

METHODS IN JAVA

A method is a **collection of statement that performs specific task**. In Java, each method is a part of a class and they define the behavior of that class. In Java, method is a jargon used for method.

Advantages of methods

- Program development and **debugging are easier**
- Increases **code sharing** and **code reusability**
- Increases program **readability**
- It makes program **modular** and easy to understanding
- It shortens the program length by **reducing code redundancy**

Types of methods

There are two types of methods in Java programming:

- Standard library methods (built-in methods or predefined methods)
- User defined methods

Standard library methods

The standard library methods are **built-in methods** in Java programming to handle tasks such as mathematical computations, I/O processing, graphics, string handling etc. These methods are already defined and come along with Java class libraries, organized in packages. In order to use built-in methods, we must import the corresponding packages. Some of library methods are listed below.

Packages	Library Methods	Descriptions
<i>java.lang.Math</i> All maths related methods are defined in this class	acos() exp() abs() log() sqrt() pow()	Computes arc cosine of the argument Computes the e raised to given power Computes absolute value of argument Computes natural logarithm Computes square root of the argument Computes the number raised to given power
<i>java.lang.String</i> All string related methods are defined in this class	charAt() concat() compareTo() indexOf() toUpperCase()	Returns the char value at the specified index. Concatenates two string Compares two string Returns the index of the first occurrence of the given character converts all of the characters in the String to upper case
<i>java.awt</i>	add()	inserts a component

contains classes for graphics	setSize()	set the size of the component
	setLayout()	defines the layout manager
	setVisible()	changes the visibility of the component

Example:

Program to compute square root of a given number using built-in method.

```
public class MathEx {
    public static void main(String[] args) {
        System.out.print("Square root of 14 is: "+
            Math.sqrt(14));
    }
}
```

Sample Output:

Square root of 14 is: 3.7416573867739413

User-defined methods

The methods created by user are called user defined methods.

Every method has the following.

- Method declaration (also called as method signature or method prototype)
- Method definition (body of the method)
- Method call (invoke/activate the method)

Method Declaration

The syntax of method declaration is:

Syntax:

```
return_type method_name(parameter_list);
```

Here, the return_type specifies the data type of the value returned by method. It will be void if the method returns nothing. **method_name indicates the unique name assigned to the method.** parameter_list specifies the list of values accepted by the method.

Method Definition

Method definition provides the actual body of the method. The instructions to complete a specific task are written in method definition. The syntax of method is as follows:

Syntax:

```
modifier return_type method_name(parameter_list)
{
    // body of the method
}
```

ROHINI COLLEGE OF ENGINEERING AND TECHNOLOGY

Here,

Modifier	- Defines the access type of the method i.e accessibility region of method in the application
return_type	- Data type of the value returned by the method or void if method returns nothing
method_name	- Unique name to identify the method. The name must follow the rules of identifier
parameter_list	- List of input parameters separated by comma. It must be like datatype parameter1,datatype parameter2,..... List will be empty () in case of no input parameters.
method body	- block of code enclosed within { and } braces to perform specific task

The first line of the method definition must match exactly with the method prototype. A method cannot be defined inside another method.

Method Call

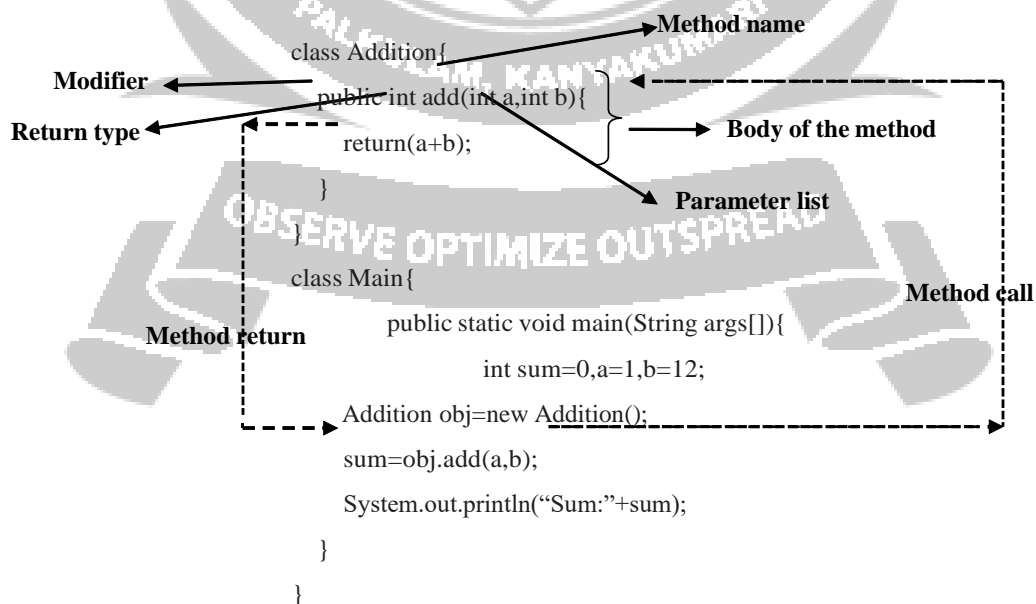
A method gets executed only when it is called. The syntax for method call is.

Syntax:

method_name(parameters);

When a method is called, the program control transfers to the method definition where the actual code gets executed and returns back to the calling point. The number and type of parameters passed in method call should match exactly with the parameter list mentioned in method prototype.

Example:



Sample Output:

Sum:13

Memory allocation for methods calls

Method calls are implemented using stack. When a method is called, the parameters passed in the call, local variables defined inside method, and return value of the method are stored in stack frame. The allocated stack frame gets deleted automatically at the end of method execution.



Types of User-defined methods

The methods in C are classified based on data flow between calling method and called method. They are:

- Method with no arguments and no return value
- Method with no arguments and a return value
- Method with arguments and no return value
- Method with arguments and a return value.

Method with no arguments and no return value

In this type of method, no value is passed in between calling method and called method. Here, when the method is called program control transfers to the called method, executes the method, and return back to the calling method.

Example:

Program to compute addition of two numbers (no argument and no return value)

```
public class Main{
    public void add(){           // method definition with no arguments and no return
        valueint a=10,b=20;
        System.out.println("Sum:"+(a+b));
    }
    public static void main(String[] args)
    {Main obj=new Main();
    obj.add();                   // method call with no arguments
    }
}
```

Sample Output:

Sum:30

Method with no arguments and a return value

In this type of method, no value is passed from calling method to called method but a value is returned from called method to calling method.

Example:

Program to compute addition of two numbers (no argument and with return value)

```

public class Main {
    public int add(){ // method definition with no arguments and with return value
        int a=10,b=20;
        return(a+b);
    }
    public static void main(String[] args)
    {int sum=0;
    Main obj=new Main();

    sum=obj.add(); // method call with no arguments. The value returned
                  /* from the method is assigned to variable sum */
    System.out.println("Sum:"+sum);
    }
}

```

Sample Output:

Sum:30

Method with arguments and no return value

In this type of method, parameters are passed from calling method to called method but no value is returned from called method to calling method.

Example:

Program to compute addition of two numbers (with argument and without return value)

```

public class Main {
    public void add(int x,int y){ // method definition with arguments and no return value
        System.out.println("Sum:"+x+y);
    }
    public static void main(String[] args)
    {int a=10,b=20;
    Main obj=new Main();
    obj.add(a,b); // method call with arguments
    }
}

```

Sample Output:

Sum:30

Method with arguments and a return value.

In this type of method, there is data transfer in between calling method and called method. Here, when the method is called program control transfers to the called method with arguments, executes the method, and return the value back to the calling method.

Example:

Program to compute addition of two numbers (with argument and return value)

```
public class Main {  
    public int add(int x,int y){ // function definition with arguments and return  
        valuereturn(x+y); //return value  
    }  
    public static void main(String[] args) {int a=10,b=20;  
  
        Main obj=new Main();  
        System.out.println("Sum:"+obj.add(a,b));  
    }  
}
```

Sample Output:

Sum:30

1.5 PARAMETER PASSING IN JAVA

The commonly available parameter passing methods are:

- Pass by value
- Pass by reference

Pass by Value

In pass by value, the value passed to the method is copied into the local parameter and any change made inside the method only affects the local copy has no effect on the original copy. In Java, parameters are always passed by value. All the scalar variables (of type int, long, short, float, double, byte, char, Boolean) are always passed to the methods by value. Only the non-scalar variables like Object, Array, String are passed by reference.

Note:

Scalar variables are singular data with one value; Non scalar variables are data with multiple values.

Example:

Pass by value

```

class Swapper{
    int a;
    int b;
    Swapper(int x, int y) // constructor to initialize variables
    {
        a = x;
        b = y;
    }
    void swap(int x, int y) // method to interchange values
    {
        int temp;
        temp = x;
        x=y;
        y=temp;
    }
}
class Main{
    public static void main(String[] args){
        Swapper obj = new Swapper(10, 20); // create object
        System.out.println("Before swapping: a="+obj.a+" b="+obj.b);
        obj.swap(obj.a,obj.b); // call the method by passing class object as
        parameterSystem.out.println("Before swapping: a="+obj.a+" b="+obj.b);
    }
}

```

/ only the local copy x, y gets swapped. The original object value a, b remains unchanged*/*

Sample Output:

Before swapping: a=10 b=20

After swapping: a=10 b=20

Here, to call method swap() first create an object for class Swapper. Then the method is called by passing object values *a* and *b* as input parameters. As these values are scalar, the parameters are passed using pass by value technique. So the changes carried out inside the method are not reflected in original value of *a* and *b*.

Pass by Reference

In pass-by-reference, reference (address) of the actual parameters is passed to the local parameters in the method definition. So, the changes performed on local parameters are reflected on the actual parameters.

Example:

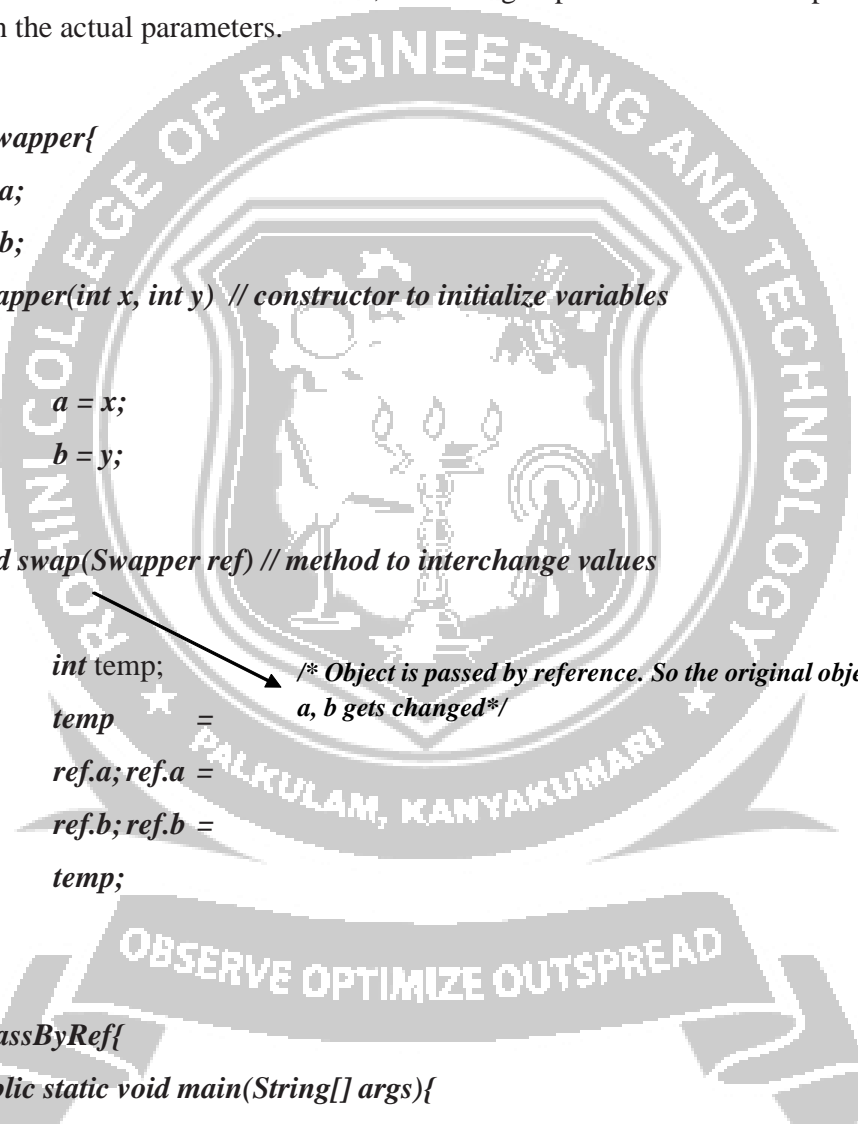
```

class Swapper{
    int a;
    int b;
    Swapper(int x, int y) // constructor to initialize variables
    {
        a = x;
        b = y;
    }
    void swap(Swapper ref) // method to interchange values
    {
        int temp;
        temp = ref.a; ref.a =
        ref.b; ref.b =
        temp;
    }
}

class PassByRef{
    public static void main(String[] args){
        Swapper obj = new Swapper(10, 20); // create object
        System.out.println("Before swapping: a="+obj.a+" b="+obj.b);
    }
}

```

/ Object is passed by reference. So the original object value a, b gets changed*/*



```

        obj.swap(obj); // call the method by passing class object as
        parameterSystem.out.println("After swapping: a="+obj.a+
        b="+obj.b);
    }
}

```

Sample Output:

Before swapping: a=10 b=20

After swapping: a=20 b=10

In this example, the class object is passed as parameter using pass by reference technique. So the method refers the original value of *a* and *b*.

Method using object as parameter and returning objects

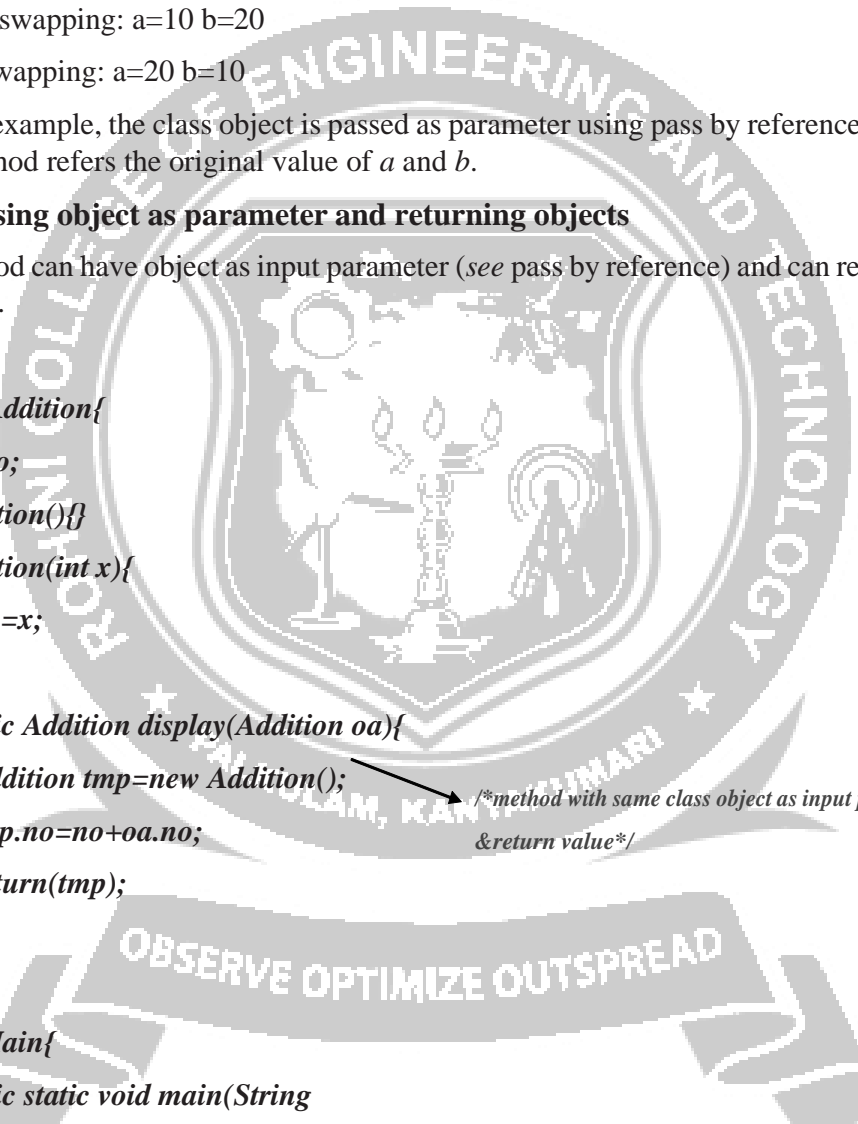
A method can have object as input parameter (*see* pass by reference) and can return a class type object.

Example:

```

class Addition{
    int no;
    Addition(){
    Addition(int x){
        no=x;
    }
    public Addition display(Addition oa){
        Addition tmp=new Addition();
        tmp.no=no+oa.no;
        return(tmp);
    }
}
class Main{
    public static void main(String
    args[]){Addition a1=new
    Addition(10); Addition a2=new
    Addition(10); Addition a3;
    a3=a1.display(a2); // method is invoked using the object a1 with input parameter a2
}
}

```



*/*method with same class object as input parameter &return value*/*

ROHINI COLLEGE OF ENGINEERING AND TECHNOLOGY

```
System.out.println("a1.no="+a1.no+" a2.no="+a2.no+" a3.no="+a3.no);  
}  
}
```

Sample Output:

a1.no=10 a2.no=10 a3.no=20

Here, display() accepts class Addition object a2 as input parameter. It also return same class object as output. This method adds the value of invoking object a1 and input parameter a2. The summation result is stored in temporary object tmp inside the method. The value returned by the method is received using object a3 inside main().

