Interfaces in Java

An interface in Java, much like abstract classes, is a mechanism to achieve abstraction. It allows us to define the structure of a class without providing the actual implementation. Interfaces focus on what a class *should* do, rather than how it should do it.

In Java, an interface is declared using the `interface` keyword, and classes implement interfaces using the `implements` keyword. Interfaces are fundamental in Java's approach to object-oriented design, enabling multiple inheritance of method signatures and helping define common behavior across different classes.

Example of an Interface

Here's a basic example of defining and implementing an interface:

```
interface Printable{
    void print();

}
class Myclass implements Printable{
    public void print(){
        System.out.println("Myclass");
    }
}


public class Test {
```

```
    public static void main(String[] args) {
        Myclass m= new Myclass();
        m.print();
    }
}
```

Explanation:

- The `Printable` interface defines an abstract method `print()`.
- The class `Myclass` implements this interface and provides the implementation of the `print()` method.
- In the `main()` method, we create an instance of `Myclass` and call the `print()` method.

Output:

```
Myclass
```

Features of Interfaces

Interfaces in Java have several important features:

- All members are public by default. Any method declared inside an interface is implicitly public.
- Data members are public, static, and final. All variables in an interface are by default public, static, and final, i.e., constants.
- All methods are abstract by default. Methods declared inside an interface are abstract unless marked as `default` or `static`.
- Default and Static methods are allowed. Interfaces can now contain concrete methods using `default` and `static` keywords.

- Multiple interface inheritance. A class can implement multiple interfaces, allowing for a form of multiple inheritance in Java.
- Interface inheritance. An interface can extend one or more interfaces.

Example of Default and Static Methods in an Interface

Java 8 introduced the ability to define `default` and `static` methods in interfaces. Default methods allow interfaces to provide some method implementations while still preserving backward compatibility. Static methods can be called on the interface itself, rather than on instances of the implementing class.

Here's an example demonstrating both:

```
interface MyInt{
    default void fun1(){
        System.out.println("fun1()");
    }
    static void fun2(){
        System.out.println("fun2()");
    }
    void fun3();

}
 class Test implements MyInt{
    public void fun3(){
        System.out.println("fun3()");
    }
    public static void main(String[] args) {
```

```
        Test t= new Test();
        t.fun1();
        MyInt.fun2();
        t.fun3();
    }
}
```

Explanation:

- The interface `MyInt` contains a `default` method `fun1()` and a `static` method `fun2()`.
- The class `Test` implements `MyInt` and provides its own implementation for the abstract method `fun3()`.
- In the `main()` method:
    - `fun1()` is called on the instance `t`, and since it is a `default` method, it works just like a regular instance method.
    - `fun2()` is a `static` method and is called on the interface itself using `MyInt.fun2()`.
    - `fun3()` is implemented in the class `Test`, and when called, it prints the respective message.

Output:

```
fun1()

fun2()

fun3()
```