Theory: Throwing exceptions

15 minutes 0 / 5 problems solved Skip this topic Start practicing

1736 users solved this topic. Latest completion was about 10 hours ago.

You've now learned the concept of exceptions and some situations when they occur. Now it's time to explore the capability of exceptions further by understanding when and how you should throw exceptions yourself.

§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 snippet of code 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 impossible to throw an object of a class that does not extend `Throwable`. For example, the line `throw new Long(10L);` does not compile at all.

§2. Throwing checked exceptions

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

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 `IOException`:

Current topic:

Throwing exceptions

Topic depends on:

✕ Hierarchy of exceptions Stage 7

Topic is required for:

Dynamic array in Java

Doubly linked list in Java

Table of contents:

```
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 is only a part of the method. The keyword `throws` following the method parameters is required since `IOException` is a checked exception.

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

```
public static void method() throws ExceptionType1, ExceptionType2, ExceptionType3
```

- 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`):

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

§3. Throwing unchecked exceptions

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

Let's see how unchecked exceptions are thrown on 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 greater than the edge, the method throws `IllegalArgumentException` .

```
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 `IllegalArgumentException` . The same is true for all other unchecked exceptions.

§4. When to throw an exception?

As you can see, technically, throwing an exception is an easy 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 method preconditions are broken, that is when it cannot be performed under the current conditions.

Here are some examples when you would want to throw an exception:

- a method should read a file, but this file does not exist (`FileNotFoundException`);
- a method should parse the month from the input string, but the string is invalid (`InvalidArgumentException`).

After some time of practice, identifying situations where an exception is needed 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.

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