**Object and Classes**

Since Java is an object oriented language, complete java language is build on classes and object. Java is also known as a strong **Object oriented programming language**(OOPS).

OOPS is a programming approach which provides solution to problems with the help of algorithms based on real world. It uses real world approach to solve a problem. So object oriented technique offers better and easy way to write program then procedural programming model such as C, ALGOL, PASCAL.

#### Main Features of OOPS

* Inheritence
* Polymorphism
* Encapsulation
* Abstraction

As an object oriented language Java supports all the features given above. We will discuss all these features in detail later.

### Class

In Java everything is encapsulated under classes. Class is the core of Java language. Class can be defined as a template/ blueprint that describe the behaviors /states of a particular entity. A class defines new data type. Once defined this new type can be used to create object of that type. Object is an instance of class. You may also call it as physical existence of a logical template class.

A class is declared using **class** keyword. A class contain both data and code that operate on that data. The data or variables defined within a **class** are called **instance variables** and the code that operates on this data is known as **methods**. Thus, the instance variables and methods are known as class members. **class** is also known as a user defined datatype.

**A class and an object can be related as follows:** Consider an ice tray(like of cube shape) as a class. Then ice cubes can be considered as the objects which is a blueprint of its class i.e of ice tray.

#### Rules for Java Class

* A class can have only public or default(no modifier) access specifier.
* It can be either abstract, final or concrete (normal class).
* It must have the class keyword, and class must be followed by a legal identifier.
* It may optionally extend one parent class. By default, it will extend java.lang.Object.
* It may optionally implement any number of comma-separated interfaces.
* The class's variables and methods are declared within a set of curly braces {}.
* Each **.java** source file may contain only one public class. A source file may contain any number of default visible classes.
* Finally, the source file name must match the public class name and it must have a .java suffix.

#### A simple class example

Suppose, Student is a **class** and student's name, roll number, age will be its property. Lets see this in Java syntax

class Student.

{

String name;

int rollno;

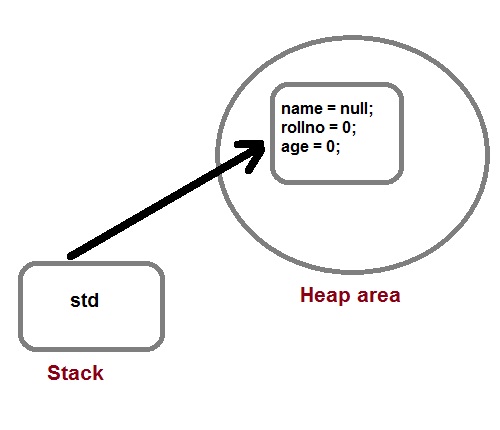
int age;

}

When a reference is made to a particular student with its property then it becomes an **object**, physical existence of Student class.

Student std=new Student();

After the above statement **std** is instance/object of Student class. Here the **new** keyword creates an actual physical copy of the object and assign it to the **std** variable. It will have physical existence and get memory in heap area. **The new operator dynamically allocates memory for an object**



#### Q. How a class is initialized in java?

A Class is initialized in Java when an instance of class is created using either **new** operator or using reflection using class.forName(). A class is also said to be initialized when a static method of **Class** is invoked or a static field of **Class** is assigned.

### Methods in Java

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

**Syntax :**

return-type methodName(parameter-list)

{

//body of method

}

#### Example of a Method

public String getName(String st)

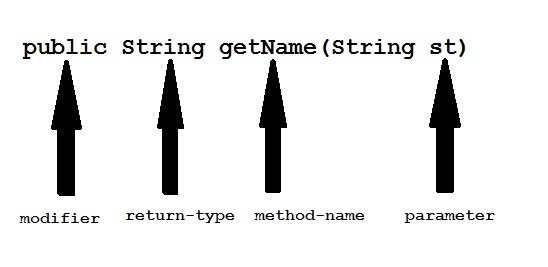
{

String name="StudyTonight";

name=name+st;

return name;

}



**Modifier :** Modifier are access type of method. We will discuss it in detail later.

**Return Type :** A method may return value. Data type of value return by a method is declare in method heading.

**Method name :** Actual name of the method.

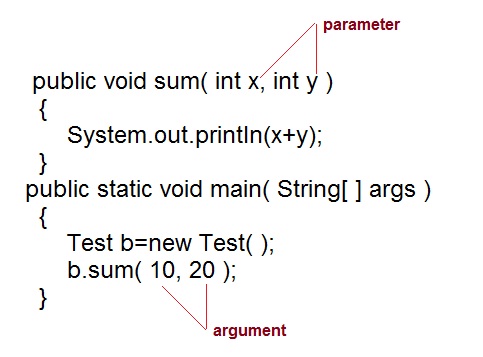
**Parameter :** Value passed to a method.

**Method body :** collection of statement that defines what method does.

#### 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 dont have scope outside the method. While **argument** is a value that is passed to a method when it is called.



#### call-by-value and call-by-reference

There are two ways to pass an argument to a method

1. **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.
2. **call-by-reference :** In this reference of an argument is pass to a method. Any changes made inside the method will affect the agrument value.

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

#### Example of call-by-value

public class Test

{

public void callByValue(int x)

{

x=100;

}

public static void main(String[] args)

{

int x=50;

Test t = new Test();

t.callByValue(x); //function call

System.out.println(x);

}

}

50

### Method overloading

If two or more method in a class have same name but different parameters, it is known as method overloading. Overloading always occur in the same class(unlike method overriding).

Method overloading is one of the ways through which java supports polymorphism. Method overloading can be done by changing number of arguments or by changing the data type of arguments. If two or more method have same name and same parameter list **but differs in return type are not** said to be overloaded method

**Note:** Overloaded method can have different access modifiers.

#### Different ways of Method overloading

There are two different ways of method overloading

#### Method overloading by changing data type of Arguments

*Example :*

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));

}

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.

}

}

Sum is 13 Sum is 8.4

You can see that sum() method is overloaded two times. The first takes two integer arguments, the second takes two float arguments.

#### Method overloading by changing no. of argument.

*Example* :

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));

}

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.

}

}

Area is 40 Area is 48

In this example the find() method is overloaded twice. The first takes two arguments to calculate area, and the second takes three arguments to calculate area.

When an overloaded method is called java look for match between the arguments to call the method and the method's parameters. This match need not always be exact, sometime when exact match is not found, Java automatic type conversion plays a vital role.

#### Example of Method overloading with type promotion.

class Area

{

void find(long l,long b)

{

System.out.println("Area is"+(l\*b)) ;

}

void find(int l, int b,int h)

{

System.out.println("Area is"+(l\*b\*h));

}

public static void main (String[] args)

{

Area ar = new Area();

ar.find(8,5); //automatic type conversion from find(int,int) to find(long,long) .

ar.find(2,4,6) //find(int l, int b,int h) is called.

}

}

Area is 40

Area is 48

### Constructors in Java

A constructor is a special method that is used to initialize an object.Every class has a constructor,if we don't explicitly declare a constructor for any java class the compiler builds a default constructor for that class. A constructor does not have any return type.

A constructor has same name as the class in which it resides. Constructor in Java can not be abstract, static, final or synchronized. These modifiers are not allowed for constructor.

class Car

{

String name ;

String model;

Car( ) //Constructor

{

name ="";

model="";

}

}

#### There are two types of Constructor

* Default Constructor
* Parameterized constructor

Each time a new object is created at least one constructor will be invoked.

Car c = new Car() //Default constructor invoked

Car c = new Car(name); //Parameterized constructor invoked

#### Constructor Overloading

Like methods, a constructor can also be overloaded. Overloaded constructors are differentiated on the basis of their type of parameters or number of parameters. Constructor overloading is not much different than method overloading. In case of method overloading you have multiple methods with same name but different signature, whereas in Constructor overloading you have multiple constructor with different signature but only difference is that Constructor doesn't have return type in Java.

#### Q. Why do we Overload constructors ?

Constuctor overloading is done to construct object in different ways.

#### Example of constructor overloading

class Cricketer

{

String name;

String team;

int age;

Cricketer () //default constructor.

{

name ="";

team ="";

age = 0;

}

Cricketer(String n, String t, int a) //constructor overloaded

{

name = n;

team = t;

age = a;

}

Cricketer (Cricketer ckt) //constructor similar to copy constructor of c++

{

name = ckt.name;

team = ckt.team;

age = ckt.age;

}

public String toString()

{

return "this is " + name + " of "+team;

}

}

Class test:

{

public static void main (String[] args)

{

Cricketer c1 = new Cricketer();

Cricketer c2 = new Cricketer("sachin", "India", 32);

Cricketer c3 = new Cricketer(c2 );

System.out.println(c2);

System.out.println(c3);

c1.name = "Virat";

c1.team= "India";

c1.age = 32;

System .out. print in (c1);

}

}

this is sachin of india this is sachin of india this is virat of india

#### Q What's the difference between constructors and normal methods?

Constructors must have the same name as the class and can not return a value. They are only called once for a single object while regular methods could be called many times and it can return a value or can be void.

#### Q. What is constructor chaining in Java?

Constructor chaining is a phenomena of calling one constructor from another constructor of same class. Since constructor can only be called from another constructor in Java, constructor chaining is used for this purpose.

class Test

{

Test()

{

this(10);

}

Test(int x)

{

System.out.println("x="+x);

}

public static void main(String arg[])

{

Test object = new Test();

}

}

x=10

#### Q. Does constructors return any value?

Yes, constructors return current instant of a class. But yet constructor signature cannot have any return type.