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 I – INTRODUCTION TO OOP AND JAVA

Topic : ACCESS SPECIFIERS - STATIC MEMBERS –

JAVA DOC COMMENTS

Access specifiers - Static members

Determine the visibility and behavior of class members.

- Access Specifiers (or access modifiers) : C**ontrol the visibility** of classes, methods, and variables. Java has four main access specifiers:

Access specifiers

Access Specifier	Scope
Public	Accessible **from anywhere** (inside and outside the package).
Private	Accessible **only within the same class**.
protected	Accessible within the **same package** and by **subclasses (even outside the package)**.
(default) (No modifier)	Accessible **only within the same package**.

Access specifiers

```
class Example {
    public int publicVar = 10;      // Accessible anywhere
    private int privateVar = 20;   // Accessible only within this class
    protected int protectedVar = 30; // Accessible in the same package and subclasses
    int defaultVar = 40;            // Accessible within the same package
public void display() {
        System.out.println("Public Variable: " + publicVar);
        System.out.println("Private Variable: " + privateVar);
        System.out.println("Protected Variable: " + protectedVar);
        System.out.println("Default Variable: " + defaultVar);
    }
}
```

Access specifiers

```
public class AccessSpecifierDemo {
    public static void main(String[] args) {
        Example obj = new Example();
        System.out.println(obj.publicVar);  // ✅ Allowed
        // System.out.println(obj.privateVar); // ❌ Not allowed (private)
        System.out.println(obj.protectedVar); // ✅ Allowed (same package)
        System.out.println(obj.defaultVar);   // ✅ Allowed (same package)
    }
}
```

Static members

- A static member **belongs to** the **class** rather than an instance of the class. (i.e) Static variables are shared among all instances of the class.
- **Static methods can be called without creating an object.**
- The static **keyword** can be **used** with **variables, methods, blocks, and nested classes.**

Static members

- **Key Points about Static Members**

✓ Static Variables Stored in the class memory (not instance memory).Shared among all instances of the class.

✓ Static Methods Can access only static variables (cannot access instance variables directly). Can be called using the class name (ClassName.methodName()).

✓ Static **BlockRuns once** when the class is loaded.

Static members

```
class StaticExample {
    static int count = 0; // Static variable (shared among all objects)
    StaticExample() {
        count++;
        System.out.println("Object created. Count: " + count);
    }
    static void displayCount() { // Static method
        System.out.println("Total objects created: " + count);
    }
}
```

Static members

```
public class StaticDemo {
    public static void main(String[] args) {
        StaticExample obj1 = new StaticExample();
        StaticExample obj2 = new StaticExample();
        StaticExample obj3 = new StaticExample();

        StaticExample.displayCount(); // Calling static method without an object
    }
}
```

Static members

Output:

```
Object created. Count: 1
Object created. Count: 2
Object created. Count: 3
Total objects created: 3
```

Static members

```
class Test {
    static {
        System.out.println("Static block executed.");
    }
}
public class StaticBlockDemo {
    public static void main(String[] args) {
        Test t1 = new Test();  // Static block runs only once
        Test t2 = new Test();  // Won't run again
    }
}
```

```
Static block executed.
```

Java Doc comments

JavaDoc comments are special comments in Java used to generate documentation for classes, methods, and fields. These comments begin with /** and end with */. JavaDoc is a tool that extracts these comments and produces HTML documentation.

```
/**
 * This is a JavaDoc comment.
 * It provides documentation for a class, method, or field.
 */
```

```
Static block executed.
```

Java Doc comments

JavaDoc Tags

Tag	Description
@author	Specifies the author of the code.
@version	Specifies the version of the class.
@param	Describes a method parameter.
@return	Describes the return value of a method.
@throws	Describes exceptions thrown by a method.
@deprecated	Marks a method or class as deprecated.
@see	Refers to another class or method for reference.

Java Doc comments

```
/**
 * The Calculator class provides basic arithmetic operations.
 * It demonstrates how to use JavaDoc comments.
 *
 * @author John Doe
 * @version 1.0
 */
public class Calculator {

    /**
     * Adds two numbers and returns the sum.
     *
     * @param a The first number.
     * @param b The second number.
     * @return The sum of a and b.
     */
    public int add(int a, int b) {
        return a + b;
    }
```

```
/**
     * Subtracts the second number from the first number.
     *
     * @param a The first number.
     * @param b The second number.
     * @return The result of a - b.
     */
    public int subtract(int a, int b) {
        return a - b;
    }

    /**
     * Multiplies two numbers.
     *
     * @param a The first number.
     * @param b The second number.
     * @return The product of a and b.
     */
    public int multiply(int a, int b) {
        return a * b;
    }
```

Java Doc comments

```
    /**
     * Divides the first number by the second number.
     *
     * @param a The numerator.
     * @param b The denominator (must not be zero).
     * @return The result of a / b.
     * @throws ArithmeticException If b is zero.
     */
    public double divide(int a, int b) throws ArithmeticException {
        if (b == 0) {
            throw new ArithmeticException("Cannot divide by zero");
        }
        return (double) a / b;
    }
}
```

Java Doc comments

Generating JavaDoc Documentation

To generate JavaDoc documentation, use the following command in the terminal

```
javadoc -d doc Calculator.java
```

-d doc specifies that the documentation should be stored in the doc directory.

Open doc/index.html in a browser to view the generated documentation.

Java Doc comments

Generating JavaDoc Documentation

Example Output :

After running javadoc, an HTML file will be created showing:

Class Description

Method Descriptions

Parameter Details

Return types

Exception

handling

`

Thank you