**Banking System Case Study**

**Objective:**

Build a basic banking system that manages different types of accounts, supports transactions, maintains customer information, and simulates transaction processing with multithreading. The system will demonstrate:

* Class design & inheritance
* Encapsulation
* Collections (ArrayList for accounts and transactions)
* Exception handling (custom exceptions)
* Threading (simulate transaction processing)
* Use of static variables/methods

**Class Design with Properties & Methods**

**1. Abstract Class: Account**

Base class representing a generic bank account.

**Properties:**

* int accountNumber — Unique account number
* String accountHolderName — Name of the account holder
* double balance — Current account balance
* static int accountCounter — Static counter to generate account numbers

**Methods:**

* abstract void deposit(double amount)
* abstract void withdraw(double amount) throws InsufficientFundsException
* void displayAccountDetails()
* int getAccountNumber()
* String getAccountHolderName(), void setAccountHolderName(String name)
* double getBalance()

**2. Concrete Classes: (Inheritance from Account)**

**a) SavingsAccount**

Savings account with interest rate.

**Properties:**

* double interestRate

**Methods:**

* void deposit(double amount) — Adds amount to balance
* void withdraw(double amount) — Subtracts amount if balance sufficient
* void addInterest() — Adds interest to balance
* Override displayAccountDetails() to show interest rate

**b) CurrentAccount**

Current account with overdraft limit.

**Properties:**

* double overdraftLimit

**Methods:**

* void deposit(double amount)
* void withdraw(double amount) — Allows overdraft up to limit
* Override displayAccountDetails() to show overdraft info

**3. Class: Customer**

Represents a bank customer.

**Properties:**

* int customerId
* String name
* String email
* String phoneNumber
* ArrayList<Account> accounts — List of accounts owned by the customer

**Methods:**

* void addAccount(Account account)
* void removeAccount(int accountNumber)
* Account getAccount(int accountNumber)
* void displayCustomerDetails()

**4. Class: Transaction**

Represents a banking transaction.

**Properties:**

* int transactionId
* int accountNumber
* String type — "Deposit" or "Withdrawal"
* double amount
* Date transactionDate

**Methods:**

* void displayTransactionDetails()

**5. Class: Bank**

Manages customers and accounts.

**Properties:**

* ArrayList<Customer> customers
* ArrayList<Transaction> transactions

**Methods:**

* void addCustomer(Customer customer)
* Customer getCustomer(int customerId)
* void removeCustomer(int customerId)
* void addTransaction(Transaction transaction)
* List<Transaction> getTransactionsByAccount(int accountNumber)
* List<Account> getAccountsByCustomer(int customerId)

**6. Custom Exception: InsufficientFundsException**

Exception thrown when withdrawal amount exceeds balance or overdraft limit.

**Properties:**

* String message

**Methods:**

* Constructor with message parameter
* String getMessage()

**7. Thread Class: TransactionProcessor (extends Thread)**

Simulates processing of transactions asynchronously.

**Properties:**

* Transaction transaction
* Bank bank

**Methods:**

* void run() — Processes transaction, updates account balance, and logs status

**8. Utility Class: BankUtils**

Contains static helper methods.

**Methods:**

* static void displayAccounts(List<Account> accounts)
* static void displayTransactions(List<Transaction> transactions)
* static List<Account> sortAccountsByBalance(List<Account> accounts, boolean ascending)
* static List<Transaction> filterTransactionsByType(List<Transaction> transactions, String type)