Theory: Hierarchy of exceptions

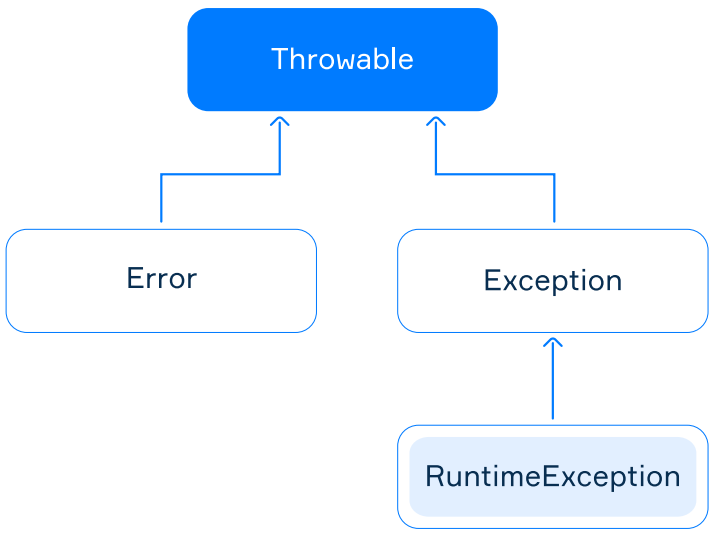
12 minutes 0 / 5 problems solved

§1. Exceptions and OOP

Java is primarily an object-oriented language. In such a paradigm, all exceptions are considered objects of special classes organized into a class hierarchy. Understanding this hierarchy is essential both for job interviews and daily programming practice.

§2. Hierarchy of exceptions

The following picture illustrates the simplified hierarchy of exceptions:



The base class for all exceptions is `java.lang.Throwable`. This class provides a set of common methods for all exceptions:

- `String getMessage()` returns the detailed string message of this exception object;
- `Throwable getCause()` returns the cause of this exception or `null` if the cause is nonexistent or unknown;
- `printStackTrace()` prints the stack trace on the standard error stream.

We will return to the methods and constructors of this class in the following topics.

The `Throwable` class has two direct subclasses: `java.lang.Error` and `java.lang.Exception`.

- subclasses of the `Error` class represents low-level exceptions in JVM, for example: `OutOfMemoryError`, `StackOverflowError`;
- subclasses of the `Exception` class deal with exceptional events inside applications, such as: `RuntimeException`, `IOException`;
- the `RuntimeException` class is rather a special subclass of `Exception`. It represents so-called **unchecked** exceptions, including: `ArithmeticException`, `NumberFormatException`, `NullPointerException`.

While developing an application, you normally will process objects of the `Exception` class and its subclasses. We won't discuss `Error` and its subclasses anymore.

> The four basic classes of exceptions (`Throwable`, `Exception`, `RuntimeException` and `Error`) are located in the `java.lang` package. They do not need to be imported. Yet their subclasses might be placed in different packages.

§3. Checked and unchecked exceptions

Current topic:
Hierarchy of exceptions Stage 7

Topic depends on:
✓ Inheritance Stage 7
✗ What is an exception Stage 7

Topic is required for:
Exception handling Stage 7
Throwing exceptions

Table of contents:
↑ Hierarchy of exceptions
§1. Exceptions and OOP
§2. Hierarchy of exceptions
§3. Checked and unchecked exceptions
Feedback & Comments

All exceptions can be divided into two groups: checked and unchecked. They are functionally equivalent but there is a difference from the compiler's point of view.

1. Checked exceptions are represented by the `Exception` class, excluding `RuntimeException` subclass. The compiler checks whether the programmer expects their occurrence in a program or not.

If a method throws a checked exception, this must be marked in the declaration using the special `throws` keyword. Otherwise, the program will not compile.

Let's take a look at the example. We use `Scanner` class, that you are already familiar with as a means to read from standard input, to read from a file:

```
1  public static String readLineFromFile() throws FileNotFoundException {
2
   Scanner scanner = new Scanner(new File("file.txt")); // java.io.FileNotFoundEx
ception
3      return scanner.nextLine();
4  }
```

Here, `FileNotFoundException` is a standard checked exception. This constructor of `Scanner` declares `FileNotFoundException` exception, because we assume that the specified file may not exist. Most importantly, there is a single line in the method that may throw an exception, so we put the `throws` keyword in the method declaration.

2. Unchecked exceptions are represented by the `RuntimeException` class and all its subclasses. The compiler does not check whether the programmer expects their occurrence in a program or not.

Here is a method that throws `NumberFormatException` when the input string has an invalid format (e.g. `"abc"`).

```
1  public static Long convertStringToLong(String str) {
2      return Long.parseLong(str); // It may throw NumberFormatException
3  }
```

This code always successfully compiles without the `throws` keyword in the declaration.

> Note, runtime exceptions may occur anywhere in a program. Adding them to each method's declaration would reduce the clarity of a program. Thus, the compiler doesn't require that you specify runtime exceptions in declarations.

The `Error` class and its subclasses are also considered as unchecked exceptions. However, they form a separate class.

Report a typo

458 users liked this theory. **11** didn't like it. **What about you?**

Start practicing

Comments (11) Hints (0) Useful links (3) Show discussion