SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya School of Engineering
(formerly K J Somaiya College of Engineering)

Somaiya TRUST

Java Programming –Core Concepts

Dr. Shila Jawale
Asst.Professor, KJSSE

2	**Class, Object, Method and Constructor**		08	CO 1, CO 2
	2.1	Class Object and Method: member, method, Modifier, Selector, iterator, State of an object. Memory allocation of object using new operator, Command line Arguments. instanceof operator in Java.		
	2.2	Method overloading & overriding, constructor, destructor in C++, Types of constructor (Default, Parameterized, copy constructor with object), Constructor overloading, this, final. super keyword, Garbage collection in Java.		

Outline

- Object as a parameter and Returning Object
- Call by value and Call by reference
- Recursion
- Static Fields/ Members
- Access Protection
- Nested Classes

Object as a parameter and Returning Object

Applicable for both C++ and Java

- General Syntax

// Inside a class

ClassName methodName (ClassName obj);

Example:

Complex add (Complex c);

```
    // Method that takes an object as parameter and returns object
    Complex add (Complex c) {
        Complex temp;
        temp.real = real + c.real;
        temp.imag = imag + c.imag;
        return temp; // Returning object
    }
};

int main() {
    Complex c1(2.5, 3.5);
    Complex c2(1.5, 4.5);
    Complex c3 = c1.add(c2); // Passing object and returning result
    cout << "First Complex Number: ";
    c1.display();
    cout << "Second Complex Number: ";
    c2.display();
    cout << "Sum: ";
    c3.display();
    return 0;
}
```

```java
class Complex {

    // Method that takes object as parameter and returns object
    public Complex add(Complex c) {
        float r = this.real + c.real;
        float i = this.imag + c.imag;
        return new Complex(r, i);  // Returning new object
    }
}
public class complexNo {
    Run | Debug
    public static void main(String[] args) {
        Complex c1 = new Complex(r:2.5f, i:3.5f);
        Complex c2 = new Complex(r:1.5f, i:4.5f);
        Complex c3 = c1.add(c2);  // Object as parameter and return type

        System.out.print(s:"First Complex Number: ");
        c1.display();

        System.out.print(s:"Second Complex Number: ");
        c2.display();

        System.out.print(s:"Sum: ");
        c3.display();
    }
}
```

```
PS E:\Academic year 25-26\OOPM_SE\JavaPgm> java complexNo
First Complex Number: 2.5 + 3.5i
Second Complex Number: 1.5 + 4.5i
Second Complex Number: 1.5 + 4.5i
Sum: 4.0 + 8.0i
```

Problem Definition: 1

Student Grade Evaluation

Define a class Student with:

Data members: name, marks1, marks2

Method compareMarks(Student s) to compare total marks and return the topper object

Goal: Pass object, compare internal data, return object.

Problem 2:

Cricket Player Stats

Define a Player class with:

- name, runs, matches
- Method betterAverage(Player p) to compare and return player with better average
- Method getAverage() returns average runs per match

Call by Value and Call by reference

-Supported by java and C++
-passing values to function as by values and by address
-Call by Value (Primitives) in both
-Java: Objects Passed (Reference by Value)

Call by Value

```
void modify(int x) {
    x = 10;
}

int main() {
    int a = 5;
    modify(a);
}
```

```
main():
+-----+        modify():
| a=5 |        +-----+
+-----+        | x=5 |  ← copy of 'a'
               | x=10| ← modified locally
               +-----+

Back in main(): a = 5 (unchanged)
```

Call by Reference

C++

```
void modify(int &x) {
    x = 10;
}

int main() {
    int a = 5;
    modify(a);
}
```

```
main():
+-----+
| a=5 |  ← same memory used by x
+-----+

modify():
x → a
→ x = 10 modifies a

After modify(): a = 10
```

Java

```
class Player {
    int runs;
    Player(int r) { runs = r; }
}

void modify(Player p) {
    p.runs = 100;
}

public static void main(String[] args) {
    Player p1 = new Player(50);
    modify(p1);
}
```

```
main():
+----------+       Heap:
| p1   ----/----> Player Object
+----------+        runs = 50

modify():
+----------+
| p    ----/----> same Player Object
+----------+        runs = 100
```

Recursion

- **Recursion** is when a function **calls itself** to solve a smaller sub-problem of the original task.
  Every recursive function must have:
- **Base Case** – when to stop the recursion.
- **Recursive Case** – when the function calls itself

```
#include <iostream>
using namespace std;

int factorial(int n) {
    if (n == 0) return 1;          // Base Case
    return n * factorial(n - 1);  // Recursive Case
}

int main() {
    cout << "Factorial of 5 = " << factorial(5);
    return 0;
}
```

```
factorial(3)
  ↳ 3 * factorial(2)
      ↳ 2 * factorial(1)
          ↳ 1 * factorial(0)
              ↳ returns 1 (base case)
          ↳ 1 * 1 = 1
      ↳ 2 * 1 = 2
  ↳ 3 * 2 = 6
```

Static Fields/ Members

Feature	Java	C++
Declaration	Inside class using `static`	Inside class with `static` keyword
Initialization	Inline or in static blocks	Outside class (once)
Shared by all objects	✅	✅
Access method	`ClassName.staticField`	`ClassName::staticField`
Memory allocation	Once per class (on load)	Once per class (at program start or first use)

What is a Static Field (or Static Member)?

- A **static field** is shared among **all instances** of a class.
  It **belongs to the class itself**, not to any specific object.

Static Fields in Java

- Declared using the static keyword.
- Shared across all objects.
- Can be accessed via the class name (e.g., ClassName.staticField).
- Memory is allocated once when the class is loaded.

```java
class Student {
    String name;
    static String college = "ABC College"; // static field

    Student(String name) {
        this.name = name;
    }

    void display() {
        System.out.println(name + " from " + college);
    }
}

public class Static_Demo {
    Run | Debug
    public static void main(String[] args) {
        Student s1 = new Student(name:"Alice");
        Student s2 = new Student(name:"Bob");

        s1.display(); // Alice from ABC College
        s2.display(); // Bob from ABC College

        Student.college = "XYZ University"; // change static field

        s1.display(); // Alice from XYZ University
        s2.display(); // Bob from XYZ University
    }
}
```

```
class Student {
    string name;
    static string college; // declaration of static field

public:
    Student(string n) { name = n; }

    void display() {
        cout << name << " from " << college << endl;
    }

    static void changeCollege(string newCollege) {
        college = newCollege; // static function accessing static field
    }
};
// Definition of static member
string Student::college = "ABC College";
int main() {
    Student s1("Alice");
    Student s2("Bob");

    s1.display(); // Alice from ABC College
    s2.display(); // Bob from ABC College

    Student::changeCollege("XYZ University");

    s1.display(); // Alice from XYZ University
    s2.display(); // Bob from XYZ University
}
```

Use Cases :

- Counting objects created from a class.
- Shared configuration or settings.
- Accessing class-wide data in static methods.

Java / C++ – Static Method

Aspect	
Definition	A method that belongs to the **class itself** and not to any instance (object).
Access	Can be called directly using the class name, **without creating an object**.
Memory Behavior	Loaded **once in memory** when the class is loaded.
Access to Members	Can access only **static data members** and **other static methods**. Cannot access non-static (instance) members directly.
Use Cases	Utility methods, helper functions (e.g., Math.pow(), Collections.sort())
Example in Java	Math.sqrt(25); — No object of Math needed
Example in C++	Utility::printMessage(); — No object of Utility needed

C++

```cpp
#include <iostream>
using namespace std;

class Calculator {
public:
    static int add(int a, int b) {
        return a + b;
    }
};

int main() {
    int result = Calculator::add(5, 3);
    cout << "Sum: " << result << endl;
    return 0;
}
```

Java

```java
class Calculator {
    public static int add(int a, int b) {
        return a + b;
    }
}

public class Main {
    public static void main(String[] args) {
        int result = Calculator.add(5, 3);  // No object creation
        System.out.println("Sum: " + result);
    }
}
```

```cpp
#include <iostream>
using namespace std;

class Calculator {
public:
    static int add(int a, int b) {
        return a + b;
    }
};

int main() {
    int result = Calculator::add(5, 3);
    cout << "Sum: " << result << endl;
    return 0;
}
```

```
class Calculator {
public static int add(int a, int b) {
return a + b;

}

}

public class Main {

public static void main(String[] args) {
int result = Calculator.add(5, 3); // No object
creation

System.out.println("Sum: " + result);

}
}
```

Characteristics of Static Nested Class

Feature	Description
Does not need object of outer class	Can be instantiated without creating outer class object
Can access only static members of outer class	Cannot access non-static members directly
Loaded only when referenced	Not loaded with outer class unless used
Defined inside another class	Always an inner (nested) class

```
class Computer {
    static class CPU {
        void print() {
            System.out.println("CPU initialized");
        }
    }
}
public class Main {
    public static void main(String[] args) {
        Computer.CPU cpu = new Computer.CPU(); // No Computer object needed
        cpu.print();
    }
}
```

Problem Definition: 1 **Utility Class – Area Calculations**

Concept: Static method without object

Task:

Create a class Geometry with:

- static double areaOfSquare(double side)
- static double areaOfCircle(double radius)

Call these methods from main() without creating any object.

Problem 2: **Bank Account with Shared Interest Rate**

Concept: Static field and method with instance data

Task:

Create a class BankAccount:

- Static variable interestRate
- Instance variable balance
- Static method setInterestRate(double)
- Method calculateInterest() that returns interest earned

Access Protection

https://codesandbox.io/p/sandbox/routing-forked-hryspj

1. public

- Members declared as public are accessible **from anywhere** in the program.
- Often used for functions that need to be called by other parts of the program.

2. private

- Members declared as private are accessible **only inside the same class**.
- This is the **default** access specifier for class members if you don't specify one.
- Used to hide internal details (encapsulation).

3 protected

- Members declared as protected are accessible:
- Inside the same class
- Inside derived (child) classes
- **Not** accessible from outside the class (like private), except via inheritance

Access Specifier	Same Class	Derived Class	Outside Class
public	✅ Yes	✅ Yes	✅ Yes
protected	✅ Yes	✅ Yes	❌ No
private	✅ Yes	❌ No	❌ No

1.public

•Accessible **from anywhere** (any class, any package).

•Used when you want full accessibility.

2. protected

Accessible:

•In the **same package**

•In **subclasses** (even if they're in different packages).

3. default *(no keyword — package-private)*

•If you don't specify any access modifier, it's **package-private**.

•Accessible **only** inside the same package.

•**Not** accessible from outside the package (unless inherited with broader access).

4. private

•Accessible **only within the same class**.

•Not visible to subclasses or other classes in the same package.

Access Modifier	Same Class	Same Package	Subclass (diff. pkg)	Outside Package
public	✅ Yes	✅ Yes	✅ Yes	✅ Yes
protected	✅ Yes	✅ Yes	✅ Yes	❌ No
default	✅ Yes	✅ Yes	❌ No	❌ No
private	✅ Yes	❌ No	❌ No	❌ No

Problem :

- Write a BankAccount class:
- private data: balance (double)
- public method: deposit(double) and showBalance()
- Try to modify balance directly from main() — what happens?

Problem 2:

Create a base class Person with:

- protected member: name
- public method: setName()

Create a derived class Employee that sets and prints name.

- Try accessing name directly in main() — should fail.