Some Important Subtleties of Inheritance and Constructors in Java

(Not necessarily an all-inclusive list.)

(Please let me know if you see any errors.)

Contents

Constructors Are Not Inherited

- Subclasses do not inherit constructors from their parent class.
- Subclasses can call a superclass constructor using .

```
super(...)
```

↑ Back to Top

Constructor Chaining Happens Automatically

- All ancestor constructors execute when a subclass is instantiated.
- They are executed from the topmost superclass down to the subclass
- If not specified, Java inserts an implicit `super()`.

```
class A {
    A() {
        System.out.println("A");
    }
}

class B extends A {
    B() {
        System.out.println("B");
    }
}
```

Output:

```
A
B
```

↑ Back to Top

If the Superclass Has No No-Arg Constructor

- The subclass must explicitly call `super(args)`.
- Otherwise, compilation fails.

↑ Back to Top

The super() Call Must Be the First Statement

- No statements may appear before `super()`.
- If omitted, Java inserts it automatically (if possible).
- Java 24 introduces a preview feature allowing pre-constructor code.

↑ Back to Top

Instance Variable Initialization Order

1. Memory allocated for all fields.
2. Superclass initialization occurs first.
3. Subclass initialization follows.

↑ Back to Top

Avoid Calling Overridden Methods in Constructors

- Overridden methods may run before subclass fields initialize.
- This can cause inconsistent or unsafe behavior.

A superclass constructor should not invoke non-final, non-static methods that may be overridden in subclasses. Because of polymorphism, such a call can execute the subclass's overridden method before the subclass's instance variables have been initialized, potentially leading to inconsistent or erroneous behavior.

↑ Back to Top

this() vs super()

- `this()` calls another constructor in the same class.
- `super()` calls a superclass constructor.
- You may use one or the other — not both.

↑ Back to Top

Each Class Has Responsibility for Its Own Fields

- Each constructor initializes only its own class's fields.
- Superclass fields must be initialized via `super() for proper initialization`.

↑ Back to Top

Default Constructors and Inheritance

- No constructors → Java supplies a default no-arg constructor.
- Defining any constructor suppresses the default.
- Constructors are not inherited, but initialization behavior is.

↑ Back to Top

Abstract Classes Still Have Constructors

- Abstract classes cannot be instantiated.
- Their constructors still run during subclass creation.

↑ Back to Top

Constructors Are Not Polymorphic

- Constructors cannot be overridden.
- Execution depends on the instantiated class.

↑ Back to Top

Static and Instance Initializers

1. Static initializers (only once, when the class loads)
2. Instance variable initializers (superclass → subclass)

3. Constructors (superclass → subclass)

↑ Back to Top

Calling super in a Chain of Inheritance

1. Each level can call a specific superclass constructor
2. Each level can pass arguments upward, possibly modifying/validating them at each level
3. This provides a clear chain of responsibility for setting up the object state.

↑ Back to Top

Exception Handling in Constructors

- Checked exceptions must be declared or handled.
- If a superclass constructor throws a checked exception, the subclass constructor must throw it or handle it.
- This ensures all parts of the hierarchy honor exception contracts.

↑ Back to Top

Object Creation Sequence Summary

Example: Class C that extends B that extends A

1. Memory allocated for all fields.
2. Static initializers: A → B → C
3. Constructors: A → B → C

↑ Back to Top