SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NAAC – UGC with 'A' Grade

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

23CSB101

OBJECT ORIENTED PROGRAMMING

JavaDoc Comments, Constants, Identifiers Garbage Collection

By

M.Kanchana

Assistant Professor/CSE

JavaDoc Comments

Javadoc is a tool which comes with JDK and it is used for generating Java code documentation in HTML format from Java source code. Java documentation can be created as part of the source code.

TYPES OF COMMENTS:

Class Comments

The class comment must be placed *after* any import statements, directly before the class definition.

```
import java.io.*;
/** class comments should be written here */
Public class sample
{
….
}
```

JavaDoc Comments

2.Method Comments

The method comments must be placed immediately before the method that it describes.

Tag	Description	Syntax
@param	It describes the method parameter	**@param name description**
@return	This tag describes the return value from a method with the exception void methods and constructors.	**@return description**
@throws	This tag describes the method that throws an exception.	**@throws class description**

JavaDoc Comments

3.Field Comments

Field comments are used to document public fields—generally that means static constants.

```
/**
* Account number
*/
public static final int acc_no = 101;
```

JavaDoc Comments

Tag	Meaning
@author	Identifies the author.
{@code}	Displays information as-is, without processing HTML styles, in code font.
@deprecated	Specifies that a program element is deprecated.
{@docRoot}	Specifies the path to the root directory of the current documentation.
@exception	Identifies an exception thrown by a method or constructor.
@hidden	Prevents an element from appearing in the documentation.
{@index}	Specifies a term for indexing.
{@inheritDoc}	Inherits a comment from the immediate superclass.
{@link}	Inserts an in-line link to another topic.
{@linkplain}	Inserts an in-line link to another topic, but the link is displayed in a plain-text font.
{@literal}	Displays information as-is, without processing HTML styles.
@param	Documents a parameter.

JavaDoc Comments

Tag	Meaning
@provides	Documents a service provided by a module.
@return	Documents a method's return value.
@see	Specifies a link to another topic.
@serial	Documents a default serializable field.
@serialData	Documents the data written by the **writeObject( )** or **writeExternal( )** methods.
@serialField	Documents an **ObjectStreamField** component.
@since	States the release when a specific change was introduced.
{@summary}	Documents a summary of an item. (Added by JDK 10.)
@throws	Same as **@exception**.
@uses	Documents a service needed by a module.
{@value}	Displays the value of a constant, which must be a **static** field.
@version	Specifies the version of a program element.

JavaDoc Comments

javadoc -d docDirectory nameOfPackage
for a single package. Or run

javadoc -d docDirectory nameOfPackage1 nameOfPackage2...
to document multiple packages.

If your files are in the default package, then instead run
javadoc -d docDirectory *.java
If you omit the -d docDirectory option, then the HTML files are extracted to the currentdirectory.

JavaDoc Comments

```
/**
 * This program performs the addition of two numbers.
 * It demonstrates the use of Javadoc comments for documentation.
 *
 * @author ICSE
 * @version 1.0
 */
public class Addition {

    /** Stores the first number for addition. */
    private int num1;

    /** Stores the second number for addition. */
    private int num2;
```

JavaDoc Comments

```
/**
 * Constructor to initialize the numbers.
 *
 * @param num1 The first number
 * @param num2 The second number
 */
public Addition(int num1, int num2) {
    this.num1 = num1;
    this.num2 = num2;
}
/**
 * Adds the two numbers and returns the sum.
 *
 * @return The sum of num1 and num2
 */
public int addNumbers() {
    return num1 + num2;
}
```

JavaDoc Comments

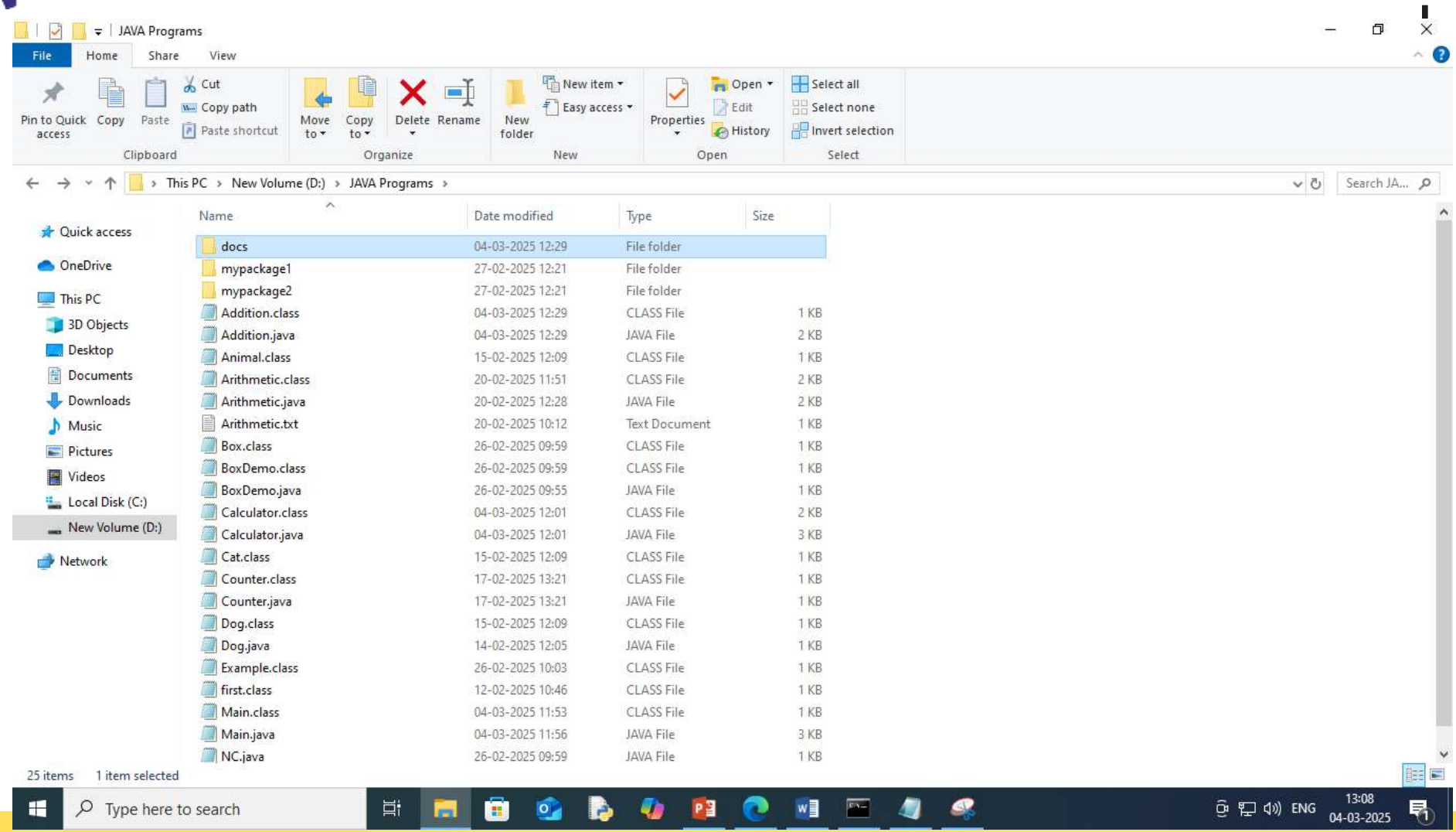
```
/**
     * Main method to test the addition operation.
     *
     * @param args Command-line arguments (not used)
     */
    public static void main(String[] args) {
        Addition addition = new Addition(10, 5); // Creating an object with
numbers 10 and 5
        int result = addition.addNumbers(); // Performing addition

        // Displaying the result
        System.out.println("The sum is: " + result);
    }
}
```

JavaDoc Comments

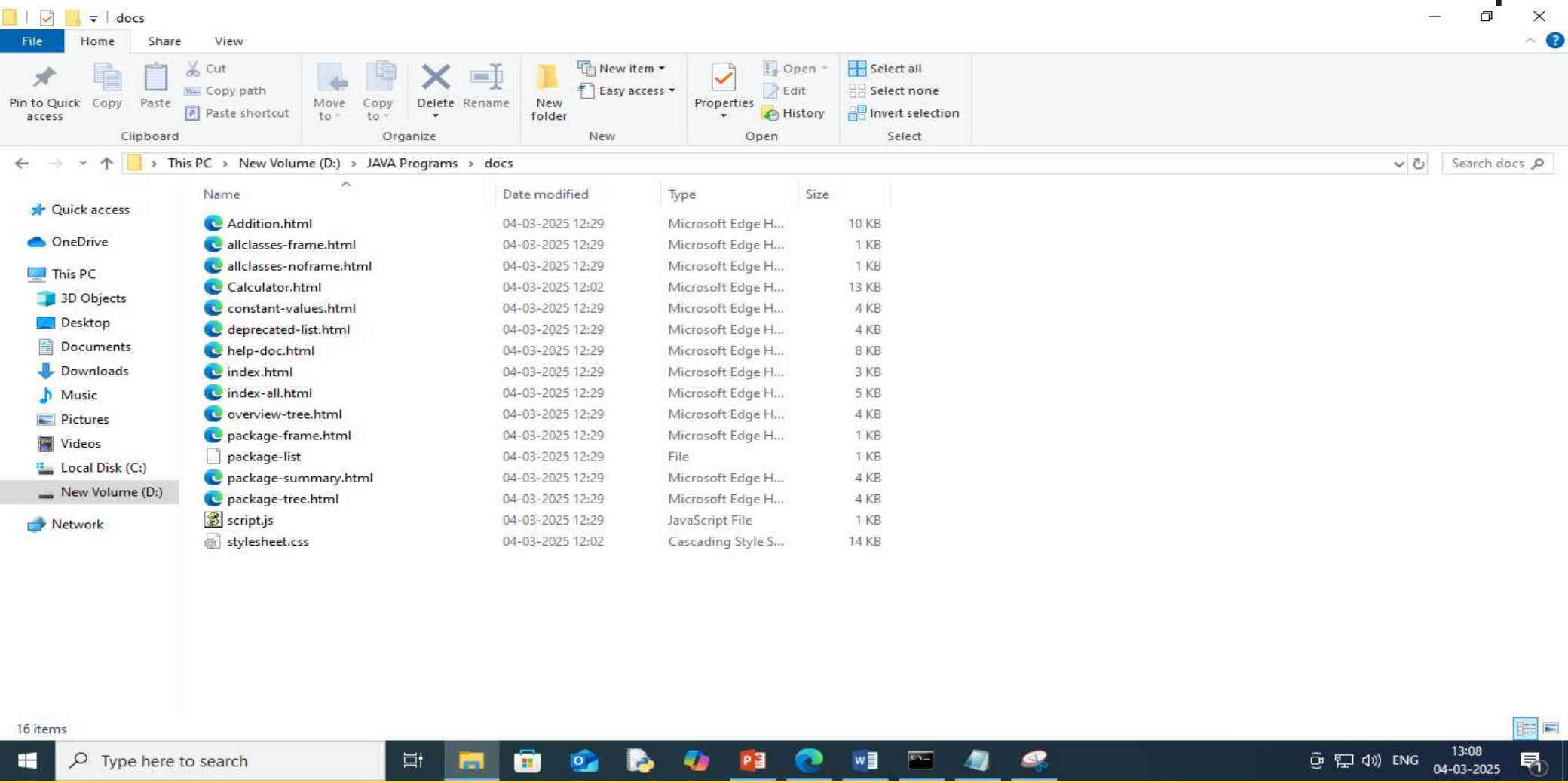
JavaDoc Comments

D:\JAVA Programs>**javadoc -d docs Addition.java**
Loading source file Addition.java...
Constructing Javadoc information...
Standard Doclet version 1.8.0_201
Building tree for all the packages and classes...
Generating docs\Addition.html...
Generating docs\package-frame.html...
Generating docs\package-summary.html...
Generating docs\package-tree.html...
Generating docs\constant-values.html...
Building index for all the packages and classes...
Generating docs\overview-tree.html...
Generating docs\index-all.html...
Generating docs\deprecated-list.html...
Building index for all classes...
Generating docs\allclasses-frame.html...
Generating docs\allclasses-noframe.html...
Generating docs\index.html...
Generating docs\help-doc.html...

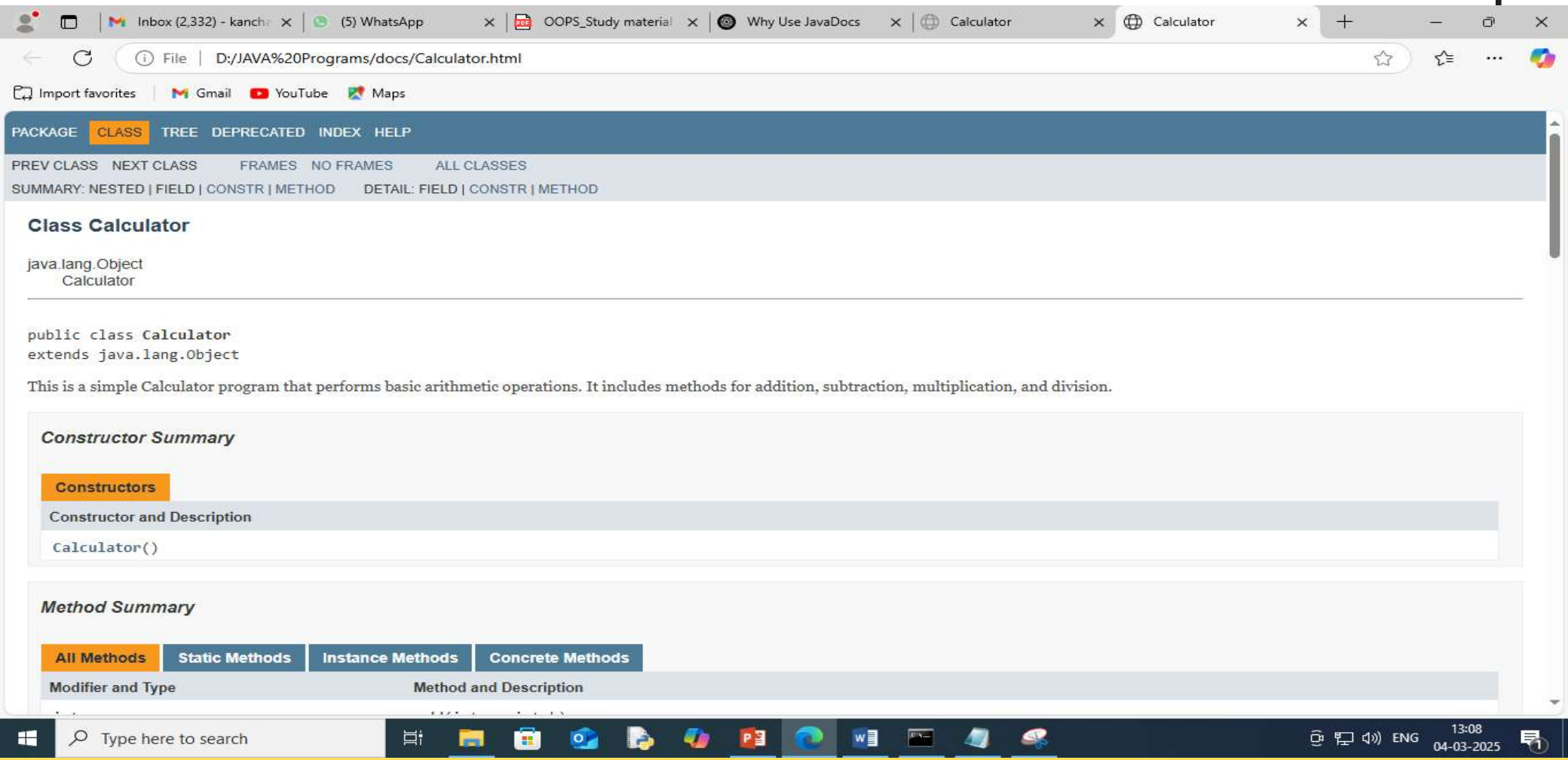
JavaDoc Comments



JavaDoc Comments



JavaDoc Comments



Java Comments

- Java comments are either explanations of the source code or descriptions of classes,interfaces, methods, and fields.
- Comments in Java do not show up in the executable program.

Line comment: When you want to make a one line comment type "//" and follow the two forward slashes with your comment.
Syntax: // text

Block Comment:
To start a block comment type "/*". Everything between the forward slash and asterisk, even if it's on a different line, will be treated as comment until the characters "*/" end the comment.

Syntax: /* *text* */

JAVA - CONSTANTS

- A constant is an identifier written in uppercase (convention and not a rule) that prevents
- its contents form being modified by the program during the execution.
- If an attempt is made to change the value, the compiler will give an error message.
- In Java, the keyword **final** is used to declare constants.
- The value of a final variable cannot change after it has been initialized.

final datatype variablename=value;

final float PI=3.14f;

JAVA - IDENTIFIERS

- Identifiers are names given to the variables, classes, methods, objects, labels, package and interface in our program.
- The name we are giving must be meaningful and it may have random length.

The following rule must be followed while giving a name:

1. The first character must not begin with a number.
2. The identifier is formed with alphabets, number, dollar sign ($) and underscore (_).
3. It should not be a reserved word.
4. Space is not allowed in between the identifier name.

JAVA – RESERVED WORDS (KEYWORDS)

abstract	**assert**	**boolean**	**break**	**byte**	**case**
catch	**char**	**class**	const*	continue	default
double	do	else	enum	extends	false
final	**finally**	**float**	**for**	goto*	**if**
implements	**import**	instanceof	**int**	**interface**	long
native	**new**	null	**package**	private	protected
public	return	short	**static**	strictfp	**super**
switch	synchronized	**this**	**throw**	**throws**	transient
true	**try**	**void**	volatile	while	

GARBAGE_COLLECTION

- Since objects are dynamically allocated by using the **new** operator, you might be wondering how such objects are destroyed and their memory released for later reallocation.
- Garbage collection (GC) in Java is a process that automatically removes unused objects from memory, helping to manage memory efficiently.
- The JVM's Garbage Collector (GC) looks for objects that no longer have references.
- Once an object isn't reachable, it is marked for deletion.
- The GC reclaims memory, preventing memory leaks.

GARBAGE_COLLECTION

The **finalize()** method allows an object to clean up resources before it is destroyed.(Think of it like packing up your things before leaving a house.)

```
class Example {
    // Constructor to create an object
    Example() {
        System.out.println("Object Created!");
    }

    // finalize() method
    @Override
    protected void finalize() {
        System.out.println("Object is being
deleted...");
    }
 public static void main(String[] args) {
        Example obj = new Example(); // Creating an object

        obj = null; // Making the object eligible for garbage
collection

        System.gc(); // Requesting garbage collection

        System.out.println("End of program.");
    }
}
```

GARBAGE_COLLECTION

Output:

Object Created!
End of program.
Object is being deleted...

THANK YOU