Java – Methods (Complete and Detailed Notes)

Introduction to Methods:

- A method is a block of code designed to perform a specific task.
- It improves code reusability, modularity, and readability.
- In Java, methods are defined inside classes and can be called to execute whenever needed.

Why Use Methods?

- Avoid repeating code.
- Organize code into logical sections.
- Perform operations on parameters and return results.

Method Syntax and Structure:

```
returnType methodName(type param1, type param2, ...) {
   // method body
   return value; // optional if returnType is void
}
```

Example:

```
int add(int a, int b) {
   return a + b;
}
```

- methodName – name of the method (e.g., add)
- returnType – type of data the method returns (e.g., int, void)
- parameters – input values (optional)

Parameter Passing in Java:

Java uses Call by Value:

- Only copies of variables are passed.
- Original values remain unchanged outside the method.

```
void change(int x) {
   x = x + 5;
}
```

Method Types in Java:

Type	Description	Example
Predefined (Library) Methods	Provided by Java	System.out.println(), Math.max()
User-defined Methods	Created by developer	calculateSalary(), printName()
Static Methods	Belong to class; no object needed	main(), utility methods
Instance Methods	Require object to call	obj.display()
Main Method	Entry point of Java program	public static void main(String[] args)

Return Type in Methods:

Return Type	Meaning	Example
void	No return value	void display()
int, double, String	Returns a single value	int sum(), String getName()
boolean	Returns true/false	boolean isEven()
Object or Array	Returns complex data	int[] getMarks()

Variable Arguments (Varargs):

Accepts zero or more arguments of the same type.

```
void showNames(String... names) {
   for (String name : names) {
      System.out.println(name);
   }
}
```

Only one vararg is allowed, and it must be the **last parameter**.

Method Calling – Static & Non-Static:

Calling a Static Method:

```
class Example {
    static void greet() {
        System.out.println("Hello!");
    }

    public static void main(String[] args) {
        greet(); // or Example.greet();
    }
}
```

Calling a Non-Static Method:

```
class Example {
    void greet() {
        System.out.println("Hello!");
    }

    public static void main(String[] args) {
        Example e = new Example();
        e.greet();
    }
}
```

Method Overloading (Compile-Time Polymorphism):

- Same method name, different parameter list.
- Resolved at compile-time.

Note: Method overloading can't be done by just changing return type.

Recursion (Method calling itself):

A recursive method calls itself to solve a smaller problem.

Example: Factorial

```
int factorial(int n) {
   if(n == 1)
      return 1;
   return n * factorial(n - 1);
}
```

Static vs Non-static Methods:

Feature	Static	Non-static
Belongs to	Class	Object
Accessed by	Class name	Object
Access	Only static data	Both static & non-static
Example	Math.max()	emp.getSalary()

Constructors vs Methods:

Feature	Constructor	Method
Name	Same as class	Any valid name
Return type	None	Must have
Call	Automatically on object creation	Manually by name
Purpose	Initialize object	Execute logic

Java Main Method:

public static void main(String[] args)

- public: accessible by JVM
- static: no need for object
- void: returns nothing
- String[] args: command-line arguments

Access Modifiers in Methods:

Modifier	Scope
public	Everywhere
private	Within class
protected	Package + subclass
(default)	Only within package

Anonymous Methods – Lambda (Java 8):

Runnable r = () -> System.out.println("Run");

- Used to implement functional interfaces (1 abstract method).
- Cleaner alternative to writing classes.

Method References (Java 8):

Shorter syntax using :: operator.

List<String> names = List.of("Ram", "Shyam");

names.forEach(System.out::println);

Best Practices:

- Use meaningful method names (calculateTax(), not do1()).
- Keep methods short and focused (Single Responsibility Principle).
- Avoid too many parameters.
- Document using JavaDoc.
- Use method overloading wisely for flexibility.