**1. What is an Interface in Java:**

An interface in Java is a reference type, similar to a class, that can contain only constants, method signatures, default methods, static methods, and nested types. Interfaces cannot contain instance fields or constructors, and they provide a way to achieve abstraction and multiple inheritance in Java.

Example:

```
public interface Animal {
    void sound(); // Abstract method
}

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

    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.sound(); // Output: Bark
    }
}
```

**2. Which Modifiers Are Allowed for Methods in an Interface? Explain with an Example:**

In an interface, methods can have the following modifiers:

- `public`: All methods in an interface are implicitly public.
- `abstract`: Methods are abstract by default, meaning they do not have a body.
- `default`: Allows methods to have a body in the interface, providing a default implementation.
- `static`: Methods can also be static, meaning they belong to the interface rather than any instance.

Example:

```
public interface Vehicle {
    void start(); // Implicitly public and abstract


    default void stop() {
        System.out.println("Vehicle stopped."); // Default method with
implementation
    }


    static void service() {
        System.out.println("Vehicle serviced."); // Static method
    }
}


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


    public static void main(String[] args) {
        Car car = new Car();
        car.start(); // Output: Car started.
        car.stop(); // Output: Vehicle stopped.
        Vehicle.service(); // Output: Vehicle serviced.
    }
}
```

**3. What is the Use of Interface in Java? Or, Why Do We Use an Interface in Java:**

Interfaces in Java are used to achieve abstraction, define contracts, and support multiple inheritance. They allow you to specify what a class should do without dictating how it should do it. Interfaces are particularly useful when different classes need to implement the same methods in different ways.

Example:

```
public interface Payment {
    void processPayment(double amount);
}

public class CreditCardPayment implements Payment {
    public void processPayment(double amount) {
        System.out.println("Processing credit card payment of $" +
amount);
    }
}

public class PayPalPayment implements Payment {
    public void processPayment(double amount) {
        System.out.println("Processing PayPal payment of $" + amount);
    }

    public static void main(String[] args) {
        Payment payment = new CreditCardPayment();
        payment.processPayment(100.0); // Output: Processing credit card
payment of $100.0

        payment = new PayPalPayment();
        payment.processPayment(50.0); // Output: Processing PayPal
payment of $50.0
    }
```

}

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

| **Abstract Class:** | **Interface** |
|---|---|
| Can have both abstract methods (without a body) and concrete methods (with a body). | Can only contain method signatures (abstract methods), default methods, static methods, and constants. |
| Can have instance variables and constructors. | Cannot have instance variables or constructors. |
| Supports inheritance and can be extended by classes using the `extends` keyword. | Supports multiple inheritance, allowing a class to implement multiple interfaces using the `implements` keyword. |
| Can provide a partial implementation that asses can use. | Defines a contract that implementing classes must follow. |
| Example of Abstract Class: `abstract class Animal {` `abstract void sound(); // Abstract method` `void sleep() { // Concrete method` `System.out.println("Animal is sleeping");` `}` `}` `class Dog extends Animal {` `void sound() {` `System.out.println("Bark");` | Example of Interface: `public interface Animal {` `void sound(); // Abstract method` `}` `public class Dog implements Animal {` `public void sound() {` `System.out.println("Bark");` `}` `public static void main(String[] args) {` `Dog dog = new Dog();` |

| | |
|---|---|
| `}`<br><br>`public static void main(String[] args) {`<br>`    Dog dog = new Dog();`<br>`    dog.sound(); // Output: Bark`<br>`dog.sleep(); // Output: Animal is sleeping`<br>`}` | `        dog.sound(); // Output: Bark`<br>`}}` |