SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade

Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

COURSE NAME : 23CSB101- OBJECT ORIENTED PROGRAMMING

I YEAR /II SEMESTER

Unit II – INHERITANCE, PACKAGES AND INTERFACES

Topic : METHOD OVERLOADING - OBJECTS AS PARAMETERS – RETURNING OBJECTS

METHOD OVERLOADING

- Method overloading allows multiple methods in the same class to have the same name but different parameter lists (number, type, or order of parameters).

METHOD OVERLOADING

- Same Method Name: All overloaded methods must have the same name.
- Different Parameters: Overloaded methods must have different types, numbers, or orders of parameters.
- Return Type Doesn't Matter: Overloading is based on the method signature (method name + parameter list), so return type alone cannot differentiate overloaded methods.
- Access Modifiers Can Vary: Overloaded methods can have different access levels (e.g., public, private, protected).Exceptions Can Be Different: Overloaded methods can throw different exceptions.

METHOD OVERLOADING

- **Advantages**:
- **Improves Readability**: Using the same method name for similar operations makes the code more intuitive.
- **Code Reusability**: Eliminates the need to create multiple method names for the same functionality.
- **Compile-Time Polymorphism**: Allows a method to behave differently based on input parameters

METHOD OVERLOADING

```
class MathOperations {
    // Method with two int parameters
    int add(int a, int b) {
        return a + b;
    }
    // Method with three int parameters
    int add(int a, int b, int c) {
        return a + b + c;
    }
```

METHOD OVERLOADING

```
// Method with two double parameters
    double add(double a, double b) {
        return a + b;    }
}
public class OverloadingExample {
    public static void main(String[] args) {
        MathOperations obj = new MathOperations();
        System.out.println(obj.add(5, 10));        // Calls add(int, int)
        System.out.println(obj.add(5, 10, 15));    // Calls add(int, int, int)
        System.out.println(obj.add(5.5, 2.2));     // Calls add(double, double)
    }
}
```

Difference between constructor overloading and method overloading

Feature	Method Overloading	Constructor Overloading
Definition	Defining multiple methods with the same name but different parameters in the same class.	Defining multiple constructors with different parameter lists in the same class.
Purpose	Used to perform different operations with the same method name.	Used to initialize objects in different ways.
Return Type	Can have different return types (but return type alone cannot differentiate methods).	No return type (constructors do not have a return type).
Calling Mechanism	Called explicitly using an object.	Called implicitly when an object is created.
Usage	Used to improve readability and code reusability.	Used to provide multiple ways to initialize an object.
Example	java class MathOps { int add(int a, int b) { return a + b; } double add(double a, double b) { return a + b; } }	java class Person { String name; int age; Person(String name) { this.name = name; } Person(String name, int age) { this.name = name; this.age = age; } }

Objects as Parameters

In Java, objects can be passed as parameters to methods and can also be returned from methods. This allows for object manipulation within methods and promotes reusability.

Passing Objects as Parameters

Since Java passes objects by reference, **modifications** to the object inside the method **affect** the **original** object.

Returning Objects from Methods

A method can return an object, which **allows object creation** within a method and returning it for further use.

Example:

Swapping Two Numbers Using Objects demonstrates:

- **Passing an object as a parameter** to a method.
  - The swap(Number obj) method takes an object **obj** as a parameter.
  - It **accesses** the **values** of a and b inside the object.
- **Returning an object** from a method.
  - The swap method **creates a new object** with swapped values.
  - This new object is **returned** and assigned to num2.

```
// Class to hold two numbers
class Number {
    int a, b;
 // Constructor
    Number(int x, int y) {
        this.a = x;
        this.b = y;
    }
```

Cont…

```
    // Method that takes an object as a parameter and returns a new object
    static Number swap(Number obj) {
        // Creating a new object with swapped values
        return new Number(obj.b, obj.a);
    }
    // Display method
    void display() {
        System.out.println("a = " + a + ", b = " + b);    }
}
```

Cont…

```
public class Main {
    public static void main(String[] args) {
 // Creating an object
        Number num1 = new Number(5, 10);
        System.out.println("Before Swapping:");
        num1.display();
 // Calling swap method which takes object as parameter and returns object
        Number num2 = Number.swap(num1);
       System.out.println("After Swapping:");
        num2.display();
    }
}
```

Output:

Before Swapping:

a = 5, b = 10

After Swapping:

a = 10, b = 5


Thank
you