*Functions/Methods in java

Page No.
Date :

- A method is a block of code which only runs when it is called.
- To reuse code: define the code once, & use it many times.

Syntax:

```
public class Main{
    static void myMethod(){
        //code
    }
}
```

This method does not have a return value.

name of method

```
Public class main{
    access-modifier return-type method(){
        //code
        return statement;
    }
}
```

"n" ends here

- method() → calling the function

- return-type :-
A return statement causes the program control to transfer back to the caller of a method. A return type may be primitive type like int, float or void type (return nothing)

Page No.
Date :

⇒ There are a few important things to understand about returning the values:

- The type of data returned by a method must be compatible with the return type specified by the method
eg: if return type of some method is boolean, we can not return an integer.

- The variable receiving the value returned by a method must also be compatible with the return type specified for the method.

⇒ Pass by value

eg1:

```
main(){
    name = 'a';
    greet(name);
}
static void greet(naam){
    print(naam)
}
```

object/value

Creating copy of value of name i.e passing value of the reference

name → a
naam → a

eg2:

```
PSVM(){
    name = "a"
    change(name);
    print(name);
}
change(naam){
    naam = "b";
}
```

name → a
naam → a

name → a
naan → b

since it is created inside f° it will not change original

not changing original object just creating new object

*A copy of the value of reference variable is passed in function.

Page No.
Date :

*Points to be noted:

1) Primitive data type like int, short, char, byte etc..
   ↳ just Pass value.

2) object & reference:
   ↳ Passing value of reference variable

eg-1:

```
psvm (){
    a = 10;
    b = 20;
    swap(a,b);
}
swap(num1, num2){
    temp = num1;
    num1 = num2;
    num2 = temp;
}
```

a → 10
b → 20
] but not here.

temp → 10
num1 → 20
num2 → 10
] at 1st scope level they are swapped.

Here they just passes the value....

eg-2): arr → [1,2,3,4,5]

nums ↗

nums[0] = 99 [now the value of 0th Position in nums will change which also changes value of arr[0]]

arr → [99, 2, 3, 4, 5]

nums →

Here Passing value of reference variable.

Page No.
Date :

* Scopes :-

- **Function scope** –
  variables declared inside a method (Ruc) scope (means inside method) can't be accessed outside the method.

eg:-

```
psvm C){
    x ;
}
all C){
  int x ;
}
```

can't be accessed outside

- **block scope** –
  Anything that is initialized outside the box. block can be used inside the block, but anything that is intialized inside the block can not be used outside the block.

  Anything that is initialized outside the block can not be intialized inside the ~~bo~~ block, but anything that is initialized inside the block can be reintialized outside the block.

```
psvm c){
   int a = 10;
   int b = 20;
      {
         int a=5 ; X
          a = 100 ; ✓
         int c = 20;
      }
   c = 10; X
   int c = 15; ✓
   a= 50; ✓ → variables like "a" here is declared outside the block, updated inside the block and can also be updated outside the block
}
```

- **Loop scope —**
  variables declared inside loop are having loop scope

※ Shadowing

Shadowing in Java is the practice of using variables in overlapping scopes with the same name where the variable is low level scope overrides the variables of high-level scope. Here the variable at high-level scope is shadowed by low-level scope variable.

Page No.
Date :

eg:- public class shadowing {
    static int x = 90;
    psvm() {
        system.out.println(x); // 90
        x = 50;   // here high level scope is
        system.out.println(x); // 50 shadowed by low level scope

※ Variable Arguments :-
variable arguments is used to take a variable numbers of arguments. A method that takes a variable number of arguments is a varargs method

Syntax :-
    static void fun(int ...a){
        // method body
    }
Here, would be array of type int[] parameters

※ Method / Func^n overloading :-
Func^n overloading happens when two func^n have same name.

eg. 1) fun(){
        // code
    }
    fun(){
        //Code
    }
X func^n overloading

Page No.
Date :

2) fun(int a) {
    //code
}
fun(int a, int b){
    //code
}

This is allowed having different arguments with same method name.

At compile time, it decides which f^n to run.