Problem Statement: Design and Develop a Bank Server Application with Core Java and Collections Framework

Description: Develop a bank server application with the following functionalities:

1. Show Balance
2. Withdraw Money
3. Deposit Money
4. Transfer Funds (from one account to another)
5. Display Last 10 Transactions

Requirements:

1. Class Design: Create a class design before starting development. Define classes for Customer, Account, and Transaction.
2. Customer-Account Relationship: Each customer has only one account, and each account belongs to one customer.
3. Account Type: There is only one type of account available in the bank at the moment, which can be termed as a savings account.
4. Input/Output Management: Since the application will run on a server, input and output statements should be written only in the Main file. No other file/module should contain input/output statements.

Functionalities:

1. Show Balance: Display the current balance of the account.
2. Withdraw Money: Allow the customer to withdraw money from their account. Perform validations to ensure the balance does not go below zero.
3. Deposit Money: Allow the customer to deposit money into their account.
4. Transfer Funds: Enable the transfer of funds from one account to another. Perform validations to ensure sufficient balance.
5. Display Last 10 Transactions: Show the last 10 transactions, including transaction date, ID, amount, credit/debit status, available balance, and description.

Note: Implement validations wherever applicable, such as ensuring the balance of an account cannot go below zero.

---

1. Develop the application using Core Java and utilize the **Collections Framework** for managing transactions and accounts efficiently.

2. **Junit and Mockito**
   Construct the Bank Server Application integrating JUnit and Mockito for testing purposes. Include the Test cases for all 5 functionalities mentioned earlier.
3. **For JPA**
   Build the Bank Server Application mentioned above, incorporating JPA for Data Persistence and validations.
4. **For Spring MVC**
   Develop the Bank Server Application as described earlier, utilizing Spring MVC and integrating JPA for Data Persistence.

5. **For Spring Boot and Data JPA**
   Construct the Bank Server Application utilizing Spring Boot and Data JPA as mentioned earlier.
6. **For RestTemplate**
   Create the Bank Server Application using Spring Boot and Data JPA as previously mentioned and incorporate RestTemplate for handling HTTP requests.
7. **For RestAssured Testing**
   Develop the Bank Server Application as described earlier, employing Spring Boot and Data JPA, and integrate RestAssured for conducting API testing.
8. **For Open Feign**
   Implement the Bank Server Application using Spring Boot and Data JPA and integrate OpenFeign for making declarative RESTful web service calls.
9. **API Gateway**
   Develop the Bank Server Application utilizing Spring Boot and Data JPA and incorporate an API Gateway for routing and managing incoming requests to various microservices.