https://github.com/AKASHYADAVO/C2TC_Core_Java/blob/master/CaseStudyAll/CaseStudy1/Java Case Study Framework - I - Shopping Account -Final.pdf

Step-by-Step Guide to Create Each Class and Interface for the tns Case study on java given on
21/6/2024 Friday 1pm
CaseStudy No : 1
Student_Name : AKASH YADAV

1. Abstract Class `ShopAcc`

- **Objective**: This class represents an abstract online shopping account.

1. **Create a new Java class file** named `ShopAcc.java`.
2. Define the class as `abstract`.
3. Add private fields `accNo` (int) and `accNm` (String).
4. Implement a constructor to initialize `accNo` and `accNm`.
5. Declare an abstract method `public abstract void bookProduct(float amount);`.
6. Override the `toString()` method to provide a string representation of the account details.

Here's an example code snippet for `ShopAcc`:

```
// ShopAcc.java
public abstract class ShopAcc {
    private int accNo;
    private String accNm;

    public ShopAcc(int accNo, String accNm) {
        this.accNo = accNo;
        this.accNm = accNm;
    }

    public abstract void bookProduct(float amount);

    @Override
    public String toString() {
        return "Account Number: " + accNo + ", Account Name: " + accNm;
    }
}
```

```java
// ShopAcc.java
public abstract class ShopAcc {
    private int accNo;
    private String accNm;

    public ShopAcc(int accNo, String accNm) {
```

```
        this.accNo = accNo;
        this.accNm = accNm;
    }

    public abstract void bookProduct(float amount);

    @Override
    public String toString() {
        return "Account Number: " + accNo + ", Account Name: " + accNm;
    }
}
```

2. Abstract Class `PrimeAcc` extending `ShopAcc`

- **Objective**: This class represents a prime online shopping account, extending `ShopAcc`.

1. **Create a new Java class file** named `PrimeAcc.java`.
2. Define the class as `abstract` and extend `ShopAcc`.
3. Implement a constructor to call the superclass constructor.
4. Override the `bookProduct(float amount)` method to provide specific logic for prime accounts (no delivery charges).

Here's an example code snippet for `PrimeAcc`:

```
// PrimeAcc.java
public abstract class PrimeAcc extends ShopAcc {
    public PrimeAcc(int accNo, String accNm) {
        super(accNo, accNm);
    }

    @Override
    public void bookProduct(float amount) {
        System.out.println("Booking product for Prime Account: " + amount);
        // Specific logic for prime accounts
    }
}
```

```java
// PrimeAcc.java
public abstract class PrimeAcc extends ShopAcc {
    public PrimeAcc(int accNo, String accNm) {
        super(accNo, accNm);
    }

    @Override
    public void bookProduct(float amount) {
        System.out.println("Booking product for Prime Account: " + amount);
        // Specific logic for prime accounts
```

```
    }
}
```

3. Abstract Class `NormalAcc` extending `ShopAcc`

- **Objective**: This class represents a normal (non-prime) online shopping account, extending `ShopAcc`.

1. **Create a new Java class file** named `NormalAcc.java`.
2. Define the class as `abstract` and extend `ShopAcc`.
3. Implement a constructor to call the superclass constructor.
4. Override the `bookProduct(float amount)` method to provide specific logic for normal accounts (with delivery charges).

Here's an example code snippet for `NormalAcc`:

```
// NormalAcc.java
public abstract class NormalAcc extends ShopAcc {
    public NormalAcc(int accNo, String accNm) {
        super(accNo, accNm);
    }

    @Override
    public void bookProduct(float amount) {
        float deliveryCharge = 10.0f; // Example delivery charge
        System.out.println("Booking product for Normal Account with delivery charge: " +
(amount + deliveryCharge));
        // Specific logic for normal accounts
    }
}
```

```java
// NormalAcc.java
public abstract class NormalAcc extends ShopAcc {
    public NormalAcc(int accNo, String accNm) {
        super(accNo, accNm);
    }

    @Override
    public void bookProduct(float amount) {
        float deliveryCharge = 10.0f; // Example delivery charge
        System.out.println("Booking product for Normal Account with delivery charge: " +
(amount + deliveryCharge));
        // Specific logic for normal accounts
    }
}
```

4. Abstract Class `ShopFactory`

- **Objective**: This abstract class provides factory methods to create instances of `PrimeAcc` and `NormalAcc`.

1. **Create a new Java class file** named `ShopFactory.java`.
2. Define the class as `abstract`.
3. Declare abstract methods `public abstract PrimeAcc getNewPrimeAccount(int accNo, String accNm);` and `public abstract NormalAcc getNewNormalAccount(int accNo, String accNm);`.

Here's an example code snippet for `ShopFactory`:

```
// ShopFactory.java
public abstract class ShopFactory {
    public abstract PrimeAcc getNewPrimeAccount(int accNo, String accNm);
    public abstract NormalAcc getNewNormalAccount(int accNo, String accNm);
}
```

```java
// ShopFactory.java
public abstract class ShopFactory {
    public abstract PrimeAcc getNewPrimeAccount(int accNo, String accNm);
    public abstract NormalAcc getNewNormalAccount(int accNo, String accNm);
}
```

5. Concrete Class `GSShopFactory` extending `ShopFactory`

- **Objective**: This concrete class implements `ShopFactory` to instantiate `GSPrimeAcc` and `GSNormalAcc`.

1. **Create a new Java class file** named `GSShopFactory.java`.
2. Define the class as `public` and extend `ShopFactory`.
3. Implement the abstract methods `getNewPrimeAccount` and `getNewNormalAccount` to return instances of `GSPrimeAcc` and `GSNormalAcc`, respectively.

Here's an example code snippet for `GSShopFactory`:

```
// GSShopFactory.java
public class GSShopFactory extends ShopFactory {
    @Override
    public PrimeAcc getNewPrimeAccount(int accNo, String accNm) {
        return new GSPrimeAcc(accNo, accNm);
    }

    @Override
    public NormalAcc getNewNormalAccount(int accNo, String accNm) {
        return new GSNormalAcc(accNo, accNm);
    }
}
```

```java
// GSShopFactory.java
public class GSShopFactory extends ShopFactory {
    @Override
    public PrimeAcc getNewPrimeAccount(int accNo, String accNm) {
        return new GSPrimeAcc(accNo, accNm);
    }

    @Override
    public NormalAcc getNewNormalAccount(int accNo, String accNm) {
        return new GSNormalAcc(accNo, accNm);
    }
}
```

6. Concrete Class `GSPrimeAcc` extending `PrimeAcc`

- **Objective**: This concrete class represents a specific implementation of a prime account.

1. **Create a new Java class file** named `GSPrimeAcc.java`.
2. Define the class as `public` and extend `PrimeAcc`.
3. Implement a constructor to call the superclass constructor.

Here's an example code snippet for `GSPrimeAcc`:

```
// GSPrimeAcc.java
public class GSPrimeAcc extends PrimeAcc {
    public GSPrimeAcc(int accNo, String accNm) {
        super(accNo, accNm);
    }
}
```

```java
// GSPrimeAcc.java
public class GSPrimeAcc extends PrimeAcc {
    public GSPrimeAcc(int accNo, String accNm) {
        super(accNo, accNm);
    }
}
```

7. Concrete Class `GSNormalAcc` extending `NormalAcc`

- **Objective**: This concrete class represents a specific implementation of a normal account.

1. **Create a new Java class file** named `GSNormalAcc.java`.
2. Define the class as `public` and extend `NormalAcc`.
3. Implement a constructor to call the superclass constructor.

Here's an example code snippet for `GSNormalAcc`:

```
// GSNormalAcc.java
public class GSNormalAcc extends NormalAcc {
    public GSNormalAcc(int accNo, String accNm) {
        super(accNo, accNm);
    }
}
```

```java
// GSNormalAcc.java
public class GSNormalAcc extends NormalAcc {
    public GSNormalAcc(int accNo, String accNm) {
        super(accNo, accNm);
    }
}
```

8. Main Application `GoShoppingApp` (Entry Point)

- **Objective**: This class serves as the entry point to test the functionality of the framework.

1. **Create a new Java class file** named `GoShoppingApp.java`.
2. Define the class as `public`.
3. Implement the `main` method to instantiate `GSShopFactory`, create instances of `PrimeAcc` and `NormalAcc` using the factory, and test their methods.

Here's an example code snippet for `GoShoppingApp`:

```
// GoShoppingApp.java
public class GoShoppingApp {
    public static void main(String[] args) {
        // Create an instance of GSShopFactory
        ShopFactory shopFactory = new GSShopFactory();

        // Instantiate GSPrimeAcc and GSNormalAcc
        PrimeAcc primeAcc = shopFactory.getNewPrimeAccount(1, "Prime Customer");
        NormalAcc normalAcc = shopFactory.getNewNormalAccount(2, "Normal Customer");

        // Invoke methods
        primeAcc.bookProduct(100.0f);
        normalAcc.bookProduct(50.0f);

        // Invoke toString() method
        System.out.println(primeAcc.toString());
        System.out.println(normalAcc.toString());
    }
}
```

```java
// GoShoppingApp.java
public class GoShoppingApp {
    public static void main(String[] args) {
        // Create an instance of GSShopFactory
        ShopFactory shopFactory = new GSShopFactory();

        // Instantiate GSPrimeAcc and GSNormalAcc
        PrimeAcc primeAcc = shopFactory.getNewPrimeAccount(1, "Prime Customer");
        NormalAcc normalAcc = shopFactory.getNewNormalAccount(2, "Normal Customer");

        // Invoke methods
        primeAcc.bookProduct(100.0f);
        normalAcc.bookProduct(50.0f);

        // Invoke toString() method
        System.out.println(primeAcc.toString());
        System.out.println(normalAcc.toString());
    }
}
```

Summary

These steps outline the creation of each class and interface based on the provided Java case study. Each class and interface is structured to fulfill its specific role within the online shopping application framework, adhering to the principles of abstraction, inheritance, and polymorphism as described in your requirements. Adjustments and additional features can be incorporated based on further project specifications or requirements.