Exceptions in Java

January 2, 2019

1 Throwable in Java

An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions. The Java virtual machine immediately aborts code execution. Exceptions can be caused by:

- serious runtime issues such as a stack overflow
- accessing attributes or calling methods of objects not pointing to a reference
- trying to write to a file with no permissions
- arithmetic exceptions - e.g. division by zero.
- programmer defined exceptions, e.g. non-conformable arguments

The process of stopping normal code execution is called *raising an exception or error*. It is important to note that all exceptions are objects. These objects must extend the THROWABLE class (which is a class and not an interface!). Java however already has two subclasses that are more convenient to extend. EXCEPTION and ERROR.

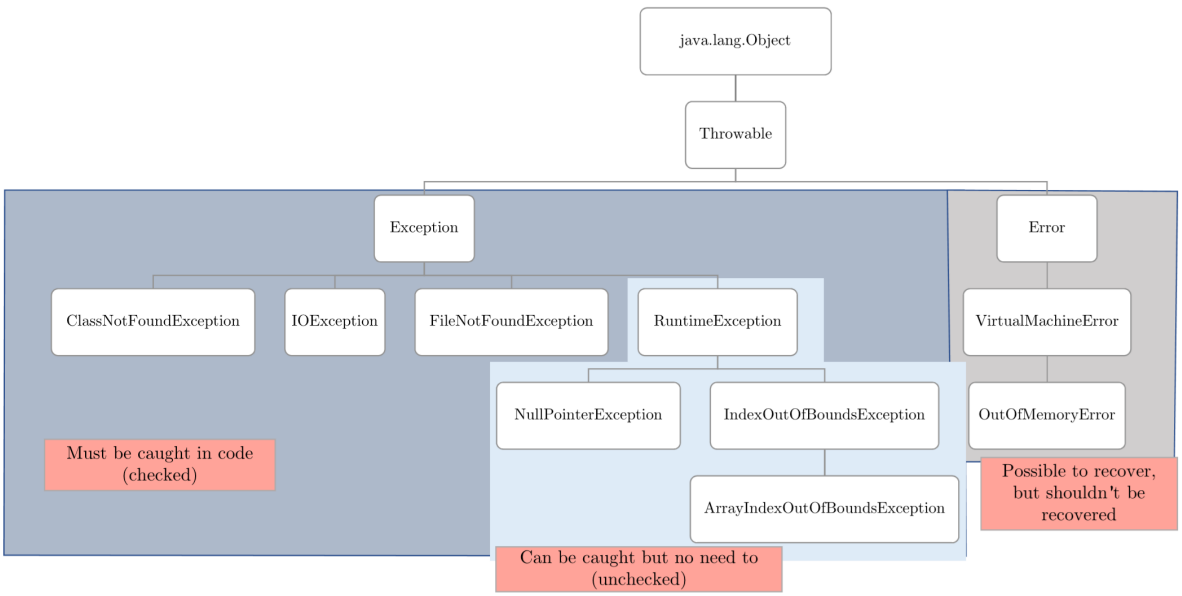

Figure 1: non complete diagram of common Java Error/Exception objects

Error Indicates serious problems that a reasonable application should not try to catch. Most such errors are abnormal conditions. But can theoretically be caught.

Exception The class Exception and its subclasses are a form of Throwable that indicates conditions that a reasonable application might want to catch. The class Exception and any subclasses that are not also subclasses of RuntimeException are checked exceptions.

RuntimeException RuntimeException is the superclass of those exceptions that can be thrown during the normal operation of the Java Virtual Machine. RuntimeException and its subclasses are unchecked exceptions. Unchecked exceptions do not need to be declared in a method or constructor's throws clause if they can be thrown by the execution of the method or constructor and propagate outside the method or constructor boundary.

2 Catching Exceptions

The following shows the basic structure of error handling in Java.

```
try{
    // do some risky things
    russian_roulette();
} catch(Exception ex){
    ex.printStackTrace();
    // handle the error
}finally{
    cleanup();
    // will always run. resources such as streams can be closed in here.
}
```

You can also catch multiple errors simultaneously:

```
catch(NullPointerException ex){
    ex.printStackTrace();
}
catch(RuntimeException ex){
    ex.printStackTrace();
}
catch(Exception ex){
    ex.printStackTrace();
}
catch(Throwable ex){
    ex.printStackTrace();
}
```

it is important to note that if we first caught the Throwable, we could not have caught the NullPointerException later on since the latter is a subclass of the first. Alternatively in Java 7 one could also do the following:

```
catch(NullPointerException | RuntimeException | Exception ex){
    // useless example but shows how it works
    ex.printStackTrace();
}
```

In the catch block the type of the exception can be determined using the typeof keyword and case distinction.

3 Throwing Exceptions / Errors

To throw an exception in a program simply write:

```
throw new Exception();
```

If the error is a subclass of RuntimeException no further action is needed since the error being thrown is an unchecked error. However, if it doesn't extend RuntimeException, the throw must be surrounded by a try catch block, or the method signature must specify that an error will be thrown.

```
void wontWork() throws Exception{
    throw new Exception();
}

void wontWork2(boolean b) throws NullPointerException, RuntimeException{
    if(b){
        throw new NullPointerException("oh no");
    } else {
        throw new RuntimeException("this is a message");
    }
}
```

You should note, that even though the second function will compile it is considered terrible style to throw more than one function.

4 Defining Custom Exceptions

This is actually very easy. One only needs to write a class extending a class that is already an exception or an error. This will give you all the behaviour of it's parent class. If you want a checked exception you usually extend Exception. For an unchecked one extends RuntimeException. Although one can extend Throwable or Error this is very bad style.

An example for an unchecked exception can be seen below:

```
class NotDuringOfficeHours extends RuntimeException
{
      // Parameterless Constructor
      public NotDuringOfficeHours() {}

      // Constructor that accepts a message
      public NotDuringOfficeHours(String message)
      {
         super(message);
      }
 }
```