Methods in Java

Function

Method

→ Work

- A **method** is a block of code that performs a specific task
- It runs **only when it is called**

Repeat

Advantages of Methods

- Reduces code duplication
- Improves program structure
- Makes debugging easier
- Enhances modular programming

{ int Sum()

Sum = a+b

}

}

Syntax of a Method

int

```
returnType methodName(parameters) {
    // method body
}
```

optional/require

- returnType – data type of returned value (void if none)
- methodName – name of the method
- parameters – inputs to the method
- method body – logic to be executed

int ← a + b

Sort(i

Example of a Method

```
public class cl_Method {
  public static void sum(int a,int b) {      // Definition
              int c=a+b;
              System.out.println(c);
       }

       public static void Hello(String name) {      // Definition

              System.out.println("Hello "+name);
       }

       public static void sum(int a) {      // Definition
              int mul=a*2;
              System.out.println(mul);
       }

}
```

Calling a Method

```java
public class cl_Method {

	public static void main(String[] args) {
	 int c=10;
	 int d=20;
	 cl_Method s=new cl_Method();
	 sum(c, d);
	 sum(c);                  //Calling
	 System.out.println(c);
	 System.out.print(ans);
	 String name="Shivam";
	 Hello(name);
	 Hello(name);
	 Hello(name);
	 System.out.println("Bye");
	 Hello(name);

	}
}
```

Calling a Method

- Used to pass values to methods

Types:

- No parameters
- Single parameter
- Multiple parameters

Hello World

name

sum = a+b

Method Overloading

- Same method name with different parameters
- Compile-time polymorphism

or

Static Binding

runtime

Oops

Dynamic

What is Shadowing

- **Shadowing** occurs when a variable declared in an inner scope has the **same name** as a variable in an outer scope
- The inner variable **hides (shadows)** the outer variable
- The outer variable becomes inaccessible within that scope

int a = 10; $\rightarrow$ 10

int a = 20; $\rightarrow$ 20