5/14/2016

Pingere (Paint)

Unleash the artist

By : Yara Mohamed Abdullatif 3674

Mostafa Ali Mansour 3250

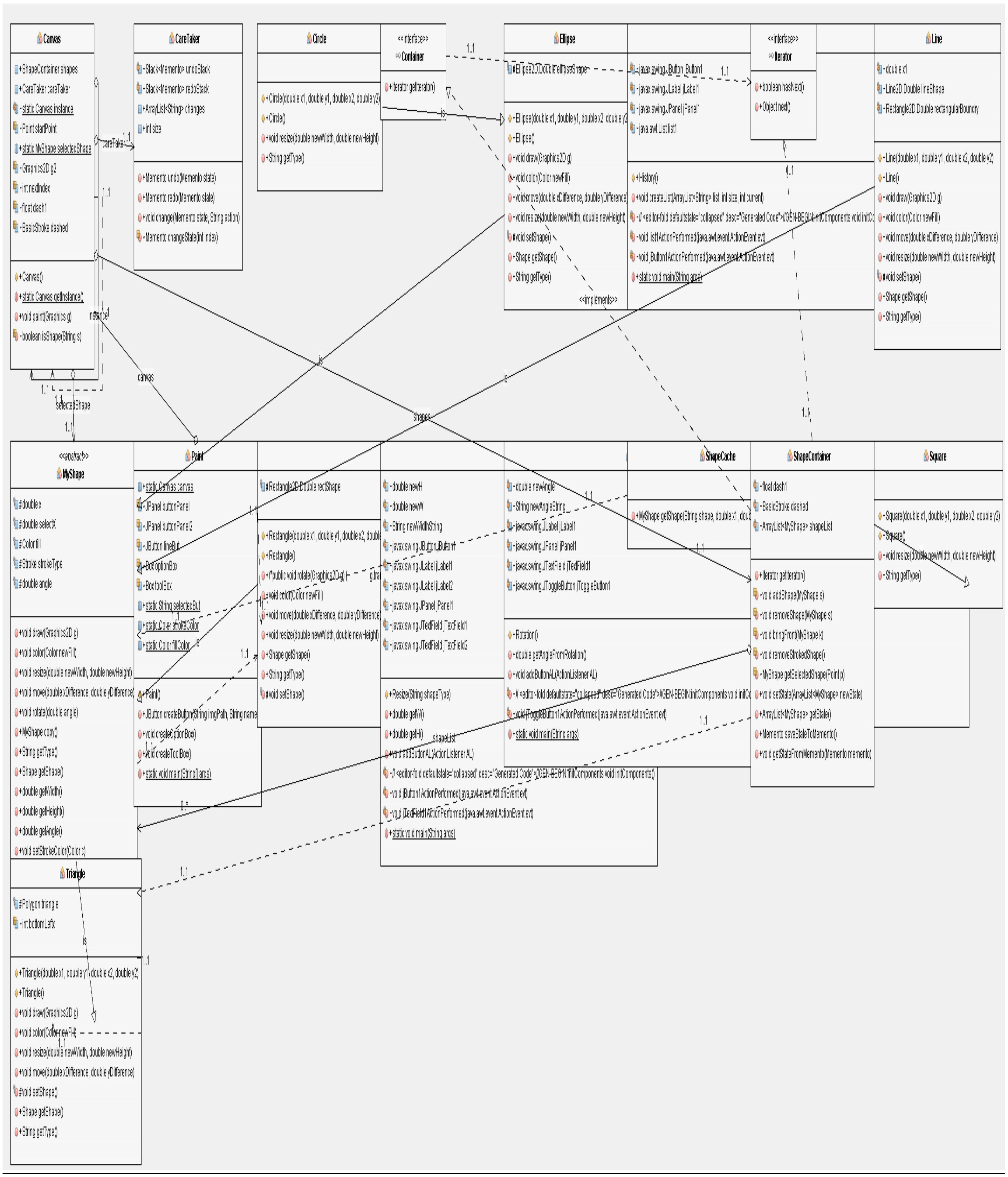

Canvas
+ShapeContainer shapes
+CareTaker careTaker
-static Canvas instance
-Point startPoint
+static MyShape selectedShape
-Graphics2D g2
-int nextIndex
-float dash1
-BasicStroke dashed
+Canvas()
+static Canvas getInstance()
+void paint(Graphics g)
-boolean isShape(String s)
CareTaker
-Stack<Memento> undoStack
-Stack<Memento> redoStack
+ArrayList<String> changes
+int size
+Memento undo(Memento state)
+Memento redo(Memento state)
+void change(Memento state, String action)
-Memento changeState(int index)
Circle
+Circle(double x1, double y1, double x2, double y2)
+Circle()
+void resize(double newWidth, double newHeight)
+String getType()
<<interface>>
Container
+Iterator getIterator()
Ellipse
#Ellipse2D.Double ellipseShape
+Ellipse(double x1, double y1, double x2, double y2)
+Ellipse()
+void draw(Graphics2D g)
+void color(Color newFill)
+void move(double xDifference, double yDifference)
+void resize(double newWidth, double newHeight)
#void setShape()
+Shape getShape()
+String getType()
-javax.swing.JButton jButton1
-javax.swing.JLabel jLabel1
-javax.swing.JPanel jPanel1
-java.awt.List list1
+History()
+void createList(ArrayList<String> list, int size, int current)
-// <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents void initCo
-void list1ActionPerformed(java.awt.event.ActionEvent evt)
-void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
+static void main(String args)
<<interface>>
Iterator
+boolean hasNext()
+Object next()
Line
-double x1
-Line2D.Double lineShape
-Rectangle2D.Double rectangularBoundry
+Line(double x1, double y1, double x2, double y2)
+Line()
+void draw(Graphics2D g)
+void color(Color newFill)
+void move(double xDifference, double yDifference)
+void resize(double newWidth, double newHeight)
#void setShape()
+Shape getShape()
+String getType()
<<implements>>
<<abstract>>
MyShape
#double x
#double selectX
#Color fill
#Stroke strokeType
#double angle
+void draw(Graphics2D g)
+void color(Color newFill)
+void resize(double newWidth, double newHeight)
+void move(double xDifference, double yDifference)
+void rotate(double angle)
+MyShape copy()
+String getType()
+Shape getShape()
+double getWidth()
+double getHeight()
+double getAngle()
+void setStrokeColor(Color c)
Paint
+static Canvas canvas
-JPanel buttonPanel
-JPanel buttonPanel2
-JButton lineBut
-Box optionBox
-Box toolBox
+static String selectedBut
+static Color strokeColor
+static Color fillColor
+Paint()
+JButton createButton(String imgPath, String name
+void createOptionBox()
+void createToolBox()
+static void main(String[] args)
#Rectangle2D.Double rectShape
+Rectangle(double x1, double y1, double x2, double
+Rectangle()
+// public void rotate(Graphics2D g) {
+void color(Color newFill)
+void move(double xDifference, double yDifference)
+void resize(double newWidth, double newHeight)
+Shape getShape()
+String getType()
#void setShape()
-double newH
-double newW
-String newWidthString
-javax.swing.JButton jButton1
-javax.swing.JLabel jLabel1
-javax.swing.JLabel jLabel2
-javax.swing.JPanel jPanel1
-javax.swing.JTextField jTextField1
-javax.swing.JTextField jTextField2
+Resize(String shapeType)
+double getW()
+double getH()
+void addButtonAL(ActionListener AL)
-// <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents void initComponents()
-void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
-void jTextField1ActionPerformed(java.awt.event.ActionEvent evt)
+static void main(String args)
-double newAngle
-String newAngleString
-javax.swing.JLabel jLabel1
-javax.swing.JPanel jPanel1
-javax.swing.JTextField jTextField1
-javax.swing.JToggleButton jToggleButton1
+Rotation()
+double getAngleFromRotation()
+void addButtonAL(ActionListener AL)
-// <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents void initCo
-void jToggleButton1ActionPerformed(java.awt.event.ActionEvent evt)
+static void main(String args)
ShapeCache
+MyShape getShape(String shape, double x1, doub
ShapeContainer
-float dash1
-BasicStroke dashed
-ArrayList<MyShape> shapeList
+Iterator getIterator()
-void addShape(MyShape s)
-void removeShape(MyShape s)
-void bringFront(MyShape k)
-void removeStrokedShape()
-MyShape getSelectedShape(Point p)
+void setState(ArrayList<MyShape> newState)
+ArrayList<MyShape> getState()
+Memento saveStateToMemento()
+void getStateFromMemento(Memento memento)
Square
+Square(double x1, double y1, double x2, double y2)
+Square()
+void resize(double newWidth, double newHeight)
+String getType()
Triangle
#Polygon triangle
-int bottomLeftx
+Triangle(double x1, double y1, double x2, double y2)
+Triangle()
+void draw(Graphics2D g)
+void color(Color newFill)
+void resize(double newWidth, double newHeight)
+void move(double xDifference, double yDifference)
#void setShape()
+Shape getShape()
+String getType()
careTaker
instance
canvas
selectedShape
shapes
shapeList
is
1..1
0..*

Figure 1 UML

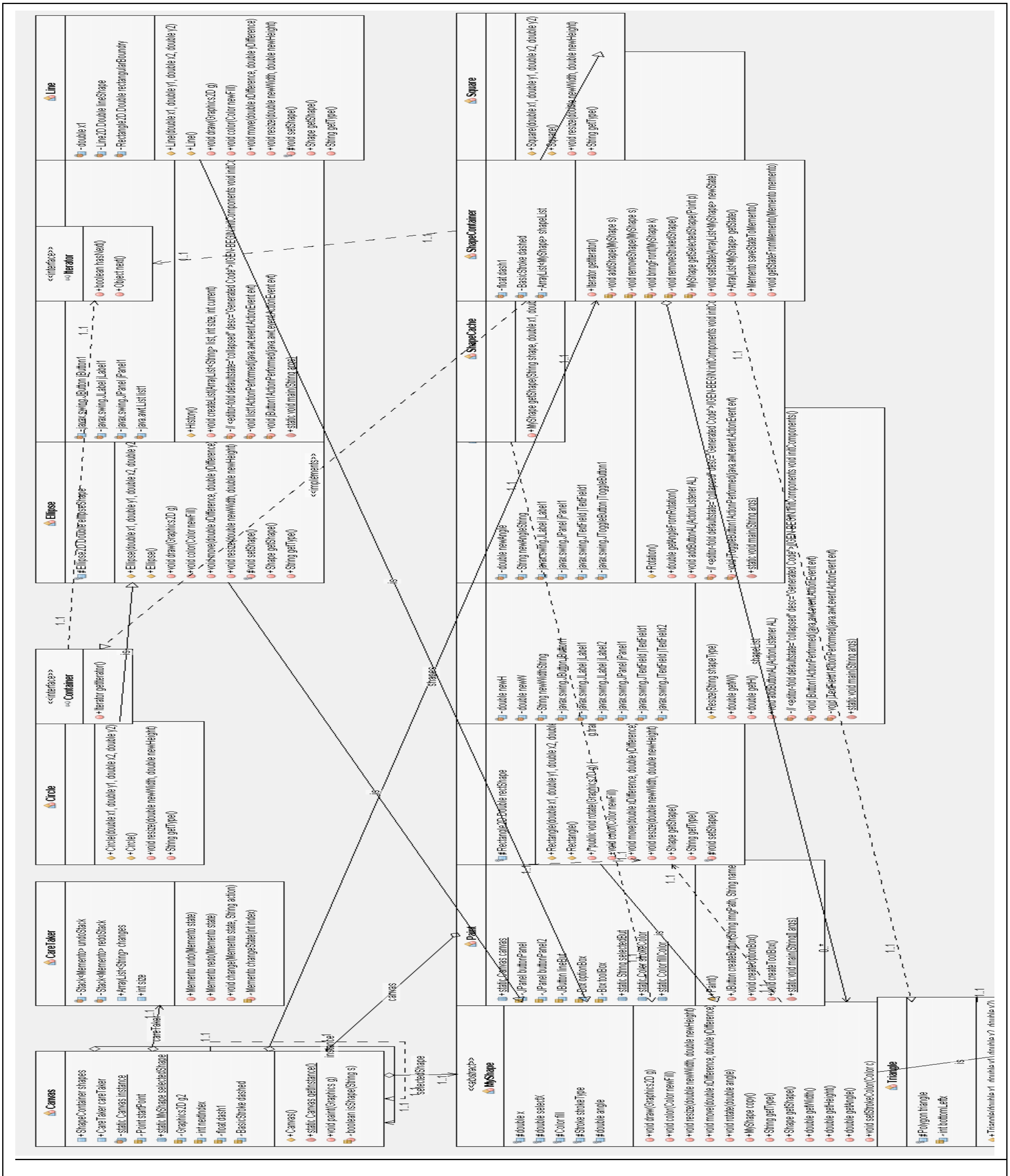

Canvas
+ShapeContainer shapes
+CareTaker careTaker
-static Canvas instance
-Point startPoint
+static MyShape selectedShape
-Graphics2D g2
-int nextIndex
-float dash1
-BasicStroke dashed
+Canvas()
+static Canvas getInstance()
+void paint(Graphics g)
-boolean isShape(String s)
CareTaker
-Stack<Memento> undoStack
-Stack<Memento> redoStack
+ArrayList<String> changes
+int size
+Memento undo(Memento state)
+Memento redo(Memento state)
+void change(Memento state, String action)
-Memento changeState(int index)
Circle
+Circle(double x1, double y1, double x2, double y2)
+Circle()
+void resize(double newWidth, double newHeight)
+String getType()
<<interface>>
Container
+Iterator getIterator()
Ellipse
#Ellipse2D.Double ellipseShape
+Ellipse(double x1, double y1, double x2, double y2
+Ellipse()
+void draw(Graphics2D g)
+void color(Color newFill)
+void move(double xDifference, double yDifference)
+void resize(double newWidth, double newHeight)
#void setShape()
+Shape getShape()
+String getType()
<<implements>>
-javax.swing.JButton jButton1
-javax.swing.JLabel jLabel1
-javax.swing.JPanel jPanel1
-java.awt.List list1
+History()
+void createList(ArrayList<String> list, int size, int current)
-// <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents void initCo
-void list1ActionPerformed(java.awt.event.ActionEvent evt)
-void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
+static void main(String args)
<<interface>>
Iterator
+boolean hasNext()
+Object next()
Line
-double x1
-Line2D.Double lineShape
-Rectangle2D.Double rectangularBoundry
+Line(double x1, double y1, double x2, double y2)
+Line()
+void draw(Graphics2D g)
+void color(Color newFill)
+void move(double xDifference, double yDifference)
+void resize(double newWidth, double newHeight)
#void setShape()
+Shape getShape()
+String getType()
<<abstract>>
MyShape
#double x
#double selectX
#Color fill
#Stroke strokeType
#double angle
+void draw(Graphics2D g)
+void color(Color newFill)
+void resize(double newWidth, double newHeight)
+void move(double xDifference, double yDifference)
+void rotate(double angle)
+MyShape copy()
+String getType()
+Shape getShape()
+double getWidth()
+double getHeight()
+double getAngle()
+void setStrokeColor(Color c)
Paint
+static Canvas canvas
-JPanel buttonPanel
-JPanel buttonPanel2
-JButton lineBut
-Box optionBox
-Box toolBox
+static String selectedBut
+static Color strokeColor
+static Color fillColor
+Paint()
+JButton createButton(String imgPath, String name
+void createOptionBox()
+void createToolBox()
+static void main(String[] args)
#Rectangle2D.Double rectShape
+Rectangle(double x1, double y1, double x2, double
+Rectangle()
+public void rotate(Graphics2D g)
+void color(Color newFill)
+void move(double xDifference, double yDifference)
+void resize(double newWidth, double newHeight)
+Shape getShape()
+String getType()
#void setShape()
-double newH
-double newW
-String newWidthString
-javax.swing.JButton jButton1
-javax.swing.JLabel jLabel1
-javax.swing.JLabel jLabel2
-javax.swing.JPanel jPanel1
-javax.swing.JTextField jTextField1
-javax.swing.JTextField jTextField2
+Resize(String shapeType)
+double getW()
+double getH()
+void addButtonAL(ActionListener AL)
-// <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents void initComponents()
-void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
-void jTextField1ActionPerformed(java.awt.event.ActionEvent evt)
+static void main(String args)
-double newAngle
-String newAngleString
-javax.swing.JLabel jLabel1
-javax.swing.JPanel jPanel1
-javax.swing.JTextField jTextField1
-javax.swing.JToggleButton jToggleButton1
+Rotation()
+double getAngleFromRotation()
+void addButtonAL(ActionListener AL)
-// <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents void initCo
-void jToggleButton1ActionPerformed(java.awt.event.ActionEvent evt)
+static void main(String args)
ShapeCache
+MyShape getShape(String shape, double x1, doub
ShapeContainer
-float dash1
-BasicStroke dashed
-ArrayList<MyShape> shapeList
+Iterator getIterator()
+void addShape(MyShape s)
+void removeShape(MyShape s)
+void bringFront(MyShape k)
+void removeStrokedShape()
+MyShape getSelectedShape(Point p)
+void setState(ArrayList<MyShape> newState)
+ArrayList<MyShape> getState()
+Memento saveStateToMemento()
+void getStateFromMemento(Memento memento)
Square
+Square(double x1, double y1, double x2, double y2)
+Square()
+void resize(double newWidth, double newHeight)
+String getType()
Triangle
#Polygon triangle
-int bottomLeftx
+Triangle(double x1, double y1, double x2, double y2)
careTaker
canvas
instance
selectedShape
shapes
shapeList
is
1..1
0..*

Figure 1 UML

Classes and Interfaces

1 - `public abstract class MyShape implements Cloneable :` used to declare and/or implement all the methods which is used to manipulate the shapes.

2 - `public class Paint extends JFrame` : it acts as a frame and container to draw the shapes and color them and set the buttons and tools used to achieve the manipulation with a default white background. It is the only class which is instantiated in its own main method to execute the program and the Canvas class instantiated inside it.

3 - `public class ShapeCache` : used to create shapes according to their type which is passed as a string parameter to `getShape()`, along with start and end point of the mouse drag. This class serves the **Prototype** creational design pattern.

4 - `public interface Iterator :`

Blueprint for ShapeIterator of ShapeContainer. Part of Iterator design pattern.

5 - `public interface Container :`

Blueprint for ShapeContainer. Part of Iterator design pattern.

6 - `public class ShapeContainer implements Container :`

Used to maintain an list of drawn shapes in ArrayList`<MyShape> shapeList` with access methods to the shapeList. This class is used in the **Iterator**, **Memento** and **Observer** design patterns. It is a way of encapsulation to keep all behavior related to the `shapeList` inside one class.

`7- public class ShapeIterator implements Iterator`

Inside the `ShapeContainer`. It implements the Iterator interface and its `hasNext()` and `next()` methods.

`8 - public class Memento`

Serves the Memento design pattern. It is used in saving or returning the states at any point as Memento to keep track of it in the `CareTaker` class.

9 - `public class Canvas extends JComponent` : contains fields of type Point to indicate the start and end point of the mouse as a mouse listener is added to it and MouseAdapter is passed as parameter so that any action can be sensed whether it was a button that is pressed or a drawing or moving the shape, the start point is determined when mouse is pressed and end point is recorded when mouse is released.

The type of the button is determined by the action listener added to the buttons while creating them.

Inside the class there is the most important method;

`public void paint(Graphics g):` The Graphics class is the abstract super class for all graphics contexts which allow an application to draw onto components that can be realized on various devices, or onto off-screen images as well.

A Graphics object encapsulates all state information required for the basic rendering operations that Java supports. State information includes the following properties.

Then it is casted to `Graphics2D` and saved in g2, the stroke's thickness of each shape is set to 4 with color black and white fill color (is these properties are the optimum and standard in the painting programs)

After that it checks if the start point and end point of the mouse are void or not then instance of `ShapeCache (sf)` and `MyShape(aShape)` are instantiated, the name of the button used to represent the shape is passed to the constructor of the `ShapeCache` instance `(sf)` so that it can return a `MyShape` to its instance variable `(aShape)`. If a shape is not selected, go away; this is achieved by if condition to check if the `aShape` variable is null or not.

We set the start and end point to null to avoid drawing extra stuff we will draw the shape once.

The color of each shape and their stroke is registered when choosing it using the `Graphics2D g2`, so, they are retrieved before drawing it in the canvas and stored in `tempPaint` and `tempStroke`. The idea of the class is that with every shape chosen it is stored inside an ArrayList of type `MyShape` inside the `ShapeContainer` called `shapeList.` Finally we iterate over the ArrayList and use the method draw (explained later) to draw each shape in the canvas.

10 - `public class Rectangle extends MyShape` : contains a `Rectangle2D.Double` shape with methods used to manipulate it (resize, move ,fill, etc..). Its constructor receives parameters of `(double x1, double y1, double x2, double y2)` which indicates the start and end point of the mouse used to draw, those parameters are passed from a mouse listener implemented in the canvas class.

11 - `public class Square extends Rectangle` : the principle of inheritance is strongly applied in this class as the square inherit fields and methods from the `Rectangle` class , it simply takes the minimum of the width and height and then assign its width and height to the value.

```
this.width = Math.min(this.width,this.height);
this.height = this.width;
```

12 - `public class Line extends MyShape Line2D.Double` shape with methods used to manipulate it (resize, move ,fill, ..). Its constructor receives parameters of `(double x1, double y1, double x2, double y2)` which indicates the start and end point of the mouse used to draw, those parameters are passed from a mouse listener implemented in the canvas class.

13 - `public class Triangle extends MyShape` : it is a `Polygon` with fields to determine the coordinates of its vertices `(int bottomLeftx, bottomLefty, bottomRightx, bottomRighty, topVertexx, topVertexy.`

Start and end points are represented by (x1,y2,x2,y2), thus, these equations are calculated to assign each position of each vertex as follow :

```
x = Math.min(x1, x2);
y = Math.min(y1, y2);
width = Math.abs(x1 - x2);
height = Math.abs(y1 - y2);
bottomLeftx = (int) x1;
bottomLefty = (int) y2;
bottomRightx = (int) x2;
bottomRighty = (int) y2;
topVertexx = (int) (x1 + x2) / 2;
topVertexy = (int) y1;
```

It is allowed to draw equilateral or isosceles triangle as is the most used shapes in designing according to graphic designers.

14 - `public class ellipse extends MyShape` : contains a `Ellipse2D.Double` shape with methods used to manipulate it (resize, move ,fill, ..). Its constructor receives parameters of `(double x1, double y1, double x2, double y2)` which indicates the start and end point of the mouse used to draw, those parameters are passed from a mouse listener implemented in the canvas class.

15 - `public class Circle extends Ellipse` : also enhance the inheritance principle along with the major one (that all the classes representing shapes are inheriting the `MyShape` class). In order to draw the circle we assign the width and height to the maximum width and height coming as if the shape was `Ellipse`.

16 - `public class CareTaker :`

Part of the Memento design pattern. Used to take care of the change in state of the ShapeContainer, the originator. It keeps the undo & redo stacks and the history list, and has methods to handle undo, redo and any change in state.

17 - `public class Resize extends javax.swing.JFrame` : It pops a windows when the user choose the resize button and select the shape he wants to resize it and it gives the user options to change the horizontal and vertical dimensions (except the line it is allowed to choose one dimension), then it fires an action listener to the canvas when the user presses ok so that the selected shape will resize itself using the resize method implemented in its class.

18 - `public class Rotation extends javax.swing.JFrame` : It pops a windows when the user choose the rotation button and select the shape he wants to rotate it and it gives the user option to rotate it in degrees as it is most conventional.

Design Patterns

1- Singleton

Singleton design pattern is a **creational** pattern used to create classes which we can have only one object from. We used the `Canvas` class as a Singleton class because we can only have one `Canvas` per Paint project.

2- Prototype

Prototype pattern is a **creational** pattern. It is useful when we want to clone objects. We apply the Prototype pattern in `MyShape` class which implements **`Cloneable`** interface, and overrides the `clone()` method. `ShapeCache` is used to return a shape from `getShape()` based on given type. For example, passing "rectangle" to `getShape()` will return a new instance of `Rectangle`.

3- Proxy

Proxy design pattern is a **structural** pattern. In proxy pattern, a class represents functionality of another class. We used the Proxy pattern to encapsulate the behavior of the ArrayList `shapeList` inside `ShapeContainer`, which contains all interaction methods of `shapeList; addShape(), removeShape(), getSelectedShape(), removeStrokedShape()` and `bringFront().`

4- Iterator

Iterator pattern is a **behavioral** pattern. It is used to access a collection without need to expose its representation to user. We created interfaces **`Container`** and **`Iterator.`** Class `ShapeContainter` implements **`Container`** and contains `shapeList` arraylist and all its access methods. The `ShapeIterator` class implements `Iterator`. It is responsible for iteration over `shapeList` in the `ShapeContainer`, so it overrides `next()` and `hasNext()` methods.

5- <u>Memento</u>

Memento design pattern is a **behavioral** pattern. It contains 3 main classes, *Originator*, *Memento* and *Caretaker*. We used the memento pattern to provide undo and redo options and history list for the Paint project. Class `ShapeContainer` is the Originator, which represents the state of the project as the ArrayList of `MyShape`. It implements `getState()`, `saveStateToMemento()` and `getStateFromMemento()` methods. The `CareTaker` class contains `undoStack` and `redoStack` which track the Memento/states at each change in the project. It also contains the `undo()`, `redo`() and `change()` methods which are responsible to manipulate the two stacks whenever change occurs, or undo and redo are required. `changeState(index)` in the `CareTaker` returns Memento at a certain index required by user.

6- <u>Observer</u>

Observer pattern is a **behavioral** pattern. It uses three actor classes. *Subject*, *Observer* and *Client*. In our code, the Subject class, **`Canvas`** contains `ShapeContainer shapes,` which is a list of *Observers*. `MyShape` objects inside the `shapeList` are the *Observers*. The shapes are notified of change inside the **`Canvas`** using the action listeners in the constructor.

User guide

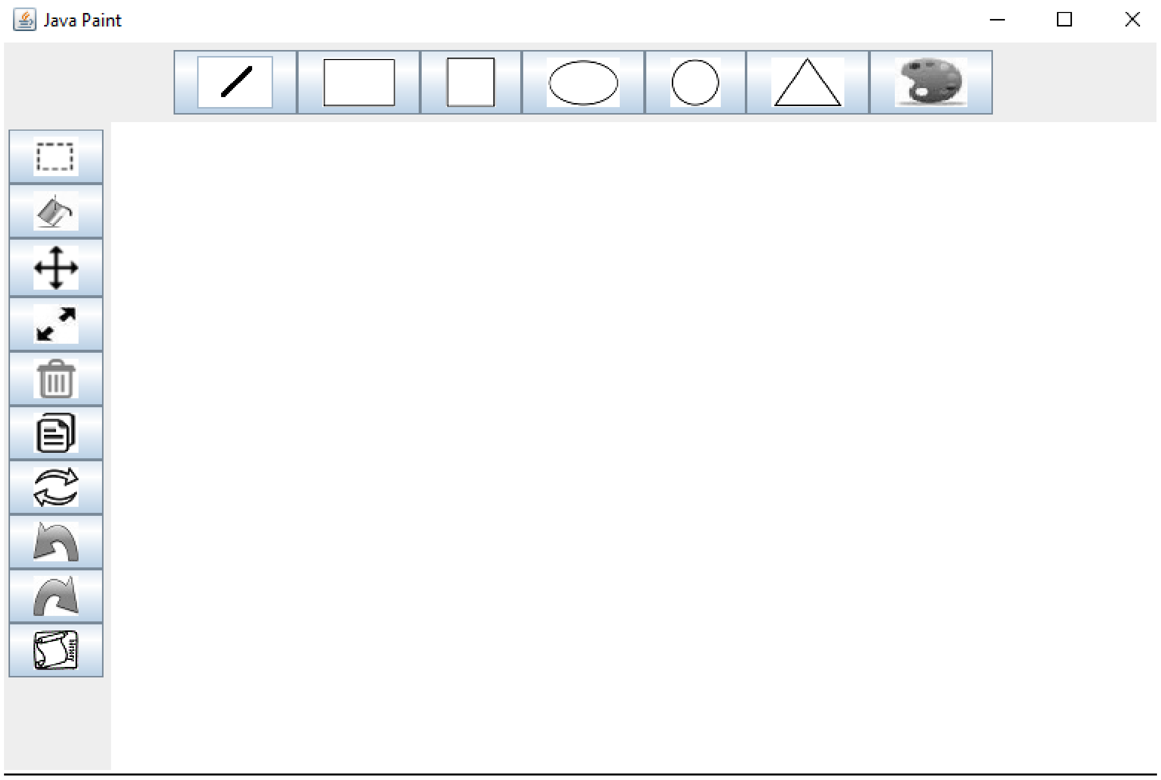


Figure 2 The interface of the program

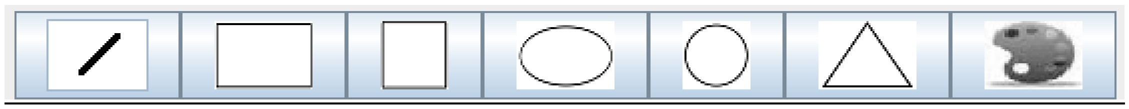


Figure 3 The option box to choose the shape and its stroke color before drawing it

In order to draw a shape the user must first click the desired shape then press the mouse over the canvas (white zone) and drag it then release it and voila ! The shape is drawn.

The stroke button is meant to be placed in the option box despite that it isn’t a shape because the conventional way is to choose the stroke’s color before drawing it.

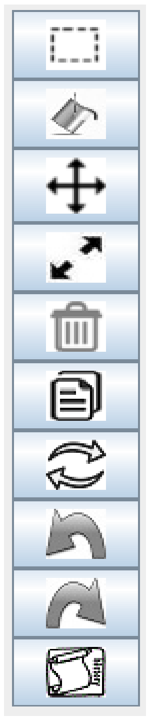


Figure 4 The toolbox used to select, fill, move, resize, delete, copy, undo, redo and show history

In order to perform any change to the shape the user must choose the command then select the shape and perform the operation.

1- <u>Draw :</u>

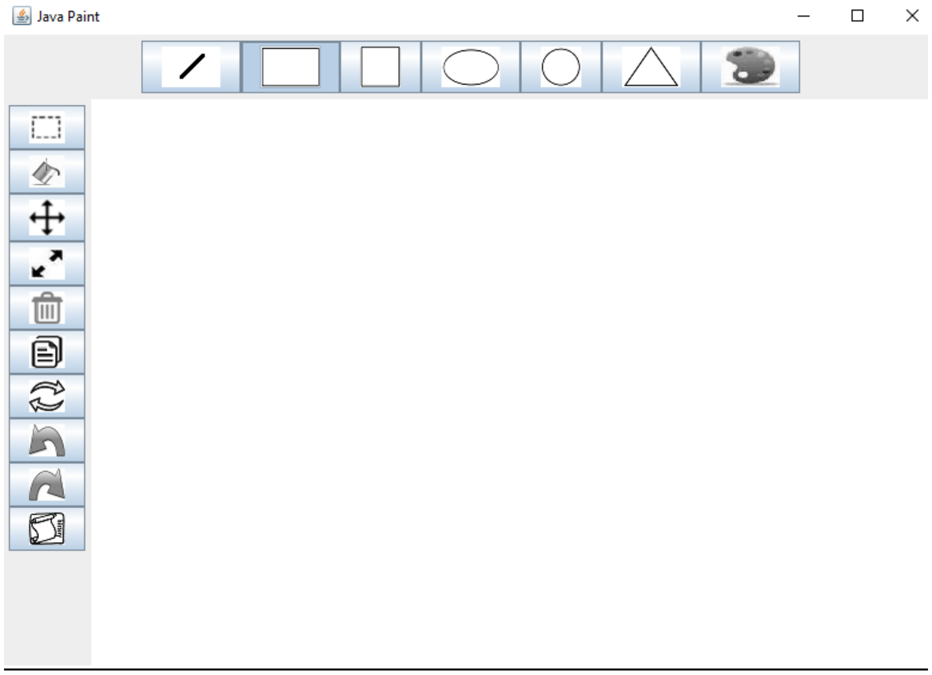


Figure 5 Press rectangle button to draw it

We place the mouse where we want to have our top left corner and drag it till the mouse to the place where we want to have our bottom right

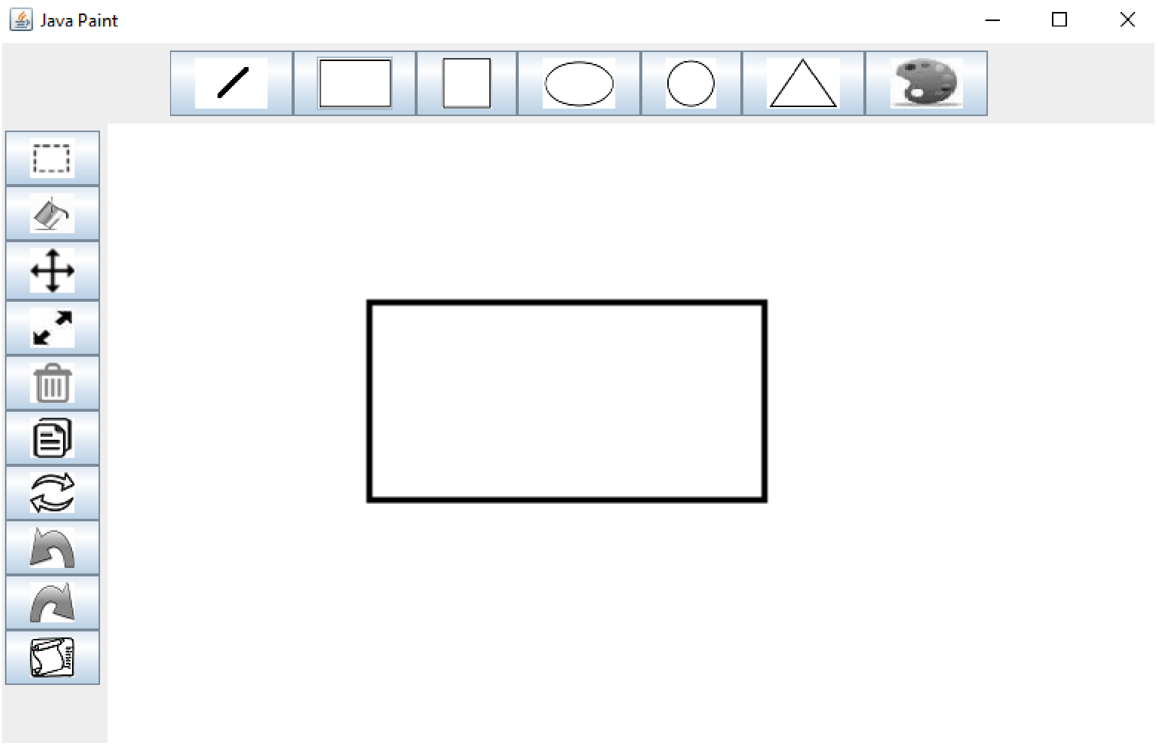


Figure 6 Rectangle is drawn

The square, ellipse and circle goes the same technique, in t line it depends whether we want to draw it horizontally, vertically or with angle. In triangle, the coordinates of bottom left vertex is determined by the coordinates where the user pressed the mouse and the bottom right is determined where the mouse was released (the y value of both buttons are equal to the y of the mouse released) while the top vertex coordinates equals the y of the mouse pressed and its x values equals half the distance between the difference of the x values of the bottom vertexes.

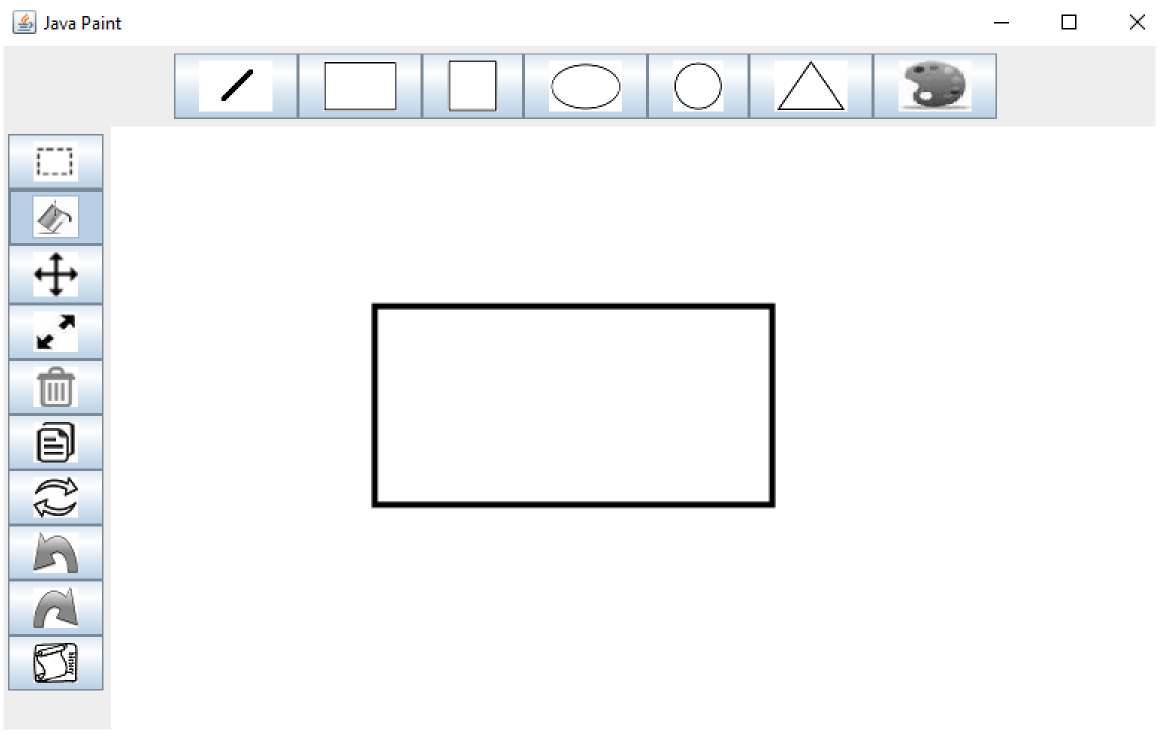


Figure 7 Fill button is pressed

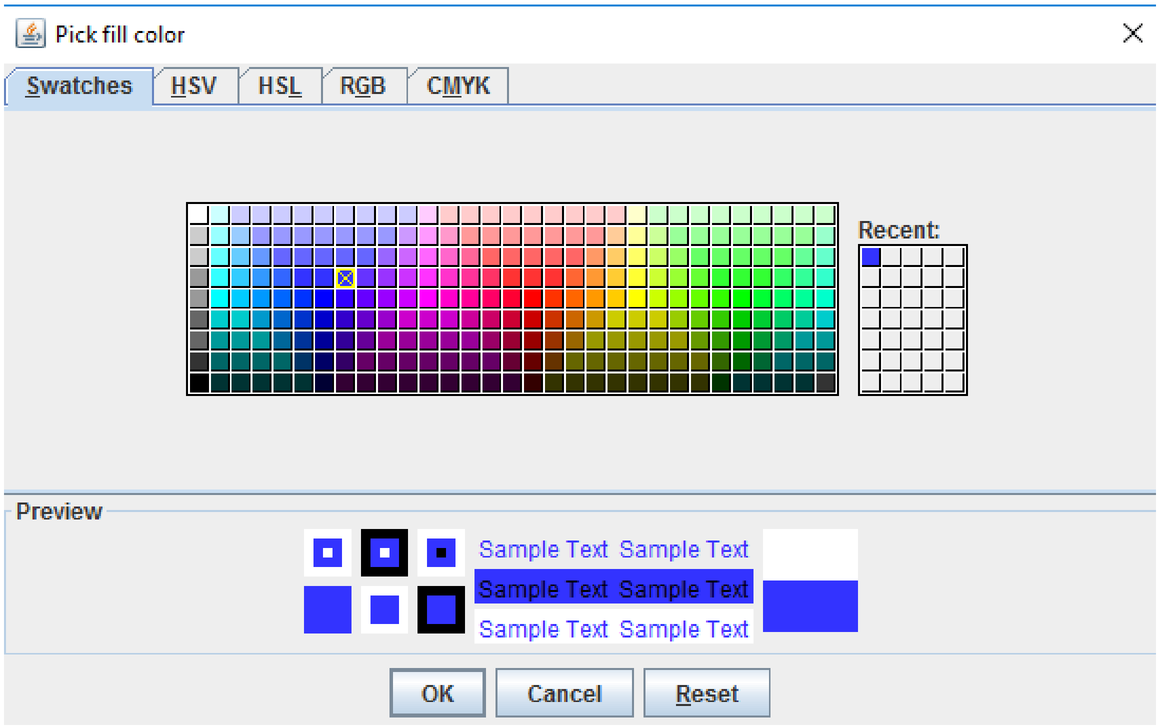


Figure 8 JColorChooser.showDialog pops and the user chooses the color blue

Notice that the default fill color is white and stroke color is black.

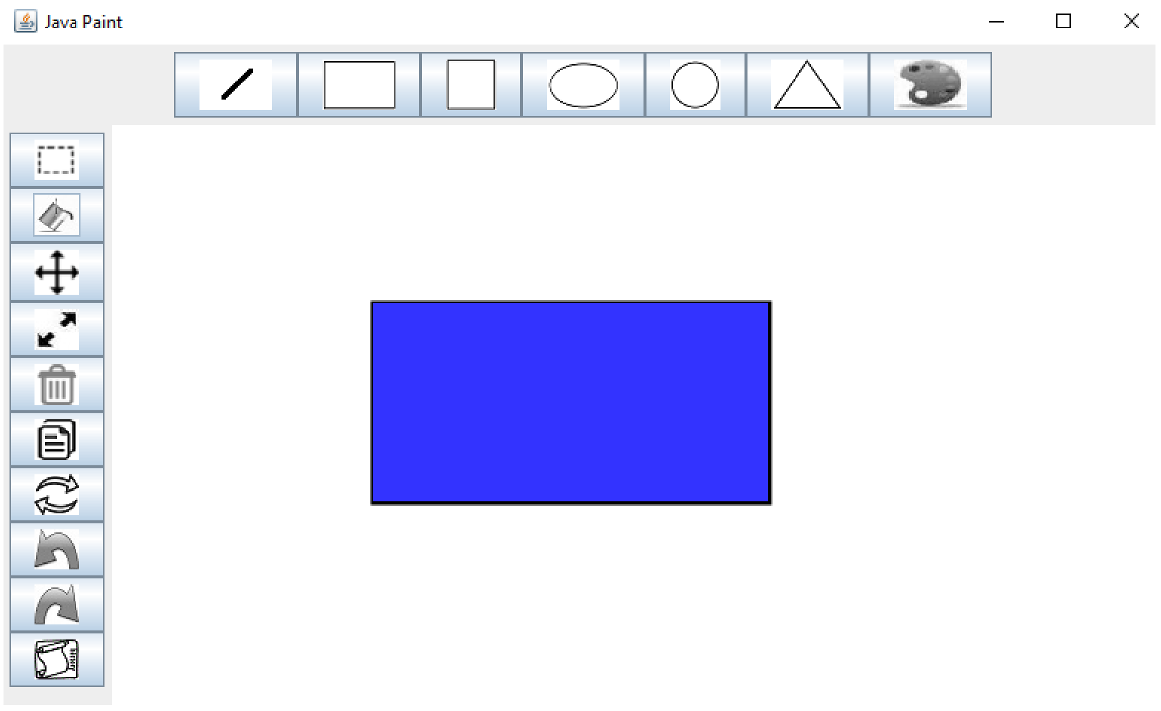


Figure 9 And we have a blue rectangle

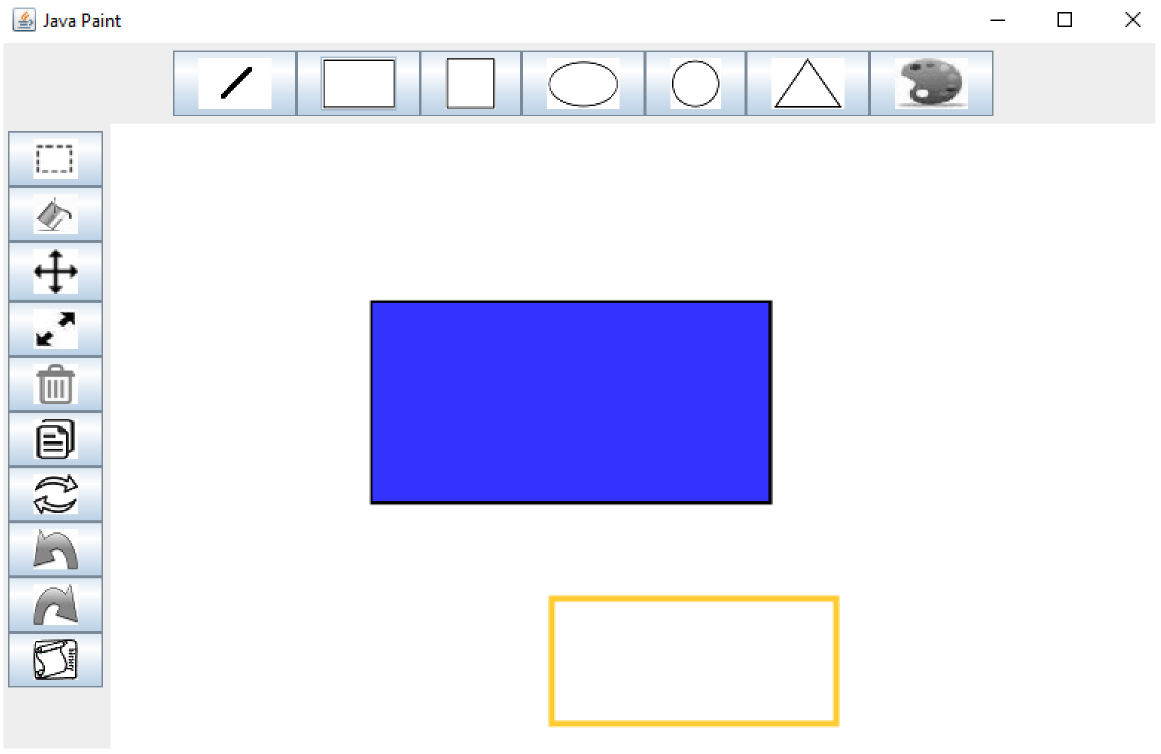


Figure 10 An extra feature that we have implemented is that the user can pick the stroke color before drawing the shape

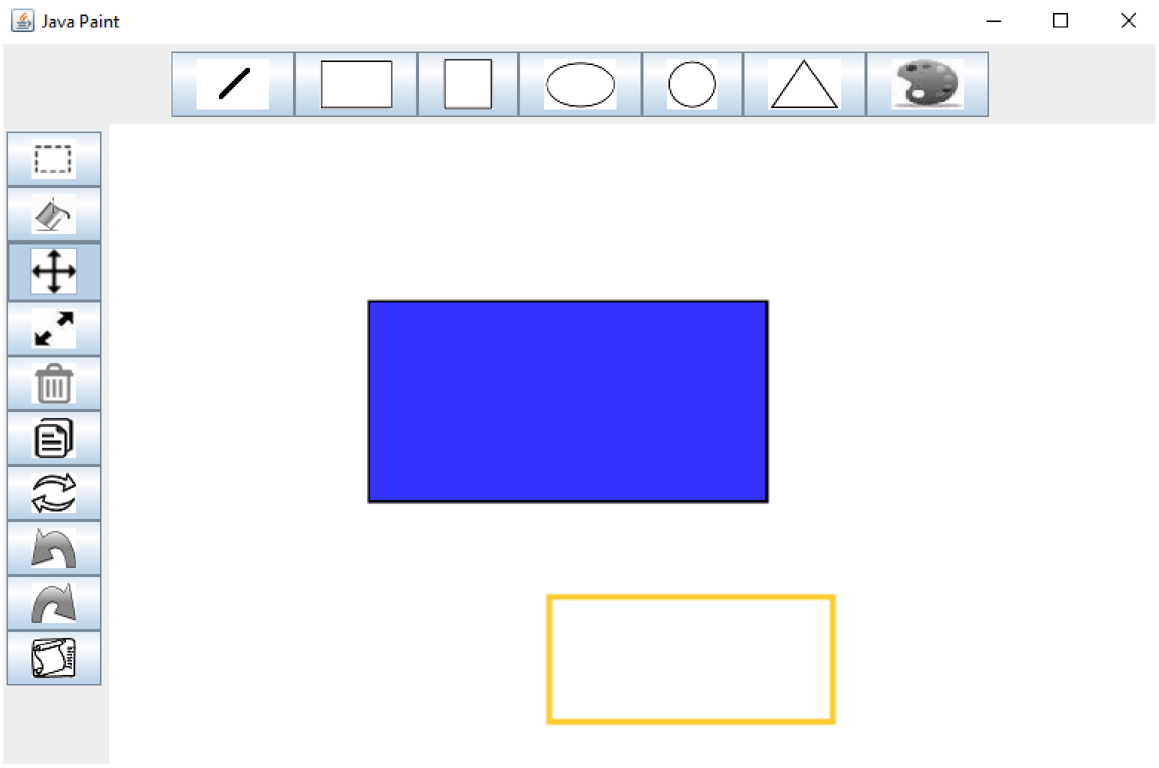


Figure 11 Move button is pressed

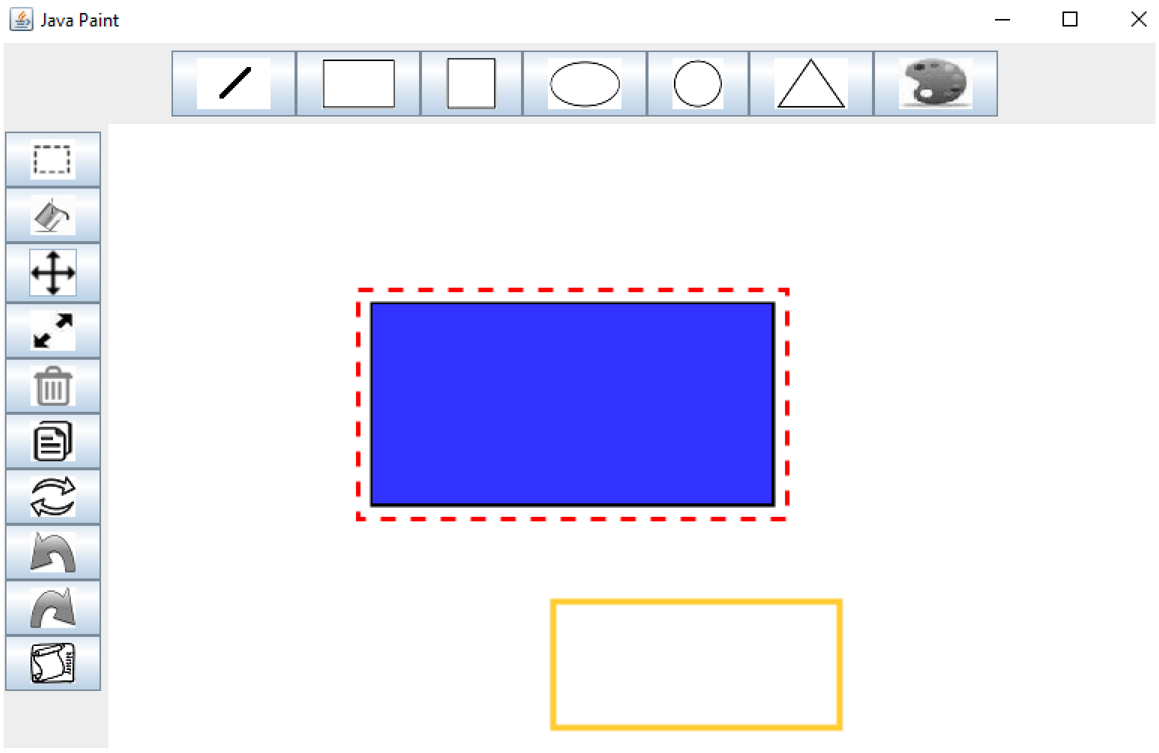


Figure 12 select the shape we want to move and a red-dotted rectangle is formed around it

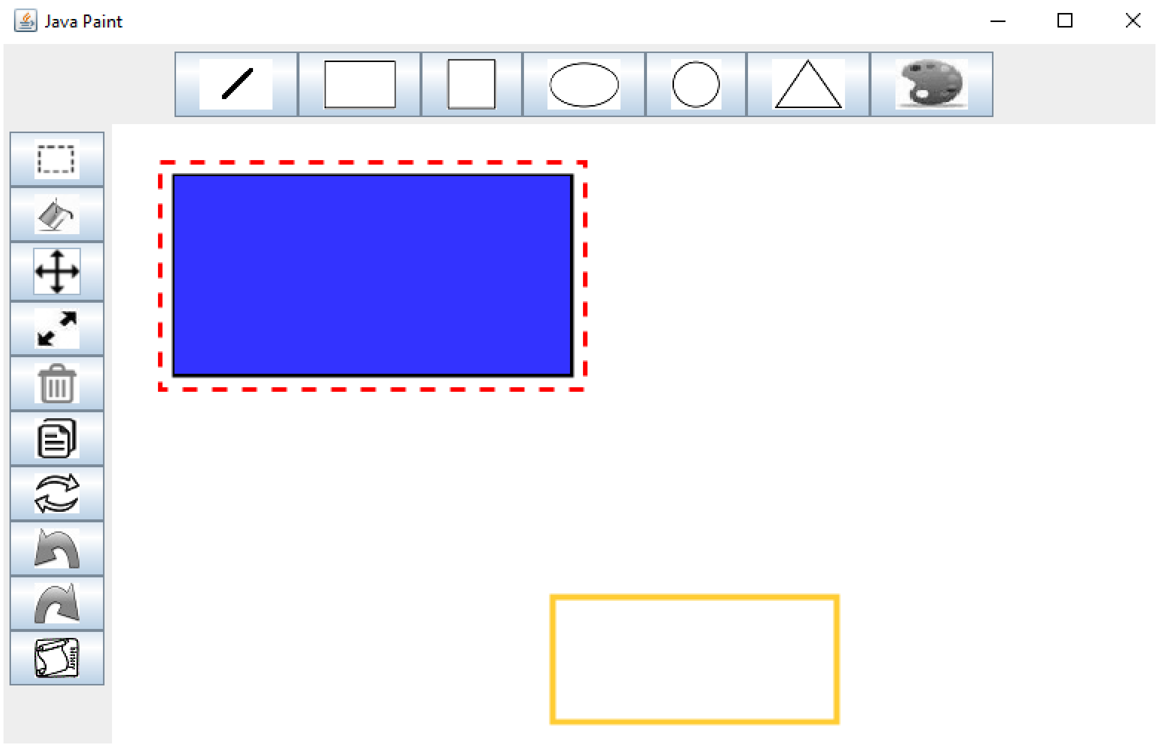
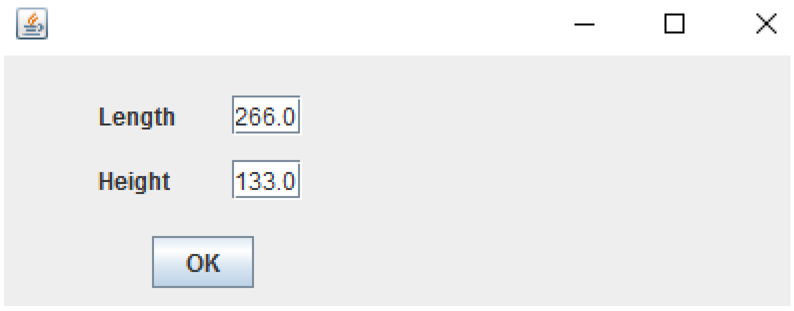


Figure 13 And we moved it

When we press the resize button then select the shape we wish to resize it a window is popped to ask the user for the new dimension(the line has only one dimension and the resizing operation of it is done using several equations).



Length 266.0

Height 133.0

Figure 14 The window is displaying the current dimensions of the shape

We will resize it to dimensions of 100 , 50 and when we press ok the visibility value of the window will be equal to false and disappears and the shape will be resized.

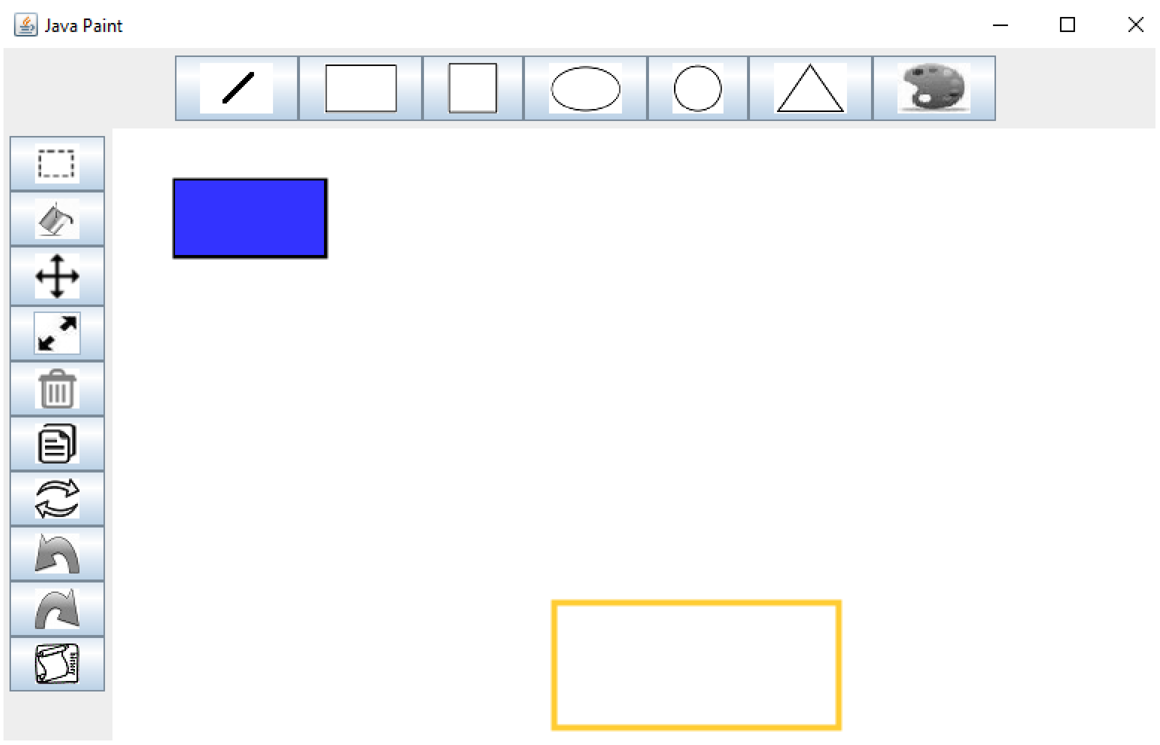


Figure 15 The new dimensions of the rectangle is applied

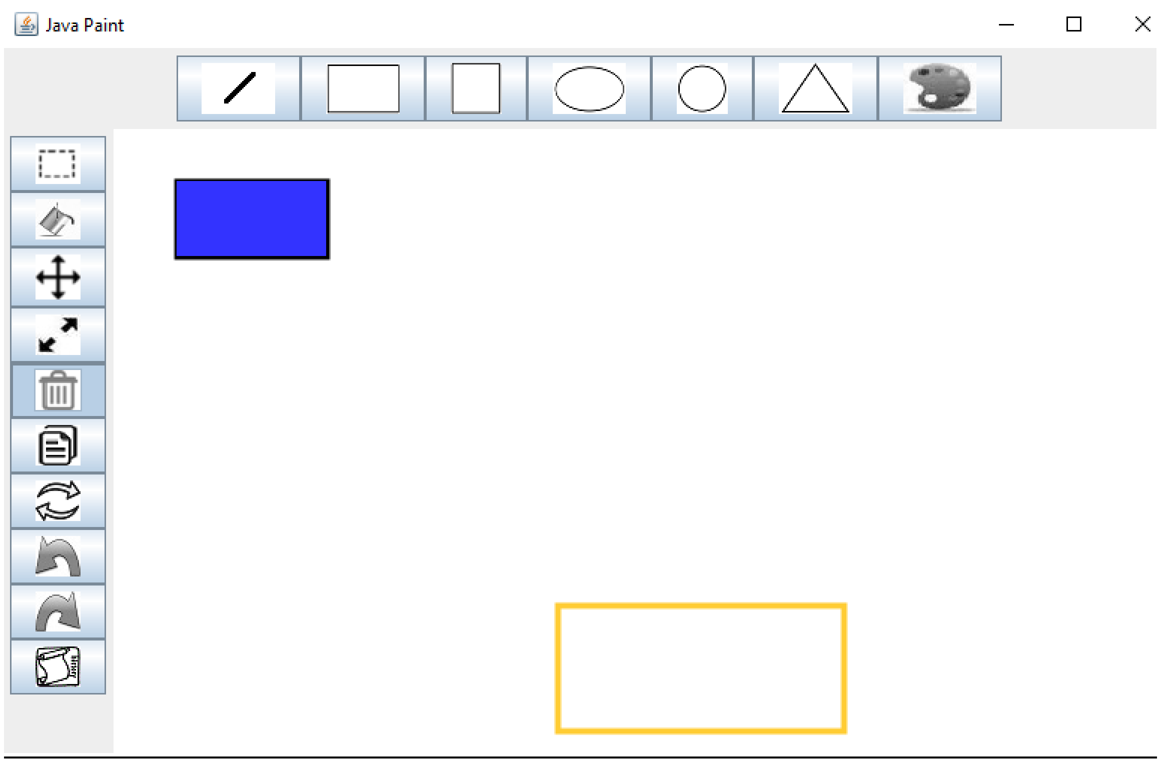


Figure 16 In order to delete we choose the delete button and press the shape we wish to delete.

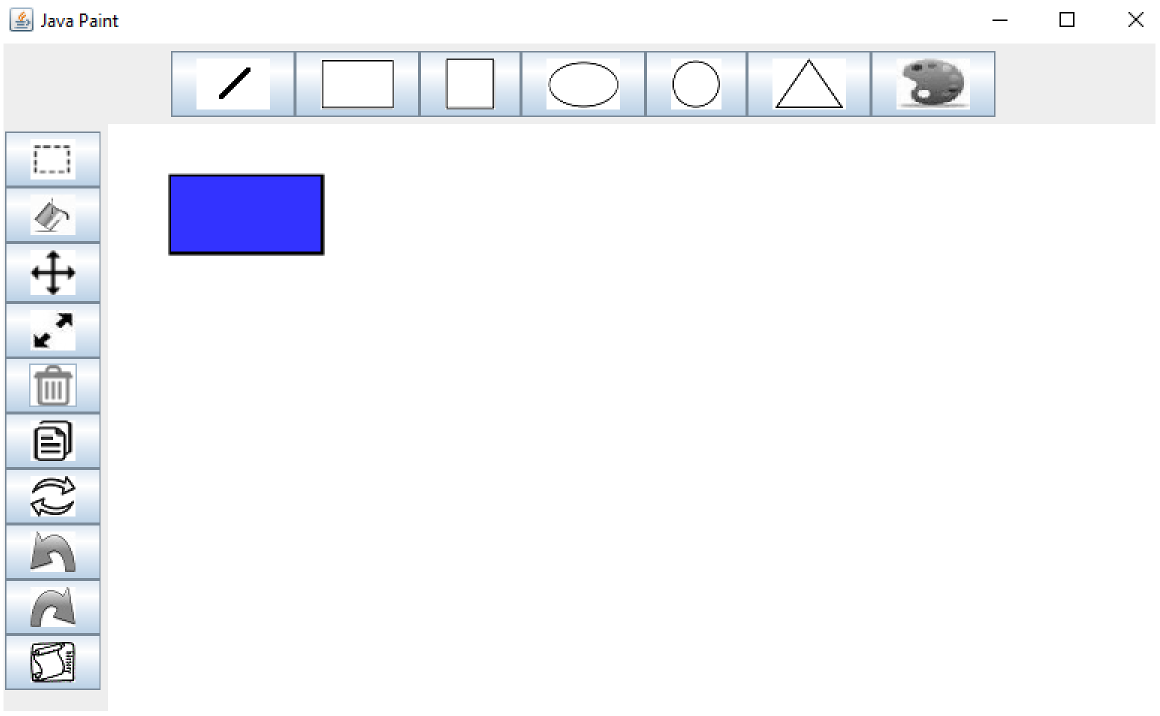


Figure 17 The orange rectangle has been deleted

In copy option we thought the optimum place to paste the shape is the coordinates of the shape itself because if the user has drawn several shapes then it could be placed over a drawn shape and cause confusion, and what a better place to redraw the shape than its old place? Then the user will simply use the move option to move it the place he desires.

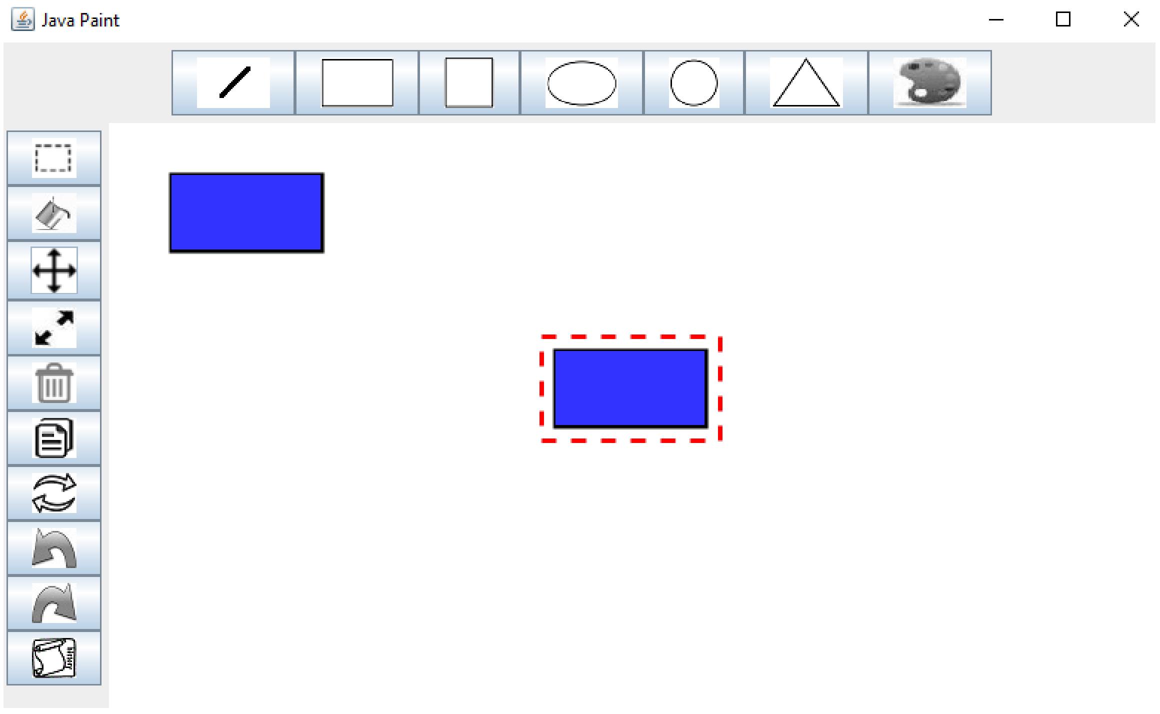


Figure 18 After the copying

The rotation way is much alike the resize only the angle is asked for, the angle entered is accumulated on the new angle because if the user wishes to rotate several times then its easier for him to update the new shape's angle not the original one. The angle's text field is continuously updated with the selected shape's angle.

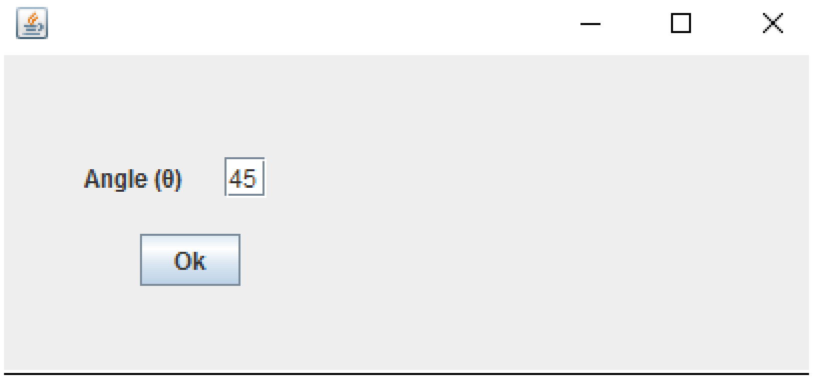


Figure 19 The new entered angle to rotate the rectangle

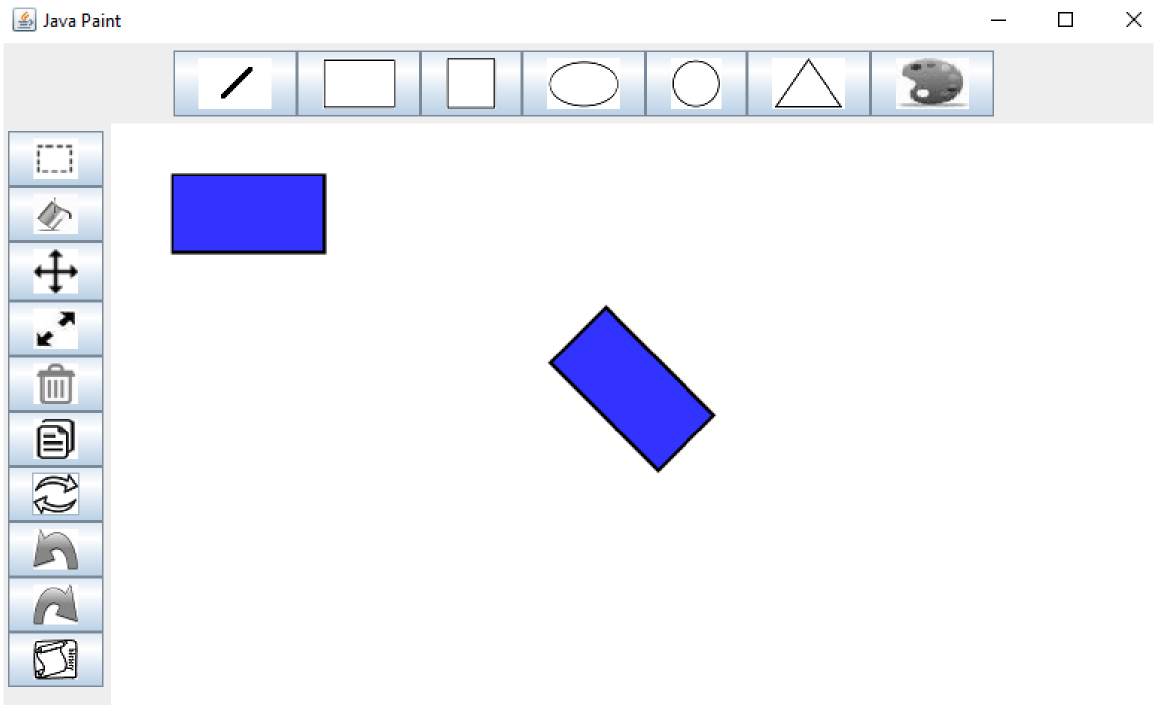
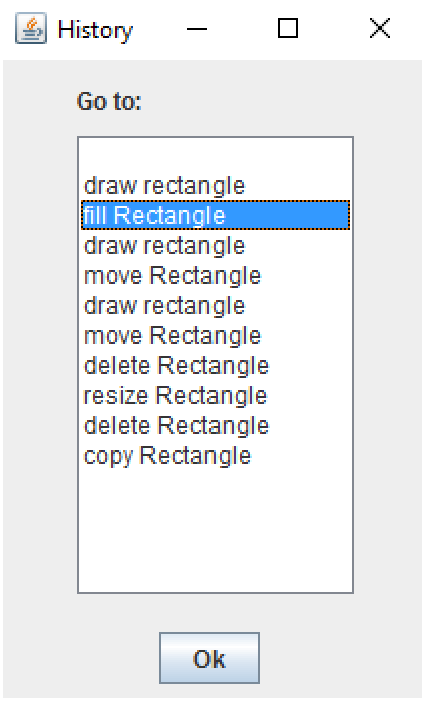


Figure 20 And the rectangle is rotated

The undo and redo is used like any undo or redo in any program, the user will have just to press it.

As for the history then a window pops containing all the operations done on the canvas and the user will pick one of them and press Ok button.



History

Go to:

draw rectangle
fill Rectangle
draw rectangle
move Rectangle
draw rectangle
move Rectangle
delete Rectangle
resize Rectangle
delete Rectangle
copy Rectangle

Figure 21 The history list

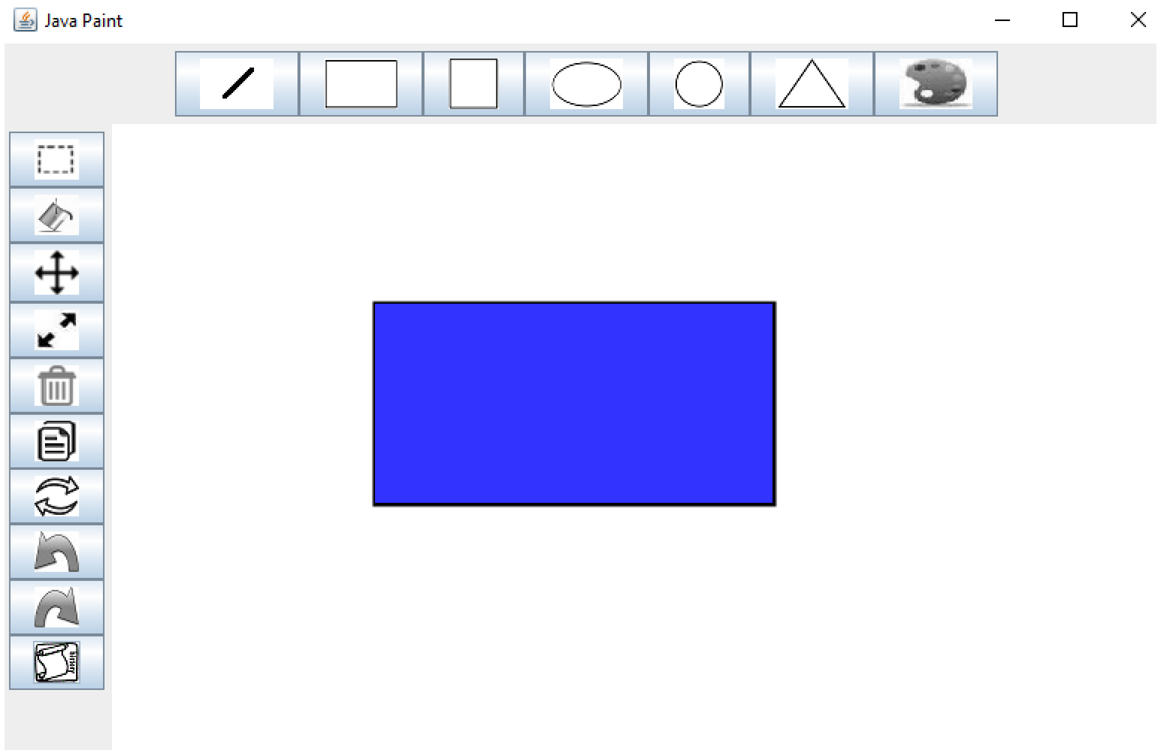


Figure 22 And we have the rectangle all alone