**Java Interface**

An **interface** in Java is a blueprint of a class that contains only method declarations (without implementations) and constant variables (static and final). It is used to achieve abstraction and multiple inheritance in Java.

**Key Points of Interface:**

* Methods in an interface are **implicitly public and abstract**.
* Fields (variables) in an interface are **implicitly public, static, and final**.
* An interface cannot have constructors since it cannot be instantiated.
* A class implements an interface using the implements keyword.
* From Java 8 onwards, interfaces can have **default methods** (with implementations) and **static methods**.
* From Java 9 onwards, interfaces can also have **private methods**.

**Example:**

|  |
| --- |
| public interface Vehicle  {  void start(); // Abstract method (implicitly public)  }  public class Car implements Vehicle  {  public void start()  {  System.out.println("Car is starting...");  }  }  public class Main  {  public static void main(String[] args)  {  Vehicle myCar = new Car();  myCar.start(); // Output: Car is starting...  }  } |

**Java Abstract Class**

An **abstract class** in Java is a class that cannot be instantiated and may contain **abstract methods** (methods without implementations) as well as **concrete methods** (methods with implementations). It is used when we want to define a common base class with shared behavior but leave some methods to be implemented by subclasses.

**Key Points of Abstract Class:**

* It can have both **abstract methods (without body)** and **concrete methods (with implementation)**.
* It **cannot be instantiated**.
* It can have **constructors**.
* It supports **instance variables** (unlike interfaces, which only support constants).
* A subclass must provide implementations for all **abstract methods**, or it must also be declared **abstract**.
* A class extends an abstract class using the extends keyword.

**Example:**

|  |
| --- |
| Public abstract class Animal  {  abstract void makeSound(); // Abstract method (must be implemented by subclasses)    public void sleep() // Concrete method  {  System.out.println("Sleeping...");  }  }  Public class Dog extends Animal  {  Public void makeSound()  {  System.out.println("Bark! Bark!");  }  }  public class Main  {  public static void main(String[] args)  {  Animal myDog = new Dog();  myDog.makeSound(); // Output: Bark! Bark!  myDog.sleep(); // Output: Sleeping...  }  } |

**Difference Between Interface and Abstract Class:**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Interface** | **Abstract Class** |
| Method Type | Only abstract (until Java 8) | Can have abstract and concrete methods |
| Fields | Only public static final constants | Can have instance variables |
| Constructor | Not allowed | Allowed |
| Multiple Inheritance | Can implement multiple interfaces | Can extend only one abstract class |
| Default Methods | Supported from Java 8 | Not needed, since methods can have implementations |
| Access Modifiers | Methods are public by default | Can have any access modifier |