1Z0-808 Exam Topic Reviewer

TopicId: 1003
Topic: Java Coding Conventions and Javadoc

August 5, 2025

Professionalism in Code: Conventions and Documentation

Writing code that works is only half the battle. Writing code that is readable, maintainable, and understandable by others (including your future self) is what separates an amateur from a professional. The exam won't directly ask you "Is this good style?", but it will use these conventions, and understanding them prevents confusion. Furthermore, knowing how comments and Javadoc work can help you spot subtle syntax errors.

1 Java Coding Conventions

These are the widely accepted standards for formatting Java code. While the compiler doesn't enforce most of them, your team and your sanity will thank you for following them.

1.1 Identifier Naming Conventions

This is the most critical convention to know. The exam often uses unconventional (but legal) naming to trip you up.

- **Classes Interfaces:** UpperCamelCase (or PascalCase). Examples: `String`, `Runnable`, `ArrayList`.
- **Methods Variables:** lowerCamelCase. Examples: `toString()`, `calculatePrice()`, `userName`, `itemCount`.
- **Constants (static final):** All uppercase, with words separated by underscores. Examples: `Integer.MAX`$_{V}ALUE$, `Math.PI`.
- `Packages: All lowercase, with levels separated by dots. Examples: java.util, java.time.format.`

Exam Trap: The compiler only cares about syntax, not style. All of the following are **100% legal and will compile**, but they violate convention. Do not let them throw you off.

```
public class my_class { // Legal, but bad style
    void MyMethod() {}   // Legal, but bad style
    int My_Int = 10;     // Legal, but bad style
}
```

2 Comments

Comments are ignored by the compiler. Their purpose is to explain the *why* behind your code, not the *what*.

- **Single-line:** Starts with `//` and goes to the end of the line.
- **Multi-line (or Block):** Starts with `/*` and ends with `*/`. Can span multiple lines.

- **Javadoc:** A special multi-line comment that starts with `/**` and ends with `*/`. Used by the `javadoc` tool.

Exam Trap: Multi-line comments **do not nest**. The first `*/` encountered will end the comment, which can lead to compilation errors.

```
/* This is an outer comment.
   /* This is a nested comment. */  <-- This ends the outer comment!
   This text here will cause a COMPILE ERROR.
*/
```

3 Javadoc: Generating API Documentation

The Javadoc tool (included in the JDK) parses these special comments to create professional HTML documentation for your code.

3.1 Structure and Placement

A Javadoc comment is placed immediately before the class, interface, method, or field it is describing.

3.2 Common Block Tags

These tags, identified by the `@` symbol, provide specific information.

- `@param parameterName description`: Describes a method's parameter.
- `@return description`: Describes a method's return value.
- `@throws ExceptionType description`: Describes an exception that a method may throw.
- `@since version`: Specifies the version the feature was added (e.g., `@since 1.2`).
- `@author name`: Specifies the author.
- `@version text`: Specifies the version of the class.

3.3 Example

Here is a well-documented method using Javadoc.

```
/**
 * Calculates the simple interest for a given principal.
 * <p>
 * This method does not handle negative values and assumes
 * a non-compounding interest calculation.
 *
 * @param principal   The initial amount of money. Must be positive.
 * @param rate        The annual interest rate as a decimal (e.g., 0.05 for 5%).
 * @param years       The time in years. Must be non-negative.
```

```
 * @return            The calculated simple interest.
 * @throws IllegalArgumentException if principal or years are negative.
 */
public double calculateSimpleInterest(double principal, double rate, int years) {
    if (principal < 0 || years < 0) {
        throw new IllegalArgumentException("Inputs must be non-negative.");
    }
    return principal * rate * years;
}
```

3.4 Generating Documentation

You use the `javadoc` command, which works very similarly to `javac`.

```
// Generate docs for one file into the 'docs' directory
javadoc -d docs src/com/mycorp/utils/Calculator.java
```

4 Key Takeaways for the 1Z0-808 Exam

- **Naming Conventions:** Know them cold. UpperCamelCase for types, lowerCamelCase for members, and $\text{ALL}_{C}APSforconstants.Don'tbefooledbylegal-but-unconventionalnames.$
- **Comment Nesting:** Remember that `/* ... */` comments do not nest.
- **Javadoc Syntax:** Recognize Javadoc comments (`/**...*/`) and know the purpose of the main tags: `@param`, `@return`, and `@throws`.