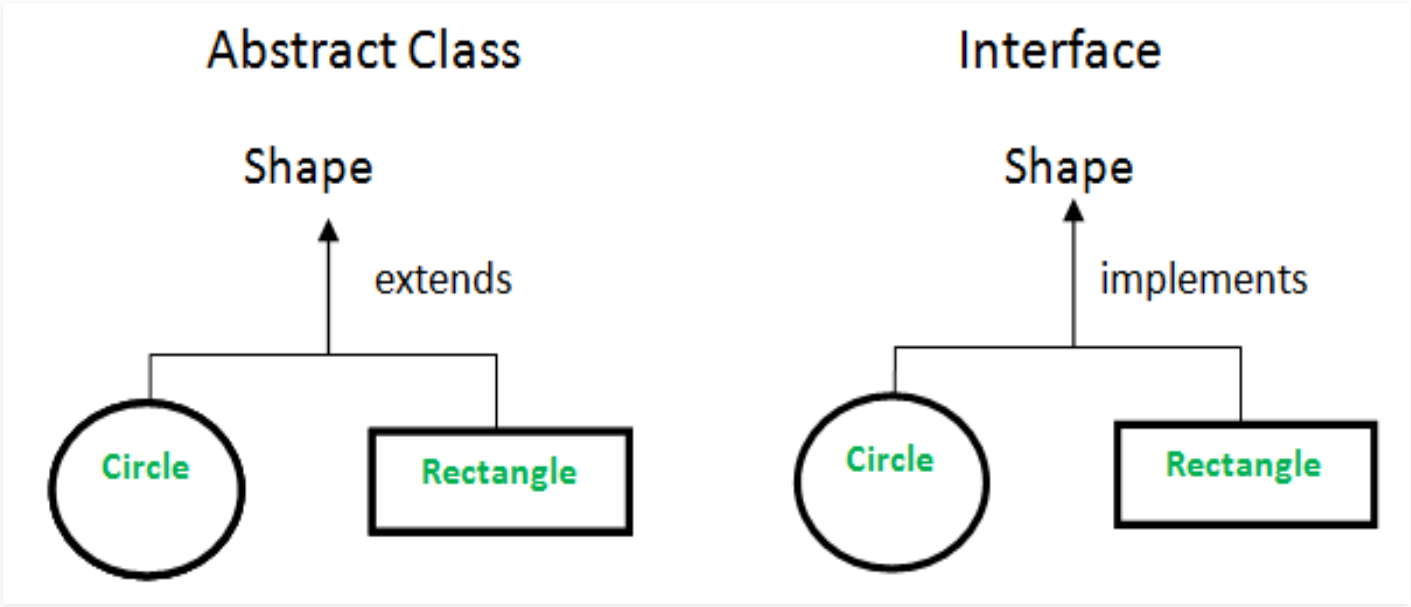
Difference between Abstract Class and Interface in Java

Difficulty Level : Easy Last Updated : 06 Dec, 2021

As we know that abstraction refers to hiding the internal implementation of the feature and only showing the functionality to the users. i.e. what it works (showing), how it works (hiding). Both abstract class and interface are used for abstraction, henceforth Interface and Abstract Class are required prerequisites



Abstract class vs Interface

- **Type of methods:** Interface can have only abstract methods. An abstract class can have abstract and non-abstract methods. From Java 8, it can have default and static methods also.
- **Final Variables:** Variables declared in a Java interface are by default final. An abstract class may contain non-final variables.
- **Type of variables:** Abstract class can have final, non-final, static and non-static variables. The interface has only static and final variables.
- **Implementation:** Abstract class can provide the implementation of the interface. Interface can't provide the implementation of an abstract class.
- **Inheritance vs Abstraction:** A Java interface can be implemented using the keyword "implements" and an abstract class can be extended using the keyword "extends".
- **Multiple implementations:** An interface can extend another Java interface only, an abstract class can extend another Java class and implement multiple Java interfaces.
- **Accessibility of Data Members:** Members of a Java interface are public by default. A

- **Accessibility of Data Members:** Members of a Java interface are public by default. A Java abstract class can have class members like private, protected, etc.

Example 1-A:

```
// Java Program to Illustrate Concept of
// Abstract Class

// Importing required classes
import java.io.*;

// Class 1
// Helper abstract class
abstract class Shape {

    // Declare fields
    String objectName = " ";

    // Constructor of this class
    Shape(String name) { this.objectName = name; }

    // Method
    // Non-abstract methods
    // Having as default implementation
    public void moveTo(int x, int y)
    {
        System.out.println(this.objectName + " "
                           + "has been moved to"
                           + " x = " + x + " and y = " + y);
    }

    // Method 2
    // Abstract methods which will be
    // implemented by its subclass(es)
    abstract public double area();
    abstract public void draw();
}

// Class 2
// Helper class extending Class 1
class Rectangle extends Shape {

    // Atributes of rectangle
    int length, width;

    // Constructor
    Rectangle(int length, int width, String name)
    {
```

```
            // Super keyword refers to current instance itself
            super(name);

            // this keyword refers to current instance itself
            this.length = length;
            this.width = width;
        }

        // Method 1
        // To draw rectangle
        @Override public void draw()
        {
            System.out.println("Rectangle has been drawn ");
        }

        // Method 2
        // To compute rectangle area
        @Override public double area()
        {
            // Length * Breadth
            return (double)(length * width);
        }
    }

    // Class 3
    // Helper class extending Class 1
    class Circle extends Shape {

        // Attributes of a Circle
        double pi = 3.14;
        int radius;

        // Constructor
        Circle(int radius, String name)
        {
            // Super keyword refers to parent class
            super(name);
            // This keyword refers to current instance itself
            this.radius = radius;
        }

        // Method 1
        // To draw circle
        @Override public void draw()
        {
            // Print statement
            System.out.println("Circle has been drawn ");
        }

        // Method 2
        // To compute circle area
        @Override public double area()
        {
            return (double)((pi * radius * radius));
        }
    }
```

```
// Class 4
// Main class
class GFG {

    // Main driver method
    public static void main(String[] args)
    {
        // Creating the Object of Rectangle class
        // and using shape class reference.
        Shape rect = new Rectangle(2, 3, "Rectangle");

        System.out.println("Area of rectangle: "
                           + rect.area());

        rect.moveTo(1, 2);

        System.out.println(" ");

        // Creating the Objects of circle class
        Shape circle = new Circle(2, "Circle");

        System.out.println("Area of circle: "
                           + circle.area());

        circle.moveTo(2, 4);
    }
}
```

Output

```
Area of rectangle: 6.0
Rectangle has been moved to x = 1 and y = 2

Area of circle: 12.56
Circle has been moved to x = 2 and y = 4
```

What if we don't have any common code between rectangle and circle then go with the interface.

Example 1-B:

```
// Java Program to Illustrate Concept of Interface

// Importing I/O classes
import java.io.*;

// Interface
interface Shape {

    // Abstract method
    void draw();
    double area();
}

// Class 1
// Helper class
class Rectangle implements Shape {

    int length, width;

    // constructor
    Rectangle(int length, int width)
    {
        this.length = length;
        this.width = width;
    }

    @Override public void draw()
    {
        System.out.println("Rectangle has been drawn ");
    }

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

// Class 2
// Helper class
class Circle implements Shape {

    double pi = 3.14;
    int radius;

    // constructor
    Circle(int radius) { this.radius = radius; }

    @Override public void draw()
    {
        System.out.println("Circle has been drawn ");
    }

    @Override public double area()
    {
```

```
            return (double)((pi * radius * radius));
        }
    }

    // Class 3
    // Main class
    class GFG {

        // Main driver method
        public static void main(String[] args)
        {
            // Creating the Object of Rectangle class
            // and using shape interface reference.
            Shape rect = new Rectangle(2, 3);

            System.out.println("Area of rectangle: "
                               + rect.area());

            // Creating the Objects of circle class
            Shape circle = new Circle(2);

            System.out.println("Area of circle: "
                               + circle.area());
        }
    }
```

Output

```
Area of rectangle: 6.0
Area of circle: 12.56
```

When to use what?

Consider using abstract classes if any of these statements apply to your situation:

- In the java application, there are some related classes that need to share some lines of code then you can put these lines of code within the abstract class and this abstract class should be extended by all these related classes.
- You can define the non-static or non-final field(s) in the abstract class so that via a method you can access and modify the state of the object to which they belong.
- You can expect that the classes that extend an abstract class have many common methods or fields, or require access modifiers other than public (such as protected and private).

Consider using interfaces if any of these statements apply to your situation:

- It is total abstraction, All methods declared within an interface must be implemented by the class(es) that implements this interface.
- A class can implement more than one interface. It is called multiple inheritances.
- You want to specify the behavior of a particular data type but are not concerned about who implements its behavior.

You can also go for **Quiz** on this topic.

This article is contributed by **Nitsdheerendra**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.