Quiz Yourself: Custom Exceptions (Advanced)

QUIZ

Quiz Yourself: Custom Exceptions (Advanced)

When exactly do you need to declare an exception?

by Simon Roberts and Mikalai Zaikin

September 27, 2019

If you have worked on our quiz questions in the past, you know none of them is easy. They model the difficult questions from certification examinations. The levels marked “intermediate” and “advanced” refer to the exams, rather than the questions. Although in almost all cases, “advanced” questions will be harder. We write questions for the certification exams, and we intend that the same rules apply: Take words at their face value and trust that the questions are not intended to deceive you, but straightforwardly test your knowledge of the ins and outs of the language.

The objective of this question is to create custom exceptions and auto-closeable resources. Given the following code:

```
class UnknownException extends Error {}
class BusinessException1 extends Exception {}
class BusinessException2 extends Exception {}

class Application {
    public Object doId() /* point n1 */ {
        // line n2
    }
}
```

Which sets of additions, made independently, will compile successfully? Choose three sets.

A. At point n1: `throws BusinessException2`
   At line n2: `return "Hi";`
B. At point n1: Nothing added
   At line n2: `throw new UnknownException()`
C. At point n1: `throws Exception`
   At line n2: `throw new BusinessException1()`
D. At point n1: `throws BusinessException1`
   At line n2: `throw new BusinessException2()`
E. At point n1: `throws BusinessException1`
   At line n2: `throw new Exception()`

Answer. Java, but not the JVM (see sidebar), groups exceptions into two broad categories: checked and unchecked. The unchecked category has two further significant divisions, but those are not relevant to this question, so those aspects are not discussed here.

Although it’s a bit early in the discussion to have a side note, this is a can of worms, so let’s put it to bed. What does the parenthetical remark, “not the JVM,” mean? First, if you don’t care, you don’t need to know. But it’s perhaps an interesting piece of background knowledge. In a Java

program, a method that contains code that *might* throw a checked exception *must* declare the fact. And, most of Java's language rules (such as accessibility and assignment compatibility) are enforced on the bytecode by the class loader. However, the rule about checked exceptions is not. From the perspective of bytecode, all exceptions are equal, and you can throw anything you like anytime you like, without declaring it.

The class `java.lang.Throwable` is the base class of anything that can be thrown. This has two subclasses called `java.lang.Error` and `java.lang.Exception`. `Exception` has a particular subclass called `java.lang.RuntimeException`. Every `Throwable` in Java is a checked exception unless it is an `Error` or `RuntimeException`. That is, those two classes, and anything that's a subclass of them, are unchecked exceptions, while *everything* else is checked. Given that, you know that `UnknownException` is unchecked, because it's a subclass of `Error`, and that both `BusinessException1` and `BusinessException2` are checked.

Now, Java imposes no constraints on throwing unchecked exceptions. That is, you can declare them or not, regardless of whether the method might throw the exception or whether it's impossible for the exceptions to be thrown.

By contrast, a method must not have any possibility of throwing a checked exception to its caller unless that method declares the possibility with a `throws` clause. It's sufficient for the method's `throws` clause to mention a parent of the actual exception that might be thrown, because the child exception is an instance of the parent exception type.

Also, a `throws` clause that lists a particular exception simply makes it OK for that method to throw that exception; it doesn't obligate it to do so. Obviously, most exceptions are thrown under exceptional conditions, so they might not arise. But it's also OK for a method to have no possibility of throwing the exception. Declaring the exception might be done to allow future changes or to allow an overriding method to throw the exception.

So, let's examine the options.

Option A declares a `throws` clause that lists a checked exception, but the body of the method has no possibility of throwing the exception. As just mentioned, this is fine, so option A is correct.

Option B throws an `UnknownException`, which is actually a subclass of `Error` (and so is rather badly named) but, more importantly, it is unchecked. Because it's an unchecked type, it doesn't matter whether there is a `throws` clause mentioning this. As a result, option B is correct too.

Option C throws the specific, and checked, exception `BusinessException1`, but declares that it throws `Exception`. As discussed, because `BusinessException1` "is an" `Exception`, this is correct. In this case, however, unlike in options A and B, the `throws` clause cannot correctly be omitted. So, option C is also correct.

Option D throws the checked exception `BusinessException2`, but it declares that it throws `BusinessException1`. The `throws` clause is permissible, but it is inadequate for allowing the throwing of the sibling exception. The problem, simply enough, is that there's no relationship between the two `BusinessException` types other than they have a similar name and shared parentage. Therefore, in this case, the method throws a checked exception that's not declared by the `throws` clause, and option D is incorrect.

Option E fails for essentially the same reason as option D. Although every `BusinessException1` "is an" `Exception`, the inheritance relationship is unidirectional, and `Exception` is not a `BusinessException1`. Consequently, the `throws` clause is inadequate for the exception that might be thrown, and option E is incorrect.

The correct answer is: options A, B, and C.

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.

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.

Share this Page