Spondon Ganguli
An Educator, Artist & Author

Inheritance – Part 2

By Spondon Ganguli

JAVA ABSTRACT CLASS

An abstract class is a class that is declared by using the **abstract** keyword. It may or may not have abstract methods. Abstract classes cannot be instantiated, but they can be extended into sub-classes.

Points of abstract class :

1. Abstract class contains abstract methods.
2. Program can't instantiate an abstract class.
3. Abstract classes contain mixture of non-abstract and abstract methods.
4. If any class contains abstract methods then it must implements all the abstract methods of the abstract class.

Here is a simple example of a class with an abstract method, followed by a class which implements the method:


```
// A Simple demonstration of an abstract class.
abstract class A {
        abstract void callme( ); //only the method declaration
        void callmetoo( )
        {
                System.out.println("Implementation of callmetoo method of class A.");
        }
}
class B extends A {
        void callme( ) //same method with complete implementation
        {
                System.out.println("Implementation of callme of class B.");
        }
}
class AbstractDemo {
        public static void main(String args[])
        {
                B b = new B( );
                b.callme( );
                b.callmetoo( );
        }
}
```

Output:

Implementation of **callme** method of class B
Implementation of **callmetoo** method of class A

```
// Using abstract methods and classes.
abstract class Figure
{
  double dim1;
  double dim2;
  Figure(double a, double b)
  {
        dim1 = a;
        dim2 = b;
  }
  abstract double area();  // area is now an abstract method
}

class DemoAreas {
  public static void main(String args[])
  {
        // Figure f = new Figure(10, 10); // illegal now
        Rectangle r = new Rectangle(9, 5);
        Triangle t = new Triangle(10, 8);
        Figure figref; // this is OK, no object is created
        figref = r;
        System.out.println("Area is " + figref.area());
        figref = t;
        System.out.println("Area is " + figref.area());
  }
}
```

```
class Rectangle extends Figure {
        Rectangle(double a, double b)
        {
                super(a, b);
        }
        double area( )  // override area for rectangle
        {
                System.out.println("Inside Area for Rectangle.");
                return dim1 * dim2;
        }
}

class Triangle extends Figure {
        Triangle(double a, double b)
        {
                super(a, b);
        }
        double area( )  // override area for right triangle
        {
                System.out.println("Inside Area for Triangle.");
                return dim1 * dim2 / 2.0;
        }
}
```

What is Interface?

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods. Java Interface also **represents the IS-A relationship**.

An interface is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface. An interface is not a class. Writing an interface is similar to writing a class, but they are two different concepts. A class describes the attributes and behaviors of an object. An interface contains behaviors that a class implements.

An interface is similar to a class in the following ways:

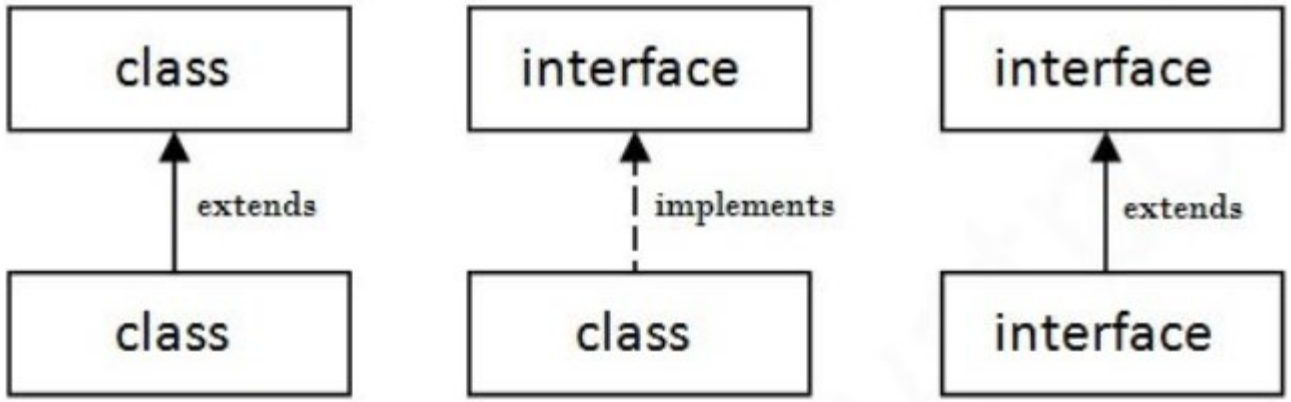
1. An interface can contain any number of methods as like a class.
2. An interface is written in a file with a **.java** extension like a class.
3. The bytecode of an interface appears in a **.class** file.

However, an interface is different from a class in several ways, including:

1. You cannot instantiate an interface.
2. An interface does not contain any constructors.
3. All of the methods in an interface are **abstract**.
4. An interface contains fields that must be declared with both **static** and **final**.
5. An interface is not extended by a class; it is implemented by a class.
6. An interface can **extend** multiple interfaces.

The relationship between classes and interfaces

As shown in the figure given below, a class extends another class, an interface extends another interface, but a **class implements an interface**.



```java
// A simple interface
interface Interface
{
        // public, static and final
        static final int a = 10; // public and abstract
        void display();
}

// A class that implements the interface.
class TestClass implements Interface
{
        // Implementing the capabilities of the interface.
        public void display()
        {
                System.out.println("Hello");
        }
        public static void main(String[] args)
        {
                TestClass tobj = new TestClass();
                tobj.display();
                System.out.println(tobj.a);
        }
}
```

Multiple inheritance in Java by interface

If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.

Multiple Inheritance in Java

An interface **Data** is defined with a data member and an abstract method compute( ). A superclass **Base** has been defined to contain the radius of a circle. Define a subclass **CalVol** which uses the properties of the interface Data and the class Base and calculate the volume of a cylinder. The details of the members of the interface and both the classes are given below:

Interface: Data
Data Member: pi =3.142
Member function: double compute()

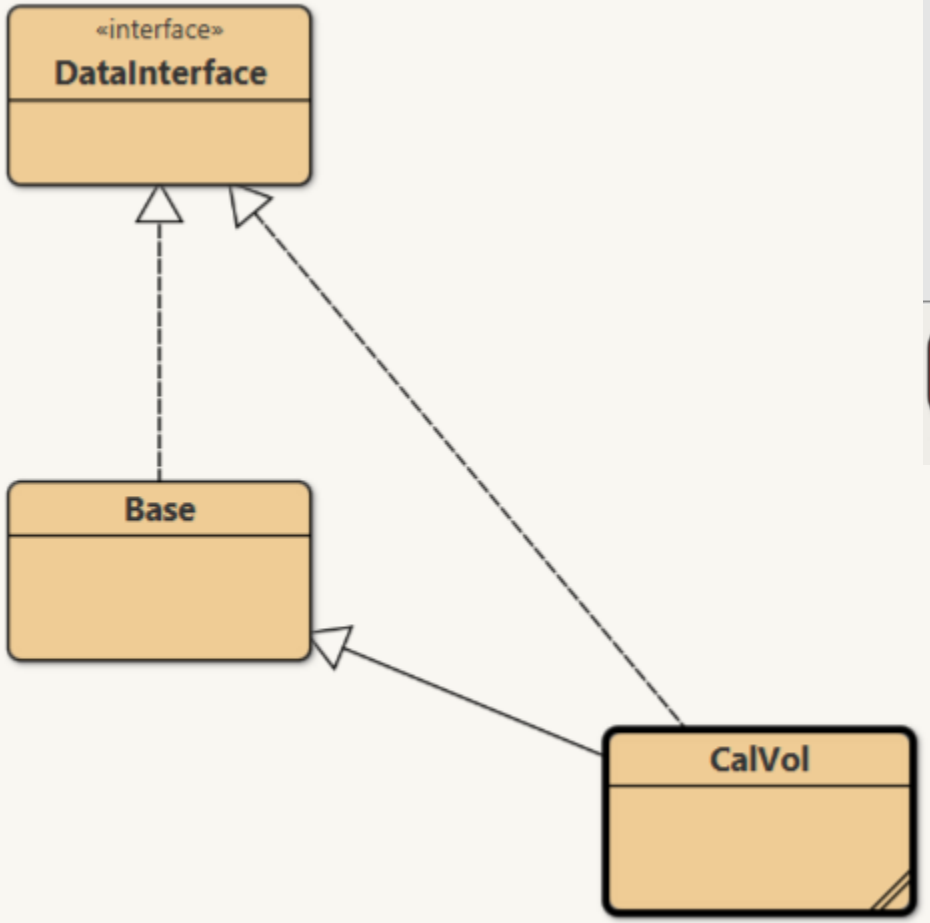
Class: Base
Data Member: r (radius)
Member function: double compute() – compute the area of a circle

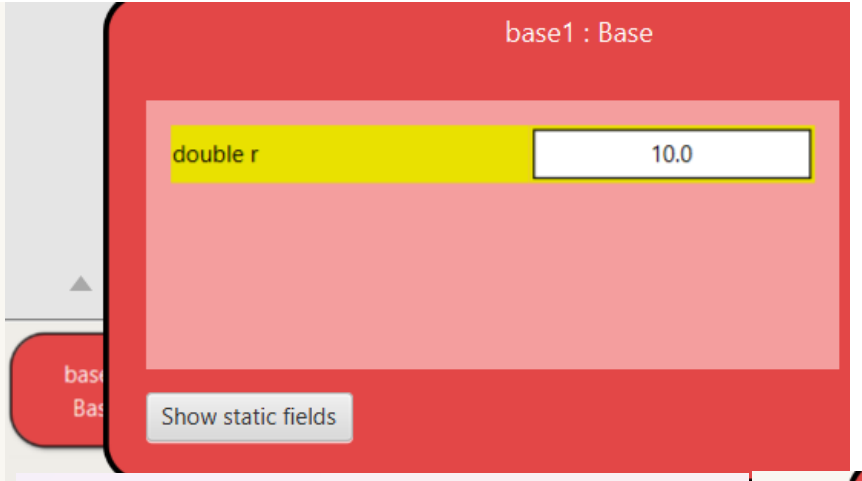
Class: CalVol
Data Member: ht (height)
Member function: double compute() – compute the volume of a cylinder

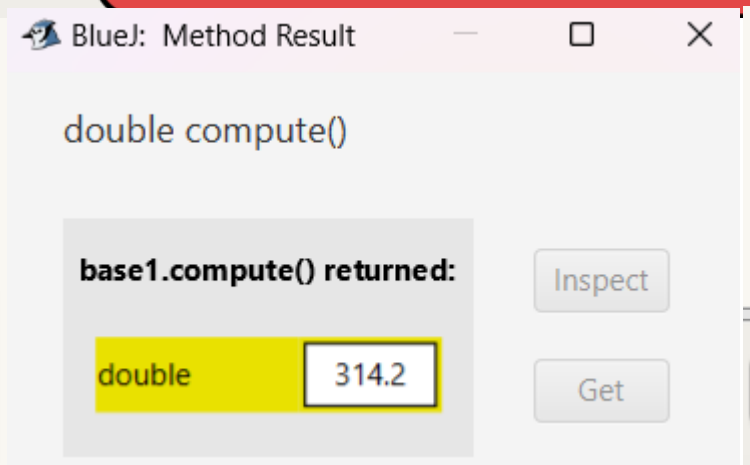
```java
public interface DataInterface
{
    final double pi=3.142;
    public double compute();
}
```

```java
class Base implements DataInterface
{
    double r;
    public Base(double a)
    {
        r=a;
    }
    public double compute()
    {
        double ar=r*r*pi;
        return ar;
    }
}
```

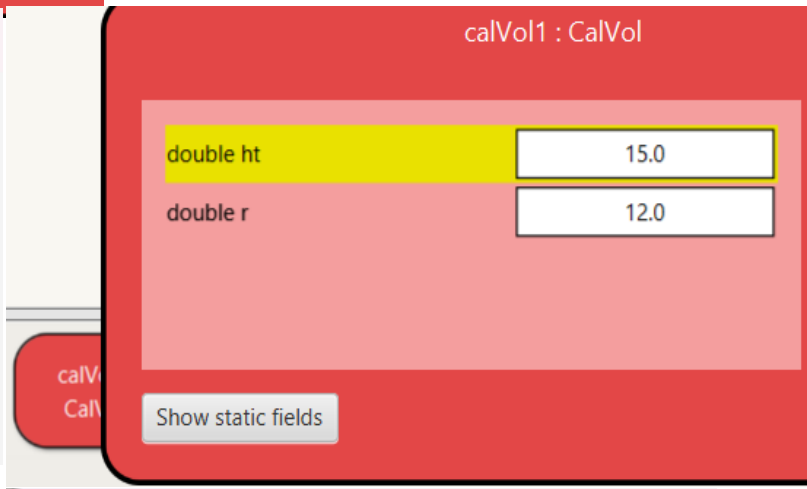
```java
class CalVol extends Base implements DataInterface
{
    double ht;
    CalVol(double a, double b)
    {
        super(a);
        ht=b;
    }
    public double compute()
    {
        double vol=ht*r*r*pi;
        return vol;
    }
}
```

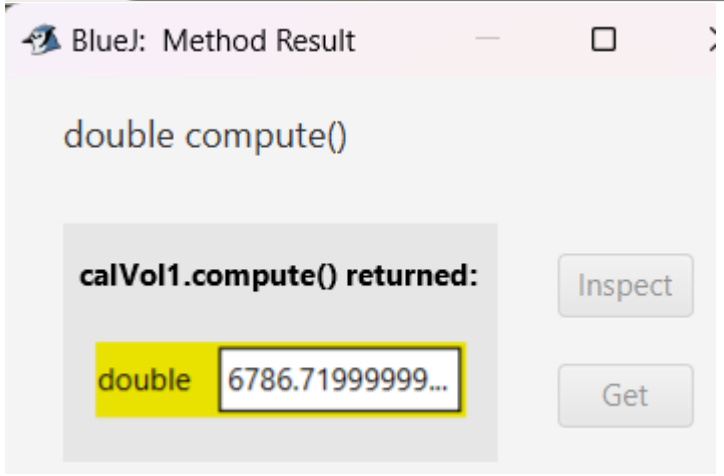






Computing the area:
= 3.142*10*10
= 312.4

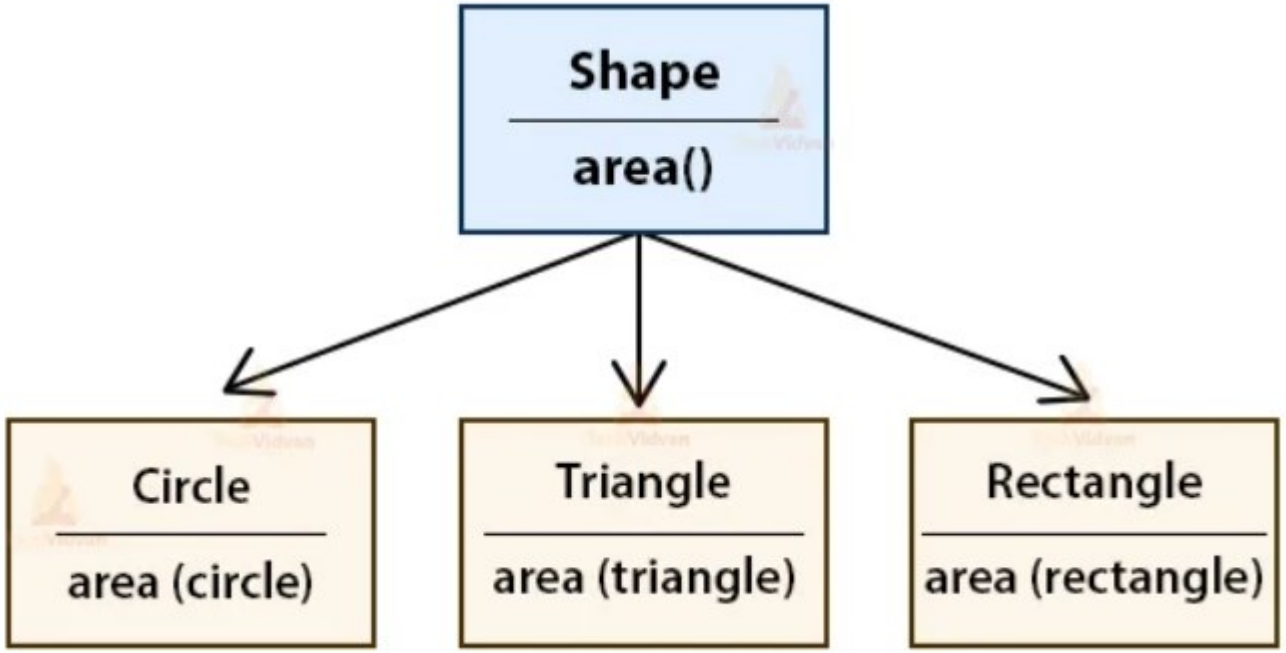




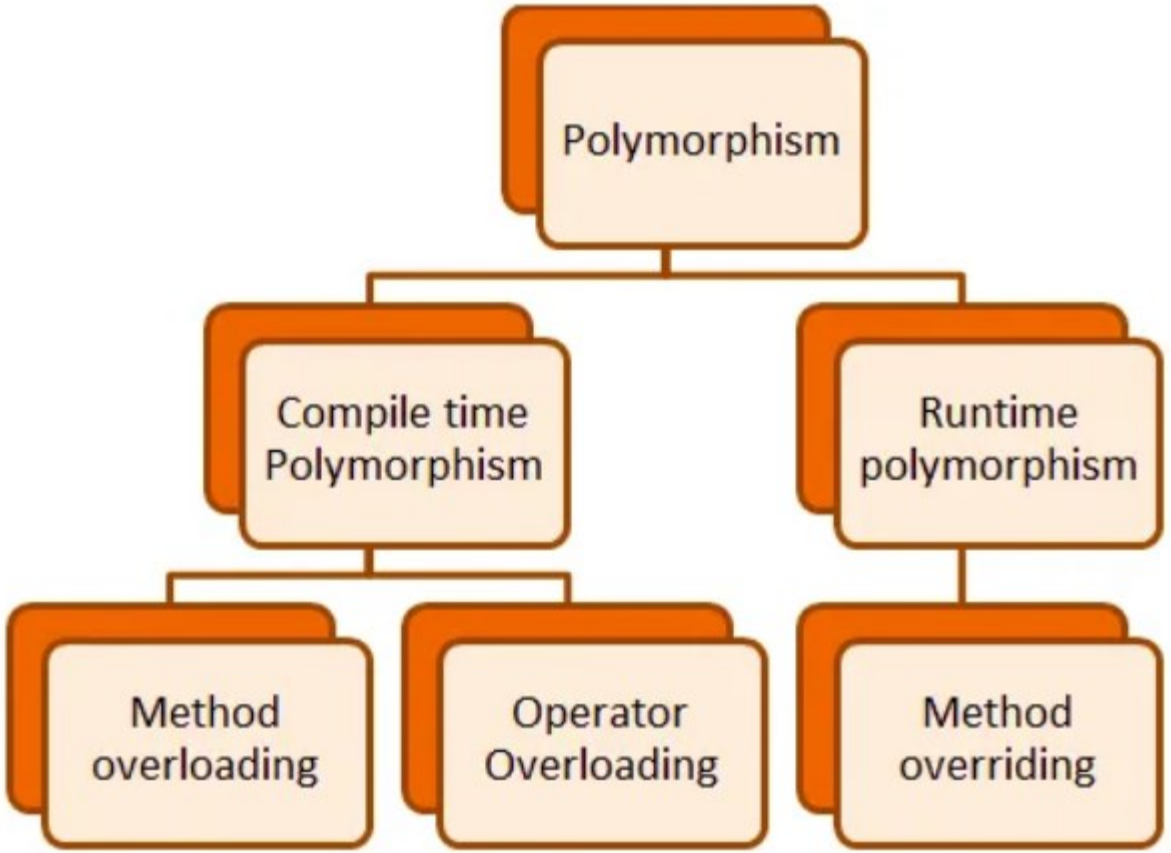
Computing the volume:
= 3.142*12*12*15
= 6786.72

What is Polymorphism?

Polymorphism is derived from two Greek words, “poly” and “morph”, which mean “many” and “forms”, respectively. Hence, polymorphism meaning in Java refers to the ability of objects to take on many forms. In other words, it allows different objects to respond to the same message or method call in multiple ways.



Types of Polymorphism



Differences between Function Overloading and Function Overriding

Function Overloading	Function Overriding
It is the process of having more than one function with same name in a particular class.	It is the process of having two functions having same name but in two different classes where one is a base class and another is its child class.
Function signature should be different for every function.	Function signature should be same for both the functions.
It is the concept of Polymorphism	It is the concept of Inheritance

Difference between multiple inheritance and interface

Multiple Inheritance	Interface
It is not supported by classes in Java because of ambiguity.	It is used in Java as a replacement of multiple inheritance

What is Inheritance?

It is the capability or feature of one class so that it can inherit properties from another class.

- **Base Class:** It is the class whose properties are inherited by another class. It is also called Super Class or Parent Class.
- **Derived Class:** It is the class that inherits properties from the base class. It is also called a sub-class or child class.

What is super keyword in Inheritance?

The super keyword in Java is a reference variable **used to refer to the immediate parent class object**. Whenever you create the instance of a subclass, an instance of the parent class is created implicitly, referred to by a super reference variable.