FUNCTIONS / METHODS

• Functions/ Methods Introduction.

⇒ a block of code which only runs when it is called.

⇒ it is defined once and reused multiple times.

Syntax :-

```
access modifier return-type name () {

        // body

         return statement   ← ends here

     }
```

⊙ 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, char or void type (returns nothing).

★ The type of data returned by a method must be compatible with the return type specified by the methoc

★ The variable recieving the value returned by a method must also be compatible with the return_type specified for the method.

★ No pass by reference in JAVA
only pass by value.

⊙ Arguments.

⇒ these are used to pass the value of numbers/ arguments when you are calling the f^n in the main() method.

• Pass By Value

```
main () {
    name = "XYZ";
    greet (name);
}
```

object

$$\begin{tikzcd} \text{name} \arrow[r] & \text{"XYZ"} \\ \text{naam} \arrow[ur] & \end{tikzcd}$$

cope of the value of reference variabu

```
greet (naam){
      print (naam)
}
```

```
// Swapping name

psvm () {
     name = "XYZ"
     change (name)
     print (name)
}
```

this will give XYZ

$$\begin{tikzcd} \text{name} \arrow[r] & \text{"XYZ"} \\ \text{naam} \arrow[ur] & \end{tikzcd}$$

```
change (naam){
    naam = "ABX"
}
```

naam —— "ABX"

this does not change where name was Pointing to

★ 1. primitive data type :→ just pass value
2. object & reference :→ pass value of reference

eg
```
psvm () {
    a=10
    b = 20
    swap(a, b)
}
```

a → 10
num1 → 10

b → 20
num2 → 20

```
swap (num1, num2) {
    temp = num 1;
    num1 = num2;
    num 2 = temp;
}
```

temp → 10
num1 → 20
num2 → 10

at f^n scope level they are swapped

but original one is not changed

eg:-

$$\begin{tikzcd}
\text{arr} \arrow[r] & {[1, 3, 2, 45, 6]} \\
\text{nums} \arrow[ur]
\end{tikzcd}$$

→ pass by the copy of the value of references

nums [0] = 99

$$\begin{tikzcd}
\text{arr} \arrow[r] & {[99, 3, 2, 45, 6]} \\
\text{nums} \arrow[ur, no head]
\end{tikzcd}$$

★ strings are final classes, you cannot modify it. they are immutable for security reasons.

• Scoping

1. **f^n scope**

variables defined/declared inside a method / f^n scope (means inside method) can't be accessed outside the method.

2. **block scope**

```
psvm() {

    int a = 10;      ] variables initialised outside the
    int b = 20;      ] block can be updated inside the
                       block

  {
      int a = 5     ✗   ] variables initialised inside
      a = 100;      ✓   ] the block cannot be updated
      int c = 20;   ✓   ] accessed outside the block
  }

  c = 10      ✗   ] thus variables declared inside
  int c = 15;  ✓  ] inside the block must be
   a = 50     ✓   ] reinitialised outside the block
}                    to be used outside.
```

variables like 'a' that are declared outside the block, can be updated inside + outside the block.

3. **loop scope**

variables declared inside loop ~~scop~~ are having loop scope.

• Shadowing
practice of using variables with the same name in overlapping scopes where the variable in low-level scope overrides the variable of high level scope.

eg:-
```
public class Shadowing () {

    static int x = 90;
    psvm () {

        int x ;

        x = 40 ;
    }
}
```

sscope of this local variable that shadows the class variable is not beginning at the point where the local scope begins but shadowing begins actually when local variable is declared.
└→ x = 40 ;

• Variable Arguments .
variable arguments is used to take a variable no. of arguments. A method that takes a variable number of arguments is a varargs method.

function called be called by more than one argument.
the input will always be of type array.

variable length arguments always at end.

• function Overloading /Method overloading.

Two or more functions can exist of the same name if the parameters are different.

eg:-

```
fun (int a) {
        // code
}

fun (String a) {
        // code
}
```

This is allowed. having different arguments with same method name.

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