Exercice 1 : Définir une Classe Abstraite et Implémenter des Sous-classes

1. **Définissez une classe abstraite `Shape` avec les méthodes suivantes :**
   - Une méthode abstraite `double getArea();`
   - Une méthode abstraite `double getPerimeter();`
   - Une méthode concrète `void display();` qui affiche "This is a shape".
2. **Créez une classe `Circle` qui étend la classe abstraite `Shape`. Dans cette classe :**
   - Ajoutez un champ `radius`.
   - Implémentez les méthodes `getArea()` et `getPerimeter()`. Utilisez la formule (Area = PI ×radius^2) et (Perimeter = 2 ×PI ×radius).
   - Surchargez la méthode `display()` pour afficher "This is a circle".
3. **Créez une classe `Rectangle` qui étend la classe abstraite `Shape`. Dans cette classe :**
   - Ajoutez les champs `width` et `height`.
   - Implémentez les méthodes `getArea()` et `getPerimeter()`. Utilisez la formule (Area = width ×height) et (Perimeter = 2 ×(width + height)).
   - Surchargez la méthode `display()` pour afficher "This is a rectangle".
4. **Créez une classe principale `Main` avec une méthode `main` pour tester vos classes `Circle` et `Rectangle`.**

```java
// Define the abstract Shape class
abstract class Shape {
    abstract double getArea();
    abstract double getPerimeter();

    void display() {
        System.out.println("This is a shape");
    }
}

// Implement the Circle class
class Circle extends Shape {
    private double radius;
```

```
    public Circle(double radius) {
        this.radius = radius;
    }

    @Override
    double getArea() {
        return Math.PI * radius * radius;
    }

    @Override
    double getPerimeter() {
        return 2 * Math.PI * radius;
    }

    @Override
    void display() {
        System.out.println("This is a circle");
    }
}

// Implement the Rectangle class
class Rectangle extends Shape {
    private double width;
    private double height;

    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }

    @Override
    double getArea() {
        return width * height;
    }

    @Override
    double getPerimeter() {
        return 2 * (width + height);
    }
```

```
    @Override
    void display() {
        System.out.println("This is a rectangle");
    }
}

// Main class to test Circle and Rectangle classes
public class Main {
    public static void main(String[] args) {
        Shape circle = new Circle(5);
        circle.display();
        System.out.println("Circle Area: " + circle.getArea());
        System.out.println("Circle Perimeter: " + circle.getPerimeter());

        Shape rectangle = new Rectangle(4, 7);
        rectangle.display();
        System.out.println("Rectangle Area: " + rectangle.getArea());
        System.out.println("Rectangle Perimeter: " + rectangle.getPerimeter());
    }
}
```

Exercice 2 : Classe Abstraite avec des Méthodes Abstraites et Concrètes

1. **Définissez une classe abstraite `Employee` avec les méthodes suivantes :**
   - Une méthode abstraite `double calculateSalary();`
   - Une méthode concrète `void displayDetails();` qui affiche les détails de l'employé.
   - Ajoutez des champs `name` et `id`.
2. **Créez une classe `FullTimeEmployee` qui étend la classe abstraite `Employee`. Dans cette classe :**
   - Ajoutez un champ `salary`.
   - Implémentez la méthode `calculateSalary()` pour retourner le salaire.
   - Surchargez la méthode `displayDetails()` pour afficher les détails de l'employé à temps plein.
3. **Créez une classe `PartTimeEmployee` qui étend la classe abstraite `Employee`. Dans cette classe :**

- Ajoutez des champs `hourlyRate` et `hoursWorked`.
- Implémentez la méthode `calculateSalary()` pour retourner le produit de `hourlyRate` et `hoursWorked`.
- Surchargez la méthode `displayDetails()` pour afficher les détails de l'employé à temps partiel.

4. **Créez une classe principale `Main` avec une méthode `main` pour tester vos classes `FullTimeEmployee` et `PartTimeEmployee`.**

```
// Define the abstract Employee class
abstract class Employee {
    String name;
    int id;

    public Employee(String name, int id) {
        this.name = name;
        this.id = id;
    }

    abstract double calculateSalary();

    void displayDetails() {
        System.out.println("Employee Name: " + name);
        System.out.println("Employee ID: " + id);
    }
}

// Implement the FullTimeEmployee class
class FullTimeEmployee extends Employee {
    private double salary;

    public FullTimeEmployee(String name, int id, double salary) {
        super(name, id);
        this.salary = salary;
    }

    @Override
    double calculateSalary() {
        return salary;
    }
```

```
    @Override
    void displayDetails() {
        super.displayDetails();
        System.out.println("Full-Time Employee Salary: " + salary);
    }
}

// Implement the PartTimeEmployee class
class PartTimeEmployee extends Employee {
    private double hourlyRate;
    private int hoursWorked;

    public PartTimeEmployee(String name, int id, double hourlyRate, int hoursWorked)
        super(name, id);
        this.hourlyRate = hourlyRate;
        this.hoursWorked = hoursWorked;
    }

    @Override
    double calculateSalary() {
        return hourlyRate * hoursWorked;
    }

    @Override
    void displayDetails() {
        super.displayDetails();
        System.out.println("Part-Time Employee Hourly Rate: " + hourlyRate);
        System.out.println("Part-Time Employee Hours Worked: " + hoursWorked);
        System.out.println("Part-Time Employee Salary: " + calculateSalary());
    }
}

// Main class to test FullTimeEmployee and PartTimeEmployee classes
public class Main {
    public static void main(String[] args) {
        Employee fullTime = new FullTimeEmployee("Alice", 101, 5000);
        fullTime.displayDetails();

        Employee partTime = new PartTimeEmployee("Bob", 102, 20, 80);
        partTime.displayDetails();
```

```
    }
}
```

Exercice 3 : Héritage Multiple via Interfaces et Classes Abstraites

1. **Définissez une interface `Drivable` avec les méthodes suivantes :**
   - `void drive();`
   - `void stop();`
2. **Définissez une classe abstraite `Vehicle` avec les champs et méthodes suivants :**
   - Un champ `make`.
   - Une méthode abstraite `void start();`
   - Une méthode concrète `void displayMake();` qui affiche la marque du véhicule.
3. **Créez une classe `Car` qui étend la classe abstraite `Vehicle` et implémente l'interface `Drivable`. Dans cette classe :**
   - Implémentez les méthodes `start()`, `drive()`, et `stop()` avec des affichages appropriés.
4. **Créez une classe principale `Main` avec une méthode `main` pour tester votre classe `Car`.**

```java
// Define the Drivable interface
interface Drivable {
    void drive();
    void stop();
}

// Define the abstract Vehicle class
abstract class Vehicle {
    String make;

    public Vehicle(String make) {
        this.make = make;
    }

    abstract void start();

    void displayMake() {
        System.out.println("Vehicle Make: " + make);
```

```
    }
}

// Implement the Car class
class Car extends Vehicle implements Drivable {
    public Car(String make) {
        super(make);
    }

    @Override
    void start() {
        System.out.println("Car is starting");
    }

    @Override
    public void drive() {
        System.out.println("Car is driving");
    }

    @Override
    public void stop() {
        System.out.println("Car has stopped");
    }
}

// Main class to test Car class
public class Main {
    public static void main(String[] args) {
        Car car = new Car("Toyota");
        car.displayMake();
        car.start();
        car.drive();
        car.stop();
    }
}
```