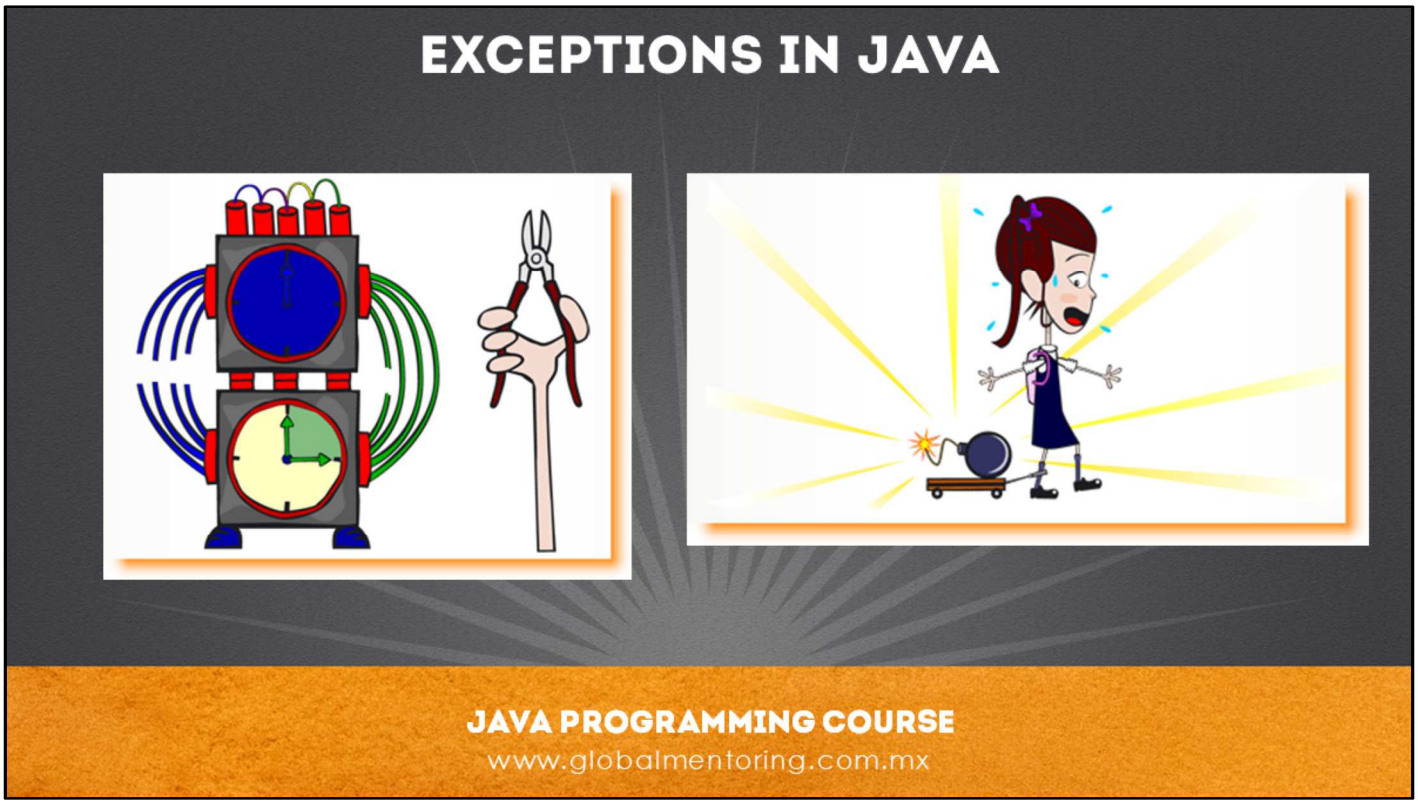
Hello, Ubaldo Acosta greets you again. I hope you're ready to start with this lesson.

We are going to study the subject of exceptions in Java.

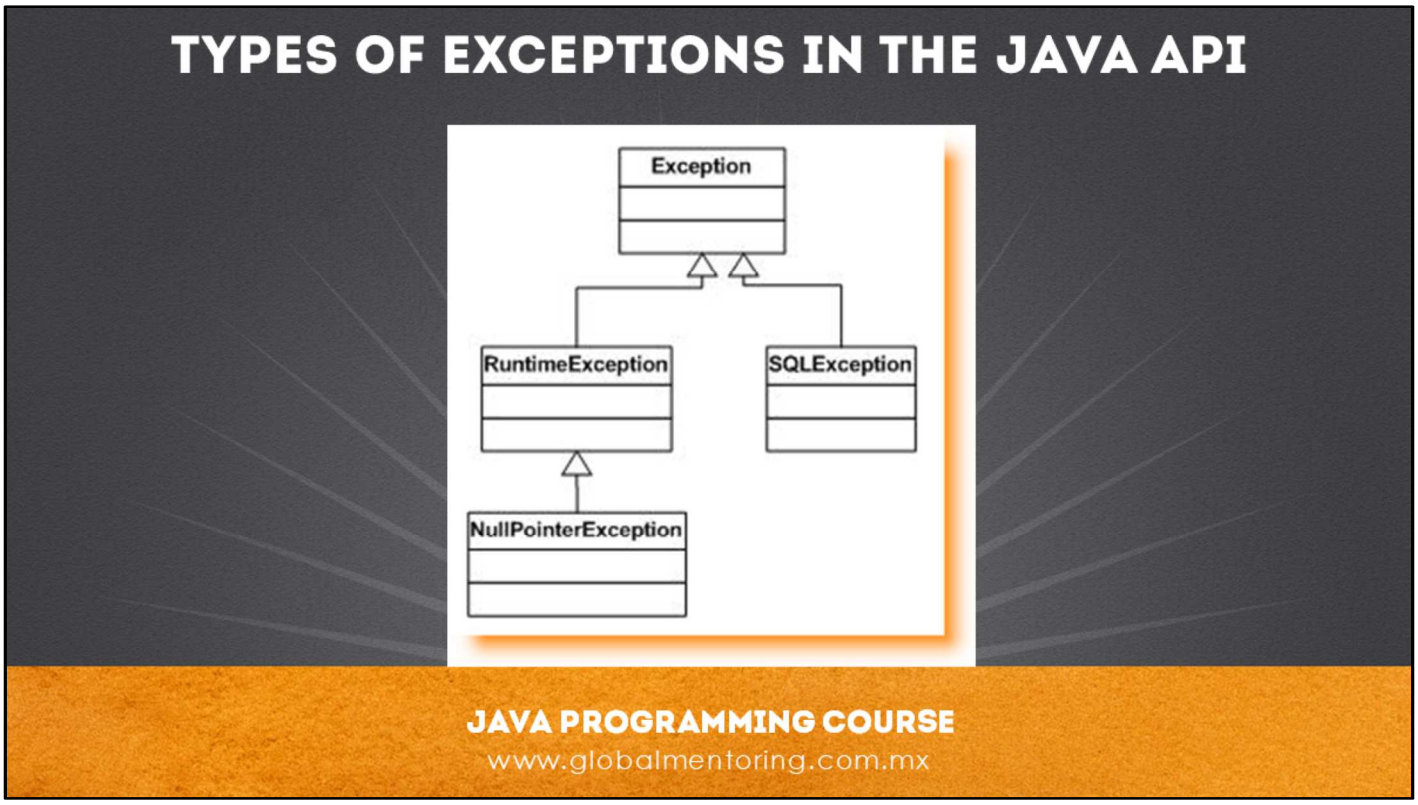
Are you ready? Come on!



In this lesson we are going to study the subject of exceptions. An exception is an unexpected situation in the execution of a program.

An example of an exception is, for example, if an argument is valid or not, if a data type conversion is incompatible, if there is a failure in the connection to the database, etc.

There are many kinds of Exceptions already created in the Java API to solve several of the mentioned problems, but if the class does not exist that suits our needs we can create our own exception classes, we will see next what types of exceptions They exist and how to use each of them.



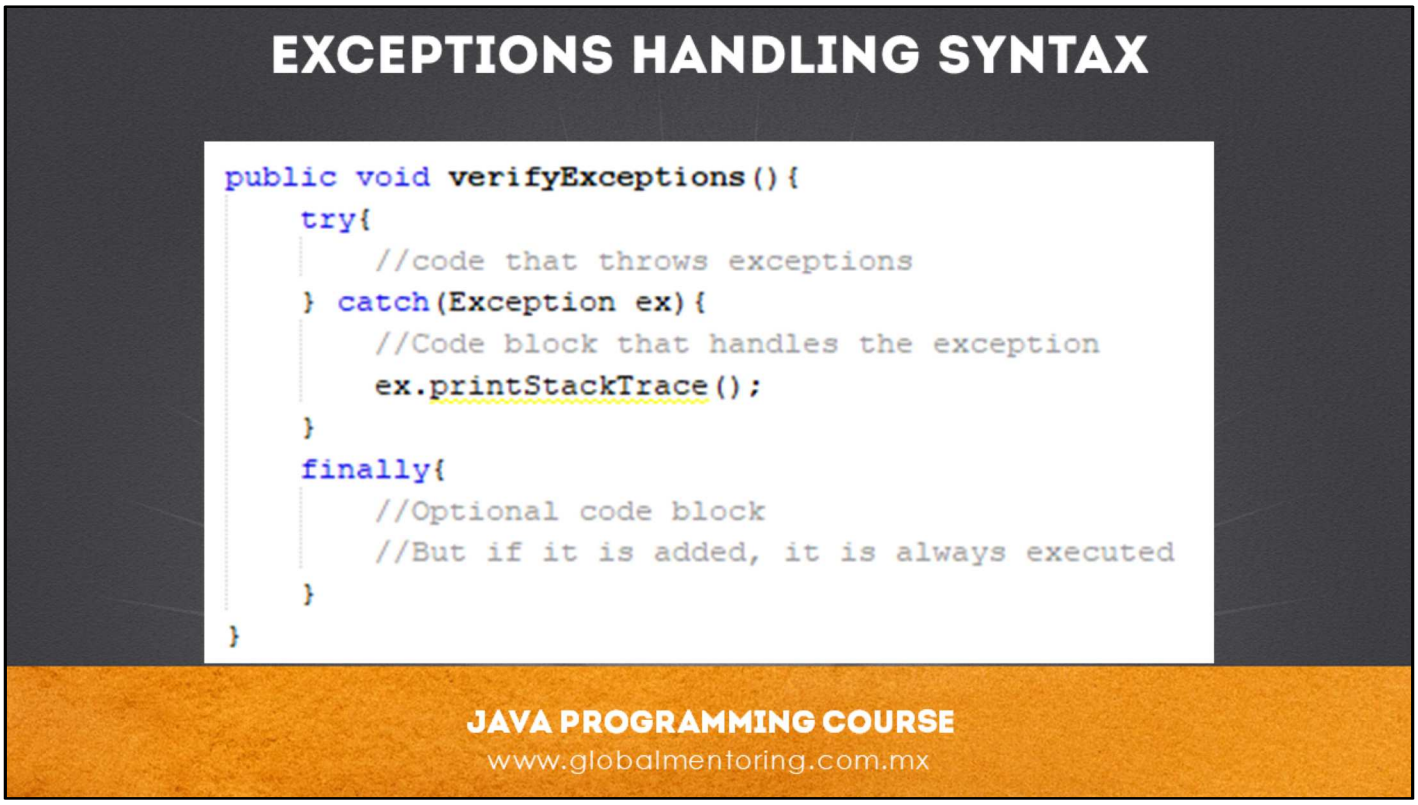
We are going to study the types of exceptions in the Java API. All Java classes are descended from the Throwable class and later there is a hierarchy of classes, however in general there are two types of exceptions with which we are going to work, which are known as checked exceptions and uncheked exceptions. Let's see what each one consists of:

A) Checked Exceptions or Exceptions that inherit from the Exception class: If our program throws this type of exception, the compiler will request to perform some activity with this type of exception. Later we will see how to process an exception of this type, for example to process them within a try / catch block or by throwing them into the method declaration. An example of this type of exception of type checked exception is the class SQLException, which descends from the class Exception and is thrown when there is a problem with issues related to the use and management of databases.

B) Uncheked Exceptions or Exceptions that inherit from the RuntimeException class: These types of exceptions are NOT required to process them, so it is optional to use the try / catch block or in the method declaration. These types of exceptions are also known as runtime exceptions. Examples of this type of exceptions are: NullPointerException, ArrayOutOfBoundsException, among many others.

There is a controversy regarding which of the two types of exceptions to use and the strategy to follow regarding the handling of exceptions in our programs. However, in recent years there has been a tendency regarding the use of exceptions of type uncheked exceptions, which does not oblige the programmer to process this type of exceptions and therefore it is optional to add some exception handling code in our methods, obtaining thus a cleaner code that allows choosing the team of programmers if they process the exceptions by adding some code for it or simply leave a cleaner code and will be responsible for processing the exception who is finally in need of such processing.

Examples of work teams that have opted for the use of unchecked exceptions is the working team of the Spring framework, which chose not only to use this type of exception, but when using the Spring framework many of the type exceptions checked, converted them to uncheked type, so it is optional to process, in most cases, the exceptions thrown by the Spring framework.



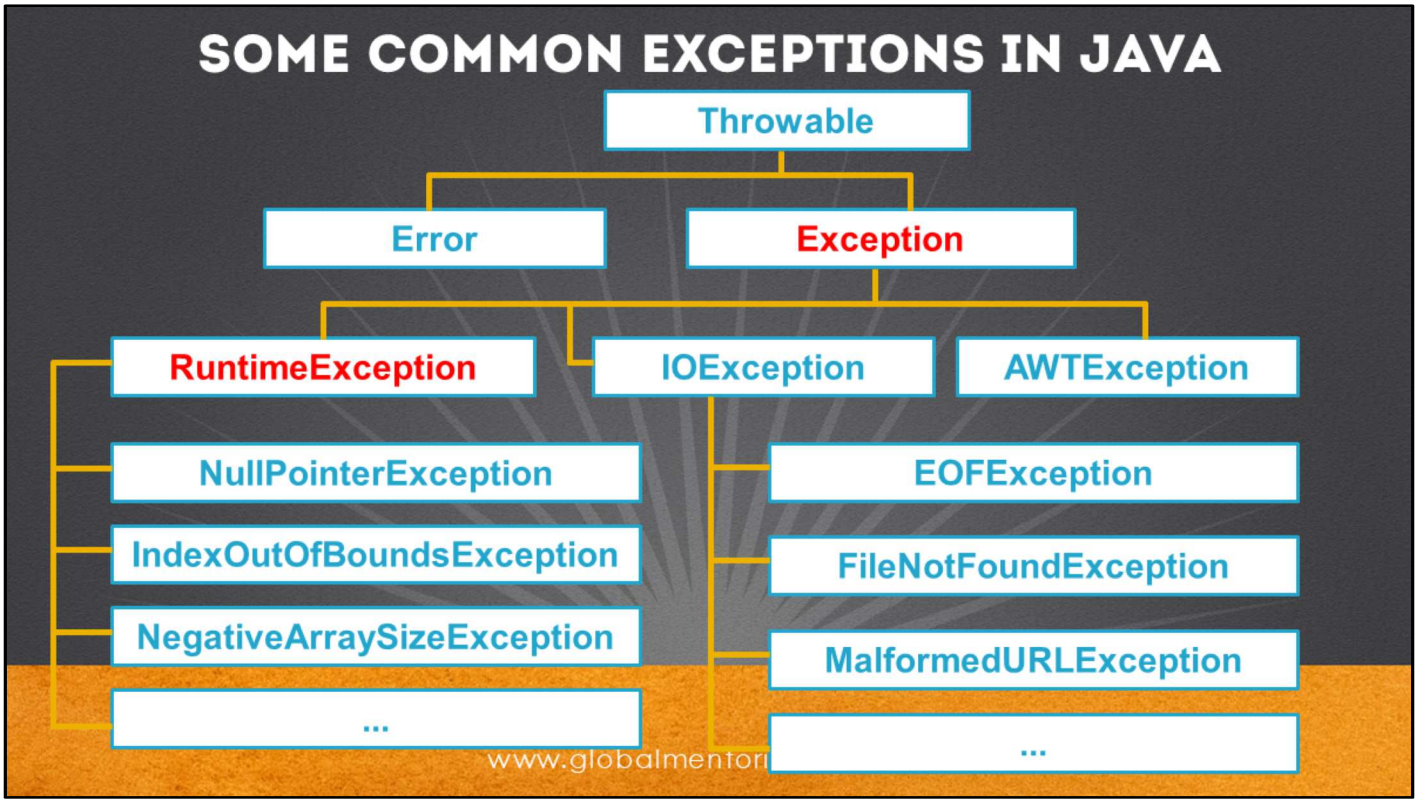
We will now review the general syntax for handling exceptions, for this we use the try / catch block.

The finally block is optional, but if it is set, it will always be executed, even if the exception does not occur, that is why sometimes we will use it to make sure that regardless of the problem that occurs, the code of the finally block is executed.

We can see that the code that possibly throws the exception must be wrapped by the try block. And if we want to process the exception, we can do it inside the catch block. This block is the one that receives the type of exception that we want to process, being able to be several catch blocks and for each block a different type of exception. It should be mentioned that the order to process more than one exception will be to process the exception that is the lowest hierarchy class and finally the highest hierarchy exception according to the class hierarchy that describes the exceptions that we are going to process.

An example of an exception, is to access an element of an out of range array, this would throw us an exception of type ArrayOutOfBoundException, so by means of the try / catch / finally block it is possible to process this type of exception.

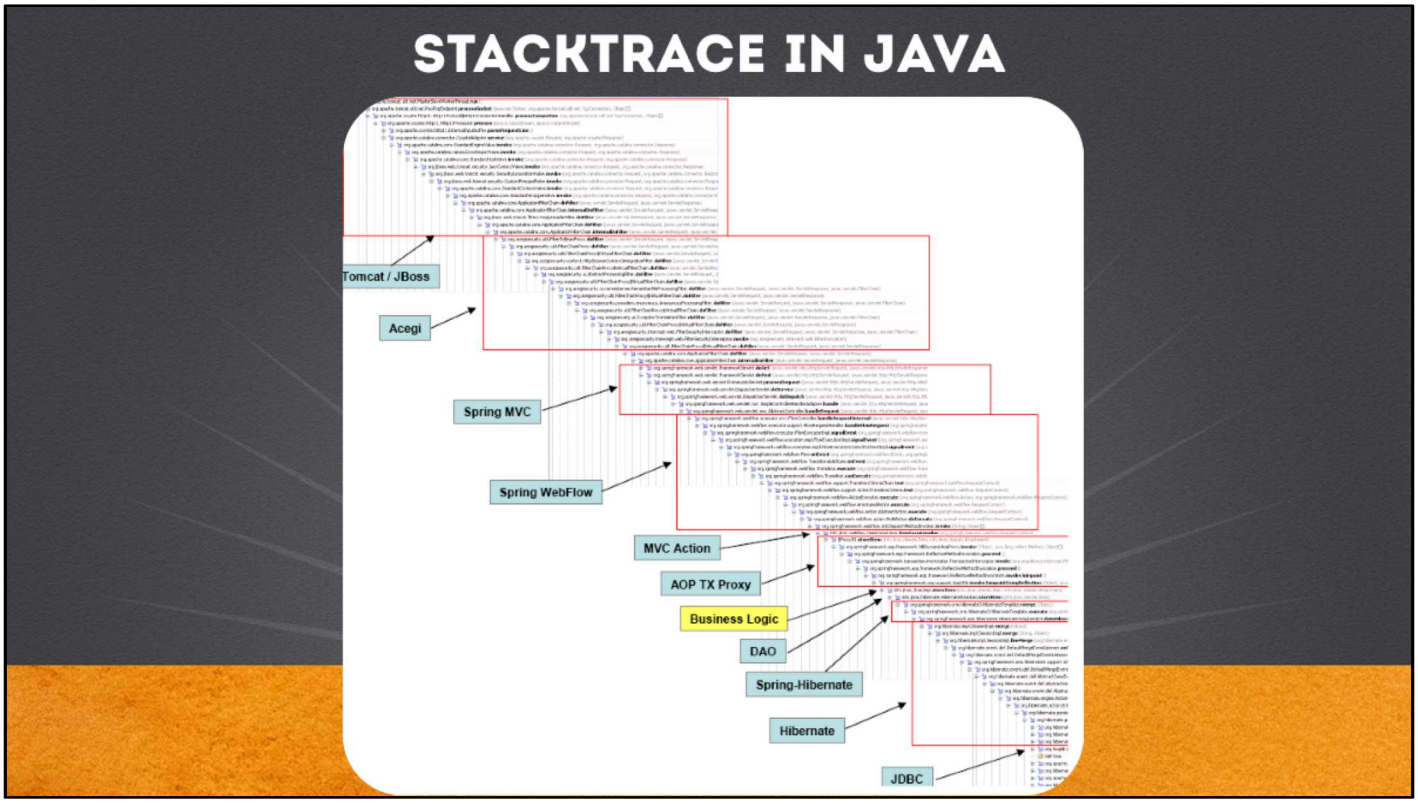
The try / catch block is optional to add for exceptions of type unchecked exceptions, that is, exceptions that descend from the Runtime Exception class. Throughout the exercises that we are creating we will be seeing several examples of both checked and unchecked exceptions, and so we will be familiar with each of these types of exceptions.



In the figure we can observe some of the most common exceptions in Java, and we can observe which are the parent classes of some of them. We can see that of the most important exceptions to be identified are the Exception class and the RuntimeException, since these are the types of exceptions that we have commented previously, the first one as exceptions of type checked exception and the exceptions of type unchecked exceptions.

In fact we can see that RuntimeException is a subclass of the Exception class, however the compiler will treat the exceptions in a different way to all the exceptions that descend from the RuntimeException class, in such a way that we must know and be clear about this division and how to work with each of the types of exception we have mentioned.

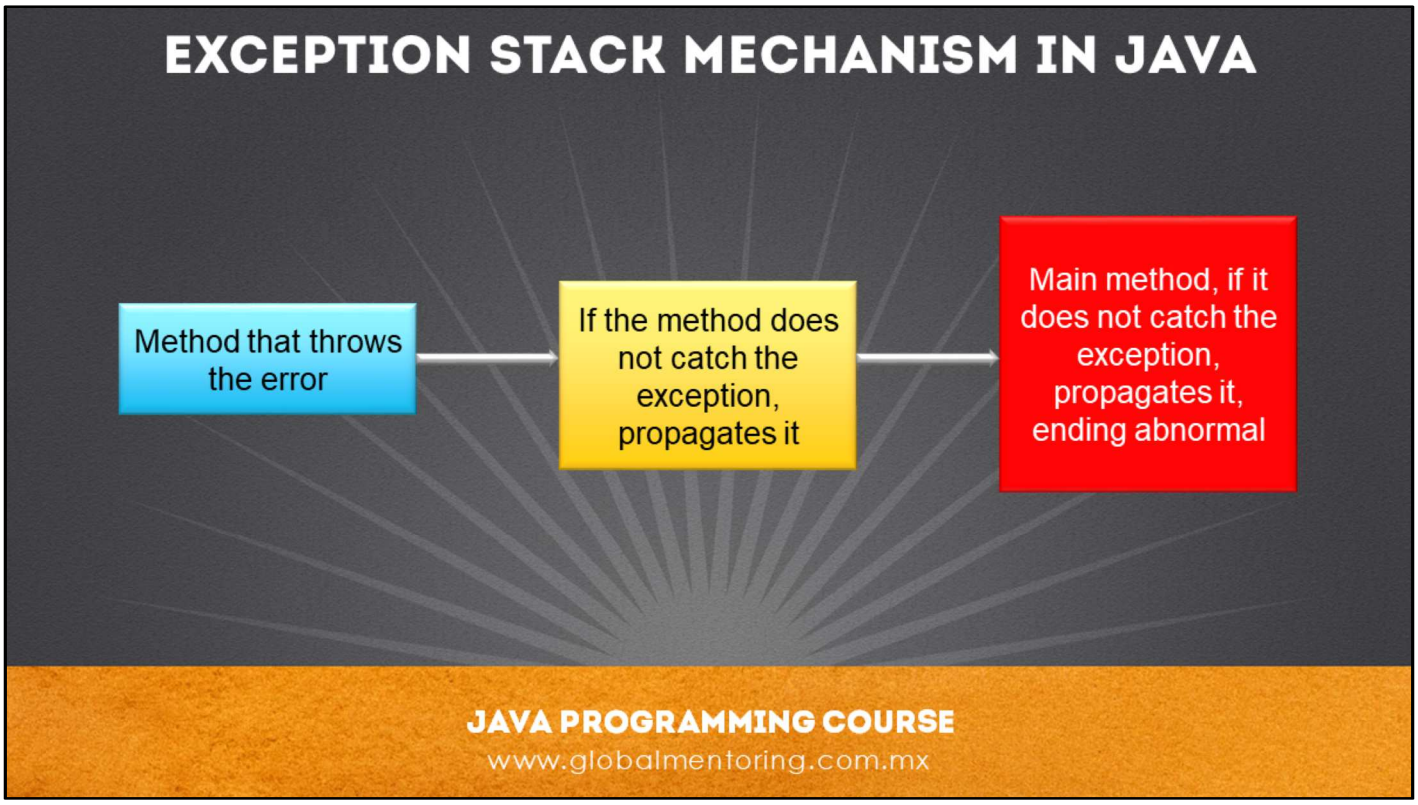
We see that the parent class of all exception classes is the Throwable class, and from this we will have the most important exceptions. The exceptions of type Error are those that are normally thrown by the Java virtual machine, so they will normally be errors that we can not recover, instead the exceptions of type Exception or RuntimeException will be caused by our code or the code of others that we are using in our program.



The StackTrace (trace of the error stack) of an exception is the set of error messages from the origin of the error to the last class that receives the error. In this way it is precisely as its name says, a tracking of the error of the exception and with it we can know more easily the origin of the error and therefore how to correct it.

The log or log of an application usually includes messages but also the stack trace of the errors that are happening in our application and therefore the log files, which are usually text files, will be very important to detect the root of the problems that happen in our application.

So we should be familiar with these terms because they are the ones we will use the most when we have problems in our Java applications, and we will see how to analyze the errors we face and thus start gaining experience to solve them.



We can observe in the figure in a more graphic way the mechanism in which the stack of exceptions or stacktrace works. As we have said, a stack of exceptions is one that accumulates the exceptions from the method that originated the problem, to the main method, or the last method that received the problem, if that exception is not caught before.

If an exception is not caught with a try / catch block, the exception is propagated to the method that calls it, and so on until some method catches it, or else the last method that receives it (for example the main method) throws finally the exception, ending the program abnormally.