύουμε στη μεταβλητή atan.

Τέλος, περιστρέφουμε την καμπύλη κατά atan.

```
public void modify(Point start, Point next) {
  double x1, x2, y1, y2, ctrl_x, ctrl_y, d;
  x1 =position.x;
  y1 = position.y;
  x2 = next.x;
  y2 = next.y;
  d = Math.sqrt(Math.pow((x2 - x1), 2) + Math.pow((y2 - y1), 2));
  x2 = curve.getX1() + d;
  ctrl_x = curve.getX1() + (x2 - curve.getX1())/2;
  ctrl_y = -((x2 - curve.getX1())/2);
  curve.setCurve(curve.getX1(), curve.getY1(), ctrl_x,
              ctrl_y, x2, curve.getY1());
  length = curve.getX2() - curve.getX1();
  this.rotate(-this.angle);
  double atan, perp, perp2;
  if(next.x < position.x && next.y >= position.y)
  {
     perp = next.y - position.y;
     perp2 = position.x - next.x;
     atan = 1.5707963267948966 + Math.atan((perp2/perp));
  }
  else if(next.x < position.x && next.y < position.y)
  {
     perp = next.y - position.y;
     perp2 = position.x - next.x;
     atan = -1.5707963267948966 + Math.atan((perp2/perp));
  }
  else
  {
     perp = next.x - position.x;
     perp2 = next.y - position.y;
     atan = Math.atan((perp2/perp));
  }
  this.rotate(atan);
}
```

Κώδικας της μεθόδου modify της κλάσης Curve

- **addElementNode**: Με αυτή την κλάση αποθηκεύουμε το σχήμα της καμπύλης με βάση το πρότυπο SVG. Δέχεται σαν όρισμα μία τιμή τύπου **Document** η οποία δείχνει στο αρχείο SVG στο οποίο θα αποθηκευτεί το σκίτσο. Η μέθοδος δεν έχει τιμή επιστροφής.

  Αρχικά, δημιουργούμε ένα tag **<g></g>** με το οποίο ομαδοποιούμε τα δεδομένα που ακολουθούν, έτσι ώστε να μπορούμε να τα διαχειριστούμε εύκολα.

Έπειτα, δημιουργούμε την ιδιότητα **transform** του <g> και της δίνουμε τις κατάλληλες τιμές (**translate** και **rotate**) έτσι ώστε να μετακινείται και να περιστρέφεται το σχήμα όπως πρέπει.

Μετά, κατασκευάζουμε το tag **<path/>** που συμβολίζει τον κύκλο στο SVG πρότυπο, και του προσθέτουμε την ιδιότητα "d".

Τέλος, προσθέτουμε το <path> που φτιάξαμε στο <g> και τοποθετούμε το <g> μέσα στο αρχείο.

```
public void addElementNode(Document doc) {
  org.w3c.dom.Element gElement = doc.createElement("g");
  Attr rotAttr = doc.createAttribute("transform");
  rotAttr.setValue("translate(" + position.getX() + "," +
                position.getY() + ")" + " rotate(" +
                ((angle*100)/(1.7388)) + ")");
  gElement.setAttributeNode(rotAttr);
  org.w3c.dom.Element curveElement = doc.createElement("path");
  Attr attr1 = doc.createAttribute("d");
  attr1.setValue("M" + 0 + "," + 0 + " C" +
              (curve.getCtrlX() - 20) + "," +
              (curve.getCtrlY() + 10) + " " +
              (curve.getCtrlX() + 20) + "," +
              (curve.getCtrlY() + 10) + " " +
              curve.getX2() + "," + curve.getY2());
  curveElement.setAttributeNode(attr1);
  Attr attr2 = doc.createAttribute("fill");
  attr2.setValue("none");
  curveElement.setAttributeNode(attr2);
  Attr attr3 = doc.createAttribute("stroke");
  attr3.setValue("black");
  curveElement.setAttributeNode(attr3);
  Attr attr4 = doc.createAttribute("stroke-width");
  attr4.setValue("1");
  curveElement.setAttributeNode(attr4);
  gElement.appendChild(curveElement);
  doc.getDocumentElement().appendChild(gElement);
}
```

Κώδικας της μεθόδου addElementNode της κλάσης Curve

- **toString**: Μέθοδος που χρησιμοποιείται για να παίρνουμε τι είδους σχήμα είναι αυτό. Δεν έχει ορίσματα και επιστρέφει μία τιμή **String**.

```
@Override
public String toString()
{
   return "curve";
}
```

Κώδικας της μεθόδου toString της κλάσης Curve

- **getDimensions**: Αυτή η μέθοδος μας δίνει την ακτίνα του κύκλου που έχει σχεδιάσει ο χρήστης. Δεν έχει ορίσματα και επιστρέφει ένα **array τύπου double**.

Αφού δηλώσουμε τις απαραίτητες μεταβλητές, βάζουμε τη μεταβλητή **length** της κλάσης Curve στην πρώτη θέση του array. Μετά, μηδενίζουμε τις υπόλοιπες θέσεις και επιστρέφουμε το array.

```
@Override
public double[] getDimensions()
{
   double[] ar = new double[3];
   ar[0] = length;
   ar[1] = 0;
   ar[2] = 0;

   return ar;
}
```

Κώδικας της μεθόδου getDimensions της κλάσης Curve

- **getText**: Αυτή η μέθοδος **δεν** χρησιμεύει στην κλάση Curve, οπότε επιστρέφουμε **null**.

```
public String getText()
{
   return "null";
}
```

Κώδικας της μεθόδου getText της κλάσης Curve

3.9.5 Κείμενο

Για το σχήμα του κειμένου έχει δημιουργηθεί η κλάση **Text**, η οποία κάνει extend την κλάση Element. Μέσα στην Text υπάρχουν δύο constructors.

- Ο πρώτος δέχεται σαν ορίσματα **πέντε** τιμές. Η πρώτη τιμή είναι τύπου Font και μας χρησιμεύει στο να ορίζουμε τη γραμματοσειρά του κειμένου. Η δεύτερη τιμή είναι τύπου String (για να ορίζουμε το κείμενο). Η τρίτη είναι τύπου **Point** (σε πιό σημείο θέλουμε να τοποθετηθεί το κείμενο). Η τέταρτη μία τιμή τύπου **Color** (σε περίπτωση που θέλουμε να έχουμε διαφορετικά χρώματα για κάθε κείμενο). Το χρώμα που χρησιμοποιείται στην εφαρμογή είναι αποκλειστικά το μαύρο. Τέλος, η πέμπτη τιμή είναι μία **java.awt.Rectangle** bounds η οποία ορίζει τα όρια του κειμένου.

  Αυτός ο constructor χρησιμοποιείται κάθε φορά που ο χρήστης θέλει να εισάγει κείμενο στην περιοχή σχεδίασης.

```
public Text(Font font, String text, Point position,
        Color color, java.awt.Rectangle bounds) {
  super(color);
  this.font = font;
  this.position = position;
```

```
    this.position.y -= (int)bounds.getHeight();
    this.text = text;
    this.bounds = new java.awt.Rectangle(origin.x, origin.y,
                                         bounds.width,
                                         bounds.height);
  }
```

Κώδικας του constructor της κλάσης Text

- Ο δεύτερος constructor χρησιμοποιείται για να δημιουργηθεί ένα αντικείμενο κειμένου όταν φορτώνεται από ένα αρχείο SVG. Δέχεται σαν όρισμα μία τιμή **org.w3c.dom.Element** η οποία αναφέρεται στο αρχείο SVG. Αρχικά, διαβάζονται από το αρχείο το είδος της γραμματοσειράς και το μέγεθος της. Έπειτα, διαβάζουμε το κείμενο από το SVG και ορίζουμε τα όρια του κειμένου.

```
  public Text(org.w3c.dom.Element xmlElement) {
    super(xmlElement);
    // Get the font details
    org.w3c.dom.Node node = xmlElement.getFirstChild();
    String font_family, font_size;
    font_family = ((org.w3c.dom.Element)node).getAttribute("font-family");
    font_size = ((org.w3c.dom.Element)node).getAttribute("font-size");
    int style = 0;
    style = Font.PLAIN;
    font = new Font(font_family, style, Integer.parseInt(font_size));
    int text_length;
    // Get the string
    org.w3c.dom.NodeList list = xmlElement.getElementsByTagName("text");
    org.w3c.dom.Element string = (org.w3c.dom.Element)list.item(0);
    list = string.getChildNodes();
   StringBuffer textStr = new StringBuffer();
   for(int i = 0 ; i<list.getLength() ; i++) {
      if(list.item(i).getNodeType()==org.w3c.dom.Node.TEXT_NODE) {
        textStr.append(((org.w3c.dom.Text)list.item(i)).getData());
      }
   }
   text = textStr.toString().trim();
   text_length = text.length();
   // Get string bounds
   this.bounds = new java.awt.Rectangle(origin.x, origin.y, text_length * 6, 12);
  }
```

Κώδικας του constructor της κλάσης Text

Μετά από τους constructors, υπάρχουν οι μέθοδοι της κλάσης που διαχειρίζονται διάφορα θέματα. Αναλυτικά:

- **draw**: Μέθοδος η οποία καλεί τη μέθοδο **Element.draw** για να ζωγραφιστεί το σχήμα του κειμένου. Δέχεται σαν όρισμα ένα **Graphics2D** context και δεν έχει τιμή επιστροφής.

```
  public void draw(Graphics2D g2D)  {
    g2D.setPaint(highlighted ? Color.MAGENTA : color);
    Font oldFont = g2D.getFont();
```

```
  g2D.setFont(font);
  AffineTransform old = g2D.getTransform();
  g2D.translate(position.x, position.y);
  g2D.rotate(angle);
  g2D.drawString(text, origin.x, origin.y+(int)bounds.getHeight());
  g2D.setTransform(old);
  g2D.setFont(oldFont);
 }
```

Κώδικας της μεθόδου draw της κλάσης Text

- **getBounds**: Μέθοδος που χρησιμοποιείται για να πέρνουμε τα όρια μέσα στα οποία είναι σχεδιασμένο το κείμενο. Δεν έχει ορίσματα και επιστρέφει ένα **java.awt.Rectangle** που περιβάλλει το κείμενο. Το java.awt.Rectangle το παίρνει καλώντας την Element.getBounds.

```
public java.awt.Rectangle getBounds() {
  return getBounds(bounds);
}
```

Κώδικας της μεθόδου getBounds της κλάσης Text

- **modify**: Η μέθοδος αυτή δεν υλοποιείται στην κλάση Text, αφού δεν μπορούμε να πειράξουμε κάπως το κείμενο.

```
public void modify(Point start, Point last) {

}
```

Κώδικας της μεθόδου modify της κλάσης Text

- **addElementNode**: Με αυτή την κλάση αποθηκεύουμε το σχήμα του κειμένου με βάση το πρότυπο SVG. Δέχεται σαν όρισμα μία τιμή τύπου **Document** η οποία δείχνει στο αρχείο SVG στο οποίο θα αποθηκευτεί το σκίτσο. Η μέθοδος δεν έχει τιμή επιστροφής.

  Αρχικά, δημιουργούμε ένα tag **<g></g>** με το οποίο ομαδοποιούμε τα δεδομένα που ακολουθούν, έτσι ώστε να μπορούμε να τα διαχειριστούμε εύκολα.

  Έπειτα, δημιουργούμε την ιδιότητα **transform** του <g> και της δίνουμε τις κατάλληλες τιμές (**translate** και **rotate**) έτσι ώστε να μετακινείται και να περιστρέφεται το σχήμα όπως πρέπει.

  Μετά, κατασκευάζουμε το tag **<text></text>** που συμβολίζει το κείμενο στο SVG πρότυπο και του προσθέτουμε τις κατάλληλες ιδιότητες (**fill**, **font-family**, **font-size**, **x**, **y**).
  Κατόπιν, βάζουμε το κείμενο μέσα στο tag <text></text>.

  Τέλος, προσθέτουμε το <text></text> που φτιάξαμε στο <g> και τοποθετούμε το <g> μέσα στο αρχείο.

```
public void addElementNode(Document doc) {
  org.w3c.dom.Element gElement = doc.createElement("g");
  Attr rotAttr = doc.createAttribute("transform");
  rotAttr.setValue("translate(" + position.getX() + "," +
                (position.getY()) + ") rotate(" +
                ((angle*100)/(1.7388)) + ")");
  gElement.setAttributeNode(rotAttr);
  org.w3c.dom.Element textElement = doc.createElement("text");
  Attr attr1 = doc.createAttribute("x");
  attr1.setValue("0");
  textElement.setAttributeNode(attr1);
  Attr attr2 = doc.createAttribute("y");
  attr2.setValue("0");
  textElement.setAttributeNode(attr2);
  Attr attr3 = doc.createAttribute("font-family");
  attr3.setValue("Arial");
  textElement.setAttributeNode(attr3);
  Attr attr4 = doc.createAttribute("font-size");
  attr4.setValue(String.valueOf(12));
  textElement.setAttributeNode(attr4);
  Attr attr5 = doc.createAttribute("fill");
  attr5.setValue("black");
  textElement.setAttributeNode(attr5);
  textElement.appendChild(doc.createTextNode(text));
  gElement.appendChild(textElement);
  doc.getDocumentElement().appendChild(gElement);
}
```

Κώδικας της μεθόδου addElementNode της κλάσης Text

- **toString**: Μέθοδος που χρησιμοποιείται για να παίρνουμε τι είδους σχήμα είναι αυτό. Δεν έχει ορίσματα και επιστρέφει μία τιμή **String**.

```
@Override
public String toString()
{
   return "text";
}
```

Κώδικας της μεθόδου toString της κλάσης Text

- **getDimensions**: Αυτή η μέθοδος μας δίνει το πλήθος των χαρακτήρων που έχει δώσει ο χρήστης. Δεν έχει ορίσματα και επιστρέφει ένα **array τύπου double**.

  Αφού δηλώσουμε τις απαραίτητες μεταβλητές, βάζουμε το **text.toString().length()** της μεταβλητής που έχει το κείμενο στην πρώτη θέση του array. Μετά, μηδενίζουμε τις υπόλοιπες θέσεις και επιστρέφουμε το array.

```
@Override
public double[] getDimensions()
{
   double[] ar = new double[3];
   ar[0] = text.toString().length();
   ar[1] = 0;
   ar[2] = 0;
```

```
    return ar;
}
```

Κώδικας της μεθόδου getDimensions της κλάσης Text

- **getText**: Αυτή η μέθοδος επιστρέφει το κείμενο που έχει δώσει ο χρήστης.

```
public String getText()
{
    return text;
}
```

Κώδικας της μεθόδου getText της κλάσης Text

3.9.6 Αντικείμενα

Τα αντικείμενα που έχει στη διάθεση του ο χρήστης (όπως τραπέζια κλπ) δεν είναι τίποτε άλλο από αρχεία SVG με τα βασικά σχήματα που δίνει η εφαρμογή. Αποτελούνται δηλαδή από γραμμές, ορθογώνια, κύκλους, καμπύλες και κείμενο. Η διαχείριση τους γίνεται σχεδόν όπως και των απλών σχημάτων. Η μόνη διαφορά τους είναι κάποιες λεπτομέρειες στη διαχείριση τους με τα συμβάντα που προκαλούνται από το ποντίκι. Αυτές θα αναλυθούν σε επόμενη ενότητα. Η αποθήκευση τους και η ανάκτηση τους γίνεται κανονικά, όπως και στα υπόλοιπα σχήματα.

3.10 Υλοποίηση μετασχηματισμών

Τα σχήματα που έχει στη διάθεση του ο χρήστης μπορούν να δεχτούν 4 διαφορετικούς μετασχηματισμούς. Ο χρήστης μπορεί να μετακινήσει ένα σχήμα πάνω στην περιοχή σχεδίασης, μπορεί να περιστρέψει σχήματα, να διαγράψει σχήματα και να τα μετακινήσει στο υπόβαθρο. Όλα αυτά υλοποιούνται μέσα στην κλάση **SketchView**, στο event **MouseDragged**. Οι μετασχηματισμοί περιγράφονται παρακάτω.

3.10.1 Μετακίνηση

Για τη μετακίνηση, φτιάχνουμε ένα νέο Object (το **ob**) και του εισάγουμε το επιλεγμένο αντικείμενο. Έπειτα δίνουμε σε μία μεταβλητή το index στο οποίο βρίσκεται το αντικείμενο μέσα στη λίστα με τα elements. Επίσης, σε δύο μεταβλητές τύπου String (**elemStr** και **elemTxt**) δίνουμε τον τύπο και το κείμενο του αντικειμένου αντίστοιχα.

Ανάλογα με τον τύπο του αντικειμένου, γίνεται η μετακίνηση και σε κάποια άλλα αντικείμενα τα οποία πάνε "πακέτο" με το επιλεγμένο (π.χ. το κείμενο που συνοδεύει τα έτοιμα αντικείμενα όπως είναι το τραπέζι). Τέλος, γίνεται μετακίνηση και στο επιλεγμένο αντικείμενο με τη μέθοδο **move** του αντικειμένου, δίνοντας σαν ορίσματα της θέση στην οποία θέλουμε να πάει.

```
else if(button1Down && mode == MOVE && selectedElement != null) {
    Object ob = new Object();
    ob = selectedElement;
    int indx = theApp.sketch.elements.indexOf(ob);
    String elemStr = new String();
    String elemTxt = new String();
    try
    {
       elemStr = theApp.sketch.elements.get(indx + 1).toString();
       elemTxt = theApp.sketch.elements.get(indx + 1).getText();
    }
    catch(Exception ee)
    {
       elemStr = "null";
       elemTxt = "null";
    }
    if(elemStr.compareTo("text") == 0)
    {
       if(elemTxt.compareTo("Bed 1") == 0 ||
         elemTxt.compareTo("Bed 2") == 0 ||
         elemTxt.compareTo("Chair") == 0 ||
         elemTxt.compareTo("Sofa 1") == 0 ||
         elemTxt.compareTo("Sofa 2") == 0 ||
         elemTxt.compareTo("Table 1") == 0 ||
         elemTxt.compareTo("Table 2") == 0 ||
         elemTxt.compareTo("Table 3") == 0 ||
         elemTxt.compareTo("Window") == 0)
       {
          theApp.sketch.elements.get(indx + 1).move(last.x - start.x, last.y - start.y);
       }
    }
    else if(elemStr.compareTo("curve") == 0)
    {
       String elemStr2 = theApp.sketch.elements.get(indx).toString();
       double dim[] = theApp.sketch.elements.get(indx).getDimensions();
       double wid = dim[0];
       double hei = dim[1];
       if(elemStr2.compareTo("rectangle") == 0 &&
         wid == 4.0 && hei == 26.0)
       {
          theApp.sketch.elements.get(indx + 1).move(last.x - start.x, last.y - start.y);
       }
    }
    selectedElement.draw(g2D);              // Draw to erase the element
    selectedElement.move(last.x-start.x, last.y-start.y);  // Move it
    selectedElement.draw(g2D);              // Draw in its new position
    start = last;                           // Make start current point
}
```

Κώδικας για τη μετακίνηση ενός σχήματος

3.*10.2 Περιστροφή*

Για τη περιστροφή, φτιάχνουμε ένα νέο Object (το **ob**) και του εισάγουμε το επιλεγμένο αντικείμενο. Έπειτα δίνουμε σε μία μεταβλητή το index στο οποίο βρίσκεται το αντικείμενο μέσα στη λίστα με τα elements. Επίσης, σε δύο μεταβλητές τύπου String (**elemStr** και **elemTxt**) δίνουμε τον τύπο και το κείμενο του αντικειμένου αντίστοιχα.

Μετά, γίνεται η περιστροφή του αντικειμένου και αφού γίνει ταυτοποίηση αυτού, μετακινούμε και τα άλλα αντικείμενα που πάνε "πακέτο" μαζί του στα όρια που το περιβάλλουν, έτσι ώστε να μην πέφτουν πάνω στο αντικείμενο μετά την περιστροφή.

```
Object ob = new Object();
      ob = selectedElement;
      int indx = theApp.sketch.elements.indexOf(ob);
      String elemStr = new String();
      String elemTxt = new String();
      try
      {
         elemStr = theApp.sketch.elements.get(indx + 1).toString();
         elemTxt = theApp.sketch.elements.get(indx + 1).getText();
      }
      catch(Exception ee)
      {
         elemStr = "null";
         elemTxt = "null";
      }
      Point initial_rect_pos = new Point();
      initial_rect_pos.setLocation(selectedElement.getBounds().x,
                              selectedElement.getBounds().y);
      selectedElement.draw(g2D);                  // Draw to erase the element
      double ang = getAngle(selectedElement.getPosition(), start, last);
      selectedElement.rotate(ang);
      selectedElement.draw(g2D);                  // Draw in its new position
      if(elemStr.compareTo("text") == 0)
      {
         if(elemTxt.compareTo("Bed 1") == 0 ||
           elemTxt.compareTo("Bed 2") == 0 ||
           elemTxt.compareTo("Chair") == 0 ||
           elemTxt.compareTo("Sofa 1") == 0 ||
           elemTxt.compareTo("Sofa 2") == 0 ||
           elemTxt.compareTo("Table 1") == 0 ||
           elemTxt.compareTo("Table 2") == 0 ||
           elemTxt.compareTo("Table 3") == 0 ||
           elemTxt.compareTo("Window") == 0)
         {
            int xx, yy;
            xx = selectedElement.getBounds().x -
                theApp.sketch.elements.get(indx + 1).position.x;
            xx -= theApp.sketch.elements.get(indx + 1).getBounds().width - 3;
            yy = selectedElement.getBounds().y -
                theApp.sketch.elements.get(indx + 1).position.y;
            yy -= theApp.sketch.elements.get(indx + 1).getBounds().height - 3;
            theApp.sketch.elements.get(indx + 1).move(xx, yy);
         }
```

```
    }
    else if(elemStr.compareTo("curve") == 0)
    {
       String elemStr2 = theApp.sketch.elements.get(indx).toString();
       double dim[] = theApp.sketch.elements.get(indx).getDimensions();
       double wid = dim[0];
       double hei = dim[1];
       if(elemStr2.compareTo("rectangle") == 0 && wid == 4.0 && hei == 26.0)
       {
          ang = getAngle(selectedElement.getPosition(), start, last);
          theApp.sketch.elements.get(indx + 1).rotate(ang);
       }
    }
    start = last;                          // Make start current point
```

Κώδικας για την περιστροφή ενός σχήματος

3.10.3 Διαγραφή

Όταν ο χρήστης κάνει δεξί κλικ πάνω σε ένα αντικείμενο και επιλέξει να το διαγράψει, τότε καλείται η μέθοδος **actionPerformed** που βρίσκεται μέσα στην κλάση **SketchView**, και η οποία διαχειρίζεται τα κλικ που γίνονται πάνω στο μενού.

Για τη διαγραφή, φτιάχνουμε ένα νέο Object (το **ob**) και του εισάγουμε το επιλεγμένο αντικείμενο. Έπειτα δίνουμε σε μία μεταβλητή το index στο οποίο βρίσκεται το αντικείμενο μέσα στη λίστα με τα elements. Επίσης, σε δύο μεταβλητές τύπου String (**elemStr** και **elemTxt**) δίνουμε τον τύπο και το κείμενο του αντικειμένου αντίστοιχα.

Έπειτα, ελέγχουμε τον τύπο του αντικειμένου, έτσι ώστε να διαγράψουμε και τα αντικείμενα που τυχών πάνε "πακέτο" με το επιλεγμένο αντικείμενο. Αφού διαγραφούν αυτά, διαγράφουμε το αντικείμενο από τη λίστα όπως επίσης και κάθε αναφορά προς αυτό.

```
else if(source == deleteItem) {
    if(highlightElement != null) {                  // If there's an element
      Object ob = new Object();
      ob = highlightElement;
      int indx = theApp.sketch.elements.indexOf(ob);
      String elemStr = new String();
      String elemTxt = new String();
      try
      {
         elemStr = theApp.sketch.elements.get(indx + 1).toString();
         elemTxt = theApp.sketch.elements.get(indx + 1).getText();
      }
      catch(Exception ee)
      {
         elemStr = "null";
         elemTxt = "null";
      }
      if(elemStr.compareTo("text") == 0)
      {
         if(elemTxt.compareTo("Bed 1") == 0 ||
           elemTxt.compareTo("Bed 2") == 0 ||
```

```
            elemTxt.compareTo("Chair") == 0 ||
            elemTxt.compareTo("Sofa 1") == 0 ||
            elemTxt.compareTo("Sofa 2") == 0 ||
            elemTxt.compareTo("Table 1") == 0 ||
            elemTxt.compareTo("Table 2") == 0 ||
            elemTxt.compareTo("Table 3") == 0 ||
            elemTxt.compareTo("Window") == 0)
          {
             theApp.getModel().remove(theApp.sketch.elements.get(indx + 1));
          }
       }
       else if(elemStr.compareTo("curve") == 0)
       {
          String elemStr2 = theApp.sketch.elements.get(indx).toString();
          double dim[] = theApp.sketch.elements.get(indx).getDimensions();
          double wid = dim[0];
          double hei = dim[1];

          if(elemStr2.compareTo("rectangle") == 0 && wid == 4.0 && hei == 26.0)
          {
             theApp.getModel().remove(theApp.sketch.elements.get(indx + 1));
          }
       }
       theApp.getModel().remove(highlightElement);     // then remove it
       highlightElement = null;                        // Remove the reference
     }

   }
```

Κώδικας για τη διαγραφή ενός σχήματος

3.10.4 Μετακίνηση στο υπόβαθρο

Για τη μετακίνηση ενός αντικειμένου στο υπόβαθρο, χρησιμοποιείται και πάλι η μέθοδος actionPerformed που βρίσκεται μέσα στη κλάση SketchView. Εδώ, ουσιαστικά διαγράφουμε το επιλεγμένο αντικείμενο και το προσθέτουμε εκ νέου στη λίστα, οπότε αυτομάτως βρίσκεται πίσω από όλα τα άλλα αντικείμενα.

```
else if(source == sendToBackItem) {
     if(highlightElement != null) {
       theApp.getModel().remove(highlightElement);
       theApp.getModel().add(highlightElement);
       highlightElement.setHighlighted(false);
       highlightElement = null;
       repaint();
     }
   }
```

Κώδικας για τη μετακίνηση στο υπόβραθρο ενός σχήματος

3.11 Υλοποίηση Toolbars

Στην εφαρμογή υπάρχουν δύο toolbars, ένα στο πάνω μέρος του παραθύρου κάτω από τα μενού, και ένα στο δεξί μέρος. Το πρώτο παρέχει όλες τις δυνατότητες διαχείρισης και σχεδίασης (save, load κλπ). Το δεύτερο περέχει στον χρήστη διάφορα αντικείμενα για να τα εισάγει στο σκίτσο του. Και τα δύο toolbars είναι τύπου **jToolBar** και υλοποιούνται μέσα στην κλάση **SketchFrame**.

3.11.1 Οριζόντιο toolbar

Για το οριζόντιο toolbar έχει δηλωθεί μία jToolBar μεταβλητή (η **toolBar**). Επίσης, έχει δημιουργηθεί μία μέθοδος, η **addToolBarButton**, για να προστίθενται κουμπιά στο toolBar.

Αναλυτικά, η addToolBarButton δέχεται 2 ορίσματα, ένα **Action** για το τι θα κάνει το κουμπί (βλ ενότητα **3.14**), και ένα **String** που θα χρησιμοποιηθεί σαν κείμενο για το κουμπί και έχει σαν τιμή επιστροφής ένα **jButton**. Έπειτα, δημιουργείται ένα **jButton** στο οποίο τοποθετούμε ένα εικονίδιο (ανάλογα με το κείμενο του κουμπιού) και τοποθετείται στο toolBar. Τέλος, δημιουργούμε **mouseListener** για να αλλάζει ο δείκτης του ποντικιού εικονίδιο και επιστρέφουμε το κουμπί που μόλις δημιουργήσαμε.

```
 private JButton addToolBarButton(Action action, String caption)
{
   JButton button = new JButton(action);
   button.setIcon(new ImageIcon(getClass().getResource("Images/"
                                + caption + ".gif")));
   button.setText(caption);
   toolBar.add(button);
   button.addMouseListener(new java.awt.event.MouseAdapter() {
      @Override
      public void mouseEntered(java.awt.event.MouseEvent evt) {
         setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
      }
      @Override
      public void mouseExited(java.awt.event.MouseEvent evt) {
         setCursor(new java.awt.Cursor(java.awt.Cursor.CROSSHAIR_CURSOR));
      }
   });
   return button;
}
```

Κώδικας της μεθόδου addToolBarButton της κλάσης SketchFrame

3.11.2 Κάθετο toolbar

Για το κάθετο toolbar έχει δηλωθεί μία jToolBar μεταβλητή (η **toolBar2**). Επίσης, έχει δημιουργηθεί μία μέθοδος, η **addToolBarButton2**, για να προστίθενται κουμπιά στο toolBar2.

Αναλυτικά, η addToolBarButton2 δέχεται 2 ορίσματα, ένα **Action** για το τι θα κάνει το κουμπί (βλ ενότητα **2.14**), και ένα **String** που θα χρησιμοποιηθεί σαν κείμενο για το κουμπί και έχει σαν τιμή επιστροφής ένα **jButton**. Έπειτα, δημιουργείται ένα **jButton** στο οποίο τοποθετούμε ένα εικονίδιο (ανάλογα με το κείμενο του κουμπιού) και τοποθετείται στο toolBar. Τέλος, δημιουργούμε **mouseListener** για να αλλάζει ο δείκτης του ποντικιού εικονίδιο και επιστρέφουμε το κουμπί που μόλις δημιουργήσαμε.

```
private JButton addToolBarButton2(Action action, String caption)
{
   JButton button = new JButton(action);
   button.setIcon(new ImageIcon(getClass().getResource("Images/"
                           + caption + ".gif")));
   button.setText(caption);
   toolBar2.add(button);

   button.addMouseListener(new java.awt.event.MouseAdapter() {
      @Override
      public void mouseEntered(java.awt.event.MouseEvent evt) {
         setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
      }
      @Override
      public void mouseExited(java.awt.event.MouseEvent evt) {
         setCursor(new java.awt.Cursor(java.awt.Cursor.CROSSHAIR_CURSOR));
      }
   });
   return button;
}
```

Κώδικας της μεθόδου addToolBarButton2 της κλάσης SketchFrame

3.12 Υλοποίηση Status Bar

Το Status Bar που υπάρχει στο κάτω μέρος του παραθύρου χρησιμεύει στο να πληροφορεί τον χρήστη για το σχήμα που έχει επιλέξει προς σχεδίαση, τη θέση του δείκτη του ποντικιού και τις διαστάσεις κάποιου σχήματος.

Για το σκοπό αυτό έχει υλοποιηθεί μία ξεχωριστή κλάση, η **StatusBar** που κάνει extend το **jPanel**, η οποία περιέχει όλο τον κώδικα για τις παραπάνω λειτουργίες. Η κάθε πληροφορία εμφανίζεται μέσα σε ένα **StatusPane** (κλάση που έχει υλοποιηθεί μέσα στην

StatusBar) το οποίο κάνει extend το **jLabel**.

Η κλάση StatusBar έχει έναν constructor χωρίς ορίσματα, ο οποίος δημιουργεί την εμφάνιση της StatusBar και προσθέτει τους **listeners** για τα mouse events. Για τη δημιουργία των απαραίτητων StatusPanes, χρησιμοποιούνται κάποιες μέθοδοι, οι οποίες αναλύονται στις επόμενες ενότητες.

```
public StatusBar() {
  setLayout(new FlowLayout(FlowLayout.LEFT, 10, 3));
  setBackground(Color.LIGHT_GRAY);
  setBorder(BorderFactory.createLineBorder(Color.DARK_GRAY));
  setTypePane(DEFAULT_ELEMENT_TYPE);
  add(typePane);
  add(mousePaneX);
  add(mousePaneY);
  add(dimPane);
  addMouseListener(new java.awt.event.MouseAdapter() {
     @Override
     public void mouseEntered(java.awt.event.MouseEvent evt) {
        setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
     }
     @Override
     public void mouseExited(java.awt.event.MouseEvent evt) {
        setCursor(new java.awt.Cursor(java.awt.Cursor.CROSSHAIR_CURSOR));
     }
  });
  typePane.addMouseListener(new java.awt.event.MouseAdapter() {
     ...
  });
  mousePaneX.addMouseListener(new java.awt.event.MouseAdapter() {
     ...
  });
  mousePaneY.addMouseListener(new java.awt.event.MouseAdapter() {
     ...
  });
  dimPane.addMouseListener(new java.awt.event.MouseAdapter() {
     ...
  });
}
```

Constructor της κλάσης StatusBar

Η κλάση StatusPane έχει έναν constructor, ο οποίος έχει σαν όρισμα μία String τιμή (το κείμενο που θα εμφανίζεται μέσα στο StatusPane). Ο constructor ορίζει επίσης την εμφάνιση που θα έχει το StatusPane.

```
class StatusPane extends JLabel {
  public StatusPane(String text) {
    setOpaque(true);
    setBackground(Color.YELLOW);
    setForeground(Color.BLACK);
    setFont(paneFont);
    setHorizontalAlignment(CENTER);
```

```
    setBorder(BorderFactory.createLineBorder(Color.BLACK));
    setPreferredSize(new Dimension(135,20));
    setText(text);
  }
  private Font paneFont = new Font("Arial", Font.PLAIN, 10);
}
```

Κώδικας της κλάσης StatusPane

3.12.1 Ένδειξη ενεργού σχήματος

Για την ένδειξη του επιλεγμένου σχήματος από τον χρήστη, έχει δημιουργηθεί η μέθοδος **setTypePane**, η οποία έχει σαν όρισμα έναν ακέραιο (ανάλογα με την τιμή του υποδηλώνει εάν πρόκειται για σχήμα γραμμής, ορθογώνιου, κύκλου, καμπύλης ή κειμένου). Οι ακέραιες σταθερές **Line**, **Rect**, **Circle**, **Curve** και **Text** που χρησιμοποιούνται για αυτό το λόγο είναι ορισμένες στην κλάση **SketcherConstants**. Ανάλογα με την τιμή που θα έχει το όρισμα της μεθόδου, τυπώνεται το κατάλληλο κείμενο στο StatusPane.

```
public void setTypePane (int elementType) {
  String text = null;              // Text for the type pane
  switch(elementType) {
    case LINE:
      text = "Selected Shape: LINE";
      break;
    case RECTANGLE:
      text = "Selected Shape: RECT";
      break;
    case CIRCLE:
      text = "Selected Shape: CIRCLE";
      break;
    case CURVE:
      text = "Selected Shape: CURVE";
      break;
    case TEXT:
      text = "Selected Shape: TEXT";
      break;
    default:
      assert false;
  }
  typePane.setText(text);              // Set the pane text
}
```

Κώδικας της μεθόδου setTypePane της κλάσης StatusBar

3.12.2 Ενδείξεις θέσης δείκτη ποντικιού

Για τις ενδείξεις της θέσης του δείκτη του ποντικιού έχει δημιουργηθεί η μέθοδος **setMouseCoord** η οποία δέχεται σαν ορίσματα **δύο ακέραιους** (έναν για τη θέση του κέρσορα στον οριζόντιο άξονα και έναν για τη θέση στον κατακόρυφο άξονα). Η μέθοδος εμφανίζει σε δύο ξεχωριστά StatusPanes τη θέση σε δύο μορφές

(σε **pixels** και σε **μέτρα** – ένα μέτρο είναι 25 pixels).

```
public void setMouseCoord(int x, int y)
{
   String text = null;
   text = "X: " + String.valueOf(x) + " px  -  " + String.valueOf(x/25.0) +  " m";
  mousePaneX.setText(text);
   text = "Y: " + String.valueOf(y) + "px  -  " + String.valueOf(y/25.0) + " m";
   mousePaneY.setText(text);
}
```

Κώδικας της μεθόδου setMouseCoord της κλάσης StatusBar

3.12.3 Ένδειξη διαστάσεων σχήματος

Για την ένδειξη των διαστάσεων των διάφορων σχημάτων έχει δημιουργηθεί η μέθοδος **setDimensions**, η οποία δέχεται σαν ορίσματα τρεις **double** τιμές και μία **String** (για το πλάτος, το ύψος, την ακτίνα και τον τύπο του σχήματος αντίστοιχα). Ανάλογα με το τι τύπο σχήματος έχουμε παίρνουμε και τις ανάλογες ενδείξεις:

- **Γραμμή**: εμφανίζει το μήκος της γραμμής (χρησιμοποιείται η μεταβλητή **w**).
- **Ορθογώνιο**: εμφανίζει το ύψος και το πλάτος (χρησιμοποιούνται οι μεταβλητές **w** και **h**).
- **Κύκλος**: εμφανίζει την ακτίνα του κύκλου (χρησιμοποιείται η μεταβλητή **rad**).
- **Καμπύλη**: εμφανίζει την απόσταση του πρώτου σημείου της καμπύλης με το τελευταίο (χρησιμοποιείται η μεταβλητή **w**).
- **Κείμενο**: εμφανίζει πόσους χαρακτήρες περιέχει το κείμενο (χρησιμοποιείται η μεταβλητή **w**).

```
public void setDimensions(double w, double h, double rad, String type)
{
   if(type.compareTo("line") == 0)
   {
      String str;
      str = String.valueOf(w);
      if(str.length() > 4)
      {
        dimPane.setText("Length: " + str.substring(0, 4) + " m");
      }
      else
      {
         dimPane.setText("Length: " + str.substring(0, 3) + " m");
      }}
   else if(type.compareTo("rectangle") == 0)
   {
      dimPane.setText("W: " + w  + " m" + " | H: " + h + " m");
   }
   else if(type.compareTo("circle") == 0)
   {
```

```
        String str;
        str = String.valueOf(rad);
        if(str.length() > 4)
        {
          dimPane.setText("Radius: " + str.substring(0, 4) + " m");
        }
        else
        {
           dimPane.setText("Radius: " + str.substring(0, 3) + " m");
        }
    }
    else if(type.compareTo("curve") == 0)
    {
       dimPane.setText("Length: " + w + " m");
    }
    else if(type.compareTo("text") == 0)
    {
       dimPane.setText("Characters: " + w);
    }
    else
    {
       dimPane.setText(type);
    }
 }
```

Κώδικας της μεθόδου setDimensions της κλάσης StatusBar

3.13 Υλοποίηση Μενού

Το μενού που έχει στη διάθεση του ο χρήστης έχει υλοποιηθεί μέσα στην κλάση **SketchFrame**. Για την υλοποίηση του κεντρικού μενού χρησιμοποιήθηκε ένα **jMenuBar** (το **menuBar**) το οποίο είναι ο container για κάθε μενού που θέλουμε να δημιουργήσουμε στην εφαρμογή μας.

Αφού προσθέσουμε το menuBar στο frame μας και του εισάγουμε **mouseListener**, του προσθέτουμε σιγά σιγά τα μενού. Το πως υλοποιούνται τα μενού αναλύεται παρακάτω.

```
public SketchFrame(String title, Sketcher theApp)
{
   ...
   setJMenuBar(menuBar);
   menuBar.addMouseListener(new java.awt.event.MouseAdapter() {
      @Override
      public void mouseEntered(java.awt.event.MouseEvent evt) {
         setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
      }
      @Override
      public void mouseExited(java.awt.event.MouseEvent evt) {
         setCursor(new java.awt.Cursor(java.awt.Cursor.CROSSHAIR_CURSOR));
      }
```

```
    });
    ...
    menuBar.add(fileMenu);
    menuBar.add(elementMenu);
    menuBar.add(helpMenu);
    ...
}
private JMenuBar menuBar = new JMenuBar();
```

Κώδικας για το menuBar στην κλάση SketchFrame

3.13.1 Μενού File

Το μενού File είναι ένα αντικείμενο τύπου **Jmenu**. Αφού το δηλώσουμε και το δημιουργήσουμε, του ορίζουμε σαν πλήκτρο συντόμευσης (όταν πατηθεί το κουμπί ALT μαζί με το κουμπί συντόμευσης ανοίγει το μενού) το γράμμα **F**.

Έπειτα, προσθέτουμε στο μενού ένα **mouseListener** για τα **mouseEntered** και **mouseExited** events, έτσι ώστε όταν το ποντίκι περνάει πάνω από το μενού να αλλάζει ο κέρσορας στο κλασικό δείκτη, και όταν βγαίνει από το μενού και περνάει πάνω από την περιοχή σχεδίασης να αλλάζει ο κέρσορας σε σταυρό.

Μετά, φτιάχνουμε τα **actions** για τις επιλογές που θα παρέχει το μενού. Δημιουργούμε ένα action για κάθε λειτουργία, όπως Νέο Αρχείο, Αποθήκευση κλπ. Στη συνέχεια δημιουργούμε τα αντικείμενα που απαρτίζουν το μενού (**menu items**) με βάση τα actions.

Αφού τα δημιουργήσουμε, τους προσθέτουμε **actionListeners**, ώστε και πάλι να αλλάζει ο κέρσορας ανάλογα με τη θέση του. Τέλος, προσθέτουμε τα menu items στο μενού.

```
JMenu fileMenu = new JMenu("File");       // Create File menu
fileMenu.setMnemonic('F');                // Create shortcut
fileMenu.addMouseListener(new java.awt.event.MouseAdapter() {
    @Override
    public void mouseEntered(java.awt.event.MouseEvent evt) {
        setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
    }
    @Override
    public void mouseExited(java.awt.event.MouseEvent evt) {
        setCursor(new java.awt.Cursor(java.awt.Cursor.CROSSHAIR_CURSOR));
    }
});
// Create the action items for the file menu
newAction = new FileAction("New", KeyStroke.getKeyStroke('N',
                        CTRL_DOWN_MASK),"Create new sketch");
exportAction = new XMLExportAction("Save SVG",
                                KeyStroke.getKeyStroke('S',
                                CTRL_DOWN_MASK), "Save sketch as
                                an SVG file");
importAction = new XMLImportAction("Open SVG",
```

```
                                KeyStroke.getKeyStroke('O',
                                CTRL_DOWN_MASK), "Open sketch
                                from an SVG file");
closeAction = new FileAction("Exit",KeyStroke.getKeyStroke('X',
                            CTRL_DOWN_MASK ), "Exit Sketcher");

// Construct the file drop-down menu
JMenuItem it1 = new JMenuItem(newAction);
JMenuItem it2 = new JMenuItem(importAction);
JMenuItem it3 = new JMenuItem(exportAction);
JMenuItem it4 = new JMenuItem(closeAction);
it1.addMouseListener(new java.awt.event.MouseAdapter() {
   @Override
   public void mouseEntered(java.awt.event.MouseEvent evt) {
      setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
   }

   @Override
   public void mouseExited(java.awt.event.MouseEvent evt) {
      setCursor(new java.awt.Cursor(java.awt.Cursor.CROSSHAIR_CURSOR));
   }
});
it2.addMouseListener(new java.awt.event.MouseAdapter() {
   ...
});
it3.addMouseListener(new java.awt.event.MouseAdapter() {
   ...
});
it4.addMouseListener(new java.awt.event.MouseAdapter() {
   ...
});
fileMenu.add(it1);
fileMenu.add(it2);
fileMenu.addSeparator();              // Add separator
fileMenu.add(it3);
fileMenu.addSeparator();              // Add separator
fileMenu.add(it4);
```

Κώδικας για το File menu στην κλάση SketchFrame

3.13.2 Μενού Draw

Το μενού **Draw** είναι ένα αντικείμενο τύπου **JMenu**. Αφού το δηλώσουμε και το δημιουργήσουμε, του ορίζουμε σαν πλήκτρο συντόμευσης (όταν πατηθεί το κουμπί ALT μαζί με το κουμπί συντόμευσης ανοίγει το μενού) το γράμμα **E**.

Έπειτα, προσθέτουμε στο μενού ένα **mouseListener** για τα **mouseEntered** και **mouseExited** events, έτσι ώστε όταν το ποντίκι περνάει πάνω από το μενού να αλλάζει ο κέρσορας στο κλασικό δείκτη, και όταν βγαίνει από το μενού και περνάει πάνω από την περιοχή σχεδίασης να αλλάζει ο κέρσορας σε σταυρό.

Μετά, φτιάχνουμε τα **actions** για τις επιλογές που θα παρέχει το μενού. Δημιουργούμε ένα action για κάθε λειτουργία, όπως Γραμμή κλπ. Στη συνέχεια

δημιουργούμε τα αντικείμενα που απαρτίζουν το μενού (**menu items**) με βάση τα actions.

Αφού τα δημιουργήσουμε, τους προσθέτουμε **actionListeners**, ώστε και πάλι να αλλάζει ο κέρσορας ανάλογα με τη θέση του. Τέλος, προσθέτουμε τα menu items στο μενού.

Επιπρόσθετα, ο χρήστης έχει τη δυνατότητα να εισάγει τα έτοιμα αντικείμενα στην περιοχή σχεδίασης μέσω του υπομενού **Objects** που βρίσκεται μέσα στο μενού Draw. Και αυτό το υπομενού είναι τύπου Jmenu.

Αφού το δηλώσουμε και το δημιουργήσουμε, του προσθέτουμε ένα **mouseListener** για τα **mouseEntered** και **mouseExited** events, έτσι ώστε όταν το ποντίκι περνάει πάνω από το μενού να αλλάζει ο κέρσορας στο κλασικό δείκτη, και όταν βγαίνει από το μενού και περνάει πάνω από την περιοχή σχεδίασης να αλλάζει ο κέρσορας σε σταυρό. Μετά το προσθέτουμε στο μενού Draw.

Το υπομενού Objects απαρτίζεται από άλλα υπομενού, τα οποία είναι οι κατηγορίες των έτοιμων αντικειμένων. Αυτά τα υπομενού είναι επίσης τύπου **Jmenu**. Αφού τα δηλώσουμε και τα δημιουργήσουμε, προσθέτουμε και σε αυτά τους **actionListeners** για την αλλαγή του κέρσορα.

Μετά, τα προσθέτουμε στο υπομενού Objects και δημιουργούμε τα απαραίτητα menu items με βάση τα actions που απαιτούνται. Για μία ακόμη φορά προσθέτουμε σε αυτά τα menu items τους απαραίτητους actionListeners για την αλαγή του κέρσορα. Τέλος, προσθέτουμε τα menu items στα κατάλληλα υπομενού.

```
JMenu elementMenu = new JMenu("Draw");// Create Elements menu
elementMenu.setMnemonic('E');              // Create shortcut
...
JMenuItem it5 = new JMenuItem(lineAction = new TypeAction("Line", LINE, "Draw lines"));
JMenuItem it6 = new JMenuItem(rectangleAction = new TypeAction("Rectangle", RECTANGLE, "Draw
rectangles"));
JMenuItem it7 = new JMenuItem(circleAction = new TypeAction("Circle", CIRCLE, "Draw circles"));
JMenuItem it8 = new JMenuItem(curveAction = new  TypeAction("Curve", CURVE, "Draw curves"));
JMenuItem it9 = new JMenuItem(textAction = new TypeAction("Text", TEXT, "Draw text"));
          ...
// Construct the Element drop-down menu
elementMenu.add(it5);
elementMenu.add(it6);
elementMenu.add(it7);
elementMenu.add(it8);
elementMenu.add(it9);
elementMenu.addSeparator();

JMenu objectsMenu = new JMenu("Objects");
objectsMenu.addMouseListener(new java.awt.event.MouseAdapter() {
    ...
});
elementMenu.add(objectsMenu);
//Construct the Objects drop-down menu
JMenu chairsMenu = new JMenu("Chairs");  // Objects sub-menu
JMenu sofaMenu = new JMenu("Sofas");     // Objects sub-menu
JMenu bedsMenu = new JMenu("Beds");      // Objects sub-menu
```

```
JMenu tableMenu = new JMenu("Tables");   // Objects sub-menu
JMenu doorMenu = new JMenu("Doors");     // Objects sub-menu
JMenu windowMenu = new JMenu("Windows"); // Objects sub-menu
...
objectsMenu.add(chairsMenu);
objectsMenu.add(sofaMenu);
objectsMenu.add(bedsMenu);
objectsMenu.add(tableMenu);
objectsMenu.add(doorMenu);
objectsMenu.add(windowMenu);
JMenuItem it10 = new JMenuItem(objectAction = new ObjectsAction("Chair"));
JMenuItem it11 = new JMenuItem(objectAction = new  ObjectsAction("Sofa1"));
JMenuItem it12 = new JMenuItem(objectAction = new ObjectsAction("Sofa2"));
JMenuItem it13 = new JMenuItem(objectAction = new ObjectsAction("Bed1"));
JMenuItem it14 = new JMenuItem(objectAction = new ObjectsAction("Bed2"));
JMenuItem it15 = new JMenuItem(objectAction = new  ObjectsAction("Door"));
JMenuItem it16 = new JMenuItem(objectAction = new ObjectsAction("Window1"));
JMenuItem it17 = new JMenuItem(objectAction = new  ObjectsAction("Table1"));
JMenuItem it18 = new JMenuItem(objectAction = new ObjectsAction("Table2"));
JMenuItem it20 = new JMenuItem(objectAction = new ObjectsAction("Window2"));
...
chairsMenu.add(it10);
sofaMenu.add(it11);
sofaMenu.add(it12);
bedsMenu.add(it13);
bedsMenu.add(it14);
tableMenu.add(it17);
tableMenu.add(it18);
doorMenu.add(it15);
windowMenu.add(it16);
windowMenu.add(it20);
```

Κώδικας για το Draw menu στην κλάση SketchFrame

3.13.3 Μενού Help

Το μενού Help είναι ένα αντικείμενο τύπου **Jmenu**. Αφού το δηλώσουμε και το δημιουργήσουμε, του ορίζουμε σαν πλήκτρο συντόμευσης (όταν πατηθεί το κουμπί ALT μαζί με το κουμπί συντόμευσης ανοίγει το μενού) το γράμμα **H**.

Έπειτα, προσθέτουμε στο μενού ένα **mouseListener** για τα **mouseEntered** και **mouseExited** events, έτσι ώστε όταν το ποντίκι περνάει πάνω από το μενού να αλλάζει ο κέρσορας στο κλασικό δείκτη, και όταν βγαίνει από το μενού και περνάει πάνω από την περιοχή σχεδίασης να αλλάζει ο κέρσορας σε σταυρό.

Στη συνέχεια δημιουργούμε τα αντικείμενα που απαρτίζουν το μενού (**menu items**). Αφού τα δημιουργήσουμε, τους προσθέτουμε **actionListeners**, ώστε και πάλι να αλλάζει ο κέρσορας ανάλογα με τη θέση του. Μετά, προσθέτουμε τα menu items στο μενού.

Μέσα στην κλάση SketchFrame υπάρχει μία μέθοδος που διαχειρίζεται τα "κλικ" που γίνονται πάνω στο μενού (**actionPerformed**). Όταν γίνει κλικ πάνω στο About, τότε εμφανίζεται ένα παράθυρο με πληροφορίες για το πρόγραμμα.

```
JMenu helpMenu = new JMenu("Help");       // Create Help menu
helpMenu.setMnemonic('H');                // Create shortcut
    ...
  // Add the About item to the Help menu
aboutItem = new JMenuItem("About");
...
aboutItem.addActionListener(this);   // Listener is the frame
helpMenu.add(aboutItem);             // Add item to menu
```

Κώδικας για το About menu στην κλάση SketchFrame

```
// Handle menu action events
public void actionPerformed(ActionEvent e)  {
  Object source = e.getSource();
  if(source == aboutItem) {
    // Create about dialog with the menu item as parent
    JOptionPane.showMessageDialog(this,              // Parent
                  "SVG Editor\nProgramming: Dionisis Kladis " +
                  "(dionisis84@gmail.com)\nSupervisors:Athanasios Malamos\n" +
                  "                  Georgios Mamakis\n" +
                  "\nTechnological Educational Institute of Crete\n" +
                  "Department of Applied Informatics and Multimedia\n" +
                  "Multimedia Content Lab\n" +
                  "\nhttp://www.epp.teicrete.gr\n"+
                  "http://mclab.medialab.teicrete.gr/\n"+
                  "\nBased on \"Sketcher\" " +
                  "program by Ivor Horton.",         // Message
                  "About SVG Editor",                  // Title
                  JOptionPane.INFORMATION_MESSAGE);      // Message type
  }
}
```

Κώδικας για το actionPerformed στην κλάση SketchFrame

3.13.4 PopUp μενού

Το PopUp μενού είναι ένα αντικείμενο τύπου **JPopupMenu**. Το δηλώνουμε και το δημιουργούμε στο κάτω μέρος της κλάσης SketchFrame. Μετά, στον constructor προσθέτουμε τις επιλογές που θα έχει ο χρήστης μέσω του PopUp μενού.

3.14 Υλοποίηση Αποθήκευσης Σκίτσου

Για την αποθήκευση των σκίτσων που δημιουργεί ο χρήστης, έχουν υλοποιηθεί τρεις μέθοδοι μέσα στην κλάση **sketchFrame** με τις ονομασίες **saveXML(), saveXMLSketch()** και **writeXMLFile()**. Με τη βοήθεια τους το σκίτσο μετατρέπεται στην κατάλληλη μορφή με βάση το πρότυπο svg και αποθηκεύεται σε κάποιο σημείο του υπολογιστή του χρήστη,

σημείο που το επιλέγει ο ίδιος ο χρήστης.

3.14.1 saveXML()

Με αυτή τη μέθοδο εμφανίζεται στο χρήστη ένα παράθυρο, στο οποίο πρέπει να επιλέξει το μέρος όπου θα αποθηκευτεί το σκίτσο, όπως επίσης και το όνομα που θα έχει στο σκίτσο.

Έπειτα, αφού γίνουν τα παραπάνω, γίνεται έλεγχος για το αν υπάρχει ήδη το συγκεκριμένο αρχείο. Εάν υπάρχει, τότε εμφανίζεται μήνυμα ερώτησης για το εάν ο χρήστης επιθυμεί να γίνει αντικατάσταση ή όχι.

Μετά, δημιουργείται ένα string το οποίο περιέχει το πλήρες path του αρχείου προς αποθήκευση, όπως επίσης και το όνομα του αρχείο με την κατάληξη svg.

Τέλος, δημιουργείται ένα αντικείμενο τύπου **File** με τα παραπάνω χαρακτηριστικά και καλείται η μέθοδος saveXMLSketch.

```
    public void saveXML()
    {
        JFileChooser chooser = new JFileChooser(DEFAULT_DIRECTORY);
        chooser.setDialogType(chooser.SAVE_DIALOG);
        chooser.setDialogTitle("Save as SVG file");
        chooser.setApproveButtonText("Save");
        ExtensionFilter xmlFiles = new ExtensionFilter(".svg",
                                        "SVG files (*.svg)");
        chooser.addChoosableFileFilter(xmlFiles);            // Add the filter
        chooser.setFileFilter(xmlFiles);                     // and select it
        File file = null;
        if(chooser.showDialog(SketchFrame.this, null) == chooser.APPROVE_OPTION){
          file = chooser.getSelectedFile();
          if(file.exists()) {                        // Check file exists
            if(JOptionPane.NO_OPTION ==                // Overwrite warning
              JOptionPane.showConfirmDialog(SketchFrame.this,
                        file.getName()+" exists. Overwrite?",
                        "Confirm Save As",
                        JOptionPane.YES_NO_OPTION,
                        JOptionPane.WARNING_MESSAGE))
              return;                                  // No overwrite
          }
          String file_name = file.getPath();

          if(!(file_name.substring(file_name.length() - 4).equals(".svg")))
          {
             file_name = file_name + ".svg";
          }
          File end_file = new File(file_name);
          saveXMLSketch(end_file);
    }
}
```

Κώδικας για τη μέθοδο saveXML στην κλάση SketchFrame

3.14.2 saveXMLSketch(File outFile)

Με αυτή τη μέθοδο δημιουργείται μία ροή αρχείου (**FileOutputStream**) η οποία "δείχνει" προς το αρχείο που θα δεχτεί τα δεδομένα του σκίτσου. Έπειτα, δημιουργείται ένα κανάλι αρχείου (**FileChannel**) μέσω του οποίου θα διοχετευθούν τα δεδομένα του σκίτσου στο αρχείο. Τέλος, καλείται η μέθοδος writeXMLFile για να ολοκληρωθεί η διαδικασία αποθήκευσης.

Σαν όρισμα, η μέθοδος, δέχεται ένα αντικείμενο τύπου **File**, που αντιπροσωπεύει το αρχείο στο οποίο θα προστεθούν τα δεδομένα προς αποθήκευση.

```
private void saveXMLSketch(File outFile) {
  FileOutputStream outputFile = null;      // Stores an output stream reference
  try {
    outputFile = new FileOutputStream(outFile);    // Output stream for the file
    FileChannel outChannel = outputFile.getChannel(); // Channel for file stream
    writeXMLFile(theApp.getModel().createDocument(), outChannel);
  } catch(FileNotFoundException e) {
    e.printStackTrace(System.err);
    JOptionPane.showMessageDialog(SketchFrame.this,
                "Sketch file " + outFile.getAbsolutePath() + " not found.",
                "File Output Error",
                JOptionPane.ERROR_MESSAGE);
    return;                              // Serious error - return
  }
}
```

Κώδικας για τη μέθοδο saveXMLSketch στην κλάση SketchFrame

3.14.3 **writeXMLFile(Document doc, FileChannel channel)**

Με αυτή τη μέθοδο ξεκινά η διαδικασία εγγραφής των δεδομένων του σκίτσου σε μορφή svg μέσα στο αρχείο. Πρώτα γίνεται εγγραφή δύο συγκεκριμένων γραμμών xml που προσδιορίζουν των τύπο των δεδομένων που θα περιέχονται μέσα στο αρχείο. Αμέσως μετά γίνεται η μετατροπή των σχημάτων του σκίτσου σε svg με τη βοήθεια της μεθόδου **getDocumentNode()** (βλ. επόμενη υπο-ενότητα). Μόλις επιστραφούν τα δεδομένα σε μορφή string, αυτά προστίθενται στο τελικό αρχείο

Σαν ορίσματα, η μέθοδος, δέχεται ένα αντικείμενο τύπου **Document**, το οποίο αντιπροσωπεύει το σκίτσο με όλα τα σχήματα του και τις ιδιότητες τους, και ένα αντικείμενο τύπου **FileChannel** μέσω του οποίου θα εγγραφούν τα δεδομένα του σκίτσου μέσα στο αρχείο.

```
private void writeXMLFile(org.w3c.dom.Document doc, FileChannel channel) {
  StringBuffer xmlDoc = new StringBuffer("<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"no\"?
>");
  xmlDoc.append("\n<!DOCTYPE svg PUBLIC \"-//W3C//DTD SVG
1.1//EN\" \"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd\">");
```

```
  xmlDoc.append(getDocumentNode(doc.getDocumentElement(), ""));
  try {
    channel.write(ByteBuffer.wrap(xmlDoc.toString().getBytes("UTF-8")));
  } catch(UnsupportedEncodingException e) {
    System.out.println(e.getMessage());
  } catch(IOException e) {
    ...
  }
}
```

Κώδικας για τη μέθοδο saveXMLSketch στην κλάση SketchFrame

3.14.3 getDocumentNode(Node node, String indent)

Με τη βοήθεια αυτής της μεθόδου, αναλύονται τα στοιχεία του σκίτσου και μετατρέπονται σε γλώσσα svg. Ένα ένα τα στοιχεία μετατρέπονται σε ένα string το οποίο περιέχει όλες τις πληροφορίες, ιδιότητες κλπ.

Αφού γίνει αυτή η μετατροπή, επιστρέφεται ένα string στην προηγούμενη μέθοδο για να εισαχθεί μέσα στο αρχείο.

Σαν ορίσματα, η μέθοδος, δέχεται ένα αντικείμενο τύπου **Node**, το οποίο αντιπροσωπεύει ένα το σκίτσο που έχει σχεδιάσει ο χρήστης (μέσω αυτού θα αποθηκευτούν τα υπόλοιπα σχήματα), και ένα αντικείμενο τύπου **String** που θα χρησιμοποιηθεί σαν "εσοχή" για την μορφοποίηση του svg κώδικα.

```
private String getDocumentNode(Node node, String indent) {
  StringBuffer nodeStr = new StringBuffer().append(NEWLINE).append(indent);
  String nodeName = node.getNodeName();       // Get name of this node
  switch(node.getNodeType()) {
    case Node.ELEMENT_NODE:
    nodeStr.append(TAG_START);
    nodeStr.append(nodeName);
    if(node.hasAttributes()) {            // If the element has attributes...
      org.w3c.dom.NamedNodeMap attrs = node.getAttributes();   // ...get them
      for(int i = 0 ; i<attrs.getLength() ; i++) {
        org.w3c.dom.Attr attribute = (org.w3c.dom.Attr)attrs.item(i);

        nodeStr.append(' ').append(attribute.getName()).append('=')
                .append(QUOTE).append(attribute.getValue()).append(QUOTE);
      }
    }
    if(!node.hasChildNodes()) {        // Check for no children for this element
      nodeStr.append(EMPTY_TAG_END);   // There are none-close as empty element
      return nodeStr.toString();       // and return the completed element
    } else {                           // It has children
      nodeStr.append(TAG_END);                  // so close start-tag
      NodeList list = node.getChildNodes();       // Get the list of child nodes
      assert list.getLength()>0;                // There must be at least one
     // Append child nodes and their children...
      for(int i = 0 ; i<list.getLength() ; i++) {
        nodeStr.append(getDocumentNode(list.item(i), indent+" "));
```

```
      }
    }
    nodeStr.append(NEWLINE).append(indent).append(END_TAG_START)
                                .append(nodeName).append(TAG_END);
    break;
    case Node.TEXT_NODE:
    nodeStr.append(replaceQuotes(((org.w3c.dom.Text)node).getData()));
    break;
    default:
    assert false;
  }
  return nodeStr.toString();
}
```

Κώδικας για τη μέθοδο getDocumentNode στην κλάση SketchFrame

3.15 Υλοποίηση Φόρτωσης Αποθηκευμένου Σκίτσου

Για την φόρτωση των αποθηκευμένων σκίτσων που έχει δημιουργήσει ο χρήστης, έχουν υλοποιηθεί δύο μέθοδοι μέσα στην κλάση **sketchFrame** με τις ονομασίες **openXMLSketch()** και **createSketchModel()**. Με τη βοήθεια τους το αποθηκευμένο σε svg μορφή σκίτσο μετατρέπεται στη μορφή που απαιτεί η εφαρμογή και εμφανίζεται στο κεντρικό παράθυρο.

3.15.1 openXMLSketch(File xmlFile)

Με τη βοήθεια αυτής της μεθόδου γίνεται η ανάγνωση των svg δεδομένων από ένα αρχείο svg και η μετατροπή τους σε σκίτσο που εμφανίζεται στο κεντρικό παράθυρο της εφαρμογής.

Στην αρχή δημιουργούμε ένα αντικείμενο τύπου **DocumentBuilder** το οποίο θα αναγνώσει το svg αρχείο. Μετά γίνεται ένας έλεγχος για να διαπιστωθεί εάν έχουν γίνει αλλαγές στο τρέχων σκίτσο (εάν υπάρχει κάποιο). Έπειτα, προστίθεται το μοντέλο του σκίτσου στην εφαρμογή, το οποίο το επιστρέφει σαν αντικείμενο τύπου **SketchModel** η μέθοδος **createSketchModel** (βλ. επόμενη υπο-ενότητα). Σε αυτή τη φάση, το σκίτσο εμφανίζεται στο κεντρικό παράθυρο της εφαρμογής.

Σαν όρισμα, η μέθοδος, δέχεται ένα αντικείμενο τύπου **File** που αντιπροσωπεύει το αρχείο από το οποίο θα γίνει η ανάγνωση του σκίτσου.

```
private void openXMLSketch(File xmlFile) {
  ...
  try {
    DocumentBuilder builder = builderFactory.newDocumentBuilder();
    builder.setErrorHandler(new DOMErrorHandler());
    checkForSave();
    theApp.insertModel(createSketchModel(builder.parse(xmlFile)));
    filename = xmlFile.getName();              // Update the file name
```

```
    setTitle(frameTitle+xmlFile.getPath());    // Change the window title
    sketchChanged = false;                     // Status is unchanged

  } catch(ParserConfigurationException e) {
    ...
  } catch(org.xml.sax.SAXException e) {
     ...
  } catch(IOException e) {
    ...
  }
}
```

Κώδικας για τη μέθοδο openXMLSketch στην κλάση SketchFrame

3.15.2 createSketchModel(Document doc)

Αυτή η μέθοδος κατασκευάζει το μοντέλο του σκίτσου, παίρνοντας τις πληροφορίες που παρέχει το αποθηκευμένο αρχείο svg.

Αρχικά, αποθηκεύεται σε μία μεταβλητή το πρώτο node που υπάρχει μέσα στο svg. Έπειτα, με τη βοήθεια ενός βρόγχου, γίνεται ανάγνωση του κάθε node-παιδιού του αρχικού node. Ανάλογα με το όνομα που έχει κάθε παιδί, προστίθεται στο μοντέλο ένα νέο αντικείμενο κατάλληλου τύπου (π.χ. εάν το όνομα του παιδιού είναι line, τότε θα δημιουργηθεί ένα αντικείμενο τύπου Element.Line).

Αφού τελειώσει η διαδικασία δημιουργίας των κατάλληλων αντικειμένων, και αφού αυτά προστεθούν στο μοντέλο, επιστρέφεται το μοντέλο στην προηγούμενη μέθοδο.

Σαν όρισμα, η μέθοδος, δέχεται ένα αντικείμενο τύπου **Document** που αντιπροσωπεύει το αρχείο από το οποίο θα γίνει η ανάγνωση του σκίτσου.

```
private SketchModel createSketchModel(org.w3c.dom.Document doc) {
  SketchModel model = new SketchModel();      // The new model object
  org.w3c.dom.Node node = doc.getDocumentElement().getFirstChild();
  while (node != null) {
    assert node instanceof org.w3c.dom.Element; // Should all be Elements
    org.w3c.dom.Node node2 = node.getFirstChild();
    String name = ((org.w3c.dom.Element)node2).getTagName();
    if(name.equals("line")) {                   // Check for a line
       model.add(new Element.Line((org.w3c.dom.Element)node));
    } else if(name.equals("rect")) {      // Check for a rectangle
      model.add(new Element.Rectangle((org.w3c.dom.Element)node));
    } else if(name.equals("circle")) {        // Check for a circle
    model.add(new Element.Circle((org.w3c.dom.Element)node));
    } else if(name.equals("path")) {          // Check for a curve
    model.add(new Element.Curve((org.w3c.dom.Element)node));
    } else if(name.equals("text")) {          // Check for a text
      model.add(new Element.Text((org.w3c.dom.Element)node));
    }
    node = node.getNextSibling();                     // Next child node
  }
```

```
  return model;
}
```

Κώδικας για τη μέθοδο createSketchModel στην κλάση SketchFrame

3.16 Υλοποίηση παραθύρου πληροφοριών

Το παράθυρο πληροφοριών εμφανίζεται στο χρήστη, όταν αυτός επιλέξει από το μενού **Help** την επιλογή **About**. Το παράθυρο αυτό παρέχει πληροφορίες για την ίδια την εφαρμογή, όπως ποιος την έφτιαξε κλπ.

Μέσα στον κώδικα τώρα, αφού έγινε προσθήκη του μενού Help και των επιλογών του, έγινε χρήση της μεθόδου **actionPerformed** της κλάσης **sketchFrame**. Η κλάση sketchFrame περιέχει αυτή τη μέθοδο γιατί υλοποιεί το interface **ActionListener**, δηλαδή μπορεί να διαχειρίζεται τα events που παράγονται από το ποντίκι κλπ.

Μέσα στην actionPerformed, λοιπόν, ελέγχουμε την **προέλευση** του event. Εάν έχει έρθει από την επιλογή About του μενού Help, τότε εμφανίζουμε ένα **message dialog** (**JOptionPane**) που περιέχει τις επιθυμητές πληροφορίες. Η εφαρμογή θα παραμείνει αδρανής μέχρι ο χρήστης να κλείσει το παράθυρο με τις πληροφορίες.

```
public void actionPerformed(ActionEvent e)  {
  Object source = e.getSource();
  if(source == aboutItem) {
    // Create about dialog with the menu item as parent
    JOptionPane.showMessageDialog(this,                    // Parent
                    "SVG Editor\nProgramming: Dionisis Kladis " +
                    "(dionisis84@gmail.com)\nSupervisors: Athanasios Malamos\n" +
                    "                    Georgios Mamakis\n" +
                    "\nTechnological Educational Institute of Crete\n" +
                    "Department of Applied Informatics and Multimedia\n" +
                    "Multimedia Content Lab\n" +
                    "\nhttp://www.epp.teicrete.gr\n"+
                    "http://mclab.medialab.teicrete.gr/\n"+
                    "\nBased on \"Sketcher\" " +
                    "program by Ivor Horton.",          // Message
                    "About SVG Editor",                  // Title
                    JOptionPane.INFORMATION_MESSAGE);      // Message type
  }
}
```

Κώδικας για την εμφάνιση των πληροφοριών στην κλάση SketchFrame

3.17 Υλοποίηση αλλαγής εικονιδίου δείκτη ποντικιού

Για να διευκολύνεται ο χρήστης στη σχεδίαση, πρέπει κάθε φορά που ο κέρσορας του ποντικιού βρίσκεται μέσα στην περιοχή σχεδίασης να αλλάζει μορφή και να γίνεται σταυρός (crosshair). Με αυτό τον τρόπο η σχεδίαση ενός σχήματος είναι πολύ πιο εύκολη.

Η υλοποίηση έχει γίνει μέσα στην κλάση **SketchFrame**. Με τη δημιουργία του κεντρικού παραθύρου, αλλάζουμε τον κέρσορα σε σταυρό. Δεν θέλουμε, όμως, να παραμένει σταυρός όταν ο κέρσορας περνάει πάνω από τα διάφορα κουμπιά και μενού. Για αυτό τον λόγο, έχουμε προσθέσει **mouseListeners** σε όλα τα αντικείμενα που απαρτίζουν το κεντρικό παράθυρο (κουμπιά, μενού, μπάρες κλπ).

Με τη βοήθεια αυτών των mouseListeners διαχειριζόμαστε τα events που δημιουργούνται όταν το ποντίκι μπαίνει ή βγαίνει στην περιοχή που ορίζουν τα διάφορα αντικείμενα του παραθύρου (**mouseEntered, mouseExited**)

Κάθε φορά που το ποντίκι μπαίνει μέσα στην περιοχή ενός αντικειμένου (mouseEntered) αλλάζουμε τον κέρσορα στην κλασσική μορφή βέλους.

Κάθε φορά που το ποντίκι βγαίνει από την περιοχή ενός αντικειμένου (mouseExited) αλλάζουμε τον κέρσορα σε σταυρό.

```
menuBar.addMouseListener(new java.awt.event.MouseAdapter() {
    @Override
    public void mouseEntered(java.awt.event.MouseEvent evt) {
        setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
    }

    @Override
    public void mouseExited(java.awt.event.MouseEvent evt) {
        setCursor(new java.awt.Cursor(java.awt.Cursor.CROSSHAIR_CURSOR));
    }
});
```

Κώδικας για την αλλαγή του κέρσορα στην κλάση SketchFrame

3.18 Υλοποίηση των Actions

Τα actions είναι αντικείμενα μίας οποιασδήποτε κλάσης, η οποία υλοποιεί το interface **javax.swing.Action**. Αυτό το interface παρέχει μεθόδους που λειτουργούν πάνω σε ένα αντικείμενο Action. Επίσης, το interface ενσωματώνει τις λειτουργίες που παρέχει το interface **ActionListener**, οπότε λειτουργεί και με αυτό τον τρόπο.

Μερικά αντικείμενα του Swing έχουν μία μέθοδο add() που δέχεται σαν όρισμα ένα αντικείμενο Action. Όταν προσθέτουμε ένα action σε ένα αντικείμενο swing τότε δημιουργείται ένα "συστατικό" του δεύτερου το οποίο έχει τη σωστή μορφή. Δηλαδή, αν

για παράδειγμα προσθέσουμε ένα αντικείμενο action σε ένα JMenu, τότε στο μενού θα προστεθεί ένα JMenuItem. Αν προσθέσουμε ένα αντικείμενο action σε ένα JToolbar, τότε στο toolbar θα προστεθεί ένα JButton,

Με λίγα λόγια, υπάρχει η δυνατότητα να δημιουργηθούν διαφορετικά αντικείμενα, τα οποία θα επιτελούν την ίδια λειτουργία (αυτή που ορίζει το action). Αυτό είναι και το μεγάλο πλεονέκτημα των Actions, και για αυτό γίνεται η χρήση τους στην εφαρμογή μας.

Μέσα στην κλάση SketchFrame έχουμε δημιουργήσει 5 υποκλάσεις, η οποίες υλοποιούν με τη σειρά τους το interface Action. Κάθε μία από αυτές τις κλάσεις, επιτελεί μία διαφορετική λειτουργία. Αναλυτικότερα έχουμε:

- **FileAction**: Χρησιμοποιείται για τη δημιουργία νέου σκίτσου και την έξοδο από την εφαρμογή.
- **XMLExportAction**: Χρησιμοποιείται για την αποθήκευση των σκίτσων που έχει σχεδιάσει ο χρήστης σε svg.
- **XMLImportAction**: Χρησιμοποιείται για την ανάγνωση αρχείων svg, ώστε να μπορέσει ο χρήστης να τα επεξεργαστεί.
- **TypeAction**: Χρησιμοποιείται για τη δημιουργία των βασικών σχημάτων που έχει στη διάθεση του ο χρήστης (π.χ. γραμμές, τετράγωνα κλπ).
- **ObjectsAction**: Χρησιμοποιείται για τη δημιουργία των υπόλοιπων σχημάτων που έχει στη διάθεση του ο χρήστης (π.χ. καρέκλες κλπ).

3.18.1 FileAction

Η συγκεκριμένη κλάση χρησιμεύει στο να δημιουργούμε ένα νέο σκίτσο ή να κλείνουμε την εφαρμογή. Περιέχει 4 διαφορετικούς constructors με διαφορετικά ορίσματα ο καθένας φυσικά. Ανάλογα με τον constructor που θα χρησιμοποιηθεί, μπορείτε να δώσετε σαν ορίσματα ένα όνομα για το action, μία συντόμευση πληκτρολογίου και κάποιο βοηθητικό μήνυμα που θα εμφανίζεται όταν ο χρήστης αφήσει για λίγη ώρα το ποντίκι πάνω από ένα αντικείμενο.

Επίσης, περιέχει και μία μέθοδο actionPerformed η οποία είναι και ο διαχειριστής των events που θα παράγονται από τα αντικείμενα. Ανάλογα με το όνομα που έχει το action (New, Exit) εκτελείται και η ανάλογη ενέργεια (δημιουργία νέου σκίτσου ή έξοδος). Και στις δύο περιπτώσεις, γίνεται έλεγχος για το εάν έχει γίνει κάποια αλλαγή στο σκίτσο, έτσι ώστε να εμφανιστεί στον χρήστη μήνυμα για την αποθήκευση του. Για τον έλεγχο αυτό καλείται η μέθοδος **checkForSave()**.

Η κλάση αυτή εφαρμόζεται στα κουμπιά και τα μενού στα οποία θέλουμε τη λειτουργικότητα δημιουργίας νέου σκίτσου και εξόδου από την εφαρμογή.

```
public void actionPerformed(ActionEvent e) {
  String name = (String)getValue(NAME);
  if(name.equals(newAction.getValue(NAME))) {
```

```
    checkForSave();
    theApp.insertModel(new SketchModel());
    modelFile = null;
    filename = DEFAULT_FILENAME;
    setTitle(frameTitle);
    sketchChanged = false;
   }
   else if(name.equals(closeAction.getValue(NAME))) {
    checkForSave();
    System.exit(0);
   }
  }
```

Κώδικας της μεθόδου actionPerformed στην κλάση FileAction

3.18.2 XMLExportAction

Η συγκεκριμένη κλάση χρησιμεύει στο να αποθηκεύουμε ένα σκίτσο στον υπολογιστή. Περιέχει 2 διαφορετικούς constructors με διαφορετικά ορίσματα ο καθένας φυσικά. Ανάλογα με τον constructor που θα χρησιμοποιηθεί, μπορείτε να δώσετε σαν ορίσματα ένα όνομα για το action, μία συντόμευση πληκτρολογίου και κάποιο βοηθητικό μήνυμα που θα εμφανίζεται όταν ο χρήστης αφήσει για λίγη ώρα το ποντίκι πάνω από ένα αντικείμενο.

Επίσης, περιέχει και μία μέθοδο actionPerformed η οποία είναι και ο διαχειριστής των events που θα παράγονται από τα αντικείμενα. Η μόνη ενέργεια που γίνεται εδώ είναι η κλήση της μεθόδου **saveXML()** (για περισσότερες πληροφορίες δείτε την ενότητα **2.8**), μέσω της οποίας πραγματοποιείται η αποθήκευση.

Η κλάση αυτή εφαρμόζεται στα κουμπιά και τα μενού στα οποία θέλουμε τη λειτουργικότητα της αποθήκευσης του σκίτσου που έχει φτιάξει ο χρήστης σε κάποιο σημείο στον υπολογιστή του.

```
public void actionPerformed(ActionEvent e) {
     saveXML();
  }
```

Κώδικας της μεθόδου actionPerformed στην κλάση XMLExportAction

3.18.3 XMLImportAction

Η συγκεκριμένη κλάση χρησιμεύει στο να διαβάζουμε ένα σκίτσο που βρίσκεται αποθηκευμένο στον υπολογιστή. Περιέχει 2 διαφορετικούς constructors

με διαφορετικά ορίσματα ο καθένας φυσικά. Ανάλογα με τον constructor που θα χρησιμοποιηθεί, μπορείτε να δώσετε σαν ορίσματα ένα όνομα για το action, μία συντόμευση πληκτρολογίου και κάποιο βοηθητικό μήνυμα που θα εμφανίζεται όταν ο χρήστης αφήσει για λίγη ώρα το ποντίκι πάνω από ένα αντικείμενο.

Επίσης, περιέχει και μία μέθοδο actionPerformed η οποία είναι και ο διαχειριστής των events που θα παράγονται από τα αντικείμενα. Εδώ εμφανίζουμε στο χρήστη ένα παράθυρο επιλογής αρχείων, έτσι ώστε να διαλέξει το svg σκίτσο από το μέρος που το έχει αποθηκεύσει. Εάν όλα γίνουν σωστά (από τη μεριά του χρήστη) τότε καλείται η μέθοδος **openXMLSketch()** (για περισσότερες πληροφορίες δείτε την ενότητα **2.9**), η οποία και θα διαβάσει το αρχείο και θα το εμφανίσει στο κεντρικό παράθυρο.

Η κλάση αυτή εφαρμόζεται στα κουμπιά και τα μενού στα οποία θέλουμε τη λειτουργικότητα φόρτωσης ενός ήδη αποθηκευμένου σκίτσου στην εφαρμογή, από κάποιο σημείο του υπολογιστή του χρήστη.

```
public void actionPerformed(ActionEvent e) {
  JFileChooser chooser = new JFileChooser(DEFAULT_DIRECTORY);
  chooser.setDialogTitle("Open SVG file");
  chooser.setApproveButtonText("Open");
  ExtensionFilter xmlFiles = new ExtensionFilter(".svg", "SVG files (*.svg)");
  chooser.addChoosableFileFilter(xmlFiles);
  chooser.setFileFilter(xmlFiles);
  if(chooser.showDialog(SketchFrame.this, null) == chooser.APPROVE_OPTION)
    openXMLSketch(chooser.getSelectedFile());
}
```

Κώδικας της μεθόδου actionPerformed στην κλάση XMLExportAction

3.18.4 TypeAction

Η συγκεκριμένη κλάση χρησιμεύει στο να προσδιορίζουμε τι τύπου είναι κάθε στοιχείο ενός σκίτσου (π.χ. γραμμή, τετράγωνο κλπ). Περιέχει 2 διαφορετικούς constructors με διαφορετικά ορίσματα ο καθένας φυσικά. Ανάλογα με τον constructor που θα χρησιμοποιηθεί, μπορείτε να δώσετε σαν ορίσματα ένα όνομα για το action, μιένα τύπο στοιχείου και κάποιο βοηθητικό μήνυμα που θα εμφανίζεται όταν ο χρήστης αφήσει για λίγη ώρα το ποντίκι πάνω από ένα αντικείμενο.

Επίσης, περιέχει και μία μέθοδο actionPerformed η οποία είναι και ο διαχειριστής των events που θα παράγονται από τα αντικείμενα. Εδώ δίνουμε στη μεταβλητή **elementType** ποιό σχήμα είναι ενεργό κάθε φορά (δηλαδή το ποντίκι είναι πάνω από το σχήμα). Η μεταβλητή elementType είναι ορισμένη μέσα στην κλάση sketchFrame και μας δείχνει το ενεργό καθε φορά σχήμα. Επίσης, μέσα στη μέθοδο καλείται η μέθοδος **setTypePane** (η μέθοδος περιλαμβάνεται μέσα στο αντικείμενο **statusBar**), η οποία εμφανίζει στο status bar στο κάτω μέρος του

κεντρικού παραθύρου το επιλεγμένο σχήμα.

```
public void actionPerformed(ActionEvent e) {
  elementType = typeID;
  statusBar.setTypePane(typeID);
}
```

Κώδικας της μεθόδου actionPerformed στην κλάση TypeAction

3.18.5 ObjectsAction

Η συγκεκριμένη κλάση χρησιμεύει στο να προσθέτουμε στο σκίτσο κάποιο αντικείμενο, όπως είναι μία καρέκλα. Περιέχει ένα μόνο constructor, ο οποίος δέχεται σαν όρισμα το όνομα του αντικειμένου (π.χ. chair).

Επίσης, περιέχει και μία μέθοδο actionPerformed η οποία είναι και ο διαχειριστής των events που θα παράγονται από τα αντικείμενα. Εδώ ανάλογα με το όνομα που έχει το κάθε αντικείμενο, η μέθοδος διαβάζει το κατάλληλο svg (τα αντικείμενα αυτά είναι αποθηκευμένα μαζί με την εφαρμογή σαν ξεχωριστά αρχεία svg) και το εμφανίζει στο κεντρικό παράθυρο μαζί με το υπόλοιπο σκίτσο.

Η κλάση αυτή εφαρμόζεται στα κουμπιά και τα μενού στα οποία θέλουμε τη λειτουργικότητα εισαγωγής αντικειμένων όπως καρέκλες κλπ μέσα στο σκίτσο.

```
public void actionPerformed(ActionEvent ee){
                        ...
  String txt = ee.getSource().toString();
  txt = txt.substring(txt.indexOf("text=") + 5);
  try
  {
     txt = txt.substring(0, txt.indexOf(","));
  }
  catch(Exception eee)
  {
     txt = txt.substring(0, txt.indexOf("]"));
  }
  xmlfile = "./Items/" + txt + ".svg";
  try{
    DocumentBuilder builder = builderFactory.newDocumentBuilder();
    builder.setErrorHandler(new DOMErrorHandler());
    theApp.insertModel_2(createSketchModel(builder.parse(xmlfile)));
  }
  catch(ParserConfigurationException e) {
                        ...
}
```

Κώδικας της μεθόδου actionPerformed στην κλάση ObjectsAction

4 Επίλογος

4.1 Σύνοψη και συμπεράσματα

Στην παρούσα πτυχιακή εργασία έγινε παρουσίαση των γραφικών δύο διαστάσεων, και συγκεκριμένα των διανυσματικών γραφικών. Έγιναν σαφείς οι διαφορές των διάφορων τύπων γραφικών και η πτυχιακή εργασία επικεντρώθηκε στα διανυσματικά γραφικά SVG.

Το SVG (Scalable Vector Graphics) είναι ένα πρότυπο διανυσματικών γραφικών, το οποίο μας δίνει πολλές δυνατότητες, μιας και για η δομή του βασίζεται στη γλώσσα XML. Στην πτυχιακή εργασία αυτή, έγινε παρουσίαση των βασικών δομικών στοιχείων του πρότυπου SVG, όπως επίσης παρουσιάστηκε η γενική μορφή των αρχείων SVG με παραδείγματα.

Έπειτα, έγινε ανάλυση στο πως έγινε η ανάπτυξη της εφαρμογής, η οποία ήταν και το κύριο αντικείμενο της πτυχιακής εργασίας. Παρουσιάστηκαν όλα τα βασικά μέρη της, όπως επίσης και ο κώδικας που αποτελεί κάθε ένα από αυτά, έτσι ώστε ο αναγνώστης να έχει μία εικόνα για το τι ακριβώς συμβαίνει μέσα στην εφαρμογή.

4.2 Μελλοντικές επεκτάσεις

Η εφαρμογή που αναπτύχθηκε για τις ανάγκες της συγκεκριμένης πτυχιακής εργασίας έχει τη δυνατότητα να αναβαθμιστεί περαιτέρω και να επεκταθεί, έτσι ώστε να προσφέρει νέες λειτουργίες και δυνατότητες.

Τα διανυσματικά γραφικά SVG, επειδή ακριβώς είναι σε μία μορφή εύκολα επεξεργάσιμη, δηλαδή XML, έχουν τη δυνατότητα να μετατρέπονται με σχετικά εύκολο

τρόπο σε άλλες μορφές. Πιο συγκεκριμένα, μία πολύ καλή εφαρμογή αυτής της μετατροπής είναι η μεταφορά των διανυσματικών γραφικών στις τρεις διαστάσεις.

Το X3D είναι ένα πρότυπο τρισδιάστατης απεικόνισης, το οποίο βασίζεται και αυτό στη γλώσσα XML. Επομένως, είναι σχετικά εύκολο να δημιουργηθεί μία εφαρμογή, η οποία θα κάνει τις απαραίτητες μετατροπές και θα μεταφέρει τα σχέδια SVG σε X3D. Το τελικό αποτέλεσμα θα είναι η τρισδιάστατη απεικόνιση των αντικειμένων που έχει σχεδιάσει ο χρήστης σαν SVG. Τέλος, θα είναι δυνατό να αποθηκεύσουμε αυτό το τρισδιάστατο μοντέλο, ώστε να μπορούμε ανά πάση στιγμή να το ανοίξουμε και να το δούμε στην οθόνη.

Εάν πάμε ένα βήμα παραπάνω, η εφαρμογή μπορεί να γίνει μέρος μίας σουίτας εφαρμογών, αντικείμενο των οποίων θα είναι ο τομέας της εσωτερικής σχεδίασης χώρων. Με αυτό κατά νου, θα έχουμε τη δυνατότητα να σχεδιάζουμε την κάτοψη ενός χώρου (χρησιμοποιώντας διανυσματικά γραφικά SVG), θα επιλέγουμε το στυλ διακόσμησης που προτιμάμε, ίσως θα επιλέγουμε και τα αγαπημένα μας χρώματα, και με βάση αυτές τις επιλογές και το σχέδιο του χώρου θα μπορούμε να βλέπουμε την τρισδιάστατη αναπαράσταση του εν λόγω χώρου. Έτσι θα έχουμε τη δυνατότητα να εξετάζουμε το κατά πόσο ο χώρος που σχεδιάσαμε είναι λειτουργικός και όμορφος.

Εφόσον μπορούμε να αποθηκεύσουμε το τρισδιάστατο μοντέλο του χώρου που σχεδιάσαμε, έχουμε ακόμα μία δυνατότητα. Στην περίπτωση που δεν μας αρέσει ο χώρος και θέλουμε να αλλάξουμε ή διαγράψουμε κάποια στοιχεία από αυτόν, μέσω μίας άλλης εφαρμογής, μπορούμε να μετατρέψουμε το X3D μοντέλο (το οποίο, όπως είπαμε και πιο πάνω, βασίζεται στο XML) ξανά σε διανυσματικά γραφικά SVG. Γίνεται, δηλαδή, η αντίστροφη διαδικασία με πριν, έτσι ώστε από τρισδιάστατη αναπαράσταση να πάρουμε την αντίστοιχη σε δύο διαστάσεις. Επομένως, μπορούμε να κάνουμε αλλαγές στο αρχικό σχέδιο του χώρου, και αφού είμαστε ευχαριστημένοι να το μετατρέψουμε ξανά σε τρισδιάστατο μοντέλο και να δούμε πως θα φαίνεται στην πραγματικότητα.

Βιβλιογραφία

[1]	Cornell University, Program of Computer Graphics, "What is Computer Graphics?", www.graphics.cornell.edu/online/tutorials
[2]	Wikipedia, the free encyclopedia, "Vector Graphics", en.wikipedia.org/wiki/Vector_graphics
[3]	Wikipedia, the free encyclopedia, "Scalable Vector Graphics", en.wikipedia.org/wiki/SVG
[4]	Wikipedia, the free encyclopedia, "2D Computer graphics", en.wikipedia.org/wiki/2D_computer_graphics
[5]	Φώτης Λαζαρίνης, "Τεχνολογίες Πολυμέσων. Θεωρία, Υλικό, Λογισμικό", Κλειδάριθμος, ISBN 978-960-461-034-1
[6]	Σ.Ν. Δημητριάδης, Α.Σ. Πομπόρτσης, Ε.Γ. Τριανταφύλλου, "Τεχνολογία Πολυμέσων, Θεωρία και Πράξη", Εκδόσεις Τζιόλα, ISBN 960-418-025-8
[7]	World Wide Web Consortium, "Scalable Vector Graphics (SVG) 1.1 Specification", www.w3.org/TR/SVG11
[8]	Wikipedia, the free encyclopedia, "Color depth", en.wikipedia.org/wiki/Color_depth
[9]	Wikipedia, the free encyclopedia, "Halftone", en.wikipedia.org/wiki/Halftone
[10]	Kristamo "Karuna51", "Flower -pansy", karuna51.deviantart.com
[11]	Wikipedia, the free encyclopedia, "Interior Design", en.wikipedia.org/wiki/Interior_design
[12]	National Council for Interior Design Qualification, "Differences Between Interior Design & Decorating", http://www.ncidq.org/AboutUs/AboutInteriorDesign/DifferencesBetweenInteriorDesignDecorating.aspx
[13]	Wikipedia, the free encyclopedia, "Computer – Aided design", en.wikipedia.org/wiki/Computer-aided_design
[14]	Ivor Horton, "Ivor Horton's Beginning Java 2 JDK 5 Edition", Wrox, ISBN 0-7845-6874-4
[15]	Wikipedia, the free encyclopedia, "Swing (Java)",

	en.wikipedia.org/wiki/Swing_(Java)
[16]	Wikipedia, the free encyclopedia, “Widget toolkit”, http://en.wikipedia.org/wiki/Widget_toolkit

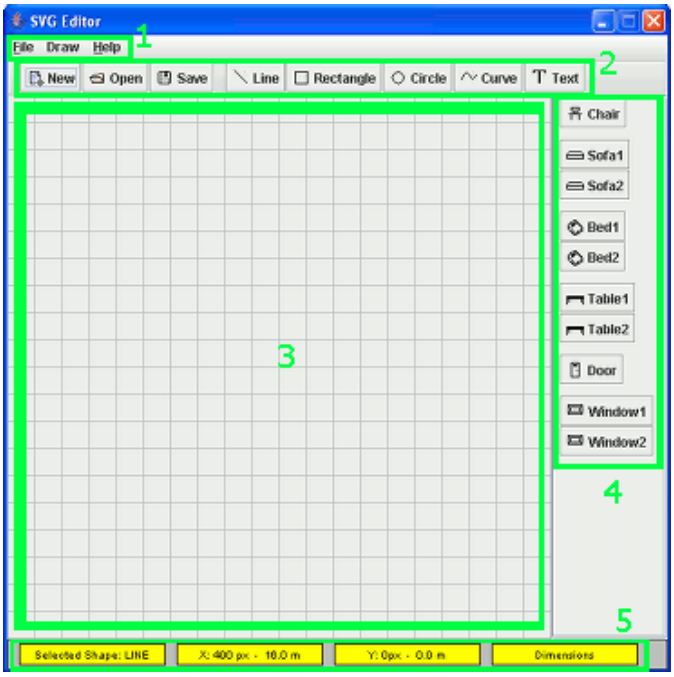
Οδηγίες χρήσης της εφαρμογής

5.1 Εκκίνηση της εφαρμογής

Για να εκκινήσετε την εφαρμογή σχεδιάσης, κάντε διπλό κλικ στο αρχείο svgEditor_final.jar. Για να τρέξει η εφαρμογή, πρέπει να υπάρχει εγκατεστημένο στο σύστημα σας το Java Runtime Environment (JRE). Αν δεν είναι εγκατεστημένο, μεταβείτε στην ακόλουθη διεύθυνση ίντερνετ για να το κατεβάσετε.

http://www.java.com/en/download/manual.jsp

5.2 Το γραφικό περιβάλλον



Εικόνα 18: Η εφαρμογή SVG Editor

Το γραφικό περιβάλλον της εφαρμογής έχει σχεδιαστεί όσο το δυνατόν πιο απλό και κατανοητό για τον απλό χρήστη. Αποτελείται από τα παρακάτω στοιχεία:

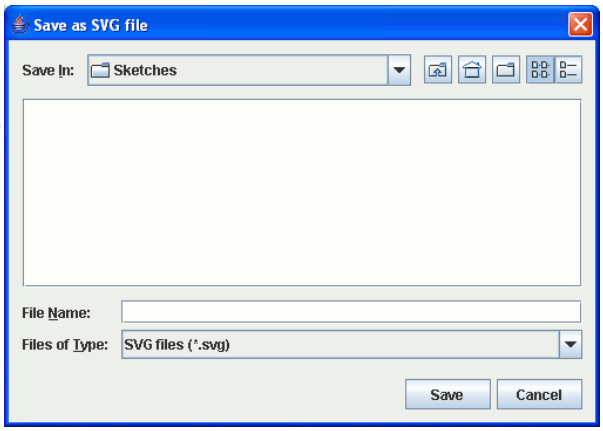
1. **Κεντρικό μενού.** Από εδώ μπορείτε να ανοίξετε, αποθηκεύσετε, δημιουργήσετε σκίτσα και να σχεδιάσετε σχήματα.
2. **Μπάρα εργαλείων.** Με αυτά τα κουμπιά μπορείτε να κάνετε ακριβώς τα ίδια πράγματα με το κεντρικό μένου.
3. **Περιοχή σχεδίασης.** Εδώ είναι η περιοχή στην οποία σχεδιάζετε τα σκίτσα.
4. **Βοηθητική μπάρα.** Εδώ, παρέχονται διάφορα έτοιμα σχήματα τα οποία αναπαρηστούν κάποια αντικείμενα, όπως τραπέζια, καρέκλες κλπ.
5. **Μπάρα πληροφοριών.** Η μπάρα πληροφοριών παρέχει πληροφορίες σχετικά με το επιλεγμένο σχήμα, τη θέση του δείκτη του ποντικιού στην περιοχή σχεδίασης και τις διαστάσεις του σχήματος που σχεδιάζετε.

5.3 Σχεδίαση

Για να σχεδιάσετε κάποιο σχήμα στην περιοχή σχεδίασης, απλά κάντε κλικ στο αντίστοιχο κουμπί στη μπάρα εργαλίων ή επιλέξτε το κατάλληλο σχήμα από το μενού "**Draw**". Έπειτα, κάντε κλικ στην επιθυμητή θέση στην περιόχη σχεδίασης, και κρατώντας πατημένο το κουμπί του ποντικιού, σύρετε το ποντίκι μέχρι εκεί που θέλετε. Μετά, αφήστε το κουμπί του ποντικιού για να σχεδιαστεί το σχήμα στη θέση του.

5.4 Αποθήκευση σκίτσων

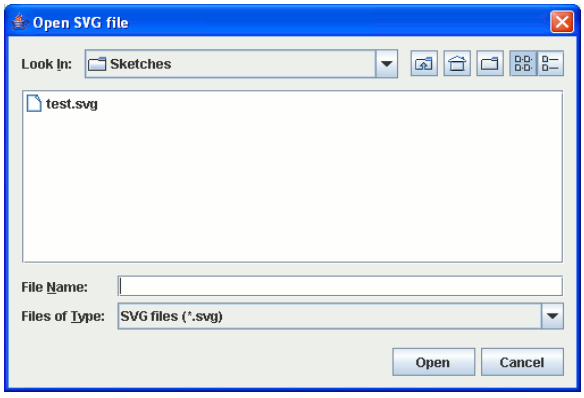
Για να αποθηκεύσετε ένα σκίτσο που έχετε σχεδιάσει, κάντε κλικ στο κουμπί "**Save**" ή επιλέξτε "**Save SVG**" από το μενού "**File**". Αμέσως θα εμφανιστεί το παράθυρο της διπλανής εικόνας. Δώστε ένα όνομα στο σκίτσο σας και μετά πατήστε το κουμπί "**Save**". Τα σκίτσα αποθηκεύονται στο φάκελο "**Sketches**" που βρίσκεται στον ίδιο φάκελο με το svgEditor_Final.jar.



Εικόνα 19: Αποθήκευση σκίτσων

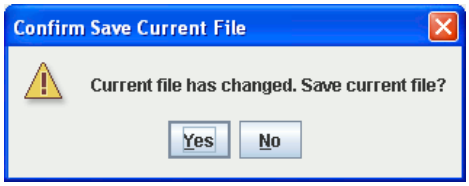
5.5 Άνοιγμα σκίτσων

Για να ανοίξετε ένα ήδη αποθηκευμένο σκίτσο, κάντε κλικ στο κουμπί "**Open**" της μπάρας εργαλείων ή επιλέξτε "**Open SVG**" από το μενού "**File**". Αμέσως θα εμφανιστεί το παράθυρο της διπλανής εικόνας. Επιλέξτε από εκεί το σκίτσο που επιθυμείτε και πατήστε το κουμπί "**Open**".



Εικόνα 20: Άνοιγμα σκίτσων

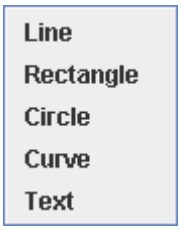
5.6 Δημιουργία νέου σκίτσου



Εικόνα 21: Δημιουργία νέου σκίτσου

Για να δημιουργήσετε ένα νέο σκίτσο, απλά κάντε κλικ στο κουμπί "**New**" της μπάρας εργαλείων ή επιλέξτε "**New**" από το μενού "**File**". Αν υπάρχει κάποιο σκίτσο στη περιοχή σχεδίασης, τότε θα εμφανιστεί ένα παράθυρο επιβεβαίωσης, όπως αυτό της διπλανής εικόνας. Αν θέλετε να σώσετε το τρέχον σκίτσο, πατήστε "**Yes**" και θα εμφανιστεί το παράθυρο αποθήκευσης. Σε αντίθετη περίπτωση, πατήστε "**No**".

5.7 Popup μενού

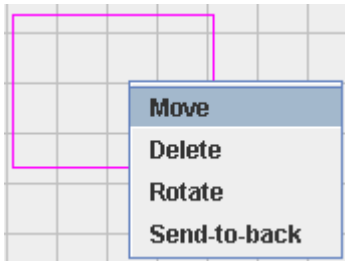


Εικόνα 22: Popup μενού

4.7.1 Μενού σχεδίασης

Όποτε κάνετε δεξί κλικ στη περιοχή σχεδίασης, ένα popup μενού εμφανίζεται, το οποίο σας δίνει τη δυνατότητα να επιλέξετε ένα άλλο σχήμα για να σχεδιάσετε. Το μενού είναι το διπλανό. Το μενού έχει ακριβώς τις ίδιες επιλογές με το μενού "**Draw**" και με τη μπάρα εργαλείων.

4.7.2 Μενού διαχείρισης σχήματος



Εικόνα 23: Μενού διαχείρισης σχήματος

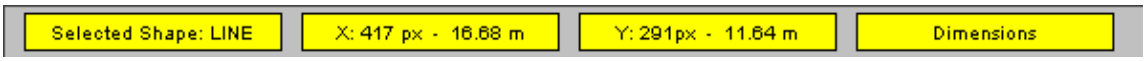
Εάν ο δείκτης του ποντικιού βρίσκεται πάνω από ένα ήδη σχεδιασμένο σχήμα, τότε αυτό το σχήμα αλλάζει χρώμα (γίνεται μωβ) και τότε, κάνοντας δεξί κλικ, εμφανίζεται το popup μενού της διπλανής εικόνας. Με αυτό το μενού μπορείτε να μετακινήσετε το επιλεγμένο σχήμα (**Move**), να διαγράψετε το επιλεγμένο σχήμα (**Delete**), να περιστρέψετε το επιλεγμένο σχήμα (**Rotate**) και να στείλετε το επιλεγμένο σχήμα πιο πίσω από τα άλλα σχήματα (**Send-to-back**).

5.8 Βοηθητική Μπάρα

Εικόνα 24: Βοηθητική μπάρα

Η βοηθητική μπάρα σας παρέχει διάφορα έτοιμα αντικείμενα, τα οποία συμβολίζουν τραπέζια, πόρτες κλπ. Για να τοποθετήσετε ένα τέτοιο αντικείμενο στην περιοχή σχεδίασης, απλά κάντε κλικ στο αντίστοιχο κουμπί της βοηθητικής μπάρας. Το αντικείμενο θα τοποθετηθεί στην περιοχή σχεδίασης σε ένα προκαθορισμένο σημείο. Μετά, εσείς μπορείτε να το μετακινήσετε ή γενικά να το διαχειριστείτε όπως κάθε άλλο σχήμα, μέσω του popup μενού διαχείρισης (βλ. Ενότητα 4.7.2).

5.9 Μπάρα πληροφοριών



Η μπάρα πληροφοριών παρέχει διάφορες ενδείξεις σχετικά με το σκίτσο που δημιουργείτε. Αναλυτικά, παρέχονται οι παρακάτω ενδείξεις.

1. Η πρώτη ένδειξη δείχνει **πιο σχήμα** έχετε επιλέξει για να σχεδιάσετε.
2. Η επόμενη ένδειξη δείχνει τη **θέση του δείκτη** του ποντικιού στον **οριζόντιο άξονα**, σε pixels και σε μέτρα. Κάθε τετράγωνο της περιοχής σχεδίασης αντιστοιχεί σε ένα τετραγωνικό μέτρο του αληθινού κόσμου.
3. Η τρίτη ένδειξη δείχνει τη **θέση του δείκτη** του ποντικιού στον **κατακόρυφο άξονα**, σε pixels και σε μέτρα.
4. Η τελευταία ένδειξη δείχνει τις **διαστάσεις του σχήματος** που σχεδιάζετε ή του σχήματος που βρίσκεται κάτω από το δείκτη του ποντικιού. Ανάλογα με το τι σχήμα σχεδιάζετε, εμφανίζονται και διαφορετικές διαστάσεις.
   - **Γραμμή**: εμφανίζει το μήκος της γραμμής.
   - **Ορθογώνιο**: εμφανίζει το ύψος και το πλάτος.
   - **Κύκλος**: εμφανίζει την ακτίνα του κύκλου.
   - **Καμπύλη**: εμφανίζει την απόσταση του πρώτου σημείου της καμπύλης με το τελευταίο.
   - **Κείμενο**: εμφανίζει πόσους χαρακτήρες περιέχει το κείμενο.