# Java Programming Day 7

*By Kavuri Santosh Kumar*

## CLASS MEMBERS:

**class members:**

In Java, **class members** refer to the components of a class, such as fields, methods, constructors, and nested classes. These members can be classified as **local** and **global** based on their usage.

---

### 1. Local Members

Local members are variables or methods declared and used **within a method, constructor, or block** of code. They are accessible only within the scope in which they are defined.

- **Accessibility**: Not accessible outside the method or block where they are defined.

**Examples of Local Members**

```
class ExampleLocal {
    void display() {
        int localVariable = 10; // Local variable
        System.out.println("Local Variable: " + localVariable);
    }

    public static void main(String[] args) {
        ExampleLocal obj = new ExampleLocal();
        obj.display(); // Accessing the method with local members
    }
}
```

- Here, localVariable is a local member, and it can only be accessed inside the display method.

---

## 2. Global Members

Global members are variables, methods, or nested classes declared **within the class but outside any method, constructor, or block**. These members are accessible throughout the class, and their accessibility depends on their access modifiers.

### Characteristics:

- **Scope**: Throughout the class (and possibly beyond, depending on access modifiers).
- **Types**: Can be **instance variables** (non-static) or **class variables** (static).
  - **Instance Variables**: Associated with an object of the class.
  - **Class Variables**: Shared among all objects of the class.
- **Default Initialization**: Automatically initialized to default values if not explicitly initialized.

### Examples of Global Members

```
class ExampleGlobal {
   int globalInstanceVariable = 20; // Instance variable (global)
   static int globalStaticVariable = 30; // Class variable (global)

   void display() {
      System.out.println("Instance Variable: " + globalInstanceVariable);
      System.out.println("Static Variable: " + globalStaticVariable);
   }

   public static void main(String[] args) {
      // Create an instance of the class to access the instance variable
      ExampleGlobal obj = new ExampleGlobal();
      obj.display(); // Accessing instance and static variables via the method

      // Access static variable directly without an instance
      System.out.println("Accessing Static Variable Directly: " +
ExampleGlobal.globalStaticVariable);
   }
}
```

- Here, globalInstanceVariable and globalStaticVariable are global members accessible throughout the Example class.

---

**Key Differences**

| Aspect | Local Members | Global Members |
|---|---|---|
| **Defined in** | Inside methods, constructors, or blocks | Inside the class but outside methods |
| **Scope** | Limited to the block in which defined | Available throughout the class |
| **Default Initialization** | Not initialized by default | Automatically initialized |
| **Modifiers Allowed** | No (cannot have access modifiers) | Yes (can use access modifiers like private, public, etc.) |

---

**Global Members**

**I. Static Global Members/Class Members**

**II. Non-Static Global Members/Object Members/Instance Members/Instances**

**I. Static Global Members/Class Members:-**

**i. Static Variable/Class Variable.**
**ii. Static Initializer Block(SIB).**
**iii. Static Method/Function.**

**II. Non-Static Global Members/Object Members/Instance Members/Instances:-**

**i. Non-Static Variable/ Instance Variable/Object.**
**ii. Non-Static Initializer Block(IIB)/ Instance Initializer Block(IIB)**
**iii. Non-Static Method/Function.**

**I. Static Global Members/Class Members:-**

**i. Static Variable/Class Variable.**
**ii. Static Initializer Block(SIB).**
**iii. Static Method/Function.**

**Example:-**

```
class Sample1 {
   static int i;
   static float f;
   static double d;
   static char ch;
   static boolean b;
   static String s;
   public static void main(String[] args) {
      System.out.println(i);
      System.out.println(f);
      System.out.println(d);
      System.out.println(ch);
      System.out.println(b);
      System.out.println(s);
   }
}
```

**Output:**

0

0.0

0.0

false

null

**Syntax:**

```
class ClassName {
    // Static Variable
    static DataType staticVariable = initialValue;

    // Static Initializer Block
    static {
        // Initialization logic for static variables or one-time execution code
    }

    // Static Method
    static ReturnType staticMethodName(Parameters) {
        // Code that operates on static members
    }

    public static void main(String[] args) {
        // Access static variable
        System.out.println(ClassName.staticVariable);

        // Call static method
        ClassName.staticMethodName(Arguments);
    }
}
```

**Example:**

```
class StaticMembersExample {
    // Static Variable
    static int staticVariable;

    // Static Initializer Block
    static {
        staticVariable = 10;
        System.out.println("Static Initializer Block Executed: staticVariable = " + staticVariable);
    }

    // Static Method
    static void display() {
        System.out.println("Static Method Called: staticVariable = " + staticVariable);
    }

    public static void main(String[] args) {
        System.out.println("Main Method Started");

        // Accessing static variable directly
        System.out.println("Accessing Static Variable Directly: " + staticVariable);

        // Accessing static method
        StaticMembersExample.display();
    }
}
```

**Output:**

```
Static Initializer Block Executed: staticVariable = 10
Main Method Started
Accessing Static Variable Directly: 10
Static Method Called: staticVariable = 10
```

**Note:**

- Any static variables can be accessed within the static content (content of static initializer blocks or static methods) in two ways:
  i. Directly using the variable name.
  ii. By using the class name followed by the variable name (i.e., ClassName.variableName).
- We can have **local** and **global** variables with the same name, but they will be treated as different variables.
- We **cannot** declare two local variables and two global variables with the same name in the same scope.
- **Local variables** do not have default values, whereas **global variables** (both static and non-static) have default values assigned by the JVM.

---

**Loading Process of the Class:**

- The **class loader** creates a static area with a name similar to the class name.
- It loads all the static members of the class and assigns them default values.
- It executes all the **static initializers** (including **static initializer blocks (SIBs)**) one by one, from top to bottom, in the order they are defined in the class.

**Note:-**

Any static members(static variable or methods) can be accessed or used by two ways:

**i. Using only the name of the variable or method.**

**ii. Using CalssName.variableName**

**Or**

**ClassName.methodName(actual argument is optional)**

**Static Initializer Block:-**

It is an anonymous block declared with a static keyword and it executes during lading process of the class.

We can write any number of SIB's and they all executes one by one from top to bottom order.

---

**Non-Static Global Members:**

- Any global member declared without the `static` keyword is known as a non-static global member.
- There are 3 types of non-static members:
  i. **Non-static variables/Instance variables**
  ii. **Non-static initializer blocks/Instance initializer blocks (IIB)**
  iii. **Non-static methods**

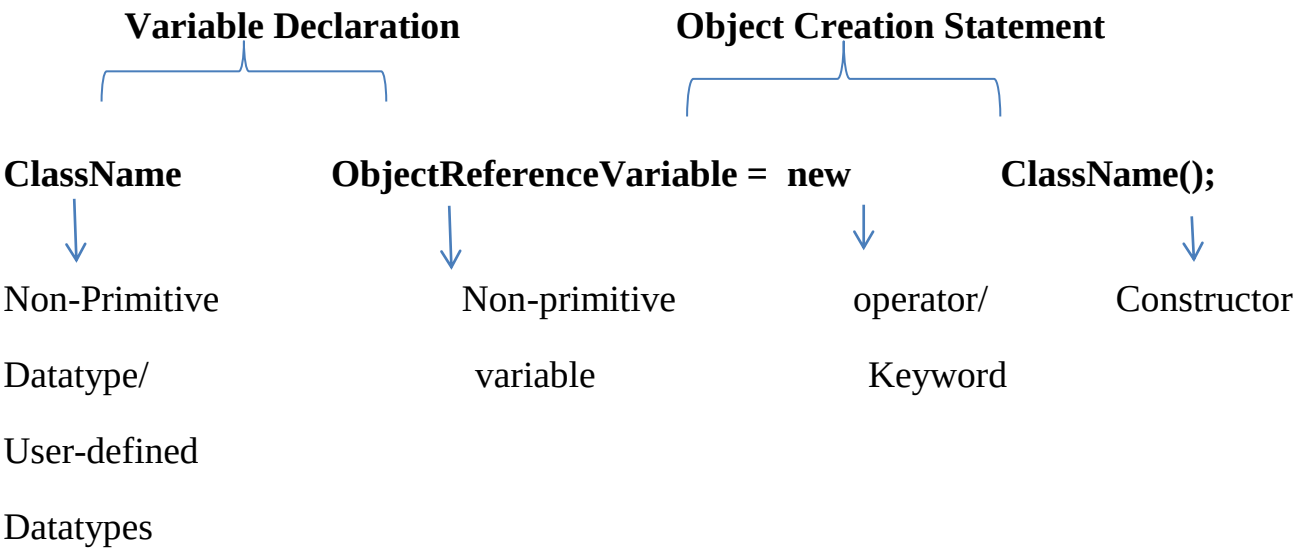
**Execution Details:**

- **Static Members**:
  Static members are automatically loaded into the class's static area by the class loader. The JVM automatically executes static methods without the need for explicit object creation.
- **Non-Static Members**:
  Non-static members are not automatically loaded by the class loader. Developers must explicitly create an object using the `new` operator and a constructor to allocate memory for non-static members. Once the object is created, non-static members can be accessed and executed using an object reference variable.

## Object/Instance:-

-object is a block of memory created during the runtime to load non-static members of the class.

## Syntax to create object:-



**i. Non-static variable/instance variable:-**

-All global variables declared without static are known as non-static variable.

-To execute non-static variables , first we need to create an object by using object reference variable only. So that we can fetch non-static members.

**Loading process of an object:-**

-new operator creates an object with the help of reference/address.

-loads all the non-static members with default values.

-Executes non-static initializer from top to bottom order one by one.

-Once the object is created (instantiation is done) new operator returns the reference.

**Note:-**

**-**Static members can have single copy of memory block, where as non-static members can have multiple copies of memory block. Since objects can be created any number of times by the developer using new operator.

-Any number of objects created by the developer will be automatically having internal reference to class static area by JVM.

**Syntax:**

```
class ClassName {
   // Non-static Variable (Instance Variable)
   DataType instanceVariable = initialValue;

   // Non-static Initializer Block (IIB)
   {
      // Code to initialize instance variables or other instance-related logic
   }

   // Non-static Method
   ReturnType nonStaticMethodName(Parameters) {
      // Code that operates on instance variables
   }
```

```
    public static void main(String[] args) {
        // Create an instance of the class
        ClassName obj = new ClassName();

        // Access instance variable
        System.out.println(obj.instanceVariable);

        // Call non-static method
        obj.nonStaticMethodName(Arguments);
    }
}
```

**Example:-**

```
class Example {
    // Non-static Variable
    int instanceVariable = 10;

    // Non-static Initializer Block (IIB)
    {
        instanceVariable += 5; // Modify the instance variable
        System.out.println("Instance Initializer Block: instanceVariable = " + instanceVariable);
    }

    // Non-static Method
    void display() {
        System.out.println("Non-static Method: instanceVariable = " + instanceVariable);
    }

    public static void main(String[] args) {
        System.out.println("Main Method Started");

        // Create an object to access non-static members
        Example obj = new Example();

        // Access instance variable
        System.out.println("Accessing Instance Variable: " + obj.instanceVariable);

        // Call non-static method
        obj.display();
    }
}
```

**Output:**

```
Main Method Started
Instance Initializer Block: instanceVariable = 15
Accessing Instance Variable: 15
Non-static Method: instanceVariable = 15
```

**Example for both static and Non static:**

```
class MixedExample {
    // Static Variable
    static int staticVariable = 100;

    // Non-static Variable
    int instanceVariable = 50;

    // Static Initializer Block
    static {
        staticVariable += 10; // Initialize or modify static variable
        System.out.println("Static Initializer Block: staticVariable = " + staticVariable);
    }

    // Non-static Initializer Block
    {
        instanceVariable += 5; // Initialize or modify instance variable
        System.out.println("Instance Initializer Block: instanceVariable = " + instanceVariable);
    }

    // Static Method
    static void staticMethod() {
        System.out.println("Static Method: staticVariable = " + staticVariable);
        // Cannot directly access instance variables here
    }

    // Non-static Method
    void nonStaticMethod() {
        System.out.println("Non-static Method: instanceVariable = " + instanceVariable);
        System.out.println("Accessing staticVariable in Non-static Method: " + staticVariable);
    }
```

```
public static void main(String[] args) {
        System.out.println("Main Method Started");

        // Access static members
        MixedExample.staticMethod();
        System.out.println("Accessing Static Variable Directly: " +
MixedExample.staticVariable);

        // Create an instance of the class to access non-static members
        MixedExample obj = new MixedExample();
        System.out.println("Accessing Instance Variable: " + obj.instanceVariable);
        obj.nonStaticMethod();
    }
}
```

**Difference between Static members and non-static members**

| Static Members | Non-static Members |
|---|---|
| Static members have single copy of memory block. | Non-static members can have multiple copies of memory block. |
| Static members are automatically loaded into class static area. | Non-static members must be loaded into the object explicitly by the developer using new operator. |
| This keyword cannot be used in a static context. | This keyword can be used within non-static context. |
| We don’t have to create an object to execute static members. | Creation of object is mandatory to execute non-static members. |
| If any change occurs to the static members it is common to the all objects. | If any non-static member of an object is modified, then the change will modified only that particular object and the non-static members of other objects are unchanged. |