Abstract Class vs Interface in Java

In Java, abstraction is used to hide the complexity of a program and only expose essential parts. Both abstract classesand interfaces are used for abstraction, but they have important differences. Here's a detailed comparison:

Similarities:

- Both abstract classes and interfaces are used to achieve abstraction.
- You cannot create instances of either an abstract class or an interface.
- You can create references to both abstract classes and interfaces.

Abstract Classes and Interfaces in Java:

Feature	Abstract Class	Interface
Constructors	Can have constructors.	Cannot have constructors.
Data Members	Can have non-static, non-final data members.	All data members are `public static final`.
Methods	Can have methods with any access modifier (`public`, `private`, `protected`).	All methods are `public` by default.

Inheritance	A class can extend only one abstract class.	A class can implement multiple interfaces.
Multiple Inheritance	Not supported (no multiple class inheritance).	Supported (an interface can extend multiple interfaces).
Implementation	Can implement interfaces and extend other classes.	Can only extend other interfaces, not classes.
Access Modifiers for Methods	Can have `private`, `protected`, and `default`methods.	Methods are `public` by default.
Non-abstract Methods	Can have regular (non-abstract) methods.	Can have `default` and `static`methods.

Key Points:

- An abstract class can implement an interface
- A normal class can extend an abstract class and can implement one or more interfaces.

Example: Abstract Class and Interface in Action

Consider the following simple example that demonstrates both an interface and an abstract class in Java:

```
interface Printable {
    void print();
}

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

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

Output:

```
MyClass
```

Here, the `Printable` interface defines an abstract method `print()`, which is implemented by the `MyClass` class. This shows how an interface can be used to define a contract that a class must follow.

When to Use Abstract Classes?

Abstract classes are typically used when:

1. There is a logical "is-a" relationship between the classes. For example, `Shape` and `Rectangle` or `Employee` and `SalesEmployee`.
2. You need protected, private, or default methods.
3. You require non-static, non-final data members.

When to Use Interfaces?

Interfaces are ideal when:

1. You need to provide a set of functionalities that can be implemented by unrelated classes. For example, a `Comparable` interface can be implemented by both a `Student` class and an `Employee` class to compare their IDs.
2. You need multiple implementations for the same functionality.
3. You need multiple inheritance, where an interface can extend multiple other interfaces.