**ONLINE BANKING SYSTEM**

UCS2313 – Object-Oriented Programming Laboratory

A PROJECT REPORT

Submitted By

Thirumurugan RA - 3122 23 5001 149

Varun Prakash - 3122 23 5001 154

Vishal Muralidharan - 3122 23 5001 162

A blue and white logo

Description automatically generated

Department of Computer Science and Engineering

Sri Sivasubramaniya Nadar College of Engineering

(An Autonomous Institution, Affiliated to Anna University)

Kalavakkam – 603110

November 2024

TABLE OF CONTENTS

[Abstract: 3](#_Toc181732404)

[Acknowledgements: 4](#_Toc181732405)

[1. Problem Statement: 5](#_Toc181732406)

[2. Motivation for the Problem: 5](#_Toc181732407)

[3. Scope and Limitations: 6](#_Toc181732408)

[4. Design of the Solution (Class Diagram): 7](#_Toc181732409)

[5. Modules Split Up: 11](#_Toc181732410)

[6. Implementation Specifics: 13](#_Toc181732411)

[7. Output Screenshots: 16](#_Toc181732412)

[8. Object Oriented Features Used 38](#_Toc181732413)

[9. Database Schema 40](#_Toc181732414)

[10. Inference and Future Extension 43](#_Toc181732415)

[11. References: 44](#_Toc181732416)

# **Abstract:**

The "Online Banking System" project implements a simplified online banking platform using Java and JDBC. It provides a foundation for customers to perform common banking operations, including balance checks, deposits, withdrawals, and fund transfers.

Accountants have access to administrative functionalities for managing customer accounts, such as adding new customers, editing account information, deleting accounts, and viewing customer details and transaction histories.

Basic fixed deposit and loan account management features are also included, enabling accountants to create and oversee these accounts for customers.

While the current implementation offers core banking functionalities, it has limitations regarding advanced features, robust security measures, and user interface complexity. Future development will focus on enhancing these areas to deliver a more comprehensive and secure online banking experience.

# **Acknowledgements:**

We would like to thank God for being able to complete this Project Work with success.

We are thankful to and fortunate enough to get constant encouragement, support, and guidance from our project guide Dr. S. Rajalakshmi, B.E., M.E., Ph.D., Assistant Professor, Computer Science and Engineering Department, who helped us in successfully completing our Project Work.

In the meantime, we would also like to thank our Parents who gave us this wonderful opportunity to be in SSN.

We respect and thank our management for providing us with such an opportunity to do the project work in SSN College of Engineering and giving us all support and facilities, which helped us complete the project work.

We take this opportunity to express our profound gratitude and deep regards to Dr. T.T Mirnalinee M.E., Ph.D., HoD of Computer Science and Engineering Department for her exemplary guidance, monitoring, and constant encouragement throughout the course of this project.

# **1. Problem Statement:**

The project tackles the inefficiencies and limitations of traditional, in-person banking by developing a simplified online banking system.

Traditional banking often requires customers to physically visit a branch during limited business hours to perform transactions, access account information, and interact with bank personnel.

This can be inconvenient, time-consuming, and prone to errors. Moreover, managing customer accounts and transactions manually can be a complex and resource-intensive process for the bank.

This project aims to alleviate these issues by providing an online platform that allows customers to manage their accounts and perform transactions remotely, while also offering accountants a centralized system for efficient account administration.

# **2. Motivation for the Problem:**

The driving force behind this project is the need for a more accessible, efficient, and user-friendly banking experience. The limitations of traditional banking, such as restricted access and manual processes, create a demand for a modernized solution. This project is motivated by several factors:

* **Enhanced Customer Convenience:** Customers can access their accounts and perform transactions 24/7, eliminating the need for branch visits and aligning with the increasing demand for on-demand services.
* **Improved Operational Efficiency:** Automating transactions reduces manual processing time for both customers and bank staff, minimizing errors and freeing up resources for other tasks.
* **Centralized Account Management:** The system provides accountants with a centralized platform to manage customer accounts, transactions, and related functionalities, streamlining administrative tasks and improving oversight.
* **Increased Accessibility:** Online banking removes geographical barriers, allowing customers to access their accounts from anywhere with an internet connection, promoting financial inclusion and catering to a wider user base.

# **3. Scope and Limitations:**

The system covers essential banking operations, including customer login, account balance viewing, deposits, withdrawals, transfers, and transaction history.

Accountants can add, edit, remove, and view customer accounts. The system also supports basic fixed deposit and loan account management.

However, it lacks advanced features like checkbook requests, online bill payment and UPI payments.

Security measures could be strengthened with password hashing and more robust input validation.

The UI, based on JOptionPane, is functional but limited in its interactivity.

The interest calculation for all types (Savings, Loan and Fixed Deposit) is simplified and doesn't account for compounding frequency.

Loan management lacks features like amortization schedules and penalty handling.

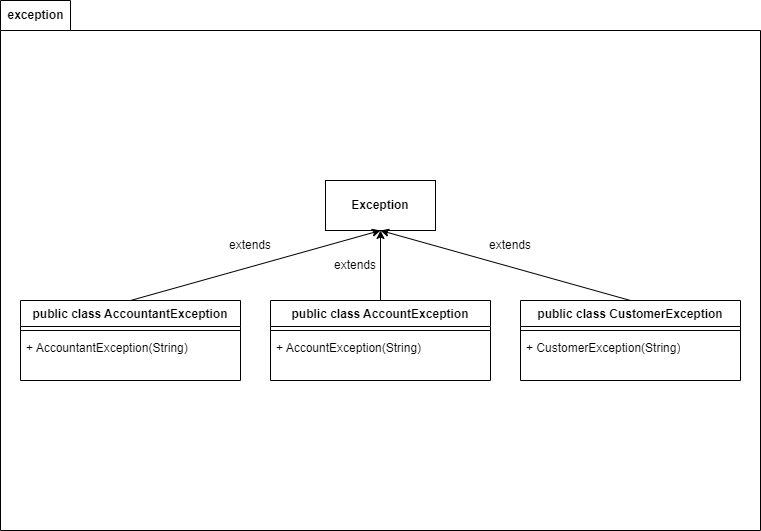
# **4. Design of the Solution (Class Diagram):**

A close-up of a document

Description automatically generated

A close-up of a document

Description automatically generated



A screenshot of a computer

Description automatically generated

A white paper with black text

Description automatically generated

# **5. Modules Split Up:**

The project follows a modular design, separating concerns into distinct packages:

* *com.obs*: The root package for the project.
* *com.obs.bean*: Contains the data model classes (beans) representing entities like Customer, Accountant, Transaction, LoanAccount, and FixedDepositAccount.
* *com.obs.dao*: Houses the Data Access Objects (DAOs), including interfaces (AccountantDAO, CustomerDAO) and their implementations (AccountantDAOimpl, CustomerDAOimpl). These DAOs interact with the database through DBUtil.
* *com.obs.exception*: Contains custom exception classes for handling specific error scenarios within the application.
* *com.obs.mainwork*: Contains the main application class (MainGUI\_Desktop), database initialization script (Create\_DB), and accountant creation script (Create\_Accountants).
* *com.obs.utility*: This package now explicitly contains DBUtil.java, which is responsible for providing database connections and managing database-related operations.

**Package Diagram:**

com

└── obs

├── bean

│ ├── AccountantBean

│ ├── CustomerBean

│ ├── FixedDepositAccountBean

│ ├── LoanAccountBean

│ └── TransactionBean

├── dao

│ ├── AccountantDAO

│ ├── AccountantDAOimpl

│ ├── CustomerDAO

│ └── CustomerDAOimpl

├── exception

│ ├── AccountException

│ ├── AccountantException

│ └── CustomerException

├── mainwork

│ ├── Create\_Accountants

│ ├── Create\_DB

│ └── MainGUI\_Desktop

└── utility

└── DBUtil

com.obs.dao --> com.obs.bean

com.obs.dao --> com.obs.exception

com.obs.mainwork --> com.obs.bean

com.obs.mainwork --> com.obs.dao

com.obs.mainwork --> com.obs.exception

com.obs.mainwork --> com.obs.utility

The --> indicates a dependency relationship, meaning the package on the left depends on the package on the right.

# **6. Implementation Specifics:**

**1. Programming Language: Java**

Java is used as the primary programming language for the project. This leverages Java's platform independence, object-oriented features, and rich ecosystem of libraries for database connectivity, UI development, and exception handling.

**2. Database: MySQL**

MySQL is the chosen database system. MySQL is a popular open-source relational database management system (RDBMS) well-suited for this type of application.

**3. Database Connectivity: JDBC (Java Database Connectivity)**

The project uses JDBC for interacting with the MySQL database. JDBC provides a standard Java API for connecting to and interacting with relational databases.

The DBUtil class (in com.obs.utility) encapsulates the logic for establishing database connections, executing queries, and handling result sets.

The DAO implementation classes (AccountantDAOimpl and CustomerDAOimpl) use JDBC to perform database operations.

**4. User Interface: Swing (JOptionPane)**

The user interface is implemented using Swing, specifically employing JOptionPane for creating simple dialog boxes for user input and displaying messages.

**5. Data Model: Java Beans**

The project uses Java Beans (Plain Old Java Objects or POJOs) to represent data entities. Classes like CustomerBean, AccountantBean, TransactionBean, LoanAccountBean, and FixedDepositAccountBean encapsulate data related to customers, accountants, transactions, loans, and fixed deposits, respectively. These beans provide a structured way to store and manage data within the application.

**6. Data Access Layer: Data Access Objects (DAOs)**

The project employs the DAO pattern to separate database access logic from the rest of the application. Interfaces like AccountantDAO and CustomerDAO define the methods for interacting with the database, while their implementations (AccountantDAOimpl and CustomerDAOimpl) provide the concrete database interaction logic using JDBC.

**7. Exception Handling: Custom Exceptions**

Custom exception classes (AccountException, CustomerException, AccountantException) are used to handle specific error scenarios related to accounts, customers, and accountants. This approach allows for more granular error handling and reporting.

**8. Interest Calculation Logic: InterestCalculator**

The interest calculation system is implemented through an InterestCalculator interface that defines the calculateInterest method taking principal amount and time in months as parameters. The SavingsInterestCalculator offers a straightforward 4% annual interest rate for savings accounts. The LoanInterestCalculator implements a tiered interest rate structure depending on the Loan term. The FixedDepositInterestCalculator also uses a tiered structure based on Deposit duration.

**9. Application Logic: MainGUI\_Desktop**

The MainGUI\_Desktop class in the com.obs.mainwork package serves as the main application class, handling the user interface logic, user input, and interaction with the DAO classes.

**10. Database and Accountant Initialization:**

The Create\_DB and Create\_Accountants classes in com.obs.mainwork provide utility functions for initializing the database schema and creating default accountant accounts, respectively.

# **7. Output Screenshots:**

1. **A screenshot of a computer

   Description automatically generatedWelcome Screen**

1. A screenshot of a computer

   Description automatically generated**Accountant Portal**
2. **Accountant Operations**A screenshot of a computer

   Description automatically generated
3. **Add new customer account**

**A screenshot of a computer

Description automatically generated**

A screenshot of a account

Description automatically generated

1. **Edit Existing Account**

**A screenshot of a computer

Description automatically generated**

1. **A screenshot of a computer

   Description automatically generatedRemove Account**

**A screenshot of a computer

Description automatically generated**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. **View Account Details**

A screenshot of a computer

Description automatically generated

1. A screenshot of a computer

   Description automatically generated**View All Account Details**
2. **A screenshot of a computer

   Description automatically generatedView Customer Transactions**
3. A screenshot of a computer

   Description automatically generated**A screenshot of a computer

   Description automatically generatedAdd New Account for existing customer**

A screenshot of a computer error

Description automatically generated**A screenshot of a computer

Description automatically generated**

1. A screenshot of a computer

   Description automatically generated**Manage Fixed Deposit Accounts**
2. **A screenshot of a computer screen

   Description automatically generatedCreate Fixed Deposit**

**A screenshot of a computer error

Description automatically generated**

1. **A screenshot of a computer

   Description automatically generatedView All Fixed Deposit of a single customer**
2. **A screenshot of a computer

   Description automatically generatedView Fixed Deposit Account Details**

**A screenshot of a computer

Description automatically generated**

1. **A screenshot of a computer screen

   Description automatically generatedClose fixed deposit**

A close-up of a computer screen

Description automatically generated

1. **Manage Loan Accounts**

A screenshot of a computer

Description automatically generated

1. A screenshot of a computer

   Description automatically generated**Manage Loan Accounts**

A screenshot of a computer

Description automatically generated

1. A screenshot of a computer

   Description automatically generated **View all Loan Accounts of a customer**
2. **View Loan details**

A screenshot of a computer

Description automatically generated**A screenshot of a computer screen

Description automatically generated**

1. **Update Loan Payment**

**A screenshot of a computer error

Description automatically generated**

**A close-up of a screen

Description automatically generated**

**A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated**

1. A screenshot of a computer screen

   Description automatically generated**A screenshot of a computer

   Description automatically generatedUpdate Loan Payment**

**A screenshot of a computer error

Description automatically generated**

**A screenshot of a computer error

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer error

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

1. A screenshot of a computer

   Description automatically generated**Customer Portal Login**
2. **A screenshot of a computer

   Description automatically generated Customer Operations**
3. **Forgot Password**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer error

Description automatically generatedA screenshot of a computer login box

Description automatically generated**

1. **A screenshot of a computer error

   Description automatically generatedView Balance**
2. **Deposit Money**

**A screenshot of a computer screen

Description automatically generated**

A screenshot of a computer error

Description automatically generated

1. **Withdraw Money**

**A screenshot of a computer error

Description automatically generatedA screenshot of a computer error

Description automatically generated**

1. **A screenshot of a computer screen

   Description automatically generatedTransfer Money**

**A screenshot of a computer error

Description automatically generated**

1. **A screenshot of a computer

   Description automatically generatedView Transaction History**
2. **A screenshot of a computer

   Description automatically generatedView Fixed Deposit Accounts**
3. **View Loan Accounts**

**A screenshot of a computer error

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

# **8. Object Oriented Features Used**

1. Data Encapsulation:

The project effectively uses classes to represent key entities within the banking system. Classes like CustomerBean, AccountantBean, TransactionBean, LoanAccountBean, and FixedDepositAccountBean act as blueprints for creating objects that encapsulate data related to these entities. This allows for structured data representation and manipulation.

2. Data Abstraction:

Encapsulation is employed using private instance variables (fields) within the bean classes. Access to these variables is controlled through public getter and setter methods. This protects data integrity and promotes information hiding, which is a key principle of object-oriented programming.

3. Inheritance:

The DAO (Data Access Object) layer demonstrates the use of interfaces and their implementations. Interfaces like AccountantDAO and CustomerDAO define the contracts for database interaction, while classes like AccountantDAOimpl and CustomerDAOimpl provide the concrete implementations of these interfaces.

Interface like Interest Calculator is used to define the calculation contracts. Classes like FixedDepositInterestCalculator, LoanInterestCalculator, SavingsInterestCalculator provide the concrete implementations of these the calculateInterest method.

Inheritance is also used for exception handling. The custom exception classes (AccountException, CustomerException, AccountantException) inherit from the base Exception class. This allows for specialized exception handling within the application.

This separation of interface and implementation promotes abstraction and allows for flexibility in switching database implementations without affecting other parts of the application.

4. Polymorphism:

In this Java implementation, different types of interest calculators—`FixedDepositInterestCalculator`, `LoanInterestCalculator`, and `SavingsInterestCalculator`—each provide their own calculation logic for interest based on specific criteria.

By implementing the `InterestCalculator` interface, each calculator class overrides the `calculateInterest` method, enabling polymorphism.

The `FixedDepositInterestCalculator` adjusts interest rates based on the deposit duration, with higher rates for shorter periods and decreasing rates as the period increases.

The `LoanInterestCalculator` varies rates according to loan terms, with higher rates for shorter loan terms and lower rates for longer terms.

Lastly, the `SavingsInterestCalculator` applies a fixed annual rate of 4%, reflecting typical savings account interest. This setup allows for easy extension and modification of interest calculation logic while adhering to a uniform interface.

5. Packages

The code has a logical grouping of classes using packages such as com.obs.bean, com.obs.dao, and com.obs.exception, each with a distinct responsibility. This package structure ensures a clear separation of concerns, promoting a modular architecture that keeps different parts of the application organized and easy to navigate.

6. Collections

The codebase effectively utilizes collections to manage and organize different calculation strategies, particularly in the AccountantDAOimpl class, where a HashMap named interestCalculators maps account types like "SAVINGS," "FIXED\_DEPOSIT," and "LOAN" to their respective interest calculator instances. This collection-based approach enables quick lookups for the appropriate calculator, allowing runtime flexibility in selecting the correct interest calculation method based on account type.

For displaying account and transaction information, the DAO layer uses ResultSet collections from database queries to handle tasks such as retrieving account details, displaying transaction history, managing balance inquiries, and accessing customer information. The system accommodates multiple account types, including savings accounts with a fixed 4% interest rate, fixed deposit accounts with tiered rates based on deposit duration, loan accounts with rates that vary by loan term, and current accounts with overdraft facilities. This structure facilitates efficient retrieval and display of account details, interest calculations across different types, transaction history tracking, and streamlined management of diverse account offerings.

7. Generics (Generic Method)

The MainGUI\_Desktop.java uses a generic method displayAccountDetails<T> that efficiently handles the display of different account types (CustomerBean, LoanAccountBean, and FixedDepositAccountBean) using a single method implementation. This method takes two parameters: a generic type T representing the account object and a String title for the display window. The method uses type checking with instanceof to determine the specific account type and formats the relevant details, accordingly, making it highly reusable and maintainable while reducing code duplication. The formatted details are then displayed in a standardized way through a JOptionPane dialog, providing a consistent user experience regardless of the account type being viewed.

# **9. Database Schema**

A screenshot of a computer

Description automatically generated

1. **Account Table**

A screen shot of a computer

Description automatically generated

1. **Fixed Deposit Account Table**

**A screen shot of a computer

Description automatically generated**

1. **A screen shot of a computer

   Description automatically generatedLoan Account Table**
2. **A black screen with white text

   Description automatically generatedInfoaccountant Table**
3. **Infocustomer Table**

**A screenshot of a computer screen

Description automatically generated**

1. **Transaction table**

**A screen shot of a computer program

Description automatically generated**

# **10. Inference and Future Extension**

Future extensions for this project could include:

* Security Enhancements: Implement stronger password hashing algorithms and input validation to prevent vulnerabilities.
* User Interface: Develop a user interface (e.g., using JavaFX, or a web framework) to improve user interface and look thereby improving the user interaction.
* Reporting and Analytics: Generate reports on account activity, customer demographics, and overall bank performance.
* Integration with External Systems: Integrate with payment gateways, credit bureaus, or other financial services.
* Unit Testing: Implement comprehensive unit tests to ensure code quality and prevent regressions.

# **11. References:**

For Making Class Diagrams:

<https://app.diagrams.net/>

Oracle's Swing Tutorial:

<https://docs.oracle.com/javase/tutorial/uiswing/index.html>

Java Swing Tutorial:

<https://www.geeksforgeeks.org/introduction-to-java-swing/>

JDBC Tutorial:

<https://www.geeksforgeeks.org/jdbc-tutorial/>

MySQL Documentation:

[https://dev.mysql.com/doc/](https://dev.mysql.com/doc/%20)

Java Documentation:

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

Stack Overflow:

<https://stackoverflow.com/>