Date – 24/07/2025

✅ Topics Covered:

- Class
- Fields (Instance Variables)
- Methods
- Declaring Objects
- new Operator
- Assigning Object Reference Variables

1. What is a Class?

A **class** is a blueprint for creating objects.
It defines the structure (fields) and behavior (methods) of objects.

Syntax:

```
class ClassName {
    // fields
    // methods
}
```

Example:

```
class Student {
    int rollNo;         // field
    String name;        // field

    void display() {    // method
        System.out.println("Roll No: " + rollNo);
        System.out.println("Name: " + name);
    }
}
```

2. What are Fields?

Fields are also called **instance variables**.
They store the state/data of an object.

Fields are declared **inside the class but outside methods**.

```
int rollNo;

String name;
```

3. What are Methods?

Methods define the behavior of objects.
They contain code to perform actions using object data.

Syntax:

```
returnType methodName(parameters) {

    // method body

}
```

Example:

```
void display() {

    System.out.println("Hello");

}
```

4. Declaring Objects

To use a class, we create **objects**.

Syntax:

```
ClassName obj;
```

Example:

```
Student s1;   // declares an object reference of type Student
```

Note: This only **declares**, it doesn’t create the object yet.

5. new Operator

The new keyword is used to **create an actual object** in memory.

Syntax:

```
obj = new ClassName();
```

Or in one line:

```
Student s1 = new Student();
```

6. Assigning Object Reference Variables

You can assign one object reference to another:

```
Student s1 = new Student();

Student s2 = s1;
```

Now both s1 and s2 refer to the **same object** in memory.

Complete Example

```
class Student {
    int rollNo;
    String name;

    void display() {
        System.out.println("Roll No: " + rollNo);
        System.out.println("Name: " + name);
    }

    public static void main(String[] args) {
        Student s1 = new Student();   // object creation
        s1.rollNo = 101;
        s1.name = "Amit";

        Student s2 = s1;  // assigning reference

        s1.display();     // accessing via s1
        s2.display();     // accessing same object via s2
    }
}
```

Summary Table

Concept	Purpose
Class	Defines blueprint for objects
Fields (Instance Vars)	Store data
Methods	Define behavior
Declaring Object	Reserve reference (but no memory)
new Operator	Allocates memory
Assigning Reference	Makes another variable point to same object

Next….

📘 Topics:

- The main Class
- Command Line Arguments
- finalize() Method

1. The main Class (Actually: the main Method)

In Java, **execution always starts from the main() method**, which must be defined in some class.

✅ **Syntax:**

```
public class MyClass {
    public static void main(String[] args) {
        // code starts here
    }
}
```

🔍 **Explanation of Keywords:**

Keyword	**Meaning**
public	accessible from anywhere
static	no object needed to call main()
void	does not return any value
String[] args	receives command-line arguments

🧪 **Example:**

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}
```

2. Command Line Arguments

Java allows you to pass input to your program from the command line.

These inputs are received in the String[] args array of the main() method.

✅ **Example:**

```
public class CommandLineExample {
    public static void main(String[] args) {
        System.out.println("Number of arguments: " + args.length);
        for (int i = 0; i < args.length; i++) {
            System.out.println("Argument " + (i+1) + ": " + args[i]);
        }
    }
}
```

▶️ **Running It:**

If compiled as CommandLineExample.java:

```
> javac CommandLineExample.java

> java CommandLineExample India Gujarat 2025
```

🖨 **Output:**

```
Number of arguments: 3

Argument 1: India

Argument 2: Gujarat

Argument 3: 2025
```

3. finalize() Method (Deprecated in Java 9+)

- The finalize() method is **called by the Garbage Collector** before an object is destroyed.
- It is used to perform **cleanup activities**, such as closing file handles or releasing resources.

✅ **Syntax:**

```
protected void finalize() throws Throwable {
   // cleanup code
}
```

⚠️ **Note:**

- It is not guaranteed to run.
- Java 9 and onwards **deprecated** this method.
- Recommended alternative: use **try-with-resources** or AutoCloseable.

🧪 **Example:**

```
public class FinalizeExample {
    public void finalize() {
        System.out.println("Object is garbage collected");
    }

    public static void main(String[] args) {
        FinalizeExample obj = new FinalizeExample();
        obj = null;
        System.gc(); // Request GC
        System.out.println("End of main method");
    }
}
```

Output (may vary):

```
End of main method
Object is garbage collected
```

Summary Table

Concept	**Description**
main() method	Entry point of Java program
args[] parameter	Holds command-line inputs
finalize() method	Cleanup before GC destroys an object (Deprecated)