**Assignment Interface**

Q1. 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 cannot contain instance fields or constructors. They are used to specify a set of methods that a class must implement, but they do not provide the implementation of these methods.

1. **Defining an Interface**: You define an interface using the interface keyword.

public interface Animal {

void eat();

void sleep();

}

1. **Implementing an Interface**: A class implements an interface using the implements keyword. The class must provide concrete implementations for all the methods declared in the interface. For example:

public class Dog implements Animal {

@Override

public void eat() {

System.out.println("Dog is eating");

}

@Override

public void sleep() {

System.out.println("Dog is sleeping");

}

}

1. **Default Methods**: Since Java 8, interfaces can also have default methods public interface Animal {

void eat();

default void breathe() {

System.out.println("Animal is breathing");

}

}

1. **Static Methods**: Interfaces can also have static methods. These methods belong to the interface itself, not to instances of the interface. For example:

public interface Animal {

static void info() {

System.out.println("This is an Animal interface");

}

}

1. **Multiple Inheritance**: Interfaces support multiple inheritance. A class can implement multiple interfaces, providing a way to achieve multiple inheritance in Java.

Interfaces are useful for defining a contract that other classes can follow, promoting loose coupling and enhancing code flexibility.

Q2. Which modifiers are allowed for methods in an Interface?Explain with an example

In Java, methods in an interface can have the following modifiers:

1. \*\*`public`\*\*: All methods in an interface are implicitly `public`, so you must explicitly declare them as `public` when implementing them in a class.

2. \*\*`default`\*\*: Introduced in Java 8, `default` methods have a default implementation. Classes implementing the interface can use this default implementation or override it.

3. \*\*`static`\*\*: Static methods belong to the interface itself rather than any instance. They cannot be overridden by implementing classes and are called on the interface, not on an instance.

4. \*\*`abstract`\*\*: By default, methods in an interface are abstract, meaning they do not have an implementation and must be implemented by the classes that use the interface. The `abstract` modifier is optional in the interface definition.

Here's an example illustrating these modifiers:

```java

public interface Animal {

// Abstract method (public and abstract by default)

void eat();

// Default method (provides a default implementation)

default void breathe() {

System.out.println("Animal is breathing");

}

// Static method (belongs to the interface, not instances)

static void info() {

System.out.println("This is the Animal interface");

}

}

public class Dog implements Animal {

@Override

public void eat() {

System.out.println("Dog is eating");

}

// Optionally override the default method

@Override

public void breathe() {

System.out.println("Dog is breathing heavily");

}

}

public class Main {

public static void main(String[] args) {

// Using static method

Animal.info();

// Using default method

Dog dog = new Dog();

dog.breathe();

dog.eat();

}

}

Q3. What is the use of interface in Java?Or, why do we use an interface in Java

Interfaces in Java are used for several important purposes:

1. **Define a Contract**: Interfaces define a contract that implementing classes must adhere to. This means any class that implements the interface must provide concrete implementations for the methods declared in the interface. This contract-based design promotes a clear separation between what operations are performed and how they are implemented.
2. **Achieve Abstraction**: Interfaces provide a way to achieve abstraction. They allow you to define methods without specifying how they are implemented, thus focusing on what should be done rather than how it should be done. This abstraction helps in reducing complexity by hiding the implementation details.
3. **Enable Multiple Inheritance**: Java does not support multiple inheritance of classes, but it does support multiple inheritance of interfaces. A class can implement multiple interfaces, allowing it to inherit the behavior from multiple sources. This provides a way to achieve a form of multiple inheritance.
4. **Decouple Code**: By programming to an interface rather than a specific implementation, you achieve loose coupling between components. This decoupling makes your code more flexible and easier to maintain, as changes in one component do not necessarily require changes in others.
5. **Support for Polymorphism**: Interfaces enable polymorphism. You can use an interface type to refer to any class that implements the interface, allowing you to write code that works with different implementations in a uniform way.
6. **Facilitate Testing and Mocking**: In unit testing, interfaces allow you to create mock implementations or stubs for testing purposes. This is useful for isolating and testing individual components without relying on their actual implementations.

Q4. What is the difference between abstract class and interface in Java?

|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| 1) Abstract class can **have abstract and non-abstract** methods. | Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also. |
| 2) Abstract class **doesn't support multiple inheritance**. | Interface **supports multiple inheritance**. |
| 3) Abstract class **can have final, non-final, static and non-static variables**. | Interface has **only static and final variables**. |
| 4) Abstract class **can provide the implementation of interface**. | Interface **can't provide the implementation of abstract class**. |
| 5) The **abstract keyword** is used to declare abstract class. | The **interface keyword** is used to declare interface. |
| 6) An **abstract class** can extend another Java class and implement multiple Java interfaces. | An **interface** can extend another Java interface only. |
| 7) An **abstract class** can be extended using keyword "extends". | An **interface** can be implemented using keyword "implements". |
| 8) A Java **abstract class** can have class members like private, protected, etc. | Members of a Java interface are public by default. |
| 9)**Example:** public abstract class Shape{ public abstract void draw(); } | **Example:** public interface Drawable{ void draw(); } |