Sukkur Institute of Business Administration University
Department of Computer Science

Object Oriented Programming

BS – II (CS/SE/AI)
Spring 2025

Lab # 06: To become familiar with Methods, Overloading, Object Passing, Argument Passing, and Instance Variable Hiding

Instructor: Moona Solangi

Lab Report Rubrics (Add the points in each column, then add across the bottom row to find the total score)					**Total Marks**
S.No	**Criterion**	**0.5**	**0.25**	**0.125**	
1	Accuracy	☐ Desired output	☐ Minor mistakes	☐ Critical mistakes	
2	Timing	☐ Submitted within the given time	☐ 1 day late	☐ More than 1 day late	

Submission Profile

Name:
Enrollment ID:
Comments:

Submission date (dd/mm/yy):
Receiving authority name and signature:

_____

Instructor Signature

Note: Submit this lab hand-out before the next lab with attached solved activities and exercises

Objectives

After performing this lab, students will be able to understand,

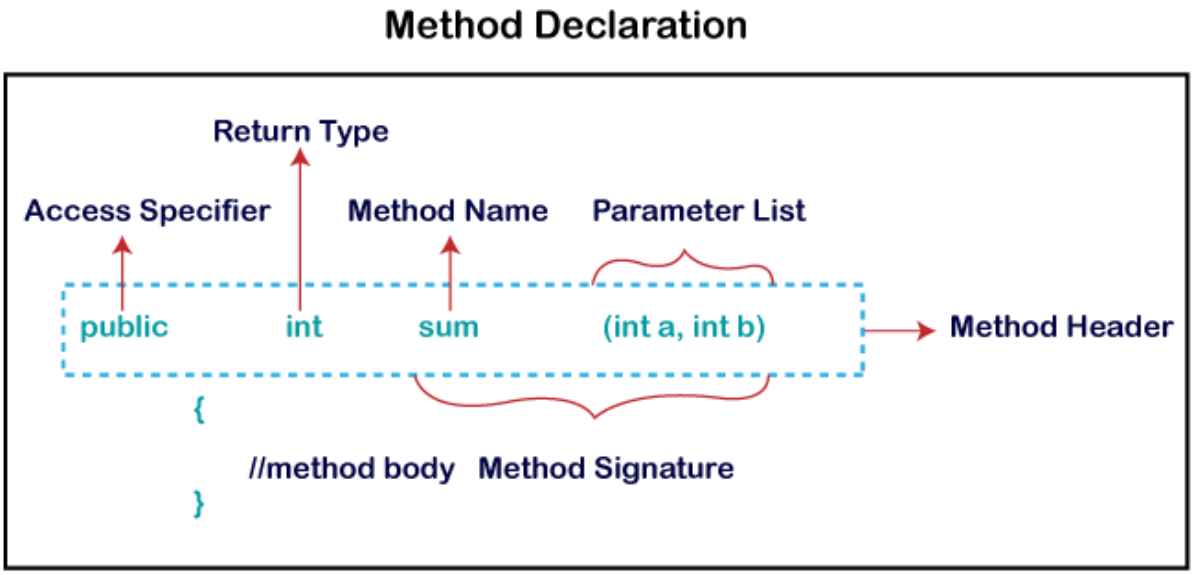
- ✅ Understand and implement **Methods in Java**.
- ✅ Implement **Method Overloading** and **Automatic Conversion in Overloading**.
- ✅ Use **Overloaded Constructors** effectively.
- ✅ Pass **Objects as Parameters to Methods** and **Constructors**.
- ✅ Understand and use **Instance Variable Hiding** and how to resolve it.

Methods in Java

What is a Method?

A **method** in Java is a block of code that performs a specific task. Methods **reduce redundancy** and **increase code reusability**.

Syntax of a Method in Java:



Example: A Simple Method

```java
class Calculator {
    // Method to add two numbers
    int add(int a, int b) {
        return a + b;
    }
}

public class MethodExample {
    public static void main(String[] args) {
        Calculator obj = new Calculator();
        int sum = obj.add(5, 10);
        System.out.println("Sum: " + sum);
    }
}
```

Method Overloading

What is Method Overloading?

- **Method Overloading** allows multiple methods with the **same name** but **different parameters**.
- The **return type alone cannot differentiate overloaded methods**.

Example: Overloaded Methods

```java
class Display {
    // Method with one parameter
    void show(int a) {
        System.out.println("Integer: " + a);
    }

    // Method with two parameters
    void show(int a, double b) {
        System.out.println("Integer: " + a + ", Double: " + b);
    }
}

public class OverloadingExample {
    public static void main(String[] args) {
        Display obj = new Display();
        obj.show(5);
        obj.show(10, 12.5);
    }
}
```

Method Overloading with Automatic Type Conversion

- Java **automatically promotes smaller data types** to larger types when overloading.

Example:

```
class AutoConversion {
    void show(int a) {
        System.out.println("Integer: " + a);
    }

    void show(double a) {
        System.out.println("Double: " + a);
    }
}

public class AutoConversionExample {
    public static void main(String[] args) {
        AutoConversion obj = new AutoConversion();
        obj.show(10); // int is automatically converted to double if needed
        obj.show(10.5);
    }
}
```

Overloading Constructors

What is Constructor Overloading?

- **Multiple constructors** in the same class with **different parameters.**
- Helps to create objects with **different initial values**.

Example:

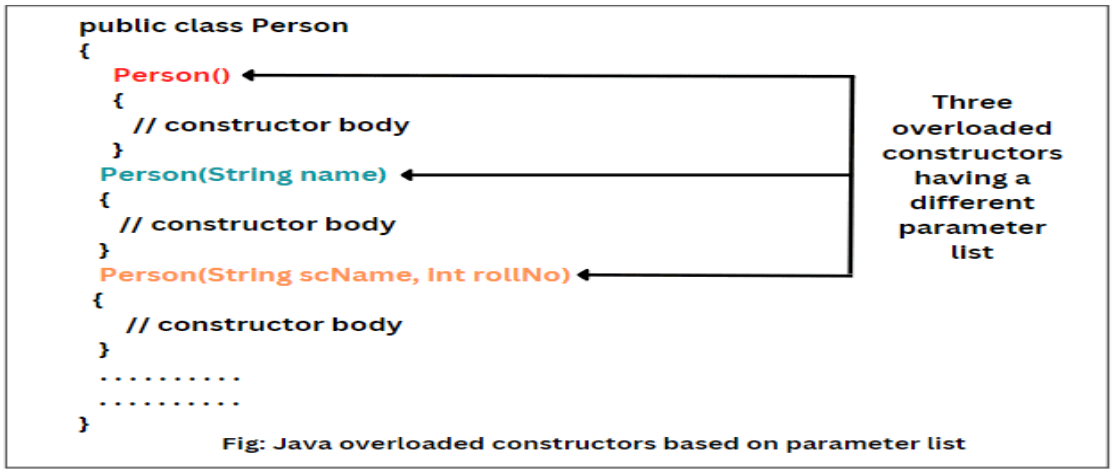


Fig: Java overloaded constructors based on parameter list

Passing Objects as Parameters to Methods and Constructors in Java

Introduction:

In Java, objects can be passed **as parameters** to **methods** and **constructors**. This allows methods and constructors to **operate on complex data**, modify objects, and establish relationships between objects.

Why Pass Objects?

- To **modify object attributes** inside methods.
- To **compare objects**.
- To **pass one object's data to another object** using constructors.
- To **increase code reusability**.

Passing Objects as Parameters to Methods

In Java, when an object is passed to a method:

- The **reference** to the object is passed (not the object itself).
- The **original object** can be modified inside the method.
- Changes made inside the method reflect **outside the method**.

Example 1: Modifying an Object Inside a Method

```
class Person {
   String name;
   int age;

   Person(String n, int a) {
      name = n;
      age = a;
   }

   void updateAge(int newAge) {
      age = newAge;  // Modifies the original object
   }
}
```

```
public class ObjectParameterExample {
    static void modifyPerson(Person p) {
        p.age += 5;  // Changes the age
        System.out.println("Inside method - Age: " + p.age);
    }

    public static void main(String[] args) {
        Person person1 = new Person("John", 25);
        System.out.println("Before method call - Age: " + person1.age);

        modifyPerson(person1);  // Passing object

        System.out.println("After method call - Age: " + person1.age);
    }
}
```

Output:

```
C:\Users\Moona\Desktop\OOP\Solutions\Lab6_Solutions>java ObjectParameterExample
Before method call - Age: 25
Inside method - Age: 30
After method call - Age: 30

C:\Users\Moona\Desktop\OOP\Solutions\Lab6_Solutions>
```

Key Explanation:

- person1 is passed **by reference**.
- The method **modifies** the age field of person1, and this change reflects outside the method.

Example 2: Comparing Two Objects Using a Method

```
class Box {

    double width, height, depth;

    Box(double w, double h, double d) {

        width = w;

        height = h;

        depth = d;
```

```
    }

    // Method to compare volumes of two objects
    boolean isLarger(Box b) {
        return (this.width * this.height * this.depth) > (b.width * b.height * b.depth);
    }
}

public class CompareObjects {
    public static void main(String[] args) {
        Box box1 = new Box(3, 4, 5);
        Box box2 = new Box(2, 3, 6);


        if (box1.isLarger(box2)) {
            System.out.println("Box 1 is larger than Box 2");
        } else {
            System.out.println("Box 2 is larger than Box 1");
        }
    }
}
```

Output:

```
C:\Users\Moona\Desktop\OOP\Solutions\Lab6_Solutions>java CompareObjects
Box 1 is larger than Box 2

C:\Users\Moona\Desktop\OOP\Solutions\Lab6_Solutions>
```

Key Explanation:

- The method isLarger(Box b) **accepts an object** and compares its properties.
- The this keyword refers to the calling object (box1).

Passing Objects to Constructors

- Sometimes, an object needs to be initialized using another object. In this case, an object can be **passed as a parameter to a constructor**.

Example 1: Copy Constructor (Passing Object to Constructor)

```
class Student {
    String name;
    int age;
    // Constructor with parameters
    Student(String n, int a) {
        name = n;
        age = a;
    }
    // Constructor that accepts an object (Copy Constructor)
    Student(Student s) {
        name = s.name;
        age = s.age;
    }
    void display() {
        System.out.println("Name: " + name + ", Age: " + age);
    }
}
public class ObjectToConstructor {
    public static void main(String[] args) {
        Student s1 = new Student("Alice", 20);
        Student s2 = new Student(s1);  // Passing object to constructor (copy)
```

```
        s1.display();

        s2.display();

    }

}
```

Output:

```
C:\Users\Moona\Desktop\OOP\Solutions\Lab6_Solutions>java ObjectToConstructor
Name: Alice, Age: 20
Name: Alice, Age: 20
```

Key Explanation:

- s1 is passed **to the constructor of s2**.
- s2 is initialized using s1's values.

Example 2: Aggregation (One Object Inside Another)

```
class Engine {
    String type;

    Engine(String type) {
        this.type = type;
    }
}

class Car {
    String model;
    Engine engine;  // Object as an instance variable

    Car(String model, Engine engine) {
        this.model = model;
        this.engine = engine;
    }

    void display() {
        System.out.println("Car Model: " + model + ", Engine Type: " + engine.type);
    }
}
```

```
public class AggregationExample {
    public static void main(String[] args) {
        Engine e1 = new Engine("V8");
        Car car1 = new Car("Ford Mustang", e1);

        car1.display();
    }
}
```

Output:

```
C:\Users\Moona\Desktop\OOP\Solutions\Lab6_Solutions>java AggregationExample
Car Model: Ford Mustang, Engine Type: V8
```

Key Explanation:

- The Engine object is passed **to the Car constructor**.
- The Car object now contains an **Engine object inside it** (Aggregation).

Key Observations

Concept	Explanation
Passing Objects to Methods	Allows methods to access and modify object properties.
Passing Objects to Constructors	Helps initialize objects using other objects.
Objects are Passed by Reference	Methods modify the original object (not a copy).
Aggregation (Object Inside Another Object)	A class can contain objects of another class as attributes.
Copy Constructor	Helps create a copy of an existing object.

Advantages of Passing Objects

✅ **Encapsulation** – Objects store related data together.
✅ **Reusability** – Same methods work on different objects.
✅ **Efficiency** – Instead of passing multiple variables, we pass a single object.
✅ **Flexibility** – Enables complex relationships (e.g., Car contains an Engine object).

Exercises

Question 1: Write a Java program that defines a **class TemperatureConverter**. This class should contain:

- **Overloaded methods convert()** that:
  - Converts **Celsius to Fahrenheit**.
  - Converts **Fahrenheit to Celsius**.

The main method should:

- Take temperature input from the user.
- Call the appropriate **overloaded method**.
- Display the converted temperature.

Expected Output:

```
Enter temperature: 100
Convert to (C/F): C
Converted to Celsius: 37.0°C

C:\Users\Moona\Desktop\OOP\Solutions\Lab6_Solutions>java TemperatureConverterApp
Enter temperature: 37
Convert to (C/F): F
Converted to Fahrenheit: 98.6°F
```

Question 2: Write a Java program that defines a **class Product** with:

- Attributes: name, price
- Method **comparePrice(Product p)** that compares the price of two products.

In the main method:

- Create **two product objects**.
- Compare them using comparePrice().
- Display which product is more expensive.

Expected Output:

```
C:\Users\Moona\Desktop\OOP\Solutions\Lab6_Solutions>java ProductComparison
Laptop is more expensive than Smartphone
```

Question 3: Write a Java program that defines a **class Employee** with:

- Attribute: salary
- Method **increaseSalary(int amount)** that tries to **increase salary** but **does not modify the original salary** (demonstrating Call by Value).

In the main method:

- Create an Employee object with a salary.
- Try to increase salary using the method.
- Print the salary **before and after the method call**.

Expected Output:

```
C:\Users\Moona\Desktop\OOP\Solutions\Lab6_Solutions>java CallByValueExample
Before method call - Salary: 5000.0
Inside Method - Salary after increment: 5500.0
After method call - Salary: 5500.0
```

Question 4: Write a Java program that declares a **class Example** where:

- A **method parameter hides an instance variable**.
- Use the **this keyword** to differentiate between the local and instance variables.

In the main method:

- Call the method and observe how **this keyword** solves the issue.

Question 5: Write a Java program that defines a **class Student** with:

- Attributes: name, grade
- Method **compareGrade(Student s)** to compare the grades of two students.

In the main method:

- Create **two Student objects**.
- Compare grades and display which student has a higher grade.

Question 6: Write a Java program that defines a **class Book** with:

- Attributes: title, price
- A **copy constructor** that initializes a new book using another book.

In the main method:

- Create **a book object** and **copy it** using the constructor.
- Display the details of both books.

Expected Output:

```
C:\Users\Moona\Desktop\OOP\Solutions\Lab6_Solutions>java BookCopyExample
Book: Clean Code, Price: $30.0
Book: Clean Code, Price: $30.0
```

Question 7: Write a Java program that defines a class **ShapeCalculator** with an **overloaded method calculateArea()** that calculates the area of:

- A **circle** (using radius).
- A **rectangle** (using length and width).
- A **triangle** (using base, height, and Boolean value).

The method should support **automatic type conversion** where necessary.
In the main method, call all versions of calculateArea() and display the results.

Expected Output:

```
C:\Users\Moona\Desktop\OOP\Solutions\Lab6_Solutions>java ShapeAreaCalculator
Circle Area: 95.03317777109123
Rectangle Area: 50
Triangle Area: 25.0
```

Question 8: Write a Java program that defines a class **Customer** with attributes: name, age, and accountBalance.

Create another class **LoanAccount** that:

- Uses a **constructor that accepts a Customer object**.
- Determines **loan eligibility** based on the customer's balance.

In the main() method:

- Create a **customer object**.

- Pass it to the **LoanAccount constructor**.
- Display **loan approval details**.

Expected Output:

```
C:\Users\Moona\Desktop\OOP\Solutions\Lab6_Solutions>java LoanAccountExample
Customer: Alice, Age: 30
Account Balance: $5000.0
Loan Approved: Yes
```

Bonus Question: Tricky Scenario-Based Question Covering All Objectives

🛒 **Scenario: E-Commerce Order Management System**

An **E-Commerce platform** needs an **Order Management System** where:

- **Customers** can **place orders** with different types of products.
- **Order processing logic** should handle **discounts & taxes** differently for **various categories of products**.
- The system should store **customer details, order details, and apply pricing rules efficiently**.

📝 **Task Requirements:**

1. **Create a class Customer** with:
   - Attributes: name, email, phoneNumber.
   - **Overloaded constructors**:
     - Default constructor sets "Guest" customer.
     - Parameterized constructor accepts name, email, and phone number.
2. **Create a class Product** with:
   - Attributes: productName, price, category.
   - A method **applyDiscount(double discountRate)** that applies a discount **based on category**.
3. **Create a class Order** that:
   - Has attributes: Customer object, Product object, quantity.
   - Uses a **constructor that accepts a Customer and Product object**.
   - Implements a method **calculateTotal(int quantity)** that:
     - Applies **different discount rates** using **overloaded methods** (for int and double discount values).
     - Demonstrates **automatic type conversion** if an int is passed instead of a double.
4. **Demonstrate Instance Variable Hiding** by:
   - Having an **order parameter named quantity** that hides the **instance variable**.
   - Use the **this keyword** to properly reference the instance variable.

```
C:\Users\Moona\Desktop\OOP\Solutions\Lab6_Solutions>java ECommerceSystem
Enter Customer Name (or press enter for Guest): Moona
Enter Email: moona@gmail.com
Enter Phone Number: 123456789
Enter Product Name: Laptop
Enter Product Price: 1000
Enter Product Category (Electronics, Clothing, Grocery): Electronics
Enter Order Quantity: 2

------ Order Summary ------
Customer: Moona, Email: moona@gmail.com
Product: Laptop, Category: Electronics
Original Price: $1111.111111111111
Discounted Price: $1000.0
Total Amount (after tax): $1980.0
--------------------------
```