**Міністерство освіти і науки України**  
**Національний технічний університет України**  
**«Київський політехнічний інститут імені Ігоря Сікорського»**  
**Факультет інформатики та обчислювальної техніки**  
**Кафедра обчислювальної техніки**

**Лабораторна робота №5**

з дисципліни  
«Об’єктно-орієнтоване програмування»

Виконав: Перевірив:

студент групи ІМ-31 Порєв В. М.  
Литвиненко Сергій Андрійович  
номер у списку групи: 11

Київ 2024

**Варіант завдання**

Singleton Маєрса

Файл Main.java.

import javafx.application.Application;

import javafx.scene.Scene;

import javafx.scene.layout.AnchorPane;

import javafx.scene.layout.BorderPane;

import javafx.stage.Stage;

import javafx.fxml.FXMLLoader;

import javafx.scene.control.ScrollPane;

public class Main extends Application {

private final String pathToView = "./resources/Main.fxml";

private final String pathToViewTable = "./resources/Table.fxml";

private final String titleMain = "Lab 5";

private final String titleTable = "Table";

private final double width = 900;

private final double height = 900;

static void main(String[] args) {

launch(args);

}

public Stage startTable() throws Exception {

final var stage = new Stage();

final ScrollPane root = FXMLLoader.load(getClass().getResource(pathToViewTable));

final var scene = new Scene(root);

stage.setScene(scene);

stage.setTitle(titleTable);

stage.setWidth(width);

stage.setHeight(height);

stage.show();

return stage;

}

@Override

public void start(Stage stage) throws Exception {

final BorderPane root = FXMLLoader.load(getClass().getResource(pathToView));

final Scene scene = new Scene(root);

final var pane = (AnchorPane)((BorderPane)root.getCenter()).getCenter();

stage.setScene(scene);

pane.setPrefWidth(width);

pane.setPrefHeight(height);

stage.setTitle(titleMain);

final var tableStage = startTable();

tableStage.setOnCloseRequest((\_) -> { stage.close(); });

stage.setOnCloseRequest((\_) -> { tableStage.close(); });

stage.show();

}

}

Файл resources/Main.fxml.

<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>

<?import javafx.scene.control.Menu?>

<?import javafx.scene.control.MenuBar?>

<?import javafx.scene.control.MenuItem?>

<?import javafx.scene.control.RadioMenuItem?>

<?import javafx.scene.control.ToolBar?>

<?import javafx.scene.control.Tooltip?>

<?import javafx.scene.image.Image?>

<?import javafx.scene.image.ImageView?>

<?import javafx.scene.layout.AnchorPane?>

<?import javafx.scene.layout.BorderPane?>

<?import javafx.scene.canvas.Canvas?>

<BorderPane fx:id="borderPane" maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" xmlns="http://javafx.com/javafx/22" xmlns:fx="http://javafx.com/fxml/1" fx:controller="controllers.MenuController">

<top>

<MenuBar id="menuBar" BorderPane.alignment="CENTER">

<menus>

<Menu mnemonicParsing="false" text="File">

<items>

<MenuItem mnemonicParsing="false" onAction="#saveAs" text="Save as..." />

<MenuItem mnemonicParsing="false" onAction="#open" text="Open..." />

<MenuItem mnemonicParsing="false" onAction="#exit" text="Close" />

</items>

</Menu>

<Menu fx:id="objectsMenu" mnemonicParsing="false" text="Objects">

<items>

<Menu mnemonicParsing="false" text="Rectangle">

<items>

<RadioMenuItem id="rectangleCenter" mnemonicParsing="false" text="From center" />

<RadioMenuItem id="rectangleCorner" mnemonicParsing="false" text="From corner" />

</items>

</Menu>

<Menu mnemonicParsing="false" text="Elipse">

<items>

<RadioMenuItem id="ellipseCenter" mnemonicParsing="false" text="From center" />

<RadioMenuItem id="ellipseCorner" mnemonicParsing="false" text="From corner" />

</items>

</Menu>

<RadioMenuItem id="cube" mnemonicParsing="false" text="Cube" />

<RadioMenuItem id="line" mnemonicParsing="false" text="Line" />

<RadioMenuItem id="line-ellipse" mnemonicParsing="false" text="Line Ellipse" />

<RadioMenuItem id="point" mnemonicParsing="false" text="Point" />

<RadioMenuItem id="brush" mnemonicParsing="false" text="Brush" />

</items>

</Menu>

<Menu mnemonicParsing="false" text="Reference">

<items>

<MenuItem mnemonicParsing="false" text="About" />

</items>

</Menu>

<Menu mnemonicParsing="false" text="Settings">

<items>

<Menu fx:id="colors" mnemonicParsing="false" onAction="#colors" text="Colors">

<items>

</items>

</Menu>

<RadioMenuItem mnemonicParsing="false" onAction="#fill" text="Fill" />

</items>

</Menu>

</menus>

</MenuBar>

</top>

<center>

<BorderPane BorderPane.alignment="CENTER">

<top>

<ToolBar BorderPane.alignment="CENTER" fx:id="toolBar">

<items>

<Button id="rectangleCenter-button" mnemonicParsing="false">

<graphic>

<ImageView fitHeight="32.0" fitWidth="32.0">

<image>

<Image url="@icons/rectangle-center.png" />

</image>

</ImageView>

</graphic>

<tooltip>

<Tooltip text="Rectangle Center" />

</tooltip>

</Button>

<Button id="rectangleCorner-button" mnemonicParsing="false">

<graphic>

<ImageView fitHeight="32.0" fitWidth="32.0">

<image>

<Image url="@icons/rectangle-corner.png" />

</image>

</ImageView>

</graphic>

<tooltip>

<Tooltip text="Rectangle Corner" />

</tooltip>

</Button>

<Button id="cube-button" mnemonicParsing="false">

<graphic>

<ImageView fitHeight="32.0" fitWidth="32.0">

<image>

<Image url="@icons/cube.png" />

</image>

</ImageView>

</graphic>

<tooltip>

<Tooltip text="Cube" />

</tooltip>

</Button>

<Button id="ellipseCenter-button" mnemonicParsing="false">

<graphic>

<ImageView fitHeight="32.0" fitWidth="32.0">

<image>

<Image url="@icons/ellipse-center.png" />

</image>

</ImageView>

</graphic>

<tooltip>

<Tooltip text="Ellipse Center" />

</tooltip>

</Button>

<Button id="ellipseCorner-button" mnemonicParsing="false">

<graphic>

<ImageView fitHeight="32.0" fitWidth="32.0">

<image>

<Image url="@icons/ellipse-corner.png" />

</image>

</ImageView>

</graphic>

<tooltip>

<Tooltip text="Elipce Corner" />

</tooltip>

</Button>

<Button id="line-button" mnemonicParsing="false">

<graphic>

<ImageView fitHeight="32.0" fitWidth="32.0">

<image>

<Image url="@icons/line.png" />

</image>

</ImageView>

</graphic>

<tooltip>

<Tooltip text="Line" />

</tooltip>

</Button>

<Button id="line-ellipse-button" mnemonicParsing="false">

<graphic>

<ImageView fitHeight="32.0" fitWidth="32.0">

<image>

<Image url="@icons/line-ellipse.png" />

</image>

</ImageView>

</graphic>

<tooltip>

<Tooltip text="Line ELlipse" />

</tooltip>

</Button>

<Button id="point-button" mnemonicParsing="false">

<graphic>

<ImageView fitHeight="32.0" fitWidth="32.0">

<image>

<Image url="@icons/point.png" />

</image>

</ImageView>

</graphic>

<tooltip>

<Tooltip text="Point" />

</tooltip>

</Button>

<Button id="brush-button" mnemonicParsing="false">

<graphic>

<ImageView fitHeight="32.0" fitWidth="32.0">

<image>

<Image url="@icons/brush.png" />

</image>

</ImageView>

</graphic>

<tooltip>

<Tooltip text="Brush" />

</tooltip>

</Button>

</items>

</ToolBar>

</top>

<center>

<AnchorPane fx:id="anchorPane" BorderPane.alignment="CENTER">

<Canvas fx:id="canvas" />

</AnchorPane>

</center>

</BorderPane>

</center>

</BorderPane>

Файл resources/Table.fxml

<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.ScrollPane?>

<?import javafx.scene.control.TableColumn?>

<?import javafx.scene.control.TableView?>

<?import javafx.scene.layout.VBox?>

<ScrollPane fx:id="scrollPane" maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" xmlns="http://javafx.com/javafx/22" xmlns:fx="http://javafx.com/fxml/1" fx:controller="controllers.TableController" fitToWidth="true" fitToHeight="true">

<content>

<VBox>

<children>

<TableView fx:id="tableView">

<columns>

<TableColumn fx:id="nameColumn" prefWidth="120.0" text="Name" />

<TableColumn fx:id="x1Column" prefWidth="90.0" text="x1" />

<TableColumn fx:id="y1Column" prefWidth="90.0" text="y1" />

<TableColumn fx:id="x2Column" prefWidth="90.0" text="x2" />

<TableColumn fx:id="y2Column" prefWidth="90.0" text="y2" />

</columns>

</TableView>

</children>

</VBox>

</content>

</ScrollPane>

Файл tableview/PointPair.java

package tableview;

import javafx.beans.property.SimpleDoubleProperty;

import javafx.beans.property.SimpleStringProperty;

public class PointPair {

private final SimpleStringProperty name;

private final SimpleDoubleProperty x1;

private final SimpleDoubleProperty y1;

private final SimpleDoubleProperty x2;

private final SimpleDoubleProperty y2;

public PointPair(String name, double x1, double y1, double x2, double y2) {

this.name = new SimpleStringProperty(name);

this.x1 = new SimpleDoubleProperty(x1);

this.y1 = new SimpleDoubleProperty(y1);

this.x2 = new SimpleDoubleProperty(x2);

this.y2 = new SimpleDoubleProperty(y2);

}

public PointPair setPoints(double x1, double y1, double x2, double y2) {

this.x1.set(x1);

this.y1.set(y1);

this.x2.set(x2);

this.y2.set(y2);

return this;

}

public SimpleStringProperty getName() { return name; }

public SimpleDoubleProperty getX1() { return x1; }

public SimpleDoubleProperty getY1() { return y1; }

public SimpleDoubleProperty getX2() { return x2; }

public SimpleDoubleProperty getY2() { return y2; }

}

Файл settings/Color.java

package settings;

import java.util.Map;

import java.util.Collection;

public class Color {

static private Color instance = new Color();

private javafx.scene.paint.Color currentColor = javafx.scene.paint.Color.BLACK;

private Map<String, javafx.scene.paint.Color> colors = Map.of(

"black", javafx.scene.paint.Color.BLACK,

"red", javafx.scene.paint.Color.RED,

"blue", javafx.scene.paint.Color.BLUE,

"green", javafx.scene.paint.Color.GREEN,

"yellow", javafx.scene.paint.Color.YELLOW,

"purple", javafx.scene.paint.Color.PURPLE,

"pink", javafx.scene.paint.Color.PINK,

"gold", javafx.scene.paint.Color.GOLD,

"brown", javafx.scene.paint.Color.BROWN,

"light blue", javafx.scene.paint.Color.LIGHTBLUE

);

public void setColor(final String color) {

if (!colors.containsKey(color)) return;

currentColor = colors.get(color);

}

public javafx.scene.paint.Color getCurrentColor() {

return currentColor;

}

public Collection<? extends String> getStringColors() {

return colors.keySet();

}

public Collection<? extends javafx.scene.paint.Color> getColors() {

return colors.values();

}

public static Color getInstance() {

return instance;

}

}

Файл settings/Fill.java

package settings;

public class Fill {

private static Fill instance = new Fill();

private boolean fill = false;

public boolean getFill() {

return fill;

}

public void setFill(final boolean flag) {

fill = flag;

}

public static Fill getInstance() {

return instance;

}

}

Файл shapes/Shape.java

package shapes;

import javafx.scene.canvas.GraphicsContext;

import javafx.scene.paint.Color;

import java.util.ArrayList;

import java.util.List;

import javafx.util.Pair;

public abstract class Shape {

protected List<Double> coords;

public Color color = Color.BLACK;

public boolean fill = false;

public double dashes = 0;

public boolean useDashes = true;

public Shape() {

this(new ArrayList<>(List.of(0.0, 0.0, 0.0, 0.0)));

}

public Shape(final List<Double> points) {

coords = points;

}

protected void prepareContext(final GraphicsContext context) {

context.setStroke(color);

context.setFill(color);

context.setLineDashes(dashes);

}

public abstract void draw(final GraphicsContext context);

public abstract void setCoords(double x1, double y1, double x2, double y2);

public void onStart(GraphicsContext context, double x, double y) {

}

public abstract Pair<Pair<Double, Double>, Pair<Double, Double>> getDisplayCoords();

public abstract String getName();

public List<Double> getCoords() {

return List.copyOf(coords);

}

}

Файл shapes/RectangleCorner.java

package shapes;

import javafx.scene.canvas.GraphicsContext;

import javafx.util.Pair;

import java.util.List;

public class RectangleCorner extends Shape implements Rectangable {

public RectangleCorner() {

super();

}

public RectangleCorner(final List<Double> coords) {

super(coords);

}

@Override

public void draw(GraphicsContext context) {

prepareContext(context);

Rectangable.super.drawRectangle(

context,

coords.get(0),

coords.get(1),

coords.get(2),

coords.get(3),

fill

);

}

@Override

public void setCoords(double x1, double y1, double x2, double y2) {

coords.set(0, Math.min(x1, x2));

coords.set(1, Math.min(y1, y2));

coords.set(2, Math.abs(x2 - x1));

coords.set(3, Math.abs(y2 - y1));

}

@Override

public Pair<Pair<Double, Double>, Pair<Double, Double>> getDisplayCoords() {

final var first = new Pair<>(coords.get(0), coords.get(1));

final var second = new Pair<>(coords.get(2), coords.get(3));

return new Pair<>(first, second);

}

@Override

public String getName() {

return "Rectangle";

}

}

Файл shapes/RectangleCenter.java

package shapes;

public class RectangleCenter extends RectangleCorner {

@Override

public void setCoords(double x1, double y1, double x2, double y2) {

super.setCoords(2 \* x1 - x2, 2 \* y1 - y2, x2, y2);

}

}

Файл shapes/Rectangable.java

package shapes;

import javafx.scene.canvas.GraphicsContext;

public interface Rectangable {

default void drawRectangle(GraphicsContext context, double x, double y, double dx, double dy, boolean fill) {

final var width = context.getLineWidth();

if (fill) context.fillRect(x, y, dx + width, dy + width);

else context.strokeRect(x, y, dx + width, dy + width);

}

}

Файл shapes/Point.java

package shapes;

import java.util.ArrayList;

import java.util.List;

import javafx.scene.canvas.GraphicsContext;

import javafx.util.Pair;

public class Point extends Shape {

public Point() {

this(new ArrayList<Double>(List.of(0.0, 0.0)));

}

public Point(final List<Double> coords) {

super(coords);

}

@Override

public void onStart(GraphicsContext context, double x, double y) {

this.setCoords(0, 0, x, y);

this.draw(context);

}

@Override

public void draw(GraphicsContext context) {

prepareContext(context);

final var x = coords.get(0);

final var y = coords.get(1);

final var width = context.getLineWidth();

context.fillOval(x - width, y - width, width \* 2, width \* 2);

}

@Override

public void setCoords(double x1, double y1, double x2, double y2) {

coords.set(0, x2);

coords.set(1, y2);

}

@Override

public Pair<Pair<Double, Double>, Pair<Double, Double>> getDisplayCoords() {

final var x = coords.get(0);

final var y = coords.get(1);

final var point = new Pair<>(x, y);

return new Pair<>(point, point);

}

@Override

public String getName() {

return "Point";

}

}

Файл shapes/LineEllipse.java

package shapes;

import java.util.List;

import javafx.scene.canvas.GraphicsContext;

import javafx.util.Pair;

public class LineEllipse extends Shape implements Linable, Ellipsable {

final static int ellipseRadius = 20;

public LineEllipse() {

super();

}

public LineEllipse(final List<Double> coords) {

super(coords);

}

@Override

public void draw(GraphicsContext context) {

prepareContext(context);

final var x1 = coords.get(0);

final var y1 = coords.get(1);

final var x2 = coords.get(2);

final var y2 = coords.get(3);

final var dx = x2 - x1;

final var dy = y2 - y1;

final var angle = Math.atan2(dy, dx);

final var lineWidth = context.getLineWidth();

final var length = ellipseRadius / 2 + lineWidth;

Linable.super.drawLine(

context,

x1 + length \* Math.cos(angle),

y1 + length \* Math.sin(angle),

x2 + length \* Math.cos(Math.PI + angle),

y2 + length \* Math.sin(Math.PI + angle)

);

Ellipsable.super.drawEllipse(context,

x1 - ellipseRadius / 2,

y1 - ellipseRadius / 2,

ellipseRadius,

ellipseRadius,

fill

);

Ellipsable.super.drawEllipse(context,

x2 - ellipseRadius / 2,

y2 - ellipseRadius / 2,

ellipseRadius,

ellipseRadius,

fill

);

}

@Override

public void setCoords(double x1, double y1, double x2, double y2) {

coords.set(0, x1);

coords.set(1, y1);

coords.set(2, x2);

coords.set(3, y2);

}

@Override

public Pair<Pair<Double, Double>, Pair<Double, Double>> getDisplayCoords() {

final var first = new Pair<>(coords.get(0), coords.get(1));

final var second = new Pair<>(coords.get(2), coords.get(3));

return new Pair<>(first, second);

}

@Override

public String getName() {

return "LineEllipse";

}

}

Файл shapes/Line.java

package shapes;

import java.util.List;

import javafx.scene.canvas.GraphicsContext;

import javafx.util.Pair;

public class Line extends Shape implements Linable {

public Line() {

super();

}

public Line(final List<Double> coords) {

super(coords);

}

@Override

public void draw(GraphicsContext context) {

prepareContext(context);

Linable.super.drawLine(

context,

coords.get(0),

coords.get(1),

coords.get(2),

coords.get(3)

);

}

@Override

public void setCoords(double x1, double y1, double x2, double y2) {

coords.set(0, x1);

coords.set(1, y1);

coords.set(2, x2);

coords.set(3, y2);

}

@Override

public Pair<Pair<Double, Double>, Pair<Double, Double>> getDisplayCoords() {

final var first = new Pair<>(coords.get(0), coords.get(1));

final var second = new Pair<>(coords.get(2), coords.get(3));

return new Pair<>(first, second);

}

@Override

public String getName() {

return "Line";

}

}

Файл shapes/Linable.java

package shapes;

import javafx.scene.canvas.GraphicsContext;

public interface Linable {

public default void drawLine(GraphicsContext context, double x1, double y1, double x2, double y2) {

context.strokeLine(x1, y1, x2, y2);

}

}

Файл shapes/ElipseCorner.java

package shapes;

import java.util.List;

import javafx.scene.canvas.GraphicsContext;

import javafx.util.Pair;

public class EllipseCorner extends Shape implements Ellipsable {

public EllipseCorner() {

super();

}

public EllipseCorner(final List<Double> coords) {

super(coords);

}

@Override

public void draw(GraphicsContext context) {

prepareContext(context);

Ellipsable.super.drawEllipse(

context,

coords.get(0),

coords.get(1),

coords.get(2),

coords.get(3),

fill

);

}

@Override

public void setCoords(double x1, double y1, double x2, double y2) {

final double dx = Math.abs(x2 - x1);

final double dy = Math.abs(y2 - y1);

coords.set(0, (x1 + x2 - dx) / 2);

coords.set(1, (y1 + y2 - dy) / 2);

coords.set(2, dx);

coords.set(3, dy);

}

@Override

public Pair<Pair<Double, Double>, Pair<Double, Double>> getDisplayCoords() {

final var first = new Pair<>(coords.get(0), coords.get(1));

final var second = new Pair<>(coords.get(2), coords.get(3));

return new Pair<>(first, second);

}

@Override

public String getName() {

return "Ellipse";

}

}

Файл shapes/EllipseCenter.java

package shapes;

public class EllipseCenter extends EllipseCorner {

@Override

public void setCoords(double x1, double y1, double x2, double y2) {

super.setCoords(2 \* x1 - x2, 2 \* y1 - y2, x2, y2);

}

}

Файд shapes/Ellipsable.java

package shapes;

import javafx.scene.canvas.GraphicsContext;

public interface Ellipsable {

public default void drawEllipse(GraphicsContext context, double x, double y, double dx, double dy, boolean fill) {

final var width = context.getLineWidth();

if (fill) context.fillOval(x, y, dx + width, dy + width);

else context.strokeOval(x, y, dx + width, dy + width);

}

}

Файл shapes/Cube.java

package shapes;

import java.util.List;

import javafx.scene.canvas.GraphicsContext;

import javafx.util.Pair;

public class Cube extends Shape implements Linable, Rectangable {

private static final int deltaX = 50;

private static final int deltaY = -40;

public Cube() {

super();

}

public Cube(final List<Double> coords) {

super(coords);

}

@Override

public void draw(GraphicsContext context) {

prepareContext(context);

fill = false;

final var x1 = coords.get(0);

final var y1 = coords.get(1);

final var dx = coords.get(2);

final var dy = coords.get(3);

Rectangable.super.drawRectangle(context, x1, y1, dx, dy, fill);

Rectangable.super.drawRectangle(context, x1 + deltaX, y1 + deltaY, dx, dy, fill);

Linable.super.drawLine(context, x1, y1, x1 + deltaX, y1 + deltaY);

Linable.super.drawLine(context, x1 + dx, y1, x1 + dx + deltaX, y1 + deltaY);

Linable.super.drawLine(context, x1, y1 + dy, x1 + deltaX, y1 + dy + deltaY);

Linable.super.drawLine(context, x1 + dx, y1 + dy, x1 + dx + deltaX, y1 + dy + deltaY);

}

@Override

public void setCoords(double x1, double y1, double x2, double y2) {

coords.set(0, Math.min(x1, x2));

coords.set(1, Math.min(y1, y2));

coords.set(2, Math.abs(x2 - x1));

coords.set(3, Math.abs(y2 - y1));

}

@Override

public Pair<Pair<Double, Double>, Pair<Double, Double>> getDisplayCoords() {

final var first = new Pair<>(coords.get(0), coords.get(1));

final var second = new Pair<>(coords.get(2), coords.get(3));

return new Pair<>(first, second);

}

@Override

public String getName() {

return "Cube";

}

}

Файл editors/Brush.java

package shapes;

import java.util.ArrayList;

import java.util.List;

import javafx.scene.canvas.GraphicsContext;

import javafx.util.Pair;

public class Brush extends Shape {

public Brush(final List<Double> coords) {

super(coords);

useDashes = false;

}

public Brush() {

this(new ArrayList<>());

}

@Override

public void onStart(GraphicsContext context, double x, double y) {

this.setCoords(0, 0, x, y);

}

@Override

public void draw(GraphicsContext context) {

prepareContext(context);

final var size = coords.size();

if (size <= 2) return;

var prevX = coords.get(0);

var prevY = coords.get(1);

for (int i = 2; i < size; i += 2) {

final var x = coords.get(i);

final var y = coords.get(i + 1);

context.strokeLine(prevX, prevY, x, y);

prevX = x;

prevY = y;

}

}

@Override

public void setCoords(double x1, double y1, double x2, double y2) {

coords.add(x2);

coords.add(y2);

}

@Override

public Pair<Pair<Double, Double>, Pair<Double, Double>> getDisplayCoords() {

final var x1 = coords.get(0);

final var y1 = coords.get(1);

final var x2 = coords.get(coords.size() - 2);

final var y2 = coords.get(coords.size() - 1);

final var first = new Pair<>(x1, y1);

final var second = new Pair<>(x2, y2);

return new Pair<>(first, second);

}

@Override

public String getName() {

return "Brush";

}

}

Файл editors/Editor.java

package editors;

import shapes.Shape;

import java.util.List;

import java.util.Map;

import java.util.ArrayList;

import java.util.HashMap;

import javafx.scene.canvas.Canvas;

import javafx.scene.canvas.GraphicsContext;

import settings.Color;

import settings.Fill;

import java.util.function.Consumer;

public class Editor {

private static final double lineDashes = 10;

private double startX = 0;

private double startY = 0;

private boolean drawing = false;

private List<Shape> shapes = new ArrayList<Shape>();

private Canvas canvas;

private GraphicsContext context;

private Map<String, List<Consumer<Shape>>> listeners = new HashMap<>();

private static Editor instance = null;

public Editor setCanvas(final Canvas canvas) {

this.canvas = canvas;

context = canvas.getGraphicsContext2D();

canvas.widthProperty().addListener((\_) -> {

clear();

drawAll();

});

canvas.heightProperty().addListener((\_) -> {

clear();

drawAll();

});

return this;

}

public static Editor getInstance() {

instance = instance == null ? new Editor() : instance;

return instance;

}

private void redraw() {

clear();

drawAll();

}

private void drawAll() {

for (final var shape: shapes) shape.draw(context);

}

private void clear() {

context.clearRect(0, 0, canvas.getWidth(), canvas.getHeight());

}

public void add(final Shape shape) {

shapes.add(shape);

}

public void addToCanvas(final Shape shape) {

this.add(shape);

redraw();

this.emit("create", shape);

}

public void pop() {

if (shapes.size() == 0) return;

final var shape = shapes.removeLast();

redraw();

this.emit("delete", shape);

}

public void onLeftButtonDown(double x, double y) {

startX = x;

startY = y;

final var shape = shapes.getLast();

shape.dashes = shape.useDashes ? lineDashes : 0;

shape.color = Color.getInstance().getCurrentColor();

shape.fill = Fill.getInstance().getFill();

shape.onStart(context, x, y);

}

public void onMouseMove(double x, double y) {

if (drawing) clear();

else drawing = true;

shapes.getLast().setCoords(startX, startY, x, y);

drawAll();

}

public void onLeftButtonUp(double x, double y) {

clear();

final var shape = shapes.getLast();

shape.setCoords(startX, startY, x, y);

shape.dashes = 0;

drawAll();

drawing = false;

emit("create", shape);

}

public Editor on(final String eventName, final Consumer<Shape> listener) {

final var exists = listeners.containsKey(eventName);

if (exists) listeners.get(eventName).add(listener);

else listeners.put(eventName, List.of(listener));

return this;

}

public Editor emit(final String eventName, final Shape shape) {

final var exists = listeners.containsKey(eventName);

if (!exists) return this;

for (final var listener: listeners.get(eventName)) {

listener.accept(shape);

}

return this;

}

public Editor reset() {

for (final var shape: shapes) emit("delete", shape);

shapes.clear();

clear();

return this;

}

public List<Shape> shapes() {

return List.copyOf(shapes);

}

}

Файл controllers/MenuController.java

package controllers;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.scene.layout.AnchorPane;

import javafx.scene.layout.BorderPane;

import javafx.stage.FileChooser;

import javafx.stage.Stage;

import javafx.scene.canvas.Canvas;

import javafx.scene.control.Button;

import javafx.scene.control.Menu;

import javafx.scene.control.RadioMenuItem;

import javafx.scene.control.ToolBar;

import javafx.scene.input.MouseButton;

import javafx.scene.input.MouseEvent;

import javafx.scene.control.MenuItem;

import javafx.application.Platform;

import javafx.scene.input.KeyCode;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.FileWriter;

import java.io.IOException;

import java.nio.file.Files;

import java.util.ArrayList;

import java.util.List;

import settings.Color;

import settings.Fill;

import shapes.\*;

import java.util.Map;

import editors.Editor;

public class MenuController {

@FXML

private BorderPane borderPane;

@FXML

private Menu objectsMenu;

@FXML

private AnchorPane anchorPane;

@FXML

private Menu colors;

@FXML

private RadioMenuItem lastSelected = null;

@FXML

private Canvas canvas;

@FXML

private ToolBar toolBar;

private final Map<String, Class<? extends Shape>> editors = Map.of(

"rectangleCenter", RectangleCenter.class,

"rectangleCorner", RectangleCorner.class,

"ellipseCenter", EllipseCenter.class,

"ellipseCorner", EllipseCorner.class,

"line", Line.class,

"point", Point.class,

"brush", Brush.class,

"line-ellipse", LineEllipse.class,

"cube", Cube.class

);

private final Map<String, Class<? extends Shape>> shapes = Map.of(

"Brush", Brush.class,

"Cube", Cube.class,

"Ellipse", EllipseCorner.class,

"Line", Line.class,

"LineEllipse", LineEllipse.class,

"Point", Point.class,

"Rectangle", RectangleCorner.class

);

private boolean isPrimary(final MouseEvent event) {

return event.getButton().equals(MouseButton.PRIMARY);

}

private void processEvent(final Shape shape, final RadioMenuItem item) {

final var editor = Editor.getInstance();

anchorPane.setOnMousePressed((event) -> {

if (isPrimary(event) && item.isSelected()) {

editor.add(shape);

editor.onLeftButtonDown(event.getX(), event.getY());

}

});

anchorPane.setOnMouseDragged((event) -> {

if (isPrimary(event) && item.isSelected()) {

editor.onMouseMove(event.getX(), event.getY());

}

});

anchorPane.setOnMouseReleased((event) -> {

if (isPrimary(event) && item.isSelected()) {

editor.onLeftButtonUp(event.getX(), event.getY());

processEvent(getShape(item.getId()), item);

}

});

}

@FXML

private void exit() {

Platform.exit();

}

@FXML

private void saveAs() throws IOException {

final var stage = (Stage)borderPane.getScene().getWindow();

final var savefile = new FileChooser();

savefile.setTitle("Save File");

final var file = savefile.showSaveDialog(stage);

if (file == null) return;

final var filewriter = new FileWriter(file, false);

try (BufferedWriter writer = new BufferedWriter(filewriter)) {

final var shapes = Editor.getInstance().shapes();

for (final var shape: shapes) {

writer.write(shape.getName() + " ");

final var coords = shape.getCoords();

for (final var coord: coords) {

writer.write(String.valueOf(coord) + " ");

}

writer.newLine();

}

} catch (IOException e) {

e.printStackTrace();

}

}

@FXML

private void colors(final ActionEvent event) {

final var item = (MenuItem)event.getTarget();

final var text = item.getText();

Color.getInstance().setColor(text);

}

@FXML

private void fill() {

final var fill = Fill.getInstance().getFill();

Fill.getInstance().setFill(!fill);

}

private void drawShape(String name, final List<Double> coords) {

final var exists = shapes.containsKey(name);

if (!exists) return;

final var constructor = shapes.get(name);

try {

final var declared = constructor.getDeclaredConstructor(List.class);

final var shape = declared.newInstance(coords);

Editor.getInstance().addToCanvas(shape);

} catch (Exception e) {

e.printStackTrace();

}

}

@FXML

private void open() {

final var stage = (Stage)borderPane.getScene().getWindow();

final var fileChooser = new FileChooser();

final var extention = new FileChooser.ExtensionFilter("Text Files", "\*.txt");

fileChooser.getExtensionFilters().add(extention);

final var file = fileChooser.showOpenDialog(stage);

if (file == null) return;

try (BufferedReader reader = Files.newBufferedReader(file.toPath())) {

Editor.getInstance().reset();

while (true) {

final var line = reader.readLine();

if (line == null) return;

final var columns = line.split("\\s+");

final var name = columns[0];

final var numbers = new ArrayList<Double>();

for (int index = 1; index < columns.length; index++) {

final var column = columns[index];

final var number = Double.parseDouble(column);

numbers.add(number);

}

drawShape(name, numbers);

}

} catch (Exception e) {

e.printStackTrace();

}

}

private void addColors() {

final var items = new ArrayList<MenuItem>();

for (final var color: Color.getInstance().getStringColors()) {

items.addLast(new MenuItem(color));

}

colors.getItems().addAll(items);

}

private Shape getShape(final String id) {

final var constructor = editors.get(id);

try {

final var declared = constructor.getDeclaredConstructor();

final var shape = declared.newInstance();

return shape;

} catch (Exception e) {

e.printStackTrace();

return null;

}

}

@SuppressWarnings("unused")

private void addItemsEvenets(final Menu root) {

for (final var item: root.getItems()) {

if (item instanceof Menu menu) {

addItemsEvenets(menu);

continue;

}

final var selected = (RadioMenuItem)item;

final var fullPath = getFullName(selected, objectsMenu);

item.setOnAction((event) -> {

if (lastSelected != null) lastSelected.setSelected(false);

selected.setSelected(true);

lastSelected = selected;

final var window = (Stage)borderPane.getScene().getWindow();

window.setTitle(fullPath);

final var shape = getShape(selected.getId());

processEvent(shape, selected);

});

final var buttonId = selected.getId() + "-button";

final var button = (Button)toolBar.getItems().filtered((node) -> {

return node.getId().equals(buttonId);

}).getFirst();

button.setOnAction((event) -> item.fire());

}

}

@FXML

private void initialize() {

canvas.widthProperty().bind(anchorPane.widthProperty());

canvas.heightProperty().bind(anchorPane.heightProperty());

addColors();

addItemsEvenets(objectsMenu);

final var editor = Editor.getInstance().setCanvas(canvas);

borderPane.sceneProperty().addListener((\_) -> {

final var scene = borderPane.getScene();

scene.setOnKeyPressed((event) -> {

if (event.isControlDown() && (event.getCode() == KeyCode.Z)) editor.pop();

});

});

}

private String getFullName(final MenuItem selected, final Menu root) {

final StringBuilder result = new StringBuilder(root.getText() + " -> ");

boolean find = false;

for (final MenuItem item: root.getItems()) {

if (item instanceof final Menu menu) {

final var subpath = getFullName(selected, menu);

if (subpath.length() == 0) continue;

find = true;

result.append(subpath);

break;

}

if (!item.equals(selected)) continue;

find = true;

result.append(item.getText());

break;

}

return find ? result.toString() : "";

}

}

Файл controllers/TableController.java

package controllers;

import java.util.Map;

import java.util.HashMap;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.fxml.FXML;

import javafx.scene.control.TableView;

import shapes.Shape;

import javafx.scene.control.ScrollPane;

import tableview.PointPair;

import javafx.scene.control.TableColumn;

import editors.Editor;

public class TableController {

@FXML

private ScrollPane scrollPane;

@FXML

private TableView<PointPair> tableView;

@FXML

private TableColumn<PointPair, String> nameColumn;

@FXML

private TableColumn<PointPair, Double> x1Column;

@FXML

private TableColumn<PointPair, Double> y1Column;

@FXML

private TableColumn<PointPair, Double> x2Column;

@FXML

private TableColumn<PointPair, Double> y2Column;

private ObservableList<PointPair> points = FXCollections.observableArrayList();

@FXML

private void initialize() {

tableView.prefWidthProperty().bind(scrollPane.widthProperty());

tableView.prefHeightProperty().bind(scrollPane.heightProperty());

nameColumn.setCellValueFactory((cellData) -> cellData.getValue().getName());

x1Column.setCellValueFactory((cellData) -> cellData.getValue().getX1().asObject());

y1Column.setCellValueFactory((cellData) -> cellData.getValue().getY1().asObject());

x2Column.setCellValueFactory((cellData) -> cellData.getValue().getX2().asObject());

y2Column.setCellValueFactory((cellData) -> cellData.getValue().getY2().asObject());

final var editor = Editor.getInstance();

tableView.setItems(points);

final Map<Shape, PointPair> shapes = new HashMap<>();

editor.on("create", (shape) -> {

final var pair = shape.getDisplayCoords();

final var first = pair.getKey();

final var second = pair.getValue();

final var point = new PointPair(

shape.getName(),

first.getKey(),

first.getValue(),

second.getKey(),

second.getValue()

);

points.add(point);

shapes.put(shape, point);

});

editor.on("delete", (shape) -> {

final var point = shapes.get(shape);

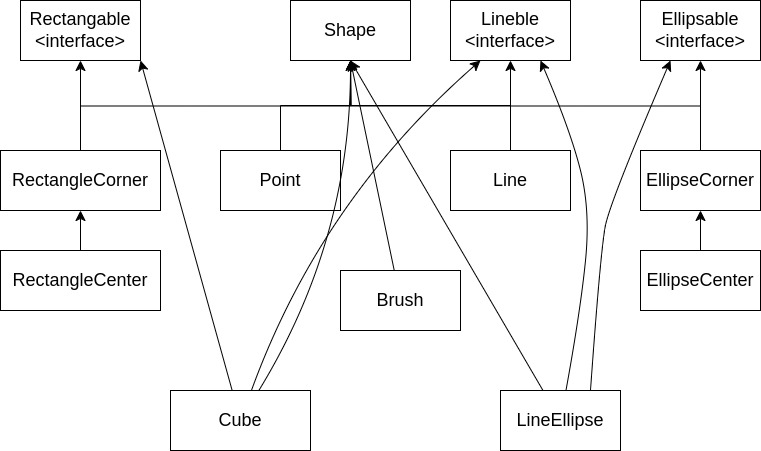
points.remove(point);

});

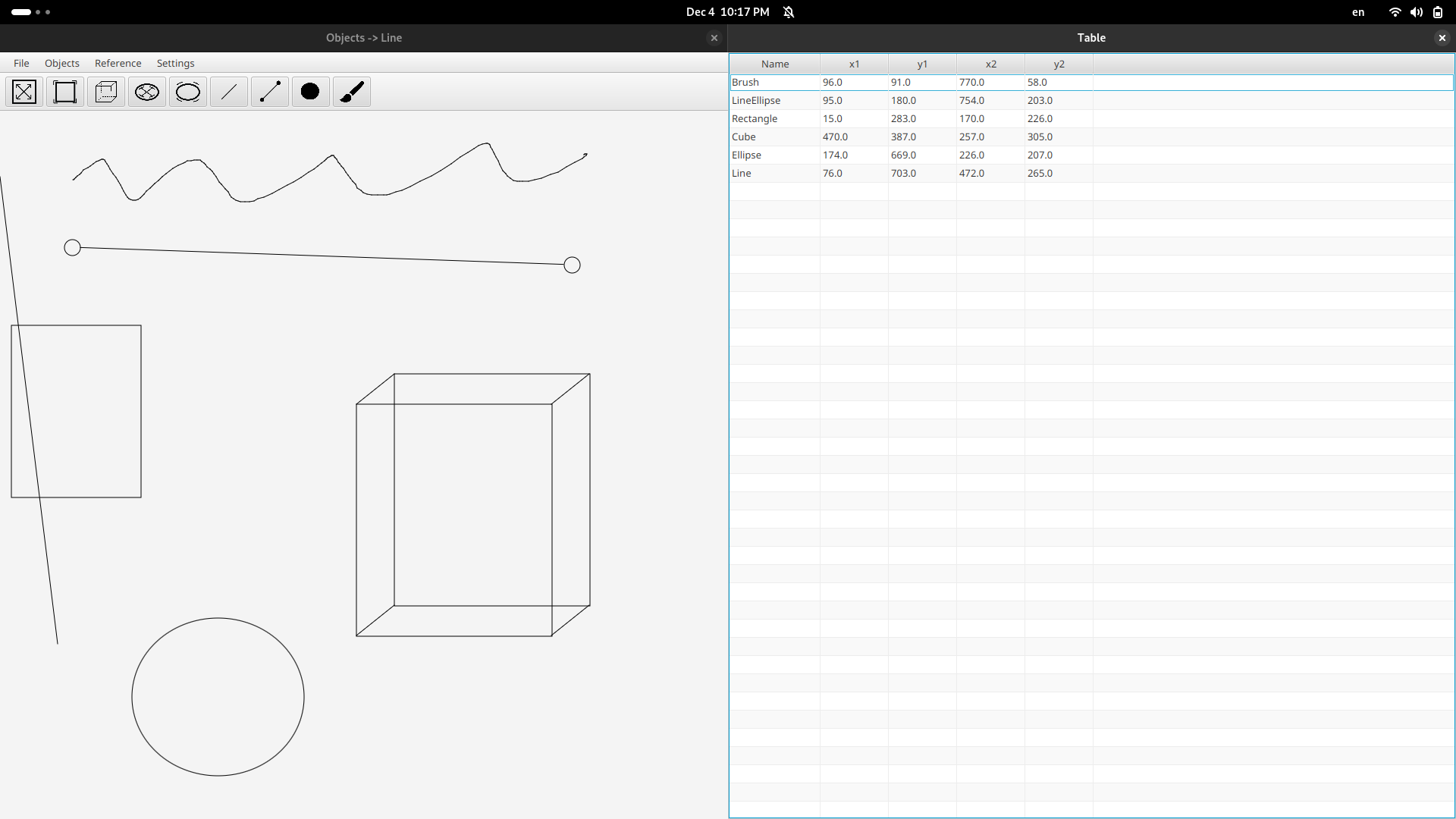
}

}

**Діаграма наслідування**



**Скріншоти виконання**



**Висновки**

Під час виконання лабораторної роботи я здобув навички використання інкапсуляції, абстрактних типів, успадкування та поліморфізму, вичвив патерни Singleton та Observer, створив простий графічний редактор та вдосконалив свої вміння програмування на Java. Протягом виконання я отримав теоретичні знання з архітектури розробки графічних додатків, та дізнався про кращі практики написання коду в об’єктно орієнтованому стилі використовуючи поліморфізм.