1Z0-808 Exam Topic Reviewer

TopicId: 1011
Topic: Break, Continue, and Labels

August 5, 2025

Topic 1011: Break, Continue, and Labels

Thinking Like the Compiler: Altering the Flow

Once a loop starts, it doesn't have to run to completion. Java provides tools to exit a loop early (`break`) or skip an iteration (`continue`). For complex nested loops, labels give you precise control over which loop to alter. The exam tests your ability to trace these jumps in execution flow accurately.

The `break` Statement

When the JVM encounters a `break` statement, it immediately terminates the innermost loop (or `switch`) statement it's in. Execution continues at the first statement *after* the terminated loop.

```
for (int i = 0; i < 10; i++) {
    if (i == 3) {
        break; // Exits the for loop entirely
    }
    System.out.print(i); // Prints 012
}
System.out.print(" End"); // Prints " End"
// FINAL OUTPUT: 012 End
```

The `continue` Statement

The `continue` statement immediately ends the *current iteration* and proceeds to the next one.

- In a `for` loop, `continue` jumps to the **update** expression.
- In a `while` or `do-while` loop, `continue` jumps to the **condition** check.

```
for (int i = 0; i < 5; i++) {
    if (i == 2) {
        continue; // Skips printing 2, jumps to i++
    }
    System.out.print(i); // Prints 0134
}
```

Exam Trap: A poorly placed `continue` in a `while` loop can cause an infinite loop by skipping the update statement.

Labels: Controlling Nested Loops

Labels give you the power to direct `break` and `continue` to a specific outer loop instead of just the innermost one.

Syntax: A label is a valid identifier followed by a colon, placed before a loop.
`myLabel: for(...)`

Labeled `break`

Exits the loop that has the specified label.

```
OUTER: for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        if (i == 1 && j == 1) {
            break OUTER; // Exits the OUTER loop, not just the inner one
        }
        System.out.print(i + "," + j + " | ");
    }
}
// OUTPUT: 0,0 | 0,1 | 0,2 | 1,0 |
```

Labeled `continue`

Skips the current iteration of the labeled loop and proceeds to its next iteration.

```
OUTER: for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        if (i == 1 && j == 1) {
            continue OUTER; // Skips rest of inner loop and goes to OUTER's updat
        }
        System.out.print(i + "," + j + " | ");
    }
}
// OUTPUT: 0,0 | 0,1 | 0,2 | 1,0 | 2,0 | 2,1 | 2,2 |
```

Key Takeaways for the 1Z0-808 Exam

- **`do-while` vs. `while`:** The `do-while` loop body is guaranteed to execute at least once.
- **`for` Loop Flexibility:** Remember that the three parts of a basic `for` loop are optional. `for(;;)` is a valid infinite loop.
- **Enhanced `for`:** The loop variable is a copy. Reassigning it (e.g., `var = ...`) does not change the source collection, but calling methods on it (e.g., `var.setValue()`) can.
- **`break`:** Exits the loop completely.
- **`continue`:** Skips the current iteration. In a `for` loop, it jumps to the update statement.
- **Labels:** When you see a labeled `break` or `continue`, carefully identify which loop is being targeted and trace the jump in execution.