Object Oriented Programming

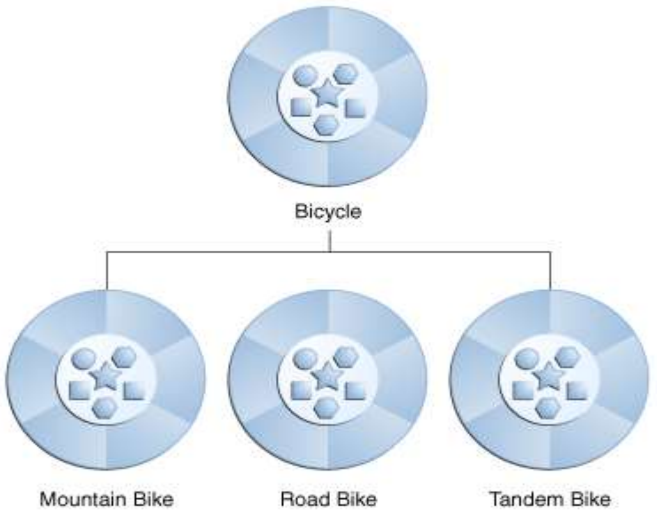
If you've never used an object-oriented programming language before, you'll need to learn a few basic concepts before you can begin writing any code. This lesson will introduce you to objects, classes, inheritance, interfaces, and packages. Each discussion focuses on how these concepts relate to the real world.

What Is a Class?

A class is a blueprint (It is user defined data types it could be anything) or prototype from which objects are created. This section defines a class that models the state and behavior of a real-world object. It intentionally focuses on the basics, showing how even simple classes can cleanly model state and behavior.

E.g.

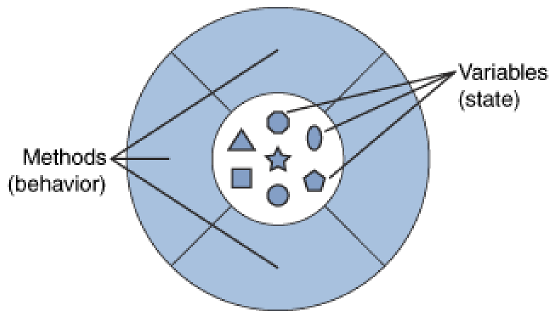
```
class Demo {
   public static void main (String args[]) {
        System.out.println("Welcome to Java”);
   }
}
```

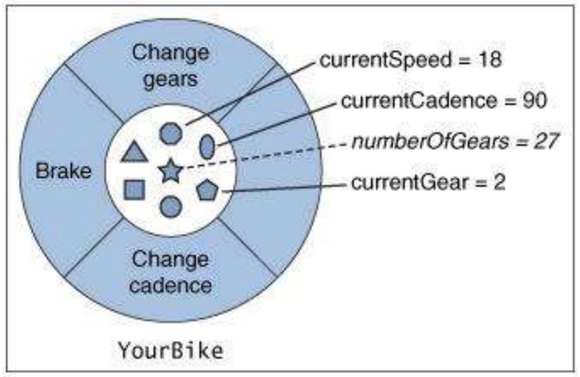


What Is an Object?

An object is a software bundle of related state and behavior. Software objects are often used to model the real-world objects that you find in everyday life (Object is real world Entity to represent a physical instance of a Class). A software object maintains its state in variables and implements its behavior with methods.

E.g.





Instance of a Class

What Is a Package?

A Java package is a mechanism for organizing Java classes into namespaces similar to the modules of Modula. Java packages can be stored in compressed files called JAR files, allowing classes to download faster as a group rather than one at a time. Programmers also typically use packages to organize classes belonging to the same category or providing similar functionality.

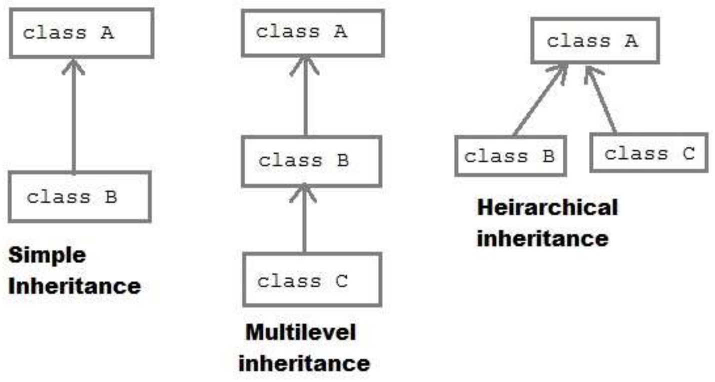
- A package provides a unique namespace for the types it contains.
- Classes in the same package can access each other's package-access members.

E.g:-

```
import java.lang.*;

import java.util.*;

import java.io.*;

import java.awt.*;
```

What Is Inheritance?

Inheritance provides a powerful and natural mechanism for organizing and structuring your software. Now we will explain how classes inherit state and behavior from their super classes, and explains how to derive one class from another using the simple syntax provided by the Java programming language.



E.g. Single Inheritance

```
class A
{
//statements;
}
class B extends A
{
    public static void main (String ar[])
    {
        System.out.println ("Welcome to Java Programming");
    }
}
```

E.g. : Multilevel Inheritance

```
class A
{
     //statements;
}
class B extends A
{
    //statements;
}
class C extends B
{
   //statements;
   public static void main(String ar[])
   {
           //statements
   }
}
```

E.g. Hierarchal Inheritance

```
class A
{
//statements;
}
class B extends A
{
//statements;
}
class C extends A
{
    public static void main(String ar[])
    {
        //statements;
    }
}
```

What is an Abstraction?

Abstraction is the process of abstraction in Java is used to hide certain details and only show the essential features of the object. In other words, it deals with the outside view of an object (interface).

Abstract class cannot be instantiated; the class does not have much use unless it is subclass. This is typically how abstract classes come about during the design phase. A parent class contains the common functionality of a collection of child classes, but the parent class itself is too abstract to be used on its own.

E.g.

```
abstract class A
{
   public abstract void sum(int x, int y);
}
class B extends A
{
     public void sum(int x,int y)
     {
           System.out.println(x+y);
     }
     public static void main(String ar[])
     {
           B obj=new B();
           obj.sum(2,5);
     }
}
```

What Is an Interface?

An interface is a collection of abstract methods (it means all methods are only declared in an Interface). A class implements an interface, thereby inheriting the abstract methods of the interface. And that class implements interface then you need to defined all abstract function which is present in an 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.

E.g.

```
interface A
{
    public void sumData(int x, int y);
}
class Demo implements A
{
      public void sumData (int x, int y)
      {
            System.out.println ("Total is "+(x+y));
      }
      public static void main (String ar[])
      {
            Demo d=new Demo ();
            d.sumData (10, 20);
      }
}
```

What Is An Encapsulation?

Encapsulation is one of the four fundamental OOP concepts. The other three are inheritance, polymorphism, and abstraction.

Encapsulation is the technique of making the fields in a class private and providing access to the fields via public methods. If a field is declared private, it cannot be accessed by anyone outside the class, thereby hiding the fields within the class. For this reason, encapsulation is also referred to as data hiding.

E.g.

```
public class EncapTest
{
    private String name;
    private String idNum;
    private int age;
    public int getAge()
    {
        return age;
    }
    public String getName()
    {
        return name;
    }
    public String getIdNum()
    {
        return idNum;
    }
public void setAge( int newAge)
     {
          age = newAge;
     }
     public void setName(String newName)
     {
           name = newName;
     }
     public void setIdNum( String newId)
     {
          idNum = newId;
     }
}
```

```
public class RunEncap
{
      public static void main(String args[])
      {
            EncapTest encap = new EncapTest();
            encap.setName("James");
            encap.setAge(20);
            encap.setIdNum("12343ms");
System.out.print("Name : " + encap.getName()+" Age
:"+encap.getAge());
      }
}
```

What is Polymorphism?

Method overloading and method overriding uses concept of Polymorphism in Java where method name remains same in two classes but actual method called by JVM depends upon object at run time and done by dynamic binding in Java. Java supports both overloading and overriding of methods. In case of overloading method signature changes while in case of overriding method signature remains same and binding and invocation of method is decided on runtime based on actual object.

Method overloading

In Method overloading we have two or more functions with the same name but different arguments. Arguments must be changed on the bases of Number, orders and Data types.

E.g.

```
class A
{
	public void f1(int x)
	{
		System.out.println(x*x);
	}
	public void f1(int x,int y)
	{
		System.out.println(x*y);
	}
```

```
    public static void main(String ar[])
    {
          A a=new A();
          a.f1(5);
          a.f1(2,3);
     }
}
```

Method Overriding

We have two classes and both classes have a function with the same name and same Parameters inheritance is necessary.

Eg.

```
class B
{
        public void f1(int x,int y)
        {
              System.out.println(x+y);
        }
}
```

```
class A extends B
{
      public void f1(int x,int y)
      {
            System.out.println(x*y);
      }


public static void main(String ar[])
      {
             A a=new A();
             a.f1(5,5);
             B b=new B();
             b.f1(2,3);
      }
}
```