Static in Java – Complete In-Depth Notes

---

1. Background: How Java Program Runs (Big Picture)

Java is **both compiled and interpreted**:

1. **javac** compiles `.java` source code → **bytecode** (`.class` file).
2. **JVM** loads the class, verifies bytecode, and executes it.
3. Execution happens via:
    - **Interpreter** (line-by-line initially)
    - **JIT Compiler** (converts frequently used bytecode to native machine code)

Before `main()` runs, **class loading** happens. This is where **static members play their role**.

---

2. JVM Memory Areas (Important for Static)

When a Java program runs, JVM creates several memory areas:

2.1 Method Area (a.k.a. Class Area / MetaSpace)

Stores **class-level data**:

- Class structure
- Static variables
- Static methods
- Static blocks
- Constant pool

 Static members live here (one copy per class).

2.2 Heap Memory

Stores:

- Objects
- Instance variables

 One object = one set of instance variables.

2.3 Stack Memory

Stores:

- Method calls
- Local variables
- References

 Each thread has its own stack.

---

3. What Does `static` Mean?

`static` means:

Belongs to the class, not to an object

Key idea:

- Static members are created **once**, when the class is loaded.
- Instance members are created **every time an object is created**.

---

4. Static Variables

4.1 Definition

A variable declared with `static` keyword is a **class variable**.

```
class Demo {
    static int x = 10;
}
```

4.2 Memory Behavior

- Memory allocated in **Method Area**
- Only **one copy**, shared by all objects

```
Class Demo
└── static x = 10   (Method Area)
```

4.3 Access Rules

- Can be accessed:
    - Using class name → `Demo.x`
    - Using object (allowed but NOT recommended)

4.4 Why Static Variables?

- To store **common/shared data**
- Saves memory

Real-world examples:

- Bank interest rate
- College name
- Company name

---

5. Instance vs Static Variables (Comparison)

Feature	Static Variable	Instance Variable
Belongs to	Class	Object
Memory	Method Area	Heap
Copies	One	One per object
Access	Class name	Object reference

---

6. Static Methods

6.1 Definition

Method declared using `static` keyword.

```
static void show() {
    System.out.println("Hello");
}
```

6.2 Key Rules

- Can access **only static members directly**
- Cannot access instance variables without object

 Not allowed directly:

```
int a = 10;
static void test() {
    System.out.println(a); // error
}
```

✔ Correct:

```
static void test() {
    Demo d = new Demo();
    System.out.println(d.a);
}
```

6.3 Why `main()` Is Static?

```
public static void main(String[] args)
```

Reason:

- JVM must call `main()` **without creating object**
- Static allows JVM to execute it during class loading

---

7. Static Blocks

7.1 Purpose

- Used to **initialize static variables**
- Executes **automatically during class loading**

```
class Demo {
    static int x;

    static {
        x = 100;
        System.out.println("Static block executed");
    }
}
```

7.2 Execution Order

1. Static variables (default values)
2. Static blocks (top to bottom)
3. `main()` method

---

8. Order of Execution (Very Important)

```
class Test {
    static int x = 10;

    static {
        System.out.println("Static Block");
    }

    Test() {
```

```
        System.out.println("Constructor");
    }

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

Output Order:

1. Static variable initialization
2. Static block
3. Main method
4. Constructor

---

9. Static vs Non-Static Blocks

Static Block

- Runs **once**
- During **class loading**
- Used for class-level initialization

Instance Initialization Block (IIB)

- Runs **every time object is created**
- Executes **before constructor**

---

10. Static Methods & Object Creation

Static members:

- Do **not require object**
- Loaded first

Instance members:

- Require object
- Loaded after static phase

 That's why **static cannot directly use instance data**.

---

11. Static and Garbage Collection

- Static variables are **not garbage-collected** until:
  - JVM shuts down
  - Class is unloaded

Reason:

- Static variables are referenced by class loader

---

12. Static Use-Cases (When to Use)

✔ Utility methods (`Math`, `Collections`)
✔ Constants (`public static final`)
✔ Shared configuration
✔ Entry point (`main()`)

 Avoid static for:

- Object-specific data
- Heavy mutable data (can cause memory leaks)

---

13. Common Interview Traps

 Can static access non-static?
→ Directly NO, ✔ via object

 Can constructor be static?
→ No (constructor needs object)

 Can static be overridden?
→ No (method hiding happens)

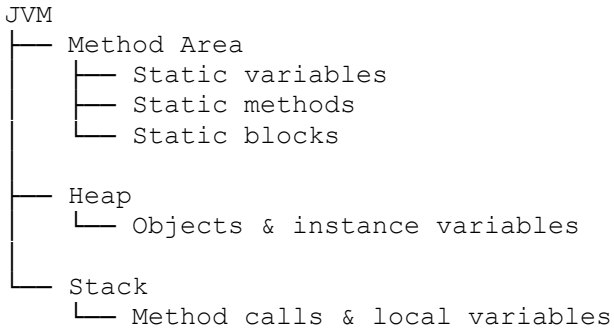
---

14. One-Line Summary

Static members belong to class, stored in Method Area, loaded once during class loading, shared by all objects, and executed before `main()` and object creation.

---

15. Mental Memory Diagram


JVM
Method Area
Static variables
Static methods
Static blocks
Heap
Objects & instance variables
Stack
Method calls & local variables