Shem Thuo SCT212-0529/2022

BIT 2203 Advanced Programming

Assignment 1; Quiz 1

1. **Define the Interface:** Create an interface named `TransactionInterface` with abstract methods.

   java

   ```
   import java.util.Calendar;

   public interface TransactionInterface {
       double getAmount();
       Calendar getDate();
       String getTransactionID();
       void printTransactionDetails();
       void apply(BankAccount ba);
   }
   ```

2. **Concrete Class: BaseTransaction** Implement the interface in your `BaseTransaction` class.

   java

   ```
   import java.util.Calendar;

   public class BaseTransaction implements TransactionInterface {
       protected double amount;
       protected Calendar date;
       protected String transactionID;

       public BaseTransaction(double amount, Calendar date, String
   transactionID) {
           this.amount = amount;
           this.date = date;
           this.transactionID = transactionID;
       }

       @Override
       public double getAmount() {
           return amount;
       }

       @Override
       public Calendar getDate() {
           return date;
       }

       @Override
       public String getTransactionID() {
   ```

```
        return transactionID;
    }

    @Override
    public void printTransactionDetails() {
        System.out.println("Transaction ID: " + transactionID);
        System.out.println("Amount: " + amount);
        System.out.println("Date: " + date.getTime());
    }

    @Override
    public void apply(BankAccount ba) {
        // Default apply implementation for BaseTransaction
    }
}
```

3. **Derived Classes: DepositTransaction and WithdrawalTransaction** Override the `apply` method in the derived classes.

   java

```
import java.util.Calendar;

public class DepositTransaction extends BaseTransaction {

    public DepositTransaction(double amount, Calendar date, String
transactionID) {
        super(amount, date, transactionID);
    }

    @Override
    public void apply(BankAccount ba) {
        ba.deposit(amount);
        System.out.println("Deposited: " + amount);
    }
}

public class WithdrawalTransaction extends BaseTransaction {

    public WithdrawalTransaction(double amount, Calendar date, String
transactionID) {
        super(amount, date, transactionID);
    }

    @Override
    public void apply(BankAccount ba) {
        ba.withdraw(amount);
        System.out.println("Withdrew: " + amount);
    }
}
```

4. **Custom Exception Class:** Create a custom exception class to handle specific transaction-related errors.

java

```
public class InsufficientFundsException extends Exception {
    public InsufficientFundsException(String message) {
        super(message);
    }
}
```

5. **Handling Exceptions:** Demonstrate the use of the custom exception in your transaction methods.

java

```
public class BankAccount {
    private double balance;

    public BankAccount(double initialBalance) {
        this.balance = initialBalance;
    }

    public void deposit(double amount) {
        balance += amount;
    }

    public void withdraw(double amount) throws
InsufficientFundsException {
        if (amount > balance) {
            throw new InsufficientFundsException("Insufficient funds
for withdrawal");
        }
        balance -= amount;
    }

    public double getBalance() {
        return balance;
    }
}
```

6. **Client Code:** Implement client code to demonstrate the functionality.

java

```
import java.util.Calendar;

public class Main {
    public static void main(String[] args) {
        BankAccount account = new BankAccount(1000);
        Calendar date = Calendar.getInstance();

        DepositTransaction deposit = new DepositTransaction(200, date,
"TXN001");
        WithdrawalTransaction withdrawal = new
WithdrawalTransaction(150, date, "TXN002");
```

```
        deposit.apply(account);
        withdrawal.apply(account);

        System.out.println("Final Balance: " + account.getBalance());
    }
}
```