COMP 2502 HCI Homework I Report

by Yağız Efe Atasever – 19COMP1035

In this report, I will try to show how I did the requested homework step by step with early phases of coding. I will divide this report to a few parts. The parts would be;

- Making of the base panel and buttons
- Making of the Shapes
- Making of the Movement
- Making of the "Erase"

1.Making of the base panel and buttons

In the first part of the project, I needed a window for the application. So I used **javax.swing** package and JFrame class which is extended by **java.awt.frame.**

```
public class Paint_Demo extends JFrame{

    public Paint_Demo(){
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setSize(600,800);
        setLocationRelativeTo(this);
        setLayout(null);

        JPanel back = new JPanel();
        back.setSize(getSize());
        back.setBackground(Color.WHITE);
        back.setLayout(null);
        back.setLocation(0,0);
```

Shortly after writing the code, I realized It wouldn't work. Because I would have some shapes on the base window and If I write the base this way, I won't be able to move or do anything to the shapes. So I came up with another idea, which is using the **JLayeredPane** class. Using this class allows me to create shapes as JPanels and adding this JPanels to the base JLayeredPane. In this way, I could move objects without getting into trouble and making a mess with my code.

Below you can see the final code, with some extra stuff like JButtons and a JLabel. They all will become clear as I move forward in the report.

(To be clear I should mention that I defined the "base" object outside of the constructor)

```
public MsPaintKiller(){

    setDefaultCloseOperation(EXIT_ON_CLOSE);//when pressed "x", program shuts down.
    setSize(1366,768);
    setExtendedState(JFrame.MAXIMIZED_BOTH);//tam ekran başlatır

    base = new JLayeredPane();

    setVisible(true);//pencereyi görünür yapıyor
    base.setSize(getSize());
    base.setLayout(null);//this thing allows ME to do the positioning of the buttons, not any layout manager.
    add(base);
```

And than using the same package and in the package using the **JButton** class, I created some buttons and added them on my base. Below, you can see my erase button's code.

```
buttonEraser = new JButton("ERASE");
buttonEraser.setBounds(50,100,95,30);
buttonEraser.setFocusable(false);
buttonEraser.addActionListener(listener);
buttonEraser.setLocation(600,10);
buttonEraser.setBackground(Color.white);
base.add(buttonEraser);
```

I wanted to buttons look "clicked" when we click on them, so I added a little color changer for buttons. When you click a button It will be green from now on. It is a long if-else code, so I will show below just one part of the code but you can take a deep look in the project itself

```
public static ActionListener listener = new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            if(e.getSource() == buttonSquare){
                if(choice != 1){
                    choice = 1;
                    buttonSquare.setBackground(Color.green);
                    buttonTriangle.setBackground(Color.white);
                    buttonCircle.setBackground(Color.white);
                    buttonLineH.setBackground(Color.white);
                    buttonLineV.setBackground(Color.white);
                    buttonEraser.setBackground(Color.white);
                }else if (choice == 1) {
                    choice = 0;
                    buttonSquare.setBackground(Color.white);
                }
            }else if(e.getSource() == buttonTriangle){
                if(choice != 2){
```

And I should mention that I used **ActionListener** class in order to get input from the user to click the buttons. Other than that, if you look closely, you will be able to see **choice** variable. This choice variable determines which button pressed and which not. And as you now when we pressed buttons, they will turn green. But that is not the only thing they do.

2. Making of the Shapes

The **choice** variable takes us to making the shapes. I created another class called **Shapes** implemented **MouseListener** to it. Because I want to create shapes as JPanels as I mentioned before, and I want to create them at the tip of my cursor. In other words when I click, there will be a shape, that is what I want. So first of all I detected the mouse

```
public class Shapes implements MouseListener{
    static boolean isOn=false;
    MouseMovement hareket = new MouseMovement();
    static int id =1;

    public Shapes(Component panel){
        panel.addMouseListener(this);
    }

    @Override
    public void mouseClicked(MouseEvent e) {
        MouseMovement.movX = e.getX();
        MouseMovement.movY = e.getY();
```

Then the **choice** variable gets involved along with the buttons. The choice variable determines which shape the program draws. Below, you can see the square code

```
if(choice == 1){    //KARE
    isOn = false;
    JPanel panel = new JPanel();
    panel.setLocation(movX, movY);
    panel.setBackground(Color.blue);
    panel.setSize(50, 50);
    base.add(panel,++objectCount);
    hareket.addComp(panel);
    base.repaint();
    Delete listener = new Delete();
    panel.addMouseListener(listener);

}
```

So, I can draw shapes when I click the corresponding button and then clicking any empty space. But I can't move or erase them. So lets see movement

3. Making of the Movement

I created a class called **MouseMovement,** and most of the movement codes takes place in here. This class also implements **MouseListener** along with **MouseMotionListener** class.

```
public class MouseMovement implements MouseListener,MouseMotionListener{

    public static int oldX,oldY;
    public static int movX,movY;

    public MouseMovement(){
    }

    public MouseMovement(Component... pns){
        for(Component panel : pns){
            panel.addMouseListener(this);
            panel.addMouseMotionListener(this);
        }
    }
```

And then to detect clicking to any free space I wrote this:

```
@Override
public void mousePressed(MouseEvent e) {
    movX = e.getX();
    movY = e.getY();
}
```

We can move on to the last part.

4. Making of the "Erase"

To be honest that has been the most tricky part for me. Even now it is not fully functional. I created shapes as JPanels, and I can move them, that is nice but somehow I need to delete them. When I think I came up with the idea, that i need to detect the click of the mouse, on the specific shape, specific JPanel. So that is the code below. I created a class called **delete,** implemented **MouseListener** class and wrote the code.

But after all that, It works with a giant flaw. It only works when you click any empty space once, after you clicked the “Erase” button. You created some shapes and want to delete them, you clicked the “Erase” button and than clicked the shape you want to delete. It won’t work sadly. You have to click some empty are without moving your mouse, than you can click any shape to delete.

Like I said it works, but it has a giant flaw. At the end of the day, who hasn’t flaws right?

```java
@Override
public void mousePressed(MouseEvent e) {
    Object source = e.getSource();
        if(source instanceof JPanel){
            JPanel panelPressed = (JPanel) source;
            if (isOn == true) {
                base.remove(panelPressed);
                base.repaint();
            }
        }
}
```

SOURCE

https://www.codegrepper.com/code-examples/java/how+to+detect+if+someone+clicks+on+a+jpanel+in+java

https://stackoverflow.com/questions/35299786/draw-circle-on-jpanel-after-mouse-click/35300018

https://www.delftstack.com/howto/java/java-draw-triangle/#:~:text=awt%20%2C%20javax.-,swing%20and%20drawPolygon%20to%20Draw%20a%20Triangle%20in%20Java,on%20the%20Graphics%20object%20g%20

https://docs.oracle.com/javase/tutorial/uiswing/events/mousemotionlistener.html

https://www.youtube.com/watch?v=AxmHMfDnIAk