1. **Abstract Class with Abstract Method:**

// Define an abstract class named Shape with an abstract method draw().

// Create two concrete subclasses: Circle and Rectangle that extend Shape and implement the draw() method.

1. **Abstract Class with Constructor and Method:**

// Create an abstract class Animal with a constructor that initializes a name field.

// Add a method called sound() that prints a generic sound message.

// Implement a subclass Dog that provides its own implementation for the sound() method.

1. **Abstract Class with Static Method:**

// Define an abstract class MathOperations with a static method calculate(int a, int b).

// Attempt to extend this class with a subclass that overrides the static method and show the result.

1. **Abstract Class with Final Method:**

// Create an abstract class Account with a final method getAccountType().

// Implement a subclass SavingsAccount that uses the final method and demonstrates its usage.

1. **Interface Definition:**

// Define an interface Vehicle with methods start() and stop().

// Implement this interface in a class Car and provide concrete implementations for start() and stop().

1. **Interface with Default Method:**

// Create an interface Printer with a default method print() that prints "Printing".

// Implement this interface in a class InkjetPrinter and override the print() method to print "Inkjet Printing".

1. **Interface Inheritance:**

// Define two interfaces Worker and Manager, each with a method work().

// Create a new interface Executive that extends both Worker and Manager.

// Implement Executive in a class CEO and provide implementations for the work() method.

1. **Abstract Class with Method Overriding:**

// Create an abstract class Vehicle with an abstract method fuelEfficiency().

// Implement a concrete subclass Car that overrides fuelEfficiency() to return a specific value.

1. **Abstract Class with Multiple Methods:**

// Define an abstract class Device with methods turnOn(), turnOff(), and getStatus().

// Implement a concrete class Smartphone that provides its own implementations for these methods.

1. **Interface Implementation with Multiple Interfaces:**

// Define two interfaces MusicPlayer and VideoPlayer with methods play() and stop().

// Create a class MultimediaPlayer that implements both interfaces and provides concrete implementations for play() and stop().

**1. Abstract Class with Abstract Method**

// Abstract class

abstract class Shape {

abstract void draw();

}

// Concrete subclass Circle

class Circle extends Shape {

@Override

void draw() {

System.out.println("Drawing a Circle");

}

}

// Concrete subclass Rectangle

class Rectangle extends Shape {

@Override

void draw() {

System.out.println("Drawing a Rectangle");

}

}

// Test the classes

public class TestShape {

public static void main(String[] args) {

Shape circle = new Circle();

Shape rectangle = new Rectangle();

circle.draw();

rectangle.draw();

}

}

**2. Abstract Class with Constructor and Method**

// Abstract class

abstract class Animal {

String name;

// Constructor

Animal(String name) {

this.name = name;

}

// Method

void sound() {

System.out.println("Some generic animal sound");

}

}

// Concrete subclass Dog

class Dog extends Animal {

Dog(String name) {

super(name);

}

@Override

void sound() {

System.out.println(name + " says Woof!");

}

}

// Test the classes

public class TestAnimal {

public static void main(String[] args) {

Animal dog = new Dog("Buddy");

dog.sound();

}

}

**3. Abstract Class with Static Method**

// Abstract class

abstract class MathOperations {

// Static method

static int calculate(int a, int b) {

return a + b;

}

}

// Concrete class

class AdvancedMath extends MathOperations {

// Attempt to override static method (not allowed)

// Uncommenting the following code will cause a compilation error

// static int calculate(int a, int b) {

// return a \* b;

// }

}

// Test the classes

public class TestMathOperations {

public static void main(String[] args) {

System.out.println("Sum: " + MathOperations.calculate(5, 3));

// System.out.println("Product: " + AdvancedMath.calculate(5, 3)); // This will give an error

}

}

**4. Abstract Class with Final Method**

// Abstract class

abstract class Account {

// Final method

final String getAccountType() {

return "Generic Account Type";

}

}

// Concrete subclass

class SavingsAccount extends Account {

// Uses the final method from Account

void displayAccountType() {

System.out.println("Account Type: " + getAccountType());

}

}

// Test the classes

public class TestAccount {

public static void main(String[] args) {

SavingsAccount savings = new SavingsAccount();

savings.displayAccountType();

}

}

**5. Interface Definition**

// Interface

interface Vehicle {

void start();

void stop();

}

// Concrete class

class Car implements Vehicle {

@Override

public void start() {

System.out.println("Car is starting");

}

@Override

public void stop() {

System.out.println("Car is stopping");

}

}

// Test the class

public class TestVehicle {

public static void main(String[] args) {

Vehicle car = new Car();

car.start();

car.stop();

}

}

**6. Interface with Default Method**

// Interface

interface Printer {

default void print() {

System.out.println("Printing");

}

}

// Concrete class

class InkjetPrinter implements Printer {

@Override

public void print() {

System.out.println("Inkjet Printing");

}

}

// Test the class

public class TestPrinter {

public static void main(String[] args) {

Printer printer = new InkjetPrinter();

printer.print();

}

}

**7. Interface Inheritance**

// Interfaces

interface Worker {

void work();

}

interface Manager {

void work();

}

// Interface extending both Worker and Manager

interface Executive extends Worker, Manager {}

// Concrete class

class CEO implements Executive {

@Override

public void work() {

System.out.println("CEO is working");

}

}

// Test the class

public class TestExecutive {

public static void main(String[] args) {

Executive ceo = new CEO();

ceo.work();

}

}

**8. Abstract Class with Method Overriding**

// Abstract class

abstract class Vehicle {

abstract int fuelEfficiency();

}

// Concrete subclass

class Car extends Vehicle {

@Override

int fuelEfficiency() {

return 25; // miles per gallon

}

}

// Test the class

public class TestVehicle {

public static void main(String[] args) {

Vehicle car = new Car();

System.out.println("Fuel Efficiency: " + car.fuelEfficiency() + " MPG");

}

}

**9. Abstract Class with Multiple Methods**

// Abstract class

abstract class Device {

abstract void turnOn();

abstract void turnOff();

abstract String getStatus();

}

// Concrete class

class Smartphone extends Device {

private String status = "Off";

@Override

void turnOn() {

status = "On";

System.out.println("Smartphone is turned on");

}

@Override

void turnOff() {

status = "Off";

System.out.println("Smartphone is turned off");

}

@Override

String getStatus() {

return status;

}

}

// Test the class

public class TestDevice {

public static void main(String[] args) {

Device phone = new Smartphone();

phone.turnOn();

System.out.println("Status: " + phone.getStatus());

phone.turnOff();

System.out.println("Status: " + phone.getStatus());

}

}

**10. Interface Implementation with Multiple Interfaces**

// Interfaces

interface MusicPlayer {

void play();

void stop();

}

interface VideoPlayer {

void play();

void stop();

}

// Concrete class

class MultimediaPlayer implements MusicPlayer, VideoPlayer {

@Override

public void play() {

System.out.println("Playing media");

}

@Override

public void stop() {

System.out.println("Stopping media");

}

}

// Test the class

public class TestMultimediaPlayer {

public static void main(String[] args) {

MultimediaPlayer player = new MultimediaPlayer();

player.play();

player.stop();

}

}