--------------------GUI.java------------------------

public class GUI extends JFrame{

private static final long serialVersionUID = 1L;

private class Platno extends JComponent{

private static final long serialVersionUID = 1L;

DocumentModel dm;

boolean shiftDown = false;

boolean ctrlDown = false;

public Platno(DocumentModel dm) {

setFocusable(true);

this.dm = dm;

this.dm.addDocumentModelListener(this::repaint);

keyboardAndMouseListeners();

}

private void keyboardAndMouseListeners() {

addKeyListener(new KeyAdapter() {

@Override

public void keyPressed(KeyEvent e) {

System.out.println("Usao");

switch (e.getKeyCode()) {

case KeyEvent.VK\_SHIFT -> shiftDown = true;

case KeyEvent.VK\_CONTROL -> ctrlDown = true;

case KeyEvent.VK\_ESCAPE -> changeState(new IdleState());

}

currentState.keyPressed(e.getKeyCode());

}

@Override

public void keyReleased(KeyEvent e) {

switch (e.getKeyCode()) {

case KeyEvent.VK\_SHIFT -> shiftDown = false;

case KeyEvent.VK\_CONTROL -> ctrlDown = false;

}

}

});

addMouseListener(new MouseAdapter() {

@Override

public void mousePressed(MouseEvent e) {

currentState.mouseDown(new Point(e.getPoint().x,e.getPoint().y), shiftDown, ctrlDown);

}

@Override

public void mouseReleased(MouseEvent e) {

currentState.mouseUp(new Point(e.getPoint().x,e.getPoint().y), shiftDown, ctrlDown);

}

});

addMouseMotionListener(new MouseAdapter() {

@Override

public void mouseDragged(MouseEvent e) {

currentState.mouseDragged(new Point(e.getPoint().x,e.getPoint().y));

}

});

}

@Override

public Dimension getPreferredSize() {

return new Dimension(500,500);

}

@Override

public void paintComponent(Graphics g) {

Graphics2D g2d = (Graphics2D)g;

g2d.setColor(Color.white);

g2d.fillRect(0, 0, 500, 500);

Renderer r = new G2DRendererImpl(g2d);

for(GraphicalObject object : dm.list()) {

object.render(r);

currentState.afterDraw(r, object);

}

currentState.afterDraw(r);

}

}

List<GraphicalObject> objects;

DocumentModel dm;

Platno platno;

private State currentState;

private static final Map<String,GraphicalObject> TAGS = new HashMap<>();

static {

GraphicalObject o = new CompositeShape(null);

TAGS.put(o.getShapeID(), o);

}

public GUI(List<GraphicalObject> objects) {

setDefaultCloseOperation(WindowConstants.DISPOSE\_ON\_CLOSE);

setFocusable(false);

setLocation(20, 20);

currentState = new IdleState();

this.objects = objects;

for (GraphicalObject graphicalObject : objects) {

TAGS.put(graphicalObject.getShapeID(), graphicalObject);

}

dm = new DocumentModel();

initGUI();

pack();

}

private void initGUI() {

setLayout(new BorderLayout());

platno = new Platno(dm);

add(platno,BorderLayout.CENTER);

platno.requestFocusInWindow();

JToolBar t = new JToolBar();

t.add(new JButton(createAction("Učitaj", ()->{

try {

Myimport();

} catch (IOException e) {

e.printStackTrace();

}

})));

t.add(new JButton(createAction("Pohrani", ()->{

try {

Myexport();

} catch (IOException e) {

e.printStackTrace();

}

})));

t.add(new JButton(createAction("SVG export", ()->{

try {

SVGexport();

} catch (IOException e) {

e.printStackTrace();

}

})));

for(GraphicalObject obj: objects) {

t.add(new JButton(createAction(obj.getShapeName(), ()->changeState(new AddShapeState(obj, dm)))));

}

t.add(new JButton(createAction("Selektiraj", ()->changeState(new SelectShapeState(dm)))));

t.add(new JButton(createAction("Brisalo", ()->changeState(new EraserState(dm)))));

this.add(t, BorderLayout.PAGE\_START);

}

private void Myimport() throws IOException {

Path filePath = getPath();

byte[] bytes = Files.readAllBytes(filePath);

String text = new String(bytes, StandardCharsets.UTF\_8);

List<String> rows = new ArrayList<String>(Arrays.asList(text.split("\n")));

Stack<GraphicalObject> stog = new Stack<GraphicalObject>();

try {

for(String row: rows) {

row = row.trim();

String[] parts = row.split(" ",2);

if(parts.length != 2) throw new IllegalArgumentException("Nepoznati row");

GraphicalObject go = TAGS.get(parts[0]);

if(go == null) throw new IllegalArgumentException("TAG: "+ parts[0] +" nije pronađen.");

go.load(stog, parts[1]);

}

}catch (IllegalArgumentException e) {

e.printStackTrace();

return;

}

for (GraphicalObject graphicalObject : stog) {

dm.addGraphicalObject(graphicalObject);

dm.notifyListeners();

}

}

private void Myexport() throws IOException {

Path filePath = getPath();

List<String> output = new ArrayList<String>();

for(GraphicalObject o: dm.list()) {

o.save(output);

}

byte[] podatci = output.stream().collect(Collectors.joining("\n")).getBytes(StandardCharsets.UTF\_8);

Files.write(filePath, podatci);

}

private void SVGexport() throws IOException {

Path filePath = getPath();

SVGRendererImpl r = new SVGRendererImpl(filePath);

for(GraphicalObject o: dm.list()) {

o.render(r);

}

r.close();

}

private Path getPath() throws IOException {

Path filePath = fileChoice(this);

if(filePath == null) return null;

if(!Files.exists(filePath)) Files.createFile(filePath);

if(!Files.isReadable(filePath)) throw new IllegalArgumentException("Datoteka: " + filePath.toAbsolutePath() + "ne postoji!");

return filePath;

}

private Action createAction(String name,Runnable onPress) {

return new AbstractAction(name) {

private static final long serialVersionUID = 1L;

@Override

public void actionPerformed(ActionEvent e) {

onPress.run();

platno.requestFocusInWindow();

}

};

}

private void changeState(State newState) {

this.currentState.onLeaving();

this.currentState = newState;

dm.notifyListeners();

}

private static Path fileChoice(Component parent) {

JFileChooser fc = new JFileChooser();

fc.setDialogTitle("Open file");

if(fc.showOpenDialog(parent)!=JFileChooser.APPROVE\_OPTION) {

return null;

}

File fileName = fc.getSelectedFile();

Path filePath = fileName.toPath();

return filePath;

}

public static void main(String[] args) {

Runnable r = new Runnable() {

@Override

public void run() {

List<GraphicalObject> objects = new ArrayList<>();

objects.add(new LineSegment());

objects.add(new Oval());

GUI gui = new GUI(objects);

gui.setVisible(true);

gui.platno.requestFocusInWindow();

}

};

SwingUtilities.invokeLater(r);

}

}

---------------- G2DRendererImpl.java----------------

public class G2DRendererImpl implements Renderer {

private Graphics2D g2d;

public G2DRendererImpl(Graphics2D g2d) {

this.g2d = g2d;

}

@Override

public void drawLine(Point s, Point e) {

g2d.setColor(Color.blue);

g2d.drawLine(s.getX(), s.getY(), e.getX(), e.getY());

}

@Override

public void fillPolygon(Point[] points) {

int[] xs = Arrays.stream(points).mapToInt(Point::getX).toArray();

int[] ys = Arrays.stream(points).mapToInt(Point::getY).toArray();

g2d.setColor(Color.blue);

g2d.fillPolygon(xs, ys, points.length);

g2d.setColor(Color.red);

g2d.drawPolygon(xs, ys, points.length);

}}

------------------ GeometryUtil.java-------------------

public class GeometryUtil {

public static double distanceFromPoint(Point point1, Point point2) {

int dx = point1.getX() - point2.getX();

int dy = point1.getY() - point2.getY();

return Math.hypot(dx, dy);

}

public static double distanceFromLineSegment(Point s, Point e, Point p) {

if(s.compareTo(e) > 0) {

Point t = s;

s = e;

e = t;

}

if(isBellowOrAboveLine(s, e, p)) {

return distanceBetweenLineAndPoint(s, e, p);

}

return Math.min(distanceFromPoint(s, p), distanceFromPoint(e, p));

}

private static boolean isBellowOrAboveLine(Point pointOnLine1, Point pointOnLine2, Point testPoint) {

double k = coefficentLine(pointOnLine1, pointOnLine2);

double kVertical = -1.0 / k;

return testPoint.getY() > kVertical \* (testPoint.getX() - pointOnLine1.getX()) + pointOnLine1.getY() &&

testPoint.getY() < kVertical \* (testPoint.getX() - pointOnLine2.getX()) + pointOnLine2.getY();

}

private static double coefficentLine(Point pointOnLine1, Point pointOnLine2) {

return (pointOnLine2.getY() - pointOnLine1.getY()) \* 1.0 / (pointOnLine2.getX() - pointOnLine1.getX());

}

private static double distanceBetweenLineAndPoint(Point pointOnLine1, Point pointOnLine2, Point point) {

double k = coefficentLine(pointOnLine1, pointOnLine2);

if(k == Double.NEGATIVE\_INFINITY || k == Double.POSITIVE\_INFINITY) return Math.abs(point.getX() - pointOnLine1.getX());

if(k ==Double.NaN) return distanceFromPoint(pointOnLine1, point);

return Math.abs(-k \* point.getX() + point.getY() - pointOnLine1.getY() + k \* pointOnLine1.getX()) / Math.sqrt(k\*k +1);

}

public static double selectinDistance(Point mousePoint, Rectangle boundingBox) {

Point A = new Point(boundingBox.getX(),boundingBox.getY());

Point B = new Point(A.getX(), A.getY() + boundingBox.getHeight());

Point C = new Point(A.getX() + boundingBox.getWidth(), A.getY() + boundingBox.getHeight());

Point D = new Point(A.getX() + boundingBox.getWidth(), A.getY());

double AB = GeometryUtil.distanceFromLineSegment(A, B, mousePoint);

double BC = GeometryUtil.distanceFromLineSegment(B, C, mousePoint);

double CD = GeometryUtil.distanceFromLineSegment(C, D, mousePoint);

double DA = GeometryUtil.distanceFromLineSegment(D, A, mousePoint);

if(AB <= boundingBox.getHeight() && CD <= boundingBox.getHeight()

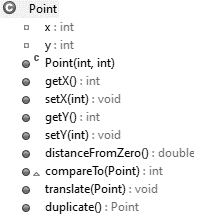
&& BC <= boundingBox.getWidth() && DA <=boundingBox.getWidth()) return 0.0;

return Math.min(AB, Math.min(BC, Math.min(CD, DA)));

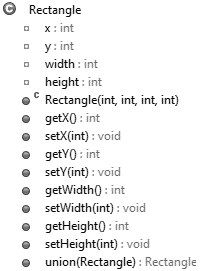
}

}

--------------Point.java---------------------



----------------- Rectangle.java ------------



-----------Renderer.java-------------



-------------------- AbstractGraphicalObject.java ------------

public abstract class AbstractGraphicalObject implements GraphicalObject{

Point[] hotPoints;

boolean[] hotPointSelected;

boolean selected;

List<GraphicalObjectListener> listeners = new ArrayList<>();

public AbstractGraphicalObject(Point... hotPoints) {

super();

this.hotPoints = hotPoints;

}

@Override

public Point getHotPoint(int index) {

return hotPoints[index];

}

@Override

public void setHotPoint(int index, Point point) {

hotPoints[index] = point;

notifyListeners();

}

@Override

public int getNumberOfHotPoints() {

return hotPoints.length;

}

@Override

public double getHotPointDistance(int index, Point mousePoint) {

return GeometryUtil.distanceFromPoint(hotPoints[index], mousePoint);

}

@Override

public boolean isHotPointSelected(int index) {

return hotPointSelected[index];

}

@Override

public void setHotPointSelected(int index, boolean selected) {

hotPointSelected[index] = selected;

}

@Override

public boolean isSelected() {

return selected;

}

@Override

public void setSelected(boolean selected) {

this.selected = selected;

notifySelectionListeners();

}

@Override

public void translate(Point delta) {

for (Point point : hotPoints) {

point.translate(delta);

}

notifyListeners();

}

@Override

public void addGraphicalObjectListener(GraphicalObjectListener l) {

listeners.add(l);

}

@Override

public void removeGraphicalObjectListener(GraphicalObjectListener l) {

listeners.remove(l);

}

public void notifyListeners() {

for (GraphicalObjectListener l : listeners) {

l.graphicalObjectChanged(this);

}

}

public void notifySelectionListeners() {

for (GraphicalObjectListener l : listeners) {

l.graphicalObjectSelectionChanged(this);

}

}

}

----------------- CompositeShape.java-------------

public class CompositeShape extends AbstractGraphicalObject {

private List<GraphicalObject> shapes;

public CompositeShape(List<GraphicalObject> shapes) {

super();

this.shapes = shapes;

}

@Override

public Rectangle getBoundingBox() {

Rectangle compositeRectangle = null;

for(GraphicalObject obj : shapes) {

if(compositeRectangle == null)

compositeRectangle = obj.getBoundingBox();

else

compositeRectangle = compositeRectangle.union(obj.getBoundingBox());

}

return compositeRectangle;

}

@Override

public double selectionDistance(Point mousePoint) {

Rectangle boundingBox = getBoundingBox();

return GeometryUtil.selectinDistance(mousePoint, boundingBox);

}

@Override

public void render(Renderer r) {

shapes.forEach(g -> g.render(r));

}

@Override

public String getShapeName() {

// TODO Auto-generated method stub

return null;

}

@Override

public GraphicalObject duplicate() {

// TODO Auto-generated method stub

return null;

}

@Override

public String getShapeID() {

return "@COMP";

}

@Override

public void load(Stack<GraphicalObject> stack, String data) {

String[] parts = data.split(" ");

if(parts.length != 1) throw new IllegalArgumentException("Shape: "+getShapeID()+" očekuje 1 argumenta");

try {

int size = Integer.parseInt(parts[0]);

List<GraphicalObject> shapes = IntStream.range(0, size).mapToObj(i -> stack.pop()).collect(Collectors.toList());

stack.push(new CompositeShape(shapes));

} catch (NumberFormatException e) {

throw new IllegalArgumentException(e);

}

}

@Override

public void save(List<String> rows) {

rows.add(getShapeID() + " " + shapes.size());

}

public List<GraphicalObject> getShapes() {

return shapes;

}

}

----------------- DocumentModel.java -------------------

public class DocumentModel {

public final static double SELECTION\_PROXIMITY = 10;

// Kolekcija svih grafičkih objekata:

private List<GraphicalObject> objects = new ArrayList<>();

// Read-Only proxy oko kolekcije grafičkih objekata:

private List<GraphicalObject> roObjects = Collections.unmodifiableList(objects);

// Kolekcija prijavljenih promatrača:

private List<DocumentModelListener> listeners = new ArrayList<>();

// Kolekcija selektiranih objekata:

private List<GraphicalObject> selectedObjects = new ArrayList<>();

// Read-Only proxy oko kolekcije selektiranih objekata:

private List<GraphicalObject> roSelectedObjects = Collections.unmodifiableList(selectedObjects);

// Promatrač koji će biti registriran nad svim objektima crteža...

private final GraphicalObjectListener goListener = new GraphicalObjectListener() {

@Override

public void graphicalObjectChanged(GraphicalObject go) {

notifyListeners();

}

@Override

public void graphicalObjectSelectionChanged(GraphicalObject go) {

if(go.isSelected()) selectedObjects.add(go);

else selectedObjects.remove(go);

notifyListeners();

}

};

// Konstruktor...

public DocumentModel() {}

// Brisanje svih objekata iz modela (pazite da se sve potrebno odregistrira)

// i potom obavijeste svi promatrači modela

public void clear() {

for(GraphicalObject obj: objects) {

removeGraphicalObject(obj);

}

notifyListeners();

}

// Dodavanje objekta u dokument (pazite je li već selektiran; registrirajte model kao promatrača)

public void addGraphicalObject(GraphicalObject obj) {

if(obj.isSelected()) {

selectedObjects.add(obj);

}

objects.add(obj);

obj.addGraphicalObjectListener(goListener);

}

// Uklanjanje objekta iz dokumenta (pazite je li već selektiran; odregistrirajte model kao promatrača)

public void removeGraphicalObject(GraphicalObject obj) {

if(obj.isSelected()) {

selectedObjects.remove(obj);

}

objects.remove(obj);

obj.removeGraphicalObjectListener(goListener);

}

// Vrati nepromjenjivu listu postojećih objekata (izmjene smiju ići samo kroz metode modela)

public List<GraphicalObject> list() {

return roObjects;

}

// Prijava...

public void addDocumentModelListener(DocumentModelListener l) {

listeners.add(l);

}

// Odjava...

public void removeDocumentModelListener(DocumentModelListener l) {

listeners.remove(l);

}

// Obavještavanje...

public void notifyListeners() {

for(DocumentModelListener l : listeners) {

l.documentChange();

}

}

// Vrati nepromjenjivu listu selektiranih objekata

public List<GraphicalObject> getSelectedObjects() {

return roSelectedObjects;

}

// Pomakni predani objekt u listi objekata na jedno mjesto kasnije...

// Time će se on iscrtati kasnije (pa će time možda veći dio biti vidljiv)

public void increaseZ(GraphicalObject go) {

moveZForDelta(go, 1);

notifyListeners();

}

// Pomakni predani objekt u listi objekata na jedno mjesto ranije...

public void decreaseZ(GraphicalObject go) {

moveZForDelta(go, -1);

notifyListeners();

}

private void moveZForDelta(GraphicalObject go, int delta) {

int index = objects.indexOf(go);

if(index == -1) return;

if(index + delta < 0 || index + delta > objects.size() -1 ) return;

objects.remove(index);

objects.add(index + delta ,go);

}

// Pronađi postoji li u modelu neki objekt koji klik na točku koja je

// predana kao argument selektira i vrati ga ili vrati null. Točka selektira

// objekt kojemu je najbliža uz uvjet da ta udaljenost nije veća od

// SELECTION\_PROXIMITY. Status selektiranosti objekta ova metoda NE dira.

public GraphicalObject findSelectedGraphicalObject(Point mousePoint) {

double min = SELECTION\_PROXIMITY;

GraphicalObject goRet = null;

for(GraphicalObject go: objects) {

if(go.selectionDistance(mousePoint) < min) {

min = go.selectionDistance(mousePoint);

goRet = go;

}

}

return goRet;

}

// Pronađi da li u predanom objektu predana točka miša selektira neki hot-point.

// Točka miša selektira onaj hot-point objekta kojemu je najbliža uz uvjet da ta

// udaljenost nije veća od SELECTION\_PROXIMITY. Vraća se indeks hot-pointa

// kojeg bi predana točka selektirala ili -1 ako takve nema. Status selekcije

// se pri tome NE dira.

public int findSelectedHotPoint(GraphicalObject object, Point mousePoint) {

double min = SELECTION\_PROXIMITY;

int index = -1;

for(int i = 0; i< object.getNumberOfHotPoints(); i++) {

double hp = GeometryUtil.distanceFromPoint(object.getHotPoint(i),mousePoint);

if(hp < min) {

index = i;

min = hp;

}

}

return index;

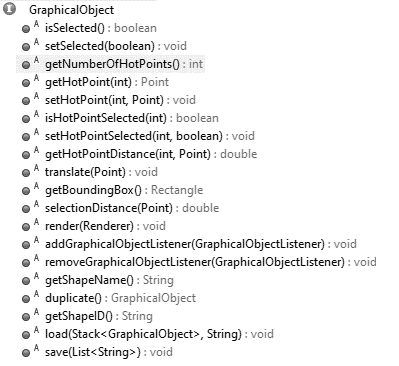
}

}

----------------- DocumnetModelListener.java---------



--------------GraphicalObject.java -------------



------------------GraphicalObjectListener.java-------



-------------------LineSegment.java----------------

public class LineSegment extends AbstractGraphicalObject{

public LineSegment(Point start, Point end) {

super(start,end);

// TODO Auto-generated constructor stub

}

public LineSegment() {

this(new Point(0, 0), new Point(10, 0));

}

@Override

public Rectangle getBoundingBox() {

int minX = Math.min(getHotPoint(0).getX(), getHotPoint(1).getX());

int minY = Math.min(getHotPoint(0).getY(), getHotPoint(1).getY());

int maxX = Math.max(getHotPoint(0).getX(), getHotPoint(1).getX());

int maxY = Math.max(getHotPoint(0).getY(), getHotPoint(1).getY());

return new Rectangle(minX, minY, maxX-minX, maxY-minY);

}

@Override

public double selectionDistance(Point mousePoint) {

return GeometryUtil.distanceFromLineSegment(getHotPoint(0), getHotPoint(1), mousePoint);

}

@Override

public void render(Renderer r) {

r.drawLine(getHotPoint(0), getHotPoint(1));

}

@Override

public String getShapeName() {

return "Linija";

}

@Override

public GraphicalObject duplicate() {

return new LineSegment(getHotPoint(0).duplicate(), getHotPoint(1).duplicate());

}

@Override

public String getShapeID() {

return "@LINE";

}

@Override

public void load(Stack<GraphicalObject> stack, String data) {

String[] parts = data.split(" ");

if(parts.length != 4) throw new IllegalArgumentException("Shape: "+getShapeID()+" očekuje 4 argumenta");

try {

Point s = new Point(Integer.parseInt(parts[0]), Integer.parseInt(parts[1]));

Point e = new Point(Integer.parseInt(parts[2]), Integer.parseInt(parts[3]));

stack.push(new LineSegment(s, e));

} catch (NumberFormatException e) {

throw new IllegalArgumentException(e);

}

}

@Override

public void save(List<String> rows) {

Point s = getHotPoint(0);

Point e = getHotPoint(1);

String line = getShapeID() + " " +s.getX()+ " " +s.getY()+ " " + e.getX() + " " + e.getY();

rows.add(line);}}

-------------- Oval.java-----------------

public class Oval extends AbstractGraphicalObject{

public Oval(Point down, Point right) {

super(down,right);

}

public Oval() {

this(new Point(0, 10), new Point(10, 0));

}

@Override

public Rectangle getBoundingBox() {

int heigth = (getHotPoint(1).getX() - getHotPoint(0).getX()) \* 2;

int weigth = (getHotPoint(0).getY() - getHotPoint(1).getY()) \*2 ;

return new Rectangle(getHotPoint(1).getX() - heigth, getHotPoint(0).getY() - weigth, heigth, weigth);

}

@Override

public double selectionDistance(Point mousePoint) {

Rectangle boundingBox = getBoundingBox();

return GeometryUtil.selectinDistance(mousePoint, boundingBox);

}

@Override

public void render(Renderer r) {

Rectangle boundingBox = getBoundingBox();

int p = getHotPoint(0).getX();

int q = getHotPoint(1).getY();

Point[] points = eclipsePoints(p, q, boundingBox.getWidth() / 2, boundingBox.getHeight() / 2);

r.fillPolygon(points);

}

// srediste S(p,q), a velika poluos, b mala poluos

private Point[] eclipsePoints(int p,int q, int a, int b) {

List<Point> points = new ArrayList<Point>();

int lastX = 0;

for(int y = q + b; y >= q - b; y--) {

double x = a \* 1.0 /b \* Math.sqrt(b\*b - Math.pow((y - q),2));

int xInt = Math.round((float)x);

// if(y != q + b && xInt == lastX) continue;

// lastX = xInt;

points.add(new Point(xInt + p, y));

}

return Stream.concat(

points.stream().skip(1),

IntStream.

range(0, points.size()).

mapToObj(i -> {

int index = points.size() - i - 1;

Point point = points.get(index);

return new Point((point.getX() - p) \* -1 + p, point.getY());

}).skip(1)).toArray(Point[]::new);

}

@Override

public String getShapeName() {

return "Oval";

}

@Override

public GraphicalObject duplicate() {

return new Oval(getHotPoint(0).duplicate(), getHotPoint(1).duplicate());

}

@Override

public String getShapeID() {

return "@OVAL";

}

@Override

public void load(Stack<GraphicalObject> stack, String data) {

String[] parts = data.split(" ");

if(parts.length != 4) throw new IllegalArgumentException("Shape: "+getShapeID()+" očekuje 4 argumenta");

try {

Point right = new Point(Integer.parseInt(parts[0]), Integer.parseInt(parts[1]));

Point down = new Point(Integer.parseInt(parts[2]), Integer.parseInt(parts[3]));

stack.push(new Oval(down,right));

} catch (NumberFormatException e) {

throw new IllegalArgumentException(e);

}

}

@Override

public void save(List<String> rows) {

Point s = getHotPoint(1);

Point e = getHotPoint(0);

String line = getShapeID() + " " +s.getX()+ " " +s.getY()+ " " + e.getX() + " " + e.getY();

rows.add(line);

}

}

------------------ SVGRendererImpl.java ---------------

public class SVGRendererImpl implements Renderer {

private List<String> lines = new ArrayList<>();

private Path filePath;

public SVGRendererImpl(Path filePath) {

this.filePath = filePath;

lines.add(

"""

<svg xmlns="http://www.w3.org/2000/svg"

xmlns:xlink="http://www.w3.org/1999/xlink">

"""

);

}

public void close() throws IOException {

lines.add("</svg>");

if(filePath != null) {

byte[] podatci = lines.stream().collect(Collectors.joining("\n")).getBytes(StandardCharsets.UTF\_8);

Files.write(filePath, podatci);

}

// u lines još dodaj završni tag SVG dokumenta: </svg>

// sve retke u listi lines zapiši na disk u datoteku

// ...

}

@Override

public void drawLine(Point s, Point e) {

lines.add("<line x1=\""+s.getX()+"\" y1=\""+s.getY()+"\" x2=\""+e.getX()+"\" y2=\""+e.getY()+"\" style=\"stroke:#0000FF;\"/>");

}

@Override

public void fillPolygon(Point[] points) {

String start = "<polygon points=\"";

String end = "\" style=\"stroke:#FF0000; fill:#0000FF;\"/>";

lines.add(Stream.of(points).

map(p -> p.getX() + "," + p.getY()).

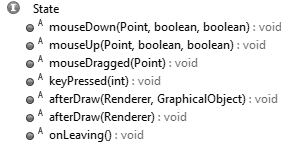
collect(Collectors.joining(" ",start,end))

);

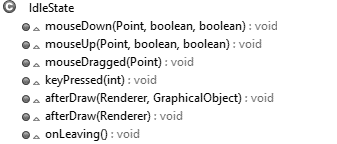
}

}

------------State.java--------------



---------------IdleState.java-----------



----------------- AddShapeState.java---------------

public class AddShapeState extends IdleState {

private GraphicalObject prototype;

private DocumentModel model;

public AddShapeState(GraphicalObject prototype, DocumentModel model) {

super();

this.prototype = prototype;

this.model = model;

}

@Override

public void mouseDown(Point mousePoint, boolean shiftDown, boolean ctrlDown) {

GraphicalObject og = prototype.duplicate();

model.addGraphicalObject(og);

og.translate(mousePoint);

}

}

-------------------EraserState.java-------------

public class EraserState extends IdleState {

private DocumentModel model;

Set<GraphicalObject> removeObjects = new HashSet<GraphicalObject>();

List<Point> points = new ArrayList<Point>();

public EraserState(DocumentModel model) {

super();

this.model = model;

}

@Override

public void mouseDragged(Point mousePoint) {

points.add(mousePoint);

model.notifyListeners();

GraphicalObject go = model.findSelectedGraphicalObject(mousePoint);

if(go == null) return;

removeObjects.add(go);

}

@Override

public void mouseUp(Point mousePoint, boolean shiftDown, boolean ctrlDown) {

for(GraphicalObject go : removeObjects) {

model.removeGraphicalObject(go);

}

points.clear();

model.notifyListeners();

}

@Override

public void afterDraw(Renderer r) {

for(int i = 1; i<points.size();i++) {

r.drawLine(points.get(i-1), points.get(i));

}

}

@Override

public void onLeaving() {

removeObjects.clear();

points.clear();

}

}

-------------------- SelectShapeState.java---------------

public class SelectShapeState extends IdleState {

private static final Map<Integer, Runnable> KEYS = new HashMap<>();

public void createKeys() {

KEYS.put(KeyEvent.VK\_PLUS,() -> {

if(model.getSelectedObjects().size() != 1) return;

GraphicalObject go = model.getSelectedObjects().get(0);

model.increaseZ(go);

});

KEYS.put(KeyEvent.VK\_MINUS,() -> {

if(model.getSelectedObjects().size() != 1) return;

GraphicalObject go = model.getSelectedObjects().get(0);

model.decreaseZ(go);

});

KEYS.put(KeyEvent.VK\_G, () -> {

List<GraphicalObject> objects = new ArrayList<GraphicalObject>(model.getSelectedObjects());

for(GraphicalObject obj: objects) {

model.removeGraphicalObject(obj);

}

GraphicalObject go = new CompositeShape(objects);

model.addGraphicalObject(go);

go.setSelected(true);

model.notifyListeners();

});

KEYS.put(KeyEvent.VK\_U, () -> {

if(model.getSelectedObjects().size() != 1) return;

GraphicalObject go = model.getSelectedObjects().get(0);

if(!(go instanceof CompositeShape)) return;

CompositeShape cs = (CompositeShape) go;

List<GraphicalObject> objects = cs.getShapes();

model.removeGraphicalObject(go);

for(GraphicalObject object: objects) {

model.addGraphicalObject(object);

object.setSelected(true);

}

model.notifyListeners();

});

}

private DocumentModel model;

private static final int HP\_SIZE = 4;

public SelectShapeState(DocumentModel model) {

super();

this.model = model;

createKeys();

}

@Override

public void mouseDown(Point mousePoint, boolean shiftDown, boolean ctrlDown) {

if(!ctrlDown) {

onLeaving();

}

GraphicalObject newGo= model.findSelectedGraphicalObject(mousePoint);

if(newGo == null) return;

newGo.setSelected(true);

}

@Override

public void mouseDragged(Point mousePoint) {

if(model.getSelectedObjects().size() == 1) {

GraphicalObject go = model.getSelectedObjects().get(0);

int indexSelectedHotPoint = model.findSelectedHotPoint(go, mousePoint);

if(indexSelectedHotPoint == -1) return;

go.setHotPoint(indexSelectedHotPoint, mousePoint);

}

}

@Override

public void keyPressed(int keyCode) {

//System.out.println(keyCode);

Runnable action = KEYS.get(keyCode);

if(action != null) action.run();

}

@Override

public void afterDraw(Renderer r, GraphicalObject go) {

if(go.isSelected()) {

Rectangle boundingBox = go.getBoundingBox();

//Kutevi Oval

Point A = new Point(boundingBox.getX(),boundingBox.getY());

Point B = new Point(A.getX(), A.getY() + boundingBox.getHeight());

Point C = new Point(A.getX() + boundingBox.getWidth(), A.getY() + boundingBox.getHeight());

Point D = new Point(A.getX() + boundingBox.getWidth(), A.getY());

r.drawLine(A, B);

r.drawLine(B, C);

r.drawLine(C, D);

r.drawLine(D, A);

if(model.getSelectedObjects().size() == 1) drawHotPoints(r,go);

}

}

private void drawHotPoints(Renderer r,GraphicalObject go) {

for(int i = 0; i < go.getNumberOfHotPoints(); i++) {

Point hp = go.getHotPoint(i);

Point A = new Point(hp.getX() - HP\_SIZE, hp.getY() - HP\_SIZE);

Point B = new Point(hp.getX() + HP\_SIZE, hp.getY() - HP\_SIZE);

Point C = new Point(hp.getX() + HP\_SIZE, hp.getY() + HP\_SIZE);

Point D = new Point(hp.getX() - HP\_SIZE, hp.getY() + HP\_SIZE);

r.drawLine(A, B);

r.drawLine(B, C);

r.drawLine(C, D);

r.drawLine(D, A);

}

}

@Override

public void onLeaving() {

List<GraphicalObject> objects = new ArrayList<GraphicalObject>(model.getSelectedObjects());

for(GraphicalObject obj: objects) {

obj.setSelected(false);

}

}

}