Note-Taking Application

A PROJECT REPORT

Submitted by

PRASHANTH REDDY C

NOVEMBER - 2024

ABSTRACT

The Note-Taking Application is a GUI-based software that enables users to create, edit, save, delete, and organize textual notes. Built using Java Swing, this application provides a clean and user-friendly interface, ensuring an optimal note-taking experience. Notes are saved as .txt files in a designated directory, making them accessible and easily manageable. The application offers features such as real-time text editing, organized note viewing through a list, and robust error handling to ensure a seamless user experience. It demonstrates practical use of Java's GUI capabilities and emphasizes simplicity, functionality, and maintainability.

INTRODUCTION

Note-taking is an essential activity for organizing thoughts, recording ideas, and storing information in various domains such as education, work, and personal management. The Note-Taking Application is designed to meet these needs by offering a digital solution for creating and managing text-based notes.

By leveraging Java Swing, the application provides an interactive user interface with features that enable users to perform essential note-taking tasks effortlessly. The system ensures data persistence through the file system, which allows users to save and retrieve their notes at any time. This project showcases Java's ability to develop GUI-based desktop applications with practical real-world applications.

OBJECTIVE

The primary objectives of this project are:

- To develop a responsive and user-friendly note-taking application.
- To provide features for creating, editing, saving, organizing, and deleting notes.
- To ensure notes are securely stored as .txt files for easy retrieval.
- To implement a robust error-handling mechanism to enhance usability.

LITERATURE REVIEW

Java Swing is one of the most popular libraries for developing desktop applications due to its ease of use and versatility. Studies on GUI-based applications highlight the importance of user experience, including simplicity in design, responsiveness, and reliability. While existing note-taking software, such as Microsoft OneNote or Evernote, provides advanced features like cloud synchronization, this project focuses on creating a lightweight, locally-stored, and beginner-friendly solution.

Research into similar applications suggests that basic functionalities like file saving, organized display of notes, and a clean interface significantly enhance user adoption. This project builds upon these principles to deliver an efficient and practical tool for note-taking.

SYSTEM DESIGN AND IMPLEMENTATION

Architecture

The application is divided into three main components:

1. Frontend (User Interface): Created using Java Swing components such as JFrame, JList, and JTextArea.
2. Backend (Logic): Manages note creation, saving, loading, and deletion.
3. File Storage: Notes are stored as .txt files in a dedicated directory named "Notes" for persistence.

Components

1. Notes List:
    - Displays the titles of saved notes.
    - Allows users to select a note for editing or deletion.
2. Note Editor:
    - A text area (JTextArea) for writing and editing notes.
    - Integrated with save functionality to persist changes.
3. Buttons:
    - New Note: Clears the editor for creating a new note.
    - Save Note: Saves the note content to a .txt file.
    - Delete Note: Deletes the selected note permanently.

User Interface (UI) Design

The user interface is minimalistic and functional:

- The application window is divided into two sections: a list of notes on the left and a note editor on the right.
- The top panel includes buttons for creating, saving, and deleting notes.
- The UI components are styled for simplicity, ensuring that users can focus on their tasks without distractions.

CODE

NoteTakingApp.java

```
import javax.swing.*;
public class NoteTakingApp {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(NoteAppFrame::new);
    }
}
```

NoteAppFrame.java

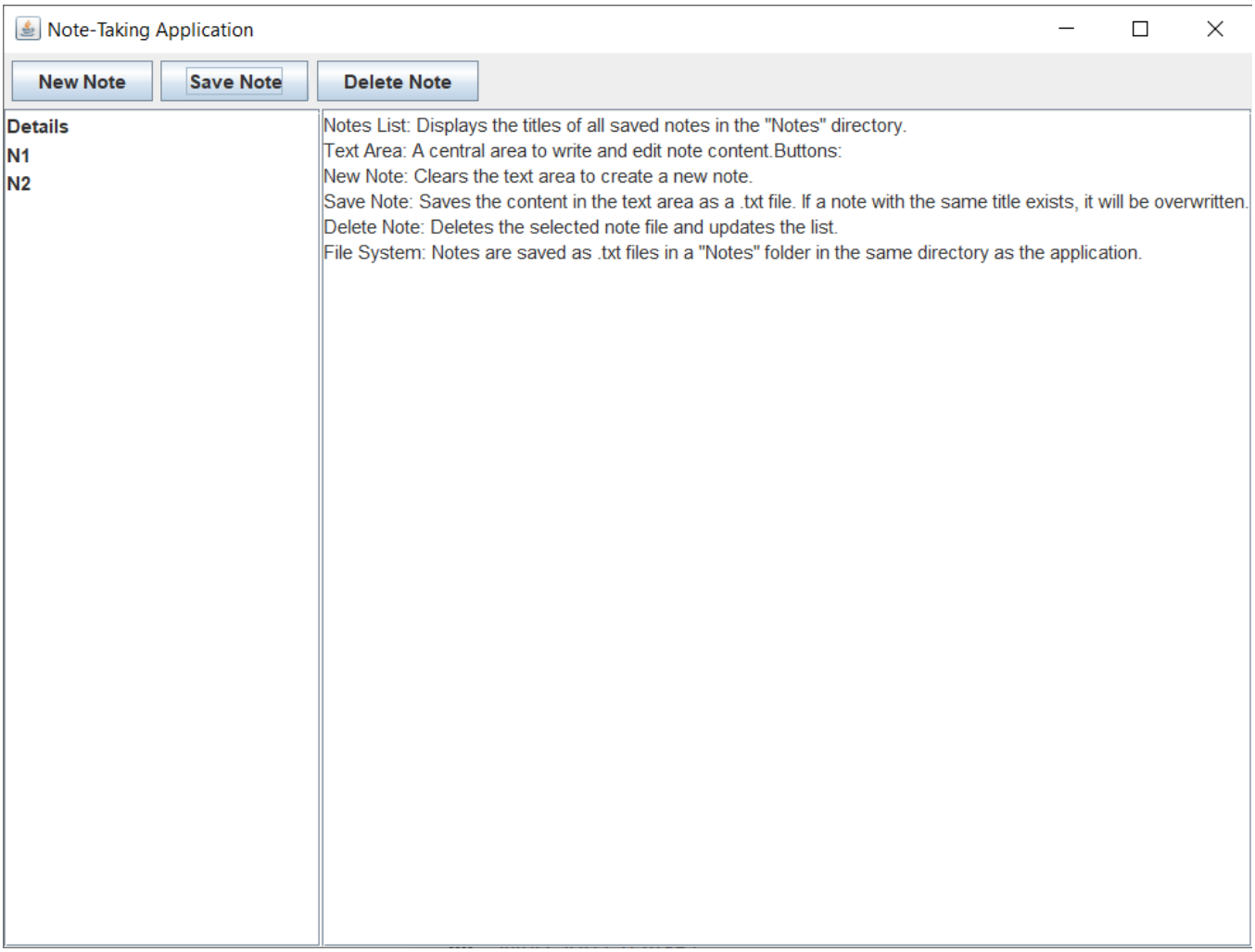
```
import javax.swing.*;
import java.awt.*;
import java.io.*;
import java.util.ArrayList;

class NoteAppFrame extends JFrame {
    private final JTextArea noteArea;
    private final DefaultListModel<String> notesListModel;
    private final JList<String> notesList;
    private final ArrayList<File> notesFiles;
    private final File notesDirectory;
    public NoteAppFrame() {
        // Configure Frame
        setTitle("Note-Taking Application");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(800, 600);
        setLayout(new BorderLayout());
        // Notes Storage Directory
        notesDirectory = new File("Notes");
        if (!notesDirectory.exists()) {
            notesDirectory.mkdir();
        }
        // Notes List Panel
        notesListModel = new DefaultListModel<>();
        notesList = new JList<>(notesListModel);
        notesList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        notesFiles = new ArrayList<>();
        loadNotes();
        notesList.addListSelectionListener(e -> {
            if (!e.getValueIsAdjusting()) {
                loadSelectedNote();
            }
        });
```

```
        JScrollPane notesScrollPane = new JScrollPane(notesList);
        notesScrollPane.setPreferredSize(new Dimension(200, getHeight()));
        add(notesScrollPane, BorderLayout.WEST);

        // Note Area
        noteArea = new JTextArea();
        JScrollPane noteScrollPane = new JScrollPane(noteArea);
        add(noteScrollPane, BorderLayout.CENTER);
        // Buttons Panel
        JPanel buttonsPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
        add(buttonsPanel, BorderLayout.NORTH);
        JButton newButton = new JButton("New Note");
        newButton.addActionListener(e -> createNewNote());
        buttonsPanel.add(newButton);

        JButton saveButton = new JButton("Save Note");
        saveButton.addActionListener(e -> saveCurrentNote());
        buttonsPanel.add(saveButton);

        JButton deleteButton = new JButton("Delete Note");
        deleteButton.addActionListener(e -> deleteSelectedNote());
        buttonsPanel.add(deleteButton);

        setVisible(true);
    }

    private void loadNotes() {
        notesListModel.clear();
        notesFiles.clear();
        for (File file : notesDirectory.listFiles((dir, name) -> name.endsWith(".txt"))) {
            notesFiles.add(file);
            notesListModel.addElement(file.getName().replace(".txt", ""));
        }
    }

    private void loadSelectedNote() {
        int index = notesList.getSelectedIndex();
        if (index != -1) {
            File selectedFile = notesFiles.get(index);
            try (BufferedReader reader = new BufferedReader(new FileReader(selectedFile))) {
                noteArea.setText("");
                String line;
                while ((line = reader.readLine()) != null) {
                    noteArea.append(line + "\n");
                }
            } catch (IOException e) {
```

```
            JOptionPane.showMessageDialog(this, "Error loading note: " + e.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
        }
    }
}

private void createNewNote() {
    noteArea.setText("");
    notesList.clearSelection();
}

private void saveCurrentNote() {
    String noteTitle = JOptionPane.showInputDialog(this, "Enter Note Title:");
    if (noteTitle == null || noteTitle.isEmpty()) {
        JOptionPane.showMessageDialog(this, "Note title cannot be empty.", "Error",
JOptionPane.ERROR_MESSAGE);
        return;
    }

    File noteFile = new File(notesDirectory, noteTitle + ".txt");
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(noteFile))) {
        writer.write(noteArea.getText());
        loadNotes();
    } catch (IOException e) {
        JOptionPane.showMessageDialog(this, "Error saving note: " + e.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
    }
}

private void deleteSelectedNote() {
    int index = notesList.getSelectedIndex();
    if (index != -1) {
        File selectedFile = notesFiles.get(index);
        if (selectedFile.delete()) {
            loadNotes();
            noteArea.setText("");
        } else {
            JOptionPane.showMessageDialog(this, "Error deleting note.", "Error",
JOptionPane.ERROR_MESSAGE);
        }
    } else {
        JOptionPane.showMessageDialog(this, "No note selected.", "Error",
JOptionPane.ERROR_MESSAGE);
    }
}
}
```

RESULTS

Screenshots:



CONCLUSION

The Note-Taking Application successfully demonstrates the development of a simple and functional note management tool. By combining Java Swing for the user interface and file I/O for storage, this project provides a robust solution for personal note-taking needs.

REFERENCES

[1] Oracle Documentation on Java.

[2] Effective Java, 3rd Edition by Joshua Bloch

[3] Java: The Complete Reference by Herbert Schildt