1Z0-808 Exam Topic Reviewer

TopicId: 1007
Topic: Variable Scope and Lifetime

August 5, 2025

Topic 1007: Variable Scope and Lifetime

Thinking Like the Compiler: Where Variables Live and Die

Not all variables are created equal. Where you declare a variable determines where it can be used (its scope) and how long it exists in memory (its lifetime). The compiler is extremely strict about scope rules. Many exam questions are designed to look correct at a glance but contain a subtle scope violation that causes a compile error. Your job is to spot it.

The Three Scopes of Data

Java has three primary types of variables based on where they are declared.

Variable Type	W
Local	Ins
Instance	Ins
Static	Ins

1. Local Variables

These are temporary variables that exist only within the block of code where they are declared.

- **No Default Values:** This is the single most important rule. A local variable **must be initialized** before it is used. The compiler will stop you if you don't.
- **Scope:** A local variable's scope starts at the line it is declared and ends when its enclosing curly brace } is reached.

```
public void calculate() {
    int sum; // Declared, but not initialized
    // System.out.println(sum); // COMPILE ERROR: variable sum might not have bee

    sum = 0; // Initialization
    System.out.println(sum); // OK now

    for (int i = 0; i < 5; i++) {
        int temp = i * 2; // 'temp' is local to the for loop block
        System.out.println(temp);
    }
    // System.out.println(temp); // COMPILE ERROR: cannot find symbol 'temp'
}
```

2. Instance Variables (Non-Static Fields)

These variables belong to an instance (an object) of a class. Each object gets its own copy.

- **Default Values:** Instance variables are **always given a default value** if you don't provide one: `0` for numbers, `false` for booleans, `'\u0000'` for chars, and `null` for objects.

- **Lifetime:** They are created when an object is instantiated with `new` and are destroyed when the object is garbage collected.

3. Class Variables (Static Fields)

These variables are shared among all instances of a class. There is only one copy, no matter how many objects are created.

- **Default Values:** Like instance variables, static variables **always receive a default value**.
- **Lifetime:** They are created when the JVM loads the class and are destroyed when the program ends. They can be accessed using the class name, e.g., `MyClass.myStaticVar`.

```
public class Counter {
    int instanceCount = 0; // Instance variable, gets default value 0
    static int staticCount = 0; // Static variable, gets default value 0

    public Counter() {
        instanceCount++;
        staticCount++;
    }
}

// In some other method:
Counter c1 = new Counter();
System.out.println(c1.instanceCount); // 1
System.out.println(Counter.staticCount); // 1

Counter c2 = new Counter();
System.out.println(c2.instanceCount); // 1 (c2 has its own copy)
System.out.println(Counter.staticCount); // 2 (staticCount is shared)
```

Exam Trap: Variable Shadowing

When a local variable has the same name as an instance or static variable, it "shadows" (hides) the field. Within the local scope, the name refers to the local variable.

```
public class ShadowTest {
    String name = "Instance"; // Instance variable

    public void printName(String name) { // Local variable (parameter)
        // 'name' here refers to the method parameter
        System.out.println("Local name: " + name);

        // To access the shadowed instance variable, use 'this'
        System.out.println("Instance name: " + this.name);
    }
```

```
}

// Usage:
ShadowTest st = new ShadowTest();
st.printName("Local");
// Output:
// Local name: Local
// Instance name: Instance
```

Key Takeaways for the 1Z0-808 Exam

- **Initialization is Key:** Local variables have no default values. Instance and static variables do. The compiler enforces this for local variables.
- **Block Scope:** A variable declared inside any {...} block (if, for, while, or just a block) is only alive inside that block.
- **Shadowing:** A local variable beats an instance/static variable with the same name. Use `this.varName` for the instance variable and `ClassName.varName` for the static variable.
- **Order Matters:** A variable cannot be referenced before it is declared. This is true even for instance variables in some initialization scenarios, but is a strict rule for local variables.