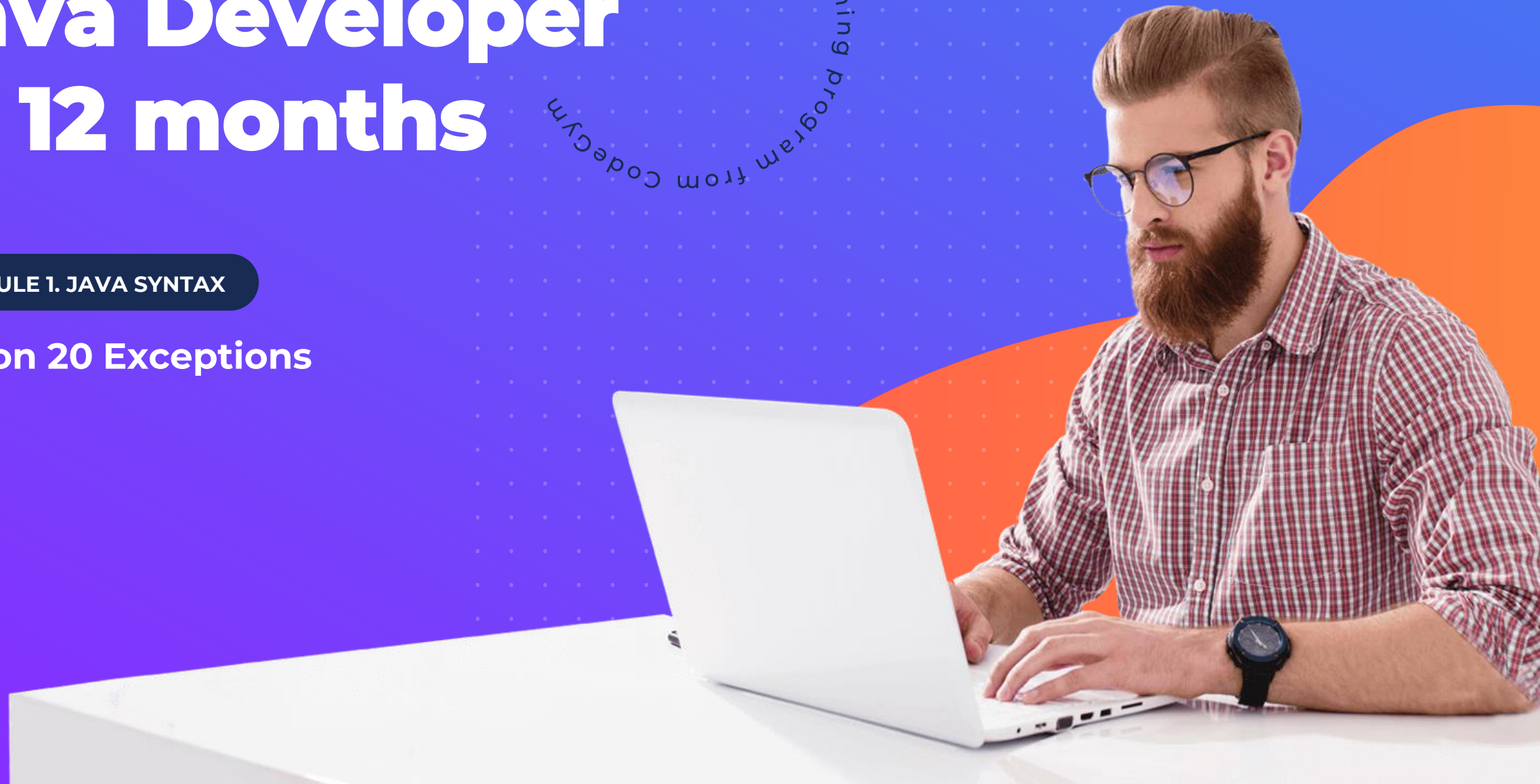
CODEGYM

Java Developer in 12 months

Mentor-supported training program from CodeGym

MODULE 1. JAVA SYNTAX

Lesson 20 Exceptions



CODEGYM

Lesson plan

- Exceptions
- Throwing exceptions
- finally keyword
- Types of exceptions
- Catching multiple exceptions

CODEGYM

A correctly working program

Beginners often saying something like "The program works, what else do you need?"

An experienced programmer knows that "works correctly" is only one of the requirements for a program, and it's not even the most important thing!

The most important thing is that the program code is understandable to other programmers. This is more important than a correctly working program.

CODEGYM

Abnormal situations

Abnormal situations may arise in the execution of any program.

For each abnormal situation, the programmer (the author of the program) must:

- anticipate it
- decide how exactly the program should handle it
- write a solution that is as close as possible to the desired one.

CODEGYM

Exceptions

When an error occurs in a Java program, such as division by 0, some wonderful things happen:

1. A special exception object is created, which contains information about the error that occurred.

2. The exception object is "thrown". Perhaps the wording here could be better. "Throwing an exception" is more like triggering a fire alarm or sounding a "DEFCON 1" alert.

When an exception is thrown to the Java machine, the normal operation of the program stops and "emergency protocols" begin.

CODEGYM

Catching exceptions: try-catch

Java has an exception catching mechanism that lets you halt this abnormal termination of methods.

```
try {
   // Code where an exception might occur
}
catch(ExceptionType name) {
   // Exception handling code
}
```

This construct is called a try-catch block.

CODEGYM

Multiple catch blocks

In theory, all sorts of exceptions can be thrown in a block of code. Some you will want to handle one way, others another way, and still others you will decide not to handle at all.

Java developers decided to help you out and let you write not one but many catch blocks after the try block.

```
try {
   // Code where an exception might occur
}
catch(ExceptionType1 name1) {
   // Code to handle ExceptionType1
}
catch(ExceptionType2 name2) {
   // Code to handle ExceptionType2
}
catch(ExceptionType3 name3) {
   // Code to handle ExceptionType3
}
```

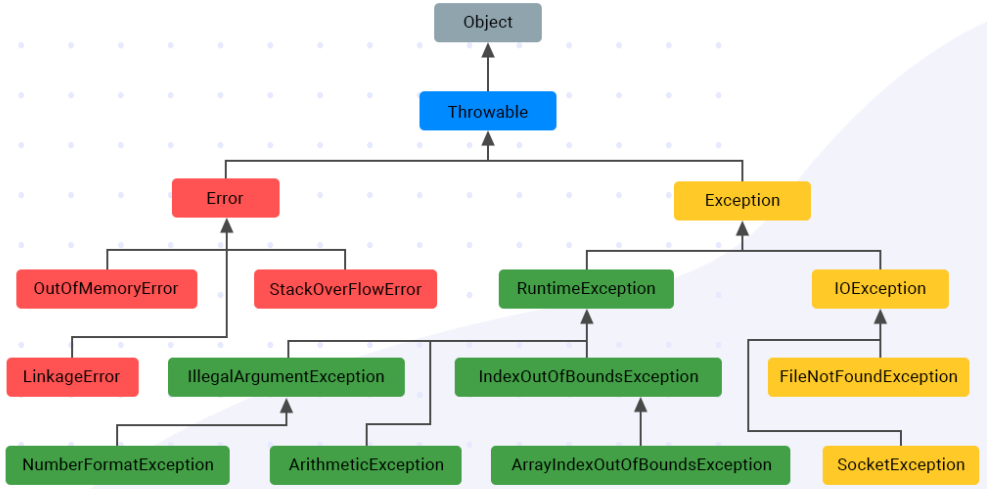
CODEGYM

Order of catch blocks
(exception hierarchy)

Exceptions that occur in a try block can only be caught by a single catch block. You cannot have an exception handling situation where the code from multiple catch blocks gets executed.

But the order of the blocks matters.
You can have a situation where an exception could be caught by multiple blocks. If that is the case, then the exception will be caught by whichever catch block comes first (closest to the try block).

How can you have a situation where multiple catch blocks can catch the same exception?
All exceptions belong to a single inheritance hierarchy.



The Throwable type is generally capable of catching every possible exception in Java. If you put it in the first catch block, then the code won't compile, since the compiler knows that there are unreachable blocks of code.

CODEGYM

Throwing exceptions

All the logic of how exceptions work is just a special way the Java machine behaves when an exception is thrown to it.

You can always rethrow a caught exception to the Java machine. To do this, you need to use the throw operator:

```
throw exception;
```

A rethrown exception cannot be caught by other catch blocks in the same try-catch block.

CODEGYM

Finally keyword

Another important point. Sometimes a programmer needs to do perform some action regardless of whether an exception occurred in the code. For example, suppose we opened a file for writing. The opened file must be closed by calling close().

To perform these mandatory actions, another kind of block (finally) was added to the try-catch construct, thereby creating the try-catch-finally construct.

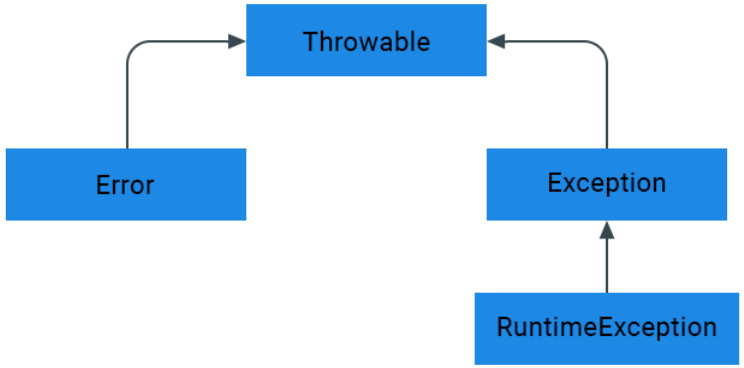
The code in the finally block will execute in any case, regardless of whether there was an exception. Even if an exception is thrown and not caught, the finally block will still execute.

By the way, if you do not want to catch an exception, but you do need a finally block, use the shorthand notation for the try-catch-finally construct: a try-finally block.

```
try {
// Code where an exception might occur
}
finally {
// Code that must executed no matter what happens
}
```

CODEGYM

Types of exceptions



Throwable class

The base class for all exceptions is the Throwable class.

The Throwable class contains the code that writes the current call stack (stack trace of the current method) to an array. We'll learn what a stack trace is a little later.

The throw operator can only accept an object that derives from the Throwable class.

Error class

The next exception class is the Error class, which directly inherits the Throwable class. The Java machine creates objects of the Error class (and its descendants) when serious problems have occurred. For example, a hardware malfunction, insufficient memory, etc.

Exception class

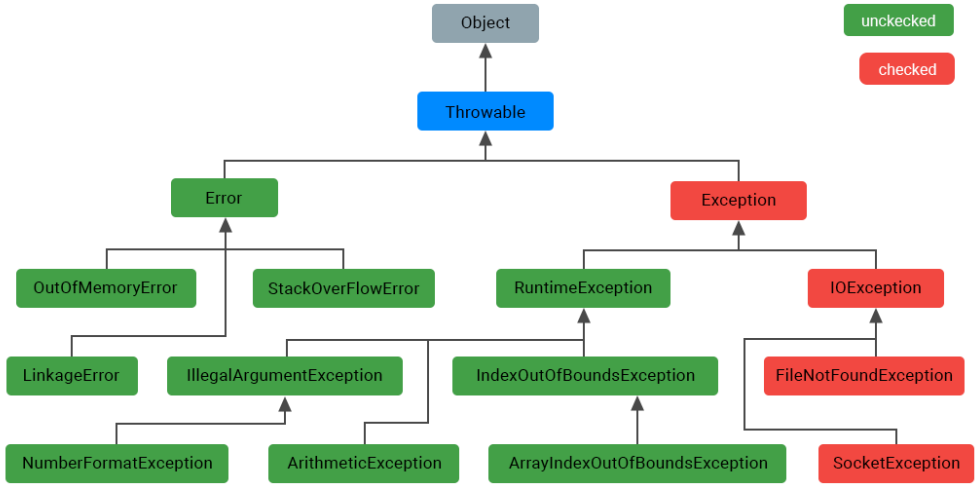
The Exception and RuntimeException classes are for common errors that happen in the operation of lots of methods.

The RuntimeExceptions

class are a subset of Exceptions. We could even say that RuntimeException is a lightweight version of ordinary exceptions (Exception) — fewer requirements and restrictions are imposed on such exceptions

CODEGYM

Checked and unchecked exceptions



All Java exceptions fall into 2 categories: checked and unchecked.

All exceptions that inherit the RuntimeException or Error are considered unchecked exceptions. All others are checked exceptions.

Twenty years after checked exceptions were introduced, almost every Java programmer thinks of this as a bug. In popular modern frameworks, 95% of all exceptions are unchecked.

The C# language, which almost copied Java exactly, did not add checked exceptions

CODEGYM

Checked exceptions, throws

There are additional requirements imposed on checked exceptions.

Requirement 1

If a method throws a checked exception, it must indicate the type of exception in its signature. That way, every method that calls it is aware that this "meaningful exception" might occur in it.

Indicate checked exceptions after the method parameters after the throws keyword (don't use the throw keyword by mistake). It looks something like this:

```
Type method (parameters) throws exception
```

If a method expects to throw multiple checked exceptions, all of them must be specified after the throws keyword, separated by commas. The order is not important.

Requirement 2

If you call a method that has checked exceptions in its signature, you cannot ignore the fact that it throws them. You must either catch all such exceptions by adding catch blocks for each one, or by adding them to a throws clause for your method.

CODEGYM

Disadvantages of checked exceptions

Checked exceptions seemed cool in theory but turned out to be a huge frustration in practice.

Suppose you have a super popular method in your project. It is called from hundreds of places in your program. And you decide to add a new checked exception to it.

And it may well be that this checked exception is really important and so special that only the main() method knows what to do if it is caught.

That means you'll have to add the checked exception to the throws clause of every method that calls your super popular method. As well as in the throws clause of all the methods that call those methods. And of the methods that call those methods.

CODEGYM

Catching multiple exceptions

Programmers really hate to duplicate code. They even came up with a corresponding development principle: Don't Repeat Yourself (DRY).

When handling exceptions, there are frequent occasions when a try block is followed by several catch blocks with the same code.

Or there could be 3 catch blocks with the same code and another 2 catch blocks with other identical code. This is a standard situation when your project handles exceptions responsibly.

Starting with version 7, in the Java language added the ability to specify multiple types of exceptions in a single catch block.

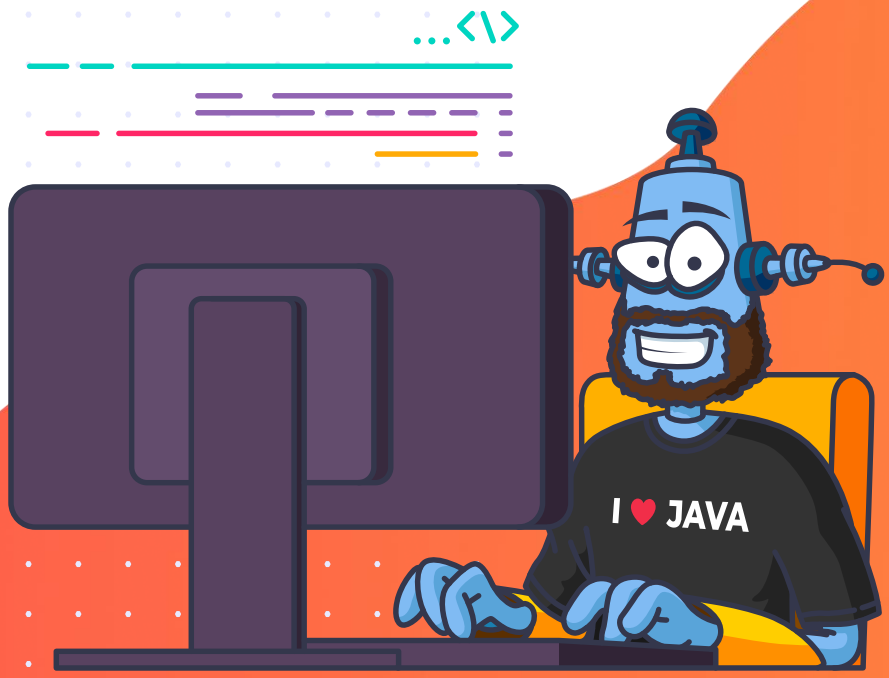
```
try {
   // Code where an exception might occur
}
catch(ExceptionType1 | ExceptionType2 | ExceptionType3 name) {
   // Exception handling code
}
```

CODEGYM

Homework

MODULE 1. JAVA SYNTAX

Complete Level 21



CODEGYM

Answers to questions

MODULE 1. JAVA SYNTAX