# Java OOP

## Classes and objects

Object-Oriented Programming (OOP) is central to Java language. Once we talk about OOP, We always focus on classes and objects. A class is a template that can be used to create an object. Therefore, object is the instance of the class.

### Creating a class

To create a class, you must use class keyword followed by the name of the class.
Example:

```
public class Student
{

}
```

A class can have two types of members-private and public members. Private member can't be accessed by the code outside the class. In contrast, public member can be access by the code outside the class. To declare a private member, you can use private keyword and to declare a public member, you can use public keyword. Variables declared without private or public keyword are private members.
Example:

```
public class Student
{
 private int stnumber;
 private String stname;
 private String stsex;
}
```

### Methods

Method is a member of a class that can be called to perform a specific task. Before you can call it, you must define it in the class.
Example:

```
public class Student
{
  private int stnumber;
  private String stname;
  private String stsex;
  public void setnumber(int n){
     stnumber=n;
}
 public void setname(String name){
   stname=name;
}
 public void setsex(String se){
   stsex=se;
}
 public int getnumber(){
   return stnumber;
```

```
}
 public String getname(){
    return stname;
}

 public String getsex(){
    return stsex;
   }

}
```

**Creating objects from a class**

To access members of the class, you must create its object. The object can be created by using the new keyword.

Example:

Student stu=new Student();

**Accessing a class's public members through objects**

To access public members of a class, you must write its object's name followed by dot sign(.) and the names of its members.

Example:

```
class Student
{
  private int stnumber;
  private String stname;
  private String stsex;
  public void setnumber(int n){
stnumber=n;
}
public void setname(String name){
  stname=name;
}
public void setsex(String se){
   stsex=se;
}
public int getnumber(){
   return stnumber;
}
public String getname(){
   return stname;
}
public String getsex(){
  return stsex;
  }
}
public class myclass{
 public static void main(String[] args){
   Student s=new Student();//object was created.
```

```
   s.setnumber(100); //The members of class were accessed.
   s.setname("Dara");
   s.setsex("M");
   System.out.print("id="+s.getnumber()+"  name="+s.getname()+"      sex="+s.getsex());
}
}
```

**Class Constructors**

Constructor is a special method that can be used to initialize objects of the class when they are created. The constructor is declared with only the name of the class followed by brackets.

Example:

```
class Student
{
  private int stnumber;
  private String stname;
  private String stsex;
  Student(){
   stnumber=001;
   stname="Piseth";
   stsex="M";
}
public void setnumber(int n){
  stnumber=n;
}
public void setname(String name){
   stname=name;
}

public void setsex(String se){
   stsex=se;
}
public int getnumber(){
   return stnumber;
}
public String getname(){
   return stname;
}
public String getsex(){
   return stsex;
   }

}
public class myclass{
 public static void main(String[] args){
    Student s=new Student();
    System.out.print("id="+s.getnumber()+" name="+s.getname()+"    sex="+s.getsex());
    s.setnumber(100);
    s.setname("Dara");
    s.setsex("M");
```

```
    System.out.print("id="+s.getnumber()+" name="+s.getname()+" sex="+s.getsex());
   }
}
```

You also can pass parameters through constructor.

Example:

```
class Student
{
  private int stnumber;
  private String stname;
  private String stsex;
  Student(int n, String name, String sex){
     stnumber=n;
     stname=name;
     stsex=sex;
}
public void setnumber(int n){
    stnumber=n;
}
public void setname(String name){
    stname=name;
}
public void setsex(String se){
   stsex=se;
}
public int getnumber(){
   return stnumber;
}
public String getname(){
   return stname;
}
public String getsex(){
   return stsex;
   }
}
public class myclass{
  public static void main(String[] args){
    Student s=new Student(002, "Channa","M");
    System.out.print("id="+s.getnumber()+" name="+s.getname()+" sex="+s.getsex());
    s.setnumber(100);
    s.setname("Dara");
    s.setsex("M");
    System.out.print("id="+s.getnumber()+" name="+s.getname()+" sex="+s.getsex());
   }
}
```

**Class inheritance**

Inheritance enables a derived class or a child class to inherit properties and methods from a base class or parent class.

**-Using inheritance**

To illustrate the use of inheritance, now we create a base class called Animal and a derived class called Dog.

```
class Animal
{
  private int numberOflegs;

  public void setnumberOflegs(int n)
  {
    numberOflegs = n;
  }

  public int getnumberOflegs()
  {
    return numberOflegs;
  }
}
```

Now we will create the Dog class. The extends keyword is used to inherit from the Animal.

```
class Dog extends Animal
{
  private String dogName;

  public void setdogName(String name)
  {
    dogName = name;
  }

  public String getdogName()
  {
    return dogName;
  }
}
```

Now we create a TestInteritance to run the program.

```
public class TestInheritance
{
  public static void main(String[] args)
  {
    Dog d= new Dog();
    d.setnumberOflegs(4);//setnumberOflegs was inherited from the Animal class
    d.setdogName("Aluk");
    System.out.println("Dog Name: " + d.getdogName());
    System.out.println("Number of legs: " + d.getnumberOflegs());
//getnumberOflegs was inherited from the Animal class
  }
}
```

**Abstract Class**
An abstract class is created for the purpose of being inherited. We cannot create objects from the abstract class.

```
abstract class Cls
{
   public int sum(int a,int b){return 0;};
}

//implementiing the abstract class

public class Calc extends Cls{

  public static void main(String ars[]){

    Calc cal=new Calc();

    int result=cal.sum(10,20);

   System.out.println("Result:"+result);

   }

//overriding the sum() method

  public int sum(int a,int b){

    return(a+b);

  }

}
```

You may add the keyword abstract before the class Animal above and run the program again.

**Interfaces**
An interface can be used as a template for other classes. Its purpose is to be inherited. It sounds like an abstract class. However, in the interface, you cannot declare variables unless they are constants. In addition, methods can be declared, but those are without bodies. You can implement the interface by using "implements" keyword.
Example:

```
interface Inter
{
 public void print();
}

public class Print implements Inter{
  public static void main(String args[]){
```

```
  Print bs=new Print();
  bs.print();
}

public void print(){
   System.out.println("Implementing interface example");
        }

}
```