### **1. What is an Interface in Java?**

In Java, an **interface** is a reference type that can contain only constants, method signatures, default methods, static methods, and nested types. Interfaces specify what a class must do, but not how. They are used to achieve abstraction and multiple inheritance in Java since a class can implement multiple interfaces but can extend only one class.

### **2. Which Modifiers are Allowed for Methods in an Interface?**

In Java 6, methods in an interface are implicitly **public** and **abstract**. You cannot use other modifiers like private, protected, or final for interface methods. This means that all methods defined in an interface are abstract by default (i.e., they only have method signatures and no implementation).

Example:

interface Vehicle {

void start(); // implicitly public and abstract

void stop(); // implicitly public and abstract

}

In Java 8 and beyond, interfaces can also contain **default** and **static** methods with concrete implementations.

Example (Java 8+):

interface Vehicle {

void start(); // implicitly public and abstract

default void honk() { // Default method with implementation

System.out.println("Honking!");

}

static void info() { // Static method with implementation

System.out.println("Vehicle Information");

}

}

### **3. What is the Use of Interface in Java?**

Interfaces are used in Java to:

* Achieve **abstraction** by specifying only the "what" of behaviors without the "how".
* Implement **multiple inheritance**, as a class can implement multiple interfaces.
* Establish a **contract** for classes, ensuring they provide specific functionalities defined in the interface.
* Promote **loose coupling**, as code can interact with interface types rather than specific class implementations.

Example:

interface Animal {

void sound();

}

class Dog implements Animal {

public void sound() {

System.out.println("Barks");

}

}

### **4. What is the Difference Between Abstract Class and Interface in Java?**

| **Feature** | **Abstract Class** | **Interface** |
| --- | --- | --- |
| **Method Implementation** | Can have both abstract and concrete methods | Only abstract methods (Java 6) |
| **Multiple Inheritance** | A class can extend only one abstract class | A class can implement multiple interfaces |
| **Constructors** | Can have constructors | Cannot have constructors |
| **Access Modifiers for Methods** | Can be any access modifier (public, protected, private, etc.) | Methods are implicitly public and abstract (Java 6) |
| **Use Case** | Partial abstraction and when shared state/fields are required | Pure abstraction and defining a contract |