**Assignment for Interface and Abstract :**

**Part - 1**

Objective:

The objective of this assignment is to reinforce your understanding of abstract classes, interfaces, method overriding, and polymorphism in Java.

Instructions:

You are tasked with creating a Java program that models a simple shape hierarchy. This hierarchy will include abstract classes, interfaces, and demonstrate polymorphic behavior. Follow the steps below to complete the assignment.

**Step 1:** Define an Interface

Create an interface named `Shape` with the following abstract methods:

public double area();

public double perimeter();

**Step 2:** Create an Abstract Class

Create an abstract class named `AbstractShape` that implements the `Shape` interface. This class should include an instance variable for the color of the shape (a string) and provide a constructor to initialize it. It should also declare abstract methods for `area()` and `perimeter()`.

**Step 3:** Implement Concrete Classes

Create two concrete classes that extend `AbstractShape`. Name them `Circle` and `Rectangle`. For each class:

- Implement the constructor(s) to initialize the necessary attributes.

- Override the abstract methods `area()` and `perimeter()` to provide specific implementations for each shape.

**Step 4:** Demonstrate Polymorphism

In your `Main` class, create an array of `Shape` references and store instances of both `Circle` and `Rectangle` in the array. Then, iterate through the array and for each shape, print out its color, area, and perimeter.

**Step 5 :**

Implement additional shape classes such as `Triangle` or `Square` and add them to the polymorphic array in the `Main` class. Demonstrate the flexibility and extensibility of your design.

**Step 6:** Testing

Write a test program in the `Main` class to demonstrate that your classes and their methods work correctly. Instantiate objects of `Circle` and `Rectangle`, and use them in your test program.

**Part – 2**

Objective:The objective of this assignment is to test your understanding of abstract classes, interfaces, method overriding, polymorphism, and exception handling in Java.

Task:You are required to create a Java program that simulates a simple banking system. You need to implement the following classes and interfaces:

1. BankAccount` (Abstract Class):This class should be an abstract class that represents a bank account. It should have the following properties and methods:

- Properties:

- `accountNumber` (int)

- `accountHolder` (String)

- `balance` (double)

- Methods:

- `abstract void deposit(double amount)` - Deposit money into the account.

- `abstract void withdraw(double amount)` - Withdraw money from the account.

- `void displayAccountInfo()` - Display the account information (account number, account holder, and balance).

2. `SavingsAccount` (Class):This class should extend the `BankAccount` abstract class and implement the `TransactionFee` interface. The `SavingsAccount` class should include the following properties and methods:

- Properties:

- `annualInterestRate` (double)

- Methods:

- `SavingsAccount(int accountNumber, String accountHolder, double balance, double annualInterestRate)` - Constructor to initialize the account properties.

- `void addInterest()` - Add annual interest to the balance based on the annual interest rate.

- `void displayAccountInfo()` - Override this method to display account information and the annual interest rate.

3. ‘TransactionFee` (Interface):This interface should define the following method:

- `void applyTransactionFee(double fee)` - Apply a transaction fee to the account balance.

4. ’Main` (Class):Create a `Main` class with a `main` method to test your `BankAccount` and `SavingsAccount` classes. Create instances of both classes, perform transactions, and demonstrate polymorphism by using the same method to display account information.

Requirements:

- Implement exception handling for withdrawal transactions to ensure that the balance does not go negative. If a withdrawal would result in a negative balance, throw a custom `InsufficientFundsException` (you should define this exception class).

- Handle exceptions gracefully by displaying error messages.

- Test your program thoroughly to demonstrate the functionality of the classes and interfaces.