iamneo

Interfaces

1

iamneo

What are Interfaces?

- An interface in Java is a reference type that resembles a class, primarily serving as a collection of abstract methods.
- A class that implements an interface must implement all the methods declared in the interface.
- Besides abstract methods, an interface can include constants, default methods, static methods, and nested types.
- The structure of writing an interface is akin to defining a class. While a class specifies attributes and behaviors of an object, an interface outlines behaviors that a class should implement.
- An essential point is that unless the implementing class is abstract, it must provide concrete definitions for all the methods declared in the interface.

2

iamneo

Properties of Interfaces

- An interface in Java cannot be instantiated directly.
- Interfaces do not have constructors.
- An interface can contain any number of methods.
- All of the methods in an interface are abstract.
- An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
- An interface is not extended by a class; it is implemented by a class.
- An interface can extend multiple interfaces

3

iamneo

Advantages of Interfaces

- Interfaces facilitate total abstraction in Java.
- In scenarios where multiple inheritances are not supported by Java classes, interfaces come into play to achieve this capability.
- While a class can extend only one class, it can implement an unlimited number of interfaces.
- Interfaces contribute to achieving loose coupling in Java.
- Abstraction implementation finds its utility through the utilization of interfaces in Java.

4

iamneo

Disadvantages of Interfaces

- Interfaces expose their member variables since they must be public. This could lead to some issues with respect to predicting the behavior of your code and testing(and the same reasons why one should avoid making class member variables public).
- Since an interface can be thought of as a contract implemented by multiple classes, in certain cases, modifying the interface could lead to unpredictable behavior for the classes implementing them.
- If interfaces are not designed carefully, one interface could contain many methods that might not all be needed for a class. Hence, these methods may end up having an empty implementation, which may make your code verbose. To overcome this, interfaces must be designed carefully, keeping the Interface Segregation Principle in mind.

5

iamneo

Declaring Interfaces

- The interface keyword is utilized to declare an interface. Here is a straightforward example of declaring an interface:

```
// You are using Java
import java.lang.*;
// Any number of import statements

public interface NameOfInterface {
   // Any number of final, static fields
   // Any number of abstract method declarations\
}
```

- An interface is implicitly abstract. You do not need to use the abstract keyword while declaring an interface.
- Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
- Methods in an interface are implicitly public.

6

iamneo

Implementing an Interface:

- Like abstract classes, we cannot create objects of interfaces.
- To use an interface, other classes must implement it. We use the implements keyword to implement an interface.

```
interface Polygon {
  void getArea(int length, int breadth);
}

// implement the Polygon interface
class Rectangle implements Polygon {

  // implementation of abstract method
  public void getArea(int length, int breadth) {
```

Output

```
The area of the rectangle is 30
```

```
    System.out.println("The area of the rectangle is " + (length * breadth));
  }
}

class Main {
  public static void main(String[] args) {
    Rectangle r1 = new Rectangle();
    r1.getArea(5, 6);
  }
}
```

- In the above example, we have created an interface named Polygon. The interface contains an abstract method getArea(). Here, the Rectangle class implements Polygon. And, provides the implementation of the getArea() method.

7

Implementing Multiple Interfaces:

iamneo

- In Java, a class can also implement multiple interfaces. For example,

```
interface A {
  // members of A
}

interface B {
  // members of B
}

class C implements A, B {
  // abstract members of A
  // abstract members of B
}
```

- Two interfaces, A and B, are defined. These interfaces can contain method declarations (abstract methods) and other types of members.
- When a class implements multiple interfaces (in this case, both A and B), it is required to provide concrete implementations for all abstract methods declared in each interface.
- Class C is inheriting and implementing the behaviors specified by both interfaces A and B.

8

Example 1

iamneo

- Consider the following Java program given below.

```
interface Vehicle {
    void start();
    void stop();
}
// Implement the Vehicle interface in the Car class
class Car implements Vehicle {
    public void start() {
        System.out.println("Car is starting...");
    }
    public void stop() {
        System.out.println("Car is stopping...");
    }
}
// Implement the Vehicle interface in the Bike class
class Bike implements Vehicle {
    public void start() {
        System.out.println("Bike is starting...");
    }
    public void stop() {
        System.out.println("Bike is stopping...");
    }
}
class Main {
    public static void main(String[] args) {
        // Create objects of Car and Bike
        Car myCar = new Car();
        Bike myBike = new Bike();
        // Start and stop the Car and Bike
        myCar.start();
        myCar.stop();
        myBike.start();
        myBike.stop();
    }
}
```

9

Example 2

iamneo

- Consider the following Java program given below.

```
// Interface representing shapes
interface Shape {
    void draw();  // Abstract method for drawing a shape
}
// Interface representing colors
interface Color {
    void setColor(String color);  // Abstract method for setting the color
}

// Class implementing both Shape and Color interfaces
class ColoredShape implements Shape, Color {
    private String color;
    // Implementation of the draw method from the Shape interface
    public void draw() {
        System.out.println("Drawing a colored shape");
    }
    // Implementation of the setColor method from the Color interface
    public void setColor(String color) {
        this.color = color;
        System.out.println("Setting color to: " + color);
    }
    public void displayInfo() {
        System.out.println("Colored shape information");
    }
}
public class Main {
    public static void main(String[] args) {
        ColoredShape coloredShape = new ColoredShape();
```

Output

```
Drawing a colored shape
Setting color to: Red
Colored shape information
```

```
        // Invoking methods from both Shape and Color interfaces
        coloredShape.draw();
        coloredShape.setColor("Red");
        coloredShape.displayInfo();
    }
}
```

10

iamneo

Extending an Interface:

- Similar to classes, interfaces can extend other interfaces. The extends keyword is used for extending interfaces. For example,

```
interface Line {
  // members of Line interface
}

// extending interface
interface Polygon extends Line {
  // members of Polygon interface
  // members of Line interface
}
```

- Here, the Polygon interface extends the Line interface. Now, if any class implements Polygon, it should provide implementations for all the abstract methods of both Line and Polygon

11

iamneo

Extending Multiple Interfaces:

- A Java class can only extend one parent class. Multiple inheritance is not allowed. Interfaces are not classes, however, and an interface can extend more than one parent interface.
- The extends keyword is used once, and the parent interfaces are declared in a comma-separated list.
- Example:

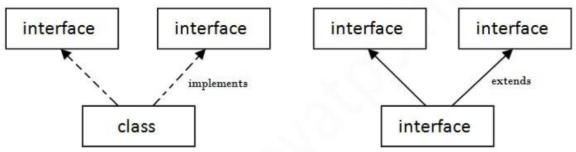
```
interface A {
   ...
}
interface B {
   ...
}

interface C extends A, B {
   ...
}
```

12

iamneo

Multiple Inheritance in Java by Interface

- If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.



Multiple Inheritance in Java

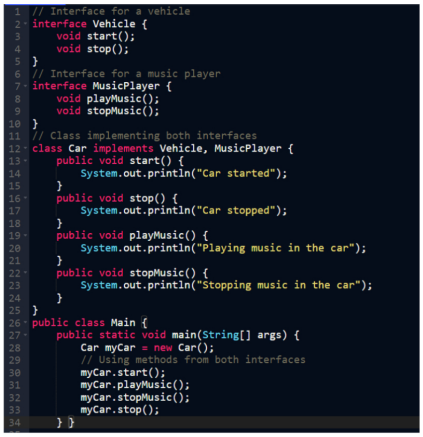
- Unlike classes, which support single inheritance (i.e., a class can extend only one superclass), a class can implement multiple interfaces. This allows the class to inherit and implement behaviors from multiple sources.

13

iamneo

Multiple Inheritance Example

- Consider the following Java program given below.



```java
// Interface for a vehicle
interface Vehicle {
    void start();
    void stop();
}
// Interface for a music player
interface MusicPlayer {
    void playMusic();
    void stopMusic();
}
// Class implementing both interfaces
class Car implements Vehicle, MusicPlayer {
    public void start() {
        System.out.println("Car started");
    }
    public void stop() {
        System.out.println("Car stopped");
    }
    public void playMusic() {
        System.out.println("Playing music in the car");
    }
    public void stopMusic() {
        System.out.println("Stopping music in the car");
    }
}
public class Main {
    public static void main(String[] args) {
        Car myCar = new Car();
        // Using methods from both interfaces
        myCar.start();
        myCar.playMusic();
        myCar.stopMusic();
        myCar.stop();
    } }
```