# UNIT II

# Contents of Unit II

- Functions in OOP
- Function overloading
- Passing & returning Objects
- Pointers to members
- Constructors
- Access specifiers and packages
- Inheritance, types of inheritance,
- Run time polymorphism,
- Abstract classes,
- Interface, collection interface

## Methods/ Function

- In general, a **method** is a way to perform some task.
- Similarly, the **method in Java** is a collection of instructions that performs a specific task.
- It provides the reusability of code.
- We can also easily modify code using **methods**.

## Types of Method

- There are two types of methods in Java:
  - Predefined Method
    - In Java, predefined methods are the method that is already defined in the Java class libraries is known as predefined methods.
    - We can directly use these methods just by calling them in the program at any point
    - Some pre-defined methods are **length(), equals(), compareTo(), sqrt(),** etc.
  - User-defined Method

## Example of Predefined methods

```
public class Demo
{
public static void main(String[] args)
{
// using the max() method of Math class

System.out.print("The maximum number is: " + Math.max(9,7));  Syst
em.out.println(“end”);
}
}
```

```
The maximum number is:9
end
```

- In the above example, we have used three predefined methods **main(), print(),** and **max()**.
- We have used these methods directly without declaration because they are predefined.
- The print() method is a method of **PrintStream** class that prints the result on the console.
- The max() method is a method of the **Math** class that returns the greater of two numbers.

## User-defined Method

- The method written by the user or programmer is known as **a user-defined** method.
- These methods are modified according to the requirement.

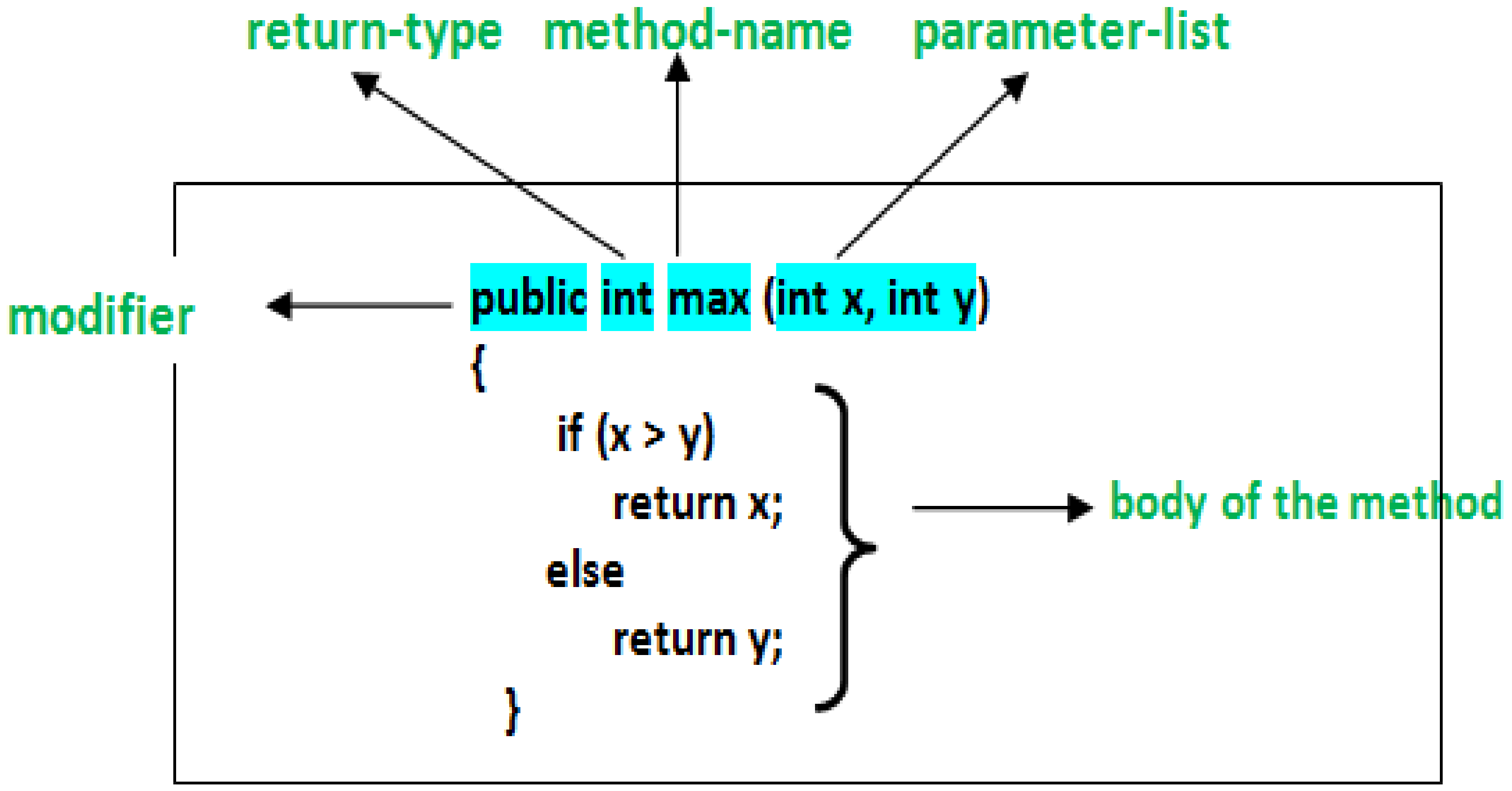
## Method Declaration

- **Modifier**-: Defines **access type** of the method i.e. from where it can be accessed in your application.
- In Java, there 4 type of the access specifiers.
  - **public**: accessible in all class in your application.
  - **protected**: accessible within the class in which it is defined and in its **subclass(es)**
  - **private**: accessible only within the class in which it is defined.
  - **default** (declared/defined without using any modifier) : accessible within same class and package within which its class is defined.

# Method Declaration

- **The return type** : The data type of the value returned by the method or void if does not return a value.
- **Method Name** : the rules for field names apply to method names as well, but the convention is a little different.
- **Parameter list** : Comma separated list of the input parameters are defined, preceded with their data type, within the enclosed parenthesis. If there are no parameters, you must use empty parentheses ().
- **Method body** : it is enclosed between braces. The code you need to be executed to perform your intended operations.

# Method Declaration



```
public int max (int x, int y)
{
     if (x > y)
          return x;
     else
          return y;
}
```

## Method signature

- It consists of the method name and a parameter list (number of parameters, type of the parameters and order of the parameters).
- The return type and exceptions are not considered as part of it.
- Method Signature of above function:
  - max(int x, int y)

## How to name a Method?

- A method name is typically a single word that should be a **verb** in lowercase or multi-word, that begins with a **verb** in lowercase followed by **adjective, noun…..**
- After the first word, first letter of each word should be capitalized.
- For example, findSum, computeMax, setX and getX

```
// Program to illustrate methodsin java
import java.io.*;

class Addition {

          int sum = 0;

          public int addTwoInt(int a, int b){

                    // adding two integer value.
                    sum = a + b;

                    //returning summation of two values.
                    return sum;
          }

}
```

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

                    // creating an instance of Addition class
                    Addition add = new Addition();

// calling addTwoInt() method to add two integer using instance
created
                    // in above step.
                    int s = add.addTwoInt(1,2);
                    System.out.println("Sum of two integer values :"+ s);

          }
}
```

# Constructors

- In Java, a constructor is a block of codes similar to the method.
- It is called when an instance of the class is created.
- At the time of calling constructor, memory for the object is allocated in the memory.

Product p1

Product p1 = new Product()

Int a

Int a =0 //initialization

- It is a special type of method which is used to initialize the object.
- Every time an object is created using the new() keyword, at least one constructor is called.
- It calls a default constructor if there is no constructor available in the class.
- In such case, Java compiler provides a default constructor by default.

- There are two types of constructors in Java: **no-arg constructor, and parameterized constructor.**
- **Note:** It is called constructor because it constructs the values at the time of object *creation.*
- It is not necessary to write a constructor for a class.
- It is because java compiler creates a default constructor if your class doesn't have any.

## Rules for creating Java constructor

- There are two rules defined for the constructor.
  - Constructor name must be the same as its class name
  - A Constructor must have no explicit return type

## Types of Java constructors

- There are two types of constructors in Java:
  - Default constructor (no-arg constructor)
  - Parameterized constructor

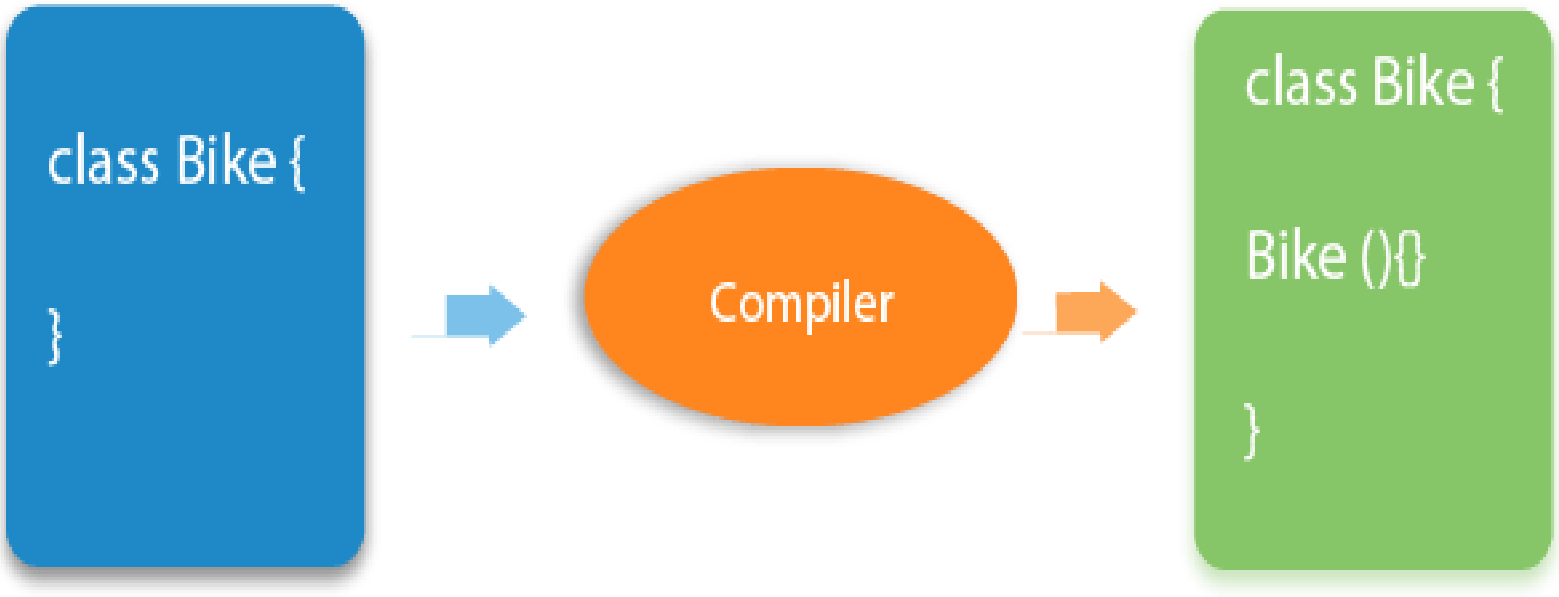
# Java Default Constructor

- A constructor is called "Default Constructor" when it doesn't have any parameter.
- <class_name>(){}

# Example of default constructor

```
//Java Program to create and call a default constructor
class Bike1
{
//creating a default constructor
Bike1()
{
System.out.println("Bike is created");
}
//main method
public static void main(String args[])
{
//calling a default constructor
Bike1 b=new Bike1();
}
}
```

Rule: If there is no constructor in a class, compiler automatically creates a default constructor.



- The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

## Example of default constructor that displays the default values

```
//Let us see another example of default constructor
//which displays the default values
Public class Student3
{
int id;
String name;
//method to display the value of id and name
void display()
{System.out.println(id+" "+name);}

public static void main(String args[])
{
//creating objects
Student3 s1=new Student3();
Student3 s2=new Student3();
//displaying values of the object
s1.display();
s2.display();
}
}
```

## Example of parameterized constructor

```
//Java Program to demonstrate the use of the parameterized construct
or.
class Student4
{
    int id;
    String name;
    //creating a parameterized constructor
    Student4(int i,String n)
      {
        id = i;
       name = n;
    }
    //method to display the values
    void display(){System.out.println(id+" "+name);}
```

```
 public static void main(String args[])
{
   //creating objects and passing values
   Student4 s3 = new Student4();
   Student4 s1 = new Student4 (111,"Karan");
   Student4 s2 = new Student4 (222,"Aryan");

//calling method to display the values of object
   s1.display();
   s2.display();
  }
}
```

## Difference between constructor and method in Java

| Java Constructor | Java Method |
|---|---|
| A constructor is used to initialize the state of an object. | A method is used to expose the behavior of an object. |
| A constructor must not have a return type. | A method must have a return type. |
| The constructor is invoked implicitly. | The method is invoked explicitly. |
| The Java compiler provides a default constructor if you don't have any constructor in a class. | The method is not provided by the compiler in any case. |
| The constructor name must be same as the class name. | The method name may or may not be same as the class name. |

## Java Copy Constructor

- There is no copy constructor in Java.
- However, we can copy the values from one object to another like copy constructor in C++.
- There are many ways to copy the values of one object into another in Java. They are:
  - By constructor
  - By assigning the values of one object into another
  - By clone() method of Object class

# Example

```
//Java program to initialize the values from one object to another object.
Public class Student6
{
   int id;
   String name;
   //constructor to initialize integer and string
   Student6(int i,  String n)
    {
      id = i;
      name = n;
   }
   //constructor to initialize another object
   Student6(Student6 s)
   {
   id = s.id;
   name =s.name;
   }
   void display(){
Int a2 = 7;
System.out.println(id+" "+name);}
```

```
 public static void main(String args[])
 {
  Student6 s1 = new Student6(111,"Karan");
  Student6 s2 = new Student6(s1);
  s1.display();
  s2.display();
  }
}
```

- In this example, we have copied the values of one object into another using Java constructor.

# Copying values without constructor

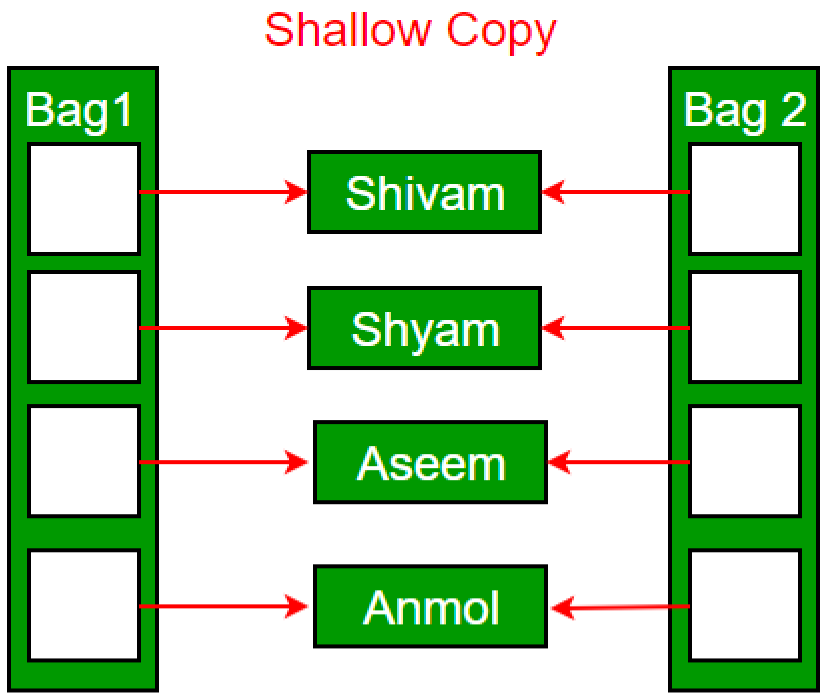
- We can copy the values of one object into another by assigning the objects values to another object.
- In this case, there is no need to create the constructor.

```
class Student7
{
    int id;
    String name;
    Student7(int i,String n)
    {
    id = i;
    name = n;
    }
    Student7(){}
    void display(){System.out.println(id+" "+name);}
public static void main(String args[])
{
   Student7 s1 = new Student7(111,"Karan");
   Student7 s2 = new Student7();
   s2.id=s1.id;
   s2.name=s1.name;
   s1.display();
   s2.display();
  }
}
```

## Some questions

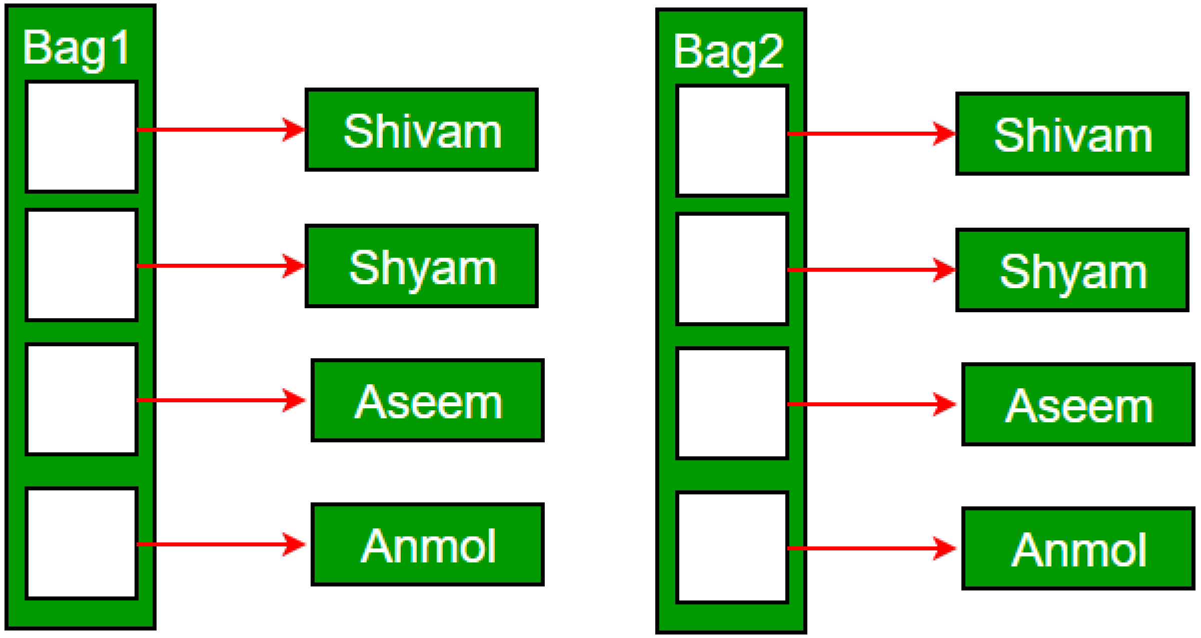
- Does constructor return any value?
  - Yes, it is the current class instance (You cannot use return type yet it returns a value).
- Can constructor perform other tasks instead of initialization?
  - Yes, like object creation, starting a thread, calling a method, etc. You can perform any operation in the constructor as you perform in the method.

# *Default constructor does only shallow copy*



- ***Deep copy is possible only with user defined copy constructor.***
- In user defined copy constructor, we make sure that pointers (or references) of copied object point to new memory locations.

Deep Copy



Bag1
Shivam
Shyam
Aseem
Anmol
Bag2
Shivam
Shyam
Aseem
Anmol

## Copy constructor vs Assignment Operator

```
MyClass t1, t2;
MyClass t3 = t1;  // ----> (1)
t2 = t1;          // -----> (2)
```

- Copy constructor is called when a new object is created from an existing object, as a copy of the existing object.
- Assignment operator is called when an already initialized object is assigned a new value from another existing object.
- In the above example (1) calls copy constructor and (2) calls assignment operator.

- How to write a class?
- **How to instantiate a class?**
- **How to initialize variables in a class?**
- How to write method in a class?
- How to call the method?
- How to print the values on screen?
- **How to assign the values to the objects, 3 ways**
- How to write a constructor?
- When constructor is called?

- how to instance a class
- Call method
- how to write a constructor
- when constructor is called?
- Assign the value to the object
- How to write call method
- Print the value on screen
- 3 ways of writing object?

```
Class Student
{
 private int rollnumber = 0;
Student(int rno)
{
   rollnumber = rno;
}
          Public void setRollNumber(int rno)
          {
            rollnumber = rno;
          }
          Public int getRollNumber()
          {
             return rollnumber
          }
}
Student s1 = new Student(101);  ///2  using constructor
s1.rollnumber = 101;   //1  // using reference variable

S1.setRollNumber(102); // 3 // using the method
Int roll_number = s1.getRollNumber();
```

s1--- object of the class
S1—instance of the class

## Destructor in Java

- Destructors in Java can be learned with the finalize method in Java.
- In case, there is a need for calling the destructor, it can be done with the help of the finalize method
- This method is not independent as it relies on Garbage Collection.
- The garbage collector is a thread that deletes or destroyed the unused object in the heap area.

- Destructor is a method called when the destruction of an object takes place.
- “ The main goal of the destructor is to free up the allocated memory and also to clean up resources like the closing of open files, closing of database connections, closing network resources, etc,

- Class Object

  ```
  {
  protected void finalize()
  {
  //statements like closure of database
  connection
  }
  }
  ```

-
  ```
  public class Demo
  {
  public static void main(String[] args)
  {
  Integer i = new Integer(2);
  i = null;
  System.gc();
  System.out.println("In the Main Method");
  }
  protected void finalize()
  {
  System.out.println("object is garbage collected ");
  }
  }
  ```

## Function Overloading

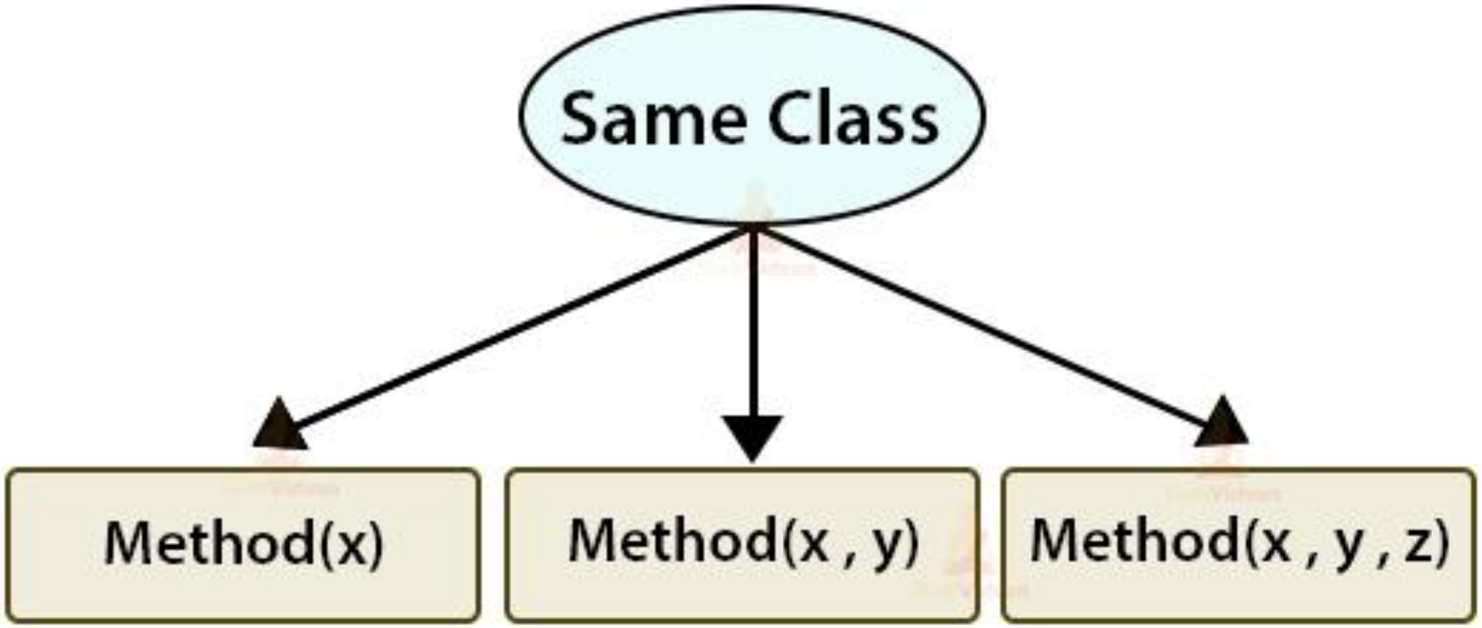
- If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.
- If we have to perform only one operation, having same name of the methods increases the readability of the program.

- Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as **a(int,int)** for two parameters, and **b(int,int,int)** for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

# Different ways to overload the method

- Overloading allows different methods to have the same name, but different signatures where the signature can differ by the number of input parameters or type of input parameters or both.
- Overloading is related to compile-time (or static) polymorphism.
- There are two ways to overload the method in java
  - By changing number of arguments
  - By changing the data type

## Method Overloading in Java



## 1) Method Overloading: changing no. of arguments

```
class Adder{
 int add(int a,int b){return a+b;}
 int add(int a,int b,int c){return a+b+c;}
}
class TestOverloading1{
public static void main(String[] args){
Adder a = new Adder();
System.out.println(a.add(11,11));
System.out.println(a.add(11,11,11));
}}
```

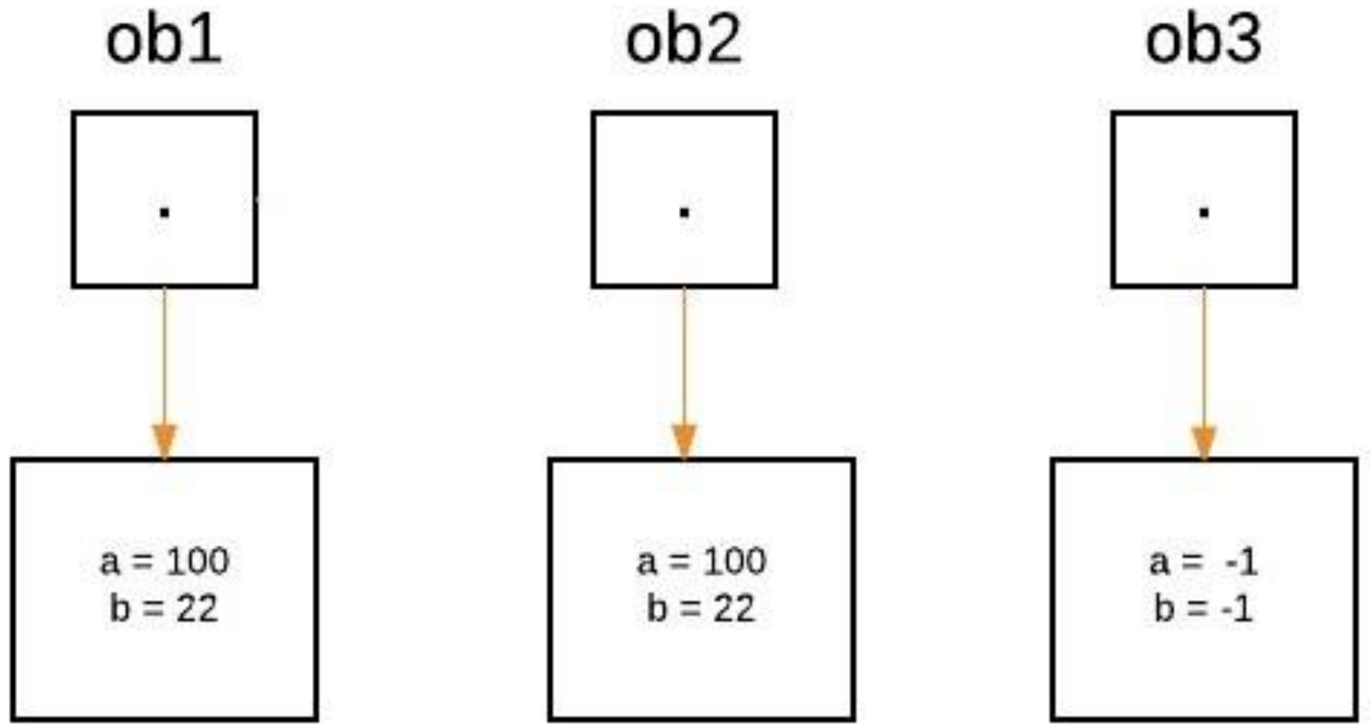
## 2) Method Overloading: changing data type of arguments

```
class Adder
{
 int add(int a, int b){return a+b;}
 double add(double a, double b){return a+b;}
}
class TestOverloading2
{
public static void main(String[] args){
Adder a = new Adder();
System.out.println(a.add(11,11));
System.out.println(a.add(12.3,12.6));
}
}
```

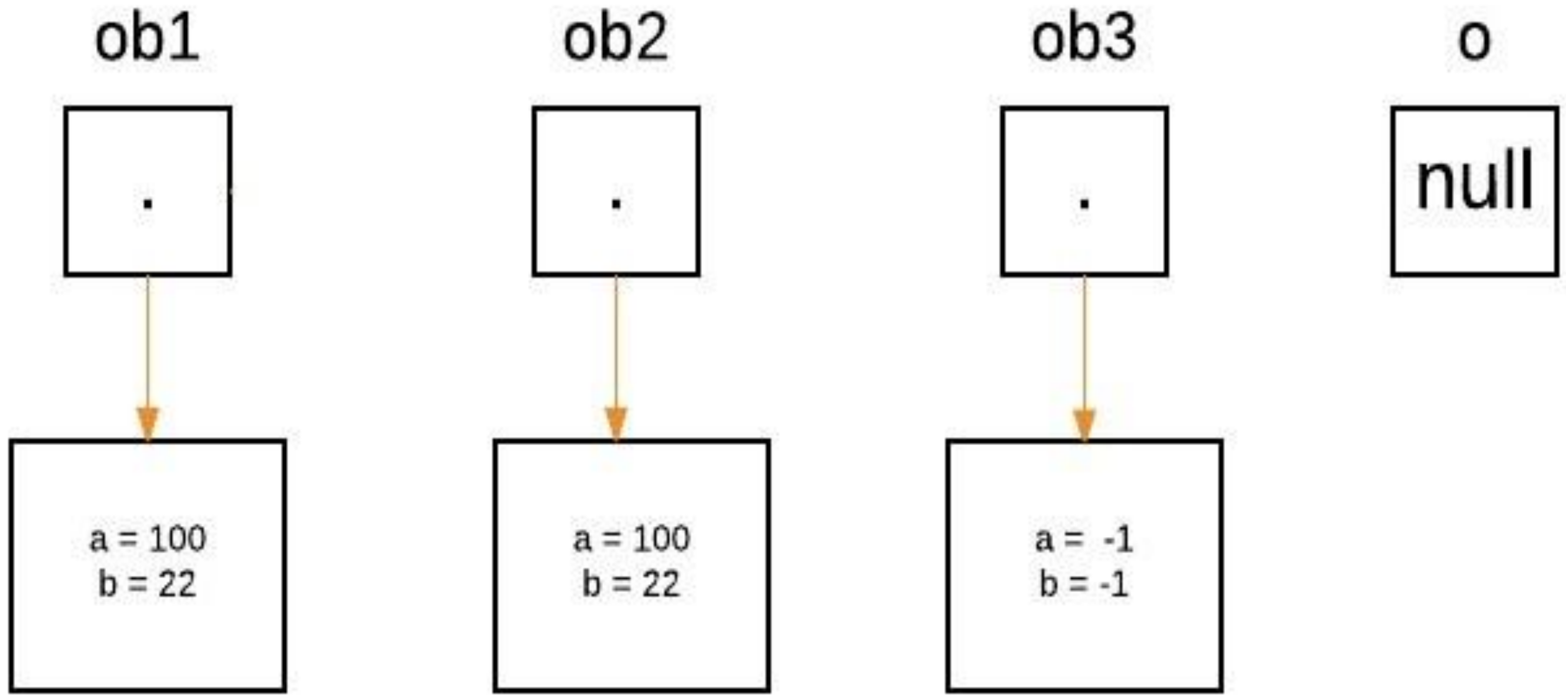
# Passing and Returning Objects in Java

- Three objects ‘ob1’ , ‘ob2’ and ‘ob3’ are created

```
ObjectPassDemo ob1 = new ObjectPassDemo(100, 22);
ObjectPassDemo ob2 = new ObjectPassDemo(100, 22);
ObjectPassDemo ob3 = new ObjectPassDemo(-1, -1);
```

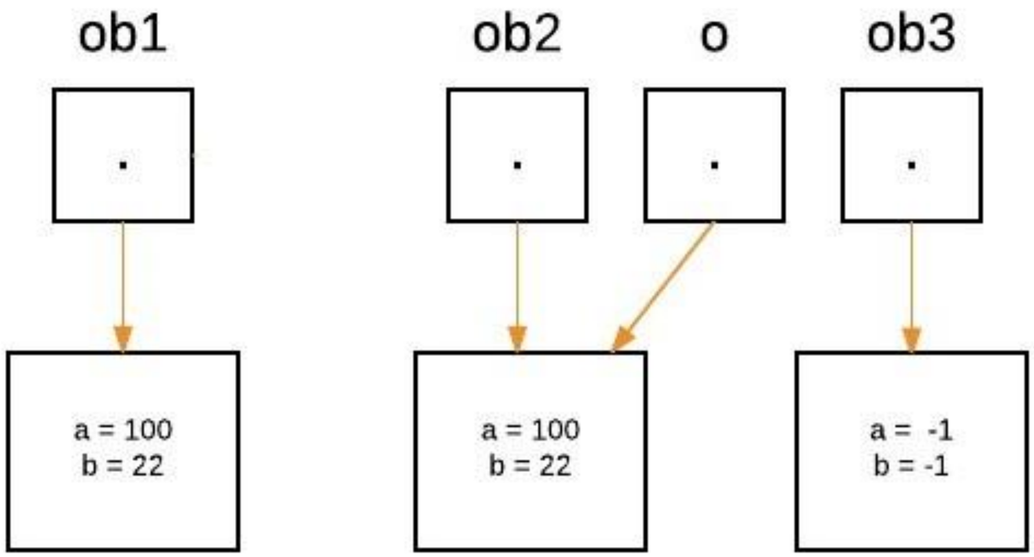


- boolean equalTo(ObjectPassDemo o);



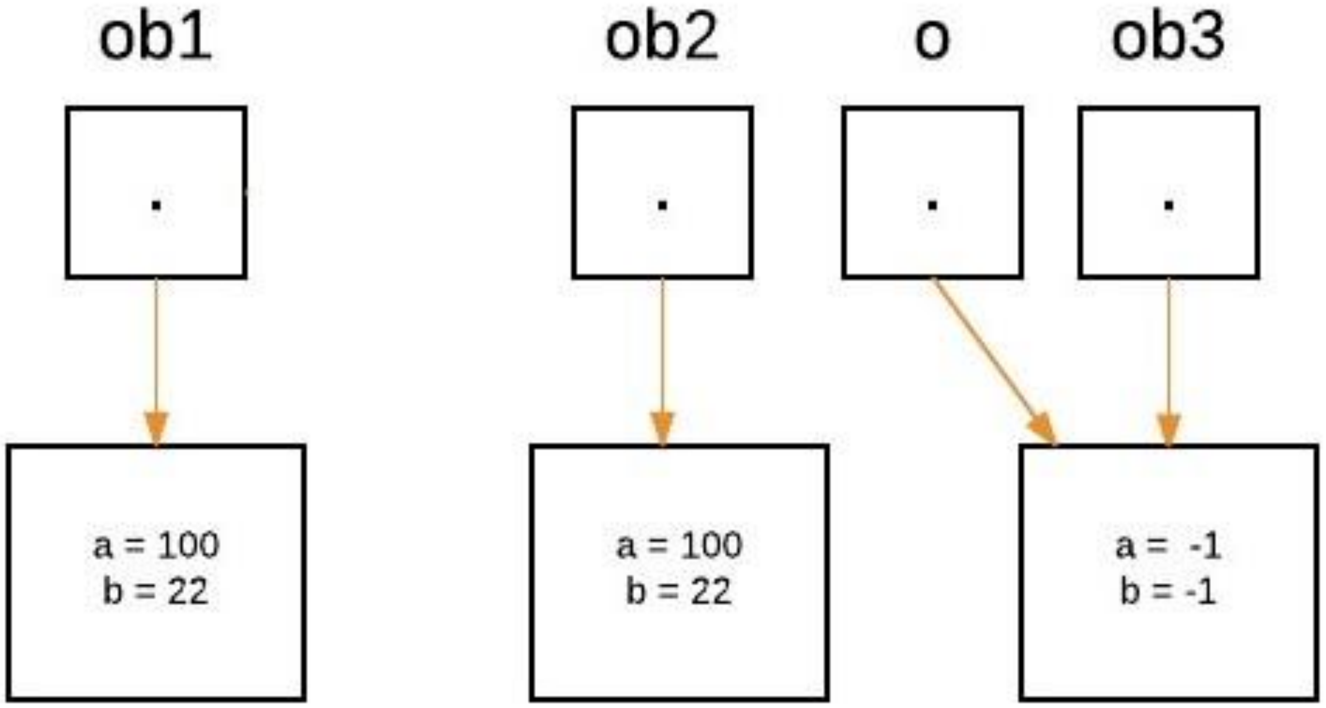
- As we call the method equalTo, the reference ‘o’ will be assigned to the object which is passed as an argument, i.e. ‘o’ will refer to ‘ob2’ as following statement execute.
- ob1.equalTo(ob2)

```
System.out.println("ob1 == ob2: " + ob1.equalTo(ob2));
```



- Now as we can see, equalTo method is called on ‘ob1’ , and ‘o’ is referring to ‘ob2’. Since values of ‘a’ and ‘b’ are same for both the references, so if(condition) is true, so boolean true will be return.
- if(o.a == a && o.b == b)

- Again ‘o’ will reassign to ‘ob3’ as the following statement execute
- System.out.println("ob1 == ob3: " + ob1.equalTo(ob3));

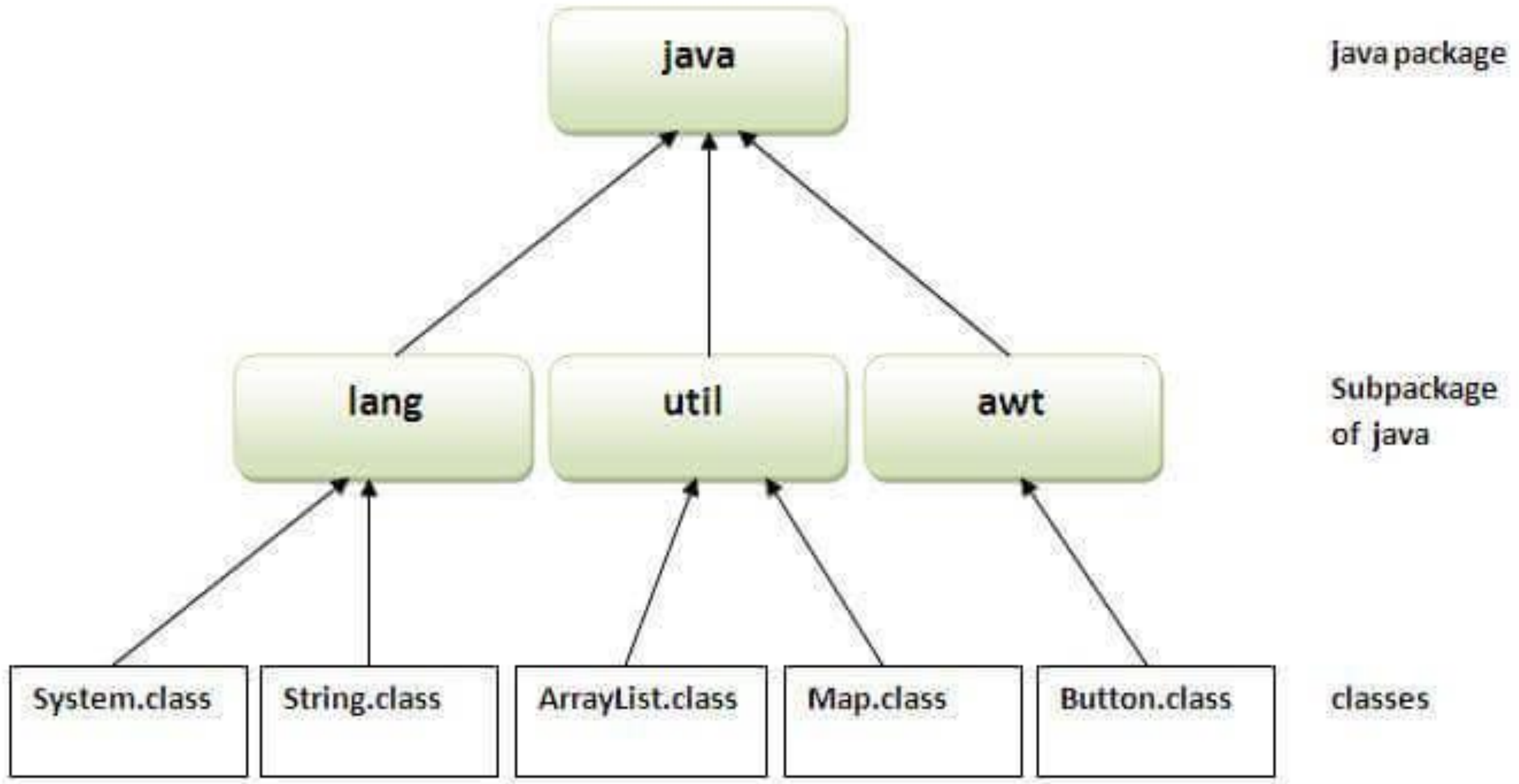


- Now as we can see, equalTo method is called on ‘ob1’ , and ‘o’ is referring to ‘ob3’. Since values of ‘a’ and ‘b’ are not same for both the references, so if(condition) is false, so else block will execute and false will be return.

## Java Package

- A **java package** is a group of similar types of classes, interfaces and sub-packages.
- Package in java can be categorized in two form, built-in package and user-defined package.
- There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.
- Java package is used to categorize the classes and interfaces so that they can be easily maintained.
- Java package provides access protection.
- Java package removes naming collision.

# Package


java
java package
lang
util
awt
Subpackage of java
System.class
String.class
ArrayList.class
Map.class
Button.class
classes

## Sample program

```
package mypack;
public class SamplePack
{
   public static void main(String args[])
   {
     System.out.println(“My first package");
   }
}
```

There are 3 ways to access the package from outside the package.

1.import package.*;

2.import package.classname;

3.fully qualified name.

## Access Modifiers in Java

1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

- **Default access modifier**
  - When we do not mention any access modifier, it is called default access modifier.
  - The scope of this modifier is limited to the package only.
  - This means that if we have a class with the default access modifier in a package, only those classes that are in this package can access this class.
  - No other class outside this package can access this class.
  - Similarly, if we have a default method or data member in a class, it would not be visible in the class of another package.

## protected Keyword

- private members can be accessed only within the class
- public members can be accessed from anywhere
- You can also assign methods and fields protected. Protected members are accessible
- from within the class
- within its subclasses
- within the same package

|  | Class | Package | Subclass (Same package) | Subclass (diff Package) | Outside class |
|---|---|---|---|---|---|
| Public | Yes | Yes | Yes | Yes | Yes |
| Protected | Yes | Yes | Yes | Yes | No |
| Default | Yes | Yes | Yes | No | No |
| private | Yes | No | No | No | No |