06/08/2025 Session – OOPJ

1. **Uses of this keyword**
2. **Uses of static keyword**
3. **Nested Classes** and **Inner Classes**

**1. Uses of this keyword**

**This** keyword refers to the **current object** in Java. It is used to remove ambiguity, call constructors, pass the current object, and more.

**Uses of this**

1. **To refer to current object**
2. **To invoke current class method**
3. **To invoke current class constructor (constructor chaining)**
4. **To pass the current object as a parameter**
5. **To return the current object from a method**
6. **To differentiate instance variables from parameters with the same name**

**Example – Differentiating instance variables**

class Student {

String name;

int age;

Student(String name, int age) {

this.name = name; // 'this' differentiates instance variable from parameter

this.age = age;

}

void display() {

System.out.println("Name: " + this.name + ", Age: " + this.age);

}

}

public class ThisExample {

public static void main(String[] args) {

Student s1 = new Student("Ishan", 21);

s1.display();

}

}

**Output:**

Name: Ishan, Age: 21

**2. Uses of static keyword**

When you declare a variable as **static** in Java, it means:

**1. Belongs to the Class, Not Objects**

* A **static variable** is shared by **all objects** of the class.
* It’s created only **once** in memory when the class is loaded (at class level), not for each object.
* All instances see the same value for that variable.

**2. Lifetime**

* Exists **for the entire program run** after the class is loaded into memory.
* Memory is allocated **in the Method Area**, not on the Heap (where object variables live).

**3. Access**

* Can be accessed using **class name** (e.g., ClassName.variable) without creating an object.
* Can also be accessed using an object reference, but that’s not recommended for clarity.

**Example**

class Counter {

static int count = 0; // static variable

Counter() {

count++; // increments shared count

}

}

public class StaticVariableDemo {

public static void main(String[] args) {

Counter c1 = new Counter();

Counter c2 = new Counter();

Counter c3 = new Counter();

System.out.println(Counter.count); // Access without object

}

}

**Output:**

3

**In above program:**

* count is **shared** among all objects.
* Every time a Counter object is created, count increases.
* Even if we create 100 objects, there is **only one count variable**.

💡 **In short:**  
A static variable is **class-level**, **shared**, **single copy**, and **lives till the program ends**.

The static keyword is used for **class-level members** that are shared among all objects.

**Where we can use static**

1. **Static variables** – shared among all instances
2. **Static methods** – can be called without creating an object
3. **Static blocks** – run only once when the class is loaded
4. **Static nested classes** – nested classes declared with static keyword

**Example – Static variable & method**

class Counter {

static int count = 0; // shared variable

Counter() {

count++;

}

static void showCount() {

System.out.println("Total Objects: " + count);

}

}

public class StaticExample {

public static void main(String[] args) {

new Counter();

new Counter();

Counter.showCount(); // calling without object

}

}

**Output:**

Total Objects: 2

**3. Nested Classes and Inner Classes**

**Nested Class**

A **nested class** is a class defined inside another class.  
Types:

1. **Non-static nested class (Inner class)**
2. **Static nested class**
3. **Local inner class** (inside a method)
4. **Anonymous inner class** (without a name, used for overriding methods or implementing interfaces)

**Example – Inner class**

class Outer {

int outerVar = 10;

class Inner { // non-static nested class

void display() {

System.out.println("Outer Variable: " + outerVar);

}

}

}

public class InnerClassExample {

public static void main(String[] args) {

Outer outer = new Outer();

Outer.Inner inner = outer.new Inner(); // create inner object

inner.display();

}

}

**Output:**

Outer Variable: 10

**Example – Static nested class**

class OuterStatic {

static int data = 100;

static class Nested {

void display() {

System.out.println("Data: " + data);

}

}

}

public class StaticNestedExample {

public static void main(String[] args) {

OuterStatic.Nested obj = new OuterStatic.Nested();

obj.display();

}

}

**Output:**

Data: 100

**Example – Local inner class**

class LocalInnerDemo {

void outerMethod() {

class LocalInner {

void show() {

System.out.println("Inside Local Inner Class");

}

}

LocalInner obj = new LocalInner();

obj.show();

}

}

public class LocalInnerExample {

public static void main(String[] args) {

new LocalInnerDemo().outerMethod();

}

}

**Output:**

Inside Local Inner Class

**Example – Anonymous inner class**

abstract class Animal {

abstract void sound();

}

public class AnonymousInnerExample {

public static void main(String[] args) {

Animal dog = new Animal() {

void sound() {

System.out.println("Dog barks");

}

};

dog.sound();

}

}

**Output:**

Dog barks

**Key Differences – Inner class vs Static nested class**

| **Feature** | **Inner Class** | **Static Nested Class** |
| --- | --- | --- |
| Requires outer class object? | Yes | No |
| Can access non-static members of outer class? | Yes | No (only static members) |
| Declared with static keyword? | No | Yes |