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

Розрахункова

з дисципліни
«Об’єктно-орієнтоване програмування»

Виконав:

Студент групи ІМ-31
Литвиненко Сергій Андрійович
номер у списку групи: 11

Перевірив:

Порєв В. М.

Київ 2024

Обґрунтування проектного рішення. Аналіз можливих варіантів рішення завдання

Темою завдання є растровий графічний редактор. Особливістю такого типу редакторів є те, що фігури відразу відображаються в bitmap. Цей підхід робить растрові графічні редактори досить швидкими та простими в реалізації, проте вони мають деякі недоліки. Основним недоліком можна виділити важу реалізацію виділення фігур, адже ми не зберігаємо усі наявні фігури в окремих структурах даних, а одразу переносимо їх на bitmap, саме це і робить графічні редактори швидкими. Вирішення цієї проблеми є досить важким, адже не завжди можна однозначно виділити графічні примітиви, наприклад через їх велику кучність, або наявність фотографі.

Як вирішення цієї проблеми я пропоную наступний підхід: під час створення зображення всі фігури будуть зберігатися в окремих структурах даних, а при збереженні користувачу будуть надані декілька можливих способів збереження: нативні - за допомогою яких можна відновити зображення з можливістю виділення та видалення останнього примітива та стандартизовані - зображення буде конвертуватися в одне зі стандартних для зображень форматів(png, jpg…).

Вихідні тексти усіх модулів програми

```
File: ./controllers/FileSaver.java
package controllers;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.nio.file.Files;
import javax.imageio.ImageIO;
import org.json.JSONObject;
import editors.Editor;
import javafx.embed.swing.SwingFXUtils;
import javafx.scene.image.Image;
import javafx.scene.image.WritableImage;
import javafx.util.Pair;
import shapes.ShapeParser;
import java.awt.image.BufferedImage;

public class FileSaver {
  public static void jsonSave(final File file, final Editor editor)
{
    try (final var writer = new BufferedWriter(new FileWriter(file,
false))) {
      final var canvas = editor.getCanvas();
      final var shapes = editor.shapes();
      final var result = new JSONObject();
      final var window = new JSONObject();
      final var data = ShapeParser.serialise(shapes);
      window.put("width", canvas.getWidth());
      window.put("height", canvas.getHeight());
      result.put("window", window);
      result.put("shapes", data);
      writer.write(result.toString());
    } catch (final Exception exception) {
      exception.printStackTrace();
    }
  }
  public static Pair<Double, Double> jsonOpen(final File file, final
Editor editor) {
    try {
      final var bytes = Files.readAllBytes(file.toPath());
      final var text = new String(bytes);
      final var json = new JSONObject(text);
      final var window = json.getJSONObject("window");
      final var shapes = json.getJSONArray("shapes");
      final var width = window.getDouble("width");
      final var height = window.getDouble("height");
      final var newshapes = ShapeParser.parse(shapes);
      final var canvas = editor.getCanvas();
      canvas.setWidth(width);
      canvas.setHeight(height);
      editor.replace(newshapes);
      return new Pair<Double, Double>(width, height);
    } catch (final Exception exception) {
      exception.printStackTrace();
      return new Pair<Double, Double>(0.0, 0.0);
    }
  }

  public static void pngSave(final File file, final Editor editor) {
    final var canvas = editor.getCanvas();
    final var width = (int)canvas.getWidth();
    final var height = (int)canvas.getHeight();
    final var writableImage = new WritableImage(width, height);
    canvas.snapshot(null, writableImage);
    final var renderedImage =
SwingFXUtils.fromFXImage(writableImage, null);
    try {
      ImageIO.write(renderedImage, "png", file);
    } catch (final IOException exception) {
      exception.printStackTrace();
    }
  }

  public static Pair<Double, Double> binaryOpen(final File file,
final Editor editor) {
    try {
      final var stream = new FileInputStream(file);
      final var image = new Image(stream);
      editor.restore().setBacground(image);
      return new Pair<Double, Double>(image.getWidth(),
image.getHeight());
    } catch (final IOException exception) {
      exception.printStackTrace();
      return new Pair<Double, Double>(0.0, 0.0);
    }
  }

  public static void jpgSave(final File file, final Editor editor) {
    final var canvas = editor.getCanvas();
    final var width = (int)canvas.getWidth();
    final var height = (int)canvas.getHeight();
    final var writableImage = new WritableImage(width, height);
    canvas.snapshot(null, writableImage);
    final var awtImage = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);
    final var renderedImage =
SwingFXUtils.fromFXImage(writableImage, awtImage);
    try {
      ImageIO.write(renderedImage, "jpg", file);
    } catch (final IOException exception) {
      exception.printStackTrace();
    }
  }

  public static String extention(final String filename) {
    final var index = filename.indexOf(".");
    return index == -1 ? "" : filename.substring(index + 1);
  }
}
```

```
File: ./controllers/MainController.java
package controllers;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.layout.BorderPane;
import javafx.scene.text.Text;
import javafx.stage.FileChooser;
import javafx.stage.Modality;
import javafx.stage.Stage;
import javafx.util.Pair;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.control.Button;
import javafx.scene.control.ChoiceBox;
import javafx.scene.control.ColorPicker;
import javafx.scene.control.ToolBar;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.function.BiConsumer;
import java.util.function.BiFunction;

import javafx.application.Platform;
import javafx.beans.value.ChangeListener;
import shapes.*;
import editors.Editor;
import javafx.scene.control.ToggleButton;
import javafx.scene.control.ScrollPane;

public class MainController {
  @FXML private BorderPane borderPane;
  @FXML private Canvas canvas;
  @FXML private ToolBar toolBar;
  @FXML private ColorPicker colorPicker;
  @FXML private ChoiceBox<Integer> choiceWidth;
  @FXML private ToggleButton fillButton;
  @FXML private ScrollPane scrollPane;
  @FXML private Text widthField;
  @FXML private Text heightField;
  @FXML private Text shapeField;
  private File background = null;

  private Editor editor;
  private ChangeListener<? super Number> widthListener = (_, _,
width) -> {
    final var canvasWidth = width.intValue() - 2;
    canvas.setWidth(canvasWidth);
    widthField.setText(String.valueOf(canvasWidth));
  };
  private ChangeListener<? super Number> heightListener = (_, _,
height) -> {
    final var canvasHeight = height.intValue() - 2;
    canvas.setHeight(canvasHeight);
    heightField.setText(String.valueOf(canvasHeight));
  };

  private final Map<String, Class<? extends Shape>> shapes = Map.of(
    "line", Line.class,
    "ellipse", Ellipse.class,
    "rectangle", Rectangle.class,
    "brush", Brush.class,
    "directedLine", DirectedLine.class,
    "bidirectedLine", BiDirectedLine.class
  );

  private final Map<String, Pair<BiConsumer<File, Editor>,
BiFunction<File, Editor, Pair<Double, Double>>>> extensions =
Map.of(
    "json", new Pair<>(FileSaver::jsonSave, FileSaver::jsonOpen),
    "png", new Pair<>(FileSaver::pngSave, FileSaver::binaryOpen),
    "jpg", new Pair<>(FileSaver::jpgSave, FileSaver::binaryOpen)
  );

  private final List<Integer> widths = List.of(1, 2, 3, 4, 5, 6, 7,
8);

  @FXML
  private void exit() {
    Platform.exit();
  }
  @FXML
  private void create() {
    try {
      final var view = "../resources/EnterCanvasSize.fxml";
      final var root = borderPane.getScene().getWindow();
      final var gui =
(Parent)FXMLLoader.load(getClass().getResource(view));
      final var scene = new Scene(gui);
      final var stage = new Stage();
      stage.setScene(scene);
      stage.initOwner(root);
      stage.initModality(Modality.WINDOW_MODAL);
      stage.setTitle("Enter Canvas Size");
      stage.show();
    } catch (final Exception exception) {
      exception.printStackTrace();
    }
  }

  public void setCanvasSize(final int width, final int height, final
boolean fixed) {
    canvas.setWidth(width);
    canvas.setHeight(height);
    heightField.setText(Integer.toString(height));
    widthField.setText(Integer.toString(width));
    if (!fixed) return;
    scrollPane.widthProperty().removeListener(widthListener);
    scrollPane.heightProperty().removeListener(heightListener);
  }

  @FXML
  private void changeWidth(final ActionEvent event) {
    final var width = choiceWidth.getValue();
    editor.changeWidth(width);
  }

  @FXML
  private void onFill(final ActionEvent event) {
    final var selected = fillButton.isSelected();
    final var text = selected ? "Fill" : "No fill";
    fillButton.setText(text);
    editor.setFill(selected);
  }
```

```
    @FXML
    private void changeColor(final ActionEvent event) {
      final var color = colorPicker.getValue();
      editor.changeColor(color);
    }

    private List<FileChooser.ExtensionFilter> getExtentionFilters() {
      final var result = new ArrayList<FileChooser.ExtensionFilter>();
      for (final var extention: extensions.keySet()) {
        final var name = "Select " + extention.toUpperCase() + "
  File";
        final var filter = new FileChooser.ExtensionFilter(name, "*."
  + extention);
        result.add(filter);
      }
      return result;
    }

    @FXML
    private void save() throws IOException {
      if (background == null) {
        saveAs();
        return;
      }
      final var extention = FileSaver.extention(background.getName());
      if (!extensions.containsKey(extention)) return;
      final var saver = extensions.get(extention).getKey();
      saver.accept(background, editor);
    }

    @FXML
    private void saveAs() throws IOException {
      final var stage = (Stage)borderPane.getScene().getWindow();
      final var fileChooser = new FileChooser();
      final var filters = getExtentionFilters();
      fileChooser.getExtensionFilters().addAll(filters);
      final var file = fileChooser.showSaveDialog(stage);
      if (file == null) return;
      final var extention = FileSaver.extention(file.getName());
      if (!extensions.containsKey(extention)) return;
      final var saver = extensions.get(extention).getKey();
      saver.accept(file, editor);
      this.background = file;
    }

    @FXML
    private void open() {
      final var stage = (Stage)borderPane.getScene().getWindow();
      final var fileChooser = new FileChooser();
      final var filters = getExtentionFilters();
      fileChooser.getExtensionFilters().addAll(filters);
      final var file = fileChooser.showOpenDialog(stage);
      if (file == null) return;
      final var extention = FileSaver.extention(file.getName());
      if (!extensions.containsKey(extention)) return;
      scrollPane.widthProperty().removeListener(widthListener);
      scrollPane.heightProperty().removeListener(heightListener);
      final var opener = extensions.get(extention).getValue();
      final var size = opener.apply(file, editor);
      widthField.setText(String.valueOf(size.getKey()));
      heightField.setText(String.valueOf(size.getValue()));
      this.background = file;
    }

    @FXML
    private void initialize() {
      scrollPane.widthProperty().addListener(widthListener);
      scrollPane.heightProperty().addListener(heightListener);
      this.editor = new Editor(canvas, borderPane);
      choiceWidth.getItems().addAll(widths);
      choiceWidth.setValue(widths.get(0));
      final var items = toolBar.getItems();
      for (final var pair: shapes.entrySet()) {
        final var name = pair.getKey();
        final var Constructor = pair.getValue();
        Button button = null;
        for (final var item: items) {
          if (!item.getId().equals(name)) continue;
          button = (Button)item;
        }
        if (button == null) continue;
        button.setOnAction((_) -> {
          editor.newShape(Constructor);
          shapeField.setText(name);
        });
      }
    }
  }


  File: ./controllers/DialogController.java
  package controllers;

  import java.util.ArrayList;
  import java.util.List;
  import javafx.fxml.FXML;
  import javafx.scene.control.TextField;
  import javafx.scene.layout.Pane;
  import javafx.stage.Stage;

  public class DialogController {
    private final List<String> defaultArgs = List.of(
      "java",

  "--module-path=/home/serhii/programming/code/java/libs/javafx/lib:/h
  ome/serhii/programming/code/java/libs/javax/lib/",

  "--add-modules=javafx.controls,javafx.fxml,javafx.swing,org.json",
      "Main"
    );
    @FXML private TextField widthField;
    @FXML private TextField heightField;
    @FXML private Pane pane;

    @FXML
    private void ok() {
      final var widthInput = widthField.getText();
      final var heightInput = heightField.getText();
      try {
        final var width = Integer.parseInt(widthInput);
        final var height = Integer.parseInt(heightInput);
        if (width <= 0 || height <= 0) return;
        final var args = new ArrayList<>(List.copyOf(defaultArgs));
        args.add(String.valueOf(width));
        args.add(String.valueOf(height));
        final var builder = new ProcessBuilder(args);
        builder.redirectErrorStream(true);
        builder.start();
```

```
        cancel();
      } catch (final Exception exception) {

      }

    }

    @FXML
    private void cancel() {
      final var root = (Stage)pane.getScene().getWindow();
      root.close();
    }
  }
```

```
File: ./editors/Editor.java
package editors;

import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.image.Image;
import javafx.scene.input.KeyCode;
import javafx.scene.input.KeyEvent;
import javafx.scene.input.MouseButton;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import shapes.Shape;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Editor {
  private static final double selectedK = 2;
  private final List<Shape> shapes = new ArrayList<>();
  private final List<Shape> selected = new ArrayList<>();
  private final Canvas canvas;
  private final GraphicsContext context;
  private Color color = Color.BLACK;
  private int width = 1;
  private Image background = null;
  private boolean fill = false;

  public Editor(final Canvas canvas, final Pane root) {
    this.canvas = canvas;
    this.context = canvas.getGraphicsContext2D();
    canvas.widthProperty().addListener((_) -> redraw());
    canvas.heightProperty().addListener((_) -> redraw());
    root.addEventHandler(KeyEvent.KEY_PRESSED, (event) -> {
      final var isCtrl = event.isControlDown();
      final var isZ = event.getCode().equals(KeyCode.Z);
      if (!(isCtrl && isZ) || shapes.isEmpty()) return;
      final var shape = shapes.removeLast();
      if (selected.contains(shape)) selected.remove(shape);
      redraw();
    });
    root.addEventHandler(KeyEvent.KEY_PRESSED, (event) -> {
      if (!event.getCode().equals(KeyCode.DELETE)) return;
    }

      for (final var shape: selected) shapes.remove(shape);
      selected.clear();
      redraw();
    });
    canvas.addEventFilter(MouseEvent.MOUSE_PRESSED, (event) -> {
      if (isPrimaryButton(event)) {
        if (selected.size() == 0) return;
        selected.clear();
        redraw();
      }
      final var x = event.getX();
      final var y = event.getY();
      for (final var shape: shapes) {
        final var contains = shape.contains(x, y);
        if (!contains || selected.contains(shape)) continue;
        selected.add(shape);
        drawShape(shape, selectedK);
      }
    });
  }

  public Canvas getCanvas() {
    return canvas;
  }

  private void clear() {
    context.clearRect(0, 0, canvas.getWidth(), canvas.getHeight());
    context.setFill(Color.WHITE);
    context.fillRect(0, 0, canvas.getWidth(), canvas.getHeight());
  }

  private void drawShape(final Shape shape, double k) {
    final var config = shape.getConfig();
    context.setLineWidth(config.getWidth() * k);
    context.setStroke(config.getColor());
    if (config.getFill()) {
      context.setFill(config.getColor());
    }
    shape.draw(context);
  }

  private void drawShape(final Shape shape) {
    drawShape(shape, 1);
  }

  private void redraw() {
    clear();
    if (background != null) context.drawImage(background, 0, 0);
    for (final var shape: shapes) {
      final var k = selected.contains(shape) ? selectedK : 1;
      drawShape(shape, k);
    }
  }

  private void drawDashes(final Shape shape) {
    context.setLineDashes(10);
    drawShape(shape);
    context.setLineDashes(0);
  }

  private static boolean isPrimaryButton(MouseEvent event) {
    return event.getButton().equals(MouseButton.PRIMARY);
  }

  public void setBacground(final Image image) {
    final var imageWidth = image.getWidth();
    final var imageHeight = image.getHeight();
    canvas.setWidth(imageWidth);
    canvas.setHeight(imageHeight);
    this.background = image;
    redraw();
  }

  public void newShape(final Class<? extends Shape> constructor) {
    canvas.setOnMousePressed((event) -> {
      if (!isPrimaryButton(event)) return;
      onClick(event, constructor);
    });
  }

  private void onClick(final MouseEvent event, final Class<? extends
Shape> constructor) {
    try {
      final var declared =
constructor.getDeclaredConstructor(double.class, double.class);
```

```
      final var shape = declared.newInstance(event.getX(),
event.getY());

shape.getConfig().setColor(color).setWidth(width).setFill(fill);
      canvas.setOnMouseDragged((info) -> {
        if (!isPrimaryButton(info)) return;
        onMove(info, shape);
      });
      canvas.setOnMouseReleased((info) -> {
        if (!isPrimaryButton(info)) return;
        onRelease(shape);
        canvas.setOnMouseDragged(null);
        canvas.setOnMouseReleased(null);
      });
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  private void onMove(final MouseEvent event, final Shape shape) {
    shape.update(event.getX(), event.getY());
    redraw();
    if (shape.useDashes) drawDashes(shape);
    else drawShape(shape);
  }

  private void onRelease(final Shape shape) {
    shapes.add(shape);
    redraw();
  }

  public List<Shape> shapes() {
    return Collections.unmodifiableList(shapes);
  }

  public Editor restore() {
    shapes.clear();
    clear();
    return this;
  }

  public Editor replace(final List<Shape> shapes) {
    this.shapes.clear();
    this.shapes.addAll(shapes);
    redraw();
    return this;
  }

  public Editor changeColor(final Color color) {
    this.color = color;
    for (final var shape: selected) {
      shape.getConfig().setColor(color);
    }
    selected.clear();
    redraw();
    return this;
  }

  public Editor changeWidth(final int width) {
    this.width = width;
    for (final var shape: selected) {
      shape.getConfig().setWidth(width);
    }
    selected.clear();
    redraw();
    return this;
  }

  public Editor setFill(final boolean fill) {
    this.fill = fill;
    for (final var shape: selected) {
      shape.getConfig().setFill(fill);
    }
    selected.clear();
    redraw();
    return this;
  }
}
```

```
File: ./Main.java
import java.util.List;

import controllers.MainController;
import javafx.application.Application;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
import javafx.util.Pair;
import javafx.fxml.FXMLLoader;

public class Main extends Application {
  private final int width = 800;
  private final int height = 600;
  private final String pathToView = "./resources/Main.fxml";
  private final String titleMain = "Raster graphic editor";

  public static void main(String[] args) {
    launch(args);
  }

  private Pair<Integer, Integer> parseCanvasSize(final List<String>
args) {
    if (args.size() != 2) return null;
    final var width = Integer.parseInt(args.get(0));
    final var height = Integer.parseInt(args.get(1));
    return new Pair<>(width, height);
  }

  @Override
  public void start(Stage stage) throws Exception {
    final var loader = new
FXMLLoader(getClass().getResource(pathToView));
    final Parent root = loader.load();
    final MainController controller = loader.getController();
    final var size = parseCanvasSize(getParameters().getUnnamed());
    final var canvasWidth = size == null ? width : size.getKey();
    final var canvasHeight = size == null ? height :
size.getValue();
    controller.setCanvasSize(canvasWidth, canvasHeight, size !=
null);
    if (canvasHeight > height || canvasWidth > width) {
      stage.setWidth(width);
      stage.setHeight(height);
    }
    final var scene = new Scene(root);
    stage.setScene(scene);
    stage.setTitle(titleMain);
    scene.setFill(Color.WHITE);
    stage.show();
  }
}
```

```
File: ./resources/Main.fxml
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.Insets?>
<?import javafx.scene.canvas.Canvas?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.ChoiceBox?>
<?import javafx.scene.control.ColorPicker?>
<?import javafx.scene.control.Menu?>
<?import javafx.scene.control.MenuBar?>
<?import javafx.scene.control.MenuItem?>
<?import javafx.scene.control.ScrollPane?>
<?import javafx.scene.control.ToggleButton?>
<?import javafx.scene.control.ToolBar?>
<?import javafx.scene.control.Tooltip?>
<?import javafx.scene.image.Image?>
<?import javafx.scene.image.ImageView?>
<?import javafx.scene.layout.BorderPane?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.paint.Color?>
<?import javafx.scene.text.Text?>

<BorderPane fx:id="borderPane" minHeight="0" minWidth="0"
xmlns="http://javafx.com/javafx/22"
xmlns:fx="http://javafx.com/fxml/1"
fx:controller="controllers.MainController">
  <top>
    <BorderPane BorderPane.alignment="CENTER">
      <top>
        <MenuBar fx:id="menuBar" BorderPane.alignment="CENTER">
          <menus>
            <Menu mnemonicParsing="false" text="File">
              <items>
                <MenuItem mnemonicParsing="false" onAction="#create"
text="Create..." />
                <MenuItem mnemonicParsing="false" onAction="#open"
text="Open..." />
                <MenuItem mnemonicParsing="false" onAction="#save"
text="Save..." />
                <MenuItem mnemonicParsing="false" onAction="#saveAs"
text="Save as..." />
                <MenuItem mnemonicParsing="false" onAction="#exit"
text="Close" />
```

```
          </items>
        </Menu>
      </menus>
    </MenuBar>
  </top>
  <bottom>
    <ToolBar fx:id="toolBar" BorderPane.alignment="CENTER">
      <items>
        <Button id="rectangle" mnemonicParsing="false">
          <graphic>
            <ImageView fitHeight="32.0" fitWidth="32.0">
              <image>
                <Image url="@icons/rectangle.png" />
              </image>
            </ImageView>
          </graphic>
          <tooltip>
            <Tooltip text="Rectangle" />
          </tooltip>
        </Button>
        <Button id="ellipse" mnemonicParsing="false">
          <graphic>
            <ImageView fitHeight="32.0" fitWidth="32.0">
              <image>
                <Image url="@icons/ellipse.png" />
              </image>
            </ImageView>
          </graphic>
          <tooltip>
            <Tooltip text="Ellipse" />
          </tooltip>
        </Button>
        <Button id="line" 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="brush" 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>
            <Button id="directedLine" mnemonicParsing="false">
              <graphic>
                <ImageView fitHeight="32.0" fitWidth="32.0">
                  <image>
                    <Image url="@icons/directedLine.png" />
                  </image>
                </ImageView>
              </graphic>
              <tooltip>
                <Tooltip text="Directed Line" />
              </tooltip>
            </Button>
            <Button id="bidirectedLine" mnemonicParsing="false">
              <graphic>
                <ImageView fitHeight="32.0" fitWidth="32.0">
                  <image>
                    <Image url="@icons/bidirectedLine.png" />
                  </image>
                </ImageView>
              </graphic>
              <tooltip>
                <Tooltip text="Bidirected Line" />
              </tooltip>
            </Button>
            <ToggleButton id="fillButton" fx:id="fillButton"
mnemonicParsing="false" onAction="#onFill" prefHeight="40.0"
text="Fill" />
            <ColorPicker id="colorPicker" fx:id="colorPicker"
onAction="#changeColor" prefHeight="40.0" prefWidth="145.0">
              <value>
```

```
                <Color />
              </value>
            </ColorPicker>
            <ChoiceBox id="choiceWidth" fx:id="choiceWidth"
onAction="#changeWidth" prefHeight="40.0" prefWidth="50.0" />
          </items>
        </ToolBar>
      </bottom>
    </BorderPane>
  </top>
  <center>
    <ScrollPane fx:id="scrollPane" hbarPolicy="AS_NEEDED"
minHeight="0" minWidth="0" style="-fx-focus-color: transparent;
-fx-faint-focus-color: transparent;" vbarPolicy="AS_NEEDED"
BorderPane.alignment="CENTER">
      <content>
        <Canvas id="canvas" fx:id="canvas" />
      </content>
    </ScrollPane>
  </center>
  <bottom>
    <BorderPane BorderPane.alignment="CENTER">
      <left>
        <HBox BorderPane.alignment="CENTER">
          <children>
            <Text strokeType="OUTSIDE" strokeWidth="0.0"
text="Shape: " />
            <Text fx:id="shapeField" strokeType="OUTSIDE"
strokeWidth="0.0" />
          </children>
          <padding>
            <Insets left="5.0" right="15.0" />
          </padding>
        </HBox>
      </left>
      <right>
        <HBox BorderPane.alignment="CENTER">
          <children>
            <Text strokeType="OUTSIDE" strokeWidth="0.0"
text="Width: " />
            <Text fx:id="widthField" strokeType="OUTSIDE"
strokeWidth="0.0">
              <HBox.margin>
```

```
                <Insets right="5.0" />
              </HBox.margin>
            </Text>
            <Text strokeType="OUTSIDE" strokeWidth="0.0" 
text="Height:" />
            <Text fx:id="heightField" strokeType="OUTSIDE" 
strokeWidth="0.0" />
          </children>
          <BorderPane.margin>
            <Insets left="15.0" right="5.0" />
          </BorderPane.margin>
        </HBox>
      </right>
    </BorderPane>
  </bottom>
</BorderPane>
```

File: ./resources/EnterCanvasSize.fxml

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

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.text.Font?>
<?import javafx.scene.text.Text?>

<AnchorPane fx:id="pane" maxHeight="-Infinity" maxWidth="-Infinity"
minHeight="-Infinity" minWidth="-Infinity" prefHeight="163.0"
prefWidth="434.0" xmlns="http://javafx.com/javafx/22"
xmlns:fx="http://javafx.com/fxml/1"
fx:controller="controllers.DialogController">
   <children>
      <Text layoutX="131.0" layoutY="43.0" strokeType="OUTSIDE"
strokeWidth="0.0" text="Enter size">
         <font>
            <Font size="30.0" />
         </font>
      </Text>
      <Text layoutX="73.0" layoutY="80.0" strokeType="OUTSIDE"
strokeWidth="0.0" text="Width">
         <font>
            <Font size="20.0" />
         </font>
      </Text>
      <Text layoutX="69.0" layoutY="114.0" strokeType="OUTSIDE"
strokeWidth="0.0" text="Height">
         <font>
            <Font size="20.0" />
         </font>
      </Text>
      <TextField fx:id="widthField" layoutX="194.0" layoutY="61.0"
/>
      <TextField fx:id="heightField" layoutX="194.0" layoutY="95.0"
/>
      <Button onAction="#ok" layoutX="348.0" layoutY="125.0"
mnemonicParsing="false" prefHeight="35.0" prefWidth="75.0" text="Ok"
/>
      <Button onAction="#cancel" layoutX="266.0" layoutY="125.0"
mnemonicParsing="false" prefHeight="35.0" prefWidth="75.0"
text="Cancel" />
```

```
    </children>
</AnchorPane>
```

```
File: ./shapes/Brush.java
package shapes;

import java.util.List;

import javafx.scene.canvas.GraphicsContext;

public class Brush extends shapes.Shape {

  public Brush(final List<Double> coords) {
    super(coords);
    useDashes = false;
  }

  public Brush(final double x, final double y) {
    this(List.of(x, y));
  }

  @Override
  public Shape update(final double x, final double y) {
    coords.add(x);
    coords.add(y);
    return this;
  }

  @Override
  public void draw(final GraphicsContext context) {
    final var size = coords.size();
    if (size == 2) {
      final var width = config.getWidth();
      final var x = coords.get(0);
      final var y = coords.get(1);
      context.fillOval(x - width, y - width, width * 2, width * 2);
    }
    for (int index = 4; index < size; index += 2) {
      final var x1 = coords.get(index - 2);
      final var y1 = coords.get(index - 1);
      final var x2 = coords.get(index);
      final var y2 = coords.get(index + 1);
      context.strokeLine(x1, y1, x2, y2);
    }
  }
  @Override
  public boolean contains(final double x, final double y) {
    final var size = coords.size();
    final var width = config.getWidth();
    for (int index = 4; index < size; index += 2) {
      final var x1 = coords.get(index - 2);
      final var y1 = coords.get(index - 1);
      final var x2 = coords.get(index);
      final var y2 = coords.get(index + 1);
      final var exists = Line.lineContaines(x1, y1, x2, y2, x, y,
width);
      if (exists) return true;
    }
    return false;
  }

}
```

```
File: ./shapes/DirectedLine.java
package shapes;

import java.util.List;

import javafx.scene.canvas.GraphicsContext;

public class DirectedLine extends shapes.Line {
  private static final double rotateAngle = Math.PI / 4;
  private static final double arrowLength = 10;

  public DirectedLine(final List<Double> coords) {
    super(coords);
  }

  public DirectedLine(final double x, final double y) {
    super(x, y);
  }

  public static void drawArrows(final GraphicsContext context,
double x1, double y1, double x2, double y2) {
    final var angle = Math.atan2(y2 - y1, x2 - x1);
    final var arrowX1 = x2 - arrowLength * Math.cos(angle -
rotateAngle);
    final var arrowY1 = y2 - arrowLength * Math.sin(angle -
rotateAngle);
    final var arrowX2 = x2 - arrowLength * Math.cos(angle +
rotateAngle);
    final var arrowY2 = y2 - arrowLength * Math.sin(angle +
rotateAngle);
    context.strokeLine(x2, y2, arrowX1, arrowY1);
    context.strokeLine(x2, y2, arrowX2, arrowY2);
  }

  @Override
  public void draw(final GraphicsContext context) {
    if (coords.size() < 4) return;
    super.draw(context);
    drawArrows(context, coords.get(0), coords.get(1), coords.get(2),
coords.get(3));
  }

}
```

```
File: ./shapes/BiDirectedLine.java
package shapes;

import java.util.List;

import javafx.scene.canvas.GraphicsContext;

public class BiDirectedLine extends DirectedLine {
  public BiDirectedLine(final List<Double> coords) {
    super(coords);
  }

  public BiDirectedLine(final double x, final double y) {
    super(x, y);
  }

  @Override
  public void draw(final GraphicsContext context) {
    if (coords.size() < 4) return;
    super.draw(context);
    drawArrows(context, coords.get(2), coords.get(3), coords.get(0),
coords.get(1));
  }
}
```

```
File: ./shapes/Ellipse.java
package shapes;

import java.util.List;

import javafx.scene.canvas.GraphicsContext;

public class Ellipse extends shapes.Shape {

  public Ellipse(final List<Double> coords) {
    super(coords);
  }

  public Ellipse(final double x, final double y) {
    super(List.of(x, y, 0.0, 0.0));
  }

  @Override
  public Shape update(final double x, final double y) {
    coords.set(2, x);
    coords.set(3, y);
    return this;
  }

  @Override
  public void draw(final GraphicsContext context) {
    if (coords.size() < 4) return;
    final var width = context.getLineWidth();
    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 = Math.abs(x2 - x1);
    final var dy = Math.abs(y2 - y1);
    final var x = (x1 + x2 - dx) / 2;
    final var y = (y1 + y2 - dy) / 2;
    if (config.getFill()) context.fillOval(x, y, dx + width, dy +
width);
    else context.strokeOval(x, y, dx + width, dy + width);
  }

  @Override
  public boolean contains(final double x, final double y) {
    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 center_x = (x1 + x2) / 2;
    final var center_y = (y1 + y2) / 2;
    final var a = x2 - center_x;
    final var b = y2 - center_y;
    final var first = ((x - center_x) * (x - center_x)) / (a * a);
    final var second = ((y - center_y) * (y - center_y)) / (b * b);
    final var sum = first + second;
    if (config.getFill()) return sum <= 1;
    final var delta = config.getWidth() / 100.0 * 2;
    return Math.abs(1 - sum) < delta;
  }
}
```

```
File: ./shapes/Line.java
package shapes;

import java.util.List;

import javafx.scene.canvas.GraphicsContext;

public class Line extends shapes.Shape {

  public Line(final List<Double> coords) {
    super(coords);
  }

  public Line(final double x, final double y) {
    super(List.of(x, y, 0.0, 0.0));
  }

  @Override
  public Shape update(final double x, final double y) {
    coords.set(2, x);
    coords.set(3, y);
    return this;
  }

  @Override
  public void draw(final GraphicsContext context) {
    if (coords.size() < 4) return;
    context.strokeLine(coords.get(0), coords.get(1), coords.get(2),
coords.get(3));
  }

  static public boolean lineContaines(double x1, double y1, double
x2, double y2, double x, double y, double width) {
    final var dx = x2 - x1;
    final var dy = y2 - y1;
    final var numerator = dx * (y1 - y) - (x1 - x) * dy;
    final var denumerator = Math.sqrt(dx * dx + dy * dy);
    final var distance = Math.abs(numerator) / denumerator;
    final var inRange = (x - x1) * (x - x2) <= width;
    return distance < width * 2 && inRange;
  }

  @Override
  public boolean contains(final double x, final double y) {
    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);
    return lineContaines(x1, y1, x2, y2, x, y, config.getWidth());
  }
}
```

File: ./shapes/Rectangle.java
```java
package shapes;

import java.util.List;
import javafx.scene.canvas.GraphicsContext;

public class Rectangle extends shapes.Shape {

  public Rectangle(final List<Double> coords) {
    super(coords);
  }

  public Rectangle(final double x, final double y) {
    super(List.of(x, y, 0.0, 0.0));
  }

  @Override
  public Shape update(final double x, final double y) {
    coords.set(2, x);
    coords.set(3, y);
    return this;
  }

  @Override
  public void draw(final GraphicsContext 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 = Math.abs(x2 - x1);
    final var dy = Math.abs(y2 - y1);
    final var x = (x1 + x2 - dx) / 2;
    final var y = (y1 + y2 - dy) / 2;
    if (config.getFill()) context.fillRect(x, y, dx, dy);
    else context.strokeRect(x, y, dx, dy);
  }

  @Override
  public boolean contains(final double x, final double y) {
    final var x1 = coords.get(0);
    final var y1 = coords.get(1);
    final var x2 = coords.get(2);
    final var y2 = coords.get(3);
    if (config.getFill()) {
      final var inside = x >= x1 && x <= x2 && y >= y1 && y <= y2;
      if (inside) return inside;
    }
    final var width = config.getWidth();
    final var up = Line.lineContaines(x1, y1, x2, y1, x, y, width);
    final var rt = Line.lineContaines(x2, y1, x2, y2, x, y, width);
    final var bt = Line.lineContaines(x2, y2, x1, y2, x, y, width);
    final var lt = Line.lineContaines(x1, y2, x1, y1, x, y, width);
    return up || rt || bt || lt;
  }
}
```

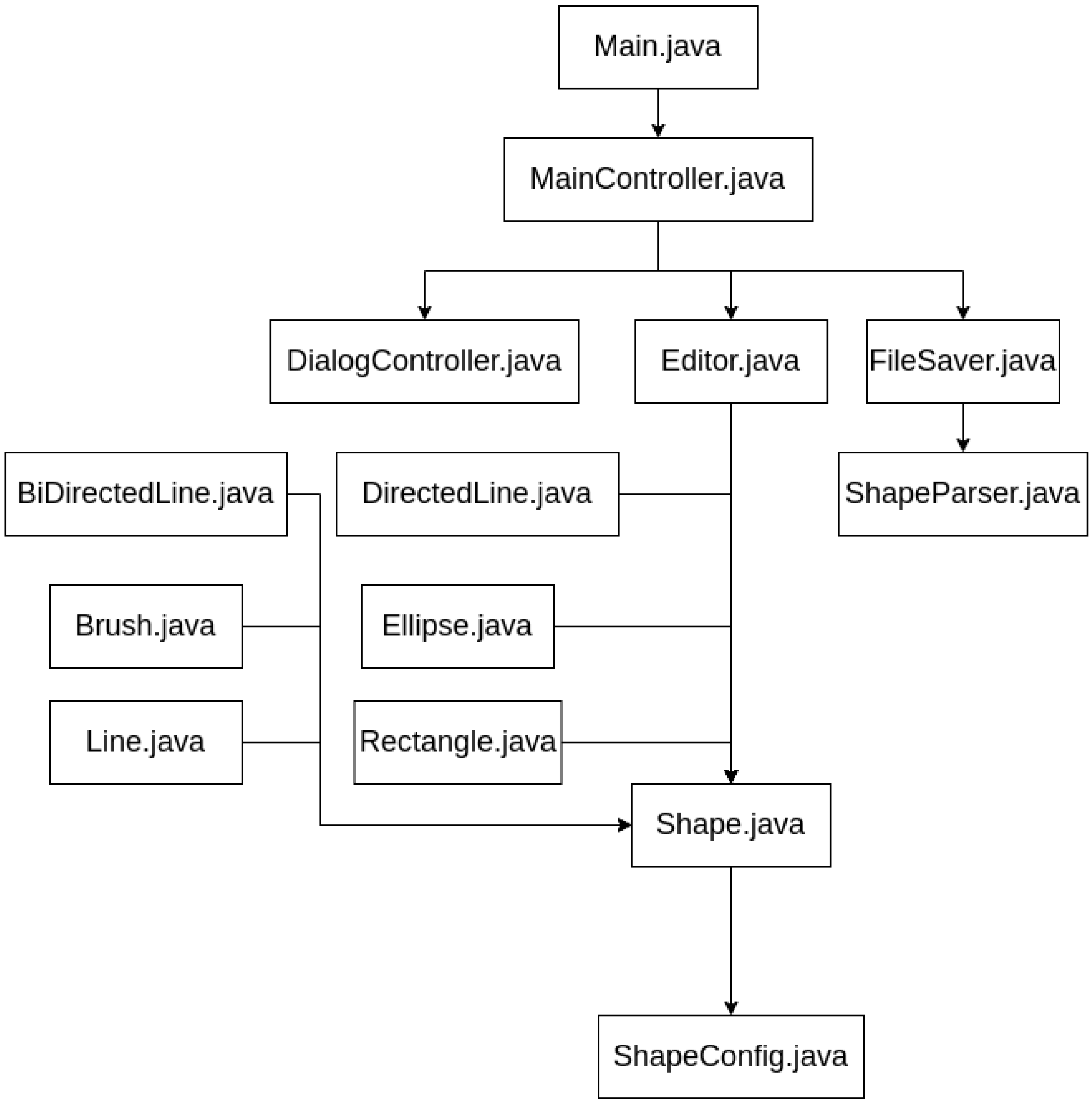
```
File: ./shapes/Shape.java
package shapes;

import java.util.ArrayList;
import java.util.List;
import javafx.scene.canvas.GraphicsContext;

public abstract class Shape {
  public boolean useDashes = true;
  protected final List<Double> coords;
  protected ShapeConfig config;

  public Shape(final List<Double> coords) {
    this.coords = new ArrayList<>(coords);
    this.config = new ShapeConfig();
  }

  public Shape(final double x, final double y) {
    this(List.of(x, y));
  }

  public Shape apply(final ShapeConfig config) {
    this.config = config;
    return this;
  }

  public ShapeConfig getConfig() { return config; }

  public List<Double> getCoords() { return coords; }

  public abstract Shape update(final double x, final double y);

  public abstract void draw(final GraphicsContext context);

  public abstract boolean contains(final double x, final double y);
}
```

File: ./shapes/ShapeConfig.java

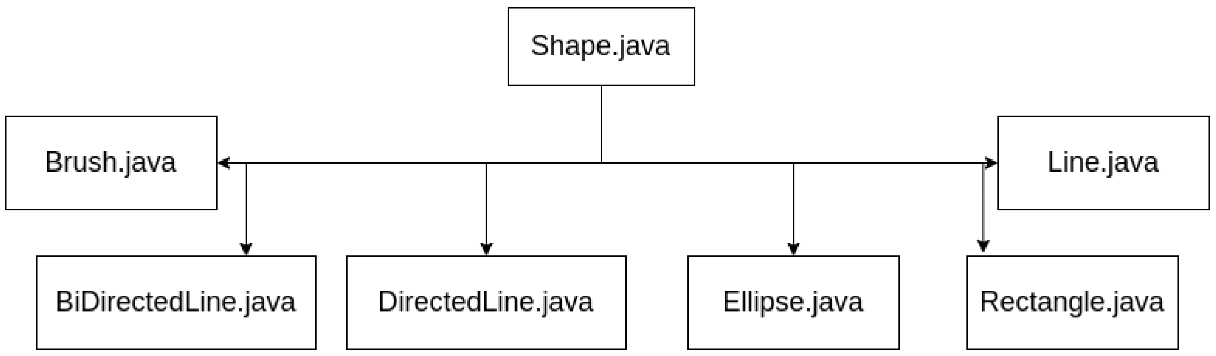
```java
package shapes;

import javafx.scene.paint.Color;

public class ShapeConfig {
  private Color color;
  private int width;
  private boolean fill;

  public ShapeConfig(final Color color, final int width, final
boolean fill) {
    this.color = color;
    this.width = width;
    this.fill = fill;
  }

  public ShapeConfig() {
    this(Color.BLACK, 1, false);
  }

  public Color getColor() { return color; }
  public ShapeConfig setColor(final Color color) {
    this.color = color;
    return this;
  }
  public int getWidth() { return width; }
  public ShapeConfig setWidth(final int width) {
    this.width = width;
    return this;
  }
  public boolean getFill() { return fill; }
  public ShapeConfig setFill(final boolean fill) {
    this.fill = fill;
    return this;
  }

}
```

```
File: ./shapes/ShapeParser.java
package shapes;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import org.json.JSONArray;
import org.json.JSONObject;
import javafx.scene.paint.Color;
import javafx.util.Pair;

public class ShapeParser {

  private static final Map<String, Class<? extends Shape>> shapes =
Map.of(
    Brush.class.getSimpleName(), Brush.class,
    Ellipse.class.getSimpleName(), Ellipse.class,
    Line.class.getSimpleName(), Line.class,
    Rectangle.class.getSimpleName(), Rectangle.class,
    DirectedLine.class.getSimpleName(), DirectedLine.class,
    BiDirectedLine.class.getSimpleName(), BiDirectedLine.class
  );

  public static Pair<Pair<ShapeConfig, String>, List<Double>>
parse(final JSONObject shape) {
    final var name = shape.getString("name");
    final var width = shape.getInt("width");
    final var color = shape.getString("color");
    final var fill = shape.getBoolean("fill");
    final var numbers = shape.getJSONArray("coords");
    final var coords = new ArrayList<Double>();
    for (final var number: numbers) {
      final var coord = Double.parseDouble(number.toString());
      coords.add(coord);
    }
    final var config = new ShapeConfig(Color.valueOf(color), width,
fill);
    return new Pair<>(new Pair<>(config, name), coords);
  }

  public static List<Shape> parse(final JSONArray shapes) {
    final var result = new ArrayList<Shape>();
    for (final var item: shapes) {
      final var shape = new JSONObject(item.toString());
      final var parsed = parse(shape);
      final var info = parsed.getKey();
      final var config = info.getKey();
      final var name = info.getValue();
      final var coords = parsed.getValue();
      try {
        final var constructor = ShapeParser.shapes.get(name);
        final var declared =
constructor.getDeclaredConstructor(List.class);
        final var newshape = declared.newInstance(coords);
        newshape.apply(config);
        result.add(newshape);
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
    return result;
  }

  public static JSONObject serialise(final Shape shape) {
    final var config = shape.config;
    final var item = new JSONObject();
    final var name = shape.getClass().getSimpleName();
    item.put("name", name);
    item.put("width", config.getWidth());
    item.put("color", config.getColor().toString());
    item.put("fill", config.getFill());
    item.put("coords", shape.getCoords());
    return item;
  }

  public static JSONArray serialise(final List<Shape> shapes) {
    final var result = new JSONArray();
    for (final var shape: shapes) {
      final var serialised = serialise(shape);
      result.put(serialised);
    }
    return result;
  }
}
```

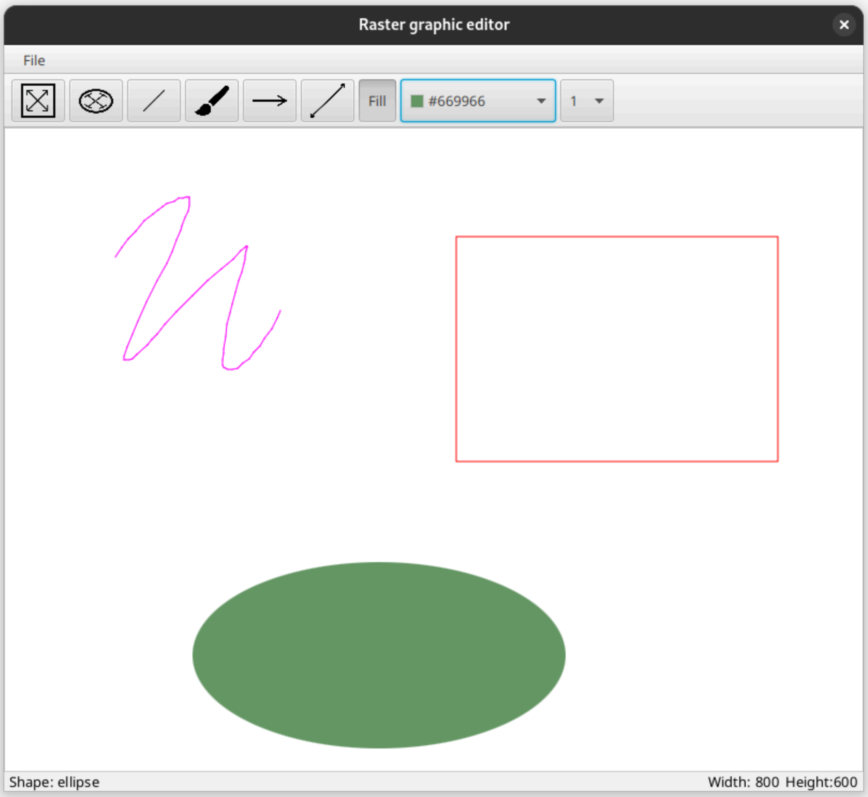
Діаграма #include-ієрархії модулів



Діаграма класів

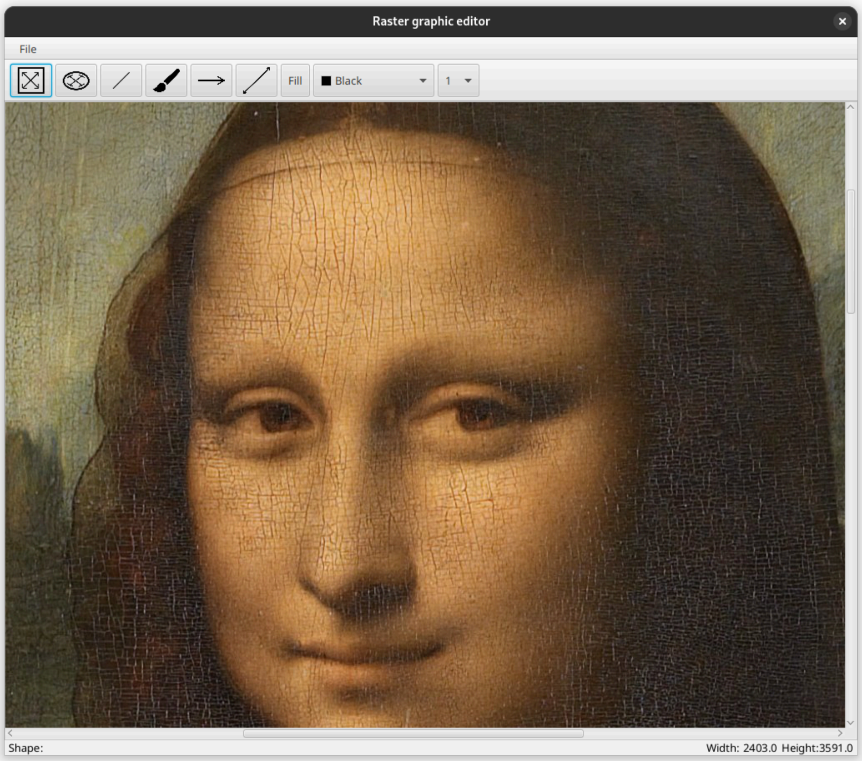


Ілюстрації (скріншоти)



Shape: ellipse

Width: 800 Height:600



Shape:

Width: 2403.0 Height:3591.0