Java

Classes, Objects, Methods, Constructors and Method and Constructor Overloading

Objects and Classes

- A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.
- Object is the instance of a class.
- A class in Java can contain:
  - **fields**
  - **methods**
  - **constructors**
  - **blocks**
  - **nested class and interface**

```
class <class_name>{
    field;
    method;
}
```

- **Instance variable in Java**
- A variable which is created inside the class but outside the method, is known as instance variable. Instance variable doesn't get memory at compile time. It gets memory at run time when object(instance) is created. That is why, it is known as instance variable.
- **Method in Java**
- In java, a method is like function i.e. used to expose behavior of an object.
- **new keyword in Java**
- The new keyword is used to allocate memory at run time.

Object and Class Example: main within class

```
class Student{
 int id;   //field or data member or instance variable
 String name;

 public static void main(String args[]){
  Student s1=new Student();        //creating an object of Student
  System.out.println(s1.id);
    //accessing member through reference variable
  System.out.println(s1.name);
 }
}
```

- **Object and Class Example: main outside class**

```
class Student{
 int id;
 String name;
}
class TestStudent1{
 public static void main(String args[]){
  Student s1=new Student();
  System.out.println(s1.id);
  System.out.println(s1.name);
 }
}
```

3 Ways to initialize object

- There are 3 ways to initialize object in java.
    - By reference variable
    - By method
    - By constructor

Object and Class Example: Initialization through reference

```
class Student{
 int id;
 String name;
}
class TestStudent3{
 public static void main(String args[]){
  //Creating objects
  Student s1=new Student();
  Student s2=new Student();
  //Initializing objects
  s1.id=101;
  s1.name="Sonoo";
  s2.id=102;
  s2.name="Amit";
  //Printing data
  System.out.println(s1.id+" "+s1.name);
  System.out.println(s2.id+" "+s2.name);
 }
}
```

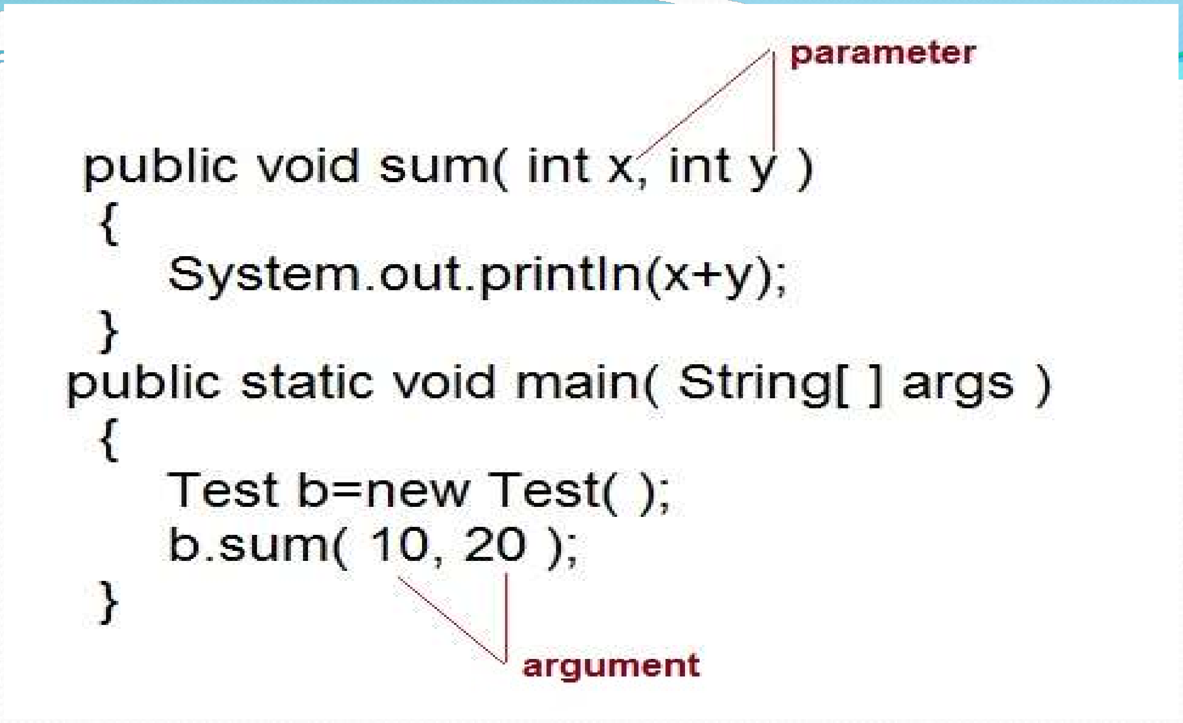
Methods in Java

Method describe behavior of an object. A method is a collection of statements that are grouped together to perform an operation.

```
return-type methodName(parameter-list)
 { //body of method }
```

Parameter Vs. Argument

While talking about method, it is important to know the difference between two terms **parameter** and **argument**. **Parameter** is variable defined by a method that receives value when the method is called. Parameter are always local to the method they don't have scope outside the method. While **argument** is a value that is passed to a method when it is called.



```
 public void sum( int x, int y )
 {
     System.out.println(x+y);
 }
public static void main( String[ ] args )
 {
     Test b=new Test( );
     b.sum( 10, 20 );
 }
```

call-by-value : In this approach copy of an argument value is pass to a method. Changes made to the argument value inside the method will have no effect on the arguments.

NOTE :There is only call by value in java, not call by reference.

```
class Operation{
 int data=50;

 void change(int data){
 data=data+100;//changes will be in the local variable only
 }

 public static void main(String args[]){
   Operation op=new Operation();

   System.out.println("before change "+op.data);
   op.change(500);
   System.out.println("after change "+op.data);

 }
}
```

Output:

```
before change 50
after change 50
```

Method Overloading

- If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.
- There are two ways to overload the method in java
  - By changing number of arguments
  - By changing the data type
- **Method overloading** is also known as **Static Polymorphism** or **Compile time Polymorphism**

Note:- In java, method overloading is not possible by changing the return type of the method only because of ambiguity.

1) Overloading – Different Number of parameters in argument list

When methods name are same but number of arguments are different.

```
class DisplayOverloading {

 public void disp(char c) {
 System.out.println(c);
}

public void disp(char c, int num) {
 System.out.println(c + " "+num);
}
}

class Sample { public static void main(String args[]) {
DisplayOverloading obj = new DisplayOverloading();
 obj.disp('a');
 obj.disp('a',10);
 }
 }
```

```
class Area {
void find(int l, int b) {
System.out.println("Area is"+(l*b)) ;
}
void find(int l, int b,int h) {
 System.out.println("Area is"+(l*b*h));
}
}

class AreaDemo{
public static void main (String[] args) {
 Area ar = new Area();
ar.find(8,5); //find(int l, int b) is method is called.
 ar.find(4,6,2); //find(int l, int b, int h) is called.
}
}
```

2) Method Overloading: changing data type of arguments

```
class Calculate {
 void sum (int a, int b) {
 System.out.println("sum is"+(a+b)) ;
 }
void sum (float a, float b) {
System.out.println("sum is"+(a+b));
}
}

class CalculateDemo {
public static void main (String[] args) {
Calculate cal = new Calculate();
 cal.sum (8,5);                    //sum(int a, int b) is method is
called.
 cal.sum (4.6f, 3.8f);        //sum(float a, float b) is called.
}
}
```

Output :

```
sum is 13
sum is 8.4
```

```java
public class Sum {
   // Overloaded sum(). This sum takes two int parameters
   public int sum(int x, int y) {
      return (x + y);
   }

   // Overloaded sum(). This sum takes three int parameters
   public int sum(int x, int y, int z) {
       return (x + y + z);
   }

   // Overloaded sum(). This sum takes two double parameters
   public double sum(double x, double y) {
       return (x + y);
   }

   // Driver code
   public static void main(String args[]) {
      Sum s = new Sum();
      System.out.println(s.sum(10, 20));
      System.out.println(s.sum(10, 20, 30));
      System.out.println(s.sum(10.5, 20.5));
   }
}
```

Can we overload java main() method?

Yes, by method overloading. You can have any number of main methods in a class by method overloading. But JVM calls main() method which receives string array as arguments only. Let's see the simple example:

```
class TestOverloading{
public static void main(String[] args){
System.out.println("main with String[]"
);
}
public static void main(String args){
System.out.println("main with String");
}
public static void main(){
System.out.println("main without args"
);
}
}
```

Output:

```
main with String[]
```

Object and Class Example: Initialization through method

```
class Student{
 int rollno;
 String name;
 void insertRecord(int r, String n){
  rollno=r;
  name=n;
 }
 void displayInformation()
{System.out.println(rollno+" "+name);}
}
class TestStudent4{
 public static void main(String args[]){
  Student s1=new Student();
  Student s2=new Student();
  s1.insertRecord(111,"Karan");
  s2.insertRecord(222,"Aryan");
  s1.displayInformation();
  s2.displayInformation();
 }
}
```

Constructor in Java

- **Constructor in java** is a *special type of method* that is used to initialize the object.
- Java constructor is *invoked at the time of object creation*. It constructs the values i.e. provides data for the object that is why it is known as constructor.
- **Types of java constructors**
- There are two types of constructors:
  - Default constructor (no-arg constructor)
  - Parameterized constructor

```
class Cricketer {
String name;
 String team;
 int age;

Cricketer () //default constructor.
 { name ="";
 team ="";
 age = 0; }

 Cricketer(String n, String t, int a) //parameterized constructor
{ name = n;
 team = t;
age = a; }
}
```

```
Class test
{ public static void main (String[] args)
{ Cricketer c1 = new Cricketer();
Cricketer c2 = new Cricketer("sachin", "India", 32);
c1.name = "Virat";
c1.team= "India";
c1.age = 32;
System .out. println (c1.name + " ," + c1.team + " ," +
  c1.age);
System .out. println (c2.name + " ," + c2.team + " ," +
  c2.age);} }
```

Constructor Overloading in Java

- Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists. The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

```
class Box
{
    double width, height, depth;
       Box(double w, double h, double d)
    {
        width = w;
        height = h;
        depth = d;
    }
        Box()
    {
        width = height = depth = 0;
    }

    Box(double len)
    {
        width = height = depth = len;
    }

    double volume()
    {
        return width * height * depth;
    }
}
```

```
public class Test
{   public static void main(String args[])
    {
        Box mybox1 = new Box(10, 20, 15);
        Box mybox2 = new Box();
        Box mycube = new Box(7);

        double vol;

        // get volume of first box
        vol = mybox1.volume();
        System.out.println(" Volume of mybox1 is " + vol);

        // get volume of second box
        vol = mybox2.volume();
        System.out.println(" Volume of mybox2 is " + vol);

        // get volume of cube
        vol = mycube.volume();
        System.out.println(" Volume of mycube is " + vol);
    }
}
```

Difference between constructor and a method

Java Constructor

- Constructor is used to initialize the state of an object.
- Constructor must not have return type.
- Constructor is invoked implicitly.
- The java compiler provides a default constructor if you don't have **any** constructor.

Java Method

- Method is used to expose behavior of an object.
- Method must have return type.
- Method is invoked explicitly.
- Method is not provided by compiler in any case.