1Z0-808 Exam Topic Reviewer

TopicId: 1029
Topic: Try-Catch-Finally Blocks

August 5, 2025

Handling Exceptions: The `try`, `catch`, and `finally` Structure

Now that we know the types of exceptions, let's look at the tools we use to handle them. The `try-catch-finally` structure is the fundamental mechanism for exception handling. The flow of control in these blocks is a favorite topic for exam questions.

0.1 The `try-catch` Flow

The basic structure involves a `try` block followed by one or more `catch` blocks.

- **`try`:** You place your "risky" code inside this block. If an exception occurs on any line, execution of the `try` block halts *immediately* and control transfers to the JVM to find a matching handler.
- **`catch`:** This block executes only if an exception of a matching type is thrown in the `try` block.

```
try {
    // Risky code
    System.out.println("1");
    int value = Integer.parseInt("abc"); // Throws NumberFormatException
    System.out.println("2"); // This line is NEVER reached
} catch (NumberFormatException e) {
    System.out.println("3"); // This block executes
} catch (Exception e) {
    System.out.println("4"); // This block is skipped
}
System.out.println("5"); // Execution continues here
// Output: 1, 3, 5
```

0.2 Catch Block Rules: Specificity and Multi-Catch

- **Order Matters (Crucial Exam Trap):** You must order multiple `catch` blocks from the most specific exception type (subclass) to the most general (superclass). Placing a superclass before a subclass makes the subclass's block unreachable, which is a **COMPILE ERROR**.

  ```
  try { ... }
  catch (Exception e) { ... }
  // catch (IOException e) { ... } // COMPILE ERROR: Unreachable catch block
  ```

- **Multi-Catch:** Since Java 7, you can catch multiple exception types in a single block using a pipe `|`. The exception variable in a multi-catch block is implicitly `final`.

  ```
  try { ... }
  catch (IOException | SQLException e) {
      // e = new IOException(); // COMPILE ERROR: e is final
      // handle error
  ```

```
    }
```

0.3 The Unstoppable `finally` Block

The `finally` block provides a mechanism to run code whether an exception occurs or not. It's for cleanup.

- **Execution Guarantee:** The `finally` block will execute after the `try` block finishes, even if there was an uncaught exception, or a `return` statement in the `try` or `catch` block.
- **When does it NOT run?** Only in extreme cases like a call to `System.exit()`, a fatal JVM error, or an infinite loop in the preceding blocks.
- **Exception Masking:** If an exception is thrown from the `try` or `catch` block, and *another* exception is thrown from the `finally` block, the exception from `finally` takes precedence and the original one is lost.

0.4 Try-with-Resources: The Modern Approach

For resources that need to be closed (like streams or database connections), Java 7 introduced the try-with-resources statement. Any object whose class implements `AutoCloseable` can be used.

```
// The resource is declared inside the parentheses
try (Scanner scanner = new Scanner(new File("test.txt"))) {
    // use the scanner
} catch (FileNotFoundException e) {
    // handle the exception
}
// No finally block needed! The scanner's close() method is called automatically
```

This is safer and cleaner than using a manual `finally` block. It also handles suppressed exceptions correctly.

Key Takeaways for the 1Z0-808 Exam

- Execution in a `try` block stops immediately when an exception is thrown.
- Order `catch` blocks from most specific to most general. A general catch block before a specific one is a compile error.
- The `finally` block almost always runs. It's the place for cleanup code.
- Prefer try-with-resources for any object that implements `AutoCloseable` to avoid resource leaks and write cleaner code.