functions/methods in java

Functions / Methods :

- 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() {        // This method doesn't have a return value
        // code                     // mymethod -> name of the method
    }
}
```

```
Public class main {
    access-modifier return type method() {
        //code
        return Statement; } function ends here
    }
}
```

- method() -> calling the function
  - method -> name of the function
- return-type :-

  A return statement causes the program control to transfer back to the caller of the method. A return type maybe premitive type like int, char or) void type (returns nothing)

=> 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 cannot return an integer

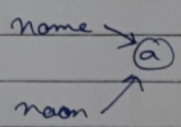
- The variable recieving the value returned by a method must also be compatible with the return type specified for the method

=> Pass by Value:

Eg: ①

```
main () {
    name = "a"  ---> object / value
    greet (name);
}
static void greet (naam) {
    Print (naam)
}
```

Creating copy of value of name

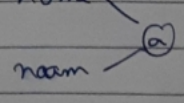


ie: Passing value of the reference

Eg 2:

```
Psvm () {
    name = "a"
    Change (name);
    Print (name);
}
Change (naam) {
    naam = "b"
}
```

Creating copy



name → a
naam → b

Since it is creating inside function it will not change the original one

* Points to be noted :

- Primitive data type like int, short, char, byte etc
  ↳ just pass value

- 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 function scope level they are swapped

Here, they just passes the value

Eg -2 :

arr → [1,2,3,4,5]

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

arr → [99,2,3,4,5]
↑
nums

Here, Passing value of reference variable

* Scopes :

- Function scope :
  Variables declared inside a method / function scope (means inside method) can't be accessed outside the method

Eg : Psvm (){

}

all () {
  int x; → can't be accessed outside
}

- block scope :-

Psvm () {
  int a = 10;
  int b = 20;
  → Variables initialized outside the block can be updated inside the box
  {
    int a = 5; ✗
    a = 100; ✓
    int c = 20;
    → Variables initialized inside the block cannot be updated outside the box but can be reinitialized outside the block
  }
  c = 10;
  int c = 15;
  a = 50;
  → Variables like 'a' here is declared outside the block, updated & inside the block 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 variable of high level scope. Here the variable at high-level scope is shadowed by the low-level scope variable

Eg :-

```
Public class Shadowing {
        static int x = 90;
        PSVm () {                          -> 90
            System.out.println(x);
            x = 50;
            System.out.println(x);
                                           -> 50;
        }
}
```

here high level scope is shadowed by low level scope //

=> Variable arguments :-

Variable arguments is used to take a variable number of arguments. A method that takes a variable number of arguments is a Variable Var args method

Syntax :

```
static void fun (int ...a) {
        // method body

}
```

Parameters

Here, a would be array of type int[]

=> method/function overloading :-

function overloading happens when two functions have same name

Eg: ① fun() {
    // code
}
fun() {
    // code
}

X Function Overloading

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 function to run

=> Armstrong number :

Suppose there is number → 153

153 → $(1)^3 + (5)^3 + (3)^3 = 1 + 125 + 27$
= 153

PDF Created Using

Camera Scanner

Easily Scan documents & Generate PDF

https://play.google.com/store/apps/details?id=photo.pdf.maker