Software Development: a JavaFX Paint application

Author: Szymon Hładyszewski

Group ID: 1

Student ID: 279772

Executive Summary

This report explores software development through the creation of a JavaFX-based Paint application. It enables users to draw geometric shapes, modify them, and save their work as a document. Development began with designing a graphical user interface (GUI) and progressed to implementing tools for drawing and editing shapes. The methodology relied on lecturer-provided materials and ChatGPT assistance, although some challenges were encountered.

A key focus of this project is the comparison of JavaFX with older frameworks such as Swing and AWT. The report highlights how modern ones simplify application development and solve the problems faced in earlier approaches. Furthermore, the project identifies areas for improvement in modern frameworks to better address common issues in similar projects.

Introduction

Graphical user interface tools have long been a great asset to software engineering. In the case of the Java programming language, these include Swing, AWT, and JavaFX. They have been a breakthrough in creating projects focusing on a user-friendly view.

This report shows the progress of software development in Java through the creation of the Paint application. It examines the older frameworks, the Swing and AWT, and the modern JavaFX to assess their impact on code structure, performance, and ease of implementation. The results provide practical insights for developers that can help them choose the right framework for future projects. Additionally, the entire Paint application is an example of the

challenges faced in the past when many event cases required manual implementation of many dependencies.

Methods and Materials

This section outlines the tools, resources, and methodologies used to develop the Paint application, which offers functionalities such as drawing geometric shapes, customizing them, and dynamically changing shape colors.

Tools Used:

- VS Code: Integrated development environment used for coding and debugging.
- Javac Compiler: Ensured seamless compilation and execution of the Java codebase.
- JavaFX Library: Provided classes for GUI creation, event handling, and graphical rendering.
- Documentation: JavaFX API and lecturer-provided materials were the primary resources for learning and troubleshooting.
- ChatGPT Assistance: Leveraged for brainstorming design solutions and addressing challenges during development.

GUI Design:

The graphical user interface includes a toolbar at the top of the window featuring options to clear all shapes from the drawing area, save the drawing area as a PNG file, open an existing one, and create a new document.

Below the toolbar, three buttons were implemented to allow drawing specific shapes: rectangles, triangles, and circles. Clicking and dragging with the left mouse button enables users to create the selected shape. Initially, shapes appeared on the canvas only after releasing the mouse button.

Functionalities implemented:

One of the key features introduced is the ability to create shapes dynamically. During the drawing process, shapes "follow" the cursor, allowing users to visualize their placement in real-time. Once the shape is finalized by releasing the mouse button, it remains fixed at the desired location.

Shape activation is another functionality, enabling users to interact with shapes after they had been drawn. Activated shapes are visually distinguished by a thicker border, providing clear feedback on which shape was selected. This mechanism facilitates editing and ensures that users can easily identify and manipulate specific shapes.

To enhance interactivity, the following features were added:

- Rotation: Activated shapes can be rotated by dragging the mouse while holding a specific key.
- Movement: Shapes can be repositioned by dragging them across the canvas.
- Color Filling: Right-clicking inside a shape opens a context menu with a list of available colors. Users can select a color by left-clicking on it, dynamically updating the shape's fill color.
- Resizing: Activated shapes can be resized using the mouse scroll wheel. Scrolling up enlarged the shape, while scrolling down reduced its size.

Findings

The development of the Paint application was efficient due to JavaFX's advanced event-handling capabilities, particularly for mouse interactions. Features like real-time shape rendering, activation, resizing, and movement were seamlessly implemented, highlighting JavaFX's superiority over legacy frameworks like Swing and AWT.

During development, however, some challenges arose that required creative problem-solving. One of the most significant issues was implementing rotation for activated shapes. Initially, shapes rotated around their top-left corner, which led to an unnatural movement. To

address this, the rotation logic was adjusted to calculate the center point of each shape. This modification provided a more intuitive and visually appealing user interaction.

Another problem involved shapes obstructing the toolbar buttons. When shapes were drawn near or over it, they could accidentally cover the buttons, making them inaccessible. To solve this issue, the toolbar was positioned above the drawing area in the application hierarchy. This adjustment ensured that shapes extending beyond the canvas would be clipped, preventing any interference with its functionality.

These solutions not only enhanced the functionality and usability of the application but also demonstrated the flexibility of JavaFX in addressing complex development challenges. Overall, the capabilities of this framework made the entire project development much more streamlined.

Discussion

Based on the results obtained, further conclusions can be drawn regarding older frameworks. JavaFX simplifies the implementation of dynamically creating objects on the canvas. This is because Swing and AWT lack automatic gesture recognition, such as detecting mouse movements and button presses and releases. As a result, each event must be manually defined, with the appropriate action assigned afterward.

Additionally, in JavaFX, the entire drawing area is created by defining a single, simple object called GraphicsContext. In contrast, Swing and AWT require a more complex approach, as the paintComponent(Graphics g) method in the JPanel class must be overridden. This process necessitates a solid understanding of low-level graphics operations.

Furthermore, drawing shapes in AWT is considerably more complex. Each shape is drawn using a function applied to the Graphics g object, which poses challenges when modifying a specific shape. In contrast, JavaFX represents each of them as a distinct object, making it significantly easier to implement and change the activation of the desired shape.

Conclusions

In conclusion, developing Paint applications using JavaFX provides a clear perspective on the evolution of software development over time. While earlier frameworks already supported the creation of complex projects, modern frameworks prioritized the ease of implementing intricate features. Defining the drawing area and generating shapes has become relatively straightforward, although certain aspects still present opportunities for further refinement.

Looking ahead, more advanced frameworks may incorporate built-in methods for managing keyboard shortcuts, such as Ctrl+Z to undo the most recent operation. This would allow developers to focus less on implementing basic commands that can be time-consuming to code.

Furthermore, saving and opening drawings from files in various formats can be significantly optimized. Due to the complex handling of each file format, applications like Paint are often restricted to simpler formats such as PNG. Streamlining this process would not only save valuable development time but also expand the range of supported formats, potentially including JPG or PDF.

Another area for improvement is image rendering performance. As the number of shapes on the canvas increases, the program may experience temporary freezes, hindering the creation of more complex drawings. This issue stems not from inefficient coding practices, but from the limitations of the current framework, which may not be fully optimized for such tasks.

Overall, this report demonstrates how the evolution of frameworks such as JavaFX has streamlined the development of user-friendly applications. While challenges remain to be addressed, the pace and quality of advancement suggest optimistic prospects for the near future. The development of complex applications is expected to become increasingly straightforward, with developers requiring less time to complete projects, enhancing overall efficiency.