**1. What is the static keyword in Java?**

**Answer:**  
The static keyword is used for memory management. It can be applied to variables, methods, blocks, and nested classes. When a member is declared static:

* It belongs to the **class**, not to instances.
* It is **shared across all instances**.

**2. Where can we use static in Java?**

**Answer:**

* Static **variable** (class-level variable)
* Static **method** (can be called without object)
* Static **block** (used for static initialization)
* Static **nested class**

**3. What is a static block? When is it executed?**

**Answer:**  
A static block is used for static initialization of a class.

java

CopyEdit

static {

System.out.println("Static block executed");

}

It runs **once**, when the class is first loaded by the JVM.

**Cross Question:**  
Q: What happens if you have multiple static blocks?  
A: They run in the **order they appear** in the class.

**4. Can a static method access instance variables?**

**Answer:**  
No. Static methods can only access **static data**. They cannot access **instance variables** or **non-static methods** directly.

**Reason:** Static context does not have a reference to this.

**5. What is the output of the following?**

java

CopyEdit

class Test {

static int count = 0;

Test() {

count++;

}

public static void main(String[] args) {

new Test();

new Test();

System.out.println(Test.count);

}

}

**Answer:**  
2. Because count is static, it’s shared across all instances.

**6. Can we override static methods in Java?**

**Answer:**  
**No**, static methods are **not overridden**, they are **hidden**.

**Example:**

java

CopyEdit

class Parent {

static void display() { System.out.println("Parent"); }

}

class Child extends Parent {

static void display() { System.out.println("Child"); }

}

Child.display() will call Child's version, but it's **method hiding**, not overriding.

**7. Why can’t we use this or super in static context?**

**Answer:**  
this and super refer to object instances. Static context doesn’t belong to any object, so these references are invalid.

**🔹 Final in Java**

**1. What is the final keyword in Java?**

**Answer:**  
final means **unchangeable** or **constant**. It can be used with:

* **Variables** (cannot be reassigned)
* **Methods** (cannot be overridden)
* **Classes** (cannot be extended)

**2. Can final variables be initialized later?**

**Answer:**  
Yes, if they are:

* **Instance variables**, they can be initialized in the **constructor**.
* **Local variables**, they must be initialized before use.
* **Blank final variables** are allowed, but must be initialized exactly once.

**3. What is a final method?**

**Answer:**  
A final method **cannot be overridden** by subclasses.

**Use case:** Prevent altering base behavior in inheritance.

**4. What is a final class?**

**Answer:**  
A class marked final **cannot be extended**.  
Example: java.lang.String is a final class.

**Reason:** To ensure immutability and security.

**5. Can a final method be overloaded?**

**Answer:**  
Yes. Overloading is based on method **signature**, not inheritance. Final restricts **overriding**, not **overloading**.

**6. Is final variable thread-safe?**

**Answer:**  
Yes. Once a final variable is constructed and assigned, it is guaranteed to be visible correctly across threads. Especially useful in **immutable objects**.

**7. What happens if you try to reassign a final variable?**

java

CopyEdit

final int x = 10;

x = 20; // Compilation error

**Answer:**  
Compile-time error: cannot assign a value to final variable x.

**8. Can a final reference variable point to a different object?**

**Answer:**  
No. But the object it points to **can be modified**, if mutable.

java

CopyEdit

final List<String> list = new ArrayList<>();

list.add("A"); // OK

list = new ArrayList<>(); // Compilation error

**9. Can we declare a constructor as final?**

**Answer:**  
No. Constructors are **not inherited**, so there's no point in marking them final.

**10. What is the difference between static final and final static?**

**Answer:**  
No difference. Order of keywords does **not matter** in Java modifiers.

java

CopyEdit

static final int MAX = 10; // same as final static int MAX = 10;

**✅ Quick Comparison**

| **Feature** | **static** | **final** |
| --- | --- | --- |
| Memory | Class-level | Constant (cannot be reassigned) |
| Inheritance | Not related | Prevents method override/class extend |
| Access | Without object | Must be initialized exactly once |
| Use Cases | Utility methods, constants | Constants, sealed behavior |

**🔶 Keyword Comparison Table**

| **Feature** | **static** | **final** | **abstract** | **synchronized** |
| --- | --- | --- | --- | --- |
| **Purpose** | Shared across instances (class-level) | Prevent modification/override/inheritance | Define contract without implementation | Thread safety (mutual exclusion) |
| **Usage** | Methods, variables, blocks, classes | Variables, methods, classes | Classes and methods | Methods or blocks |
| **Inheritance** | Inherited (but can be hidden) | Prevents override/extension | Must be extended | Inherited but locks must be understood |
| **Memory** | Allocated once in class loading | Once assigned, can't be changed | No memory until implemented | Lock acquired during execution |
| **Execution** | Called via class | Constant throughout execution | Abstract method has no body | One thread at a time can execute the block/method |
| **Example** | Utility methods, constants | Immutable variables, constant configurations | Base interfaces or template methods | Critical section, shared data access |

**🔶 Real-World Examples**

**✅ static Example — Utility Class**

java

CopyEdit

public class MathUtils {

public static int square(int x) {

return x \* x;

}

}

MathUtils.square(5); // No object needed

**Use case:** Logging utilities, helper methods, constants, or singleton accessors.

**✅ final Example — Immutable Config**

java

CopyEdit

public final class Config {

public static final String APP\_NAME = "InvoiceManager";

}

**Use case:** Constants, security (prevent subclassing), enforcing business rules.

**✅ abstract Example — Template Design Pattern**

java

CopyEdit

public abstract class PaymentProcessor {

public abstract void authenticate();

public void process() {

authenticate();

System.out.println("Transaction completed.");

}

}

**Use case:** Define *skeleton* behavior for subclasses (e.g., Spring’s AbstractController).

**✅ synchronized Example — Thread-safe Counter**

java

CopyEdit

public class Counter {

private int count = 0;

public synchronized void increment() {

count++;

}

public synchronized int getCount() {

return count;

}

}

**Use case:** Bank transactions, shared resource access (e.g., messaging, counters, cache updates).

**🔶 Coding Exercises**

**🧪 1. Static vs Instance**

java

CopyEdit

class Employee {

static int employeeCount = 0;

int empId;

public Employee(int id) {

empId = id;

employeeCount++;

}

public static int getEmployeeCount() {

return employeeCount;

}

}

**Q:** What is the output of:

java

CopyEdit

new Employee(101);

new Employee(102);

System.out.println(Employee.getEmployeeCount());

✅ **Answer:** 2 — because employeeCount is static.

**🧪 2. Final Variable Initialization**

java

CopyEdit

public class Vehicle {

final String model;

public Vehicle(String model) {

this.model = model;

}

public void printModel() {

System.out.println("Model: " + model);

}

}

**Q:** Can model be reassigned later?  
✅ **Answer:** No — final instance variables must be assigned once.

**🧪 3. Abstract Method Implementation**

java

CopyEdit

abstract class Shape {

abstract double area();

}

class Circle extends Shape {

double radius;

Circle(double r) { radius = r; }

@Override

double area() {

return Math.PI \* radius \* radius;

}

}

**Q:** What happens if you forget to implement area() in Circle?

✅ **Answer:** Compile-time error — abstract method must be implemented in a concrete subclass.

**🧪 4. Synchronized Method Test**

java

CopyEdit

class Printer {

public synchronized void print(String doc) {

System.out.println(Thread.currentThread().getName() + ": Printing " + doc);

}

}

**Q:** What is the output if two threads try to print at the same time?

✅ **Answer:** Only one thread prints at a time — due to synchronization.

**🔶 Summary of Best Practices**

| **Keyword** | **Best Practice** |
| --- | --- |
| static | Use for constants and utility methods. Avoid static state in multithreading. |
| final | Prefer final fields for immutability and thread-safety. |
| abstract | Use in frameworks or when designing base template classes. |
| synchronized | Use only when necessary. Consider ReentrantLock or java.util.concurrent. |