Unit 3: Abstract Class in Java

Abstract Methods & Classes, Packages& Interfaces:
Built-In Packages and User Defined Packages, Interfaces: Declaration, Implementation, Extending Classes and Interfaces

Abstract Class in Java
An **abstract class** is a class that **cannot be instantiated** and is meant to be extended by subclasses. It can contain:

- **Abstract methods** (methods without a body, which subclasses must implement).
- **Concrete methods** (methods with implementation, inherited by subclasses).
- **Constructors, instance variables, static methods, and final methods**.

```
abstract class Shape {
   // Abstract method (must be implemented in subclasses)
   abstract double calculateArea();


   // Concrete method (inherited by all subclasses)
   void display() {
      System.out.println("Area: " + calculateArea());
   }
}


class Circle extends Shape {
   double radius;

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


   // Implement abstract method
   @Override
   double calculateArea() {
      return Math.PI * radius * radius;
   }
}


class Rectangle extends Shape {
   double length, width;

   // Constructor
   public Rectangle(double length, double width) {
      this.length = length;
      this.width = width;
```

```
    }

    // Implement abstract method
    @Override
    double calculateArea() {
        return length * width;
    }

}

public class Main {
    public static void main(String[] args) {
        Shape circle = new Circle(5);
        circle.display(); // Calls calculateArea() from Circle

        Shape rectangle = new Rectangle(4, 6);
        rectangle.display(); // Calls calculateArea() from Rectangle
    }
}
```

Abstract class with all concepts

```
// Abstract class representing a Vehicle
abstract class Vehicle {
    protected String brand; // Instance variable (protected so subclasses can access)
    protected int speed;

    // Constructor
    public Vehicle(String brand, int speed) {
        this.brand = brand;
        this.speed = speed;
    }

    // Abstract method (must be implemented by subclasses)
    abstract void accelerate();

    // Concrete method (inherited by subclasses)
    void showDetails() {
        System.out.println("Brand: " + brand + ", Speed: " + speed + " km/h");
    }

    // Static method (shared method, belongs to the class)
    static void vehicleInfo() {
        System.out.println("Vehicles are used for transportation.");
    }
```

```
    // Final method (cannot be overridden in subclasses)
    final void stop() {
        System.out.println("Vehicle is stopping...");
    }
}

// Subclass: Car
class Car extends Vehicle {
    private int fuelCapacity;

    // Constructor for Car (calls Vehicle constructor)
    public Car(String brand, int speed, int fuelCapacity) {
        super(brand, speed);
        this.fuelCapacity = fuelCapacity;
    }

    // Implementing abstract method
    @Override
    void accelerate() {
        speed += 10;
        System.out.println(brand + " is accelerating. New speed: " + speed + " km/h");
    }

    // Method specific to Car
    void fuelInfo() {
        System.out.println("Fuel Capacity: " + fuelCapacity + " liters.");
    }
}

// Subclass: ElectricCar
class ElectricCar extends Vehicle {
    private int batteryLevel;

    // Constructor for ElectricCar
    public ElectricCar(String brand, int speed, int batteryLevel) {
        super(brand, speed);
        this.batteryLevel = batteryLevel;
    }

    // Implementing abstract method
    @Override
    void accelerate() {
        speed += 15;
        System.out.println(brand + " (Electric) is accelerating. New speed: " + speed + " km/h");
    }

    // Method specific to ElectricCar
    void batteryInfo() {
```

```
        System.out.println("Battery Level: " + batteryLevel + "%");
    }
}

// Main class to test the implementation
public class Main {
    public static void main(String[] args) {
        // Using the abstract class reference with Car instance
        Vehicle myCar = new Car("Toyota", 50, 40);
        myCar.showDetails(); // Concrete method from Vehicle
        myCar.accelerate(); // Abstract method implementation from Car
        myCar.stop(); // Final method from Vehicle
        // myCar.fuelInfo(); // Not allowed since myCar is referenced as Vehicle

        System.out.println();

        // Using the abstract class reference with ElectricCar instance
        Vehicle myElectricCar = new ElectricCar("Tesla", 60, 80);
        myElectricCar.showDetails();
        myElectricCar.accelerate();
        myElectricCar.stop();

        System.out.println();

        // Static method from abstract class
        Vehicle.vehicleInfo();
    }
}
```

Output:

```
Brand: Toyota, Speed: 50 km/h
Toyota is accelerating. New speed: 60 km/h
Vehicle is stopping...

Brand: Tesla, Speed: 60 km/h
Tesla (Electric) is accelerating. New speed: 75 km/h
Vehicle is stopping...

Vehicles are used for transportation.
```

Interface

Interface in Java

An **interface** is a contract that classes must follow.
It contains:

- **Only abstract methods (before Java 8)**
- **Static methods (from Java 8)**
- **Default methods (from Java 8)**
- **Private methods (from Java 9)**

```
interface Vehicle {
   void start(); // Abstract method
}

interface Electric {
   void chargeBattery(); // Another abstract method
}

// A class implementing multiple interfaces
class ElectricCar implements Vehicle, Electric {
   @Override
   public void start() {
      System.out.println("Electric car is starting...");
   }

   @Override
   public void chargeBattery() {
      System.out.println("Charging the car battery...");
   }
}

public class Main {
   public static void main(String[] args) {
      ElectricCar myCar = new ElectricCar();
      myCar.start(); // Calls start() from Vehicle interface
      myCar.chargeBattery(); // Calls chargeBattery() from Electric interface
   }
}
```

Example

```
// Defining an interface with various method types
interface Vehicle {
   // Abstract method (Before Java 8, all methods were abstract)
   void start();

   // Default method (From Java 8)
   default void fuelEfficiency() {
```

```
        System.out.println("This vehicle has standard fuel efficiency.");
        logVehicleType(); // Calling private method inside the interface
    }

    // Static method (From Java 8)
    static void generalInfo() {
        System.out.println("Vehicles are used for transportation.");
    }

    // Private method (From Java 9) - Can only be used inside the interface
    private void logVehicleType() {
        System.out.println("Logging vehicle type...");
    }
}

// Implementing class: Car
class Car implements Vehicle {
    @Override
    public void start() {
        System.out.println("Car is starting...");
    }

    // Overriding default method (Optional)
    @Override
    public void fuelEfficiency() {
        System.out.println("Car has high fuel efficiency.");
    }
}

// Implementing class: Bike
class Bike implements Vehicle {
    @Override
    public void start() {
        System.out.println("Bike is starting...");
    }
}

public class Main {
    public static void main(String[] args) {
        Vehicle myCar = new Car();
        myCar.start(); // Calls abstract method implementation in Car
        myCar.fuelEfficiency(); // Calls overridden default method in Car

        System.out.println();

        Vehicle myBike = new Bike();
        myBike.start(); // Calls abstract method implementation in Bike
        myBike.fuelEfficiency(); // Calls default method from Vehicle (not overridden)
```

```
        System.out.println();

        // Calling static method from the interface
        Vehicle.generalInfo();
    }
}
```

Explanation

1. Abstract Method (void start())
    - Must be implemented by all implementing classes (Car, Bike).
2. Default Method (default void fuelEfficiency())
    - Can be inherited by implementing classes.
    - Can be overridden (as done in Car).
    - Calls a private method inside the interface.
3. Static Method (static void generalInfo())
    - Belongs to the interface and cannot be overridden.
    - Called using Vehicle.generalInfo();.
4. Private Method (private void logVehicleType())
    - Only accessible within the interface.
    - Used inside the default method.

```
class Foo {
    private int i;
    public void f() { System.out.println("Foo's f() method"); }
    public void g() { System.out.println("Foo's g() method"); }
}

class Bar extends Foo {
    public int j;
    @Override
    public void g() { System.out.println("Bar's g() method"); }
}

public class Main {
    public static void main(String[] args) {
        Foo a = new Bar();
        Bar b = new Bar();

        // (1) INSERT STATEMENT HERE
        b.f(); // Inherited from Foo
        b.g(); // Calls overridden method in Bar
        ((Bar) a).j = 10; // Downcasting to access Bar-specific variable
        System.out.println(((Bar) a).j);
```

```
        a = b; // Upcasting: b (Bar) assigned to a (Foo)
        a.g(); // Calls Bar's overridden method
    }
}
```

Output:
Foo's f() method
Bar's g() method
10
Bar's g() method

Q: Given Classes:

```
class A {
    void doIt() { System.out.println("A's doIt()"); }
}

class B extends A {
    void doIt() { System.out.println("B's doIt()"); }
}

class C extends B {
    void doIt() { System.out.println("C's doIt()"); }

    void callUp() {
        // ((A) this).doIt(); // Calls doIt() from A
    }
}

public class Main {
    public static void main(String[] args) {
        C obj = new C();
        obj.callUp();
    }
}
```

Insert the expression that would call the doIt() method in A.
Select the one correct answer.
1.doIt();
2.super.doIt();
3.super.super.doIt();
4.this.super.doIt();
5.A.this.doIt();
6.((A) this).doIt();
7.It is not possible.

Q1: What will be the output of the following code?

```
abstract class A {
   int x = 10;
   abstract void display();
}

interface B {
   int x = 20;
   void show();
}

class C extends A implements B {
   public void display() {
      System.out.println("Class C: " + x);
   }

   public void show() {
      System.out.println("Interface B: " + B.x);
   }

   public static void main(String[] args) {
      C obj = new C();
      obj.display();
      obj.show();
   }
}
```

Ans: error

Q2: predict the output

```
abstract class A {
   int x = 10; // Instance variable of class A
   abstract void display();
}

interface B {
   int x = 20; // Public, static, and final (by default in an interface)
   void show();
}

class C extends A implements B {
   public void display() {
      System.out.println("Class C: " + super.x); // Referring to A's x
   }
```

```
    public void show() {
        System.out.println("Interface B: " + B.x); // Referring to B's x explicitly
    }

    public static void main(String[] args) {
        C obj = new C();
        obj.display();
        obj.show();
    }
}
```

Output:
Class C: 10
Interface B: 20

Q: Which of the following statements about abstract classes and interfaces is correct?

A. An abstract class can implement an interface without providing implementation for all methods.
B. An interface can extend multiple abstract classes.
C. An abstract class cannot have non-abstract methods.
D. An interface can have constructor(s).

Q: Predict the output

```
interface X {
    void method1();

    default void method2() { //
        System.out.println("X: method2");
    }
}

abstract class Y {
    abstract void method1();

    public void method2() {
        System.out.println("Y: method2");
    }
}

class Z extends Y implements X {
    public void method1() {
        System.out.println("Z: method1");
    }

    public static void main(String[] args) {
```

```
    Z obj = new Z();
    obj.method1();
    obj.method2(); // Y's method2() is called due to class-over-interface rule
  }
}
```

Output:
Z: method1
Y: method2

Q: Predict the output

```
abstract class Parent {
  Parent() {
    System.out.println("Parent Constructor");
  }
  abstract void method();
}

class Child extends Parent {
  Child() {
    System.out.println("Child Constructor");
  }
  void method() {
    System.out.println("Child Method");
  }
  public static void main(String[] args) {
    Parent obj = new Child();
    obj.method();
  }
}
```

Output:
Parent Constructor
Child Constructor
Child Method

Q: Predict the output

What will be printed when the following program is compiled and run?

```
class Super{
        public int getNumber( int a){
                return 2;
        }
}
public class SubClass extends Super {
        public int getNumber( int a, char ch){
                return 4;
```

```
        }
public static void main(String[] args){
        System.out.println( new SubClass().getNumber(4) );
}
}
```

Ans: 2

Q: What will be the output of the following program ?

```
class CorbaComponent{
        String ior;
        CorbaComponent(){ startUp("IOR"); }
        void startUp(String s){ ior = s; }
        void print(){ System.out.println(ior); }
}
class OrderManager extends CorbaComponent{
        OrderManager(){ }
        void startUp(String s){ ior = getIORFromURL(s); }
        String getIORFromURL(String s){ return "URL://"+s; }
}
public class Application{
public static void main(String args[]){ start(new OrderManager()); }
        static void start(CorbaComponent cc){ cc.print(); }
}
```

Ans: It will print URL://IOR

Q: Which of the following statements can be inserted at // 1 to make the code compile without errors?

```
public class InitTest{
static int si = 10;
int i;
final boolean bool;
// 1
}
```

Q: What will the following program print?

```
public class InitTest{
public InitTest(){
s1 = sM1("1");
}
static String s1 = sM1("a");
String s3 = sM1("2");{
s1 = sM1("3");
}
static{
```

```
s1 = sM1("b");
}
static String s2 = sM1("c");
String s4 = sM1("4");
public static void main(String args[]){
InitTest it = new InitTest();
}
private static String sM1(String s){
System.out.println(s); return s;
}
}
```
Ans: It will print : a b c 2 3 4 1

Q: Given:
```
//In file AccessTest.java
package a;
public class AccessTest {
int a;
private int b;
protected void c(){ }
public int d(){ return 0; }
}
//In file AccessTester.java
package b;
import a.AccessTest;
public class AccessTester extends AccessTest{
public static void main(String[] args) {
AccessTest ref = new AccessTest();
}
}
```
Ans: Only d() can be accessed by ref.

Q: Given:
```
class A {
   public A() { }
   public A(int i) {
      System.out.println(i);
   }
}

class B {
   static A s1 = new A(1); // Static variable, initialized when class B is loaded
   A a = new A(2); // Instance variable, initialized when an object of B is created

   public static void main(String[] args) {
      B b = new B();  // Object creation → instance variable `a` gets initialized
      A a = new A(3); // Local variable `a`, prints 3
   }
```

```
    static A s2 = new A(4); // Static variable, initialized when class B is loaded
}
```

Ans: 1 4 2 3