VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY

UNIVERSITY OF SCIENCE

INFORMATION TECHNOLOGY FACULTY

---oOo---

SOFTWARE ANALYSIS AND DESIGN

Abstraction And Interface

Full name: Trương Anh Tuấn

Student Id: 21120589

Email: 21120589@student.hcmus.edu.vn

Course: Software Analysis And Design

Instructor: Trần Duy Thảo

Contents

I. Abstraction

- **Definition:** A class that contains an abstract keyword on the declaration is known as an abstract class. An abstract class must have at least one abstract method. It is possible in an abstract class to contain multiple concrete methods.

- **Purpose:**

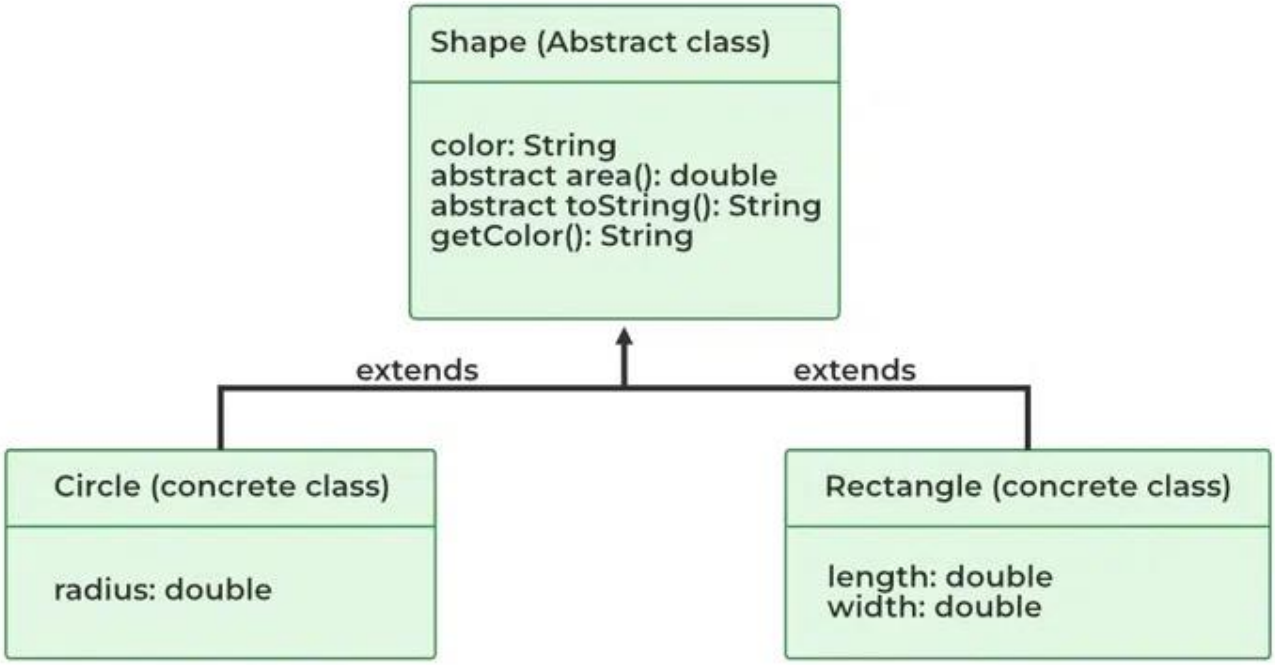
  - **Abstract classes:** abstraction is often achieved using abstract classes. An abstract class is a class that cannot be instantiated and may contain abstract methods (methods without a body) that must be implemented by concrete subclasses.
  - **Abstract methods:** Abstract methods declared in an abstract class define a contract that concrete subclasses must adhere to by providing implementations for these methods.

- **Implementation:** Abstraction can be achieved through abstract classes and methods in programming languages like **Java** and **C#**. Abstract classes may contain abstract methods that must be implemented by concrete subclasses, allowing for flexibility and customization.

- **Is-a Relationship:**

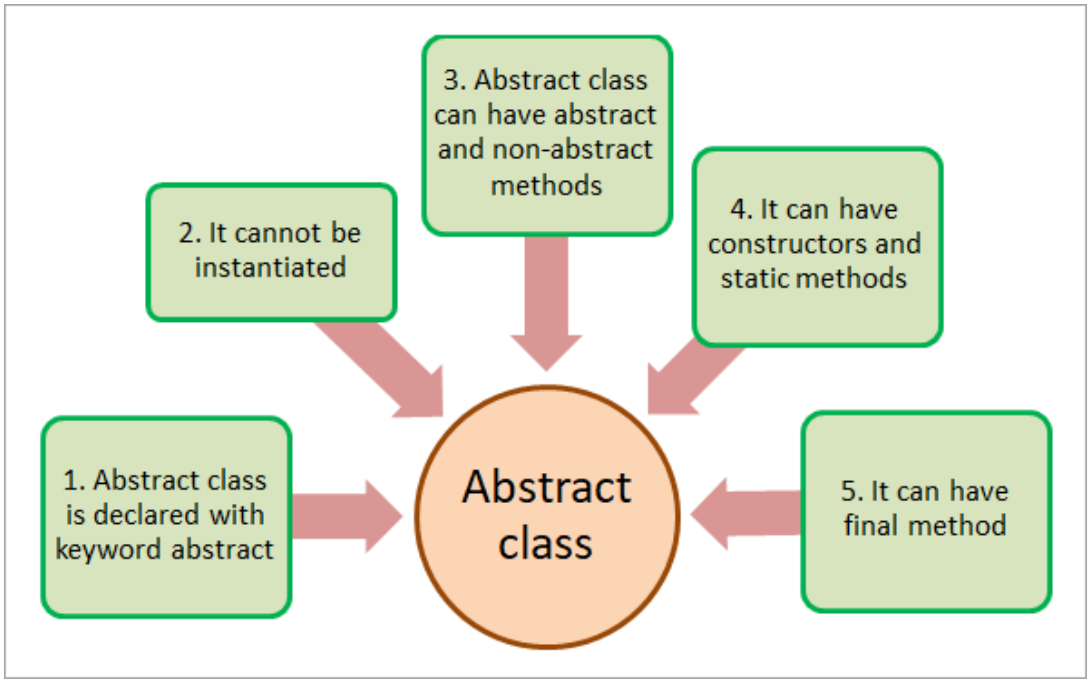
  - The **"is-a"** relationship describes inheritance, where one class is a specialization of another class.
  - It signifies a relationship where one class is a subtype of another class, implying that the subclass shares common characteristics with the superclass and may have additional specialized characteristics.
  - **Example**: If you have classes **Animal** and **Dog**, you could say that "**a Dog is an Animal**" indicating that **Dog** inherits traits and behaviors from the **Animal** class.

- Diagram:



1. *Abstraction in Java*

- Summary:



2. *Abstract class*

- **Example:** Consider a shape class with abstract methods like **calculateArea()** and **calculatePerimeter().** Concrete subclasses like **Circle** and **Square** can implement these methods according to their specific shapes:

```java
abstract class Shape {
    // Abstract method for calculating area
    abstract double calculateArea();

    // Abstract method for calculating perimeter
    abstract double calculatePerimeter();

    // Concrete method to display a message
    void displayMessage() {
        System.out.println("This is a shape.");
    }
}

// Concrete subclass Circle
class Circle extends Shape {
    double radius;

    Circle(double radius) {
        this.radius = radius;
    }

    @Override
    double calculateArea() {
        return Math.PI * radius * radius;
    }

    @Override
    double calculatePerimeter() {
        return 2 * Math.PI * radius;
    }
}
```

```java
// Concrete subclass Rectangle
class Rectangle extends Shape {
    double length;
    double width;

    Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }

    @Override
    double calculateArea() {
        return length * width;
    }

    @Override
    double calculatePerimeter() {
        return 2 * (length + width);
    }
}
```

II. Interface

- **Definition**: An interface is a sketch that is useful to implement a class. The methods used in the interface are all abstract methods. The interface does not have any concrete method.

- **Purpose**:

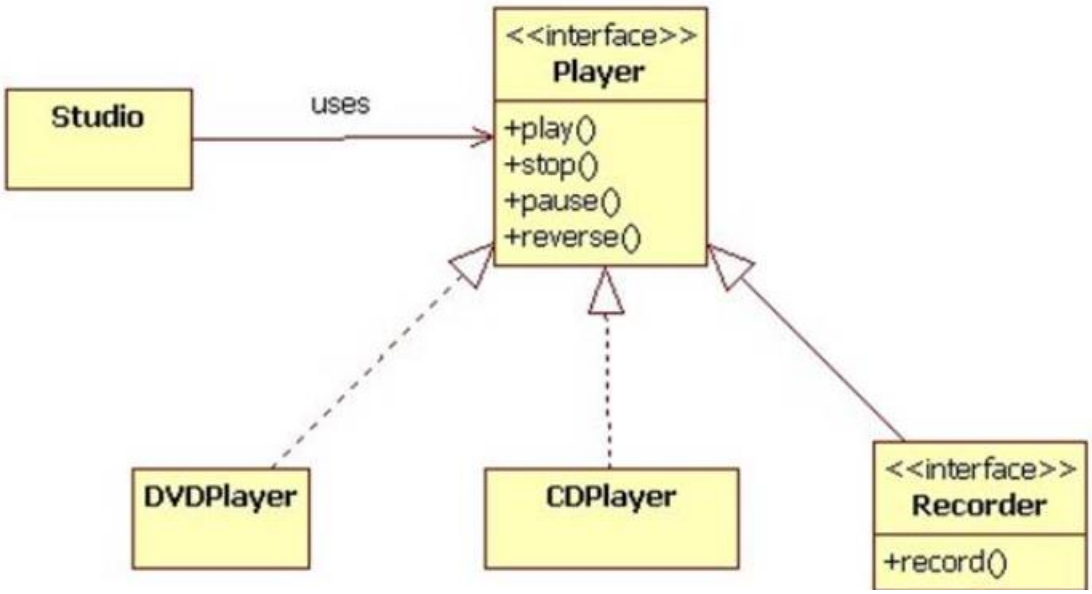
  - **Achieve multiple inheritances:** Java supports multiple interface inheritance, allowing classes to implement multiple interfaces, thus achieving a form of multiple inheritance.
  - **Enable polymorphism:** Interfaces enable polymorphism by allowing objects of different classes to be treated uniformly if they implement the same interface.
  - **Encourage loose coupling:** Interfaces promote loose coupling between components, as classes interact through interfaces rather than concrete implementations.

- **Implementation**: In languages like **Java** and **C#**, interfaces are declared using the interface keyword. Classes implementing an interface or multiple interfaces must provide implementations for all methods declared in that/those interface(s).

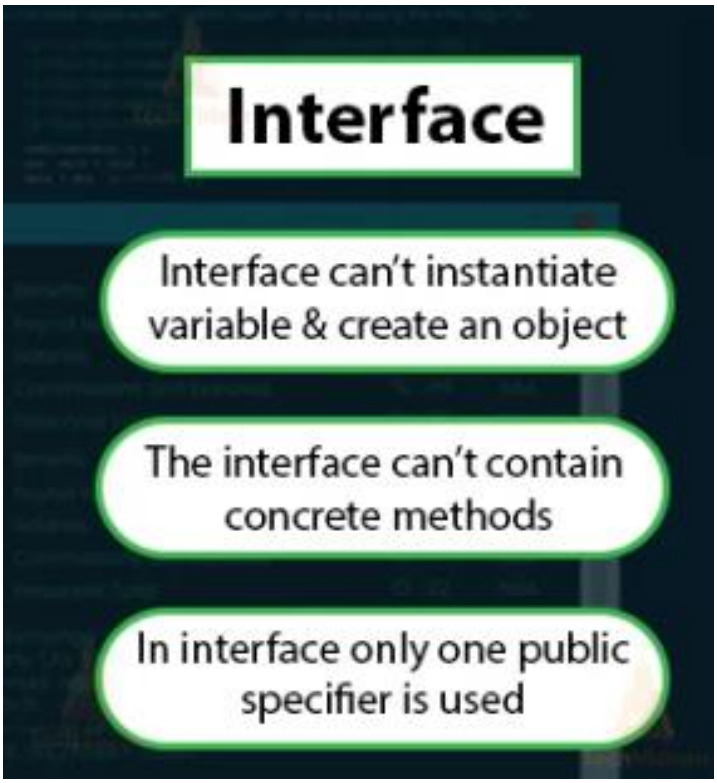
- **Can-do Relationship:**

  - The **"can-do"** relationship describes interfaces and polymorphism, where a class can perform certain actions or behaviors defined by an interface.
  - It signifies that a class can do something, as specified by an interface it implements, regardless of its inheritance hierarchy.
  - **Example:** If you have an interface **Flyable** with a method **fly(),** and a class **Bird** implements **Flyable**, you could say that **"a Bird can fly"** indicating that the Bird class provides an implementation for the **fly()** method defined in the **Flyable** interface.

- Diagram:



- Summary:



4. Interface

- **Example**: Consider a **Playable** interface with methods like **play()**, **pause()**, and stop(). Both **AudioPlayer** and **VideoPlayer** classes can implement this interface to provide different implementations for playing audio and video files.

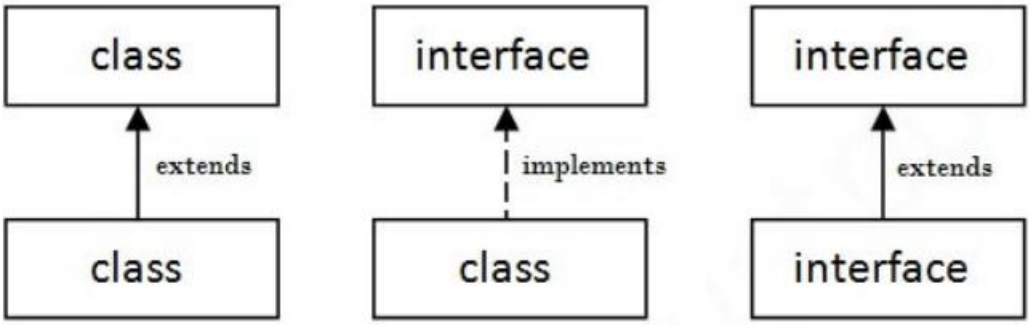
```java
//Interface definition
interface Playable {
    void play();

    void pause();
}

//Concrete class representing an audio player
class AudioPlayer implements Playable {
    String audioFile;

    AudioPlayer(String audioFile) {
        this.audioFile = audioFile;
    }

    @Override
    public void play() {
        System.out.println("Playing audio: " + audioFile);
    }

    @Override
    public void pause() {
        System.out.println("Pausing audio: " + audioFile);
    }
}
```

```java
//Concrete class representing a video player
class VideoPlayer implements Playable {
    String videoFile;

    VideoPlayer(String videoFile) {
        this.videoFile = videoFile;
    }

    @Override
    public void play() {
        System.out.println("Playing video: " + videoFile);
    }

    @Override
    public void pause() {
        System.out.println("Pausing video: " + videoFile);
    }
}
```

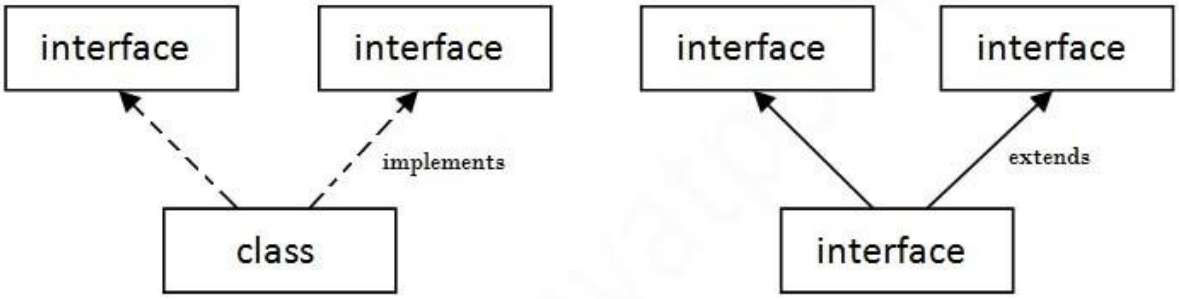
III. Comparison

Abstraction	Interface
Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods.
Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of the abstract class**.
An **abstract class** can extend another class and implement multiple interfaces.	An **interface** can extend multiple interfaces but can't extend a class.
An **abstract class** can have class members like private, protected, etc.	Members of an interface are public by default.

Example:
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}



5.1 Comparison between Abstraction and Interface



Multiple Inheritance in Java

5.2 Comparison between Abstraction and Interface

IV. References

- Java Point. *Difference between abstract class and interface?*. URL: https://www.javatpoint.com/difference-between-abstract-class-and-interface
- Nguyễn Quốc Đạt. *So sánh Interface và Abstract trong lập trình hướng đối tượng.* Feb 2017. URL: https://viblo.asia/p/so-sanh-interface-va-abstract-trong-lap-trinh-huong-doi-tuong-63vKjpk6l2R
- Nitsdheerendra. *Difference between Abstract Class and Interface in Java*, Apr 2018. URL: https://www.geeksforgeeks.org/difference-between-abstract-class-and-interface-in-java/
- Ju's Gate. *Difference between Abstract Class and Interface in Java*, Apr 2018. URL: https://byjus.com/gate/difference-between-abstract-class-and-interface-in-java/