Functions / Methods

DAY-4

A method is a block of code which only run when it is call.

Reuse : Define the code once & use it many time.

Syn:

```
public class main{
    static void mymethod(){
        //code
    }
}
```

In this method does not have any return value

Call function

Name of the method

```
public class main{
    access_modifier return_type method(){
        //code
        return statement; End func
    }
}
```

Return method → `return_type name(argument){ //body return statement; }`

Eg:

```
public class Sum{
    public static void main(String[] args){
        sum();
    }
    static void sum(){
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter n1: ");
        int num1 = sc.nextInt();
        System.out.print("Enter n2: ");
        int num2 = sc.nextInt();
        int sum = num1 + num2;
        System.out.print(sum);
    }
}
```

Call this method

O/p:

```
Enter n1: 5
Enter n2: 10
15
```

Call by value: Return Value

```
public class sum{
    public static void main (String [] args){
        int ans = sum2();
        System.out.println(ans);
    }
    public int sum2(){
        Scanner in = new Scanner (System.in);
        System.out.print (" Enter n1: ");
        int num1 = in.nextInt();
        System.out.print (" Enter n2: ");
        int num2 = in.nextInt();
        int sum = num1 + num2;
        return sum;
    }
}
```

Perform first

return this value as o/p

O/p:

```
Enter n1: 10
Enter n2: 20
30
```

Return string: Return value string

```
public class String{
    public static void main (String [] args){
        String msg = greet();
        System.out.println(msg);
    }
    static String greet(){
        String greeting = " How are you ";
        return greeting;
    }
}
```

Return

O/p:

```
How are you
```

Parameter (Int function)

```
public class Sum{
    public static void main (String [] args){
        int ans = sum (a:20, b:10);
        System.out.println (ans);
    }
    static int sum( int a, int b) {
        int sum = a+b;
        return sum;
    }
}
```

→ Values are assigned

→ Return O/p

O/p:
30

String function get I/p from user

```
public class String{
    public static void main (String [] args){
        int
        Scanner sc = new Scanner (System.in);
        System.out.print ("Enter Your Name:");
        String name = sc.next();
        String person = myGreet (name);
        System.out.print (person);
    }
    static String myGreet (String name){
        String msg = "Hello" + name;
        return msg;
    }
}
```

String Value from user

→ Final result

We can get value from the user.

O/p:

Enter Your Name: Priya

Hello Priya

① Primitive data type → int, short, char, byte Just pass value

② Obj & Ref → Pass value of ref variable

Swap value using Pass value:

```
public static void main (String [] args) {
    int a=10, b=20;
    swap (a,b);
    System.out.print (a + " " + b);
}
static void swap (int a, int b){
    int temp=a;
    a=b;
    b=temp;
}
```

Olp: 10 20

Just pass by value

Cannot swap the variables

Change value in array:

```
public class Change {
    public static void main (String [] args) {
        int [] arr = {1,3,5,7,9} ;
        Change (arr);
        System.out.print (Arrays.toString (arr));
    }
    static void Change ( int [] nums){
        nums[0]=99;
    }
}
```

This value are change to Array form

By this condition

→ Change arr value

Olp: [99,3,5,7,9]

↑ Pass value by ref variable

Abstract Class:

→ Used for classes or interfaces.

→ Abstract can only implemented using subclass or class that implement the interface

Syn: abstract type method_name (parameter_list);

Eg:

Abstract class

```
abstract class arithmetic {
    abstract void pointInfo();
}
class add extends arithmetic {
    void pointInfo() {  → Override.
        int a = 3;
        int b = 4;
        System.out.println(a+b);
    }
}
class sub extends arithmetic{
    void pointInfo() { → Override
        int c = 4;
        int d = 2;
        System.out.println(c-d);
    }
}
```

Driven Class

```
class abstraction{
  public static void main (String[] args){
      arithmetic x = new add();
      x.pointInfo();
      arithmetic y = new sub();
      y.pointInfo();
  }
}
```

Main function

O/p:

7
2

Abstract class with an abstract method

Eg:

```
import java.io.*;                          → Abstract Class
abstract class Geometry{
    Declaring abstract method
    abstract void square_area(int side);
    abstract void circle_area(float radius);
}
Extending Abstract Class
class Easy extends Geometry{               → Implementing Abstract Method of Abstract Class
    public void square_area(int side){
        int ar = side * side;
        System.out.print("Area of Square:" +ar);
    }
    public void circle_area(float radius){
        int ar = 3.14f * radius * radius;
        System.out.print("Area of Circle:" +ar);
    }
    Main function
    public static void main(String args[]){
        Easy obj = new Easy();
        obj.square_area(12);               Calling abstract method
        obj.circle_area(2.2f);
    }
}
```

Op:

Area of Square: 144

Area of Circle = 15.197601

Abstract Method in interface:

```
import java.io.*;
interface Sum{                                  Declaring Interface
    public abstract int Operation (int a, int b);   Declaring abstract method inside interface
    int Operation_three (int a, int b, int c);
}
Main Class
public class GFG implements Sum{
    public int Operation (int a, int b) {
        return a*b;
    }
    public int Operation_three (int a, int b, int c){
        return a*b*c;
    }
Main Function
    public static void main (String args []){      Operator the declared
        Sum obje = new GFG ();
        System.out.println (obj.Operation (10,20));
        System.out.println (obj.Operation_three (10,20,30));
    }
}
```