1. What is an Interface in Java?

An **interface** in Java is a **blueprint** for a class that contains **only abstract methods (before Java 8)** and **static/final variables**. It is used for achieving **100% abstraction** and **multiple inheritance**.

Example:

java
Copy code

```
interface Animal {
    void makeSound(); // Abstract method (no body)
}

class Dog implements Animal {
    public void makeSound() {
        System.out.println("Dog barks");
    }
}

public class InterfaceExample {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.makeSound();
    }
}
```

Output:

nginx
Copy code

```
Dog barks
```

---

2. Which Modifiers are Allowed for Methods in an Interface? Explain with an Example.

Before Java 8, all methods inside an interface were **implicitly public and abstract**. However, from **Java 8 onwards**, the following method modifiers are allowed in an interface:

Modifier	Description
`public`	Methods in an interface are always **public** by default.
`abstract`	Interface methods are **abstract** by default (before Java 8).
`default`	Allows methods with implementation inside the interface (introduced in Java 8).
`static`	Allows static methods inside interfaces (introduced in Java 8).
`private`	Private methods can be used inside the interface (introduced in Java 9).

Example:

java
Copy code
```
interface Vehicle {
    void start(); // Public & Abstract by default

    default void honk() { // Default method (Java 8+)
        System.out.println("Honking...");
    }

    static void stop() { // Static method (Java 8+)
        System.out.println("Vehicle stopped.");
    }
}

class Car implements Vehicle {
    public void start() {
        System.out.println("Car started.");
    }
}

public class InterfaceModifiersExample {
    public static void main(String[] args) {
        Car car = new Car();
        car.start();
        car.honk(); // Calling default method

        Vehicle.stop(); // Calling static method
    }
}
```

Output:

nginx
Copy code

```
Car started.
Honking...
Vehicle stopped.
```

---

3. What is the Use of an Interface in Java? (Why Do We Use an Interface in Java?)

Interfaces are used in Java to:

1. **Achieve Multiple Inheritance** – A class can implement multiple interfaces.
2. **Achieve 100% Abstraction** – Interfaces only define what a class should do, not how.
3. **Support Loose Coupling** – Enhances code maintainability and flexibility.
4. **Provide a Contract** – Ensures all implementing classes follow a common structure.

Example of Multiple Inheritance Using Interfaces

java
Copy code

```
interface Flyable {
    void fly();
}

interface Swimmable {
    void swim();
}
```

```
class Duck implements Flyable, Swimmable {
    public void fly() {
        System.out.println("Duck can fly.");
    }

    public void swim() {
        System.out.println("Duck can swim.");
    }
}

public class MultipleInheritanceExample {
    public static void main(String[] args) {
        Duck duck = new Duck();
        duck.fly();
        duck.swim();
    }
}
```

Output:

nginx
Copy code

```
Duck can fly.
Duck can swim.
```

---

4. What is the Difference Between an Abstract Class and an Interface?

Feature	Abstract Class	Interface
Methods	Can have both **abstract & concrete methods**	Before Java 8: Only **abstract methods** Java 8+: Can have **default & static methods**
Access Modifiers	Methods can have **any access modifier**	Methods are **public** by default

Fields (Variables)	Can have **instance variables**	Can have **only static & final variables**
Multiple Inheritance	**Not supported** (A class can extend only one abstract class)	**Supported** (A class can implement multiple interfaces)
Constructors	Can have a constructor	Cannot have a constructor
Use Case	Used when classes have common behavior but some implementation is required	Used to **define behavior** but not implementation

Example of Abstract Class

java
Copy code

```
abstract class Animal {
    abstract void makeSound(); // Abstract method

    void sleep() { // Concrete method
        System.out.println("Sleeping...");
    }
}

class Dog extends Animal {
    public void makeSound() {
        System.out.println("Dog barks");
    }
}

public class AbstractClassExample {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.makeSound();
        dog.sleep();
    }
}
```

Output:

nginx

Copy code
```
Dog barks
Sleeping...
```

Example of Interface

java
Copy code
```java
interface Animal {
    void makeSound();
}

class Dog implements Animal {
    public void makeSound() {
        System.out.println("Dog barks");
    }
}

public class InterfaceExample {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.makeSound();
    }
}
```

Output:

nginx
Copy code
```
Dog barks
```