

EXPENSE TRACKER APPLICATION



An

Object-Oriented Programming through Java Course Project Report

in partial fulfillment of the degree

Bachelor of Technology
in
Computer Science & Engineering

By

E. Sai Krishna Reddy

2103A52014

G. Mohith

2103A52017

V. Sravani

2103A52175

M. Koushik Reddy

2103A52153

Under the guidance of

Mr. P Sravan,

Assistant Professor

School of CS & AI

Submitted to





DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that the **Object Oriented Programming through Java' - Course Project** Report entitled "**Expense Tracker application**" is a record of bona fide work carried out by the students **E. Sai Krishna Reddy, G. Mohith, V. Sravani, M. Koushik Reddy** bearing Roll No(s) 2103A52014, 2103A52017, 2103A52175, 2103A52153 during the academic year 2022-23 in partial fulfillment of the award of the degree of *Bachelor of Technology* in **Computer Science & Engineering** by the SR UNIVERSITY, Ananthasagar, Hasanparthy.

Supervisor

Head of the Department

ACKNOWLEDGEMENT

First and foremost, we express our sincere thanks for the guidance and encouragement rendered by **Mr. P Sravan**, Assistant Professor in the Department of Computer Science & Artificial Intelligence, SR University, Ananthasagar, Hasanparthy. We extend our gratitude for his advice and guidance during the progress of this course project. Secondly, We express our sincere thanks to **Dr. M. Shashikala Professor & Head Department of CS & AI, SR University** who stood as silent inspiration behind this course project. Our heartfelt thanks for her endorsement and valuable suggestions. We wish to express our profound thanks to **Dr. C. V. Guru Rao, Registrar & Dean, School of CS & AI** for providing necessary facilities to make this course project a success. We thank all the members of teaching and non-teaching staff members, and also who have assisted us directly or indirectly for successful completion of this course project. Finally, we would like to express our sincere gratitude to our parents who are constantly encouraging us through-out our lives and for completion of this course project.

DECLARATION

We declare that the course project work entitled “EXPENSE TRECKER APPLICATION” recorded in this course project work does not form part of any other project work. We further declare that the course project work report is based on our work carried out at SR University, Ananthasagar, Hasanparthy, in the third year of our B.Tech course.

Date:8-10-2023

Place: SR University

ABSTRACT

The project's purpose is to track our daily expenses in our daily life. The Expense Tracker application offers an efficient and straightforward approach for users to manage their financial expenditures. Built using Java, the software enables users to monitor their expenses, set budgetary constraints, and receive alerts when spending exceeds the defined budget. Its console-based interface ensures ease of use without the need for additional graphical software. The application is extensible, which makes it adaptable for future enhancements like integrating a GUI or database-backed storage. It acts as a foundational tool that can evolve with increasing functionalities based on users' needs.

TABLE OF CONTENTS

CHAPTER 1	
1. INTRODUCTION	1
2. PROBLEM STATEMENT	1
3. OBJECTIVE	2
4. SYSTEM ANALYSIS	3
5. SYSTEM REQUIREMENTS	3
6. FEASIBILITY STUDY	4-5
7. SOFTWARE ENVIRONMENT	6-12
8. NETWORKING	13-16
9. ELEMENTS USED	17
10. DESIGN	18-20
11. ARCHITECTURE	22
CHAPTER 2	
12. CODE	28-38
CHAPTER 3	
13. RESULT SCREEN	39-43
CHAPTER 4	
14. CONCLUSION	44
16.FUTURE SCOPE	44
17.REFERENCES	45

CHAPTER:1

INTRODUCTION

The main purpose of our online ticket booking system is in today's fast-paced world, keeping track of daily expenses is crucial for financial health. With multiple transactions, both big and small, happening throughout the day, individuals often lose track of their spending habits. Enter the Expense Tracker - a Java-based application designed to provide a clear overview of all financial outgoings. By offering a straightforward console interface, users can promptly input, edit, or view their expenses. The primary advantage of the Expense Tracker lies in its simplicity and direct approach, devoid of unnecessary complications. While there are numerous financial management tools available, the Expense Tracker stands out due to its minimalistic design, ensuring quick understanding and usage. As the foundation of potential future projects, it serves as a stepping stone toward more intricate financial management tools

BACKGROUND:

In life cycle of human after birth the need of materials and belongings is obvious. In order to fulfill our needs and desire we buy goods. The rule of earth is that you must have money in order to buy desired good. So in this way the process of earning and spending goes on in our life. People in order to track their expenses use traditional paper system to keep the record of their income and expenditures. This type of traditional system is burdensome and takes more time. So there must be a management system which must help us to manage our daily earnings and expenses easily, and also helps us to analyze records efficiently. So we figured out a way to eliminate the traditional system with digital, portable, easier and simple way to record these data in just few clicks with our Android application called "Personal Expense Tracker (PET)".

PROBLEM STATEMENT:

Many individuals and businesses need a convenient way to keep track of their expenses. Manual record-keeping can be time-consuming and error-prone. To address this problem, we need to develop an Expense Tracker application that allows users to easily record, categorize, and analyze their expenses.

OBJECTIVE:

The primary objective of the Expense Tracker application is to empower individuals with a robust yet user-friendly tool, facilitating the efficient management and oversight of their daily financial outlays. In an age where financial prudence is more crucial than ever, the application aims to bridge the gap between inadvertent overspending and conscious budgeting. By allowing users to meticulously record each expense, the software ensures that individuals remain keenly aware of their spending habits. Beyond mere recording, the application provides the capability to set a monetary threshold or budget. This budgeting feature not only instills discipline in spending but also serves as a preventive measure against potential financial overreach. Whenever expenses approach or surpass the defined limit, users are promptly alerted, thereby fostering responsible financial behavior. Moreover, recognizing that mistakes happen and circumstances change, the application also offers functionalities to edit previously recorded expenses. In essence, the Expense Tracker doesn't just act as a passive ledger; it takes on the role of an active financial companion, guiding users towards informed financial decisions and healthier monetary habits.

LITERATURE RIVIEW

The Java-based Expense Tracker GUI application offers users a practical solution for managing their expenses and maintaining a budget. Leveraging the Swing framework, the application presents an intuitive graphical interface, complete with a secure login/sign-up system. The code incorporates a User class, ensuring the safety of user accounts through SHA-256 password hashing. The Expense Tracker class forms the heart of the application, enabling expense tracking, budget limit enforcement, and various expense operations. Seamless transitions between screens enhance user experience, with event listeners facilitating responsive interactions. Additionally, the application issues warnings when budget limits are exceeded, ensuring financial prudence. In summary, this code presents a commendable and functional implementation of an expense tracking system, offering user authentication and catering to personal financial management needs.

SYSTEM ANALYSIS

EXISTING SYSTEM:

The existing system of expense tracking often involves manual data entry or reliance on third-party tools with potential limitations in terms of data privacy, customization, and comprehensive analysis. Manual tracking can be time-consuming, error-prone, and may not provide real-time insights. There is a need for a more integrated, user-friendly, and secure solution that allows individuals to effectively manage their finances, income, expenses and make informed financial decisions while ensuring data security and user privacy.

PROPOSED SYSTEM:

The proposed system of Expense Tracking project aims to address the limitations of the existing manual and third-party expense tracking methods by providing a feature-rich, userfriendly, and secure personal finance management application that empowers users to manage their finances effectively. The project will offer a comprehensive platform to track income and expenses, categorize transactions, set budgets, and monitor financial goals.

SYSTEM REQUIREMENTS:

HARDWARE REQUIREMENTS:

1. SYSTEM : Pentium IV 2.4 GHz
2. HARD DISK : 40 GB
3. RAM : 256 MB

SOFTWARE REQUIREMENTS:

1. Operating System : Windows XP
2. Proffessional End : JAVA

FEASIBILITY STUDY

A system is feasible system only if it is feasible within limited recourse and time. In this system each and every process can be feasible for the user and also a developer. It proved user-friendly input such as device-independent inputs and getting a proper solution for the problem.

The different types of feasible systems that have to analyze are,

1. Technical Feasibility
2. Behavioral Feasibility
3. Economical Feasibility
4. Operational Feasibility

1. TECHNICAL FEASIBILITY:

Technical Feasibility is the assessment of the technical view of the system. The system is developed for Dot net environment; a platform-independent tool is used to develop the system. The consideration that is normally associated with technical feasibility include the following

1. Development risk
2. Resource availability
3. Technology

The development risk concerns the probability, the function of all elements and their performance should be the same in all platforms and in the system that is being developed. This system is developed according to the standards and the development software tools are selected in such a way to avoid the problems cited above. The software used to develop this system is Windows XP, visual studio Dot Net is done efficiently, and the concept of SQL helps to create the application backend. These components are also helpful in providing interactivity to Java applications.

2. BEHAVIORAL FEASIBILITY:

It is common knowledge that computer illustrations have something to do with turnover transfers, retraining, and changes in user or developer status. The main emphasis is customer service and personal contact with customers. The feasibility report is directed toward management. It evaluates the impact of the proposed changes on the area in question. The report is a formal document for management use, brief enough and sufficiently non-technical to be understood.

3.ECONOMICAL FEASIBILITY:

Economic feasibility or cost-benefit is an assessment of the economic justification for a computer-based system project. Through this system, the administrator can use the tool from anywhere within their concern. The system is developed using the existing resources. So the project is economically feasible. This is the most frequently used method for evaluating the effectiveness of a user system. More commonly, known as cost analysis the procedure is to determine the benefits and savings that are expected from a candidate system and compare them with costs.

OPTIONAL FEASIBILITY:

Operational Feasibility deals with the study of the prospects of the system. This system operationally eliminates all the tensions of the administrator and helps in effectively tracking the project's progress. This kind of automation will surely reduce the time and energy, previously consumed in manual work. Based on the study, the system proved to be operationally feasible.

SOFTWARE ENVIRONMENT

JAVA TECHNOLOGY:

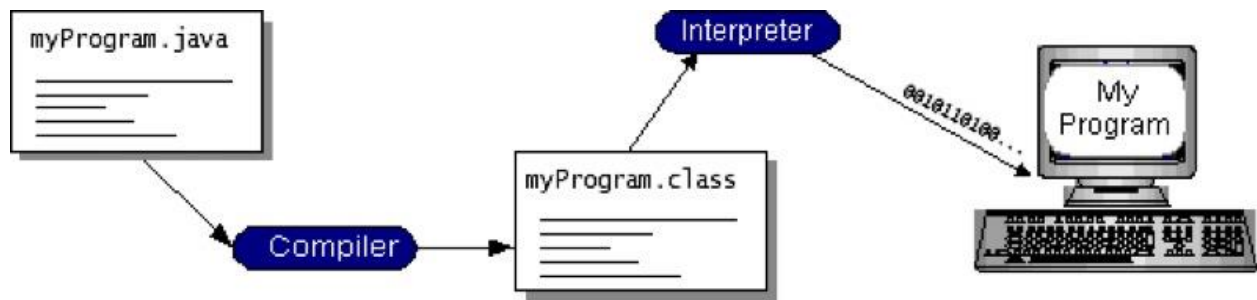
Java technology is both a programming language and a platform.

The Java Programming Language

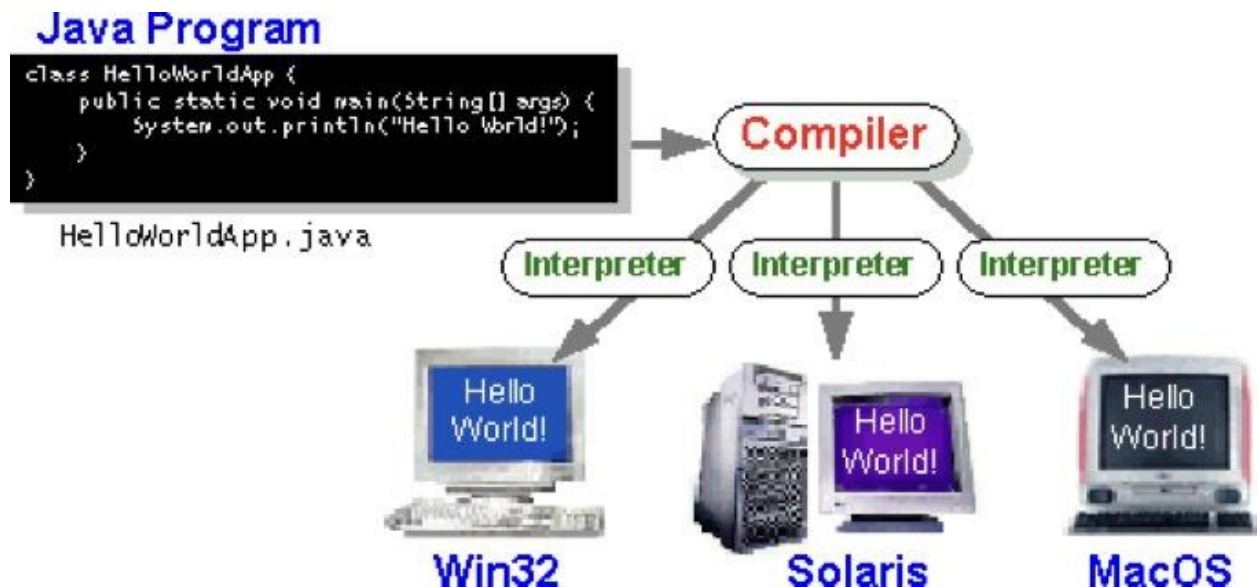
The Java programming language is a high-level language that can be characterized by all of the following buzzwords:

1. Simple
2. Architecture neutral
3. Object oriented
4. Portable
5. Distributed
6. High performance
7. Interpreted
8. Multithreaded
9. Robust
10. Dynamic
11. Secure

With most programming languages, you either compile or interpret a program so that you can run it on your computer. The Java programming language is unusual in that a program is both compiled and interpreted. With the compiler, first you translate a program into an intermediate language called Java byte codes —the platform- independent codes interpreted by the interpreter on the Java platform. The interpreter parses and runs each Java byte code instruction on the computer. Compilation happens just once; interpretation occurs each time the program is executed. The following figure illustrates how this works.



You can think of Java byte codes as the machine code instructions for the Java Virtual Machine (Java VM). Every Java interpreter, whether it's a development tool or a Web browser that can run applets, is an implementation of the Java VM. Java byte codes help make “write once, run anywhere” possible. You can compile your program into byte codes on any platform that has a Java compiler. The byte codes can then be run on any implementation of the Java VM. That means that as long as a computer has a Java VM, the same program written in the Java programming language can run on Windows 2000, a Solaris workstation, or on an iMac.



THE JAVA PLATFORM

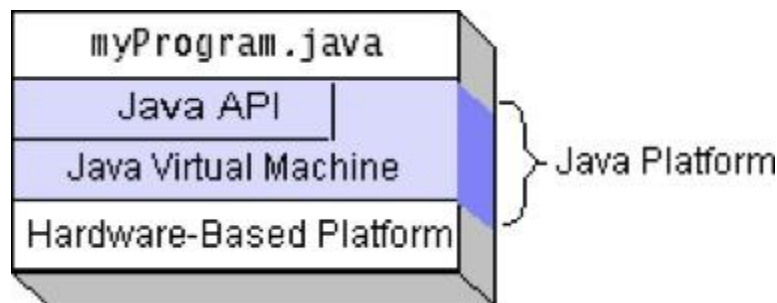
A platform is the hardware or software environment in which a program runs. We've already mentioned some of the most popular platforms like Windows 2000, Linux, Solaris, and MacOS. Most platforms can be described as a combination of the operating system and hardware. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other hardware-based platforms.

1. The Java platform has two components:
2. The Java Virtual Machine (Java VM)

The Java Application Programming Interface (Java API)

You've already been introduced to the Java VM. It's the base for the Java platform and is ported onto various hardware-based platforms.

The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries of related classes and interfaces; these libraries are known as packages. The next section, What Can Java Technology Do? Highlights what functionality some of the packages in the Java API provide. The following figure depicts a program that's running on the Java platform. As the figure shows, the Java API and the virtual machine insulate the program from the hardware.



Native code is code that after you compile it, the compiled code runs on a specific hardware platform. As a platform-independent environment, the Java platform can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and just-in-time byte code compilers can bring performance close to that of native code without threatening portability.

What Can Java Technology Do?

The most common types of programs written in the Java programming language are applets and applications. If you've surfed the Web, you're probably already familiar with applets. An applet is a program that adheres to certain conventions that allow it to run within a Java-enabled browser. An application is a standalone program that runs directly on the Java platform. A special kind of application known as a server serves and supports clients on a network. Examples of servers are Web servers, proxy servers, mail servers, and print servers. Another specialized program is a servlet. A servlet can almost be thought of as an applet that runs on the server side. Java Servlets are a popular choice for building interactive web applications, replacing the use of CGI scripts. Servlets are similar to applets in that they are runtime extensions of applications. Instead of working in browsers, though, servlets run within Java Web servers, configuring or tailoring the server.

How does the API support all these kinds of programs? It does so with packages of software components that provides a wide range of functionality. Every full implementation of the Java platform gives you the following features:

The essentials: Objects, strings, threads, numbers, input and output, data structures, system properties, date and time, and so on.

Applets: The set of conventions used by applets.

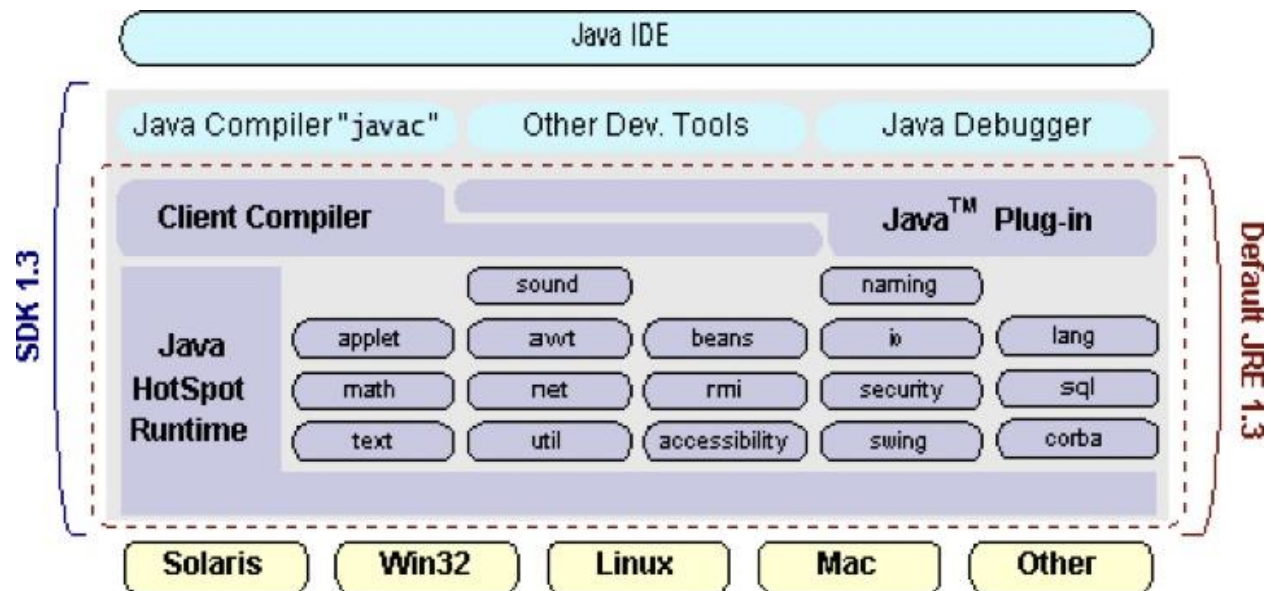
Networking: URLs, TCP (Transmission Control Protocol), UDP (User Datagram Protocol) sockets, and IP (Internet Protocol) addresses.

Internationalization: Help for writing programs that can be localized for users worldwide. Programs can automatically adapt to specific locales and be displayed in the appropriate language.

1. Security: Both low level and high level, including electronic signatures, public and private key management, access control, and certificates.
2. Software components: Known as JavaBeans™, can plug into existing component architectures.
3. Object serialization: Allows lightweight persistence and communication via Remote Method Invocation (RMI).
4. Java Database Connectivity (JDBC™): Provides uniform access to a wide range .

The Java platform also has APIs for 2D and 3D graphics, accessibility, servers, collaboration, telephony, speech, animation, and more. The following figure depicts what is included in the Java

2 SDK.



JDBC

In an effort to set an independent database standard API for Java; Sun Microsystems developed Java Database Connectivity, or JDBC. JDBC offers a generic SQL database access mechanism that provides a consistent interface to a variety of RDBMSs. This consistent interface is achieved through the use of “plug-in” database connectivity modules, or drivers. If a database vendor wishes to have JDBC support, he or she must provide the driver for each platform that the database and Java run on.

To gain a wider acceptance of JDBC, Sun based JDBC’s framework on ODBC. As you discovered earlier in this chapter, ODBC has widespread support on a variety of platforms. Basing JDBC on ODBC will allow vendors to bring JDBC drivers to market much faster than developing a completely new connectivity solution.

JDBC was announced in March of 1996. It was released for a 90 day public review that ended June 8, 1996. Because of user input, the final JDBC v1.0 specification was released soon after.

The remainder of this section will cover enough information about JDBC for you to know what it is about and how to use it effectively. This is by no means a complete overview of JDBC. That would fill an entire book.

JDBC Goals

Few software packages are designed without goals in mind. JDBC is one that, because of its many goals, drove the development of the API. These goals, in conjunction with early reviewer feedback, have finalized the JDBC class library into a solid framework for building database applications in Java.

The goals that were set for JDBC are important. They will give you some insight as to why certain classes and functionalities behave the way they do. The eight design goals for JDBC are as follows:

SQL Level API

The designers felt that their main goal was to define a SQL interface for Java. Although not the lowest database interface level possible, it is at a low enough level for higher-level tools and APIs to be created. Conversely, it is at a high enough level for application programmers to use it confidently. Attaining this goal allows for future tool vendors to “generate” JDBC code and to hide many of JDBC’s complexities from the end user.

SQL Conformance

SQL syntax varies as you move from database vendor to database vendor. In an effort to support a wide variety of vendors, JDBC will allow any query statement to be passed through it to the underlying database driver. This allows the connectivity module to handle non-standard functionality in a manner that is suitable for its users.

JDBC must be implemental on top of common database interfaces

The JDBC SQL API must “sit” on top of other common SQL level APIs. This goal allows JDBC to use existing ODBC level drivers by the use of a software interface. This interface would translate JDBC calls to ODBC and vice versa.

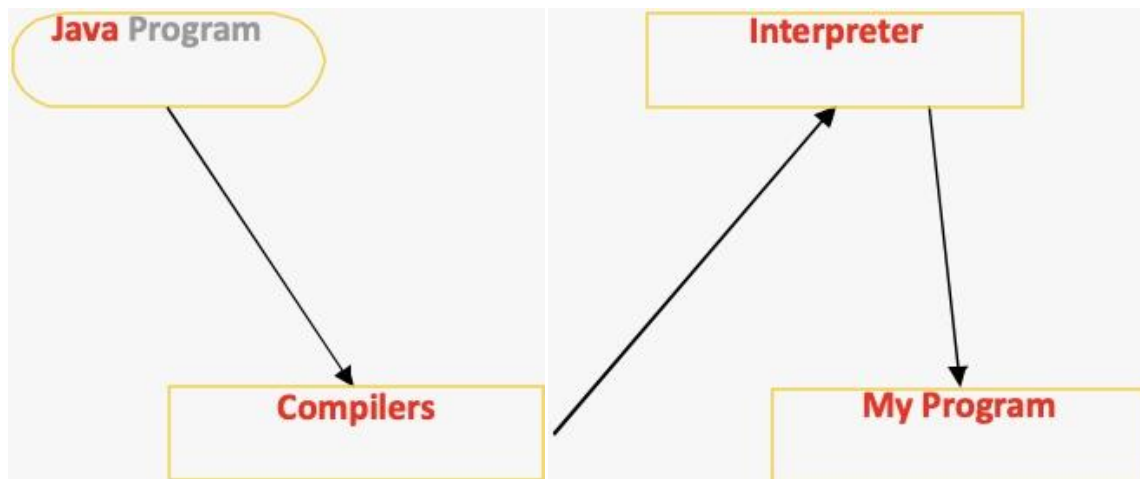
Use strong, static typing wherever possible

Strong typing allows for more error checking to be done at compile time; also, less error appear at runtime.

Keep the common cases simple

Because more often than not, the usual SQL calls used by the programmer are simple SELECT's, INSERT's, DELETE's and UPDATE's, these queries should be simple to perform with JDBC. However, more complex SQL statements should also be possible.

Finally we decided to proceed the implementation using Java Networking. And for dynamically updating the cache table we go for MS Access database.



Java is a high-level programming language that is all of the following

Simple

Object-oriented

Distributed

Interpreted

Robust

Architecture-neutral

Portable

High-performance

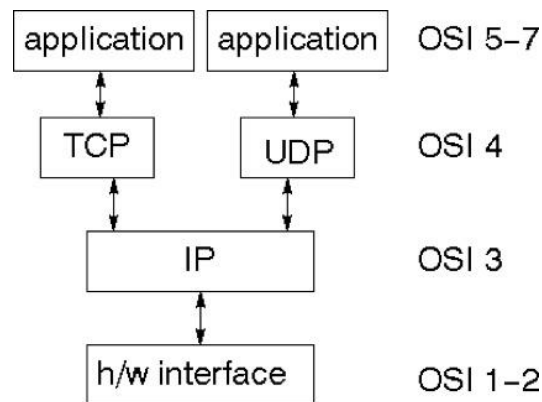
multithreaded

Dynamic

NETWORKING

TCP/IP stack

TCP is a connection-oriented protocol; UDP (User Datagram Protocol) is a connectionless protocol.



IP datagram's

The IP layer provides a connectionless and unreliable delivery system. It considers each datagram independently of the others. Any association between datagram must be supplied by the higher layers. The IP layer supplies a checksum that includes its own header. The header includes the source and destination addresses. The IP layer handles routing through an Internet. It is also responsible for breaking up large datagram into smaller ones for transmission and reassembling them at the other end.

UDP

UDP is also connectionless and unreliable. What it adds to IP is a checksum for the contents of the datagram and port numbers. These are used to give a client/server model - see later.

TCP

TCP supplies logic to give a reliable connection-oriented protocol above IP. It provides a virtual circuit that two processes can use to communicate.

Internet addresses

In order to use a service, you must be able to find it. The Internet uses an address scheme for machines so that they can be located. The address is a 32 bit integer which gives the IP address. This encodes a network ID and more addressing. The network ID falls into various classes according to the size of the network address.

Network address

Class A uses 8 bits for the network address with 24 bits left over for other addressing. Class B uses 16 bit network addressing. Class C uses 24 bit network addressing and class D uses all 32.

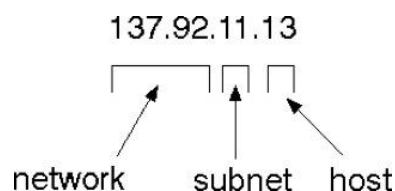
Subnet address

Internally, the UNIX network is divided into sub networks. Building 11 is currently on one sub network and uses 10-bit addressing, allowing 1024 different hosts.

Host address

8 bits are finally used for host addresses within our subnet. This places a limit of 256 machines that can be on the subnet.

Total address:



Port addresses

A service exists on a host, and is identified by its port. This is a 16 bit number. To send a message to a server, you send it to the port for that service of the host that it is running on. This is not location transparency! Certain of these ports are "well known".

Sockets

A socket is a data structure maintained by the system to handle network connections. A socket is created using the call `socket`. It returns an integer that is like a file descriptor. In fact, under Windows, this handle can be used with Read File and Write File functions.

```
#include <sys/types.h> #include <sys/socket.h>
```

```
int socket(int family, int type, int protocol);
```

Here "family" will be `AF_INET` for IP communications, protocol will be zero, and type will depend on whether TCP or UDP is used. Two processes wishing to communicate over a network create a socket each. These are similar to two ends of a pipe - but the actual pipe does not yet exist.

JFree Chart

JFreeChart is a free 100% Java chart library that makes it easy for developers to display professional quality charts in their applications. JFreeChart's extensive feature set includes:

A consistent and well-documented API, supporting a wide range of chart types;

A flexible design that is easy to extend, and targets both server-side and client-side applications;

Support for many output types, including Swing components, image files (including PNG and JPEG), and vector graphics file formats (including PDF, EPS and SVG);

JFreeChart is "open source" or, more specifically, free software. It is distributed under the terms of the GNU Lesser General Public Licence (LGPL), which permits use in proprietary applications.

Map Visualizations

Charts showing values that relate to geographical areas. Some examples include:

- (a) population density in each state of the United States,
- (b) income per capita for each country in Europe,
- (c) life expectancy in each country of the world. The tasks in this project include:

Sourcing freely redistributable vector outlines for the countries of the world, states/provinces in particular countries (USA in particular, but also other areas);

Creating an appropriate dataset interface (plus default implementation), a rendered, and integrating this with the existing XYPlot class in JFreeChart;

Testing, documenting, testing some more, documenting some more.

Time Series Chart Interactivity

Implement a new (to JFreeChart) feature for interactive time series charts --- to display a separate control that shows a small version of ALL the time series data, with a sliding "view" rectangle that allows you to select the subset of the time series data to display in the main chart.

Dashboards

There is currently a lot of interest in dashboard displays. Create a flexible dashboard mechanism that supports a subset of JFreeChart chart types (dials, pies, thermometers, bars, and lines/time series) that can be delivered easily via both Java Web Start and an applet.

Property Editors

The property editor mechanism in JFreeChart only handles a small subset of the properties that can be set for charts. Extend (or reimplement) this mechanism to provide greater end-user control over the appearance of the charts.

DEFINITIONS OF THE ELEMENTS USED IN THE PROJECT

Key Java concepts and components used in the code:

Swing: Swing is a set of GUI components provided by Java for building desktop applications. It is used extensively in the code to create the user interface elements such as buttons, text areas, and panels.

JFrame: JFrame is a top-level container used to create the main window of the application. It is the primary window that contains other GUI components.

JButton: JButton is used to create clickable buttons that trigger actions when pressed.

JTextArea: JTextArea is used to display and edit multiple lines of text. It is used to display the list of expenses and other information.

JComboBox: JComboBox is used to create a drop-down list of items. In this code, it is used to select an expense for editing or deletion.

JPanel: JPanel is used to create containers that can hold other GUI components. It is used to organize the layout of the GUI.

Layout Managers: The code uses layout managers like GridBagLayout and FlowLayout to control the positioning and sizing of GUI components within the JFrame.

ActionListeners: ActionListener is an interface used for handling button clicks and other events. It is implemented to define the actions to be performed when specific buttons are clicked.

JOptionPane: JOptionPane is used for displaying dialog boxes for input and information, such as inputting expense details, confirming deletions, and displaying error messages.

Data Structures: The code utilizes data structures such as List (specifically ArrayList) to store and manage expenses.

Object-Oriented Programming (OOP): The code follows OOP principles with the creation of classes (Expense and ExpenseTracker) to encapsulate and manage data and behavior related to expenses.

GUI Interaction: The code allows users to perform operations like adding expenses, viewing expenses, editing expenses, deleting expenses, and setting a budget limit through the GUI.

User Authentication: The code includes a simple user authentication mechanism with login and signup screens, although the actual validation logic is not implemented in the provided code.

DESIGN:

- Features Of the Expense Tracker application are as follows:
- Add Expenses: Users can input their expenses with a description and amount.
- View Expenses: Display a list of all entered expenses.
- Total Expense Overview: See the combined amount of all expenses.
- Edit Expenses: Modify a previously entered expense.
- Set Budget: Define a budget limit.
- Budget Alerts: Receive warnings if total expenses exceed the budget.

SYSTEM DESIGN AND MODULES

- **User Interface Module:** Handles user input and displays output. Contains the main menu and user prompts.
- **Expense Management Module:** Manages operations related to expenses like adding, viewing, and editing.
- **Budget Management Module:** Allows users to set, view, and check if the total expenses exceed the budget.

MODULE DETAILS

User Interface Module:

- **Main Menu:** Display available options like adding/viewing expenses, setting/viewing budget, etc.
- **User Prompts:** Guide users to input data or choose operations.
- **Feedback Displays:** Notify users about successful operations or errors.

Expense Management Module:

- **Add Expense:** Accepts description and amount to add an expense.
- **View Expenses:** Lists all added expenses with an index, description, and amount.
- **Edit Expenses:** Uses the index to modify an existing expense's description and amount.

Budget Management Module:

- **Set Budget:** Allows input of a budget limit.
- **View Budget:** Displays the set budget and provides alerts if expenses surpass the budget.

INPUT DESIGN

The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation and those steps are necessary to put transaction data in to a usable form for processing can be achieved by inspecting the computer to read data from a written or printed document or it can occur by having people keying the data directly into the system. The design of input focuses on controlling the amount of input required, controlling the errors, avoiding delay, avoiding extra steps and keeping the process simple. The input is designed in such a way so that it provides security and ease of use with retaining the privacy. Input Design considered the following things:

- What data should be given as input?
- How the data should be arranged or coded?
- The dialog to guide the operating personnel in providing input.
- Methods for preparing input validations and steps to follow when error occur.

OBJECTIVES

- Input Design is the process of converting a user-oriented description of the input into a computer-based system. This design is important to avoid errors in the data input process and show the correct direction to the management for getting correct information from the computerized system.
- It is achieved by creating user-friendly screens for the data entry to handle large volume of data. The goal of designing input is to make data entry easier and to be free from errors. The data entry screen is designed in such a way that all the data manipulates can be performed. It also provides record viewing facilities.
- When the data is entered it will check for its validity. Data can be entered with the help of screens. Appropriate messages are provided as when needed so that the user
- will not be in maize of instant. Thus the objective of input design is to create an input layout that is easy to follow

OUTPUT DESIGN

A quality output is one, which meets the requirements of the end user and presents the information clearly. In any system results of processing are communicated to the users and to other system through outputs. In output design it is determined how the information is to be displaced for immediate need and also the hard copy output. It is the most important and direct source information to the user. Efficient and intelligent output design improves the system's relationship to help user decision-making.

- Designing computer output should proceed in an organized, well thought out manner; the right output must be developed while ensuring that each output element is designed so that people will find the system can use easily and effectively. When analysis design computer output, they should Identify the specific output that is needed to meet the requirements.
- Select methods for presenting information.

- Create document, report, or other formats that contain information produced by the system.
- The output form of an information system should accomplish one or more of the following objectives.

[1] Convey information about past activities, current status or projections of the

[2] Future.

[3] Signal important events, opportunities, problems, or warnings.

[4] Trigger an action.

[5] Confirm an action.

SYSTEM DESIGN

INTRODUCTION

Software design sits at the technical kernel of the software engineering process and is applied regardless of the development paradigm and area of application. Design is the first step in the development phase for any engineered product or system. The designer's goal is to produce a model or representation of an entity that will later be built. Beginning, once system requirements have been specified and analyzed, system design is the first of the three technical activities -design, code and test that is required to build and verify software.

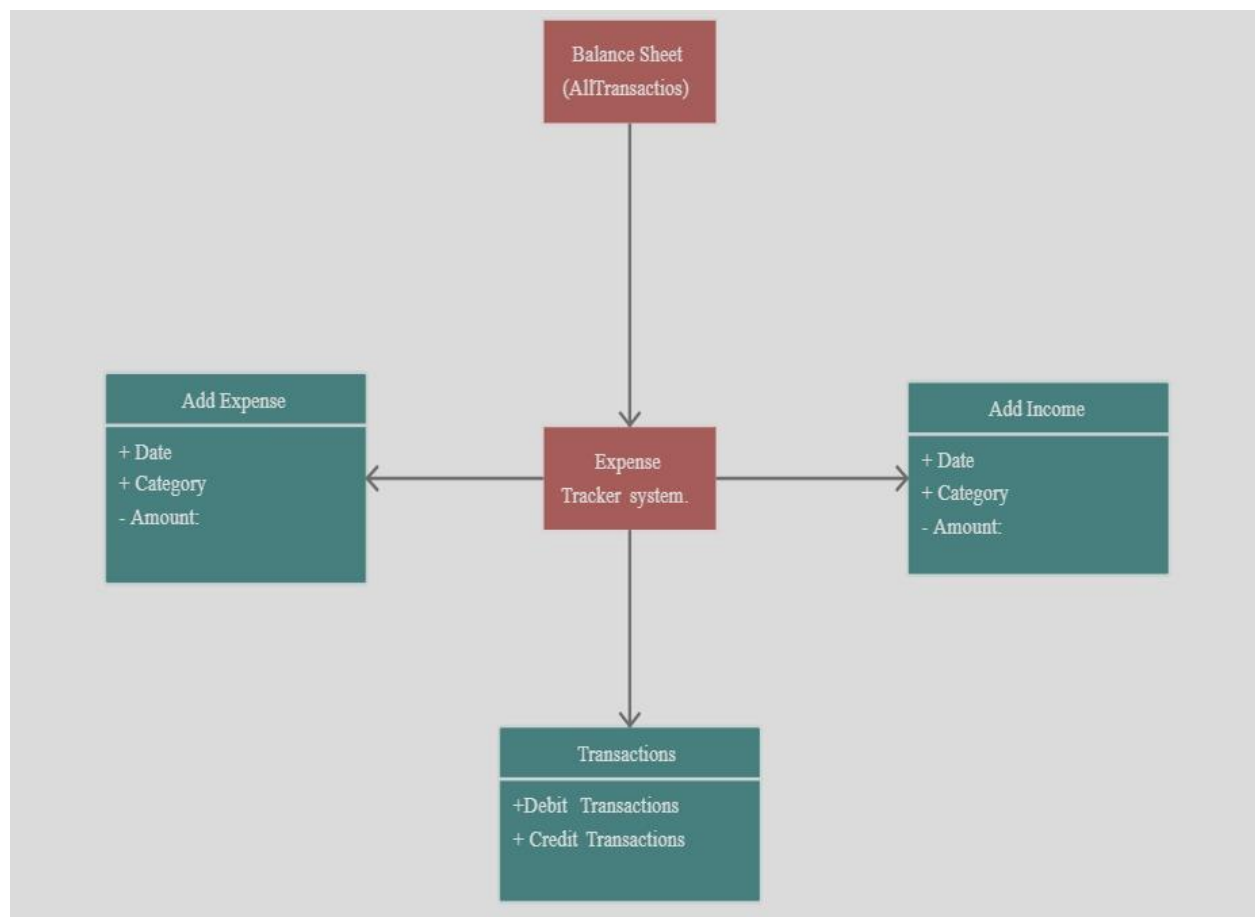
The importance can be stated with a single word "Quality". Design is the place where quality is fostered in software development. Design provides us with representations of software that can assess for quality. Design is the only way that we can accurately translate a customer's view into a finished software product or system. Software design serves as a foundation for all the software engineering steps that follow. Without a strong design we risk building an unstable system. One that will be difficult to test, one whose quality cannot be assessed until the last stage.

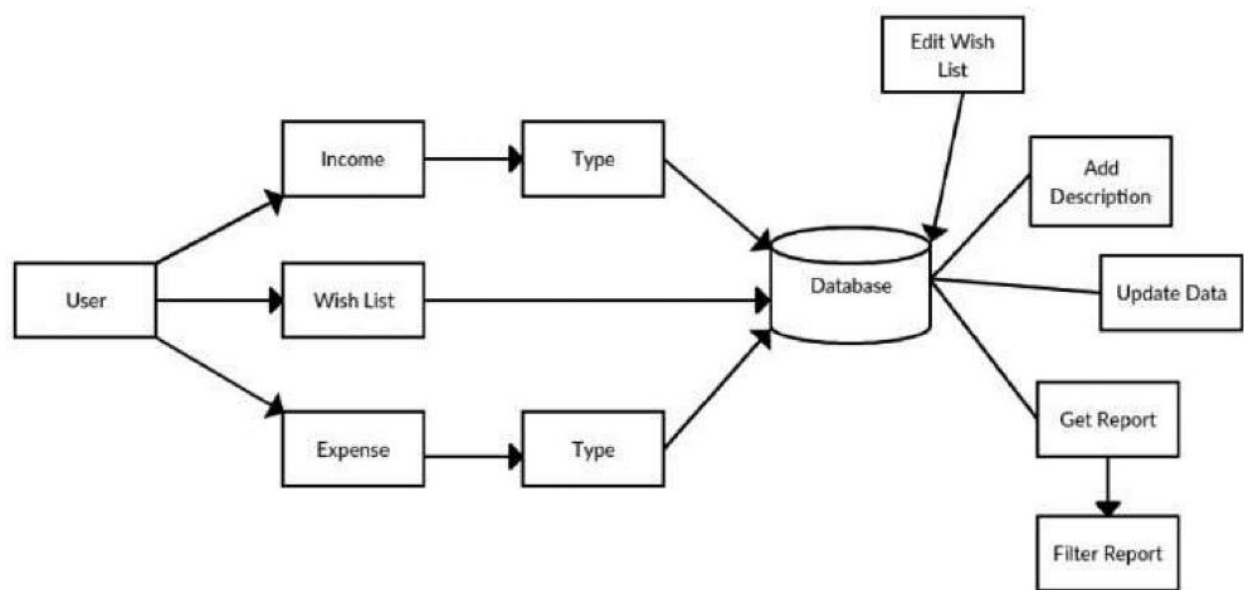
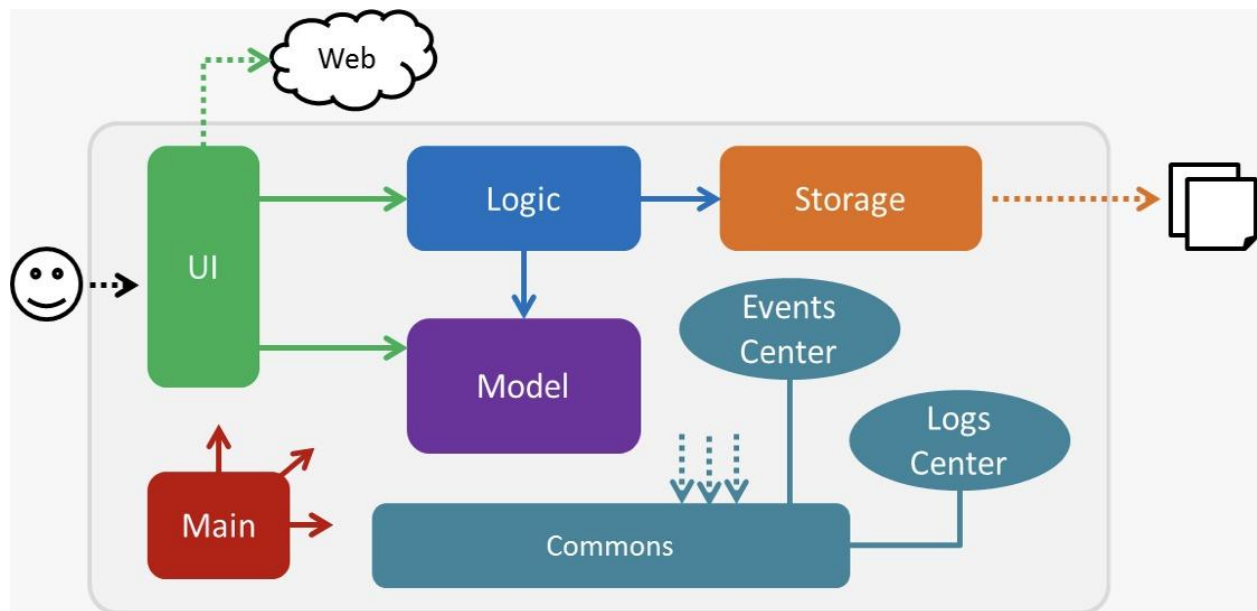
During design, progressive refinement of data structure, program structure, and procedural details are developed reviewed and documented. System design can be viewed from either technical or project management perspective. From the technical point of view, design is comprised of four activities architectural design, data structure design, interface design and procedural design.

DATA FLOW DIAGRAM:

The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of the input data to the system, various processing carried out on these data, and the output data is generated by the system.

ARCHITECTURE





BLOCK DIAGRAM

UML DIAGRAMS:

The Unified Modeling Language allows the software engineer to express an analysis model using the modeling notation that is governed by a set of syntactic semantic and pragmatic rules.

A UML system is represented using five different views that describe the system from distinctly different perspective. Each view is defined by a set of diagram, which is as follows.

User Model View:

- This view represents the system from the users perspective.
- The analysis representation describes a usage scenario from the end- users perspective.

Structural model view:

- In this model the data and functionality are arrived from inside the system.
- This model view models the static structures.

Behavioral Model View

- It represents the dynamic of behavioral as parts of the system, depicting the interactions of collection between various structural elements described in the user model and structural model view.

Implementation Model View

- In this the structural and behavioral as parts of the system are represented as they are to be built.

Environmental Model View

- In this the structural and behavioral aspects of the environment in which the system is to be implemented are represented.

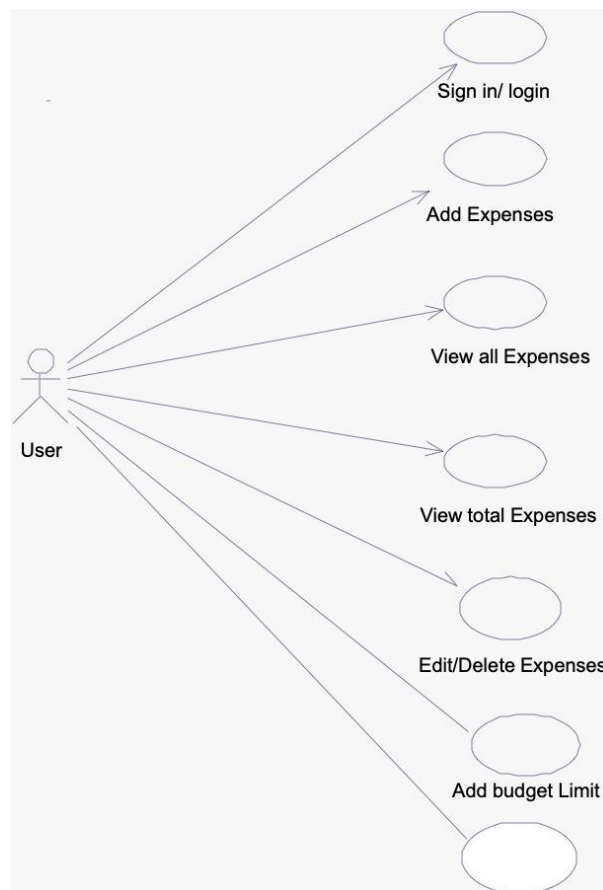
UML is specifically constructed through two different domains they are:

- ✓ UML Analysis modeling, this focuses on the user model and structural model views of the system.
- ✓ UML design modeling, which focuses on the behavioral modeling.

USE CASE DIAGRAM

Use Case: Use case describes the behavior of a system. It is used to structure things in a model. It contains multiple scenarios, each of which describes a sequence of actions that is clear enough for outsiders to understand.

Actor: An actor represents a coherent set of roles that users of a system play when interacting with the use cases of the system. An actor participates in use cases to accomplish an overall purpose. An actor can represent the role of a human, a device, or any other system



Use Case1:

Login Primary Actor: User Secondary Actor: System Stakeholders:

User: Wants to update transaction. Preconditions: User must login.

Post Conditions: Transaction is saved.

Basic flow:

- User login to system using password.
- User selects appropriate option from the menu.

Alternate flow:

At any time, user enters a wrong password then system notifies.

Use Case2: Enter Expense Primary Actor: User Secondary Actor: System Stakeholders:

Preconditions: User have been logged in. Post Conditions: New expense is saved. Basic flow:

- User selects category.

- Expense amount is updated.

Alternate flow:

At any time, system fails, system displays the error.

Use Case3: View Report Primary Actor: User Secondary Actor: System Stakeholders:

User: User have been logged in.

Preconditions: User gets informed about the transactions.

Post Conditions: Transaction is saved.

Basic flow:

- User selects category and tie frame.
- Selected result is displayed.

Alternate flow:

At any time, system fails, system displays the error.

Use Case UC4: Save Data

Primary Actor: System

Preconditions: Transactions are provided. Post conditions: Display saved message. Basic flow:
The user enters the transaction amount with category.

System saves the amount to the category and display saved message.

Alternate flow:

At any time, system fails, system regains previous status. Error message is displayed.

Use Case UC5: Prepare Report

Primary Actor: System

Preconditions: Transactions are provided.

Post conditions: Display the report graphically. Basic flow:

The user selects the result category. System prepares report.

Display prepared report graphically. Alternate flow:

At any time, system fails, system regains previous status. Error message is displayed.

Use Case UC6: Notify User

Primary Actor: System
Preconditions: Notify user setting must be enabled. Post conditions: Alert message is displayed.

Basic flow:

System generates notification message.

Display message in notification bar.

Alternate flow:

At any time, system fails, system regains previous status. Error message is displayed.

CHAPTER 2: IMPLEMENTATION

Code:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.List;
import javax.swing.border.EmptyBorder;

class Expense {
    private String description;
    private double amount;

    public Expense(String description, double amount) {
        this.description = description;
        this.amount = amount;
    }

    public double getAmount() {
        return amount;
    }

    public void setAmount(double amount) {
        this.amount = amount;
    }

    public String getDescription() {
        return description;
    }
}
```

```

    }

    public void setDescription(String description) {
        this.description = description;
    }

    @Override
    public String toString() {
        return description + ": Rs" + String.format("%.2f", amount);
    }
}

class ExpenseTracker {
    private List<Expense> expenses;
    private double budgetLimit;

    public ExpenseTracker() {
        this.expenses = new ArrayList<>();
        this.budgetLimit = 10000.0;
    }

    public void addExpense(Expense expense) {
        expenses.add(expense);
    }

    public List<Expense> getExpenses() {
        return new ArrayList<>(expenses);
    }

    public double getTotalExpenses() {
        return expenses.stream().mapToDouble(Expense::getAmount).sum();
    }
}

```

```

    }

    public void setBudget(double budget) {
        this.budgetLimit = budget;
    }

    public double getBudget() {
        return budgetLimit;
    }

    public boolean isOverBudget() {
        return getTotalExpenses() > budgetLimit;
    }

    public void editExpense(String oldDescription, String newDescription, double newAmount) {
        for (Expense expense : expenses) {
            if (expense.getDescription().equals(oldDescription)) {
                expense.setDescription(newDescription);
                expense.setAmount(newAmount);
                break;
            }
        }
    }

    public void deleteExpense(String description) {
        expenses.removeIf(expense -> expense.getDescription().equals(description));
    }
}

public class ExpenseTrackerGUI {
    private ExpenseTracker tracker;

```

```

private JFrame frame;
private JTextArea textArea;
private JComboBox<Expense> expenseComboBox;
private JPanel loginPanel;
private JTextField usernameField;
private JPasswordField passwordField;
private JPanel signupPanel;
private JTextField newUsernameField;
private JPasswordField newPasswordField;
private JPasswordField confirmPasswordField;

public ExpenseTrackerGUI() {
    tracker = new ExpenseTracker();
    createWindow();
    showOperations();
}

private void createWindow() {
    frame = new JFrame("Expense Tracker");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(500, 500);
    frame.setLayout(new FlowLayout());

    JButton startButton = new JButton("Start ExpenseTracker");
    startButton.addActionListener(e -> showOperations());
    frame.add(startButton);

    frame.setVisible(true);
}

private void showOperations() {
    frame.getContentPane().removeAll();

```

```

GridBagConstraints gbc = new GridBagConstraints();
gbc.gridwidth = GridBagConstraints.REMAINDER;
gbc.fill = GridBagConstraints.HORIZONTAL;
gbc.insets = new Insets(5, 5, 5, 5);

GridBagLayout gbl = new GridBagLayout();
frame.setLayout(gbl);

frame.add(createButton("Add Expense", this::addExpense), gbc);
frame.add(createButton("View All Expenses", e -> viewExpenses()), gbc);
frame.add(createButton("View Total Expense", e -> viewTotalExpense()), gbc);
frame.add(createButton("Edit Expense", this::editExpense), gbc);
frame.add(createButton("Delete Expense", this::deleteExpense), gbc);
frame.add(createButton("Set Budget Limit", this::setBudgetLimit), gbc);
frame.add(createButton("Exit", e -> System.exit(0)), gbc);

textArea = new JTextArea(10, 30);
textArea.setEditable(false);
JScrollPane scrollPane = new JScrollPane(textArea);
scrollPane.setPreferredSize(new Dimension(400, 200));
gbc.fill = GridBagConstraints.BOTH;
gbc.weighty = 1;
frame.add(scrollPane, gbc);

expenseComboBox = new JComboBox<>();
updateExpenseComboBox();
gbc.weighty = 0;
frame.add(expenseComboBox, gbc);

frame.pack();
frame.setLocationRelativeTo(null);

```

```

        frame.setVisible(true);
    }
    private JButton createButton(String text, ActionListener listener) {
        JButton button = new JButton(text);
        button.setPreferredSize(new Dimension(200, 50));
        button.addActionListener(listener);
        return button;
    }
    private void addExpense(ActionEvent event) {
        String description = JOptionPane.showInputDialog(frame, "Enter Expense Description:");
        String amountStr = JOptionPane.showInputDialog(frame, "Enter Amount:");
        double amount = Double.parseDouble(amountStr);

        if (tracker.getTotalExpenses() + amount > tracker.getBudget()) {
            JOptionPane.showMessageDialog(frame, "Budget Limit Exceeded!", "Warning",
JOptionPane.WARNING_MESSAGE);
            return;
        }

        tracker.addExpense(new Expense(description, amount));
        updateExpenseComboBox();
        viewExpenses();
    }

    private void viewExpenses() {
        StringBuilder expensesText = new StringBuilder();
        for (Expense expense : tracker.getExpenses()) {
            expensesText.append(expense).append("\n");
        }
        textArea.setText(expensesText.toString());
        checkBudget();
    }

```

```

    }

    private void viewTotalExpense() {
        textArea.setText("Total Expenses: $" + String.format("%.2f", tracker.getTotalExpenses()));
        checkBudget();
    }

    private void editExpense(ActionEvent event) {
        Expense selectedExpense = (Expense) expenseComboBox.getSelectedItem();
        if (selectedExpense != null) {
            String newDescription = JOptionPane.showInputDialog(frame, "Enter New Description:",
selectedExpense.getDescription());
            String newAmountStr = JOptionPane.showInputDialog(frame, "Enter New Amount:",
selectedExpense.getAmount());
            double newAmount = Double.parseDouble(newAmountStr);
            tracker.editExpense(selectedExpense.getDescription(), newDescription, newAmount);
            updateExpenseComboBox();
            viewExpenses();
        }
    }

    private void deleteExpense(ActionEvent event) {
        Expense selectedExpense = (Expense) expenseComboBox.getSelectedItem();
        if (selectedExpense != null) {
            int confirm = JOptionPane.showConfirmDialog(frame, "Are you sure you want to delete
this expense?", "Confirm Delete", JOptionPane.YES_NO_OPTION);
            if (confirm == JOptionPane.YES_OPTION) {
                tracker.deleteExpense(selectedExpense.getDescription());
                updateExpenseComboBox();
                viewExpenses();
            }
        }
    }
}

```



```

private void setBudgetLimit(ActionEvent event) {
    String budgetStr = JOptionPane.showInputDialog(frame, "Set New Budget Limit:",
tracker.getBudget());
    double budget = Double.parseDouble(budgetStr);
    tracker.setBudget(budget);
    viewExpenses();
    checkBudget();
}

private void updateExpenseComboBox() {
    expenseComboBox.removeAllItems();
    for (Expense expense : tracker.getExpenses()) {
        expenseComboBox.addItem(expense);
    }
}

private void checkBudget() {
    if (tracker.isOverBudget()) {
        JOptionPane.showMessageDialog(frame, "Budget Limit Exceeded!", "Warning",
JOptionPane.WARNING_MESSAGE);
    }
}

private void createSignupWindow() {
    signupPanel = new JPanel(new GridBagLayout());
    GridBagConstraints gbc = new GridBagConstraints();
    gbc.gridwidth = GridBagConstraints.REMAINDER;
    gbc.fill = GridBagConstraints.HORIZONTAL;

    newUsernameField = new JTextField(15);
    newPasswordField = new JPasswordField(15);
    confirmPasswordField = new JPasswordField(15);
}

```

```

signupPanel.setBorder(new EmptyBorder(50, 50, 50, 50));

signupPanel.add(new JLabel("Username:"), gbc);
signupPanel.add(new UsernameField, gbc);
signupPanel.add(new JLabel("Password:"), gbc);
signupPanel.add(new PasswordField, gbc);
signupPanel.add(new JLabel("Confirm Password:"), gbc);
signupPanel.add(confirmPasswordField, gbc);
JButton signupButton = new JButton("Signup");
signupButton.addActionListener(this::signupAction);
signupPanel.add(signupButton, gbc);

JButton backButton = new JButton("Back to Login");
backButton.addActionListener(e -> switchToLogin());
signupPanel.add(backButton, gbc);

frame.setContentPane(signupPanel);
frame.pack();
frame.setLocationRelativeTo(null);
frame.setVisible(true);
}

private void switchToLogin() {
    frame.setContentPane(loginPanel);
    frame.pack();
    frame.setLocationRelativeTo(null);
    frame.setVisible(true);
}

private void signupAction(ActionEvent event) {
    String newUsername = new UsernameField.getText();
    String newPassword = new String(new PasswordField.getPassword());

```

```

String confirmPassword = new String(confirmPasswordField.getPassword());

if (!newPassword.equals(confirmPassword)) {
    JOptionPane.showMessageDialog(frame, "Passwords do not match!", "Signup Failed",
JOptionPane.ERROR_MESSAGE);
    return;
}
System.out.println("New user created: Username - " + newUsername + ", Password - " +
newPassword);
    JOptionPane.showMessageDialog(frame, "User created successfully!", "Signup Successful",
JOptionPane.INFORMATION_MESSAGE);
    switchToLogin();
}
private void createLoginWindow() {
    loginPanel = new JPanel(new GridBagLayout());
    GridBagConstraints gbc = new GridBagConstraints();
    gbc.gridwidth = GridBagConstraints.REMAINDER;
    gbc.fill = GridBagConstraints.HORIZONTAL;

    usernameField = new JTextField(15);
    passwordField = new JPasswordField(15);

    loginPanel.setBorder(new EmptyBorder(50,50,50,50)); // Adding some padding

    loginPanel.add(new JLabel("Username:"), gbc);
    loginPanel.add(usernameField, gbc);
    loginPanel.add(new JLabel("Password:"), gbc);
    loginPanel.add(passwordField, gbc);
    JButton loginButton = new JButton("Login");
    loginButton.addActionListener(this::loginAction);
    loginPanel.add(loginButton, gbc);

```

```

JButton signupButton = new JButton("Signup");
signupButton.addActionListener(e -> createSignupWindow());
loginPanel.add(signupButton, gbc);

frame.setContentPane(loginPanel);
frame.pack();
frame.setLocationRelativeTo(null);
frame.setVisible(true);
}
private void loginAction(ActionEvent event) {
    String username = usernameField.getText();
    String password = new String(passwordField.getPassword());

    if (validateCredentials(username, password)) {
        showOperations();
    } else {
        JOptionPane.showMessageDialog(frame, "Invalid username or password", "Login
Failed", JOptionPane.ERROR_MESSAGE);
    }
}
private boolean validateCredentials(String username, String password) {

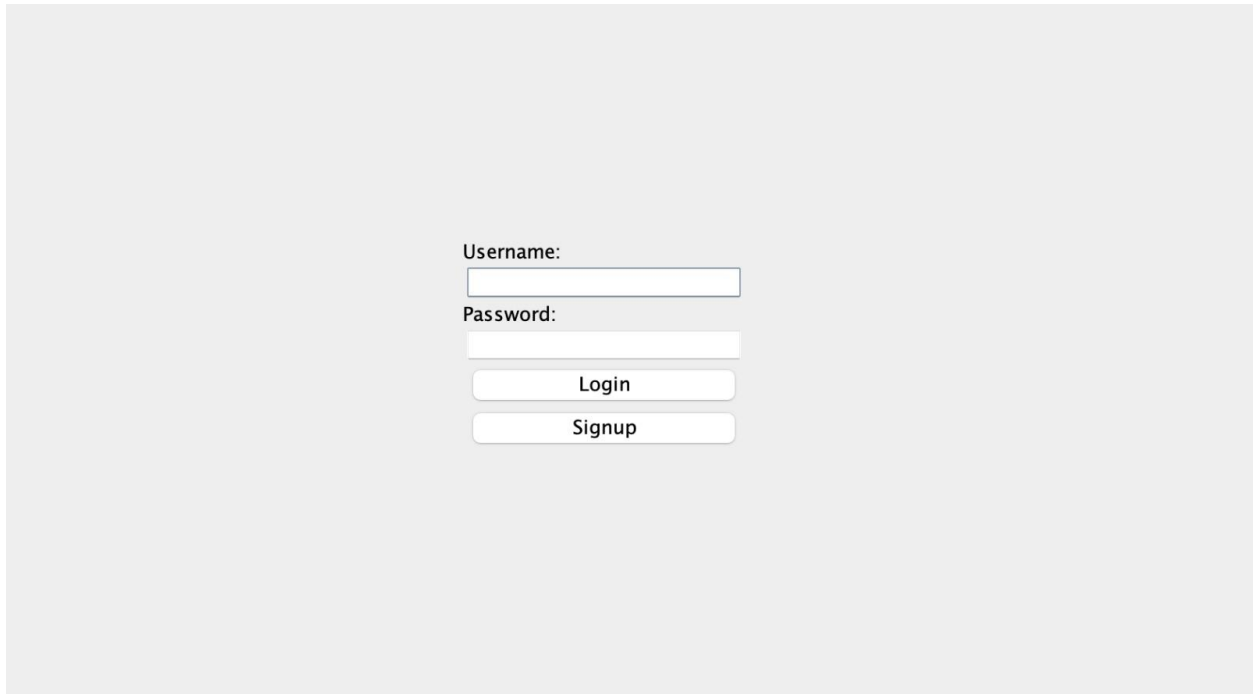
    return true;
}

public static void main(String[] args) {
    SwingUtilities.invokeLater(() -> {
        ExpenseTrackerGUI expenseTrackerGUI = new ExpenseTrackerGUI();
        expenseTrackerGUI.createLoginWindow();
    });
}

```

```
}  
}
```

CHAPTER 3: RESULTS SCREENS



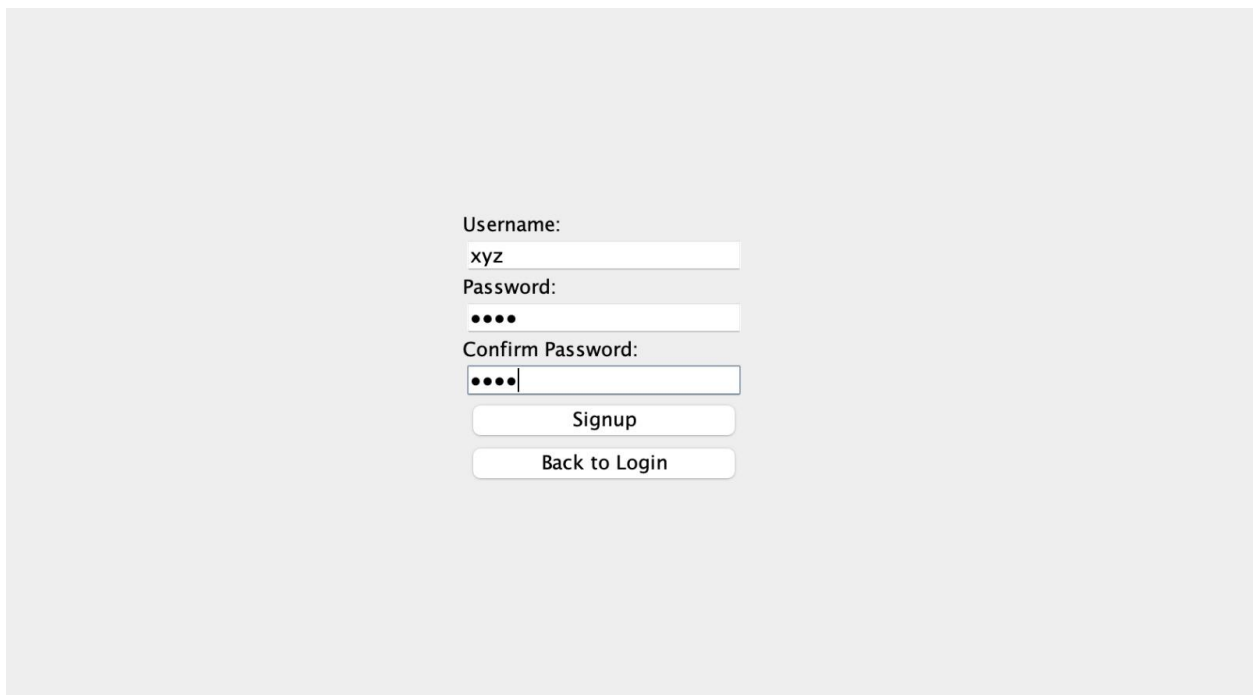
A login form UI mockup on a light gray background. It features two text input fields for 'Username:' and 'Password:'. Below the password field are two rounded rectangular buttons: 'Login' and 'Signup'.

Username:

Password:

Login

Signup



A signup form UI mockup on a light gray background. It features three text input fields: 'Username:' (containing 'xyz'), 'Password:' (containing four dots), and 'Confirm Password:' (containing four dots). Below the password fields are two rounded rectangular buttons: 'Signup' and 'Back to Login'.

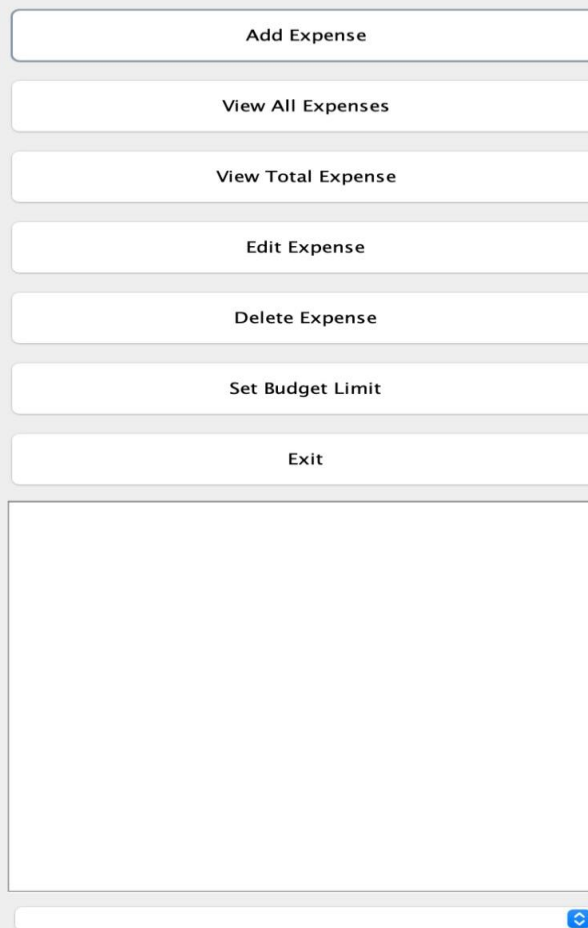
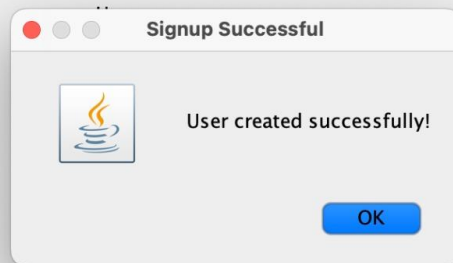
Username:

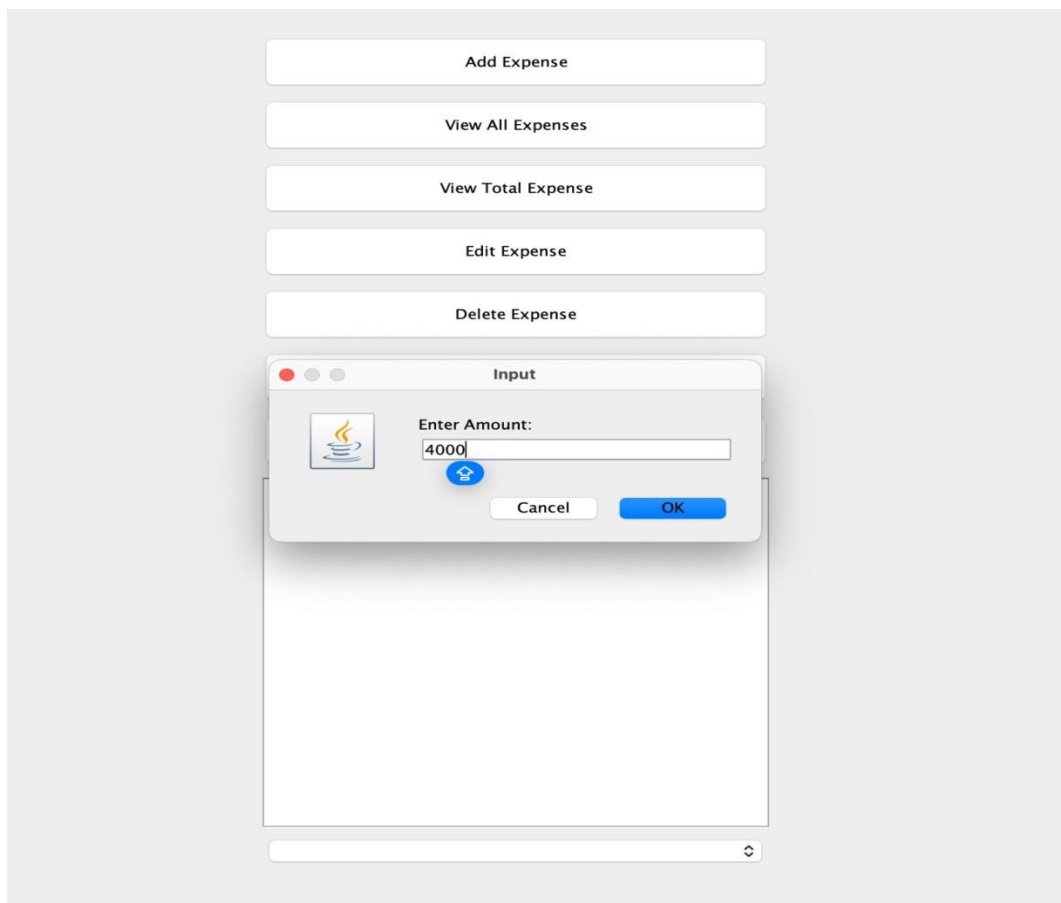
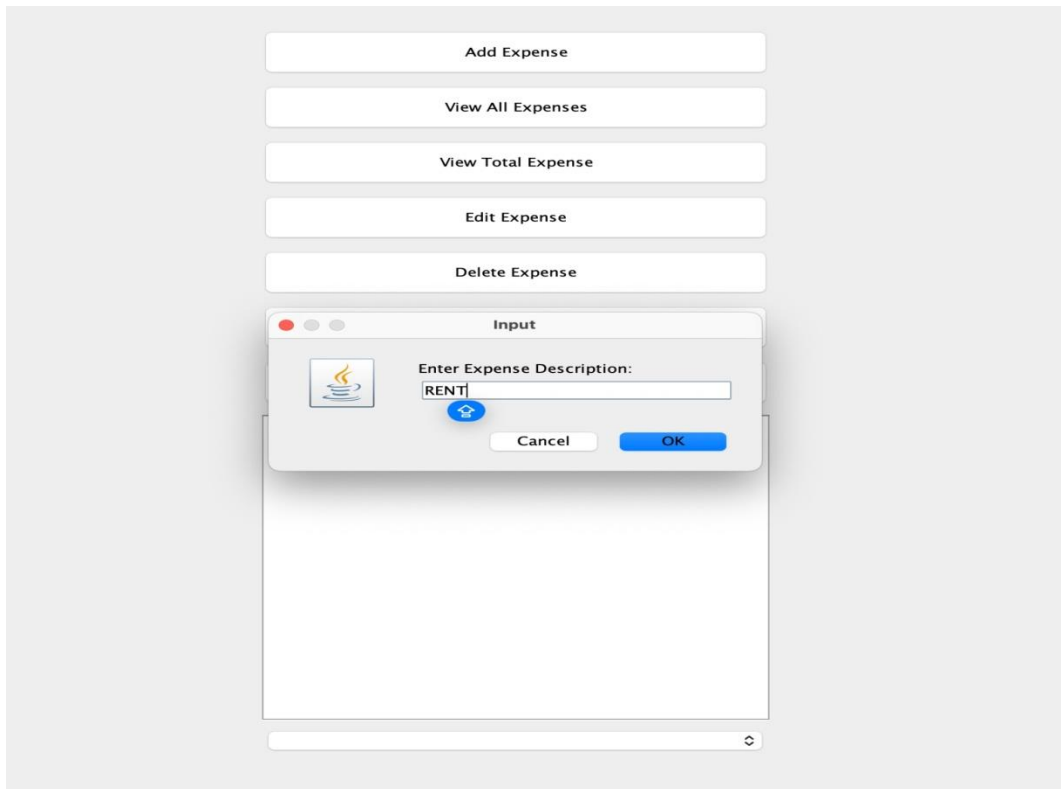
Password:

Confirm Password:

Signup

Back to Login





Add Expense

View All Expenses

View Total Expense

Edit Expense

Delete Expense

Set Budget Limit

Exit

RENT: Rs4000.00
CURRENT BILL: Rs500.00
GROCERRIES: Rs5000.00
WATER BILL: Rs1500.00

RENT: Rs4000.00

Add Expense

View All Expenses

View Total Expense


Edit Expense

Delete Expense

RENT: Rs
CURRENT
GROCERF
WATER BILL

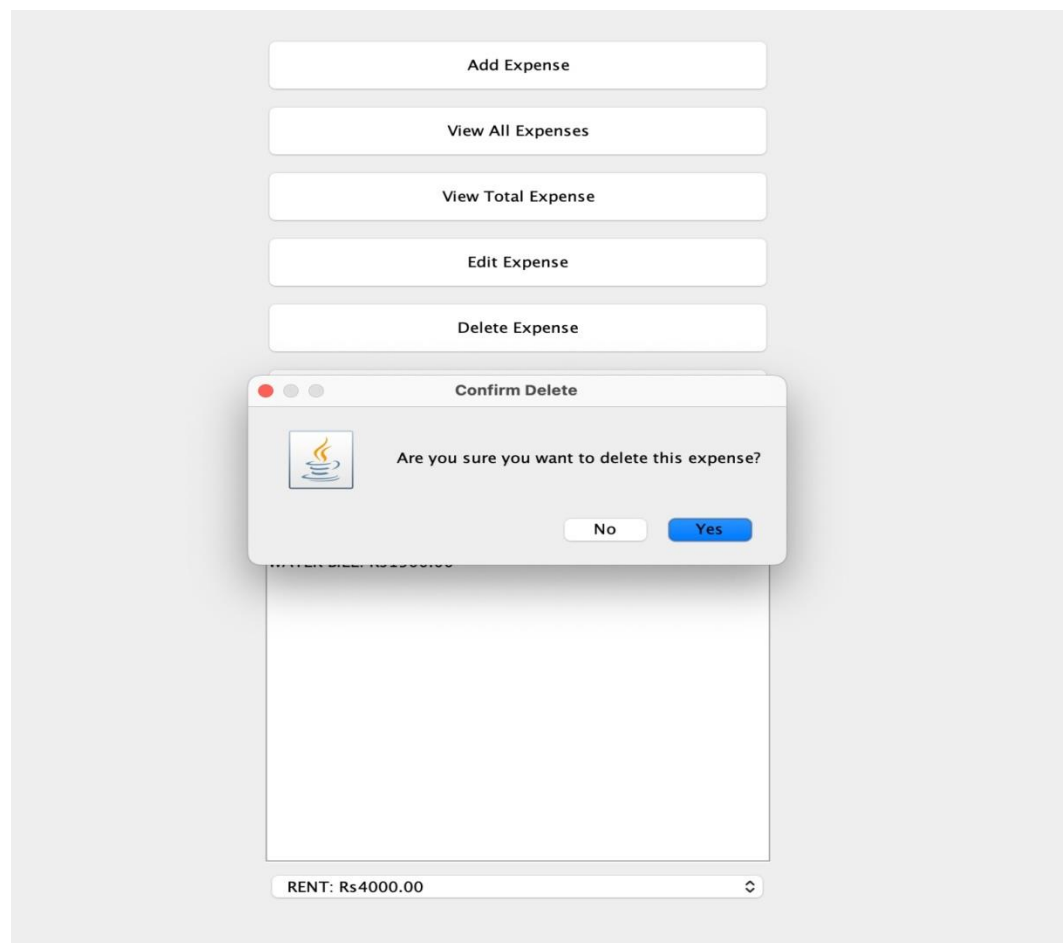
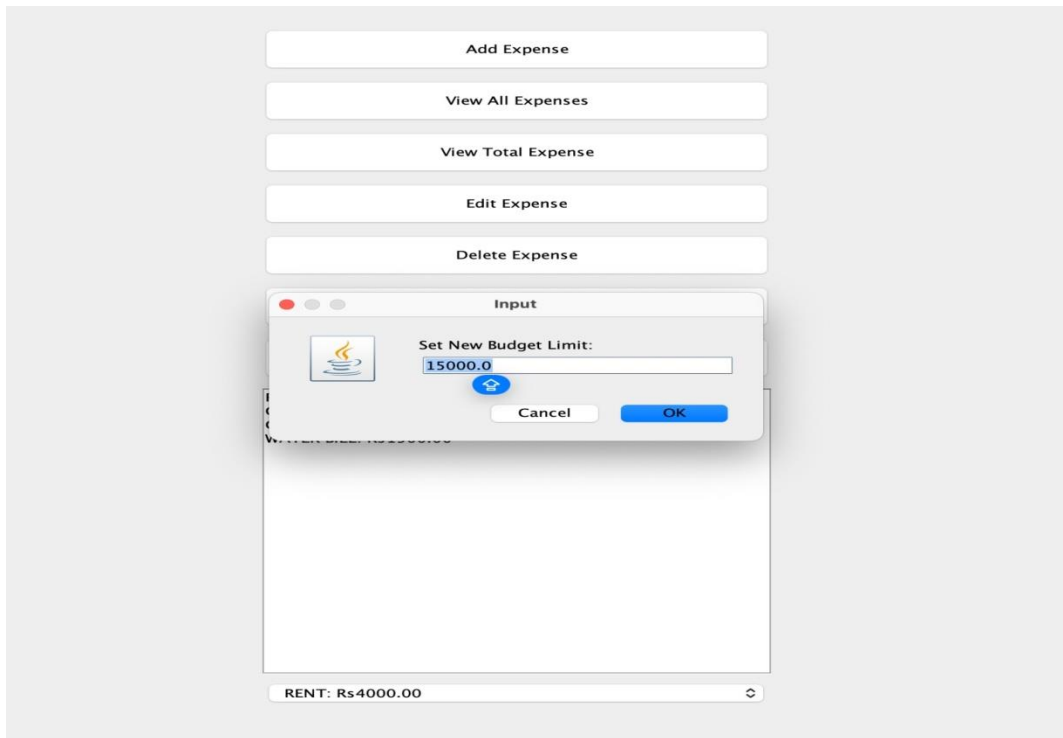
RENT: Rs4000.00

Warning



Budget Limit Exceeded!

OK



CHAPTER : 4

CONCLUSION

The Expense Tracker application stands as a testament to the potential of simplistic design combined with practical utility. By focusing on core functionalities that directly address the needs of its users, it successfully provides an intuitive platform for financial management. The application not only streamlines the process of tracking daily expenditures but also accentuates the significance of budgetary discipline. Its capacity to alert users when they are on the brink of surpassing their budget underscores its role as an active financial guide. In its current iteration, the Expense Tracker has proven to be an indispensable tool for those keen on maintaining a keen eye on their financial habits, fostering a sense of responsibility and awareness.

FUTURE SCOPE

Database integration: Future iterations may utilize a database system, allowing for long-term storage, retrieval, and backup of user data.

Analytics Tools: Introduces analytics tools that can give users insight into their spending habits, by providing visual representations such as Graphs and pie charts.

Cloud Sync: With cloud integration, users can access and manage their expenses from multiple devices, ensuring flexibility and convenience.

Mobile app version: The mobile version of Expense Tracker will provide on-the-go access, improving usability.

The future of Expense Tracker is full of possibilities, with each innovation paving the way for a more comprehensive and integrated financial management experience.

REFERENCES

“44% of World Population will Own Smartphones in 2017.” [Online].

Available: <https://www.strategyanalytics.com/strategyanalytics/blogs/smart-phones/2016/12/21/44-of-worldpopulation- will-own-smartphones- in2017#.Wq1BcehuY2y>.

[Accessed: 21-Oct-2020]. “Daily Expense 3 - Apps on GooglePlay.” [Online].

Available: <https://play.google.com/store/apps/details?id=mic.app.gas.tosdiarios>.

[Accessed: 21-Oct-2020]. “Monefy - Money Manager - Apps on Google Play.” [Online].

Available: <https://play.google.com/store/apps/details?id=com.monefy.app.lite>.

[Accessed: 21-Oct-2020]. From the figure 9 we can observe that the success rate of the “Money Lover: Budget Planner, Expense Tracker - Apps on Google Play.” [Online].

Available: <https://play.google.com/store/apps/details?id=com.bookmark.money>

[Accessed: 21-Oct-2020]. “AndroMoney (Expense Track) - Apps on Google Play.” [Online].

Available: <https://play.google.com/store/apps/details?id=com.kpmon ey.android> .

[Accessed: 21-Oct-2020]. “Optical Character Recognition (OCR) – How it works.” [Online].

Available: <https://www.nicomsoft.com/optical-character->

Sites Referred: <http://java.sun.com> <http://www.sourceforge.de.com>

<http://www.networkcomputing.com/> <http://www.roseindia.com/>