BUDGET CALCULATOR

A PROJECT REPORT

Submitted by

HARINI S - (2303811710422055)

in partial fulfillment of requirements for the award of the course

CGB1201 - JAVA PROGRAMMING

In

COMPUTER SCIENCE AND ENGINEERING

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(An Autonomous Institution, affiliated to Anna University Chennai and Approved by AICTE, New Delhi)

SAMAYAPURAM – 621 112.

NOVEMBER- 2024

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY (AUTONOMOUS)

SAMAYAPURAM – 621 112

BONAFIDE CERTIFICATE

Certified that this project report on **"BUDGET CALCULATOR"** is the bonafide work of **HARINI S (2303811710422055)** who carried out the project work during the academic year 2024 - 2025 under my supervision.

SIGNATURE

Dr.A.Delphin Carolina Rani, M.E.,Ph.D.,

HEAD OF THE DEPARTMENT

PROFESSOR

Department of CSE

K.Ramakrishnan College of Technology (Autonomous)

Samayapuram–621112.

SIGNATURE

Mr. M. Saravanan, M.E.,

SUPERVISOR

ASSISTANT PROFESSOR

Department of CSE

K.Ramakrishnan College of Technology (Autonomous)

Samayapuram–621112.

Submitted for the viva-voce examination held on 02.12.2024

INTERNAL EXAMINER

EXTERNAL EXAMINER

DECLARATION

I declare that the project report on **"BUDGET CALCULATOR"** is the result of original work done by us and best of our knowledge, similar work has not been submitted to **"ANNA UNIVERSITY CHENNAI"** for the requirement of Degree of **BACHELOR OF ENGINEERING**. This project report is submitted on the partial fulfilment of the requirement of the completion of the course **CGB1201 - JAVA PROGRAMMING.**

Signature

HARINI S

Place: Samayapuram

Date: 02.12.2024

ACKNOWLEDGEMENT

It is with great pride that I express our gratitude and in-debt to our institution "**K. Ramakrishnan College of Technology (Autonomous)**", for providing us with the opportunity to do this project.

I am glad to credit honourable chairman **Dr. K. RAMAKRISHNAN**, **B.E.,** for having provided for the facilities during the course of our study in college.

I would like to express our sincere thanks to our beloved Executive Director **Dr. S. KUPPUSAMY, MBA, Ph.D.,** for forwarding to our project and offering adequate duration in completing our project.

I would like to thank **Dr. N. VASUDEVAN, M.Tech., Ph.D.,** Principal, who gave me the opportunity to frame the project with full satisfaction.

I wholeheartedly thank **Dr. A. DELPHIN CAROLINA RANI, M.E.,Ph.D.,** Head of the department, **COMPUTER SCIENCE AND ENGINEERING** for providing her encouragement in pursuing this project.

I express our deep expression and sincere gratitude to our project supervisor **Mr. M. SARAVANAN, M.E.,** Department of **COMPUTER SCIENCE AND ENGINEERING,** for his incalculable suggestions, creativity, assistance and patience which motivated us to carry out this project.

I render our sincere thanks to the Course Coordinator and other staff members for providing valuable information during the course.

I wish to express our special thanks to the officials and Lab Technicians of our departments who rendered their help during the period of the work progress.

VISION OF THE INSTITUTION

To serve the society by offering top-notch technical education on par with global standards.

MISSION OF THE INSTITUTION

- Be a center of excellence for technical education in emerging technologies by exceeding the needs of the industry and society.
- Be an institute with world class research facilities.
- Be an institute nurturing talent and enhancing the competency of students to transform them as all-round personality respecting moral and ethical values.

VISION OF DEPARTMENT

To be a center of eminence in creating competent software professionals with research and innovative skills.

MISSION OF DEPARTMENT

M1: Industry Specific: To nurture students in working with various hardware and software platforms inclined with the best practices of industry.

M2: Research: To prepare students for research-oriented activities.

M3: Society: To empower students with the required skills to solve complex technological problems of society.

PROGRAM EDUCATIONAL OBJECTIVES

PEO 1: Domain Knowledge

To produce graduates who have a strong foundation of knowledge and skills in the field of Computer Science and Engineering.

PEO 2: Employability Skills and Research

To produce graduates who are employable in industries/public sector/research organizations or work as an entrepreneur.

PEO 3: Ethics and Values

To develop leadership skills and ethically collaborate with society to tackle real-world challenges.

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO 1: Domain Knowledge

To analyze, design and develop computing solutions by applying foundational concepts of Computer Science and Engineering.

PSO 2: Quality Software

To apply software engineering principles and practices for developing quality software for scientific and business applications.

PSO 3: Innovation Ideas

To adapt to emerging Information and Communication Technologies (ICT) to innovate ideas and solutions to existing/novel problems.

PROGRAM OUTCOMES (POs)

Engineering students will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations

6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

ABSTRACT

The Budget Calculator is a comprehensive Java-based application developed to assist users in effectively managing their finances by organizing, tracking, and summarizing expenses. With a modular architecture, the application emphasizes maintainability and scalability, ensuring each feature is logically compartmentalized. The user-friendly interface, built using Java's AWT and Swing libraries, provides an intuitive environment where users can interact with budget data. Core functionalities include adding new expenses, editing existing entries, deleting unnecessary items, and summarizing total expenditures, all of which are accessible through a clean and responsive graphical interface. A JTable component, backed by DefaultTableModel, dynamically updates to reflect user actions in real time, ensuring accurate representation of the budget.

To handle user interactions seamlessly, the application employs event-driven programming with ActionListeners for button clicks and WindowAdapters for window events, enabling smooth transitions between operations. Input validation and error-handling mechanisms play a critical role in ensuring data integrity, preventing invalid inputs such as non-numeric values, and guiding users with informative error messages displayed via JOptionPane. The application also incorporates a summarization feature that calculates and displays total expenses, empowering users to gain insights into their spending patterns.

ABSTRACT WITH POs AND PSOs MAPPING

CO 5 : BUILD JAVA APPLICATIONS FOR SOLVING REAL-TIME PROBLEMS.

ABSTRACT	POs MAPPED	PSOs MAPPED
The Budget Calculator is a comprehensive Java-based application developed to assist users in effectively managing their finances by organizing, tracking, and summarizing expenses. With a modular architecture, the application emphasizes maintainability and scalability, ensuring each feature is logically compartmentalized. The user-friendly interface, built using Java's AWT and Swing libraries, provides an intuitive environment where users can interact with budget data. Core functionalities include adding new expenses, editing existing entries, deleting unnecessary items, and summarizing total expenditures, all of which are accessible through a clean and responsive graphical interface. A JTable component, backed by DefaultTableModel, dynamically updates to reflect user actions in real time, ensuring accurate representation of the budget. To handle user interactions seamlessly, the application employs event-driven programming with ActionListeners for button clicks and WindowAdapters for window events, enabling smooth transitions between operations. Input validation and error-handling mechanisms play a critical role in ensuring data integrity, preventing invalid inputs such as non-numeric values, and guiding users with informative error messages displayed via JOptionPane. The application also incorporates a summarization feature that calculates and displays total expenses, empowering users to gain insights into their spending patterns.	**PO 1 -3** **PO 2 -3** **PO 3 -3** **PO 4 -3** **PO 5 -3** **PO 6 -3** **PO 7 -3** **PO 8 -3** **PO 9 -3** **PO 10 -3** **PO 11-3** **PO 12 -3**	**PSO 1 -3** **PSO 2 -3** **PSO 3 -3**

Note: 1- Low, 2-Medium, 3- High

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Objective

The objective of the Budget Calculator application is to provide users with a simple and interactive tool for managing their finances. It enables users to record and track various expense items along with their corresponding amounts through a user-friendly graphical interface. The application offers functionalities to add new expense items, edit existing entries to update or correct details, and delete items that are no longer relevant. Additionally, it allows users to quickly calculate and display the total expenditure based on the listed items, helping them to monitor their spending effectively. Overall, this tool serves as a convenient way for individuals or small organizations to organize and manage their budget efficiently.

1.2 Overview

The Budget Calculator is a Java-based GUI application designed to help users manage their expenses effectively. It leverages AWT and Swing components to provide a user-friendly interface where users can add, edit, and delete expense items. Each item consists of a name and an associated amount, which are displayed in a tabular format. The application also includes a feature to summarize the total expenses, giving users a quick overview of their financial status. The use of dialog boxes ensures interactive input and validation, while the structured layout with buttons for each action makes it easy to navigate. This tool is ideal for personal or small-scale budget management, offering simplicity and practicality in tracking and controlling expenditures.

1.3 Java Programming Concepts

It demonstrates several core Java programming concepts, particularly in GUI development and event-driven programming. It utilizes the Abstract Window Toolkit (AWT) and Swing libraries to create a graphical user interface, including components like Frame, JTable, JScrollPane, and JOptionPane. The concept of inheritance is evident as the class extends Frame, leveraging its properties and methods. The program employs event handling through ActionListener and WindowAdapter to respond to user interactions, such as button clicks and window closing events. Data management is handled through DefaultTableModel, which dynamically stores and updates budget items in a tabular format. These concepts work together to create an interactive, responsive, and user-friendly budget management.

CHAPTER 2
PROJECT METHODOLOGY

2.1 Proposed Work

The proposed work for the Budget Calculator application involves enhancing its functionality and usability to make it a more comprehensive and efficient tool for managing expenses. The primary focus is on improving the user interface by replacing AWT components with Swing equivalents like JFrame and JButton for a modern look and feel. Additionally, advanced validation mechanisms can be introduced to ensure accurate data entry, such as restricting non-numeric inputs for amounts and preventing empty item names. To enhance usability, features like keyboard shortcuts, the ability to clear all entries, and automatic data saving to a file could be implemented. Furthermore, adding a feature to generate and export summary reports in formats like PDF or CSV would make the application more versatile. These enhancements aim to provide users with a seamless and efficient experience in managing their budget.

2.2 Block Diagram

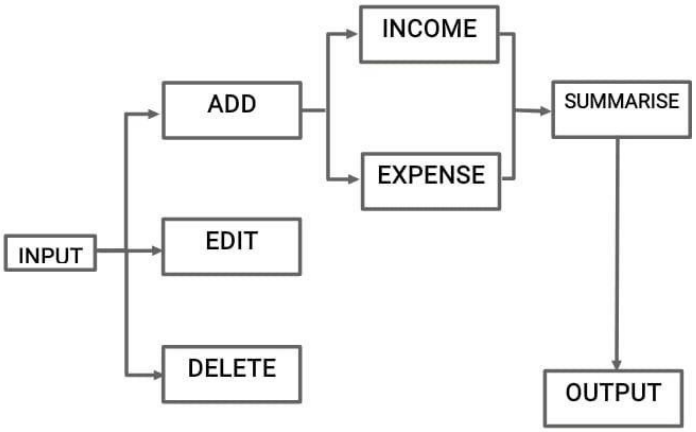


Fig 2.2 Block Diagram

CHAPTER 3
MODULE DESCRIPTION

3.1 User Interface (UI) Module

The UI module is the visual component of the application, built using Java's AWT and Swing libraries. It features a main window (JFrame) that houses a table (JTable) to display expenses and a set of buttons for actions like adding, editing, deleting, and summarizing items. The layout is organized using managers like FlowLayout or GridLayout for optimal arrangement. Dialogs and messages are handled via JOptionPane, which prompts users for inputs or displays errors, ensuring an intuitive interface for managing budget items.

3.2 Table Management Module

This module oversees the dynamic management of the expense data displayed in the JTable. Utilizing DefaultTableModel, it supports adding, editing, and deleting rows. New items are appended, existing rows are updated, and selected rows are removed with synchronized updates to the UI. The table remains responsive to user actions, ensuring accurate and up-to-date data presentation.

3.3 Action Handling Module

The Action Handling module links user interactions with functionality. Buttons are equipped with ActionListeners to trigger methods such as addItem(), editItem(), deleteItem(), and summariseItems(). These methods manage data entry, modification, and calculations, ensuring seamless interaction between the interface and the application's logic. For instance, clicking "Summarize" calculates and displays the total expenses in real-time.

3.4 Event Handling Module

This module manages user-triggered and system-generated events, using Java's event-driven programming model. Button clicks and window actions are captured by ActionListeners and WindowAdapters, respectively. It ensures the application responds appropriately to interactions, such as processing button clicks or gracefully handling window closure events, maintaining smooth functionality.

3.5 Validation and Error Handling Module

The validation module ensures data integrity by verifying user inputs, such as checking if entered amounts are numeric and non-negative. Exceptions are caught and handled gracefully, with informative error messages displayed through JOptionPane. This approach prevents invalid data from disrupting operations while guiding users to correct errors effectively.

3.6 Exit and Cleanup Module

This module ensures the application terminates smoothly, releasing resources and handling exit events. A WindowAdapter listens for the window's close action and invokes System.exit(0) for proper termination. Optionally, a confirmation dialog can prompt users before exiting. This module ensures that all resources are cleaned up, preventing potential memory leaks or errors during closure.

CHAPTER 4
CONCLUSION & FUTURE SCOPE

4.1 CONCLUSION

In conclusion, the Budget Calculator application serves as a practical and user-friendly tool for managing personal or organizational expenses. By leveraging Java's AWT and Swing libraries, it provides a straightforward interface for users to add, edit, delete, and summarize budget items efficiently. The application demonstrates essential programming concepts such as event-driven programming, exception handling, and dynamic data management through DefaultTableModel. While the current implementation offers basic functionalities, it lays a solid foundation for further enhancements, such as improved UI design, data persistence, and advanced reporting features. Overall, the project showcases the potential of Java in building interactive and effective financial management tools.

4.2 FUTURE SCOPE

The Budget Calculator application has significant potential for future development to enhance its utility and user experience. One key area of improvement is the integration of data persistence mechanisms, such as saving and loading budget data from files or databases, enabling users to maintain records over time. Additionally, incorporating advanced reporting features, such as generating detailed financial summaries in PDF or Excel formats, would provide users with comprehensive insights into their spending patterns. Expanding the application to support multiple users or categories could also make it suitable for household or small business use. Furthermore, integrating modern UI frameworks like JavaFX could offer a more intuitive and visually appealing interface. With these enhancements, the application could evolve into a robust financial management tool, catering to a wider range of users and their diverse budgeting needs.

APPENDIX A

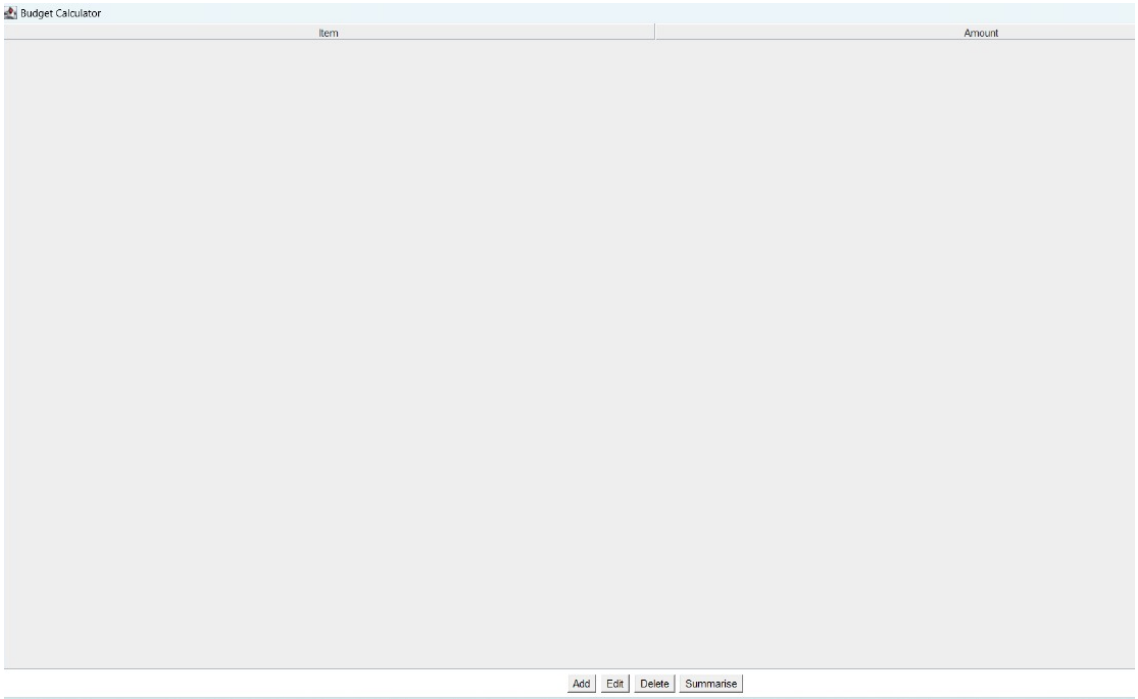
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
public class BudgetCalculator extends Frame {
    private DefaultTableModel tableModel;
    private JTable table;
    public BudgetCalculator() {
        // Frame settings
        setTitle("Budget Calculator");
        setSize(600, 400);
        setLayout(new BorderLayout());
        setVisible(true);
        // Table for budget items
        tableModel = new DefaultTableModel(new String[]{"Item", "Amount"}, 0);
        table = new JTable(tableModel);
        add(new JScrollPane(table), BorderLayout.CENTER);
        // Buttons for actions
        Panel buttonPanel = new Panel();
        Button addButton = new Button("Add");
        Button editButton = new Button("Edit");
        Button deleteButton = new Button("Delete");
        Button summariseButton = new Button("Summarise");
        buttonPanel.add(addButton);
        buttonPanel.add(editButton);
        buttonPanel.add(deleteButton);
        buttonPanel.add(summariseButton);
        add(buttonPanel, BorderLayout.SOUTH);
        // Event Listeners
        addButton.addActionListener(e -> addItem());
        editButton.addActionListener(e -> editItem());
```

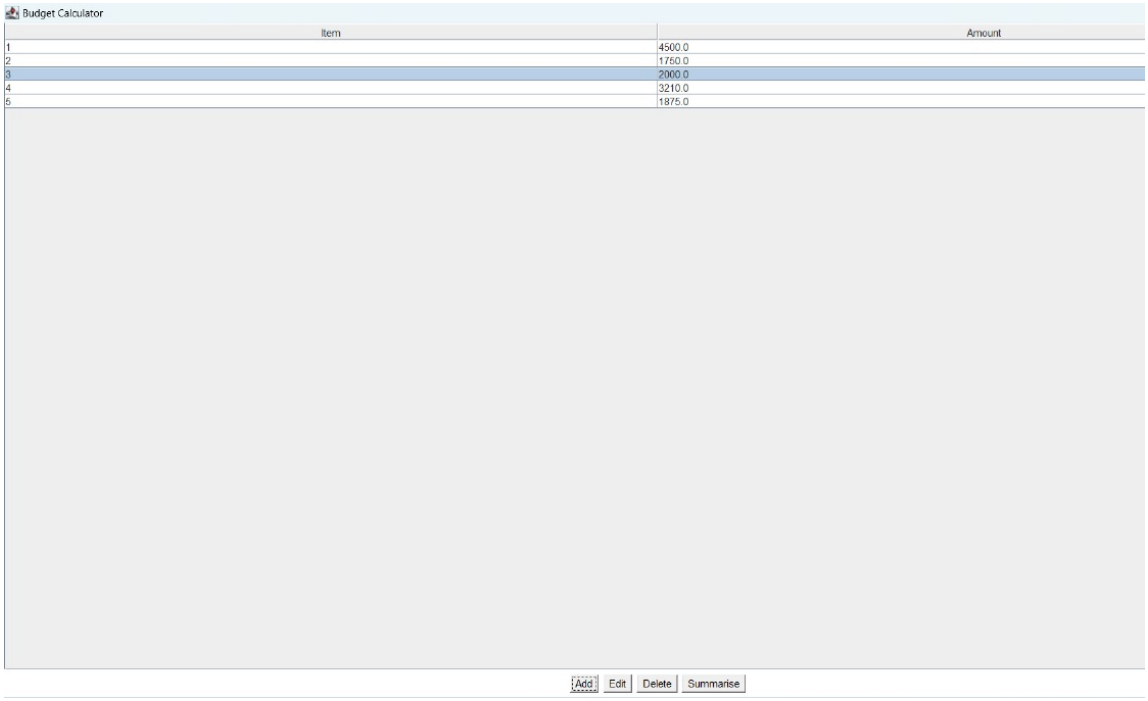
```
        deleteButton.addActionListener(e -> deleteItem());
        summariseButton.addActionListener(e -> summariseItems());

        // Close the application
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent we) {
                System.exit(0);
            }
        });
    }
    private void addItem() {
        String item = JOptionPane.showInputDialog(this, "Enter Item:");
        if (item == null || item.isEmpty()) return;
        String amountStr = JOptionPane.showInputDialog(this, "Enter Amount:");
        if (amountStr == null || amountStr.isEmpty()) return;
        try {
            double amount = Double.parseDouble(amountStr);
            tableModel.addRow(new Object[]{item, amount});
        } catch (NumberFormatException e) {
            JOptionPane.showMessageDialog(this, "Invalid amount. Please enter a number.");
        }
    }
    private void editItem() {
        int selectedRow = table.getSelectedRow();
        if (selectedRow == -1) {
            JOptionPane.showMessageDialog(this, "Please select a row to edit.");
            return;
        }
        String item = JOptionPane.showInputDialog(this, "Enter new Item:",
tableModel.getValueAt(selectedRow, 0));
        if (item == null || item.isEmpty()) return;
        String amountStr = JOptionPane.showInputDialog(this, "Enter new Amount:",
tableModel.getValueAt(selectedRow, 1));
        if (amountStr == null || amountStr.isEmpty()) return;
```

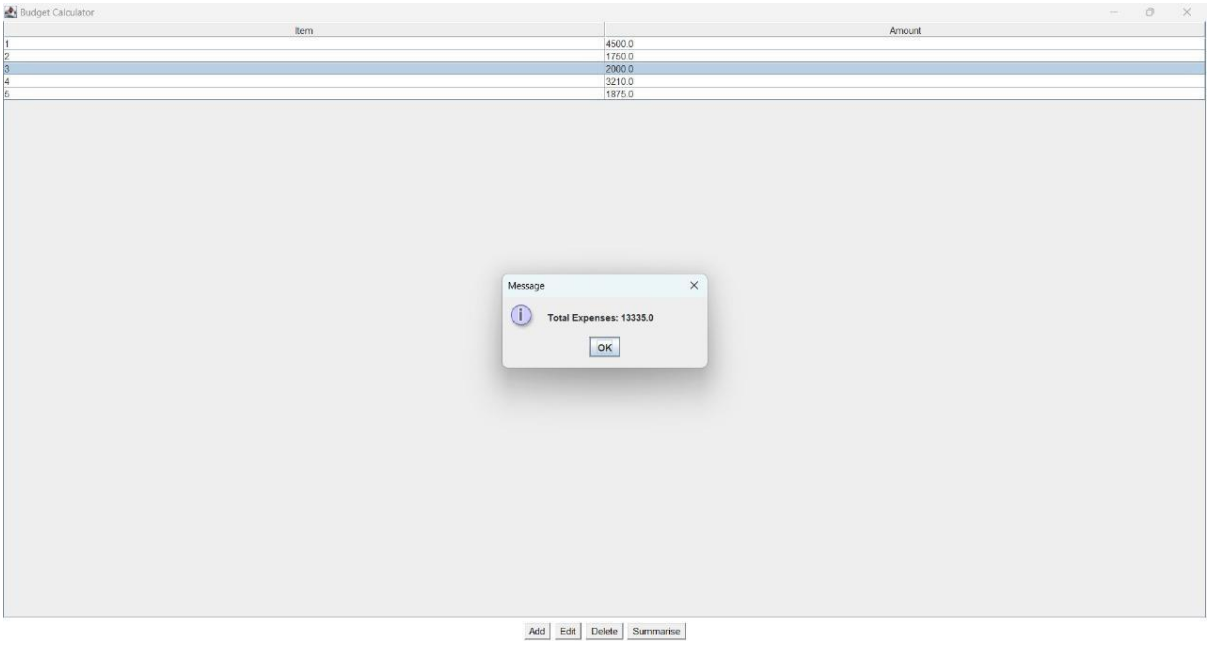
```
            try {
                double amount = Double.parseDouble(amountStr);
                tableModel.setValueAt(item, selectedRow, 0);
                tableModel.setValueAt(amount, selectedRow, 1);
            } catch (NumberFormatException e) {
                JOptionPane.showMessageDialog(this, "Invalid amount. Please enter a number.");
            }
        }
        private void deleteItem() {
            int selectedRow = table.getSelectedRow();
            if (selectedRow == -1) {
                JOptionPane.showMessageDialog(this, "Please select a row to delete.");
                return;
            }
            tableModel.removeRow(selectedRow);
        }
        private void summariseItems() {
            double total = 0;
            for (int i = 0; i < tableModel.getRowCount(); i++) {
                total += (double) tableModel.getValueAt(i, 1);
            }
            JOptionPane.showMessageDialog(this, "Total Expenses: " + total);
        }
        public static void main(String[] args) {
            new BudgetCalculator();
        }
}
```

APPENDIX B

OUTPUT






Budget Calculator
Item
Amount
1
4500.0
2
1760.0
3
2000.0
4
3210.0
5
1875.0
Message
Total Expenses: 13335.0
OK
Add
Edit
Delete
Summarise

REFERENCES

1. **Oracle Java Documentation**

   Oracle's official documentation provides comprehensive details on the Java API, including AWT, Swing, and JTable.

   https://docs.oracle.com/javase/8/docs/api/

2. **Herbert Schildt, "Java: The Complete Reference"**

   A well-known book that covers Java programming concepts, including GUI programming with AWT and Swing, and data handling with JTable.

   https://www.amazon.com/Java-Complete-Reference Eleventh/dp/1260440230

3. **TutorialsPoint – Java AWT and Swing**

   A useful resource for learning about Java GUI libraries and their components with practical examples.

   https://www.tutorialspoint.com/java/java_awt.htm