# SNS COLLEGE OF ENGINEERING

**Coimbatore-107**
**An Autonomous Institution**

**COURSE NAME : 23CSB201 & Object Oriented Programming**

**II YEAR/ III SEMESTER**

**UNIT – II INHERITANCE, PACKAGES, INTERFACE**

***Topic: Returning Object***

Dr.P.Poonkodi

Assistant Professor(SG)

Department of Computer Science and Technology

# Introduction

Methods can return objects just like they return primitive data types. This is useful when we want to:

- Create and return new objects from a method
- Modify an object’s data and return the updated object
- Chain multiple operations by returning objects

## Scenario

A university needs a system where:

- A method calculates the average marks of a student and returns a new **StudentResult** object.
- Another method compares two student results and returns the student with the higher score.
- This helps in processing student performance dynamically.

# Example

```
// Student class to store basic student details
class Student
{
    int studentId;
    String name;
    double marks1, marks2, marks3;
```

## Example

```
// Constructor
 Student(int studentId, String name, double marks1, double marks2, double
marks3)
{
        this.studentId = studentId;
        this.name = name;
        this.marks1 = marks1;
        this.marks2 = marks2;
        this.marks3 = marks3;
    }
```

## Example

```
// Method to calculate average marks and return a new StudentResult
object
    StudentResult calculateAverage()
  {
        double average = (marks1 + marks2 + marks3) / 3;
        return new StudentResult(this, average);
    }
}
// Separate class to store student results
class StudentResult
{
    Student student;
    double averageMarks;
```

# Example

```
// Separate class to store student results
class StudentResult
 {
   Student student;
   double averageMarks;
   // Constructor
   StudentResult(Student student, double averageMarks)
   {
       this.student = student;
       this.averageMarks = averageMarks;
   }
```

## Example

```
// Method to display student result
    void displayResult()
    {
        System.out.println("Student: " + student.name + " (ID: " +
student.studentId + "), Average Marks: " + averageMarks);
    }
// Method to compare results and return the student with higher marks
    static StudentResult compareResults(StudentResult s1, StudentResult s2)
{
        return (s1.averageMarks > s2.averageMarks) ? s1 : s2;
    }
}
```

# Example

```
public class StudentPerformanceSystem
{
    public static void main(String[] args)
    {
        // Creating Student objects
        Student student1 = new Student(101, "Alice", 85, 90, 80);
        Student student2 = new Student(102, "Bob", 78, 88, 92);
        // Calculating and getting StudentResult objects
        StudentResult result1 = student1.calculateAverage();
        StudentResult result2 = student2.calculateAverage();
```

# Example

```
// Displaying student results
        result1.displayResult();
        result2.displayResult();


        // Comparing results and displaying the top performer
        StudentResult topStudent = StudentResult.compareResults(result1, 
result2);
        System.out.println("Top Performer: ");
        topStudent.displayResult();
    }
}
```

## References

- Java : the complete Reference ( Eleventh Edition), Herbert Schildt, 2018.

Thank You!

# SNS COLLEGE OF ENGINEERING

**Coimbatore-107**
**An Autonomous Institution**

**COURSE NAME : 23CSB201 & Object Oriented Programming**

**II YEAR/ III SEMESTER**

**UNIT – II INHERITANCE, PACKAGES, INTERFACE**

***Topic: Static***

Dr.P.Poonkodi

Assistant Professor(SG)

Department of Computer Science and Technology

# Introduction

- **static** keyword in Java is used to define class-level variables and methods that belong to the class rather than to instances (objects) of the class
- This concept is important for optimizing memory usage and structuring code efficiently
- Static methods can be called without creating an object

# Example

- A method to get the school name

```
class StudentHelper
{
    static String getSchoolName()
   {
        return "ABC High School";
    }
}
public class Test
{
    public static void main(String[] args)
    {
        // No need to create an object to call the static method
        System.out.println(StudentHelper.getSchoolName());
    }
}
```

| Concept | Key Points |
|---|---|
| static Variables | Shared across all instances |
| static Methods | Called using the class name, no object required |
| static Block | Runs once when the class loads |

# References

- Java : the complete Reference ( Eleventh Edition), Herbert Schildt, 2018.

Thank You!

# SNS COLLEGE OF ENGINEERING

**Coimbatore-107**
**An Autonomous Institution**

**COURSE NAME : 23CSB201 & Object Oriented Programming**

**II YEAR/ III SEMESTER**

**UNIT – II INHERITANCE, PACKAGES, INTERFACE**

***Topic: Nested Class & Inner Class***

Dr.P.Poonkodi
Assistant Professor(SG)
Department of Computer Science and Technology

# Introduction

- Java allows defining a class inside another class, known as an **inner class** or **nested class**
- This approach enhances code organization, encapsulation, and logical grouping of related components
- **Nested Classes in Java**
  - **Static Nested Class**
    - Declared using the **static** keyword
    - Can access only static members of the outer class
    - Does not need an instance of the outer class to be instantiated
  - **Non-Static Inner Class**
    - Needs an instance of the outer class
    - Can access both static and non-static members of the outer class

# Introduction

- **Types of Inner Classes**
  - **Local Inner Class**
    - Defined inside a method
    - Can only be accessed within the method where it is declared
  - **Anonymous Inner Class**
    - Declared and instantiated in a single expression
    - Used to override methods of a class or an interface

# Static Nested Class

- Example

```
class OuterClass
{
        static class StaticNested
        {
                void display()
                {
                        System.out.println("Static Nested Class");
                }
        }
}
public class Main
{
        public static void main(String[] args)
        {
                OuterClass.StaticNested nested = new OuterClass.StaticNested();
        nested.display();
        }
}
```

# Non-Static Inner Class

```
class OuterClass
{
    class InnerClass
    {
        void display()
        {
                System.out.println("Inner Class");
        }
    }
}
public class Main
{
        public static void main(String[] args)
        {
                OuterClass outer = new OuterClass();
                OuterClass.InnerClass inner = outer.new InnerClass();
                inner.display();
        }
}
```

# Local Inner Class

```
class Outer
{
        void outerMethod()
        {
                class LocalInner
                {
                        void display()
                        {
                                System.out.println("Local Inner Class");
                        }
                }
                LocalInner local = new LocalInner();
                local.display();
        }
}
```

# Local Inner Class

```
public class Main
{
        public static void main(String[] args)
        {
                Outer outer = new Outer();
                outer.outerMethod();
        }
}
```

# Anonymous Inner Class

```
abstract class Animal
{
        abstract void makeSound();
}
public class Main
{
        public static void main(String[] args)
        {
                Animal cat = new Animal()
                {
                        void makeSound()
                        {
                                System.out.println("Meow");
                        }
                };
                cat.makeSound();
        }
}
```

# References

- Java : the complete Reference ( Eleventh Edition), Herbert Schildt, 2018.

Thank You!