How the Exception Handling Works in Java

In this article, we will learn how the exception handling works internally, how **JVM** will create an exception object and hand over to runtime system and how to an exception handler will handle the exceptions.

Let's first see what is an exception and it's definition.

What Is an Exception?

An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.

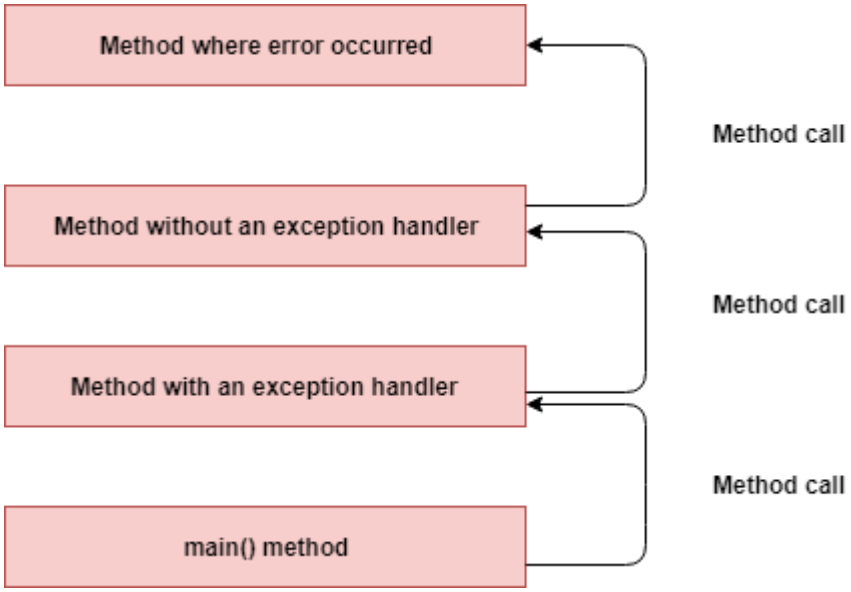
The term exception is shorthand for the phrase "exceptional event."

How the Exception Handling Works in Java

First, we will discuss conceptually how the exception handling works in Java. In the next section, we will demonstrate the same with programming example.

Step 1: When an error occurs within a method, the method creates an object and hands it off to the runtime system this object is called an *`exception object`*. The *`exception object`* contains information about the error, including its type and the state of the program when the error occurred. Creating an exception object and handing it to the runtime system is called throwing an *`exception`*.

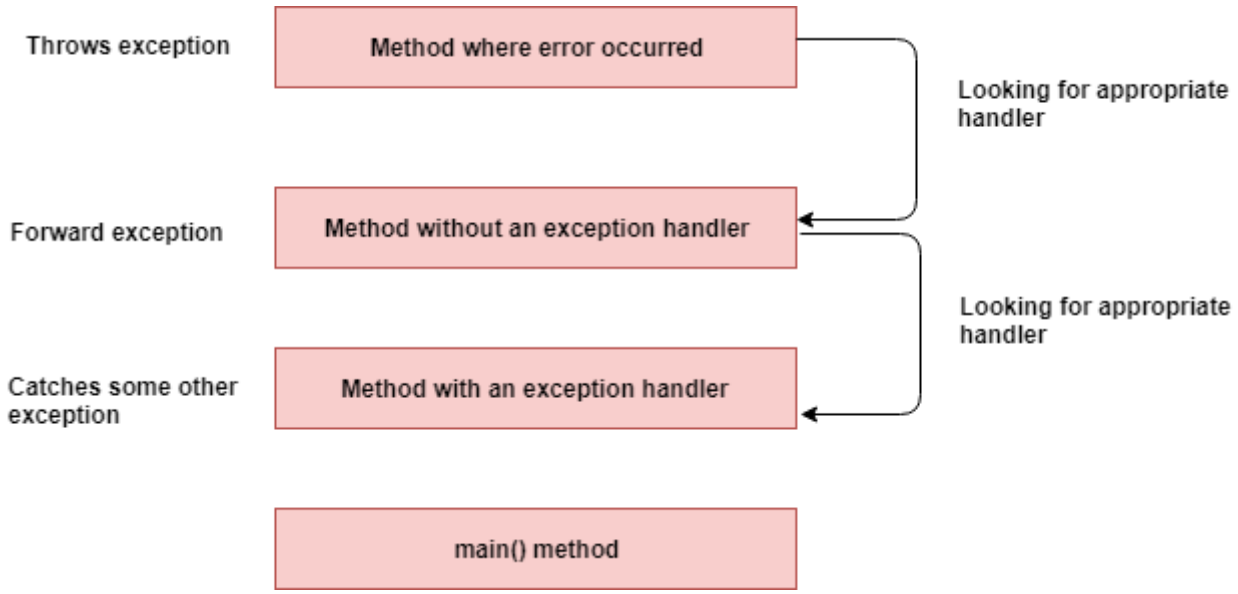
Step 2: After a method throws an exception, the runtime system attempts to find something to handle it. The set of possible "somethings" to handle the exception is the ordered list of methods that had been called to get to the method where the error occurred. The list of methods is known as the *`call stack`*. The following diagram shows the call stack of three method calls, where the first method called has the exception handler.



Step 3: The runtime system searches the call stack for a method that contains a block of code that can handle the exception. This block of code is called an *`exception handler`*. The search begins with the method in which the error occurred and proceeds through the call stack in the reverse order in which the methods were called. When an appropriate handler is found, the runtime system passes the exception to the handler.

An exception handler is considered appropriate if the type of the exception object thrown matches the type that can be handled by the handler.

Step 4: The exception handler chosen is said to catch the exception. If the runtime system exhaustively searches all the methods on the call stack without finding an appropriate exception handler, as shown in the following diagram, the runtime system (and, consequently, the program) terminates.



Let's demonstrate above how exception handling works in Java with a programmatic example.

Let's develop a program for reading a file example, Now we will create three methods which don't have exception handling code.

1. exceptionWithoutHandler()
2. exceptionWithoutHandler1()
3. exceptionWithoutHandler2()

Let's create an *exceptionHandler()* method which contains an exception handling code that is which has `catch` block to handle the exception.

1. Read file from some location, if a file does not exist in a given location this code throws `IOException` or `FileNotFoundException`.
2. The *exceptionWithoutHandler()* method don't have exception handler(catch block) to handle this exception.
3. The runtime system searches the call stack for a method that contains a block of code that can handle the exception. This block of code exists in *exceptionHandler()* method.

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

/**
 * This class demonstrate how Exception Handling Works
 * @author javaguides.net
 *
 */
public class ExceptionHandlingWorks {
```

```
    public static void main(String[] args) {
        exceptionHandler();
    }

    private static void exceptionWithoutHandler() throws IOException {
        try (BufferedReader reader = new BufferedReader(new FileReader(new
File("/invalid/file/location")))) {
            int c;
            // Read and display the file.
            while ((c = reader.read()) != -1) {
                System.out.println((char) c);
            }
        }
    }

    private static void exceptionWithoutHandler1() throws IOException {
        exceptionWithoutHandler();
    }

    private static void exceptionWithoutHandler2() throws IOException {
        exceptionWithoutHandler1();
    }

    private static void exceptionHandler() {
        try {
            exceptionWithoutHandler2();
        } catch (IOException e) {
            System.out.println("IOException caught!");
        }
    }
}
```

Output:

```
IOException caught!
```