

Lec 25 Static Keyword:

This lesson explains polymorphism and the static keyword in Java. Polymorphism allows a single action to behave differently based on the object, while static variables and methods are shared across all instances of a class and help save memory by storing common data only once.

Static keyword used in :

- **Instance variable / Method / Block / Nested class.**
- **(Not on local variable).**
- **Class variable.**
- **Memory allocated at class loading.**
-

Types of Polymorphism

There are two main types: compile-time (static/early binding) and run-time (dynamic/late binding). Compile-time polymorphism is resolved when the code is compiled, like method overloading. Run-time polymorphism is decided while the program runs, like method overriding.

Parent and Child References in Polymorphism

In run-time polymorphism, a parent class reference can point to a child class object, allowing methods to be called based on the actual object type at run time. This is called upcasting.

Introduction to the Static Keyword

The static keyword is used in Java for memory management. Static members (variables, methods, blocks, nested classes) belong to the class itself rather than any object, meaning they are shared across all instances.

Where Static Can Be Applied

Static can be used with variables (outside methods), methods, blocks, and nested classes, but not with constructors or local variables inside methods.

Static Variables vs Instance Variables

A static variable is also called a class variable and is not tied to any specific object. It is allocated memory when the class is loaded, not when an object is created, and is stored in the method area, not the heap.

Practical Example: Employee Data and Memory Management

- If a company has many employees, most data (like name, age, salary) is unique for each, but some data (like company ID) is common. Storing company ID as a static variable saves memory since it is stored only once, not with every object.
- We will analyze our data and see which field is same for all instances/objects. (Company ID → same for all emp)
- If we declare a instance variable as static then we don't need to pass it in the constructor for every creation of object (memory utilization).
- Now this Static variable will be called class variable and will not be with instance variables in memory.
- Object class's toString() method gives hexacode value.
- But if we override this toString() method in our class then it will work accordingly.

Static Data Storage Location

Static data is stored in the method area of the JVM, separate from each object's memory in the heap. This

ensures that static variables are shared and not duplicated.

The toString() Method and Overriding

The toString() method in Java comes from the Object class. Overriding it allows objects to print their data in a readable format instead of a default hexadecimal value. Printing an object reference automatically calls toString().

Printing Data: Methods vs toString()

Data can be printed using custom methods or by overriding toString(). Using toString() is more convenient because it is called automatically when printing object references.

Static vs Non-Static Variables: Behavior Demonstration

Non-static (instance) variables are unique to each object, so changes in one object do not affect others. Static variables are shared, so changes are visible across all objects. This is demonstrated by incrementing a static variable in constructors and observing its value across instances.

Static Variable Increment: Shared State Across Objects

When a static variable is incremented in the constructor, its new value is visible to all instances because it is shared. In contrast, instance variables keep their own separate values.

Static Block:

Constructor Vs Static instance variable → static will initialize at class loading and after that constructor call will update it.

Constructor → Object specific

Static Block → Class specific

Q. How to print something without writing a single line of code in main() ?

Ans. Make a static block and write in it accordingly.

This lesson explains the concepts of static variables, static blocks, and static methods in Java. It highlights how static members belong to the class rather than instances, making their data and behavior common for

all objects, and shows why static and non-static members are accessed differently.

Static Variables and Their Importance

Static variables are class-level variables shared among all instances. Changing a static variable in one object affects its value for all objects of that class.

Initialization of Static Variables

Static variables should be initialized during class loading, not through constructors. Using constructors for static variables can lead to inconsistent values across objects.

Static Block for Initialization

A static block is a code block that runs once when the class is loaded, allowing initialization of static variables before any object is created.

Static Data and Object Creation

Once a static variable is initialized, all objects access the same value. Updating the static variable updates it

for every object.

Control Flow: Static Block, Constructor, and Methods

When a program runs, the static block executes first, followed by the constructor when an object is created, and then instance methods. The static block ensures static data is ready before anything else.

JVM and Static Execution

The JVM loads classes and executes static blocks before the main method. The main method is also static, so static initialization always happens first.

Static vs Non-Static Initialization

Static blocks initialize static variables, while constructors initialize instance variables. Each object gets its own instance variables, but static variables are shared.

Creating and Using Static Methods

Static methods belong to the class and can be called without creating an object. They are used for logic that

doesn't depend on object data and are stored separately from instance methods.

Static Methods Cannot Access Non-Static Data

Static methods cannot directly use instance variables or instance methods because static context is class-level, and instance data is object-level. Attempting to access instance data from static methods causes errors.

Memory Allocation: Static vs Instance Variables

Instance variables are stored in the heap with each object, while static variables are stored in a separate area (Method area) and are not tied to any object. Only the object that owns an instance variable can access it.

Java Rules for Static and Instance Members

Java enforces that instance data cannot be accessed from static methods to maintain data integrity and security. Static data is common, but instance data is private to each object.

Static Inside Non-Static and Vice Versa

Static members can be used inside non-static methods, but not the other way around. This allows all objects to access common data, but object-specific data stays private.

Overloading the Main Method

The main method in Java can be overloaded with different parameters, but the JVM only calls the standard signature. Other main methods can be called manually if needed. Make new main below main with (int args) and call main(10) from main(String[] args).

Static and Inheritance (Preview)

The interaction of static members with inheritance is important but will be covered in detail with interfaces and inheritance in future lessons.