

**Green University of Bangladesh**

**Department of Computer Science and Engineering (CSE)**

**Faculty of Sciences and Engineering Semester: (Spring, Year: 2022), B.Sc. in CSE (Day)**

**Course Title: Object Oriented Programming Lab Course Code: CSE 202 Section: 213 D3**

**Lab report Name:** Difference between abstract and interface with an example using code

**Student Details**

|  |  |  |
| --- | --- | --- |
|  | **Name** | **ID** |
| **1.** | All-amin (Samir) | 213002019 |

|  |  |
| --- | --- |
| **Submission Date** | **: 13-12-22** |
| **Course Teacher’s Name** | **:** Ayesha Khatun |

**[For Teachers use only: Don’t Write Anything inside this box]**

**Lab report Status**

**Marks: ………………………………… Signature: .....................**

**Comments: .............................................. Date: ..............................**

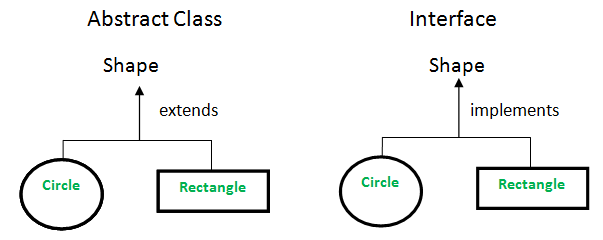
# Title: Difference between abstract and interface with an example using code.

Objective:

Explore the Difference between Abstract Class and Interface in Java. The Abstract class and Interface both are used to have abstraction. **An abstract class contains an abstract keyword on the declaration whereas an Interface is a sketch that is used to implement a class.**

Analysis.

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](https://www.geeksforgeeks.org/abstract-classes-in-java/) and [interface](https://www.geeksforgeeks.org/interfaces-in-java/) 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. From Java 9, it can have private concrete methods as well.
* **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 one or more Java interfaces; an abstract class can extend another Java class and implement multiple Java interfaces.
* **Multiple Inheritance:**  Interface supports multiple inheritance; an abstract class does not support multiple inheritance.
* **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.

Implementation

|  |
| --- |
| 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 {        // Attributes 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 a 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.

Conclusion.

he interface is a blueprint that can be used to implement a class. The interface does not contain any concrete methods (methods that have code). All the methods of an interface are abstract methods.

A class which has the abstract keyword in its declaration is called abstract class. Abstract classes should have zero or more abstract methods. i.e., methods without a body. It can have multiple concrete methods.