README – OOPBankManager

**Project Overview**

This project is implemented in Java and simulates a banking system for managing various account types. The system includes functionalities such as account creation, profit calculations, management fees, and customer handling. The code is built using Object-Oriented Programming (OOP) principles and adheres to SOLID principles, ensuring modular, readable, and extendable code.

**Project Description**

The system simulates a bank managing diverse account types (e.g., Savings, Business Checking, Mortgages). It provides a menu interface for performing operations such as creating new accounts, adding clients to existing accounts, calculating profits, and checking VIP status for Business Checking accounts. The system also includes robust error handling through custom exceptions for duplicate account numbers and invalid menu choices.

**Program Structure**

The project is designed such that each class represents an independent component of the system. Each account type is implemented in a separate class (e.g., BusinessCheckingAccount, SavingsAccount, MortgageAccount), with each class encapsulating its unique features. The user interface is handled by the Main class, which presents a menu for managing the system.

**Programming Principles**

The project implements the following SOLID principles:

* **Single Responsibility Principle**: Each class is responsible for a single functionality. For example, the BusinessCheckingAccount class handles only the management of business checking accounts.
* **Open/Closed Principle**: Classes are open for extension but closed for modification, allowing new account types to be added without altering existing code.
* **Liskov Substitution Principle**: Derived classes are substitutable for their base class Account, ensuring seamless interchangeability in the code.
* **Interface Segregation Principle**: Interfaces like ManagementFeeAccount are used for accounts requiring fee management, avoiding unnecessary method implementation in unrelated classes.
* **Dependency Inversion Principle**: The AccountsFactory class is used to create objects, reducing dependency on specific classes.

**Key Classes and Features**

**Core Classes:**

1. **Account**: The base class representing a general bank account, with attributes such as account number, bank number, balance, manager name, and opening date. Includes basic methods like deposit, withdrawal, account type retrieval, and profit calculation, which can be extended in derived classes.
2. **RegularCheckingAccount**: A subclass representing regular checking accounts with features like credit limit management.
3. **BusinessCheckingAccount**: Extends CheckingAccount and includes attributes such as business revenue and client rank, as well as VIP profit handling.
4. **SavingsAccount**: Represents savings accounts with features like deposit amount and duration.
5. **MortgageAccount**: Represents mortgage accounts, including attributes like the original mortgage amount, monthly payment, and loan term. Calculates interest, monthly payments, and annual management fees.
6. **Client**: Represents bank clients with attributes such as name and rank, used for customized profit calculations.
7. **Bank**: The central class managing accounts and operations, including account creation, client registration, profit calculation, and displaying account details.
8. **AccountsFactory**: A factory class for automatically creating accounts with predefined data, reducing dependencies on specific classes.
9. **ManagementFeeAccount**: An interface ensuring that implementing classes handle management fee-related methods.

**Custom Exceptions:**

* **DuplicationException**: Handles duplicate account numbers.
* **InvalidChoiceException**: Handles invalid menu choices.

**Key Methods and Functionalities**

* **createInitialAccounts** (Bank): Creates predefined initial accounts using the AccountsFactory.
* **addAccount** (Bank): Adds a new account to the system while ensuring unique account numbers.
* **getAllAccounts** (Bank): Displays all accounts in the system, including associated clients.
* **checkProfitVIP** (BusinessCheckingAccount): Calculates VIP profit for eligible business accounts.
* **printManagementFees** (Bank): Displays management fees for applicable accounts and calculates the CEO's annual bonus.
* **registerClientToAccount** (Bank): Registers a new client to an existing account, ensuring no duplicate client data for the same account.
* **getProfitAccounts** (Bank): Displays accounts with annual profits in descending order.
* **getAccountProfit** (Bank): Calculates and displays the annual profit for a specific account.
* **getTopCheckingAccountByProfit** (Bank): Finds and displays the most profitable regular checking account.
* **calculateProfit** (Account classes): Calculates annual profits based on account-specific conditions.
* **addClientToAccount** (Main): Allows users to add clients to existing accounts with input validation.
* **getValidIntInput / getValidDoubleInput** (Main): Ensures valid user input based on data type and value range.
* **showMenu** (Main): Displays the system menu with options for users to perform various operations.
* **toString** (Account classes): Provides a user-friendly string representation of account details, including type, balance, opening date, and unique attributes.

**Usage Instructions**

**Running the System**

Run the Main.java file to start the system. A menu will appear, displaying all available options.

**Menu Structure**

The main menu includes the following options:

1. **Create Initial Accounts**: Automatically generate predefined accounts using AccountsFactory.
2. **Add a New Account**: Manually create a new account (e.g., Regular Checking, Business Checking, Savings, or Mortgage).
3. **Add a Client**: Register a client to an existing account.
4. **View All Accounts**: Display all accounts with relevant details.
5. **Profit Accounts**: List accounts with annual profits, sorted by profit in descending order.
6. **Filter Accounts by Type**: Display accounts of a specific type.
7. **Calculate Account Profit**: Compute and display the annual profit for a specific account.
8. **Total Annual Profit**: Display the bank's total annual profit.
9. **Top Profit Account**: Identify the most profitable regular checking account.
10. **VIP Profit Check**: Check VIP profit status for business accounts.
11. **Management Fees**: Display management fees for applicable accounts and calculate the CEO's annual bonus.

**Input Validation**

The system ensures input correctness:

* Numeric inputs (e.g., account numbers or deposit amounts) must be valid.
* Invalid inputs trigger error messages and prompt for re-entry.

**Error Handling**

Custom exceptions like DuplicationException (duplicate account numbers) and InvalidChoiceException (invalid menu choices) are implemented to handle errors effectively.

**Future Expandability**

The system is designed for scalability. New account types or functionalities can be added by introducing new classes and methods without altering the existing structure.