★ lecture 7 :- Functions / Methods in Java

#1) What is the use of Methods in Java :-

Suppose, you have to make a program,

Take ~~an~~ 2 input from user and print sum.
↳ Code :- P30 SimpleSum.java

And, suppose you have to make this program 10 or 100 or 1000 times,

So if you copy - paste it 100 times or if you want to change variables of every sum block of 1000 block so is it optimal to change in copied - pasted way?

No,

That's why, we use Methods.

Methods are nothing but a block of code that you can call again and again whenever needed.

#2.) Syntax of a Method;-

→ access Modifier(learn in OOP's) static/non-static returntype ~~name~~ name(parameters) {

① ② ③ ④

⑥ //body of the fun/Method

⑤ return statement;

}

~~Explanation of Each word:-~~

→ Program follow above syntax;- P31MethodSum.java

package com.tejas;
import java.util.Scanner;

public class P31MethodSum ~~java~~
{
public ~~public~~ static void main(String[] args)
{

```
    ~~Scanner sc = new Scanner(System.in);~~
    sum();
  }

             (2)      (3)      (4)
  ~~public~~ static void sumfu() {
      Scanner sc = new Scanner(System.in);
(6)   System.out.printla("Enter 1st no: ");
       int num1 = sc.nextInt();

      System.out.print("Enter 2nd no: ");
       int num2 = sc.nextInt();

      int sum = num1 + num2;
      System.out.println("The sum is: " + sum);
  }

}
```

→ Explaination of every thing :-

(1) Access Modifier :-

These use to specify how a class or fu^{ns} or variables can be accessed by another classes etc

ex;- public
private
protected etc.

(We learn about them in OOP's)

(2) static / non-static :- This method's static and main method's static is linked.

2 scenarios :- [Code : 1) P32 MethodSum SameClass. java &
2.) P33 MethodSum ~~Same~~ Another Class. java]

1.) Your method is in same class as main method :-

In this case, ~~everp~~ a.) → Eeither make obj of same class and call that fu^n (doesn't matter if you use static in fu^n making or not)

b.) → or can directly call that fu^n (but only if you've used static during fu^n making)

ex :-

```
public ~~static~~ class P31 MethodSum {

    public static void main (String[] args)
    {
  b. ~~a~~)   sumfu ();
             // or
  a ~~b~~)   P31MethodSum obj = new P31MethodSum ();
             ~~obj. MethodSum()~~ .
             obj. sumfu ();
    }
    static void sumfu () {
     Scanner sc = new Scanner (System. in);
 System. out. println ("Enter 1st no: ");
   int num1 = sc. nextInt();
 System. out. println ("Enter 2nd no : ");
   int num 2 = sc. nextInt ();
 sum = num1 + num 2;
 System. out. print ("Sum is: " + sum);  } }
```

2.) Your fu^n is in another class of main class:-

In this case, you have to make an object of that class in order to call that fu^n and it doesn't matter that you use static keyword while making that fu^n.

ex:- code:- P 33 MethodSum AnotherClass.java

Conclusion:-

If fu^n is:-

1.) In ~~th~~ same class
- → making obj and calling fu^n with object work
- → If used static also in that fu^n making which is in same class as main method.
  - → can directly call fu^n by its name with brackets ex:- sumfu(); in main method

2.) In another class
- → can only call by making obj of contained class in main method and then call fu^n with help of that object.
- → Doesn't matter if you use static or not ~~with~~ while creating that fu^n in another class.

Now in 1st senario, we can use static to directly call fun with its name ~~by~~ without makin obj as both main method and our sum method is in same class and both are static.
That's why main method's static and ~~for~~ our sum method's static is linked.
Another example:- P34 Method Greetings.java

③ Return-type:- Return type of a fun is linked i.e ③ with return statement of that fun i.e ⑤.

2 senarios:-

1) We use void:- We use void as return type when there is no return statement
ex:- printing greetings
↳ this fun prints greetings on screen only.
ex:- P34 MethodGreetings.java
or P31 MethodSum.java
or P32 MethodSumSameClass.java
or P33 MethodSumAnotherClass.java

2) Use of datatypes as return-types:- When we use datatype as return type, the fun ~~is~~ have return statement and returning some value of same datatype as return type.

- Why wee use return statement?

To get the value from a fu^n in main for its further processing

ex:- Taking sum of 2 no. from a sumfu() fu^n to mul. it with 2 in main.

⇓

Here, fu^n returning value in int and stored in variable of a that is also of type int and then further gets multiplied by 2.

Code:- P35 SumMul.java

- Note that, Return statement also means that fu^n ends there and below return statement any statement gives error.

- Another example can be Return String values from methods like wee return ~~sum~~ integer values from method in P35 SumMul.java.

Code:- P36 ReturnString.java

④ Paramaters / Arguments :-

Let's use P36 ReturnString.java to understand Arguments.

Suppose, you are calling greet() method. 50 times in Main but on each calling you have to give name as input because we used scanner to take name inside greet() method.

To avoid this, we use parameters where single value can use again & again in multiple fu^{ns}.

or

we can say it like, pass the value of strings whenever you calling the method in main, instead of providing value of string (name). each time it calls because of scanner inside method.

Code;- P37 ArgumentsonReturnString.java

```
package com.tejas;                                   //1.
import java.util.Scanner;                            //2.

public class P36ArgumentsonReturnString {            //3
public static void main(String[] args) {             //4

    Scanner sc = new Scanner(System.in);             //5
     String name = sc.nextLine();                    //6
    String message = greet(name);                    //7
   System.out.println(message);                      //8

}                                                    //9
Static String greet(String name)                //10
  {                                              //11
     String greetings = "Hello" + name;          //12
     rethurn greetings;                          //13
  }                                   //14
}                          //15
```

in method in main

Note;- The variable name in Line 10 and Line 6,7 are need not to be same like Line 10 one can be naam while Line 6,7 one is still name. The program still work correctly because it is value that passed in Arguments not variables. [Discussed just after ⑤ and ⑥ point in the) Pass value]

Another example;- Code;- P38ArgumentsonReturnInt.java

⑤ Return statement;- Already explained in ③

⑥ Body of Method;- Every code is written in body itself so no need to explain it here.

#3.) Pass by Value :-

a) for primitives and strings :- Value ~~of~~ is passed from one ~~value~~ variable (Reference variable) to another.

ex① Code :- P40PassByValueString.java

↳ here name → "Tejas Singh"

naam ↗ (When passed in greet(name))

but then,

name → "Tejas Singh"

naam → "Nahi Dikhayunga" (updated within greet method)

Now, if you do,

```
System.out.println(name);
System.out.println(greet(name)); //provided that greet method return naam as in our code
```

O/P :- "Tejas Singh"
"Nahi Dikhayunga"

Note :- Here we ~~updated~~ created the new value of naam instead of changing it or updating it the one we getting from name because in strings we can't update or change once provided value of any variable because string are immutable for security reasons.

ex ② :- Code :- P41PassByValueInt.java

↳ Here, a → 10
num1 → (from swap parameters)

b → 20
num2 → (from swap parameters)

then, a → 10
~~[illegible]~~ → 10
~~temp~~ →
num2
num1 → 20

[swapped within method]

Note :- Here, we ~~also~~ updated ~~created~~ the ~~news~~ values for num1 & num2 because we use those a & b values i.e 10 & 20 and swapped it with help of temp variable

Primitives are mutable unlike strings

b.) Objects & Arrays & Stuff :- passing value of the reference

$\boxed{arr} = \underbrace{[1, 2, 3, 4 \ldots]}_{\text{multiple values}}$

⇓

this is the value of reference means, value of label of reference ~~value~~ variable is passed instead of value i.e, [1,2,3,4 . .] because ~~it value~~ the simple value can't be passed like primitives and strings because here multiple values exists.

ex:- Code :- P42PassBy~~label~~ValueofReference.java

↳ Here,

$$\begin{tikzcd}
arr \arrow[r] & {[1, 2, 3, 4, 5]} \\
nums \arrow[ur] &
\end{tikzcd}$$

⇓

this is not pass by reference, but pass by copy of value of reference. [as per Kunal]

or,

$$\begin{tikzcd}
arr \arrow[r] & {[1, 2, 3, 4, 5].} \\
nums \arrow[u] &
\end{tikzcd}$$

[as per me] cos it is pass by label value of reference

So, nums[0] = 99

meass

nums[0] → arr[0] → [99, 2, 3, 4, 5]

- Why we are calling it pass by value of reference or pass by copy of value of reference ~~of~~ or pass by label value of reference varible it's because java doesn't have concept of address, pointers etc.

Doubt;- How String is immutable & primitives are not with reference to P40PassByValueString.java and P41PassByValueInt.java.

#4.) Scope;-

Scope means where we can access our variables;

3 types;-

1.) Method Scope
2.) Block Scope
3.) loop Scope.

1.) Method Scope;-

Any variable you created inside method (including Arguments variables one) can only be accessed within that method.

ex:- Code:- P43MethodScope.java

main {

2) Block Scope:- Suppose we created block { inside main

}

}

~~a)~~

a) Anything declared/ initialized within block will only has it scope within block itself

b.) Any variable declared/ initialized outside block (i.e in main method) cannot be reinitialised inside block but only can be updated.

ex:- Code:- P44BlockScope.java

3.) Loop Scope:- Works same as block code

ex:- Code:- P45LoopScope.java

#5.) Shadowing:-

3points:-

1a) Class variable (global variables) have scope all over the class. But sometimes local variables (method, constructor variables etc) are created locally with same name as class variable then priority is given to local variable within local scope over class variable. This is called shadowing or data shadowing.

Note ;- Data shadowing has its scope only within a single class unlike Data hiding where the scope is within parent and child classes [Data hiding is not useful in interview terms]

ex ;- Code ;- P46 Shadowing. java

```
        package com.tejas;

        import java.util.Scanner;
        public class P46Shadowing{
            static int x=90; // class variable declared &
class variable                  initialized
x scope
except this public static void main(String[] args){
                System.out.println(x); //gives 90 cos it is
                                              class variable
                int x; //declared local variable
                                   with same name as class variable
local x         System.out.println(x); //error cos x(local one) is not
scope                                               initialized
                x = 40; //initialized local variable x
                System.out.println(x); //printed local variable x
                fun(); //print 90 although its within local variable x
                         scope but it drives its value from class
                         variable x scope.
            }
            static void fun(){
                System.out.println(x);
            }
        }
```

#6.) Variable length Arguments / Variable Arguments / Var Args :-

Using infinite parameters in method Arguments

ex :- code :- P47 VarArgs.java

#7.) Method overloading :-

Same ~~variable~~ method name with different parameters. It is done at compile time unlike Method overriding which happens at runtime (See in OOPs)
ex :- Code :- P48 MethodOverloading.java

Note :- example :- Code :- P49 VarArgs And MethodOverloading.java

↑
used varargs + Method Overloading

Explained Ambiguity [See code]

#8.) Practice Questions :-

Q1. Prime Number :- Code :- P50 PrimeNumbers.java

Q2. Armstrong Numbers :- Code :- P51 ArmStrong Numbers.java