Lecture 31

Methods in Java

- Methods/Functions are block of code that perform a specific task and are executed when called.
- Methods allows us to write code once and use it multiple times.
- Instead of duplicating code, we can call a method whenever we need its functionality, saving time & reducing redundancy.

Syntax:-

Parameters, or formal parameters

```
accessModifier return-type methodName ( parameterlist)
{
        // Code block
        return somevalue;
}
```

return – a keyword

Components:-

accessModifier:- controls the visibility of method (e.g public, private, protected)

ReturnType:- The type of the data returned by the method (e.g.- int, String) we use void when method returns nothing

methodName:- name of the method [a valid identifier]

Parameters:- There can be many parameters (formal parameters) But parameters are optional. Generally input for the method is specified within parameters.
If there are no parameter then it means the method does not take any input.

and parameter is simply a variable. So we have to specify data type of the parameter as well. It can be a primitive or object reference. If no parameters are needed, leave the parantheses empty.

and

⇒ So basically we can say methods will take some input, process it & return some value (the output). and the value is returned using return statement.

Method calling:-

methodName (Argument list);

↑
arguments or Actual Parameter

NOTE:- The Argument list in method calling must match with the parameter list in method definition. meaning the no. of arguments must match with the no. of parameters & datatype of the data passed in calling must match with the datatype of parameters.

e.g:-

```
int sum (int a, int b) {
    int c = a+b;
    return c;
}
```

} Method Definition

int result = sum (5, 4);] ⇒ Method calling.

if I write:→ int result = sum(5.2, 4);] ⇒ will give error

↓

Here a = 5.2, b = 4 but datatype of a is int in method definition & we are passing float value in arguments.

2⇒ and if I write:-

int result = sum(5); ⇒ will give error.

because here we are passing only one argument but in definition there are 2 parameters (a & b).

return:-

return is used to exit a method & optionally send a value back to the method's caller.

Syntax:-

```
return ;          // used in methods with void return Type
return somevalue; // used in methods that
                     return a specific value
```

e.g:- ① using only return in void type methods :-

```
public class Demo{
 static void findSquareRoot(int number){
        if(number <0){
          System.out.println(" Can not find square root of
                          a negative number! ");
          return;
        }
      System.out.println(Math.sqrt(number));
  }
  public static void main(String[] args) {
       findSquareRoot(25);  →//caller
  }
```

this is the case of exiting on invalid input ⇒

when encounters with return the control passes back to the caller

→ So a return without returning any value is used to exit the method early. This is useful for controlling the flo of execution in cases where we want to stop further processing under certain conditions.

e.g:- early exit
while dividing 2 numbers (num1/num2) → if num2 is 0 then return.

② Using return in non-void methods:-

e.g:-
```
public class Demo {
static int add (int a, int b) {
        return a+b;
}
public static void main (String[] args) {
    int result = add(2,3);
    System.out.println (result);
}
```

③ Multiple return :-

```
public class Demo {
  static String greet () {
     boolean isMorning = true;
     if (! isMorning) {
           return "Good Morning";
     }
     else {
           return "Hare Krishna";
} }
public static void main (String[] args) {
 System.out.println ( greet()); }
```

OTE:- • The return type of the method and the type of value that method returns must match. (there should be of compatible type)

• Once a return statement is reached, the method exits immediately and any code after it will not execute

• return type of method can be primitive, void, array, class, interface.

Quiz Question:- ①

output?

```
public class Demo{
    public static void main(String[] args) {
        int a = 3;
        if(a <= 3) {
            System.out.println("Exiting from main()...");
            return;
        }
        System.out.println("a is greater than 3");
    }
}
```

Answer:- Exiting from main()...

②

output?

```
static int add( int num1; num2) {
        return num1+num2;
    }
public static void main(String[] args) {
        float result = add(2,3);
        System.out.println(result);
```

Answer:- ~~3~~ 5.0

NOTE:- while method declaration/definition there must not be anything between return type and name of the method.

→ One method can call another method. It is not compulsory th
we have to call all the methods in main directly.

eg:-
```
static int add (int num1, int num2){
            return num1 + num2;
    }
static double average(int num1, int num2){
            double sum = add(num1, num2);
            return sum/2 ;
    }
public static void main (string[] args) {
        System.out.println(add(2,3));
        double result = average(5, 2);
        System.out.println(result);
    }
```

Here we are resusing the method ~~[illegible]~~ add() to find average of 2 numbers. We can reuse this method add in any number of methods ~~for~~ or any number of times and in another project too.

Method Signature:- In Java Method Signature is a unique identifier for a method and it includes method name & parameter list.

eg:-
```
int add (int a, int b) {
        return a+b;
    }
```
⇒ Signature for this method is: add (int, int)

→ It is used to uniquely identify and differentiate methods within a class, especially when overloading methods.

* Need or benefits of methods (why use methods)?

$\boxed{1}$ **Code Reusability**

- ↳ write code once & use it multiple times
- ↳ reduce redundancy / duplicate code

$\boxed{2}$ **Modularity & Organization**

↳ Methods divides a program into smaller, manageable parts or modules, making it easier to understand & debug. Each method is responsible for a specific task keeping code organised & structured.

$\boxed{3}$ **Code Readability**

↳ Code becomes more readable & self-explanatory by grouping related statements into a single method with a descriptive name.

$\boxed{4}$ **Easier debugging & Testing**

↳ Methods allow us to isolate code for specific task, making it easier to locate & fix errors.

$\boxed{5}$ Methods helps in achieving encapsulation, data hiding & abstraction.

$\boxed{6}$ Improve collaboration / teamwork