Theory: Throwing exceptions

34 minutes

Verify to skip Start practicing

3508 users solved this topic. Latest completion was **about 17 hours ago**.

You've now learned what exceptions are and in what situations they occur. Now it's time to explore them further by understanding when and how you should throw exceptions yourself.

1 required topic

Hierarchy of exceptions

1 dependent topic

Custom exceptions

Table of contents:

§1. The throw keyword

Any object of the `Throwable` class and all its subclasses can be thrown using the **throw** statement. The general form of the statement consists of the `throw` keyword and an object to be thrown.

In the following example, we create and throw an object of the `RuntimeException` class that extends `Throwable`.

```
public class Main {
    public static void main(String args[]) {
        RuntimeException exception = new RuntimeException("Something's bad.");
        throw exception;
    }
}
```

Let's consider the code snippet above. First, we create an object with the specified message as the constructor argument. Then, we throw this exception using the `throw` keyword. Just creating an object is not enough to throw an exception.

The program stops and prints the error with the message we provided:

```
Exception in thread "main" java.lang.RuntimeException: Something's bad.
    at Main.main(Main.java:3)
```

The common practice is to create and throw an exception in a single line:

- throwing an instance of `Throwable`

```
throw new Throwable("Something's bad.");
```

- throwing an instance of `Exception`

```
throw new Exception("An exception occurs");
```

- throwing an instance of `NullPointerException`

```
throw new NullPointerException("The field is null");
```

It is only possible to throw an object of the `Throwable` class or a class that extends `Throwable`. For example, the line `throw new Long(10L);` does not compile.

§2. Throwing checked exceptions

For example, let's take a look at the following method that reads text from a file. In case the file is not found, the method throws an `IOException`:

```java
public static String readTextFromFile(String path) throws IOException {
    // find a file by the specified path

    if (!found) {
        throw new IOException("The file " + path + " is not found");
    }

    // read and return text from the file
}
```

Here we can only see a part of the method. The `throws` keyword following the method parameters is required since an `IOException` is a checked exception.

> If a method throws a checked exception, the type of exception must be specified in the method declaration after the `throws` keyword. Otherwise, the code won't compile.

If a method throws two or more checked exceptions, they must be separated by a comma (`,`) in the declaration:

```java
public static void method() throws ExceptionType1, ExceptionType2, ExceptionT
```

If a method is declared as throwing an exception (i.e. `BaseExceptionType`), it can also throw any subclass of the specified exception (i.e. `SubClassExceptionType`):

```java
public static void method() throws BaseExceptionType
```

§3. Throwing unchecked exceptions

Let's see how unchecked exceptions are thrown in a more real-life example. The `Account` class contains the method called `deposit`, that adds the specified amount to the current balance. If the amount is not positive or exceeds the limit, the method throws an `IllegalArgumentException`.

```java
class Account {

    private long balance = 0;

    public void deposit(long amount) {
        if (amount <= 0) {
            throw new IllegalArgumentException("Incorrect sum " + amount);
        }

        if (amount >= 100_000_000L) {
            throw new IllegalArgumentException("Too large amount");
        }

        balance += amount;
    }

    public long getBalance() {
        return balance;
    }
}
```

The `deposit` method is not declared as throwing an `IllegalArgumentException`. The same is true for all other unchecked exceptions.

> If a method throws an unchecked exception, the keyword `throws` is not required in the method declaration (but you still have to use `throw`!)

§4. When to throw an exception?

As you can see, technically, throwing an exception is a rather straightforward task. But the question is, when do you need to do this? The answer is that it is not always obvious.

The common practice is to throw an exception when and only when the method preconditions are broken, that is when it cannot be performed under the current conditions.

There are different cases where you would want to throw an exception. Imagine a method that parses the input string in the dd-MM-yyyy format to get a month. Here, if the string is invalid, the method throws an `InvalidArgumentException`. Another example is reading a non-existing file that will lead to a `FileNotFoundException`.

After some time of practice, identifying situations where you need an exception will become an easier task for you. It is recommended to throw exceptions that are most relevant (specific) to the problem: it is better to throw an object of `InvalidArgumentException` than the base `Exception` class.

Another question is how to choose between checked and unchecked exceptions? There is a short guideline. If a client can reasonably be expected to recover from an exception, make it a checked exception. If a client cannot do anything to recover, make it an unchecked exception.

In the next lessons, you will learn how to create your own classes of exceptions and use them inside an application.

§5. Conclusion

In this topic, you've learned how and when to throw exceptions. You can throw any object of the `Throwable` class and all its subclasses using the **throw** statement that consists of the `throw` keyword and an object to be thrown. An exception is usually thrown when and only when the method preconditions are broken, and it cannot be performed under the current conditions.

You've also learned the specifics of throwing checked and unchecked exceptions, and how to distinguish between them. In the next topics, you will learn how to create your own classes of exceptions and use them inside your app.

Report a typo

300 users liked this piece of theory. 4 didn't like it. **What about you?**

😍 🙂 😐 🙁 😡

Start practicing   Verify to skip

Comments (7)   Hints (0)   Useful links (0)   Show discussion