Obligatorisk uppgift 3 - 1DL028

Alhassan Jawad

May 3, 2024

Information

Name: Alhassan Jawad
Contact:

- Alhassan.Jawad.9989@student.uu.se
- Alhassan.Jawad@it.uu.se

1 Introduction

In this report, I will explain and summarize how I solved the 3rd Assignment of the the course *Object Oriented Programming with Java*. I will even explain how to use the package *LekMedGeometri* and how to add figures to the package and thus show it on the GUI.

2 Organisation & Structure of the Code

Using event-driven programming, standard libraries, and object-oriented programming concepts, the program shows how to create a well-structured and executed solution. The intended functionality is achieved and the design choices are driven by sound reasoning.

The program is organized into three main classes:

- Main Class (RunMe.java)
  - Responsible for the program flow
  - Creates instances of *Rectangle*, *Circle*, and *Triangle* classes
  - Creates the frame, adds the objects to it and makes it visible
- Shape Classes (Rectangle.java, Circle.java, Triangle.java)
  - Represents the specific shapes with specific properties and behaviours
  - Makes usage of *JPanel* to enable graphic drawings
  - Implements *MouseListener* & *MouseMotionListener* to handle different mouse movement events

2.1 Most important Methods of each class

Although the given shape code files (Rectangle.java, Circle.java, Triangle.java) uses distinct classes for each shape, the most crucial functions are the same for all of these types:

- Get methods for positions and sizes: These methods enable the retrieval of the width and height (for Rectangle and Triangle) or radius of the form (for Circle), in addition to its X and Y coordinates.
- Mouse event handling: The methods make use of features connected to the MouseListener and MouseMotionListener interfaces.

The primary functions shared by all of these shape classes are essentially getting access to the form's position and size as well as reacting to mouse events in response to user input/action.

3 Initial Approach

My initial approach to solving this problem involved additional files beyond the ones included in the final submission. Among them was *DrawingPanel.java* that calls to the various shape classes, which was meant to handle creating the entire panel. In an attempt to streamline the process of coding the solution, I started by concentrating on applying one shape at a time. The rectangle was initially selected because of its more straightforward visual representation. I successfully added the rectangle to the DrawingPanel.java class and got it to appear when the mouse entered the frame. The Circle and Triangle classes were subsequently developed and entered into the *DrawingPanel.java* file.

However, this initial approach led to a problem. While the shape objects themselves were instantiated and invoked within *DrawingPanel.java* the *mouseListener* and *mouseMotionListener* methods were implemented in the main *RunMe.java* file. A conflict arose from this separation: references to the individual forms were initialized in *DrawingPanel.java* but the animation logic needed to be in the *RunMe.java*file. The decision to eliminate *DrawingPanel.java* from the final solution was finally made due to its redundancy and the confusion it made for me. Using *RunMe.java*, the shape objects are now instantiated and modified directly in their respective files, simplifying the code and removing any potential conflicts.

4 Motivation for how the code looks

The *RunMe.java* file starts the program's execution. This file serves as the entry point and is in charge of configuring the *GUI*. It generates a *JFrame* object and sets its other properties and dimensions.

RunMe.java then invokes each of the distinct shape classes when the frame has been created. These classes, which I assume are found in the same package as *RunMe.java*, stand for the various shapes that will be shown, which for now are *Rectangle*, *Circle*, and *Triangle*.

The shapes are then hidden once an internal flag (*setShouldDraw*) is originally set to false. Lastly, each form object is added to the *JFrame* object that was previously generated, thereby putting them inside the visible portion of the frame.

The current code structure prioritizes ease of navigation and future extensibility.

- Simplified Navigation & modification: By having all the shape creation and management logic within *RunMe.java*, users can easily add new shapes. This centralized location eliminates the need to search through multiple files. This is made upon the assumption that the user have their own proper shape.java class file and that it is located within the same folder/-package.

4.1 Good to know

4.1.1 Note 1

Within the individual shape files (e.g. *Rectangle.java*) exist a method *paintImmediately(...)*. This code snippet:

```
public void paintImmediately(int x, int y, int radius) {
    super.paintImmediately(x, y, radius, radius);
}
```

is was added because I had some trouble with getting the respective shape (in my case the rectangle) to show up at all times when its being moved without being overshadowed by the frames background color. If you want, you can comment out the triangle and circle declarations in the *RunMe.java* code and then run the code without having the *paintImmediately* function called under *mouseDragged()* and you will see the problem.

For the same reason and to make the drawing more smoother, the line:

```
JFrame.setDefaultLookAndFeelDecorated(true);
```

was added to the *RunMe.java* code.

4.1.2 Note 2

If you look at the *Triangle.java* code, I added 2 extra arguments to the constructor calling the *rectangle* and *circle* shapes. This I did lastly as a DIY solution to the mouse movement events overwriting each other. What I mean is the following:

- When the *RunMe.java* code is run the *rectangle* is created so its *mouseEntered* event handler will show the rectangle when the mouse enters the frame.
- Afterwards the circle is created which will have its own *mouseEntered* and other event handlers that will rewrite the ones that the rectangle had.
- So I solved the issue by calling all the previous figures as arguments to the current figure so that there is a reference for the previous figures to use inside the appropriate events handlers.

5 How to add new figures to the code

- Before you add a figure you need to make sure that you have a file with the shape that you want to add, the file needs to have methods for extracting its different coordinates and then it needs to implement the mouse event methods and modify them if needed.
- Let us say that we want to add a figure named *Kvadrat* and that we already have the appropriate file *Kvadrat.java* with the needed methods.
  1. First we need to initialize the shape and create an instance of it:

```
Kvadra kvadrat = new Kvadrat(10, 10, 100);
```

  2. Secondly, we need to decide whether to show the figure from the start or not using:

```
rectangle.setShouldDraw(false);
```

  3. Afterwards, we add the shape to the frame using:

```
frame.add(rectangle);
```

- In accordance with note 2 of section 4; subsection 4.1.2, we need to change the constructor of the *Kvadrat* shape so that it calls upon all previous shapes implemented in the *RunMe.java* file. This will be a huge headache when having a dozen shapes, so a better solution needs to be found and implemented as soon as possible!

6 Changes made after feedback from the teacher

6.1 Complete remake due to previous misunderstanding

Prior to writing this, I misunderstood a part of the assignment description that said

"…. Du ska alltså inte låta figurerna bli en del av Swing-hierarkin. De klasser som representerar bilderna ska alltså inte ärva av någon klass i Swing-biblioteket…."

and made the figure.java files (circle.java, rectangle.java and triangle.java) inherit methods from the swing library.

So for revision nr. 1, I rewrote the figure.java files so that they do not inherit anything from the swing library and simply draw themselves using the paintComponent() method. If we look at the circle.java file then I made the class so that it only takes the x, y and radius parameters (by creating a constructor that only accepts the wanted parameters):

```
public Circle (int x0, int y0, int r) {
CenterX = x0;
CenterY = y0;
radius = r; }
```

Afterwards, I made a method to draw the figure using:

```
public void draw(Graphics g) {
g.setColor(Color.RED);
g.fillOval(CenterX-radius, CenterY-radius, radius*2, radius*2); }
```

and 2 methods that moves the circle and checks if the mouse click is withing its boundaries:

```
public void move(int x1, int y1) {
CenterX = x1;
CenterY = y1; }

public boolean containsCircle(int x, int y) {
return (CenterX-x)*(CenterX-x)+(CenterY-y)*(CenterY-y) <=
    radius*radius; }
```

The same thing was done for the rectangle and triangle figures with minor modifications shown in the references section of the report.

Regarding where to create the drawing window and where to create the figures and start listening for clues on if the mouse is clicked or pressed etc., I brought back the script *DrawingPanel.java* and there I created the figures (Circle + Rectangle + Triangle) and then I created various methods and a buffer to move and find the shapes to be moved around and animated. For this I followed the bullet Points on the course webpage that said how:

- Förflyttningar ska vara animerade, dvs användaren ska se hur figuren flyttar på sig när han/hon drar musen.
- När en figur flyttas hamnar den "överst" så att den täcker andra figurer som ligger i överlappande position.
- När en figur flyttas gäller att den "översta" väljs; om flera figurer täcker muspekarens position väljs den översta.

The following code lstlistings shows how I e.g. created the circle with

```
private Square square = new Square(60, 60, 40, 40);
```

, added it to an array with shapes using:

```
shapes.add(circle);
```

, and finally wrote methods (shown in References) of how I found the object to move, removed or moved it etc.

I also wrote a *Listener* java class responsible for getting the coordinates of the object when the mouse is pressed or dragged and send it to the respective method of DrawingPanel.java. I tried to have this code snippet in the same DrawingPanel code but then I encountered problems where it wanted to have its own file with a main class belonging to it.

6.2 Main.java

For the main part of the code, meaning the script that needs to be run to start the program, I just wrote a simple public class that extends a JFrame, sets the windows visibility and boundaries and then opens it by calling:

```
public static void main(String[] argv) {
    new Main();
}
```

This way I completed all the requirements of the assignment, no figure is depended or inherits from the swing library and I got a working solution.

References

Circle.java

```
import java.awt.Color;
import java.awt.Graphics;

public class Circle
{
    private int CenterX, CenterY;
    private int radius;

    public Circle(int x0, int y0, int r) {
        CenterX = x0;
        CenterY = y0;
        radius = r; }

    public void draw(Graphics g) {
        g.setColor(Color.RED);
        g.fillOval(CenterX-radius, CenterY-radius, radius*2, radius*2); }

    public void move(int x1, int y1) {
        CenterX = x1;
        CenterY = y1; }


    // Checks if the mouse click is within the circle
    public boolean containsCircle(int x, int y) {
        return (CenterX-x)*(CenterX-x)+(CenterY-y)*(CenterY-y) <=
            radius*radius; }
}
```

Rectangle.java

```
import java.awt.Color;
import java.awt.Graphics;

public class Square
{
    private int x, y;
    private int width, height;

    public Square(int x0, int y0, int w, int h) {
        x = x0;
        y = y0;
        width = w;
        height = h; }

    public void draw(Graphics g) {
        g.setColor(Color.GREEN);
        g.fillRect(x, y, width, height); }

    public void move(int x1, int y1) {
        x = x1;
        y = y1; }

    // Checks if the mouse click is within the square
    public boolean containsSquare(int x1, int y1) {
        return x <= x1 && y <= y1 && x1 <= x + width && y1 <= y +
            height; }
}
```

Triangle.java

```
import java.awt.Color;
import java.awt.Graphics;


public class Triangle
{
    private int[] points_X;
    private int[] points_Y;

    private int TopX; // TopX is the X-component located at the top of
        the triangle
    private int TopY; // TopY is the Y-component located at the top of
        the triangle
    private int width, height;

    public Triangle(int x, int y, int w, int h) {
        TopX = x;
        TopY = y;
        width = w;
        height = h;

        points_X = new int[]{TopX, TopX + width/2, TopX - width/2}; //
            Array of x-coordinates
        points_Y = new int[]{TopY, TopY + height, TopY + height}; //
            Array of y-coordinates
    }

    public void draw(Graphics g) {
        g.setColor(Color.BLUE);
        g.fillPolygon(points_X, points_Y, 3); }
    public void move(int x1, int y1) {

        TopX = x1;
        TopY = y1;

        this.points_X[0] = x1;        // Index 0 is the x-coordinate of
            the top of the triangle
        this.points_Y[0] = y1;

        // Formulas for the x and y coordinates of the right and left
            corners of the triangle
        this.points_X[1] = x1 + width/2; // Index 1 is the x-coordinate
            of the right corner of the triangle
        this.points_Y[1] = y1 + height;
```

```
        // Formulas for the x and y coordinates of the right and left
            corners of the triangle
        this.points_X[2] = x1 - width/2; // Index 2 is the x-coordinate
            of the left corner of the triangle
        this.points_Y[2] = y1 + height;
    }

    // Checks if the mouse click is within the triangle
    public boolean containsTriangle(int x, int y) {
        int dx = x - TopX;
        int dy = y - TopY;
        return -dy <= 2*dx && 2*dx <= dy && dy <= height;
    }
}
```

DrawingPanel.java

```
import java.awt.*;
import javax.swing.*;
import java.util.*;

public class DrawingPanel extends JPanel
{
    private ArrayList <Object> shapes = new ArrayList<Object>();
    private Circle circle = new Circle(100, 100, 40); // Create a circle
        object
    private Rectangle rectangle = new Rectangle(60, 60, 40, 40); //
        Create a square object
    private Triangle triangle = new Triangle(30, 30, 60, 60); // Create
        a triangle object

    public DrawingPanel() {
        setBackground(Color.WHITE);

        Listener mouListener = new Listener(this);
        addMouseListener(mouListener);    // Add a mouse listener to the
            panel meaning that the panel listens to mouse events
        addMouseMotionListener(mouListener); // Add a mouse motion
            listener to the panel meaning that the panel listens to
            mouse motion events

        // Add the objects to the list using geoShapes.add(geoObject);
        shapes.add(circle);
        shapes.add(rectangle);
        shapes.add(triangle);
    }
    public void addGeoObject(Object o) {
        // This method adds a geometric object to the list of geometric
            objects
        shapes.add(o);
        Graphics g = getGraphics();
        // Check if the object is a circle, square or triangle
        if(o instanceof Rectangle) {
            // Cast the object to a square object
            Rectangle rectangle = (Rectangle)o;
            rectangle.draw(g); }
        else if(o instanceof Circle) {
            // Cast the object to a circle object
            Circle circle = (Circle)o;
            circle.draw(g); }
        else if(o instanceof Triangle) {
            // Cast the object to a triangle object
            Triangle triangle = (Triangle)o;
            triangle.draw(g); }}
```

```
// This method finds the object that was clicked on or pressed using
    the mouse (mouseClicked or mousePressed)
// It also takes into account that there may be multiple overlapping
    objects, for that I have created a buffer
// to function as a temporary list that stores all objects that meet
    the criteria.
// At the final stage the method returns the last object from the
    buffer thus making sure that the object that
// was drawn last is the one that is returned. This is done using
    the findGeoObject method!
public Object findGeoObject(int x, int y) {
    ArrayList<Object> buffer = new ArrayList<Object>(); // Buffer to
        save objects that meet the criteria

    for (Object obj : shapes) {
        if(obj instanceof Rectangle) {
            // If square object meets the criteria, add it to the
                buffer
            Rectangle rectangle = (Rectangle)obj;
            if (rectangle.containsSquare(x,y))
                buffer.add(rectangle); }
        else if(obj instanceof Circle) {
            // If circle object meets the criteria, add it to the
                buffer
            Circle circle = (Circle)obj;
            if (circle.containsCircle(x,y))
                buffer.add(circle); }
        else if(obj instanceof Triangle) {
            // If triangle object meets the criteria, add it to the
                buffer
            Triangle triangle = (Triangle)obj;
            if (triangle.containsTriangle(x,y))
                buffer.add(triangle); }
    }
    // If the buffer is empty, return null
    if(buffer.isEmpty()) {
        return null; }
    else {
        // Return the last object from the buffer
        Object lastObject = buffer.remove(buffer.size()-1);
        return lastObject; }
}
```

```
// This method removes the object from the list of geometric objects
// thus making it disappear from the panel and ensuring that it is
    no longer drawn
public boolean removeGeoObject(Object o) {
    boolean flag = shapes.remove(o);
    repaint();
    return flag; }

// This method moves the object that is being dragged by the mouse
// by listening to the mouseDragged event in real time. The method
    uses paintComponent
// to loop through the list of objects and draw the object's paint
    layer last.
// This ensures that the object is drawn last in the list making it
    the topmost object and making
// it seems as if its moving. This method is called frequently as
    the object's position is updated
// constantly during the move.
public void moveGeoObject(Object o, int x, int y) {
    if(o instanceof Rectangle) {
        // IF square object is moved by the mouse, swap the object to
            the last position in the list
        // before moving it. This ensures that the object is drawn
            last in the list making it the topmost object
        // and making it seems as if its moving.
        Rectangle rectangle = (Rectangle)o;
        swapObjectToLast(rectangle);
        rectangle.move(x, y);
        repaint(); }
    else if(o instanceof Circle) {
        // IF circle object is moved by the mouse, swap the object to
            the last position in the list
        // before moving it. This ensures that the object is drawn
            last in the list making it the topmost object
        // and making it seems as if its moving.
        Circle circle = (Circle)o;
        swapObjectToLast(circle);
        circle.move(x, y);
        repaint(); }
    else if(o instanceof Triangle) {
        // IF triangle object is moved by the mouse, swap the object
            to the last position in the list
        // before moving it. This ensures that the object is drawn
            last in the list making it the topmost object
        // and making it seems as if its moving.
        Triangle triangle = (Triangle)o;
        swapObjectToLast(triangle);
        triangle.move(x, y);
        repaint();
    }}
```

```
    // This method places the current object that is to be drawn with
        the mouse last in the list before it is drawn.
    public void swapObjectToLast(Object obj) {
        // Find the index of the object in the list
        // and add it to the last position in the list
        // before it is drawn. This ensures that the object
        // is drawn last in the list making it the topmost object
        // and making it seems as if its moving. End the operation by
            removing the object from its original position.
        int indexOfObj = shapes.indexOf(obj);
        shapes.add(obj);
        shapes.remove(indexOfObj);
    }

    // This method is called by the paintComponent method to draw the
        objects on the panel
    public void paintComponent (Graphics g) {
        super.paintComponent(g);
        for (Object o : shapes) {
            if(o instanceof Rectangle ) {
                // If the object is square then draw it
                Rectangle rectangle = (Rectangle)o;
                rectangle.draw(g); }
            else if(o instanceof Circle) {
                // If the object is circle then draw it
                Circle circle = (Circle)o;
                circle.draw(g); }
            else if(o instanceof Triangle ) {
                // If the object is triangle then draw it
                Triangle triangle = (Triangle)o;
                triangle.draw(g); }
        }
    }
}
```

Listener.java

```
import java.awt.event.*;

public class Listener extends MouseAdapter implements
    MouseMotionListener {
   private DrawingPanel drawPanel; // Responsible for drawing the
       geometric objects
   private Object chosenObject; // Object chosen by the user to be moved
       by pressing on it
   public Listener(DrawingPanel d) {drawPanel = d; }

   public void mousePressed(MouseEvent e) {
   int x = e.getX();
   int y = e.getY();
   chosenObject = drawPanel.findGeoObject(x, y); }

   public void mouseDragged(MouseEvent e) {
      int x = e.getX();
      int y = e.getY();
      // Checks if the object is not null meaning that the object is
          chosen
      if (chosenObject != null) {
         drawPanel.moveGeoObject(chosenObject, x, y);}
   }
}
```

Main.java

```
import javax.swing.JFrame;

public class Main extends JFrame {
   DrawingPanel drawPanel = new DrawingPanel();
   public Main() {
      add(drawPanel);
      setBounds(100, 100, 400, 400);
      setVisible(true);
      setDefaultCloseOperation(EXIT_ON_CLOSE); }

   // This is the main method that starts the program
   // by calling the JMain constructor
   public static void main(String[] argv) {new Main();}
}
```