- Object Oriented Concepts
- Instance variable versus local variables
- Object reference variable
- Constructor

- There are following concepts in ObjectOriented Programming:
- 1. Class
- 2. Object
- 3. Abstraction
- 4. Polymorphism
- 5. Encapsulation
- 6. Inheritance

1. Class

- Class is a basic building block in Object Oriented Programming (OOP). Class provides the facility to put common attributes and methods in one place.
- Class serves as a template in OOP programming. Objects are created on basis of the class. We can call that object is an instance of the class.
- Class also gives visibilty to data and methods of the class. Here, visibility means how data and methods will be used outside the class.
- If class has private visibilty, then data and methods will not be accessable to outside the class.
- If class has public visibility, then data and methods will be accessable to outside the class.

```
class student {
int rollno;
void setdata()
{
rollno = 123;
}
void showdata()
{
System.out.println("Roll no is"+rollno); }
}
```

2. Object

- Object is an instance of the class or we can say it is a variable of the class.
- Object consists of variables (or data members) which are defined within the class.
- Methods of the class are called with help of the object.
- Many objects can be created for a class. Each object has its own value for the variables which are defined in the class.
- Objects are different from other objects of the class because of values that they have.

```
class student {
   int rollno;
void setdata()
{
   rollno = 123;
}
void showdata()
{
 System.out.println("Roll no is"+rollno);
}
}
class demostudent {
public static void main (String args[]){
student obj1 = new student();
obj1.setdata();
obj1.showdata();
}
}
```

3. Abstraction

- Abstraction allows programmer to see the problem at overview level. Abstraction solves the complex problem in the easy way.
- Abstraction defines the class with its member functions. It is called abstract class. Any detail of member function is not given in the abstract class.
- Member functions are defined only with input and output parameters in the abstract class. Body of the member function is not defined in the abstract class. These member functions are called abstract functions.
- Keyword **abstract** is used to define abstract class and abstract function.
- Further class can be defined from the abstract class. These sub classes inherit members of the abstract class and provides detail of each member function.
- No object can be created for an abstract class.

```
abstract class student {
    abstract void coursename();
}
class cse_student extends student {
    void coursename() {
        System.out.println("Course name is Software Engineering");
    }
}
class demostudent {
    public static void main(String args[]) {
    cse_student obj1 = new cse_student();
    obj1.coursename();
  }
}
```

4. Polymorphism

- Polymorphism provides the facility to give **sane name** to many functions.
- 'Poly' means **many** and 'morphism' means **forms** . It means **many forms** of same thing.
- Similarly, OOP permits to have many functions with the same name.

Example

```
class polydemo {
void print(int a) {
System.out.println("value of a is "+ a);
}
void print(int a, int b) {
System.out.println("value of a is "+ a);
System.out.println("value of b is "+ b);
}
void print(char c) {
System.out.println("value of c is "+ c);
}
}
```

```
class printdemo {
    public static void main(String
args[]){
        polydemo obj1 = new
polydemo();
        obj1.print(10);
        obj1.print(20,30);
        obj1.print('A');
    }
}
```

5. Encapsulation

- Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as **data hiding**.
- To achieve encapsulation in Java −
- Declare the variables of a class as private.
- Provide public methods to modify and view the variables values.

6. Inheritance

- Inheritance extends the features of a class. When a class is working and when it has been tested properly, then it is better to extend this class instead of making changes in the class itself.
- Another view of using inheritance is to make further classes (inherited classes) more specialized classes. Here specialized class means that a class which has member functins which are specific (or working) to particular situations.
- Inheritance is also used to refine features of the class or add new features to the class.
- For more detail go to Java Inheritance.

```
class student {
void showdata() {
System.out.println("Amit Kumar");
}
}
class job extends student {
void showjob () {
System.out.println("Job is Computer Engineer");
}
}
```

Instance variable Vs Local variable

Instance variable Vs Local variable

- Instance variables are defined in the class.
- When local variable name is **same as** instance variable.
- Then local variable ***hides*** instance variable.

```
class student {

    // followings are instance variables
            String name;
        int rollno;
        String address;

      public void setdata (String name, int rollno, String address)
      {

      }
}
```

- Here local variables (name, rollno and address) have **same name** as instance variales names.
- To resolve above problem, use **this keyword.**
- Because, **this** keyword always has access to current object. (i.e. it will access the instance variables of the current object)

```
class student {

      // followings are instance variables
                String name;
                int rollno;
           String address;

       public void setdata (String name, int rollno, String address)
       {
                // following is use of  this  keyword
                this.name = name;
                this.rollno = rollno;
                this.address = address;

       }
}
```

- In the above program, **this.name** referes the instance variable (i.e.class variable) of the class **student**.
- And **name** without **this** keyword, is local variable of the method **setdata()**.
- Similarly, **this.rollno** and **this.address** are instance variables of the class **student**.
- And **rollno** and **address** without **this** keyword, are local variables of the method **setdata()**.

Now see the complete program

```
public class student {
        String name;
        int rollno;
        String address;
      public void setdata (String name, int rollno, String address)
      {
                    this.name = name;
                    this.rollno = rollno;
                    this.address = address;

      }
```

```
public void dispdata(){

System.out.println("Name is " + name);
                    System.out.println("Rollno is "+ rollno);
                    System.out.println("Address is
"+address);
                }

 public static void main(String args[])     {
          student s1 = new student();
          s1.setdata(“Amit",170067,"New Delhi");
          s1.dispdata();

        }
}
```

Object Reference Variable

Object Reference Variable

- Object reference variable points to an object.
- See the following code fragment:

  - student s1 = new student();
  - student s2 = s1;

- In this code, **s1** is an object.
- **s2** is an object reference variable which is pointing to object **s1** .
- There is no copy of object **s1** is assigned to **s2** . No separate memory is alloctaed to **s2** object. Instead **s2** is refrering to same object memory (i.e. memory of **s1** object).

```
class student {
    String name;
    int rollno;
    void setdata(int r, String n){
        name = n;
        rollno = r;
    }
    void dispdata(){
System.out.println("Student name is "+name);
System.out.println("Rollno is "+r);
    }
}
```

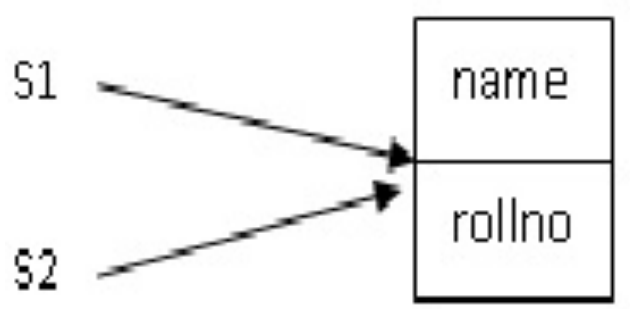
```
class demo {
    public static void main(String args[]){
        student s1 = new student();
        student s2 = s1;

        s1.setdata(1234,"Amit Kumar");
        s2.dispdata();
    }
}
```

Output

Student name is Amit Kumar

Rollno is 1234

Following figure is also showing this concept:



Here, you can see that data is stored using s1 object. Object s2 is displaying same data. It means that both (s1 and s2) are appointing to same data.

Java Constructor

Java Constructor

- Constructor is defined in the class.
- Constructor is similar to a method in the class.
- Constructor name is same as class name.
- Constructor does not has return data type.
- Constructor can receive arguments.
- Constructor is used to create and initialize the object.
- If no constructor is defined in the class, then java provides default constructor.

```
class student {
     int rollno;
     String name;
// constructor is defined, it has same name as
//class name
     student()
     {
        rollno = 10;
        name = "Rajesh";
     }

     display ()
     {
      System.out.println ("roll no is " + rollno);
         System.out.println ("name is "+ name);
        }
}
```

```
class demostudent {
   public static void main (String args[])
       {
      student s1 = new student(); x
        s1.diaplay();
      }
}
```

Parameterized Constructor

- When we passing the arguments to constructor. It is called parameterized constrictor.
- Parametrized constructors are used to initialize the variables from user.

Example

```
class student {
    int rollno;
    String name;
// Parametrized constructor
    student(int r, String n)
    {
      rollno = r;
      name = n;
     }
    display ()
     {
        System.out.println ("roll no is " + rollno);
            System.out.println ("name is "+ name);
         }
}
```

```
class demostudent {
  public static void main (String args[])
      {

    student s1 = new student(10, "Rajesh");
     s1.diaplay();
    }
}
```