**Criterion C: Development**

## Developing the Product

Figure #1: Formalized diagram of the program's layout


### General aspects of the program

Figure #2: Properties dialog for "Mind Mapper.jar" in Microsoft Windows 10

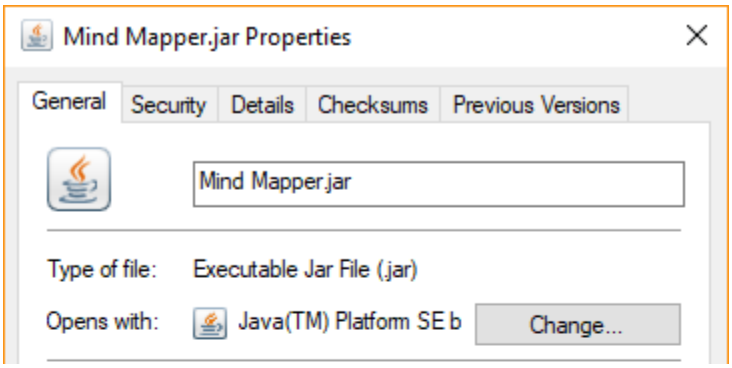


Table #1: Glossary of terms that I refer to

| Term | Definition |
|---|---|
| Mind map | The working document in the program |
| Mind map elements | The shapes and connections in the canvas |
| Mind map data | The data of all mind map elements |
| Canvas | The canvas where mind map elements are placed |
| Viewport | The visible portion of the canvas where mind map elements are drawn |
| Viewport data | The settings for offset in the x and y-axis for panning, and zoom |
| Draw | Synonymous with “paint” for rendering GUI elements on-screen |

List of advanced techniques used:

- Java Swing for all aspects of the GUI
- Encapsulation
- Runtime/compile-time polymorphism, inheritance, abstraction
- Custom listeners
- ArrayLists
- Reflection
- File input/output
- The external Google Gson library
- Universally unique identifiers (UUIDs)
- Hash tables

## Explanation of internal program mechanisms and algorithms

Figure #3: Main entry point (main method) into the program (Main.java)

```
public static void main(String args[]) {
    EventQueue.invokeLater(new Runnable() {
        public void run() {
            try {
                new AppFrame().setVisible(true);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
}
```

I use the Java EventQueue in the main method to dispatch all GUI related tasks (drawing and mouse listeners) to a separate thread called the Event Dispatching Thread (EDT). I used this

so that the GUI would not be blocked by long lasting calculations in the main thread [1] and would remain highly responsive as part of success criterion #7.

Figure #4: Constructor of the main window frame of the program (AppFrame.java)

```
public AppFrame() {
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setAppTitle(null);

    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    this.setBounds(50, 50, screenSize.width - 100, screenSize.height - 100);
    this.setLayout(new BorderLayout());

    initComponents();
}
```

The AppFrame class inherits from the JFrame class, allowing it to act as a JFrame itself to provide a simplified structure to GUI development. When created, a simple algorithm centers the window in the middle of the screen with a 50 pixel radius around it.

Figure #5: The method responsible for the creation of the overall layout (AppFrame.java)

```
private void initComponents() {
    try {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        UIManager.put("OptionPane.background", Color.white);
        UIManager.put("Panel.background", Color.white);
    } catch (Exception e) {
        e.printStackTrace();
    }

    canvasPanel = new CanvasPanel(this);
    this.add(canvasPanel, BorderLayout.CENTER);

    pickerPanel = new PickerPanel(canvasPanel);
    this.add(pickerPanel, BorderLayout.WEST);

    debugPanel = new DebugPanel();
    this.add(debugPanel, BorderLayout.EAST);

    ioCon = new IOController(this, canvasPanel);

    menubar = new Menubar(this);
    this.setJMenuBar(menubar);
}
```

[1] (Quillion, 2014)

I use the Java UIManager to dynamically set the program look and feel, so buttons and menus would appear similar to other programs on my client's target OS. All panels are organized with cardinal directions in a BorderLayout.

Figure #6: Example of the usage of instance references (CanvasPanel.java, AppFrame.java)

```
private AppFrame appFrame;

public CanvasPanel(AppFrame appFrame) {
    this.appFrame = appFrame;
    contextMenu = new ContextMenu(this);
    reset();
}
```

```
public class AppFrame extends JFrame {

    private static Menubar menubar;
    private static PickerPanel pickerPanel;
    private static CanvasPanel canvasPanel;
    private static DebugPanel debugPanel;
    private static IOController ioCon;
```

To align with OOP, instance references of panels and components are stored in the AppFrame and are passed along to the constructors of panels. I could then access methods from another class via the reference from AppFrame instead of setting everything as static.

Figure #7: A public method that sets the title of the program (AppFrame.java)

```
public void setAppTitle(String fileName) {
    if (fileName == null) this.setTitle(appName);
    else this.setTitle(fileName + " - " + appName);
}
```

I utilize encapsulation throughout the entire program to prevent direct access to object data (private, protected). Publicly-accessible methods verify and facilitate access (getters and setters). This setter adds the current working document's filename to the program title.

Figure #8: Sample getter and setter (MapShape.java)

```
public UUID getId() {
    return id;
}
public void setId(UUID id) {
    this.id = id;
}
```

Figure #9: Sample snippets of menu item creation (Menubar.java)

```
newFile = new JMenuItem("New");
newFile.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_N, KeyEvent.CTRL_DOWN_MASK));
newFile.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        appFrame.getCanvasPanel().reset();
    }
});
fileMenu.add(newFile);
```

```
centerCanvas = new JMenuItem("Center Canvas");
centerCanvas.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_PERIOD, KeyEvent.CTRL_MASK));
centerCanvas.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        appFrame.getCanvasPanel().getViewport().centerView();
    }
});
viewMenu.add(centerCanvas);
```

User interaction with menu items are handled with an ActionListener that simply calls the public method of a related class to perform the action.

Figure #10: Access modifiers and data elements within a MapShape (MapShape.java)

```
public abstract class MapShape {

    private UUID id;
    protected Shape shape;
    protected int x, y;
    private BasicStroke borderStroke;
    private Color borderColour;
    private JTextField textField;
    public boolean isHighlighted = false;
```

For encapsulation, I set different access modifiers to variables and methods depending on whether additional processing must run in a getter/setter. For example, the variable "isHighlighted" is simple enough to remain public.

Figure #11: Usage of polymorphism and abstraction

```
public abstract void setNewCoordinates(int x, int y);   // Force subclasses (shapes) to override this
```

The abstract class MapShape, which cannot be instantiated by itself, contains an abstract method that all implementations of it must override. I use this polymorphism and abstraction to allow different shapes to interface with the viewport, be drawn on screen and be saved/restored

in JSON files. All shapes inherit common variables but have individual implementations of their geometry.

Figure #12: Complete EllipseShape class that highlights polymorphism (EllipseShape.java)

```
public class EllipseShape extends MapShape {

    public EllipseShape(int x, int y, int width, int height) {
        super(new Ellipse2D.Double(x, y, width, height));
    }

    @Override
    public void setNewCoordinates(int x, int y) {
        this.x = x;
        this.y = y;
        shape = new Ellipse2D.Double(x, y, shape.getBounds().width, shape.getBounds().height);
        updateTextFieldBounds();
    }

}
```

The EllipseShape class inherits from MapShape, allowing it to set “this.x”.

Figure #13: Runtime overridden paint method (CanvasPanel.java)

```
@Override
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    viewport.drawAll(g);        // Draw all shapes, text, and lines
}
```

Runtime polymorphism of the Swing “paintComponent” method is used in the CanvasPanel to first draw itself (super method), then draw all mind map elements with a helper method in the Viewport class.

Figure #14: Method responsible for drawing mind map elements, and variables (Viewport.java)

```
// Zoom and pan variables
private boolean zooming;
public double zoomFactor = 1, prevZoomFactor = 1;
private boolean released;
public int xOffset = 0, yOffset = 0;
protected Point panStartPoint;
protected int panXDiff, panYDiff;
```

```
public void drawAll(Graphics g) {
    Graphics2D g2d = (Graphics2D) g;
    g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);

    AffineTransform at = new AffineTransform();
    if (zooming) {              // Handle zooming relative to cursor
        double xRel = MouseInfo.getPointerInfo().getLocation().getX() - canvasPanel.getX();
        double yRel = MouseInfo.getPointerInfo().getLocation().getY() - canvasPanel.getY();
        double zoomDiv = zoomFactor / prevZoomFactor;
        xOffset = (int)((zoomDiv) * (xOffset) + (1 - zoomDiv) * xRel);
        yOffset = (int)((zoomDiv) * (yOffset) + (1 - zoomDiv) * yRel);

        prevZoomFactor = zoomFactor;
        zooming = false;
    }                           // If released, reset pan diff
    if (released) {
        xOffset += panXDiff;
        yOffset += panYDiff;
        panXDiff = 0;
        panYDiff = 0;
    }
    at.translate(xOffset + panXDiff, yOffset + panYDiff);
    at.scale(zoomFactor, zoomFactor);
    g2d.transform(at);

    // Iterate and print all lines before shapes
    for (MapLine connection : canvasPanel.getMapController().getConnections()) {
        connection.updateConnection();
        g2d.setColor(Color.black);
        g2d.setStroke(connection.getStroke());
        g2d.draw(connection.getLine());
    }
    for (MapShape mapShape : canvasPanel.getMapController().getShapes()) {
        // Fill shape background with white to hide lines within the shape
        g2d.setColor(Color.white);
        g2d.fill(mapShape.getShape());
        // Draw border around shape
        if (mapShape.isHighlighted) g2d.setColor(Color.cyan);
        else g2d.setColor(mapShape.getBorderColour());
        g2d.setStroke(mapShape.getBorderStroke());
        g2d.draw(mapShape.getShape());
        // Draw text
        drawShapeText(g, mapShape);
    }
    if (!initGrid) {                            // Calculate new offset for grid to be centered
        gridOffsetX = canvasPanel.getWidth()/2;     // Only run once when canvasPanel is drawn on-screen
        gridOffsetY = canvasPanel.getHeight()/2;
        initGrid = true;
    }
    g2d.translate(gridOffsetX, gridOffsetY);        // Grid is now offset by a static amount
    if (showGrid) grid.drawGrid(g2d);
}
```

Mind map elements are drawn via the exposed Graphics object in the Swing "paintComponent" method. The algorithm responsible for zooming and panning in the canvas utilizes an AffineTransform object and calculates the relative transformations. When zooming, objects near the cursor move away at a slower rate than objects farther away, which provides the illusion of zooming in. The lines that connect shapes from center-to-center are drawn first, then the shapes are drawn with white backgrounds to conceal the line within the shape, along with border width and colour. Essentially, individual shapes have their own coordinates that govern their position, and everything is transformed at the end to fulfill panning.

Figure #15: Helper method for drawing text within shapes (Viewport.java)

```
private void drawShapeText(Graphics g, MapShape mapShape) {
    /**
     * Draw text from textfield only, if not being edited, or position the textfield correctly if being edited
     */
    if (! mapShape.equals(canvasPanel.getMapController().getEditingShape())) {
        // Offset the location of text fields to center of shape
        mapShape.getTextField().setBounds(mapShape.getX() + mapShape.getShape().getBounds().width/2 - 100 + xOffset,
                                          mapShape.getY() + mapShape.getShape().getBounds().height/2 - 50 + yOffset,
                                          200, 100);
        Graphics2D textGraphics = (Graphics2D) g.create(mapShape.getTextField().getBounds().x - xOffset,
                                          mapShape.getTextField().getBounds().y - yOffset,
                                          mapShape.getTextField().getBounds().width,
                                          mapShape.getTextField().getBounds().height);
        mapShape.getTextField().paint(textGraphics);
    } else {
        mapShape.getTextField().setBounds(mapShape.getX() + mapShape.getShape().getBounds().width/2 - 100,
                                          mapShape.getY() + mapShape.getShape().getBounds().height/2 - 50,
                                          200, 100);
    }
}
```

I use a helper method to draw the text within shapes. I ingeniously use a JTextField that is placed in the panel at the center of the shape to allow text to be edited. The JTextField handles drawing its own text; however, when the shape is not being edited, it is removed from the panel and only the text of it is drawn.

Figure #16: Frame rate limiter algorithm and related variables (Viewport.java)

```
private static final int MAX_FPS = 60;
private long lastFrameTime = 0;
```

```
protected void handleRepaint() {        // A handler to limit framerate and CPU usage
    // Calculate frame time and only repaint at the specified framerate
    if (System.currentTimeMillis() - lastFrameTime >= (1000/MAX_FPS)) {
        canvasPanel.repaint();
        lastFrameTime = System.currentTimeMillis();
    }
}
```

When the repaint method is called, Java Swing re-draws everything as fast as possible, which is costly to CPU usage if frequent. I implemented an algorithm that calculates and only runs repaint if the last time a frame is drawn is less than the targeted frame rate (60 FPS or $16.\bar{6}\ ms$). I run handleRepaint() when frequent repaints are needed, or force canvasPanel.repaint() to draw the final appearance, ensuring no unintended behavior.

Figure #17: Example of the benefit of limiting frequent repaints (Viewport.java)

```
public void pan(Point curPoint) {
    panXDiff = curPoint.x - panStartPoint.x;
    panYDiff = curPoint.y - panStartPoint.y;
    handleRepaint();
}
```

Figure #18: Example of the benefit of forcing a repaint (MapController.java)

```
public void removeSelectedShape() {
    shapes.remove(selectedShape);
    canvasPanel.repaint();
}
```

Figure #19: Algorithm for centering the canvas viewport (Viewport.java)

```
public void centerView() {
    // Calculate average center
    int numShapes = canvasPanel.getMapController().getShapes().size();
    int totalX = 0, totalY = 0;
    for (MapShape shape : canvasPanel.getMapController().getShapes()) {
        totalX += shape.getX() + shape.getShape().getBounds().getWidth()/2;
        totalY += shape.getY() + shape.getShape().getBounds().getHeight()/2;
    }
    xOffset = (int)-((totalX/numShapes - canvasPanel.getWidth()/2)/zoomFactor);
    yOffset = (int)-((totalY/numShapes - canvasPanel.getHeight()/2)/zoomFactor);

    canvasPanel.repaint();
}
```

This algorithm calculates the average coordinates of all shapes and pans the viewport to it.

Figure #20: Usage of a custom MapListener and linking to the CanvasPanel (CanvasPanel.java)

```
MapListener mapListener = new MapListener(this, viewport);
this.addMouseListener(mapListener);
this.addMouseMotionListener(mapListener);
this.addMouseWheelListener(mapListener);
```

Figure #21: All implemented mouse activity listener methods (MapListener.java)

```
public void mousePressed(MouseEvent evt) {
    viewport.panStartPoint = evt.getLocationOnScreen();
    viewport.setMouseReleased(false);

    selectShapeUnderCursor(evt);
    triggerContext(evt);
}
public void mouseReleased(MouseEvent evt) {
    viewport.setMouseReleased(true);
    canvasPanel.repaint();      // Bypass FPS limiter and force repaint to lock in position

    selectShapeUnderCursor(evt);
    triggerContext(evt);
}
public void mouseDragged(MouseEvent evt) {
    if (mapCon.getSelectedShape() == null && !canvasPanel.isContextTrigger) {          // Pan the canvas
        viewport.pan(evt.getLocationOnScreen());
    } else if (mapCon.getSelectedShape() != null && !canvasPanel.isContextTrigger) {    // Drag the selected shape
        Point curPoint = evt.getLocationOnScreen();
        if (curPoint.x  != mapCon.dragStartPoint.x || curPoint.y != mapCon.dragStartPoint.y) {
            mapCon.getSelectedShape().setNewCoordinates(                // Update coordinates
                    mapCon.getSelectedShape().getX() + (int)((curPoint.x - mapCon.dragStartPoint.x)/viewport.zoomFactor),
                    mapCon.getSelectedShape().getY() + (int)((curPoint.y - mapCon.dragStartPoint.y)/viewport.zoomFactor));
            mapCon.dragStartPoint = evt.getLocationOnScreen();          // Update drag diff reference
        }
        viewport.handleRepaint();
    }
}
public void mouseWheelMoved(MouseWheelEvent evt) {
    if (!canvasPanel.isContextTrigger) {
        if (evt.getWheelRotation() < 0) {               // Mouse wheel rolls forward
            viewport.zoomIn();
        } else if (evt.getWheelRotation() > 0) {        // Mouse wheel rolls backwards
            viewport.zoomOut();
        }
    }
}
public void mouseClicked(MouseEvent evt) {
    // Handle double-click
    if (evt.getClickCount() == 2 && mapCon.getSelectedShape() != null) {
        canvasPanel.add(mapCon.getSelectedShape().getTextField());
        mapCon.setEditingShape(mapCon.getSelectedShape());
        mapCon.getSelectedShape().getTextField().requestFocusInWindow();
        mapCon.getSelectedShape().getTextField().selectAll();
    }
}
```

A custom listener handles and responds accordingly to user interaction, mainly via the mouse. When shape is double-clicked on, I ingeniously add the JTextField of the selected shape to the canvas and focus on it.

Figure #22: Algorithm for selecting the shape under the cursor (MapListener.java)

```
private void selectShapeUnderCursor(MouseEvent evt) {
    // Select the shape that is clicked on
    if (mapCon.getShapesUnderCursor(evt.getPoint()).size() > 0) {
        mapCon.dragStartPoint = evt.getLocationOnScreen();          // Update drag diff reference
        if (mapCon.shapeSelectionIndex > mapCon.getShapesUnderCursor(evt.getPoint()).size() - 1)
            mapCon.shapeSelectionIndex = 0;     // Prevent index overflow by restarting cycle
        mapCon.setSelectedShape(mapCon.getShapesUnderCursor(evt.getPoint()).get(mapCon.shapeSelectionIndex));
        mapCon.shapeSelectionIndex++;               // Increment index to select overlapped shapes
    } else {    // Remove the selection
        for (Component com : canvasPanel.getComponents())
            if (com instanceof JTextField) canvasPanel.remove(com);     // Remove all text fields
        mapCon.setEditingShape(null);
        mapCon.setSelectedShape(null);
    }
}
```

If more than one shape is under the cursor, this algorithm cycles through the shapes and selects.

Figure #23: Algorithm for retrieving shapes under the cursor (MapController.java)

```
public List<MapShape> getShapesUnderCursor(Point cursor) {
    // Offset cursor to be consistent with shape location in viewport
    cursor.translate(-(int)viewport.xOffset, -(int)viewport.yOffset);
    cursor.x /= viewport.zoomFactor;
    cursor.y /= viewport.zoomFactor;

    List<MapShape> shapesUnderCursor = new ArrayList<MapShape>();
    for (MapShape shape : shapes)
        if (shape.getShape().getBounds().contains(cursor)) shapesUnderCursor.add(shape);
    return shapesUnderCursor;
}
```

I offset the cursor location to reflect the actual location after panning and zooming.

Figure #24: Constructor of the MapController class (MapController.java)

```
public MapController(CanvasPanel canvasPanel, Viewport viewport) {
    this.canvasPanel = canvasPanel;
    this.viewport = viewport;

    shapes = new ArrayList<MapShape>();
    connections = new ArrayList<MapLine>();
}
```

I use an ArrayList rather than a conventional array to store mind map elements and perform add and remove operations effectively.

Figure #25: Method responsible for adding a shape (MapController.java)

```
public void addShape(String shapeClassName) {
    int shapeWidth = 200, shapeHeight = 100;
    Point p = new Point();              // Point on screen to create the shape
    if (canvasPanel.isContextTrigger) {
        // Offset cursor to be consistent with shape location in viewport
        p.x = (int) ((canvasPanel.contextTriggerEvent.getX() - viewport.xOffset) / viewport.zoomFactor);
        p.y = (int) ((canvasPanel.contextTriggerEvent.getY() - viewport.yOffset) / viewport.zoomFactor);
    } else {
        // Start from center and find vacant location
        p.x = (int) ((canvasPanel.getWidth()/2 - viewport.xOffset) / viewport.zoomFactor);
        p.y = (int) ((canvasPanel.getHeight()/2 - viewport.yOffset) / viewport.zoomFactor);
        p = findVacantPoint(p);
    }

    try {
        // Create new MapShape at center of point using Java reflection
        Class<?> newMapShapeClass = Class.forName(shapeClassName);
        Constructor<?> newMapShapeCons = newMapShapeClass.getConstructor(
                                            new Class<?>[] {int.class, int.class, int.class, int.class});
        Object[] newMapShapeParameters = {p.x-(shapeWidth/2), p.y-(shapeHeight/2), shapeWidth, shapeHeight};
        MapShape newMapShape = (MapShape)newMapShapeCons.newInstance(newMapShapeParameters);

        // Continously repaint panel when editing to display changes in the text field
        newMapShape.getTextField().getDocument().addDocumentListener(new DocumentListener() {
            public void insertUpdate(DocumentEvent e) {
                canvasPanel.repaint();
            }
            public void removeUpdate(DocumentEvent e) {
                canvasPanel.repaint();
            }
            public void changedUpdate(DocumentEvent e) {
            }
        });

        shapes.add(newMapShape);

        setSelectedShape(null);
        setSelectedShape(newMapShape);      // Select the newly added shape
    } catch (Exception e) {
        e.printStackTrace();
    }

    canvasPanel.repaint();
}
```

Adding a shape via the right-click context menu adds it at the cursor location or adds at the center via other methods. I use reflection in Java to dynamically save and create mind map shapes with the name of their class, which allows for the type of shape to be saved and the unconvoluted extensibility of additional shape classes. A similar method that also utilizes reflection is used when importing shapes from a mind map file.

Figure #26: Usage of reflection to add a new shape (ContextMenu.java)

```
canvasPanel.getMapController().addShape("shapes.EllipseShape");
```

Figure #27: Algorithm of finding a vacant location when adding new shape (MapController.java)

```
private Point findVacantPoint(Point p) {
    for (MapShape shape : shapes) {
        Point shapeLocation = shape.getShape().getBounds().getLocation();
        shapeLocation.translate(shape.getShape().getBounds().width/2, shape.getShape().getBounds().height/2);
        if (p.equals(shapeLocation)) {
            p.translate(20, 20);
            return findVacantPoint(p);
        }
    }
    return p;
}
```

This recursive algorithm is used when adding a shape via the menu bar or picker panel to find a vacant location at the center of the canvas. If occupied, the location is offset southeastward.

Figure #28: Algorithm of bringing a shape to the front (MapController.java)

```
public void bringSelectedShapeToFront() {
    Collections.swap(shapes, shapes.indexOf(selectedShape), shapes.size()-1);
    canvasPanel.repaint();
}
```

The order in which shapes are drawn is determined by their order in the ArrayList. Since shapes have white backgrounds to conceal any lines that extend from it, I needed to use the Collections API in Java 8 to change the order of shapes. This method is handled safely if run again, as swapping one index with itself will cause nothing to occur.

Figure #29: Algorithm of bringing a shape forward (MapController.java)

```
public void bringSelectedShapeForwards() {
    // Do nothing if selected shape is already at the front
    if (! (shapes.indexOf(selectedShape) == shapes.size()-1)) {
        Collections.swap(shapes, shapes.indexOf(selectedShape), shapes.indexOf(selectedShape)+1);
        canvasPanel.repaint();
    }
}
```

The algorithms that I created for sending a shape backward and to the back have their indices inversed compared to the methods shown above.

Figure #30: Flowchart depicting process and data flow of saving a mind map

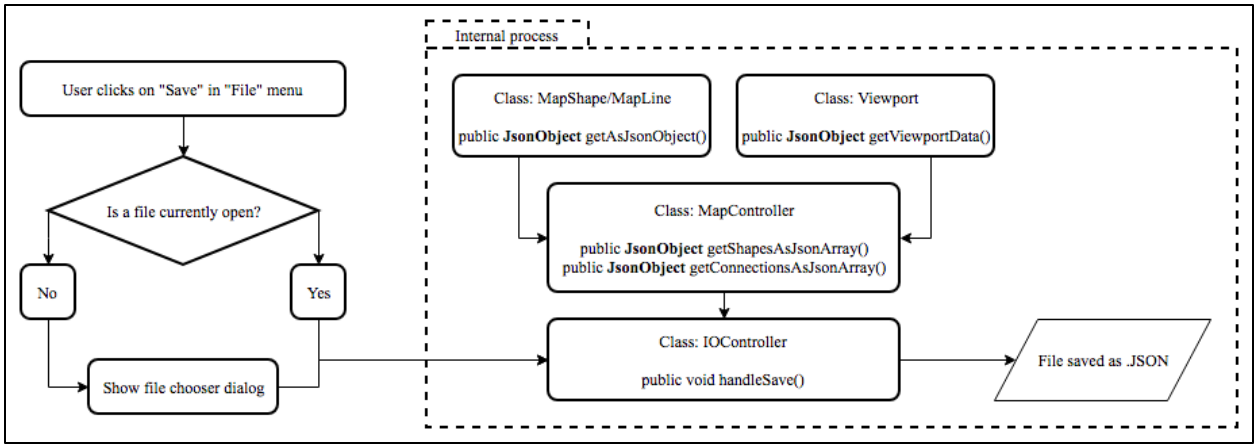



Figure #31: Save As menu item action (Menubar.java)

```
saveAs.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        fileChooser.setDialogTitle("Save Mind Map");
        fileChooser.resetChoosableFileFilters();
        fileChooser.addChoosableFileFilter(mindMapFilter);
        fileChooser.setSelectedFile(new File("Untitled.json"));

        // Change FileFilter selection label
        UIManager.put("FileChooser.filesOfTypeLabelText", "File Format:");
        SwingUtilities.updateComponentTreeUI(fileChooser);

        if (fileChooser.showSaveDialog(null) == JFileChooser.APPROVE_OPTION) {
            // Append ".json" extension if missing
            File file = fileChooser.getSelectedFile();
            if (! file.getName().toLowerCase().endsWith(".json"))
                file = new File(file.getParentFile(), file.getName() + ".json");
            appFrame.getIOCon().handleSave(file);
        }
    }
});
```

The “save” operation opens a JFileChooser. The JSON extension is added if missing. The “open” operation is similar.

Figure #32: Usage of the external Google Gson library (IOController.java)

```
private static final Gson gson = new GsonBuilder().setPrettyPrinting().create();
```

I chose to store mind map data in JSON format as it is highly structured, object-oriented, and human-readable. The external Google Gson library is used, instead of directly parsing files via a Scanner, to efficiently handle parsing of text files stored in the JSON format.

Figure #33: Handler methods for opening and saving mind map files (IOController.java)

```
public void handleOpen(File inFile) {
    System.out.print("Opening " + inFile.getAbsolutePath() + " ... ");
    try {
        JsonReader reader = new JsonReader(new FileReader(inFile));
        JsonObject data = gson.fromJson(reader, JsonObject.class);
        canvasPanel.getViewport().setViewportData(data.get("Viewport").getAsJsonObject());
        canvasPanel.getMapController().replaceShapesFromJson(data.get("Shapes").getAsJsonArray());
        canvasPanel.getMapController().replaceConnectionsFromJson(data.get("Connections").getAsJsonArray());
        canvasPanel.repaint();
        setCurrentFile(inFile);
        System.out.println("Success");
    } catch (IOException e) {
        System.out.println("File not found! " + e);
    }
}
```

```
public void handleSave(File outFile) {
    System.out.print("Saving to " + outFile.getAbsolutePath() + " ... ");
    try {
        PrintWriter p = new PrintWriter(new FileWriter(outFile));
        JsonObject output = new JsonObject();
        output.add("Viewport", gson.toJsonTree(canvasPanel.getViewport().getViewportData()));
        output.add("Shapes", gson.toJsonTree(canvasPanel.getMapController().getShapesAsJsonArray()));
        output.add("Connections", gson.toJsonTree(canvasPanel.getMapController().getConnectionsAsJsonArray()));
        p.write(gson.toJson(output));
        p.close();
        setCurrentFile(outFile);
        System.out.println("Success");
    } catch (IOException e) {
        System.out.println("File not found! " + e);
    }
}
```

The “open” operation uses Gson to get the viewport, shapes and connections data as JsonArrays. That data is converted to a JsonTree in the “save” operation. All settings of the mind map can be saved by my client.

Figure #34: Method to get MapShape attributes as a JsonObject (MapShape.java)

```
public JsonObject getAsJsonObject() {
    JsonObject thisShape = new JsonObject();
    thisShape.addProperty("ID", id.toString());
    thisShape.addProperty("Type", this.getClass().getName());
    thisShape.addProperty("X", x);
    thisShape.addProperty("Y", y);
    thisShape.addProperty("Width", shape.getBounds().width);
    thisShape.addProperty("Height", shape.getBounds().height);
    thisShape.addProperty("Border width", borderStroke.getLineWidth());
    thisShape.addProperty("Border colour", "#"+Integer.toHexString(borderColour.getRGB()).substring(2));
    thisShape.addProperty("Text", textField.getText());
    thisShape.addProperty("Text font name", textField.getFont().getName());
    thisShape.addProperty("Text font style", textField.getFont().getStyle());
    thisShape.addProperty("Text font size", textField.getFont().getSize());
    thisShape.addProperty("Font colour", "#"+Integer.toHexString(textField.getForeground().getRGB()).substring(2));
    return thisShape;
}
```

Figure #35: Method to get MapLine attributes as a JsonObject (MapLine.java)

```
public JsonObject getAsJsonObject() {
    JsonObject thisConnection = new JsonObject();
    thisConnection.addProperty("Stroke width", stroke.getLineWidth());
    thisConnection.addProperty("Origin ID", origin.getId().toString());
    thisConnection.addProperty("Termination ID", termination.getId().toString());
    return thisConnection;
}
```

Figure #36: Assigning a random UUID upon shape creation (MapShape.java)

```
    public MapShape(Shape shape) {
        setId(UUID.randomUUID());                // Generate new random UUID upon instantiation
```

Figure #37: Importance of using UUIDs in importing connections from file (MapController.java)

```
public void replaceConnectionsFromJson(JsonArray connectionsData) {
    connections = new ArrayList<MapLine>();
    for (JsonElement connection : connectionsData) {
        JsonObject thisConnection = connection.getAsJsonObject();
        MapShape origin = null, termination = null;
        for (MapShape shape : shapes) {
            if (shape.getId().equals(UUID.fromString(thisConnection.get("Origin ID").getAsString()))) {
                origin = shape;
            }
            if (shape.getId().equals(UUID.fromString(thisConnection.get("Termination ID").getAsString()))) {
                termination = shape;
            }
            if (origin != null && termination != null) break;       // Break early if connection found
        }
        if (origin != null && termination != null) connections.add(new MapLine(origin, termination));
    }
}
```

I use UUIDs to track all shapes when saving and allow for the connections between shapes to be restored. I use UUIDs instead of random numbers to greatly reduce the possibility of creating duplicate identifiers.

Figure #38: Method to get viewport data as a JsonObject (Viewport.java)

```
public JsonObject getViewportData() {
    JsonObject viewportData = new JsonObject();
    viewportData.addProperty("Zoom", zoomFactor);
    viewportData.addProperty("xOffset", xOffset);
    viewportData.addProperty("yOffset", yOffset);
    return viewportData;
}
```

Figure #39: Sample mind map file in JSON


```
{
  "Viewport": {
    "Zoom": 0.7513148009015777,
    "xOffset": 488,
    "yOffset": 94
  },
  "Shapes": [
    {
      "ID": "f53a4b04-fbab-4027-8770-a218a5726e4b",
      "Type": "shapes.EllipseShape",
      "X": 251,
      "Y": -45,
      "Width": 200,
      "Height": 100,
      "Border width": 6.0,
      "Border colour": "#ff0000",
      "Text": "Example",
      "Text font name": "Dialog",
      "Text font style": 2,
      "Text font size": 40,
      "Font colour": "#0000ff"
    },
    {
      "ID": "565373a8-5da7-4827-a63b-d434e32ed44c",
      "Type": "shapes.RectangleShape",
      "X": 420,
      "Y": 199,
      "Width": 200,
      "Height": 100,
      "Border width": 3.0,
      "Border colour": "#00ff00",
      "Text": "Example",
      "Text font name": "Times New Roman",
      "Text font style": 1,
      "Text font size": 45,
      "Font colour": "#ff0000"
    }
  ],
  "Connections": [
    {
      "Stroke width": 2.0,
      "Origin ID": "f53a4b04-fbab-4027-8770-a218a5726e4b",
      "Termination ID": "565373a8-5da7-4827-a63b-d434e32ed44c"
    }
  ]
}
```

Figure #40: File export format selection integrated into export dialog

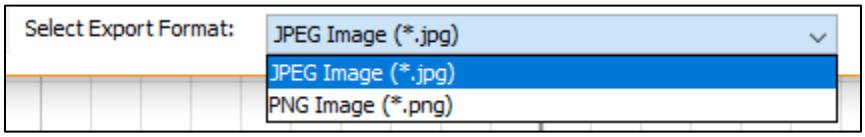



Figure #41: Method that displays a popup that prompts for the selection of export quality (Menubar.java)

```
private int showExportScalePopup() {
    // Add choosable export qualities and calculate their final dimensions
    String exportQualities[] = new String[5];
    for (int i = 1; i <= exportQualities.length; i++)
        exportQualities[i-1] = i +"x (" + appFrame.getCanvasPanel().getWidth()*i + "x" + appFrame.getCanvasPanel().getHeight()*i + ")";
    // Pop up a dialog
    String output = (String) JOptionPane.showInputDialog(fileChooser, "Image size:", "Export Options",
                                JOptionPane.PLAIN_MESSAGE, null, exportQualities, null);
    if (output != null) return Integer.parseInt(Character.toString(output.charAt(0)));
    else                return 0;
}
```

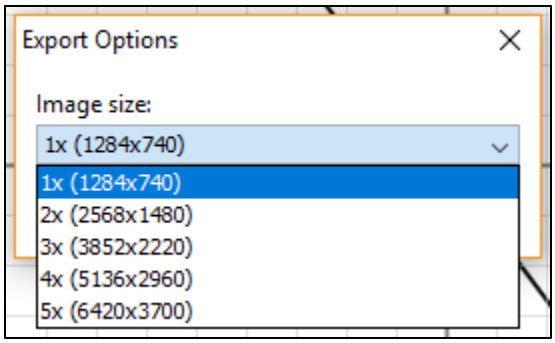



Figure #42: Handler method for exporting to an image (IOController.java)

```
public void handleExport(File file, String imgType, int scale) {
    System.out.print("Exporting to " + file.getAbsolutePath() + " ... ");
    // Upscale exported image to increase quality and enable transparency if exporting to PNG
    BufferedImage image = new BufferedImage(canvasPanel.getWidth() * scale, canvasPanel.getHeight() * scale,
                                            imgType == "png" ? BufferedImage.TYPE_INT_ARGB : BufferedImage.TYPE_INT_RGB);
    Graphics2D g2d = image.createGraphics();
    // Increase quality of exported image
    g2d.setRenderingHint(RenderingHints.KEY_ALPHA_INTERPOLATION, RenderingHints.VALUE_ALPHA_INTERPOLATION_QUALITY);
    g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
    g2d.setRenderingHint(RenderingHints.KEY_COLOR_RENDERING, RenderingHints.VALUE_COLOR_RENDER_QUALITY);
    g2d.setRenderingHint(RenderingHints.KEY_DITHERING, RenderingHints.VALUE_DITHER_ENABLE);
    g2d.setRenderingHint(RenderingHints.KEY_FRACTIONALMETRICS, RenderingHints.VALUE_FRACTIONALMETRICS_ON);
    g2d.setRenderingHint(RenderingHints.KEY_INTERPOLATION, RenderingHints.VALUE_INTERPOLATION_BILINEAR);
    g2d.setRenderingHint(RenderingHints.KEY_RENDERING, RenderingHints.VALUE_RENDER_QUALITY);
    g2d.setRenderingHint(RenderingHints.KEY_STROKE_CONTROL, RenderingHints.VALUE_STROKE_PURE);
    g2d.scale(scale, scale);        // Upscale
    canvasPanel.printAll(g2d);
    try {
        ImageIO.write(image, imgType, file);
        System.out.println("Success");
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

My client is able to export the mind map to an image with the highest rendering quality and upscale options from 1x to 5x the panel size.

Figure #43: Sample exported image

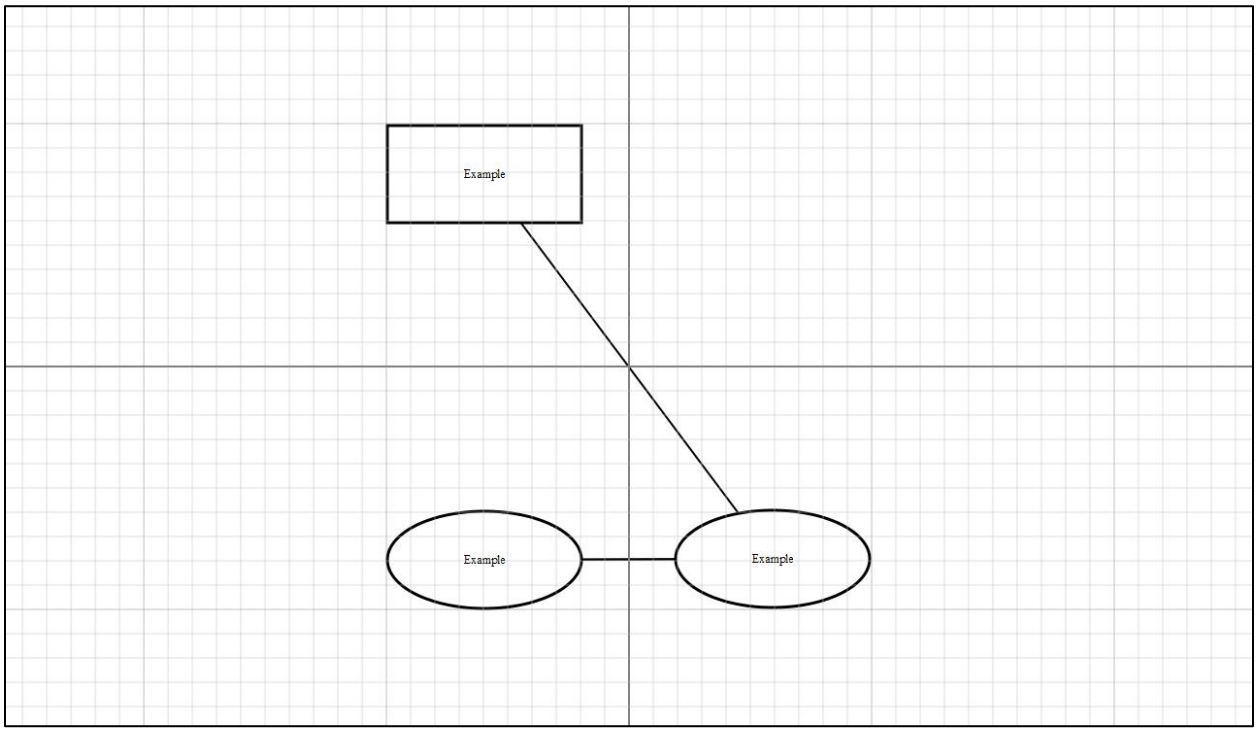



Figure #44: Method responsible for displaying right-click context menu (CanvasPanel.java)

```
// Handle displaying context menu
public void popup(MouseEvent e) {
    if (mapCon.getShapesUnderCursor(e.getPoint()).size() > 0) {
        contextMenu.updateMenuValues(mapCon.getSelectedShape());
        contextMenu.getEditMenu().setEnabled(true);
        contextMenu.getOrderMenu().setEnabled(true);
        contextMenu.getConnectionsMenu().setEnabled(true);
    } else {
        contextMenu.getEditMenu().setEnabled(false);
        contextMenu.getOrderMenu().setEnabled(false);
        contextMenu.getConnectionsMenu().setEnabled(false);
    }
    contextMenu.show(e.getComponent(), e.getX(), e.getY());
}
```

When my client right-clicks on a shape, this method enables the “Edit”, “Order” and “Connections” menus, which are only useful when editing a shape.

Figure #45: Algorithm for updating context menu values (ContextMenu.java)

```
public void updateMenuValues(MapShape selectedShape) {
    if (selectedShape != null) {
        // Update border width and font size values for the selected shape
        changeBorderWidthSlider.setValue((int)selectedShape.getBorderStroke().getLineWidth());
        changeFontSizeField.setText(Integer.toString(selectedShape.getTextField().getFont().getSize()));

        // Hide "Remove all connections" button if there are no connections
        boolean hasConnections = false;
        for (MapLine connection : canvasPanel.getMapController().getConnections()) {
            if (connection.getOrigin() == selectedShape || connection.getTermination() == selectedShape) {
                hasConnections = true;
                break;
            }
        }
        removeAllConnections.setEnabled(hasConnections ? true : false);

        if (canvasPanel.getMapController().getConnectionOrigin() == null) {
            // Allow selecting as origin
            setShapeInConnection.setEnabled(true);
            setShapeInConnection.setForeground(Color.black);
            setShapeInConnection.setText("Set as origin");
        } else {
            // Allow selecting as termination
            if (canvasPanel.getMapController().getSelectedShape().equals(
                        canvasPanel.getMapController().getConnectionOrigin())) {
                setShapeInConnection.setEnabled(false);
                setShapeInConnection.setForeground(Color.red);
                setShapeInConnection.setText("Cannot set selected shape again");
            } else {
                setShapeInConnection.setEnabled(true);
                setShapeInConnection.setForeground(Color.black);
                setShapeInConnection.setText("Set as termination");
            }
        }
    }
}
```

This algorithm dynamically updates values in the “Edit” menu to reflect the selected shape and changes menu text when creating connections.

Figure #46: Menu item that removes all connections from the selected shape (ContextMenu.java)

```
removeAllConnections = new JMenuItem("Remove all connections");
removeAllConnections.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        canvasPanel.getMapController().removeConnectionsFromSelectedShape();
    }
});
```

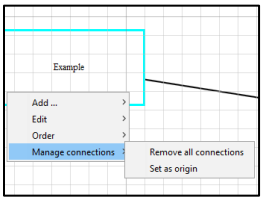


Figure #47: Algorithm responsible for removing connections from a shape (MapController.java)

```
public void removeConnectionsFromSelectedShape() {
    origin = null;
    List<MapLine> toBeRemoved = new ArrayList<MapLine>();
    for (MapLine connection : connections) {
        if (connection.getOrigin() == selectedShape || connection.getTermination() == selectedShape) {
            toBeRemoved.add(connection);
        }
    }
    connections.removeAll(toBeRemoved);
    canvasPanel.repaint();
}
```

This algorithm iterates through all connections and removes them if they involve the selected shape.

Figure #48: Hash table containing colour names and their classes (ContextMenu.java)

```
// Lookup tables for values and their names
private static final HashMap<String,Color> colours = new HashMap<String,Color>();
static {
    colours.put("Black", Color.black);
    colours.put("Red", Color.red);
    colours.put("Green", Color.green);
    colours.put("Blue", Color.blue);
    colours.put("Yellow", Color.yellow);
    colours.put("Orange", Color.orange);
    colours.put("Magenta", Color.magenta);
    colours.put("Pink", Color.pink);
}
```

I use hash tables to store key and value pairs efficiently, for example, colour names with their Color classes, font names with their equivalents, font styles with their values.

Figure #49: Usage of the colours hash table (ContextMenu.java)

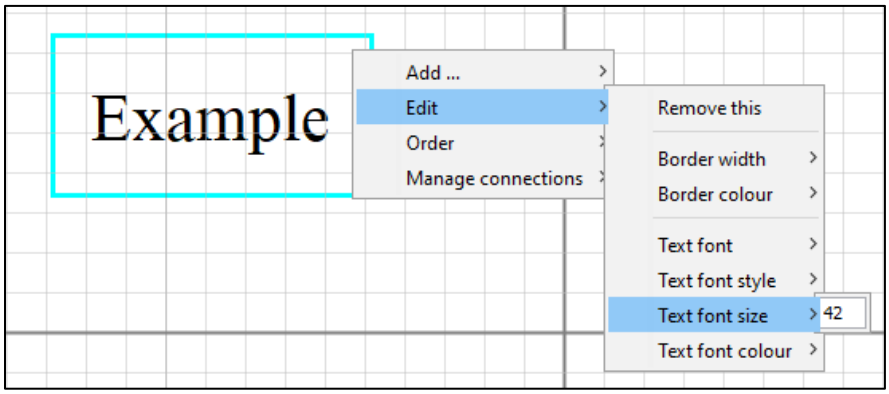
```
// Iterate through available colours and create a new menu item for each
for (String colourName : colours.keySet()) {
    JMenuItem selectBorderColour = new JMenuItem(colourName);
    selectBorderColour.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            canvasPanel.getMapController().changeBorderColour(colours.get(colourName));
            canvasPanel.getMapController().setSelectedShape(null);
        }
    });
    changeBorderColourMenu.add(selectBorderColour);
}
```

Figure #50: Listener for font size text field changes (ContextMenu.java)

```
// Create a text field for custom font sizes
changeFontSizeField = new JTextField();
changeFontSizeField.setColumns(3);
changeFontSizeField.getDocument().addDocumentListener(new DocumentListener() {
    public void insertUpdate(DocumentEvent e) {
        changeFontSize();
    }
    public void removeUpdate(DocumentEvent e) {
        changeFontSize();
    }
    public void changedUpdate(DocumentEvent e) {
    }
    private void changeFontSize() {
        // Parse as int if the text field only contains numbers
        if (changeFontSizeField.getText().matches("^\\d+$")) {
            canvasPanel.getMapController()
                                .changeFontSize(Integer.parseInt(changeFontSizeField.getText()));
        }
    }
});
```

This listener only allows numbers to be entered in the font size text field.

Figure #51: Font size text field location in context menu (ContextMenu.java)




Word count: 1099 (excluding headings, captions, tables, screenshots and flowcharts)

## Bibliography


Clayton, R. (2015, January 11). *Do you really need a UUID/GUID?* Retrieved from https://rclayton.silvrback.com/do-you-really-need-a-uuid-guid

Cook, C. E. (2005). *Blue Pelican Java.* College Station: Virtualbookworm.com.

Google. (n.d.). *google/gson*. Retrieved from GitHub: https://github.com/google/gson

gskinner. (n.d.). Retrieved from RegExr: https://regexr.com/

Henrichsen, C. (2016, April 25). *Drawing Shapes In Java*. Retrieved from YouTube: https://www.youtube.com/watch?v=YsN9g14Rlw8

Horstmann, C. (2010). *Big Java: compatible with Java 5, 6 and 7.* Hoboken: Wiley.

Marilena. (2016, November 29). *Java Swing – JFileChooser example*. Retrieved from Mkyong.com: https://www.mkyong.com/swing/java-swing-jfilechooser-example/

Oracle. (n.d.). *Controlling Rendering Quality*. Retrieved from Java Documentation: https://docs.oracle.com/javase/tutorial/2d/advanced/quality.html

Oracle. (n.d.). *How to Make Dialogs*. Retrieved from Java Documentation: https://docs.oracle.com/javase/tutorial/uiswing/components/dialog.html

Oracle. (n.d.). *How to Use Borders*. Retrieved from Java Documentation: https://docs.oracle.com/javase/tutorial/uiswing/components/border.html

Oracle. (n.d.). *How to Use Sliders*. Retrieved from Java Documentation: https://docs.oracle.com/javase/tutorial/uiswing/components/slider.html

Oracle. (n.d.). *How to Write a Component Listener* . Retrieved from Java Documentation: https://docs.oracle.com/javase/tutorial/uiswing/events/componentlistener.html

Oracle. (n.d.). *Line2D (Java Platform SE 7)*. Retrieved from Java Documentation: https://docs.oracle.com/javase/7/docs/api/java/awt/geom/Line2D.html

Oracle. (n.d.). *MouseAdapter (Java Platform SE 7)*. Retrieved from Java Documentation: https://docs.oracle.com/javase/7/docs/api/java/awt/event/MouseAdapter.html

Oracle. (n.d.). *Path2D.Double (Java Platform SE 7)*. Retrieved from Java Documentation: https://docs.oracle.com/javase/7/docs/api/java/awt/geom/Path2D.Double.html

Oracle. (n.d.). *Trail: 2D Graphics*. Retrieved from Java Documentation: https://docs.oracle.com/javase/tutorial/2d/index.html

Quillion. (2014, March 20). *java.awt.EventQueue.invokeLater explained*. Retrieved from Stack Overflow: https://stackoverflow.com/questions/22534356/java-awt-eventqueue-invokelater-explained