Java OOP: Abstraction

Press Space for next page →

Agenda

Abstraction

> Abstraction is a process of **hiding the implementation details** and **showing only functionality** to the user.

For example, sending SMS where you type the text and send the message. ***You don't know the internal processing about the message delivery***.

- One misunderstanding is that the reason for **"hiding"** something is security.
  - In reality, it means hiding the complex details of how something works, just like how a car's engine is hidden under the hood. You don't need to know how the engine works to drive the car, and you can still see the engine if you want to.
- In programming, hiding the inner details of a class means users only need to know how to use it, not how it works inside. This also allows developers to change how the class works without causing problems for the users.

Ways to achieve Abstraction

1. Abstract Class **(0 to 100%)**
2. Interface **(100%)**

Abstract Class

> A class which is **declared as abstract** is known as an abstract class. It can have **abstract and non-abstract methods**. It needs to be **extended and its method implemented**. It **cannot be instantiated**.

Abstract Class Behavior

- **Common Base Implementation:**

  Abstract classes provide a common base implementation for derived classes. When you want to define a set of methods with some default behavior that can be shared among multiple related classes, an abstract class is a good choice. Derived classes can inherit the common behavior and override specific methods as needed.

- **Inheritance and Specialization:**

  If you're using inheritance, an abstract class acts as a blueprint for specialized classes. Abstract classes allow you to define methods that must be implemented by subclasses (abstract methods) alongside concrete methods with default behavior. Subclasses extend the abstract class and provide their own implementations for the abstract methods.

- **Non-Public Members:**

  Abstract classes can have non-public members (fields, methods). If you need to encapsulate certain functionality within the class hierarchy without exposing it publicly, an abstract class is suitable.

> Remember that a class can extend only one other class (abstract or not). If you anticipate needing multiple inheritance (implementing behavior from multiple sources), interfaces are a better choice.

Code Example: Abstract Class

```java
abstract class Animal {
    abstract void sound();
    void eat() {
        System.out.println("Eating...");
    }
}

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

class AbstractClassExample {
    public static void main(String[] args) {
        Animal animal = new Dog();
        animal.sound();
        animal.eat();
    }
}
```

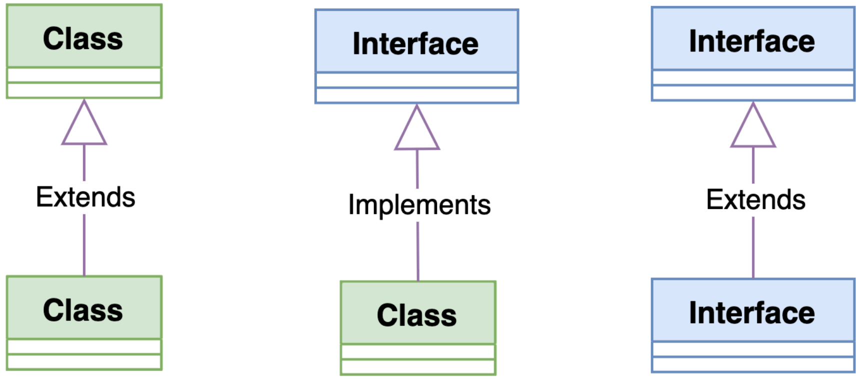
Interface

> Interface in Java is a blueprint of a class. It has static constants and abstract methods only.

> The interface in Java is a mechanism to achieve **fully abstraction**. There can be only **abstract methods** in the Java interface, not method body. It is used to achieve **fully abstraction** and **multiple inheritance** in Java.

When you want to define a clear contract for behavior without providing any implementation, interfaces are the way to go.

Relationship with Classes and Interfaces



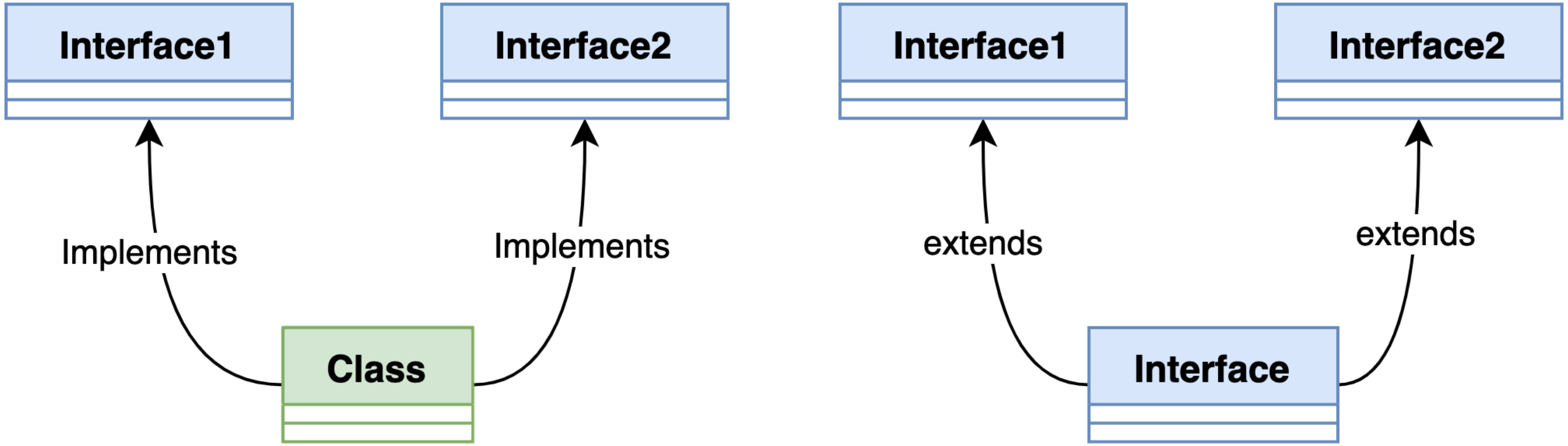
Code Example: Interface

```java
interface Animal {
    void sound();
    void eat();
}

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

class InterfaceExample {
    public static void main(String[] args) {
        Animal animal = new Dog();
        animal.sound();
        animal.eat();
    }
}
```

Multiple Inheritance in Interface



Code Example: Multiple Inheritance in Interface

```java
interface Animal {
    void sound();
}
interface Mammal {
    void eat();
}
class Dog implements Animal, Mammal {
    public void sound() {
        System.out.println("Bark");
    }
    public void eat() {
        System.out.println("Eating...");
    }
}
class InterfaceExample {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.sound();
        dog.eat();
    }
}
```

Thank you ❤️

qa-june-2024-automation-with-java-slides