**REPORT – CSD\_3464**

Group – PB\_SOFTWARE\_DEVELOPERS

Group Members:-

* Sukhdev Singh (C0895314)
* Gunraj Singh (C0893229)
* Deepankshudeep Singh (C08932965)
* Maneet Pal Singh (C893295)

**Motivation and Goals**: -

The project aims to create a basic Banking Management System using Java. We were really fascinated by our day-to-day banking transactions and how much we could do with it. We wanted to recreate something like that maybe not to the professional level but to a degree that could be achieved by us as a team at the same time keeping all the feature that are usually provided in a banking module found in real life. The motivations include streamlining and automating various banking processes, providing an efficient and secure platform for customers, enhancing customer experience, and improving overall efficiency.

**Detailed Description of the Design and Implementation:**

The code is organized into several classes, each responsible for specific aspects of the banking system.

* **Person Class**: Represents a basic person with a name. It serves as the parent class for AccountHolder.
* **AccountHolder Class:** Extends the Person class and encapsulates account-related information. Manages account details, PINs, balances, and transaction history. Provides methods for fund transfers, withdrawals, and creating fixed deposits.
* **Transaction Class**: Represents a transaction with details such as type, amount, and date. Transaction objects are stored in the AccountHolder class to maintain transaction history.
* **FixedDeposit Class:** Models fixed deposit accounts with attributes like account number, principal, interest rate, and maturity details. Calculates the maturity amount based on fixed deposit parameters.
* **Bank Class:** Serves as the main controller for the banking system. Manages a list of AccountHolder objects and facilitates account creation, login, and various banking operations. Utilizes file I/O to save and load data, maintaining a basic persistence layer.

**User Interaction**: The system employs a simple command-line interface (CLI) for user interaction.Users navigate through menus in the mainMenu method, choosing options like account creation, login, and various transactions.

Some of the Key Features:

* Account Management: Users can create new accounts with unique account numbers and PINs. Account details, including transactions, are stored and can be accessed.
* Transaction Processing: Enables fund transfers between accounts, withdrawals, and deposit operations. Maintains a transaction history for each account.
* Fixed Deposit Management: Allows users to create fixed deposits with different durations and interest rates. Supports breaking fixed deposits with appropriate calculations.

**Usage Scenarios: User Profiles, Use-Case Diagram:**

* User Profiles: Customers who can create accounts, log in, perform transactions, create fixed deposits, and view account information.
* Use-Case Diagram: The system's main functionalities include creating accounts, logging in, transferring money, withdrawing, creating fixed deposits, breaking fixed deposits, and viewing account information.

**Data Model: Database Schema:**

The data model involves classes like AccountHolder, Transaction, and FixedDeposit. Data is stored in ArrayLists and written to a file (BankRecord.txt).

**System Components and Functions: Activity Diagram, Sequence Diagram, etc :**

The system components include classes for Person, Transaction, FixedDeposit, and Bank.

Functions involve account management, transactions, fixed deposits, and file I/O.

Activity Diagram: Represents user actions and system responses during processes like login, transactions, and fixed deposit creation.

**User Interface Design:**

The user interface is primarily command-line based, with menu-driven interactions. Users can choose options from the main menu and perform various banking operations.

**System Implementation Updates and Outstanding Issues:**

The code does not mention any specific updates or issues. It could benefit from error handling improvements, code organization enhancements, and potential refinements based on user feedback.

**Results of a System Evaluation:**

An evaluation of the system could involve testing scenarios, assessing the accuracy of financial calculations, checking for error handling, and ensuring the security of PINs. The code appears to work as intended, but further testing and refinement may be needed for a production-level system.

**What Did You Learn?**

1. Java Programming: The code demonstrates proficiency in Java programming, including object-oriented concepts such as classes, inheritance, and encapsulation.
2. Object-Oriented Design: The use of classes like AccountHolder, Transaction, and FixedDeposit showcases an understanding of object-oriented design principles to model real-world entities.
3. File Input/Output: The implementation of file I/O operations for saving and loading data (BankRecord.txt) highlights skills in handling external data storage.
4. Financial Calculations: The code includes financial calculations, such as calculating maturity amounts for fixed deposits, showcasing an understanding of financial concepts.
5. User Input Validation: The validation checks for user input, such as verifying the length of the account number, demonstrate a commitment to ensuring data integrity and system reliability.
6. Error Handling: Although there's room for improvement, the code addresses some error scenarios, such as incorrect PINs during login. Further enhancements in error handling can lead to a more robust system.
7. Command-Line Interface (CLI): The command-line-based user interface illustrates a straightforward way to interact with the system. Learning to design user interfaces, even in a simple CLI form, is foundational for more complex UI development.
8. Testing and Evaluation: The code encourages the practice of testing scenarios, evaluating system behaviors, and refining the implementation based on testing results. System evaluation is crucial for ensuring correctness and reliability.
9. Project Documentation: Although not explicitly mentioned in the code, proper documentation is a crucial aspect of software development. Documenting motivations, goals, and implementation details is essential for team collaboration and future maintenance.
10. Security Considerations: While not extensively covered in the code, dealing with sensitive information like PINs prompts consideration of security measures. Learning to implement secure coding practices is imperative for applications handling financial data.
11. Continuous Improvement: Recognizing that the code could benefit from enhancements, such as code organization and user interface improvements, emphasizes the importance of continuous learning and refinement in software development.