EXERCISE:1

BANK MANAGEMENT SYSTEM USING DESIGN PATTERNS

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Scanner;
```

// BEHAVIORAL DESIGN PATTERN

```
interface Customer {
   void update(String message);
   String getName(); // New method to get the customer's name
}

// Subject Interface
interface Subject {
   void AddCustomer(Customer cus);
   void removeCustomer(Customer cus);
   void notifyCustomer(String name, String message); // Notify specific customer
}

// Bank class implementing Subject
class Bank implements Subject {
   private List<Customer> customers = new ArrayList<>();
   private Map<String, Double> loan_amt = new HashMap<>(); // Store loan amounts

   @Override
   public void AddCustomer(Customer cus) {
      customers.add(cus);
   }

   @Override
   public void removeCustomer(Customer cus){
      customers.remove(cus);
      loan_amt.remove(cus.getName()); // Remove loan amount when customer is removed
   }

   @Override
   public void notifyCustomer(String name, String message) {
      for (Customer cus :  customers) {
         if (cus.getName().equalsIgnoreCase(name)) {
            cus.update(message);
            break;
            }
        }
    }

    public void update_Acc_Balance(String Holder, double new_bal) {
        String message = "Account balance updated for " + Holder + ": " + new_bal;
        notifyCustomer(Holder, message);
    }

    public void Loan_status(String Holder, double amt_paid) {
        double rem_amt = loan_amt.getOrDefault(Holder, 0.0) - amt_paid;
        loan_amt.put(Holder, rem_amt);
        String message = "Loan status updated for " + Holder + ": Remaining amount to pay is "
+ rem_amt;
        notifyCustomer(Holder, message);
    }

    public void Ori_loan(String Holder, double Amt) {
        loan_amt.put(Holder, Amt);
    }
}

// AccountHolder class implementing
class Holder implements Customer {
    private String name;

    public Holder(String name) {
        this.name = name;
    }

    @Override
    public void update(String message) {
        System.out.println(name + " received notification: " + message);
    }

    @Override
    public String getName() {
        return name;
    }
}
```

// STRUCTURAL DESIGN PATTERN

```
interface BankAccount {
    void deposit(double amount);
    void withdraw(double amount);
```

```
}

// LegacyBankAccount Class
class LegacyBankAccount {
   public void addFunds(double amount) {
      System.out.println("Added " + amount + " to legacy account");
   }

   public void removeFunds(double amount) {
      System.out.println("Removed " + amount + " from legacy account");
   }
}

// BankAccountAdapter Class
class BankAccountAdapter implements BankAccount {
   private LegacyBankAccount legacy_Acc;

   public BankAccountAdapter(LegacyBankAccount legacy_Acc) {
      this.legacy_Acc = legacy_Acc;
   }

   @Override
   public void deposit(double amount) {
      legacy_Acc.addFunds(amount);
   }

   @Override
   public void withdraw(double amount) {
      legacy_Acc.removeFunds(amount);
   }
}

// BankComponent Interface for Composite Pattern
interface BankComponent {
   void showDetails();
}

// BankBranch Class
class BankBranch implements BankComponent {
   private String name;

   public BankBranch(String name) {
      this.name = name;
   }

   @Override
```

```
    public void showDetails() {
        System.out.println("Branch: " + name);
    }
}

// BankComposite Class
class BankComposite implements BankComponent {
    private List<BankComponent> components = new ArrayList<>();

    public void addComponent(BankComponent comp) {
        components.add(comp);
    }

    @Override
    public void showDetails() {
        for (BankComponent comp : components) {
            comp.showDetails();
        }
    }
}
```

//CREATIVE DESIGN PATTERN

```
// Abstract Factory for BankAccount and BankCard
interface BankAccountFactory {
    BankAccount createBankAccount();
    BankCard createBankCard();
}

// Concrete Factory for Savings Account
class SavingsAccountFactory implements BankAccountFactory {
    @Override
    public BankAccount createBankAccount() {
        return new BankAccountAdapter(new LegacyBankAccount());
    }

    @Override
    public BankCard createBankCard() {
        return new SavingsBankCard();
    }
}

// Concrete Factory for Current Account
class CurrentAccountFactory implements BankAccountFactory {
    @Override
    public BankAccount createBankAccount() {
        return new BankAccountAdapter(new LegacyBankAccount());
    }

    @Override
    public BankCard createBankCard() {
        return new CurrentBankCard();
    }
}

// BankCard Interface
interface BankCard {
    void cardType();
}

// SavingsBankCard Class
class SavingsBankCard implements BankCard {
    @Override
    public void cardType() {
        System.out.println("This is a Savings Bank Card");
    }
}

// CurrentBankCard Class
class CurrentBankCard implements BankCard {
    @Override
    public void cardType() {
        System.out.println("This is a Current Bank Card");
    }
}

// LoanPackage Class for Builder Pattern
class LoanPackage {
    private String loanType;
    private double principalAmount;
    private double interestRate;
    private int tenure;

    public LoanPackage(String loanType, double principalAmount, double interestRate, int
tenure) {
        this.loanType = loanType;
        this.principalAmount = principalAmount;
        this.interestRate = interestRate;
        this.tenure = tenure;
    }

    @Override
    public String toString() {
        return "LoanPackage [Type=" + loanType + ", Principal Amount=" + principalAmount
                + ", Interest Rate=" + interestRate + "%, Tenure=" + tenure + " years]";
    }
}

// Builder Interface for LoanPackage
interface LoanPackageBuilder {
    void LoanType(String loanType);
    void PrincipalAmount(double p_Amount);
    void InterestRate(double Rate);
    void Tenure(int time);
    LoanPackage build();
}

// Concrete Builder Class for LoanPackage
class ConcreteLoanPackageBuilder implements LoanPackageBuilder {
    private String loanType;
    private double p_Amount;
    private double Rate;
    private int time;

    @Override
    public void LoanType(String loanType) {
        this.loanType = loanType;
    }

    @Override
    public void PrincipalAmount(double p_Amount) {
        this.p_Amount = p_Amount;
    }

    @Override
    public void InterestRate(double interestRate) {
        this.Rate = Rate;
    }

    @Override
    public void Tenure(int time) {
        this.time = time;
    }

    @Override
    public LoanPackage build() {
        return new LoanPackage(loanType, p_Amount, Rate, time);
    }
```

```java
}

// Singleton Pattern for BankManager
class BankManager {
   private static BankManager instance;

   private BankManager() {}

   public static BankManager getInstance() {
      if (instance == null) {
         instance = new BankManager();
      }
      return instance;
   }

   public void manage() {
      System.out.println("Managing the bank");
   }
}

// Main class
public class BankManagementSystem {
   public static void main(String[] args) {
      Bank bank = new Bank();
      Scanner scanner = new Scanner(System.in);

      System.out.println("***Welcome to the XYZ Bank***");

      // Register account holders
      System.out.print("Enter the number of customers to register: ");
      int num = scanner.nextInt();
      scanner.nextLine(); // Consume the newline

      for (int i = 0; i < num; i++) {
         System.out.print("Enter the name of customer " + (i + 1) + ": ");
         String name = scanner.nextLine();
         Holder acc_Holder = new Holder(name);
         bank.AddCustomer(acc_Holder);

         // Add initial loan amount
         System.out.print("Enter the initial loan amount for " + name + ": ");
         double Amt = scanner.nextDouble();
         scanner.nextLine(); // Consume the newline
         bank.Ori_loan(name, Amt);
      }
```

```
boolean exit = false;

while (!exit) {
    System.out.println("\nSelect an option:");
    System.out.println("1. Update account balance");
    System.out.println("2. Update loan status");
    System.out.println("3. Manage legacy bank account");
    System.out.println("4. Show bank branch details");
    System.out.println("5. Create a bank account and card using abstract factory");
    System.out.println("6. Create a loan package using builder pattern");
    //System.out.println("7. Manage bank operations");
    System.out.println("7. Exit");
    System.out.print("Enter your choice: ");
    int choice = scanner.nextInt();
    scanner.nextLine(); // Consume the newline

    switch (choice) {
        case 1:
            System.out.print("Enter the customer's name: ");
            String cus_name = scanner.nextLine();
            System.out.print("Enter the Deposit amount: ");
            double new_bal = scanner.nextDouble();
            scanner.nextLine(); // Consume the newline
            bank.update_Acc_Balance(cus_name, new_bal);
            break;

        case 2:
            System.out.print("Enter the customer's name: ");
            String l_cus_name = scanner.nextLine();
            System.out.print("Enter the amount paid towards the loan: ");
            double amt_paid = scanner.nextDouble();
            scanner.nextLine(); // Consume the newline
            bank.Loan_status(l_cus_name, amt_paid);
            break;

        case 3:
            System.out.print("Enter deposit amount for legacy bank account: ");
            double legacyDeposit = scanner.nextDouble();
            scanner.nextLine(); // Consume the newline
            LegacyBankAccount legacyAccount = new LegacyBankAccount();
            BankAccountAdapter adapter = new BankAccountAdapter(legacyAccount);
            adapter.deposit(legacyDeposit);

            System.out.print("Enter withdrawal amount for legacy bank account: ");
            double legacyWithdraw = scanner.nextDouble();
            scanner.nextLine(); // Consume the newline
```

```
                adapter.withdraw(legacyWithdraw);
                break;

            case 4:
                BankComposite composite = new BankComposite();
                composite.addComponent(new BankBranch("Main Branch"));
                composite.addComponent(new BankBranch("Sub Branch"));
                composite.showDetails();
                break;

            case 5:
                System.out.print("Enter type of account to create (1 for Savings, 2 for Current):
");
                int accountType = scanner.nextInt();
                scanner.nextLine(); // Consume the newline
                BankAccountFactory factory = accountType == 1 ? new
SavingsAccountFactory() : new CurrentAccountFactory();
                BankAccount account = factory.createBankAccount();
                BankCard card = factory.createBankCard();
                System.out.print("Enter amount to deposit in new account: ");
                double factoryDeposit = scanner.nextDouble();
                scanner.nextLine(); // Consume the newline
                account.deposit(factoryDeposit);
                card.cardType();
                break;

            case 6:
                ConcreteLoanPackageBuilder builder = new ConcreteLoanPackageBuilder();
                System.out.print("Enter loan type: ");
                String loanType = scanner.nextLine();
                builder.LoanType(loanType);
                System.out.print("Enter principal amount: ");
                double p_Amount = scanner.nextDouble();
                scanner.nextLine(); // Consume the newline
                builder.PrincipalAmount(p_Amount);
                System.out.print("Enter interest rate: ");
                double Rate = scanner.nextDouble();
                scanner.nextLine(); // Consume the newline
                builder.InterestRate(Rate);
                System.out.print("Enter tenure in years: ");
                int time = scanner.nextInt();
                scanner.nextLine(); // Consume the newline
                builder.Tenure(time);
                LoanPackage loanPackage = builder.build();
                System.out.println("Loan Package Created: " + loanPackage);
                break;
            /*  case 7:
                    BankManager manager = BankManager.getInstance();
                    manager.manage();
                    break;*/

              case 7:
                    exit = true;
                    break;

              default:
                    System.out.println("Invalid choice. Please try again.");
            }
        }

        scanner.close();
        System.out.println("***Thank you for visiting us***");
    }
}
```