CODEGYM

Java Developer in 12 months

Mentor-supported training program from CodeGym

MODULE 1. JAVA SYNTAX

Lesson 21 Exceptions, part 2

CODEGYM

Lesson plan

- Creating your own exceptions
- Stack trace
- Try-with-resources statement

Creating your own exceptions

You can create an exception object yourself: it is just an object whose type is Exception or a class that inherits it. And throw it.

Code

```
class MyException extends Exception {
}


MyException myException = new MyException();
try {
   int a = 5 / 0;
} catch (Exception e) {
   System.out.println("Caught the exception");
   throw myException;
}
```

Console output

```
Caught an exception
```

In the example above, we created a new exception object whose type is MyException and immediately threw it using the throw operator.

Stack trace

A list that consists of the current method, and the method that invoked it, and method that called that one, etc. is called a stack trace

You can get it with this statement:

```
StackTraceElement[] methods = Thread.currentThread().getStackTrace();
```

You can also write it as two lines:

```
Thread current = Thread.currentThread();
StackTraceElement[] methods = current.getStackTrace();
```

CODEGYM

StackTraceElement

As its name suggests, the StackTraceElement class was created to store information about a stack trace element, i.e. one method in the stack trace.

This class has the following instance methods:

Method	Description
String getClassName()	Returns the name of the class
String getMethodName()	Returns the name of the method
String getFileName()	Returns the name of the file (one file can contain multiple classes)
int getLineNumber()	Returns the line number in the file where the method was called
String getModuleName()	Returns the name of the module (this can be null)
String getModuleVersion()	Returns the version of the module (this can be null)

CODEGYM

Stack

A stack is a data structure to which you can add elements and from which you can retrieve elements. In doing so, you can only take elements from the end: you first take the last one added, then the second to last one added, etc.

The name stack itself suggests this behavior, like how you would interact with a stack of papers. If you put sheets 1, 2 and 3 in a stack, you have to retrieve them in reverse order: first the third sheet, then the second, and only then the first.

Java even has a special Stack collection class with the same name and behavior.

This class has methods that implement stack behavior:

Methods	Description
T push(T obj)	Adds the obj element to the top of the stack
T pop()	Takes the element from the top of the stack (the stack depth decreases)
T peek()	Returns the item at the top of the stack (the stack does not change)
boolean empty()	Checks whether the collection is empty
int search(Object obj)	Searches for an object in the collection and returns its index

Displaying a stack trace during exception handling

Why is a list of method calls called a stack trace? Because if you think of the list of methods as a stack of sheets of paper with method names, then when you call the next method, you add a sheet with that method's name to the stack. And the next sheet of paper goes on top of that, and so on.

Another interesting use for stacks is during exception handling.
When an error occurs in a program and an exception is thrown, the exception contains the current stack trace — an array consisting of a list of methods starting, from the main method and ending with the method where the error occurred. There's even the line where the exception was thrown!

This stack trace is stored inside the exception and can be easily retrieved from it using the following method:

```
StackTraceElement[] getStackTrace()
```

By the way, the Throwable class has another method for working with stack traces, a method that displays all the stack trace information stored inside the exception. It is called printStackTrace(). **Quite conveniently, you can call it on any exception.**

External resources and the close() method

As a Java program runs, sometimes it interacts with entities outside the Java machine.

For example, with files on disk. These entities are usually called external resources. Internal resources are the objects created inside the Java machine.

Every time your Java program starts working with a file on disk, the Java machine asks the operating system for exclusive access to it. If the resource is free, then the Java machine acquires it.

But after you've finished working with the file, this resource (file) must be released, i.e. you need to notify the operating system that you no longer need it. If you do not do this, then the resource will continue to be held by your program.

Classes that use external resources have a special method for releasing them: **close()**.

It all seems simple. But exceptions can occur as a program runs, and the external resource won't be released. And that is very bad.

To ensure that the close() method is always called, we need to wrap our code in a **try-catch-finally** block and add the **close()** method to the finally block.

CODEGYM

try-with-resources

And here Java's creators decided to sprinkle some syntactic sugar on us. Starting with its 7th version, Java has a new try-with-resources statement.

It was created precisely to solve the problem with the mandatory call to the close() method.

```
try (Class name = new Class()) {
   // Code that works with the name variable
}
```

This is another variation of the try statement. You need to add parentheses after the try keyword, and then create objects with external resources inside the parentheses. For each object in the parentheses, the compiler adds a finally section and a call to the close() method.

CODEGYM

Several variables at the same time

"You may often encounter a situation when you need to open several files at the same time. Let's say you are copying a file, so you need two objects: the file from which you are copying data and the file to which you are copying data.

In this case, the try-with-resources statement lets you create not one but several objects in it. The code that creates the objects must be separated by semicolons.

```
try (ClassName name = new ClassName(); ClassName2 name2 = new ClassName2()) {
Code that works with the name and name2 variables
}
```

AutoCloseable interface

Java's creators came up with a special interface called AutoCloseable, which has only one method — close(), which has no parameters.

They also added the restriction that only objects of classes that implement AutoCloseable can be declared as resources in a try-with-resources statement.

As a result, such objects will always have a close() method with no parameters.

CODEGYM

Homework

MODULE 1. JAVA SYNTAX

Complete Level 22

MODULE 1. JAVA SYNTAX

CODEGYM

Answers to questions

MODULE 1. JAVA SYNTAX