1. Abstract Class `Shape` with Derived Classes `Rectangle` and `Circle`

Create an abstract class `Shape` with an abstract method `calculateArea()`. Derive two classes `Rectangle` and `Circle` from `Shape`.

Shape.java:

```java
abstract class Shape {
    public abstract void calculateArea();
}
```

Rectangle.java:

```java
class Rectangle extends Shape {
    private double length;
    private double width;

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

    @Override
    public void calculateArea() {
        double area = length * width;
        System.out.println("Area of Rectangle: " + area);
    }
}
```

Circle.java:

```java
class Circle extends Shape {
    private double radius;

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

    @Override
    public void calculateArea() {
        double area = Math.PI * radius * radius;
        System.out.println("Area of Circle: " + area);
    }
}
```

Main.java:

```java
public class Main {
    public static void main(String[] args) {
        Shape rectangle = new Rectangle(5.0, 10.0);
        rectangle.calculateArea();

        Shape circle = new Circle(7.0);
        circle.calculateArea();
    }
}
```

Output:

```
Area of Rectangle: 50.0
Area of Circle: 153.93804002589985
```

---

2. Abstract Class `Employee` with Derived Classes `Manager` and `Developer`

Create an abstract class `Employee` with abstract methods `calculateSalary()` and `displayDetails()`. Derive two classes `Manager` and `Developer`.

Employee.java:

```java
abstract class Employee {
    protected String name;
    protected String role;

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

    public abstract void calculateSalary();
    public abstract void displayDetails();
}
```

Manager.java:

```java
class Manager extends Employee {
    private double fixedSalary;

    public Manager(String name, double fixedSalary) {
        super(name, "Manager");
        this.fixedSalary = fixedSalary;
    }

    @Override
    public void calculateSalary() {
        System.out.println("Calculating Salary for Manager: " + fixedSalary);
    }
    @Override
    public void displayDetails() {
        System.out.println("Name: " + name + ", Role: " + role + ", Salary: " + fixedSalary
    }
}
```

Developer.java:

```
class Developer extends Employee {
    private double hourlyWage;
    private int hoursWorked;

    public Developer(String name, double hourlyWage, int hoursWorked) {
        super(name, "Developer");
        this.hourlyWage = hourlyWage;
        this.hoursWorked = hoursWorked;
    }

    @Override
    public void calculateSalary() {
        double salary = hourlyWage * hoursWorked;
        System.out.println("Calculating Salary for Developer: " + salary);
    }

    @Override
    public void displayDetails() {
        double salary = hourlyWage * hoursWorked;
        System.out.println("Name: " + name + ", Role: " + role + ", Salary: " + salary);
    }
}
```

Main.java:

```
public class Main {
    public static void main(String[] args) {
        Employee manager = new Manager("John Doe", 5000.0);
        manager.calculateSalary();
        manager.displayDetails();

        Employee developer = new Developer("Jane Smith", 25.0, 160);
        developer.calculateSalary();
        developer.displayDetails();
    }
}
```

Output:

```
Calculating Salary for Manager: 5000.0
```

```
Name: John Doe, Role: Manager, Salary: 5000.0
Calculating Salary for Developer: 4000.0
Name: Jane Smith, Role: Developer, Salary: 4000.0
```

---

3. Interface `Bank` with Class `Account`

Create an interface `Bank` with methods `deposit()` and `withdraw()`. Implement this interface in a class `Account`.

Bank.java:

```java
interface Bank {
    void deposit(double amount);
    void withdraw(double amount);
}
```

Account.java:

```java
class Account implements Bank {
    private double balance;

    public Account(double balance) {
        this.balance = balance;
    }

    @Override
    public void deposit(double amount) {
        balance += amount;
        System.out.println("Deposited: " + amount + ", New Balance: " + balance);
    }

    @Override
    public void withdraw(double amount) {
        if (amount <= balance) {
            balance -= amount;
            System.out.println("Withdrawn: " + amount + ", New Balance: " + balance);
        } else {
            System.out.println("Insufficient Balance");
        }
    }
}
```

BankDemo.java:

```java
public class BankDemo {
    public static void main(String[] args) {
        Bank account = new Account(1000.0);
        account.deposit(500.0);
        account.withdraw(200.0);
        account.withdraw(1500.0);
    }
}
```

Output:

```
Deposited: 500.0, New Balance: 1500.0
Withdrawn: 200.0, New Balance: 1300.0
Insufficient Balance
```

---

4. Interface `Playable` with Class `MusicPlayer`

Create an interface `Playable` with methods `play()`, `pause()`, and `stop()`. Implement this interface in a class `MusicPlayer`.

Playable.java:

```
interface Playable {
    void play();
    void pause();
    void stop();
}
```

MusicPlayer.java:

```
class MusicPlayer implements Playable {
    @Override
    public void play() {
        System.out.println("Music is playing");
    }

    @Override
    public void pause() {
        System.out.println("Music is paused");
    }

    @Override
    public void stop() {
        System.out.println("Music is stopped");
    }
}
```

TestPlayer.java:

```
public class TestPlayer {
    public static void main(String[] args) {
        Playable player = new MusicPlayer();
        player.play();
        player.pause();
        player.stop();
    }
}
```

Output:

```
Music is playing
Music is paused
Music is stopped
```