FRENCH-AZERBAIJANI UNIVERSITY

Advanced Object Oriented Programming
Paint Mini-Project #1
Java Swing

Computer Science L2
Group G6:
Shoykyat Sharafyabi
Nadir Abdullayev
Kanan Mikayilov

1 March, 2020

Introduction:

- ***Objective:***
  The main idea of this project is to create an app which serves to create a drawing project, create shapes in it, change them, to be able to save this project into a file and later load this file for making some other changes.

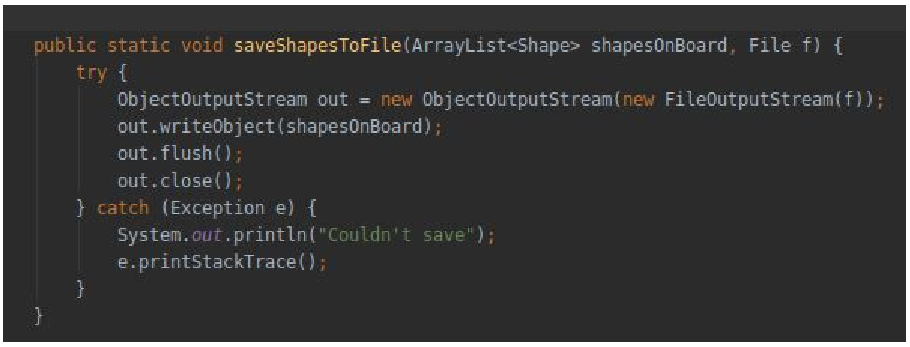
In this project Java Swing and I/O streams were implemented.

Our project consists of 10 classes:
Main : *which has main function which simply initializes the DrawingBoard class*
DrawingBoard: *class that extends JPanel class and is a draw area to draw some shapes*
Menu: *class that extends JPanel class and serves to create, open, save project and exit from the app*
FileManager: *class that serves to save objects to a file, and load shapes from file to a project , implements I/O streams*
Shape: *an abstract class which contains the main properties for a general shape, and the main methods that are necessary to implement these shapes into DrawingBoard.*
Ellipse , Rectangle , Line : *classes that extends Shape class and serve to create these shapes on the DrawingBoard*
PaintGraphics: *a class that extends JPanel and contains buttons that serve to create different shapes, move them, erase them, fill them, change color.*

The main functions:

saveShapesToFile:

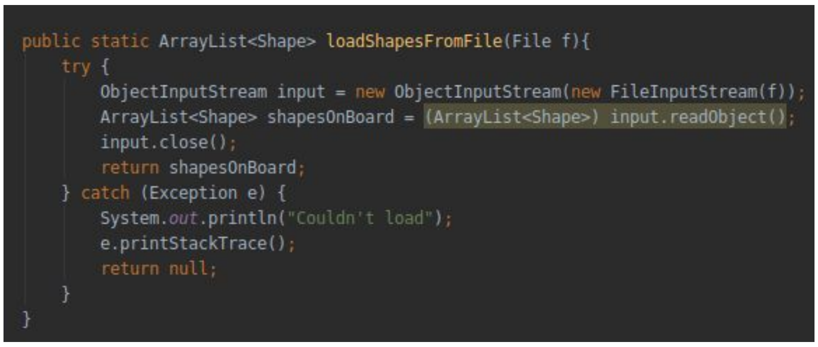
This function takes as an argument ArrayList of type shape and write each object in this ArrayList into a file, by ObjectOuputStream class implementation. But there should be careful, and write it in the try..catch block to catch an exception.



```
public static void saveShapesToFile(ArrayList<Shape> shapesOnBoard, File f) {
    try {
        ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(f));
        out.writeObject(shapesOnBoard);
        out.flush();
        out.close();
    } catch (Exception e) {
        System.out.println("Couldn't save");
        e.printStackTrace();
    }
}
```

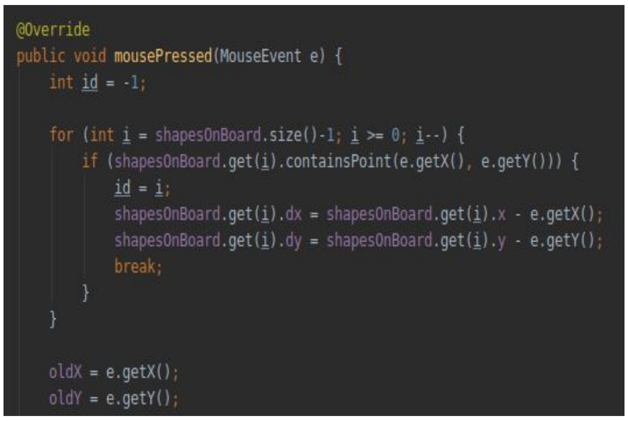
loadShapesFromFile:

This function takes as an argument file , reads object from this file and put them into an ArrayList of type shapes. Again should use try...catch block to catch an exception.



```
public static ArrayList<Shape> loadShapesFromFile(File f){
    try {
        ObjectInputStream input = new ObjectInputStream(new FileInputStream(f));
        ArrayList<Shape> shapesOnBoard = (ArrayList<Shape>) input.readObject();
        input.close();
        return shapesOnBoard;
    } catch (Exception e) {
        System.out.println("Couldn't load");
        e.printStackTrace();
        return null;
    }
}
```

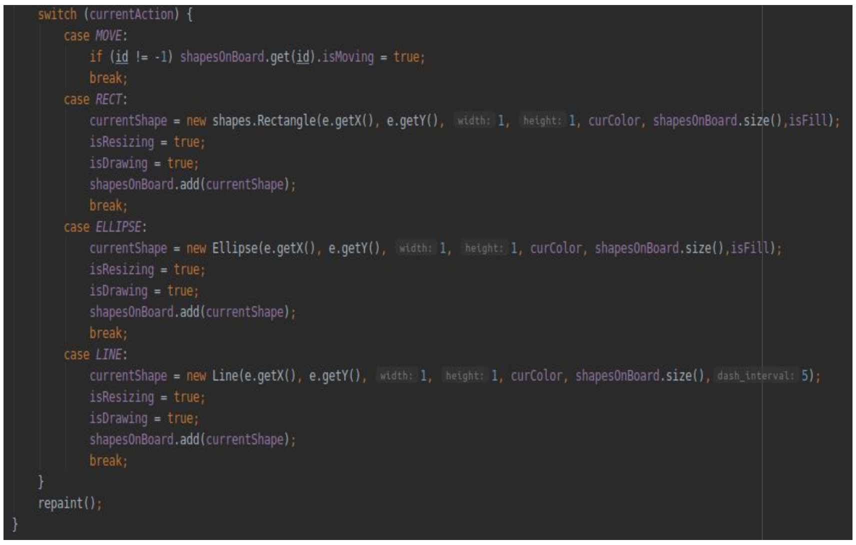
mousePressed:



```java
@Override
public void mousePressed(MouseEvent e) {
    int id = -1;

    for (int i = shapesOnBoard.size()-1; i >= 0; i--) {
        if (shapesOnBoard.get(i).containsPoint(e.getX(), e.getY())) {
            id = i;
            shapesOnBoard.get(i).dx = shapesOnBoard.get(i).x - e.getX();
            shapesOnBoard.get(i).dy = shapesOnBoard.get(i).y - e.getY();
            break;
        }
    }

    oldX = e.getX();
    oldY = e.getY();
```

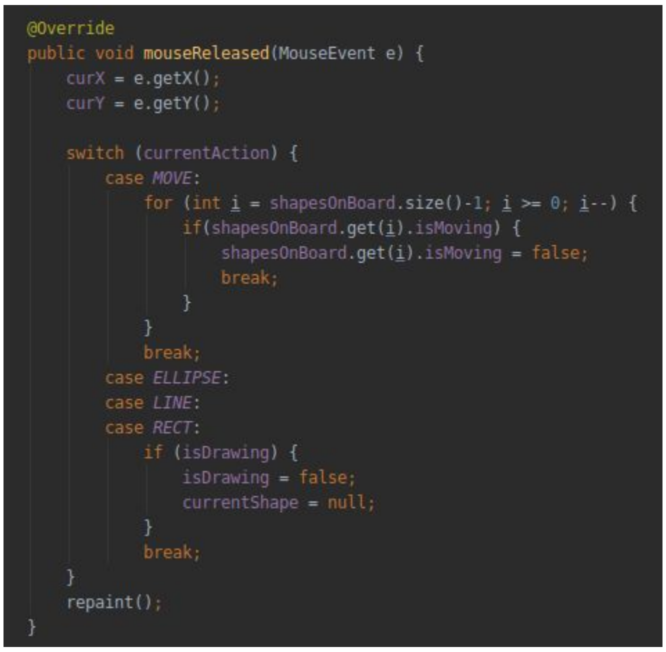


```java
    switch (currentAction) {
        case MOVE:
            if (id != -1) shapesOnBoard.get(id).isMoving = true;
            break;
        case RECT:
            currentShape = new shapes.Rectangle(e.getX(), e.getY(),  width: 1,  height: 1, curColor, shapesOnBoard.size(),isFill);
            isResizing = true;
            isDrawing = true;
            shapesOnBoard.add(currentShape);
            break;
        case ELLIPSE:
            currentShape = new Ellipse(e.getX(), e.getY(),  width: 1,  height: 1, curColor, shapesOnBoard.size(),isFill);
            isResizing = true;
            isDrawing = true;
            shapesOnBoard.add(currentShape);
            break;
        case LINE:
            currentShape = new Line(e.getX(), e.getY(),  width: 1,  height: 1, curColor, shapesOnBoard.size(), dash_interval: 5);
            isResizing = true;
            isDrawing = true;
            shapesOnBoard.add(currentShape);
            break;
    }
    repaint();
}
```

One of the 4 most important functions is mousePressed, which actually make the job done is mousePressed function. When user click somewhere, it takes the coordinates and creates there shape, depending on which tool is active on the moment of pressing the mouse and it waits until user release the mouse to create a final shape. And finally, it adds the created shape into the shapesOnBoard ArrayList of type Shape.

mouseReleased:

As I've mentioned above, when we press the mouse, a shape is created, but it waits for releasing the mouse and until we do so, the shape continues resizing. When we release the mouse the function mouseReleased takes the coordinates. In case we moving the shape, it sets the isMoving property to false to stop the shape moving.
In case of shapes themselves (ellipse,line,rectangle), it stops to draw the shapes. And at the end it calls repaint() function which let us continue our work on this app.



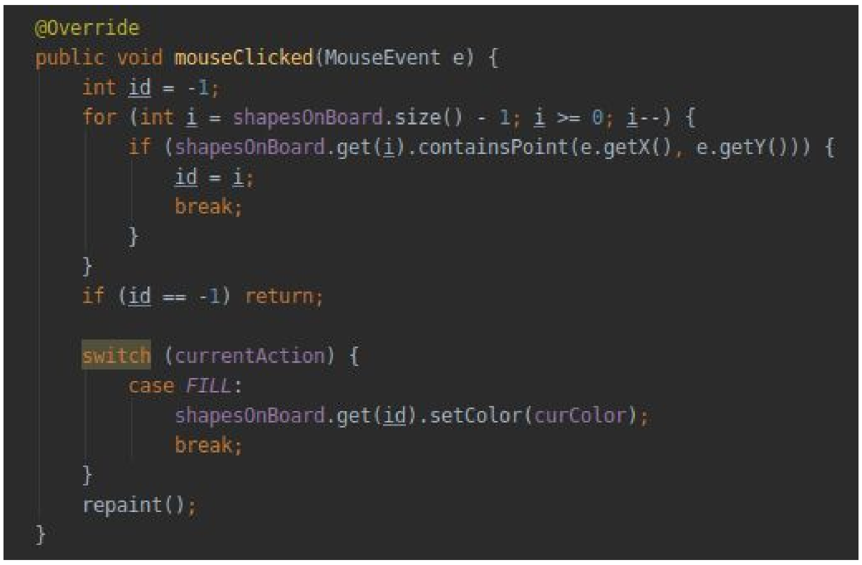
```java
@Override
public void mouseReleased(MouseEvent e) {
    curX = e.getX();
    curY = e.getY();

    switch (currentAction) {
        case MOVE:
            for (int i = shapesOnBoard.size()-1; i >= 0; i--) {
                if(shapesOnBoard.get(i).isMoving) {
                    shapesOnBoard.get(i).isMoving = false;
                    break;
                }
            }
            break;
        case ELLIPSE:
        case LINE:
        case RECT:
            if (isDrawing) {
                isDrawing = false;
                currentShape = null;
            }
            break;
    }
    repaint();
}
```

mouseClicked:

mouseClicked function serves only to fill the shapes and it the coordinates when we click somewhere and if this coordinates are inside of some shape the function takes the id of this shape and set it new color.

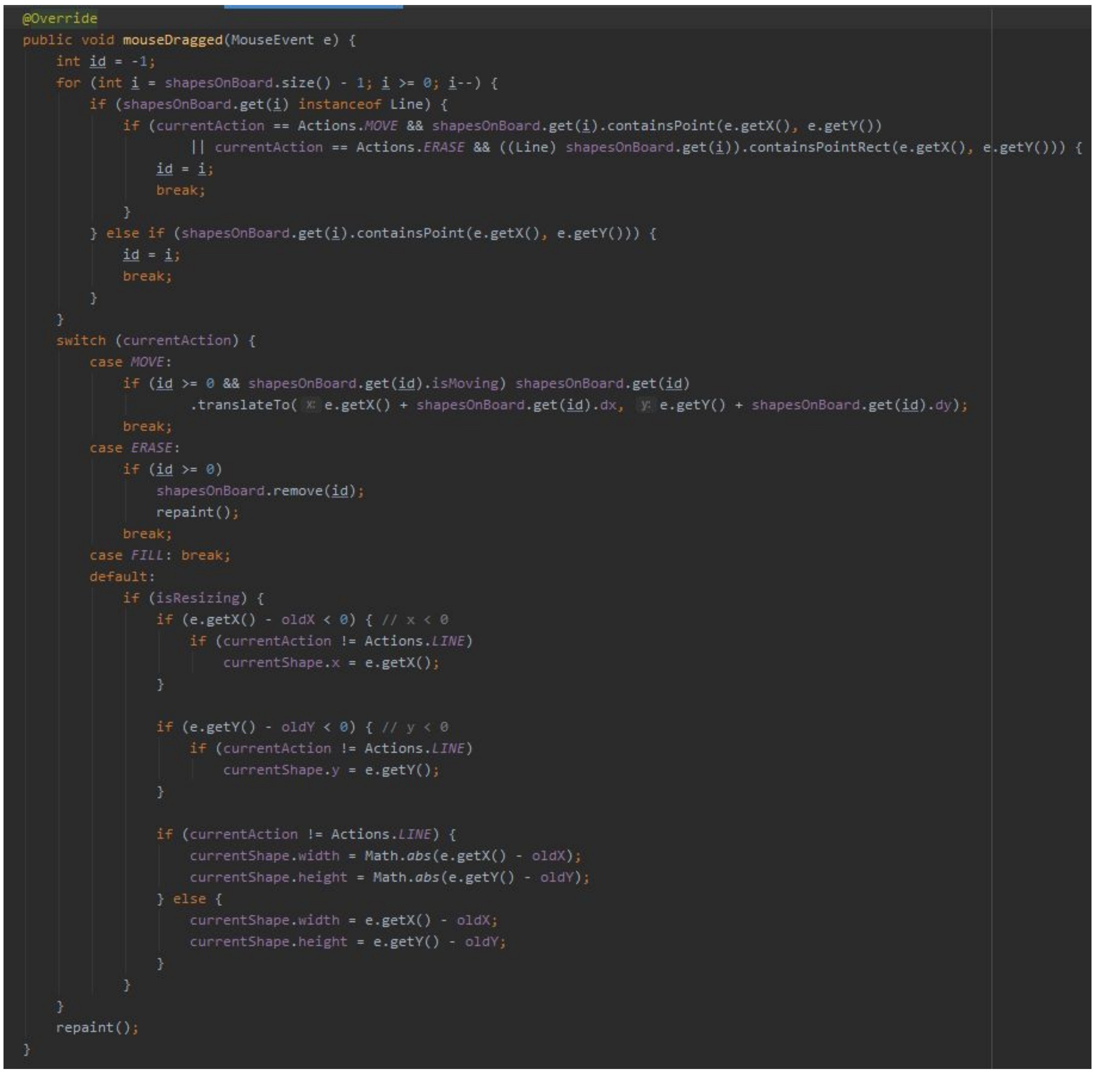
Each shape has its own id and in the shape abstract class there is containsPoint to find whether the coordinates are inside the shape.



```java
@Override
public void mouseClicked(MouseEvent e) {
    int id = -1;
    for (int i = shapesOnBoard.size() - 1; i >= 0; i--) {
        if (shapesOnBoard.get(i).containsPoint(e.getX(), e.getY())) {
            id = i;
            break;
        }
    }
    if (id == -1) return;

    switch (currentAction) {
        case FILL:
            shapesOnBoard.get(id).setColor(curColor);
            break;
    }
    repaint();
}
```

mouseDragged:



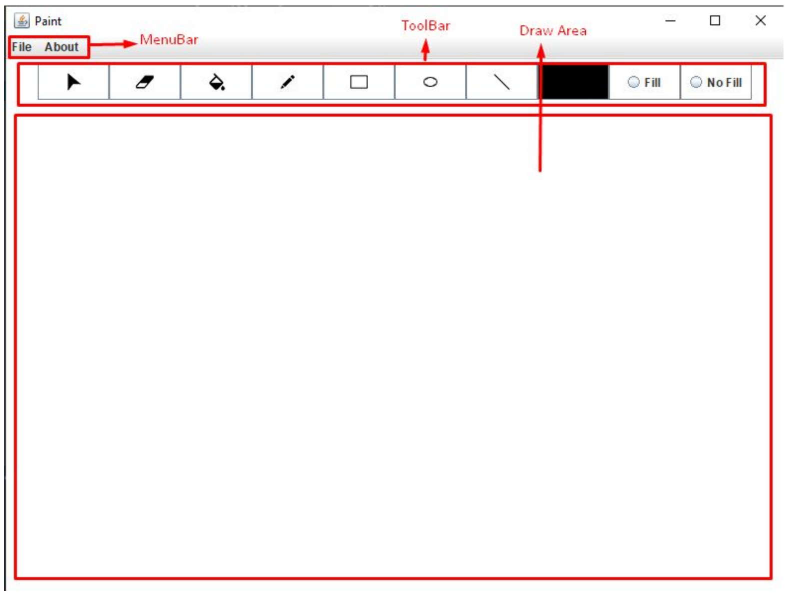
```java
@Override
public void mouseDragged(MouseEvent e) {
    int id = -1;
    for (int i = shapesOnBoard.size() - 1; i >= 0; i--) {
        if (shapesOnBoard.get(i) instanceof Line) {
            if (currentAction == Actions.MOVE && shapesOnBoard.get(i).containsPoint(e.getX(), e.getY())
                    || currentAction == Actions.ERASE && ((Line) shapesOnBoard.get(i)).containsPointRect(e.getX(), e.getY())) {
                id = i;
                break;
            }
        } else if (shapesOnBoard.get(i).containsPoint(e.getX(), e.getY())) {
            id = i;
            break;
        }
    }
    switch (currentAction) {
        case MOVE:
            if (id >= 0 && shapesOnBoard.get(id).isMoving) shapesOnBoard.get(id)
                    .translateTo( x: e.getX() + shapesOnBoard.get(id).dx,  y: e.getY() + shapesOnBoard.get(id).dy);
            break;
        case ERASE:
            if (id >= 0)
                shapesOnBoard.remove(id);
                repaint();
            break;
        case FILL: break;
        default:
            if (isResizing) {
                if (e.getX() - oldX < 0) { // x < 0
                    if (currentAction != Actions.LINE)
                        currentShape.x = e.getX();
                }

                if (e.getY() - oldY < 0) { // y < 0
                    if (currentAction != Actions.LINE)
                        currentShape.y = e.getY();
                }

                if (currentAction != Actions.LINE) {
                    currentShape.width = Math.abs(e.getX() - oldX);
                    currentShape.height = Math.abs(e.getY() - oldY);
                } else {
                    currentShape.width = e.getX() - oldX;
                    currentShape.height = e.getY() - oldY;
                }
            }
    }
    repaint();
}
```

One of the most important functions, with which we had several problems, but solved finally is mouseDragged function. It takes the coordinates and id of shape which contains these coordinates. Then in the case of Move tool, it takes the shape and moves it while we drag the mouse until we release the mouse. In the case of the erase tool, while we drag the mouse it removes every shape that we select. The main problem was with shapes. Because, if we drag the mouse to left and up the height and width of shapes were negative and shapes weren't shown in the board. And we solved this problem by taking the width and the height in the abs() function of the difference of the current coordinates and the initial coordinates which we took when we pressed the mouse. abs() function is of Math class of Java. But we take abs() of difference only when we had negative height or width. In other cases, we just take the difference.

Interface Specifications:



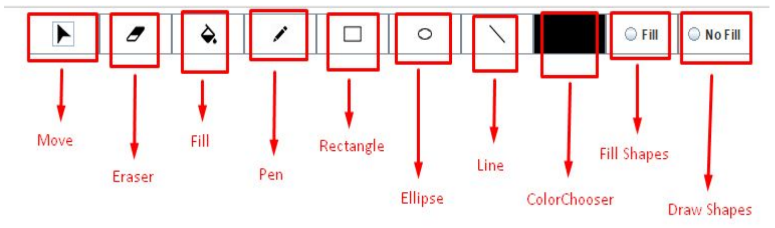
This app is very simple to use. The main window consists of 3 panels:

- MenuBar
- ToolBar
- DrawArea

In the menuBar you can 4 actions:

- Create new project (new)
- Open a new project
- Save a project
- Exit from the app

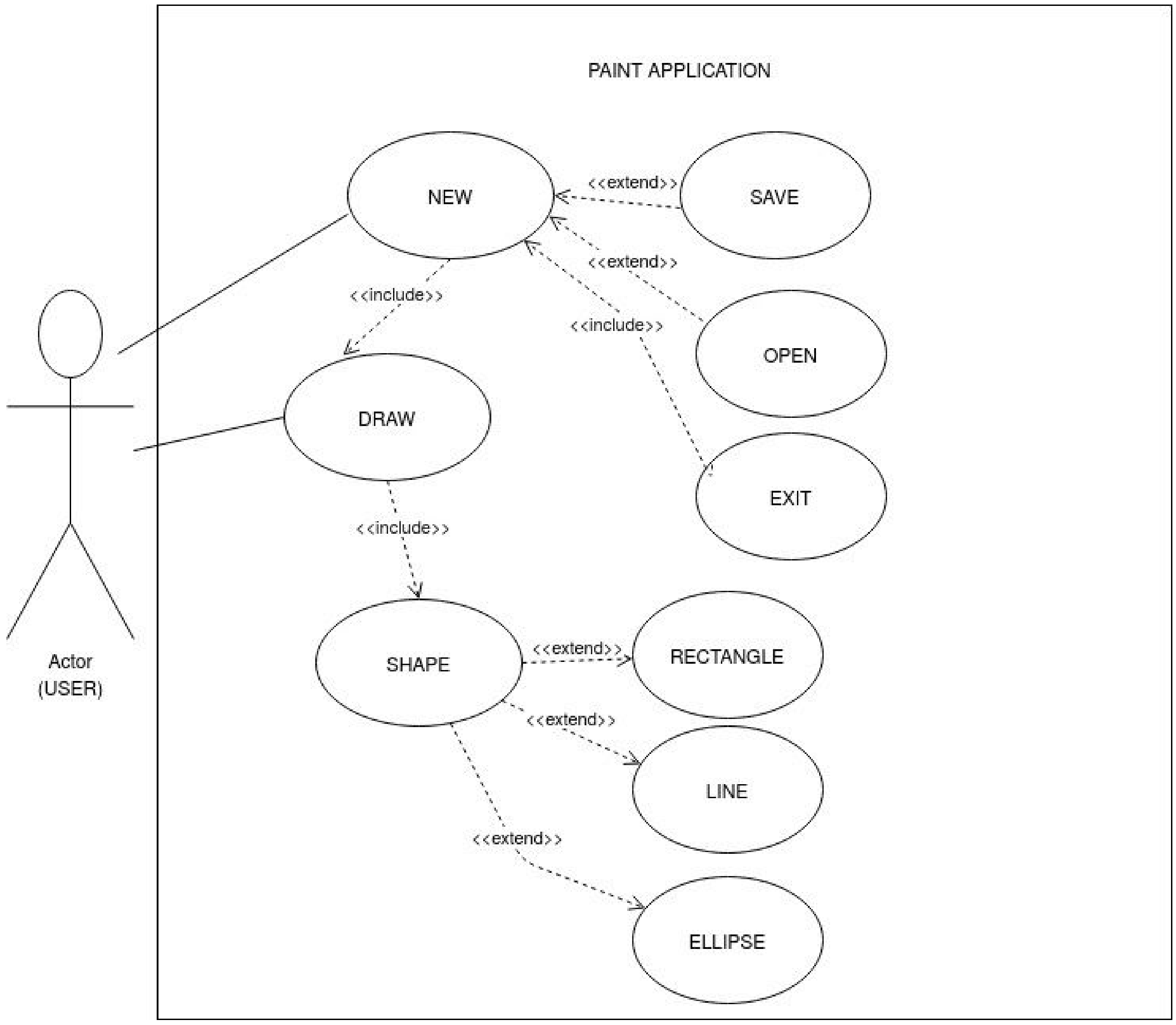
The toolbar consists of 10 panels itself:



by clicking on rectangle, line, ellipse you can draw them on the board. Initially, they will not be filled with any color, because NoFill radioButton is selected initially, and the color of shapes initially will be black. But you can change the color from colorChooser by clicking on its button and choosing the color. And if you want to fill the shapes you should choose fill radio button at the right of toolBar. by clicking on the move button and choosing the shape on the board you can move them. by clicking on the eraser button and by dragging the mouse on the board you can erase the shapes drawn on the board. By clicking on the fill button you can fill the shapes with color or change the color of the shapes initially filled by another color. The ColorChooser is opened on a new window to not flood the program itself with many swatches that it has to offer.

The interface is made very simple so that user could easily use this app.

Use cases:



Extend is used when a use case conditionally adds steps to another first-class use case. For example, imagine "Withdraw Cash" is a use case of an ATM machine. "Assess Fee" would extend Withdraw Cash and describe the conditional "extension point" that is instantiated when the ATM user doesn't bank at the ATM's owning institution. Notice that the basic "Withdraw Cash" use case stands on its own, without the extension. Include is used to extract use case fragments that are duplicated in multiple use cases.

In the case of extend, we can either have or not have these methods or classes, they simply extend our program (or class) and our program would be up and running with or without them.

What extend programmatically means (some class in Java extends another class) is that we have some parent class which has some classes and attributes that are common for other several classes (child class). These children classes simply extend from that parent class meaning that they include all the variables and the functions of that parent class in themselves for the use.

The included use case cannot stand alone and the original use case is not complete without the included one. This should be used sparingly only in cases where the duplication is significant and exists by design (rather than by coincidence). For example, the flow of events that occurs at the beginning of every ATM use case (when the user puts in their ATM card, enters their PIN, and is shown the main menu) would be a good candidate for an include.

In the case of included, it means that our program cannot function fully without these methods or classes. For example, our program cannot fully work or work at all without DRAW because that's what our Paint program mainly does.

Preliminary and Detailed Design:

Our project consists of one main src folder and two packages which are 'shapes' and 'resources.' The shapes package includes abstract class Shape which is the parent for all of our classes. Its inheritants are Rectangle, Line, and Ellipse. That is all for the shapes package.

The resources package consists of all of our icons for the buttons on the toolbar which are cursor, ellipse, rectangle, line, eraser, and fill.

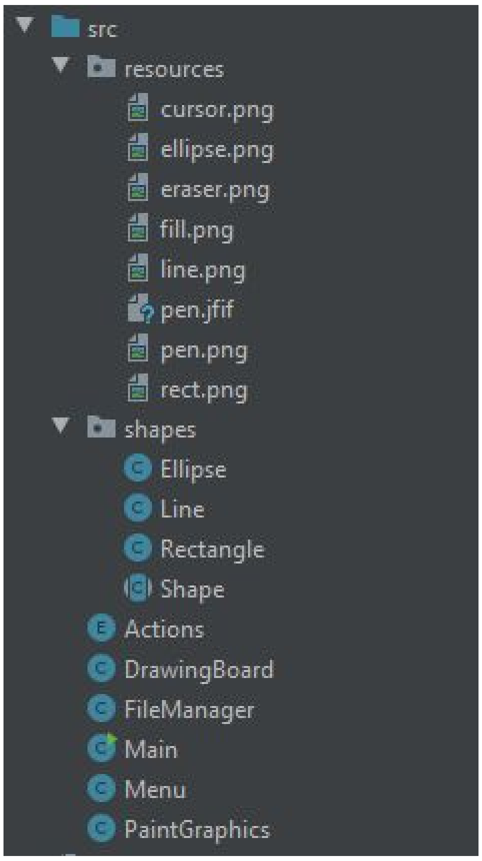
The main folder of the project which is src includes all the main classes of the project which draw and display the frame of the project, as well as the file manager which allows us to save images and load them from the memory. The exact classes are Actions, Drawing Board, FileManager, Main, Menu, PaintGraphics.

In the picture below you can observe all the packages and the classes of the project.

It is also notable that not all files are just normal classes, for example, the Actions file is not a class file but an Enum that enumerates all the actions made in the program such as MOVE, ERASE, FILL, RECT, ELLIPSE, LINE.

And, as mentioned before, the Shapes file includes the *abstract* class Shapes.

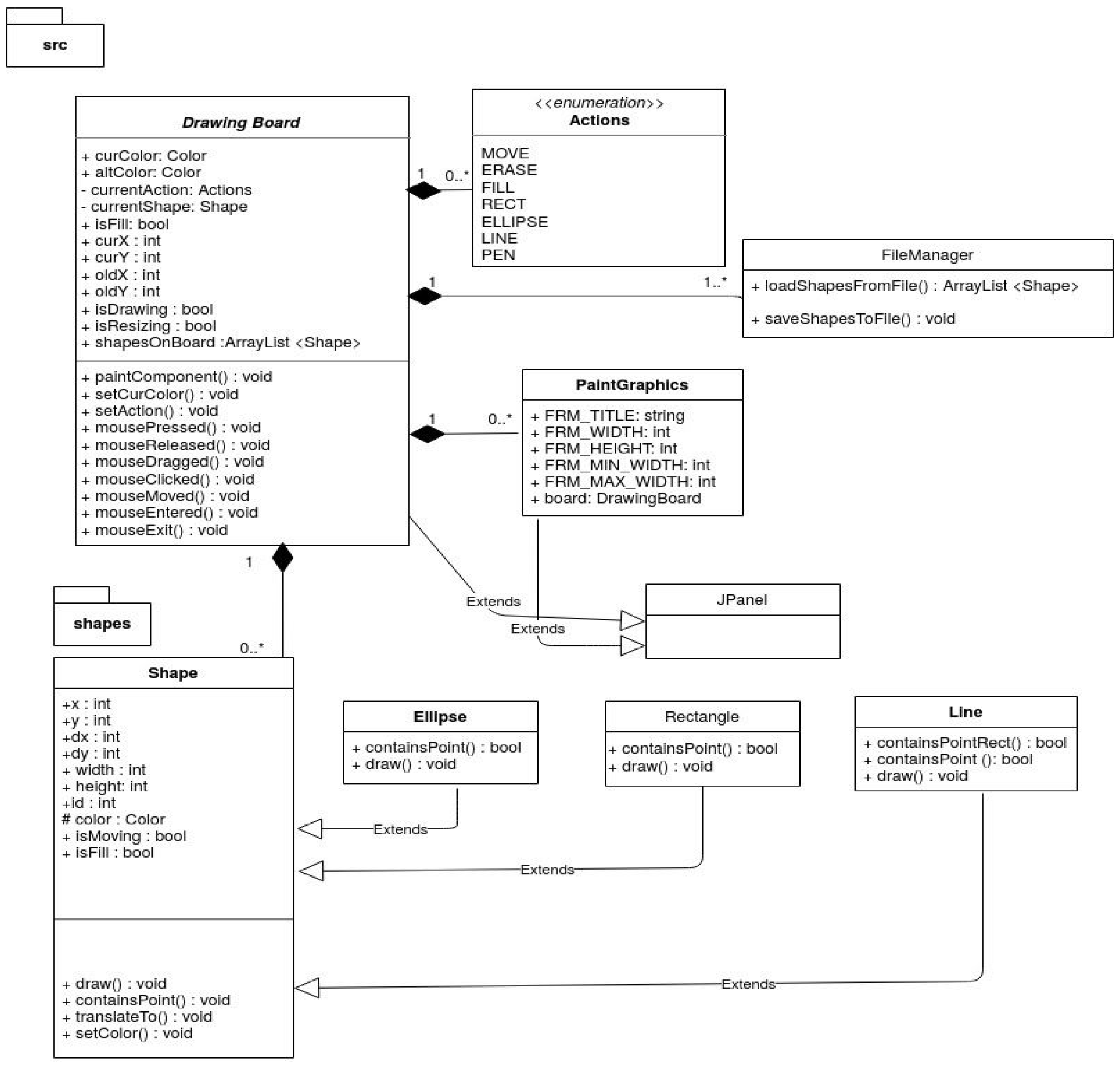
The Main class does nothing but creates the instance of the PaintGraphics which creates all the panels and the tools that exist in the project. We think it's better to leave Main empty because it should not do anything but start the program.



```
src
  resources
    cursor.png
    ellipse.png
    eraser.png
    fill.png
    line.png
    pen.jfif
    pen.png
    rect.png
  shapes
    Ellipse
    Line
    Rectangle
    Shape
  Actions
  DrawingBoard
  FileManager
  Main
  Menu
  PaintGraphics
```

As designated in the UML class diagram below, our classes have some relationships. For example, the Rectangle, Line, and Ellipse classes extend from the Shape class which consists of all the main parts of those classes. It includes the position, the width and the height of the shapes as well as their color and id. Each shape have an access to draw() , containsPoint() , translateTo() and setColor() methods. But Ellipse, Rectangle, Line classes need only draw() and containsPoint() methods to be implemented.

As is shown above in the class diagram, our project has also Actions class of type enumeration, which consists of tools we will use in the app: move, erase, fill, rect, ellipse, line, pen.



The End:

Thank you for taking the time to read this document. That is all for our project. We made it simple to use and we structured it also in a simple way to make it easier for future updates (if they would actually happen).