patrycja ignaczak mykyta
makarov wojciech szczepaniak

Zoom IN and OUT PlotterSoft

POWP PlotterSoft

Patrycja Ignaczak

Wstęp

Cele i założenia

Celem naszego projektu było opracowanie i zaimplementowanie dodatkowej funkcjonalności do aplikacji Plottersoft. Do wykonania naszego zadanie wykorzystaliśmy wzorce projektowe i zasady projektowania w ujęciu obiektowym.

Zadanie

Problemem aplikacji jest brak możliwości zmiany parametrów wyświetlanej symulacji. Rysunki są czasem niewymiarowe, ponieważ niemożliwe jest dostosowania wielkości wydruku.
Oprócz możliwości powiększania i zmniejszania obrazu można uwzględnić także np..: przesuwania, obracania, rozciągania, odbijanie, itp.

Zalecenia odnośnie projektu

Ingerencja w hierarchię poleceń jest możliwa, ale nie jest zalecana, ponieważ przekształcenia mogłyby wpłynąć nie tylko na symulację.
Przydatny przydałby się mechanizm manualnej obsługi plotera.

Rozwiązanie

Użyte wzorce projektowe

Dekorator (Decorator)

```
package edu.iis.powp.command.manager.redrawable;

public class RedrawablePlotterDecorator implements IPlotter, Redrawable {

   private IPlotter instance;
   private PlotterCommandManager commandManager;


   public RedrawablePlotterDecorator(IPlotter instance, PlotterCommandManager
commandManager) {
          super();
          this.instance = instance;
          this.commandManager = commandManager;
   }

   @Override
   public void drawTo(int arg0, int arg1) {
          if(commandManager.getCurrentCommand() == null) {
                commandManager.setCurrentCommand(new DrawToCommand(arg0,arg1));
          } else {
                commandManager.setCurrentCommand(new ArrayList<IPlotterCommand>(){
                      private static final long serialVersionUID = 1L;

                {
                      add(commandManager.getCurrentCommand());
                      add(new DrawToCommand(arg0,arg1));
                }},
                            "Saved command");
          }
          instance.drawTo(arg0, arg1);
          System.out.println("drawto: " + arg0+ " " + arg1);
   }

   @Override
   public void setPosition(int arg0, int arg1) {
          if(commandManager.getCurrentCommand() == null) {
                commandManager.setCurrentCommand(new SetPositionCommand(arg0,arg1));
          } else {
                commandManager.setCurrentCommand(new ArrayList<IPlotterCommand>(){
```

```
                    private static final long serialVersionUID = 1L;

                {
                    add(commandManager.getCurrentCommand());
                    add(new SetPositionCommand(arg0,arg1));
                }},
                            "Saved command");
        }
        instance.setPosition(arg0, arg1);
        System.out.println("setpos: " + arg0+ " " + arg1);
    }

    @Override
    public boolean isRedrawable() {
        return true;
    }

}
```

Służy do zapisywania wszystkiego, co jest uruchamiane w iPlotter do klasy PlotterCommandManager. Interfejs Redrawable istnieje dlatego, że chcieliśmy zaimplementować sprawdzanie, czy pozwalamy na wywołanie redraw, ponieważ błędy z tym związane mogą być kosztowne. Jest to według nas propozycja rozwiązania tego problemu.

Strategia (Strategy)

```
package edu.iis.powp.modification.listeners;

import edu.iis.powp.modification.listeners.handlers.*;

import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class ModifyButtonListener implements ActionListener {
    ModificationMouseAdapter mouseAdapter;

    public ModifyButtonListener(ModificationMouseAdapter mouseAdapter) {
        super();
        this.mouseAdapter = mouseAdapter;
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        JComponent source = (JComponent)e.getSource();
        switch(source.getName()) {
        case "moveButton":
            mouseAdapter.setHandler(new TranslationMouseHandler());
            break;
        case "rotateButton":
            mouseAdapter.setHandler(new RotationMouseHandler());
```

```
			break;
		case "scaleButton":
			mouseAdapter.setHandler(new ScaleMouseHandler());
			break;
		case "stretchButton":
			mouseAdapter.setHandler(new StretchMouseHandler());
			break;
		case "PointerButton":
			mouseAdapter.setHandler(new PointerMouseHandler());
			break;
		}

	}

}
```

Ten listener zmienia stany przetwarzania sygnałów od myszki w zależności od ustawionego handlera, tj. pozwala nam na manipulowanie obrazem tylko przy użyciu myszki..

Adapter (Adapter)

```
package edu.iis.powp.modification.listeners;

import java.awt.event.MouseEvent;
import javax.swing.event.MouseInputAdapter;

import edu.iis.powp.appext.FeaturesManager;
import edu.iis.powp.modification.listeners.handlers.ModificationMouseHandler;
import edu.iis.powp.plot.modification.PlotModification;
import edu.iis.powp.plot.modification.PlotModifier;


public class ModificationMouseAdapter extends MouseInputAdapter {
   protected ModificationMouseHandler handler;
   protected MouseEvent previous;
     protected int prev_x = 0;
     protected int prev_y = 0;

   public ModificationMouseAdapter() {
         super();
   }

   @Override
   public void mousePressed(MouseEvent e) {
         if(FeaturesManager.getDriverManager().getCurrentPlotter() instanceof
PlotModifier) {

   ((PlotModifier)FeaturesManager.getDriverManager().getCurrentPlotter())
               .addModification(handler.getModification());
         }
         previous = e;
         handler.update(previous, e);
```

```
            System.out.println("pressed " + handler.getClass().getName());
        }

    @Override
        public void mouseDragged(MouseEvent e) {
            handler.update(previous, e);
            previous = e;
           FeaturesManager.reDraw();
        }

    @Override
        public void mouseReleased(MouseEvent e) {
            handler.update(previous, e);
            previous = e;
           FeaturesManager.reDraw();
        }

    public void setHandler(ModificationMouseHandler handler) {
            this.handler = handler;
    }

}
```

Odpowiada za uruchomienie handlerów, które już implementują działania z PlotModification

Rozwiązanie zadania

STRETCH

```
package edu.iis.powp.plot.modification;

/**
 * Do stretching a plot image.
 */
public class StretchPlotModification extends PlotModification {

    /**
     * X-axis stretching scale. If {@code 1.0}, then no stretching.
     */
    private float xAxis;

      /**
     * Y-axis stretching scale. If {@code 1.0}, then no stretching.
     */
    private float yAxis;

    public float getxAxis() {
            return xAxis;
      }

      public void setxAxis(float xAxis) {
            this.xAxis = xAxis;
      }

      public float getyAxis() {
            return yAxis;
      }

      public void setyAxis(float yAxis) {
            this.yAxis = yAxis;
      }


    /**
     * Constructor of {@link StretchPlotModification}.
     *
     * @param xAxis x-axis stretching scale. If {@code 1.0}, then no stretching.
     * @param yAxis y-axis stretching scale. If {@code 1.0}, then no stretching.
     */
    public StretchPlotModification(float xAxis, float yAxis) {
        if (xAxis < 0 || yAxis < 0) {
            throw new IllegalArgumentException("Stretching scale must be unsigned
value");
        }
        this.xAxis = xAxis;
        this.yAxis = yAxis;
    }

    /**
     * @see PlotModification#modify(PlotPoint)
     */
    @Override
```

```
    public void modify(PlotPoint point) {
        int x = point.getX() - plotModifier.getCenterPoint().getX();
        int y = point.getY() - plotModifier.getCenterPoint().getY();

        int x1 = Math.round((xAxis - 1) * x);
        int y1 = Math.round((yAxis - 1) * y);
        point.add(x1, y1);
    }
}
```

ZOOM (scale)

```
package edu.iis.powp.plot.modification;

/**
 * Move points to change a scale of plotted image used the center point of
translation storied in {@link PlotModifier}.
 */
public class ScalePlotModification extends PlotModification {

    /**
     * Instance of similar plot modification - {@link StretchPlotModification}.
     */
    private final StretchPlotModification stretchPlotModification;

    /**
     * Scale of plot
     */
    private float scale;

    public float getScale() {
            return scale;
      }

      public void setScale(float scale) {
            this.scale = scale;
            stretchPlotModification.setxAxis(scale);
            stretchPlotModification.setyAxis(scale);
      }

      /**
     * Constructor of {@link ScalePlotModification}.
     *
     * @param scale scale of plot.
     */
    public ScalePlotModification(float scale) {
        this.scale = scale;
        this.stretchPlotModification = new StretchPlotModification(scale, scale);
    }

    @Override
    public void setUp(PlotModifier plotModifier) {
        super.setUp(plotModifier);
        stretchPlotModification.setUp(plotModifier);
    }

    /**
     * @see PlotModification#modify(PlotPoint)
     */
    @Override
    public void modify(PlotPoint point) {
        stretchPlotModification.modify(point);
    }
}
```

TRANSFORMATION

```
package edu.iis.powp.plot.modification;

/**
 * Do changing a center point of drawing and translate point using new position.
 */
public class TranslationPlotModification extends PlotModification {

    /**
     * New center point of translation.
     */
    private PlotPoint point;

    public PlotPoint getPoint() {
            return point;
      }

      public void setPoint(PlotPoint point) {
            this.point = point;
      }

      /**
     * Constructor of {@link TranslationPlotModification}.
     *
     * @param x value on x-axis
     * @param y value of y-axis
     */
    public TranslationPlotModification(int x, int y) {
        point = new PlotPoint(x, y);
    }

    /**
     * @see PlotModification#setUp(PlotModifier)
     */
    @Override
    public void setUp(PlotModifier plotModifier) {
        plotModifier.setCenterPoint(point);
        super.setUp(plotModifier);
    }

    /**
     * @see PlotModification#modify(PlotPoint)
     */
    @Override
    public void modify(PlotPoint point) {
        plotModifier.setCenterPoint(this.point);
        point.add(plotModifier.getCenterPoint());
    }
}
```

ROTATION

```
package edu.iis.powp.plot.modification;

/**
 * Do rotate of points of plot with center point of translation storied in {@link
PlotModifier}.
 */
public class RotationPlotModification extends PlotModification {

    /**
     * Angle in degrees.
     */
    private int angle;

    public int getAngle() {
            return angle;
      }

      public void setAngle(int angle) {
            this.angle = angle;
      }

      /**
     * Constructor of {@link RotationPlotModification}.
     *
     * @param angle angle in degrees
     */
    public RotationPlotModification(int angle) {
        this.angle = angle;
    }

    @Override
    public void setUp(PlotModifier plotModifier) {
        super.setUp(plotModifier);
    }

    /**
     * @see PlotModification#modify(PlotPoint)
     */
    @Override
    public void modify(PlotPoint point) {
        int x = point.getX();
        int y = point.getY();
        int xu = plotModifier.getCenterPoint().getX();
        int yu = plotModifier.getCenterPoint().getY();
        double cos = Math.cos(Math.toRadians(angle));
        double sin = Math.sin(Math.toRadians(angle));

        point.setX((int) Math.round((x - xu) * cos - (y - yu) * sin + xu));
        point.setY((int) Math.round((x - xu) * sin + (y - yu) * cos + yu));
    }
}
```