Java Magazine

Java Magazine

Java SE, Quiz

Quiz yourself: The hierarchy of Java exception handlers

Mikalai Zaikin | November 15, 2021

Simon Roberts | November 15, 2021

Text Size 100%: – +

When you have multiple catch statements on a single try, which catch goes first? Which goes last?

Given the code

```
public class FooException extends Exception {
  public FooException(String msg) {
    super(msg);
  }
}
public class BarException extends RuntimeException {
  public BarException(Throwable reason) {
    super(reason);
  }
}

public class ExceptionalClass {
  void doSomethingGood() {
    try {
      doSomethingBad();
      System.out.print("All Good"); // line n1
    } catch(FooException fe) {      // line n2
    } catch(BarException be) {
    } catch(Exception e) {          // line n3
    }
  }
  void doSomethingBad() throws FooException {
    throw new FooException("Something Bad");
  }
}
```

Copy code snippet

Which statement is correct? Choose one.

A. Compilation fails because the `catch` clause in line n2 makes the `BarException` handler unreachable. The answer is A.

B. Compilation fails because the `Exception` handler in line n3 is unreachable. The answer is B.

C. Compilation fails because the `print` statement in line n1 is unreachable. The answer is C.

D. Compilation succeeds. The answer is D.

Answer. Java requires that when you have multiple `catch` statements on a single `try`, the more-specific exception handlers *must* appear before more-general ones. So, with this inheritance hierarchy

```
ExceptionA
     ↑
ExceptionB
     ↑
ExceptionC
```

Any `catch` block that catches `ExceptionA` would catch all three types of exception, since both `ExceptionB` and `ExceptionC` would be tripped by any test that detects `ExceptionA`. If you tried to put a `catch` for `ExceptionA` before a `catch` for `ExceptionB` (or `ExceptionC`) the compiler would reject this, since all three types of exception would be handled by the first `catch` block, and the subsequent blocks would be unreachable.

However, if two exceptions are in different branches of the inheritance tree and, thus, there is no hierarchy, no exception is more or less specific than the other, so no such problem arises.

This question has

```
             Exception
              ↗     ↖
RuntimeException     FooException
        ↑
   BarException
```

Given this inheritance hierarchy, a `BarException` is not assignment-compatible with `FooException`, nor vice versa; they are unrelated. For this reason, line n2 is valid as it stands. Indeed, if the order of the `catch`es were swapped, as follows, the code would still be valid:

```
try {
  doSomethingBad();
  System.out.print("All Good");
} catch(BarException be) {
} catch(FooException fe) {
```

Copy code snippet

This means option A is incorrect.

Option B suggests that the `Exception` handler is unreachable. However, this is not the case. Many exceptions are instances of `Exception` without being either `FooException` or `BarException`. The compiler can determine that the only checked exception that can arise in the `try` block is the `FooException`; however, any other `RuntimeException` (such as a `NullPointerException`) is also possible, and the `Exception` handler will catch these. So, the `Exception` handler is considered reachable in this case. From this you know that option B is incorrect.

As a side note, the code below would not compile. The compiler knows you cannot get a `SQLException`, so it would reject a `catch` block for it.

```
try {
  doSomethingBad();
  System.out.print("All Good");
} catch (FooException fe) {
} catch (BarException be) {
} catch (SQLException e) { // Compilation fails, this cannot happen!
```

Copy code snippet

Option C suggests that the `print` statement is unreachable. By tracing out the code's behavior you can determine that the method `doSomethingBad` always throws the `FooException`. Therefore, the `try` block will never be completed normally, and the `print` will never happen.

However, the compiler does not analyze the code in that level of detail. When compiling the method `doSomethingGood`, the call to the method `doSomethingBad` is considered only in terms of its signature. That signature suggests that the caller should be ready to catch a potential `FooException` but does not indicate that it's certain to arise. Consequently, the compiler believes the method `doSomethingBad` *might* run successfully and, therefore, considers the `print` statement to be reachable. From this, you can conclude option C is incorrect.

Now you know that the code will compile successfully and, thus, option D is correct.

Conclusion. The correct answer is option D.

Related quizzes

- Quiz yourself: Rules about throwing checked exceptions in Java
- Quiz yourself: Methods that throw an exception
- Quiz yourself: Custom exceptions

Mikalai Zaikin

Mikalai Zaikin is a lead Java developer at IBA IT Park in Minsk, Belarus. During his career, he has helped Oracle with development of Java certification exams, and he has been a technical reviewer of several Java certification books, including three editions of the famous *Sun Certified Programmer for Java* study guides by Kathy Sierra and Bert Bates.

Simon Roberts

Simon Roberts joined Sun Microsystems in time to teach Sun's first Java classes in the UK. He created the Sun Certified Java Programmer and Sun Certified Java Developer exams. He wrote several Java certification guides and is currently a freelance educator who publishes recorded and live video training through Pearson InformIT (available direct and through the O'Reilly Safari Books Online service). He remains involved with Oracle's Java certification projects.

Previous Post

Quiz yourself: Streams and flatMap operations in Java

Mikalai Zaikin | 3 min read

2021 © Oracle Site Map Privacy / Do Not Sell My Info Cookie Preferences Ad Choices Careers