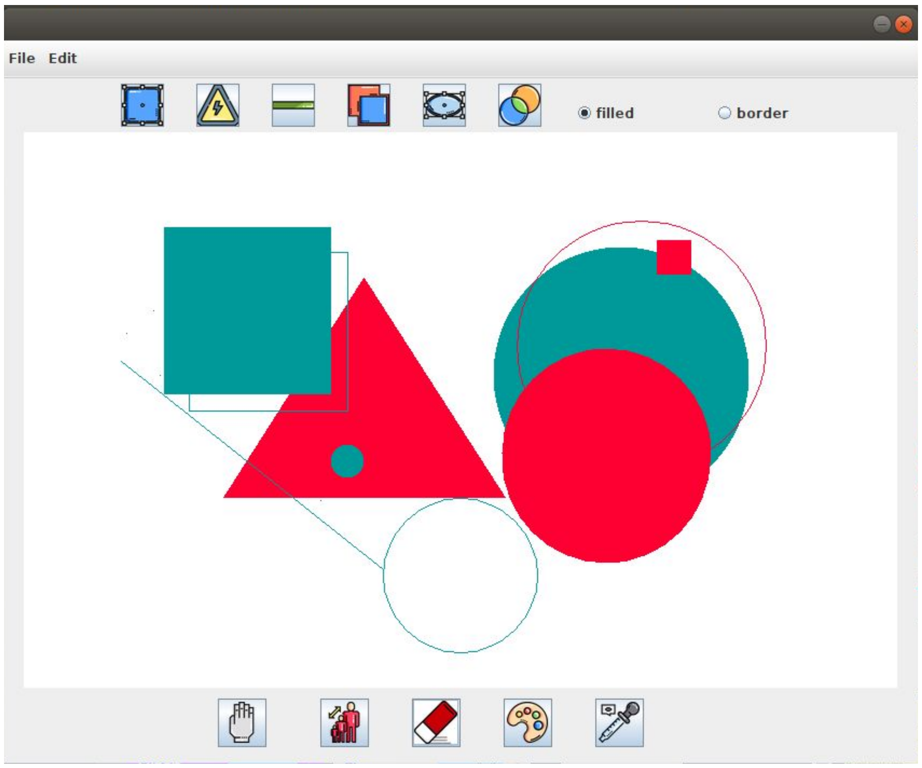
Final Report

Paint mini-project Java language

February 28, 2019



by **Amina Hajiyeva, Atifa Aghazada, Nafila Amirli** (Computer Science students License 2)

Under Supervision of Cecilia Zanni-Merk and Nuraddin Sadili

Abstract

In a graphical system, a windowing toolkit is typically responsible for providing a framework to make a graphical user interface to render the right bits to the monitor at the proper time.

This framework is provided by both the AWT (abstract windowing toolkit) and Swing. But the APIs that implement it are not well understood by some developers — a problem that has led to programs not performing as well as they could.

In this report we explain details of our Paint project that contributed by AWT and Swing, Object Oriented Programming in java. The project enables the user to Draw any geometrical shape and any graphic with any color using a mouse in coordinates and materializes Clear, Save (to any file you want), Load, Erase, Filling the figures as well. Here as the final result of our development project, the user can still use hand (mouse) as an easy way to draw Rectangle, Ellips, Line, Circle, Square, and integrated graphics such as a home, house, snowman, and so on.

Contents

General introduction

Java - Java is a widely used programming language expressly designed for use in the distributed environment of the internet. It’s a general-purpose computer programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible.

Object-oriented Programming - refers to languages that use objects in programming. It is based on hierarchy of classes where class itself is a structure that defines the data and methods to work on data. When it comes to objects, they are the instances which have data members and methods as defined by the class for that instance. As an OOP concept, Inheritance helps objects work together while defining relations among classes.

AWT (Abstract Window Toolkit) - a platform dependent API for creating Graphical User Interface (GUI) for java programs. It is *Java*'s original platform-dependent windowing, graphics, and user-interface widget toolkit, preceding Swing.

Swing - as much as AWT, it is a part of Java Foundation Classes (JFC) that is used to create window-based applications. Swing is built on the top of AWT API and entirely written in java. Java Swing provides platform-independent and lightweight components. The javax.swing package provides classes for java swing API such as *JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser* etc.

Graphical User Interface - A GUI program offers a much richer type of user interface, where the user uses a mouse and keyboard to interact with GUI components such as windows, menus, buttons, check boxes, text input boxes, scroll bars, and so on. A Swing GUI cannot exist without a top level container like (JWindow, Window, JFrame Frame or Applet), while it may exist without JPanels. *JFrame* extends *Component* and *Container*. It is a top level container used to represent the minimum requirements for a window. This includes Borders, resizability,, title bar and event handlers for various Events like windowClose, windowOpened etc. *JPanel* extends *Component*, *Container* and *JComponent.* It is a generic class used to group other Components together. It is useful when working with LayoutManagers e.g. GridLayout f.i adding components to different JPanels which will then be added to the JFrame to create the gui. It will be more manageable in terms of Layout and re-usability. And JPanel is also useful for when painting/drawing in Swing, you would override paintComponent() and have the full joys of double buffering.

Event listeners - GUIs are largely event-driven; that is, the program waits for events that are created by the behavior of the user. When an event occurs, the system reacts by executing an event-handling process. In order to program the actions of the Interface, you need to write event-handling methods to respond to events that are of interest to you.The most common technique for handling events is to use event listeners.   The listener is an entity that contains one or more methods of event handling. When an event is detected by another object, such as a button or menu, the listener object is notified and responds by running the correct event-handling process. An event is detected or is generated by an object Another object, the listener, is responsible for reacting to the event The event itself is actually a third object which carries details about the type of event, when it happened, and so on. The separation of duties makes it easier to coordinate broad programmes. As an example, consider the Undo or Redo button in our sample program. When the user clicks the button, an event is generated.

<u>Some examples of event types:</u>

- *ActionEvent* - clicking a button, pressing return on a text field
- *ItemEvent* - clicking a check box, selecting an item
- *WindowEvent* - Closing a window, opening a window
- *ContainerEvent* - component added to a container
- *ComponentEvent* - resizing a component, hiding a component
- *TextEvent* - changing a text value
- *MouseEvent* - clicking the mouse, dragging the mouse
- *KeyEvent* - pressing a key on the keyboard.

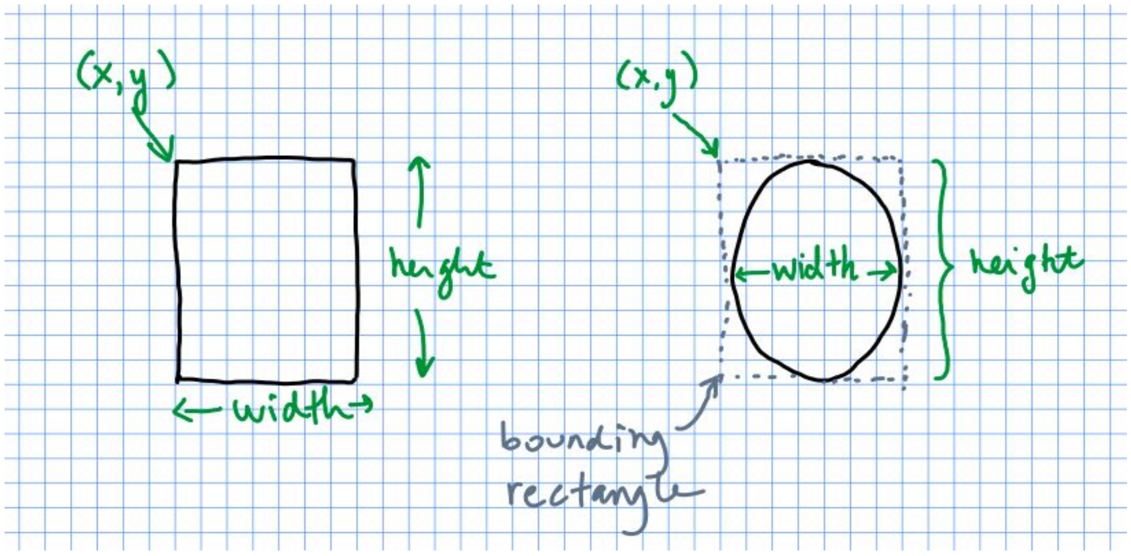
Painting/Graphics

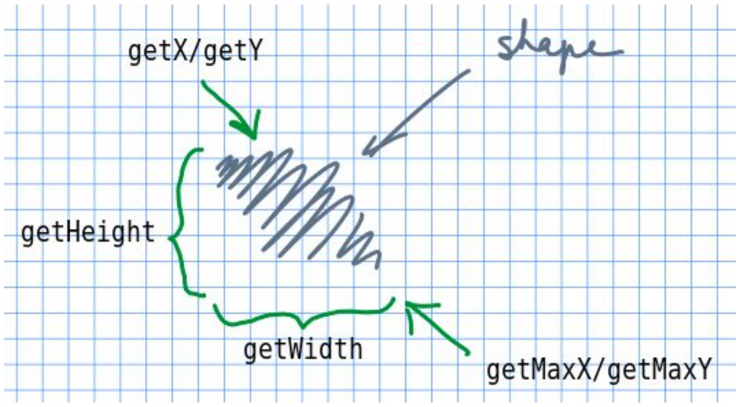
The Java API includes a range of classes and methods that are devoted to drawing. The physical structure of a GUI is built of components which refer to a visual element in a GUI, including menus, menu items, buttons, text-input boxes, check boxes, and so on. In Java, GUI components are represented by objects belonging to subclasses of the class *java.awt.Component.* In order to use graphics in Java programs, there are a number of libraries we need to import. For instance, as the libraries that we used:

import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JColorChooser;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JPanel;
import javax.swing.JRadioButton;.

Coordinates

The computer screen is a series of small squares called pixels, and the color of each pixel can be adjusted individually. So, drawing on the computer simply means changing the colors of the individual pixels. When the default transformation from user space to system space is used, the origin of the user space is the upper left corner of the drawing field of the part as (0,0). All coordinates are defined using integers, which are usually sufficient. The x coordinate increases from left to right, and the y coordinate increases from top to bottom. For any component, you can find out the size of the shape that it occupies by calling the instance methods getWidth() and getHeight(), which return the number of pixels in the horizontal and vertical directions, respectively.





For clearness, *getMaxX* and *getMaxY* return the coordinates of the bottom right corner. As we used the idea in the MouseClass for drawing the shapes but by implementing the *max/min* method of the Math library.

Colors

A color in Java is an object of the class, *java.awt.Color* and you can construct a new color by specifying its RGB components where java is designed to work with that color system.

To define the color of object, we set the initial action point to black with:

```
Color color = Color.BLACK;
```

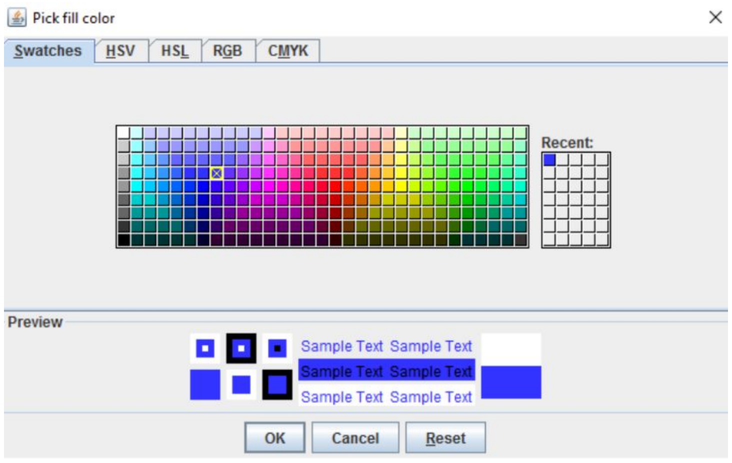
Later on, in order to enable the user set the color from the color palette, by the help of *JColorChooser.ShowDialog,* we used setters and getters for implement all of the shapes and graphs:

```
public void setColor(Color c) {
        color = c;
        }

public Color getColor() {
        return color;
        }
```

JColorChooser.ShowDialog, pops and the user chooses the color.

MouseClass **paint.color** = *JColorChooser.showDialog*(paint, "Choose a color", paint.color);



Shapes

We have a number of instance classes for drawing certain shapes such as rectangles, ellips, circle and so on which are extending from Shape class. The Shape is abstract class which contains same properties and methods for each shape and abstract methods which every class extends from it have to define. Each of shapes has (gets) initial coordinates, current coordinates, and current width vs height. By using action events, mouse events the movements as drawing the shape, filling the shape,, moving, resizing are happening .

Graphics

In the MouseEvent class for paintComponent() method, we use some properties from the Graphics class to provide more sophisticated control over geometry, coordinate transformations, color management, and text layout. This is the fundamental class for rendering 2-dimensional shapes, text and images on the Java platform. All drawing in Java is done through an object of type Graphics. The paintComponent() method of a JComponent gives you a graphics context of type Graphics that you can use for drawing on the component. In fact, the graphics context actually belongs to the sub-class Graphics2D (in Java version 1.2 and later), and can be type-cast to gain access to the advanced Graphics2D drawing methods:

```
public void paintComponent(Graphics g) {
        super.paintComponents(g);
        g.setColor(Color.WHITE);
        g.fillRect(0, 0, 900, 900);
        for (int i = 0; i < shapes.size(); i++) {
        if (shapes.get(i).isDraw()) {
        if (shapes.get(i).getName().equals("circle")) {

                g.setColor(shapes.get(i).getColor());
                if (shapes.get(i).isFilled())
                g.fillOval(((Circle) shapes.get(i)).getX(),
                        ((Circle) shapes.get(i)).getY(),
                        ((Circle) shapes.get(i)).getRadius(),
                        ((Circle) shapes.get(i)).getRadius());

                else
                g.drawOval(((Circle) shapes.get(i)).getX(),
                        ((Circle) shapes.get(i)).getY(),
                        ((Circle) shapes.get(i)).getRadius(),
                        ((Circle) shapes.get(i)).getRadius());
                         …..
                        //and goes in this sequence for all shapes
```

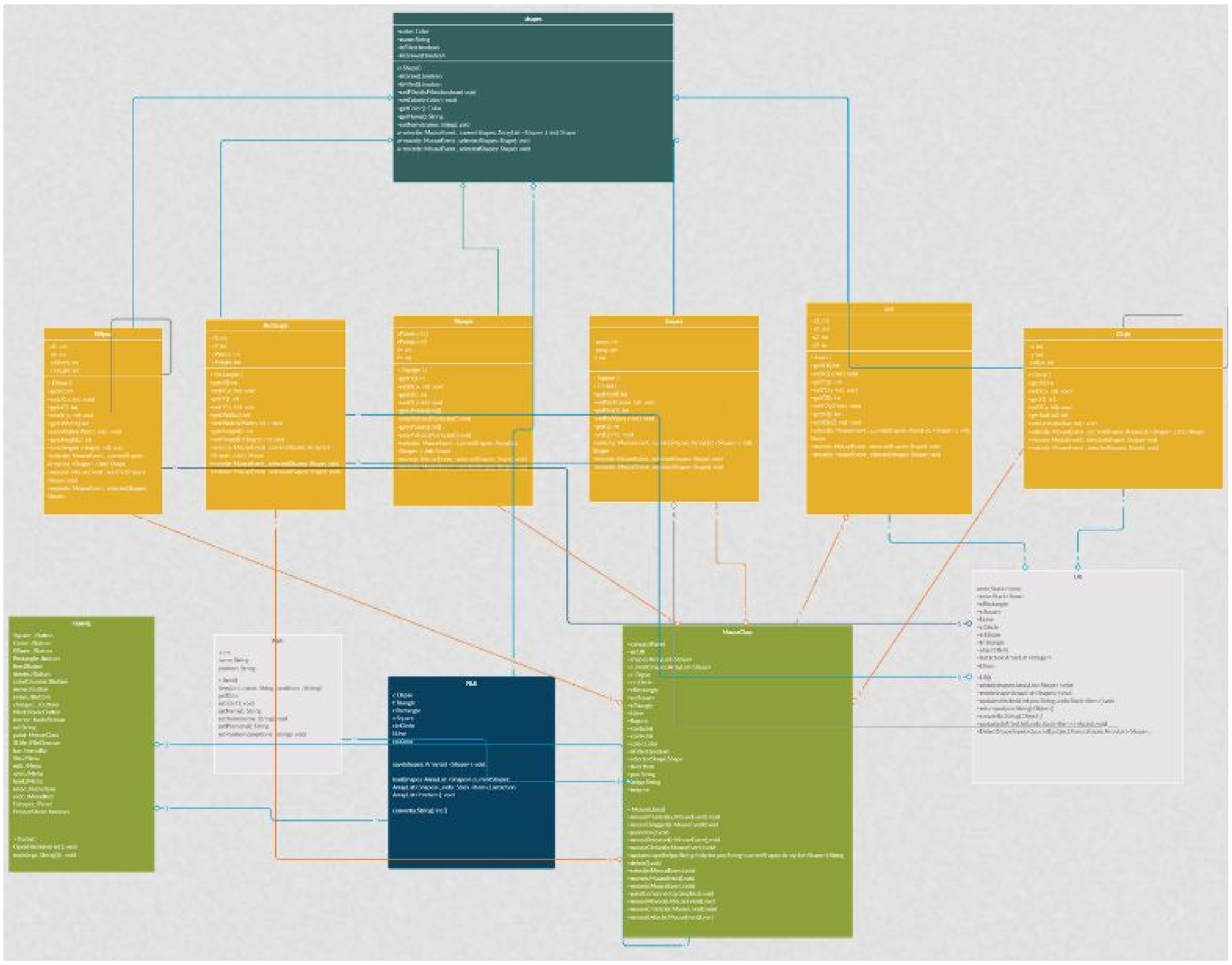
Programs must use this Graphics object (or one derived from it) to render output. They are free to change the state of the Graphics object as necessary.

Save with Serialization

In the File class the save and load operations take place. The save function takes shapes Arraylist as an argument and basically stores each shape to a file named pathname while checking each element in array to store it according to their class. This checking algorithm becomes handy when loading the previous painting. In the load function we read from file that we stored shapes before. Firstly we empty the shapes array and add elements again as we read from the file ,and by this way the previous session is on the screen.

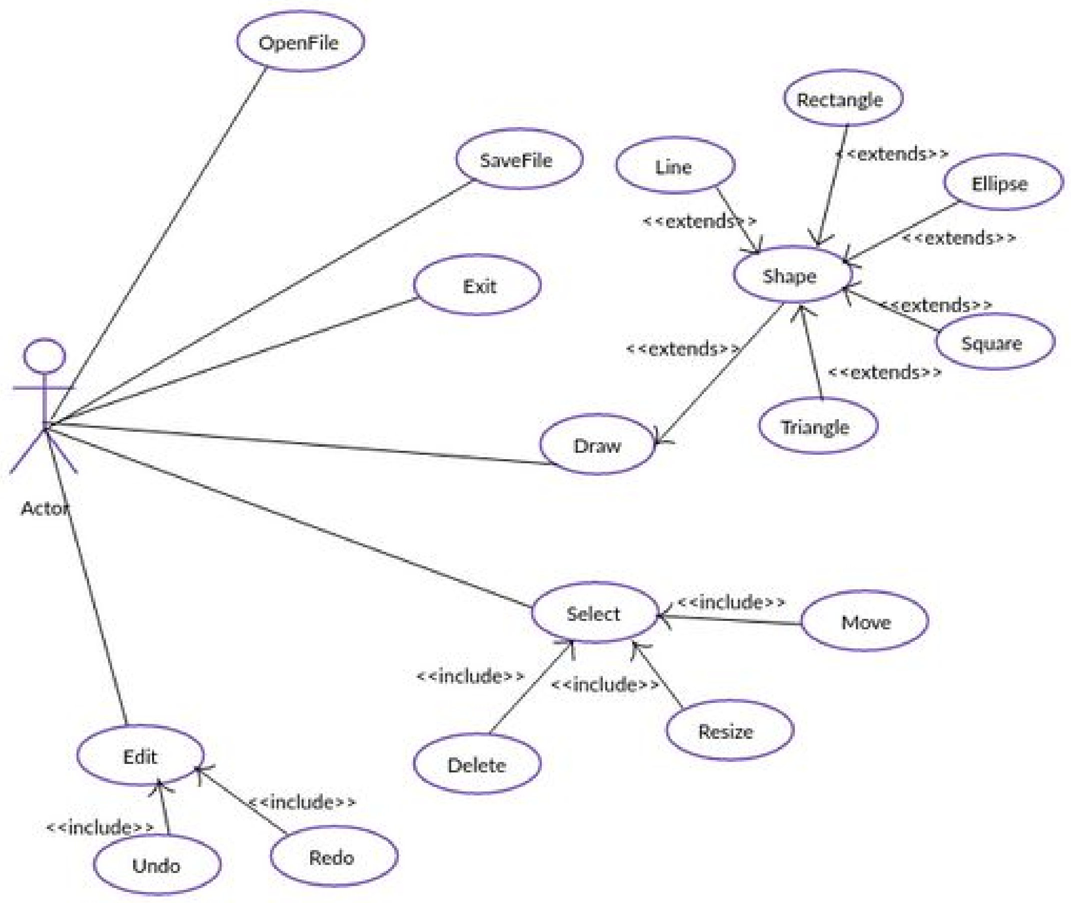
As we read and write shapes which extends from Shape class we have to make Shape class Serializable in order to operate save and load operations successfully.

UML Diagram



UML_Diagram

User Interface Specification



Summary

To get the best performance from these APIs, application programs must also take responsibility for writing programs which use the guidelines outlined in this document. In this software project we practiced using the tools of AWT and Swing while building our Graphical User Interface while developing it.

References

https://www.javatpoint.com/

http://www.cs.fsu.edu/~myers/cgs3416/notes/gui_intro.html

https://www.oracle.com/technetwork/java/painting-140037.html

https://docs.oracle.com/javase/tutorial/uiswing/

http://horstmann.com/sjsu/graphics/