Exception Handling

CS209A

Background Knowledge

This class assumes some basic familiarity of exceptions from first year CS

For a full treatment of exceptions please read the section 3.7 in Text B:

- http://math.hws.edu/eck/cs124/javanotes8/c3/s7.html

Exceptions

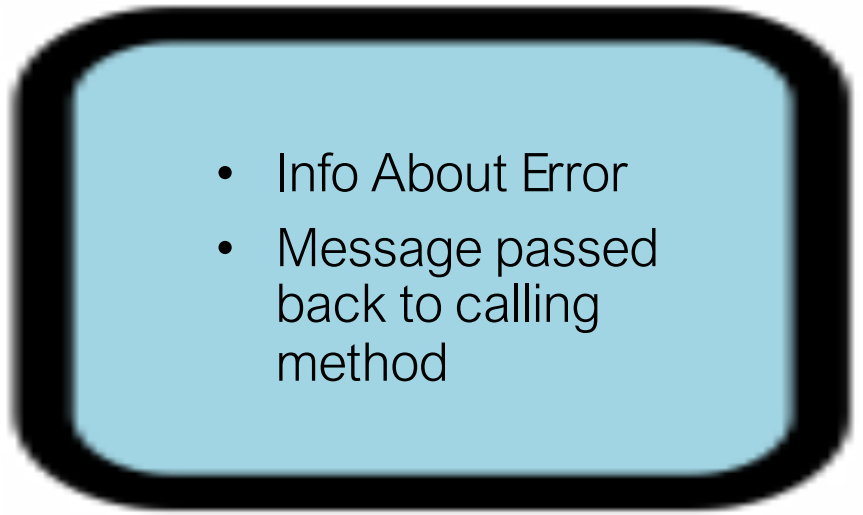
```
try {
    line = console.readLine();

    if (line.length() == 0) {
        throw new EmptyLineException("The line read from console was empty!");
    }

    console.printLine("Hello %s!" % line);
    console.printLine("The program ran successfully.");
}
catch (EmptyLineException e) {
    console.printLine("Hello!");
}
catch (Exception e) {
    console.printLine("Error: " + e.message());
}
finally {
    console.printLine("The program is now terminating.");
}
```

Earlier we looked at how the JVM works (please review if you haven't), Now we will look at the concept of "Exceptions".

In Java, exceptions are a common way to change the normal path of execution of a program; they first appeared in the 1960s in a language called LISP (designed for artificial intelligence applications). In Java exceptions are often used to control the flow of instruction execution to handle different cases. In C++ and Lisp, on the other hand, they are used for abnormal and unpredictable errors.

Exceptions are Special Objects

An exception in Java is a special object, generated by an "exceptional event" (understand, most often, an error) that is passed back to the calling method through a mechanism different from the usual messaging between objects.



Throw: Trigger an Exception

You throw an object, not a type

```
throw new MyException();
```

As an exception is an object, is must be instantiated (created from the template that a class defines as a type) using new when you need it and throw it.

Types of Errors

Compile Time

- Syntax Errors
- Wrong Type

Many things can go wrong, so we have different types of errors. Javac, as it compiles .java files to .class java bytecode files, will tell you about wrong syntax, or incompatible types when assigning data or calling methods. You have to solve all these issues before you can think about running your program.

Types of Errors

Compile Time

Link Time

Before you can run your program, the Class Loader must load it and you may have passed compilation and failed linking, because the Class Loader fails to find a .class corresponding to objects you want to use.

Types of Errors

Compile Time

Link Time

Run Time

- Detected by the application
- Detected by the library
- Detected by the Operating System / Hardware

When you run your program, you may run into other errors; one of your methods can detect that it gets parameters that are the right type but the wrong range of values. A built-in Java method may discover the same (and will throw an exception); or things may go really wrong and the operating system may discover it (hardware failure, for instance). Logic can also be wrong.

Types of Errors

Compile Time

Link Time

Run Time

Logic

Your programs logic can also be incorrect.

Java Syntax and Runtime

As some exceptions occur quite often, some methods warn about it by using **throws** followed by the name(s) of the exceptions it can throw. The javac compiler is then made aware of them.

Methods that throw exceptions say so …

```
Class FileReader

  Public FileReader (String filename) throws FileNotFoundException

      …
```

… So javac knows about it.

Java Syntax and Runtime

```
import java.io.File;

import java.io.FileReader;

public class IgnoredException {

      public static void main(String args[]) {

            FileReader fr = new FileReader("sample.txt");

      }

}
```

If you happily ignore the exceptions advertised by a method such as the previous constructor ...

Java Syntax and Runtime

```
$ javac IgnoredException.java
IgnoredException.java:7: error: unreported exception FileNotFoundException; must be caught or declared to be thrown

FileReader fr = new FileReader("sample.txt");
                ^

1 error

$
```

... javac will simply not let you do it. The message is pretty explicit: it wants you to either deal with the problem, or let the world know that something may fail and that another object has to deal with it.

So there are 2 Options

Deal with it (try … catch …)

You must either include the method call that can fail into a try ... catch block and deal with the exception in the catch block.

So there are 2 Options

Warn callers (throws …)

Or you may get away with handling the exception if you warn callers that what you are doing for them may fail, and that in that case *they* will have to deal with the problem.

Checked Exception

Handled

OR

Declared as thrown

When you are faced with this choice you are dealing with what is known as a checked exception, and javac will make sure that you follow the rules.

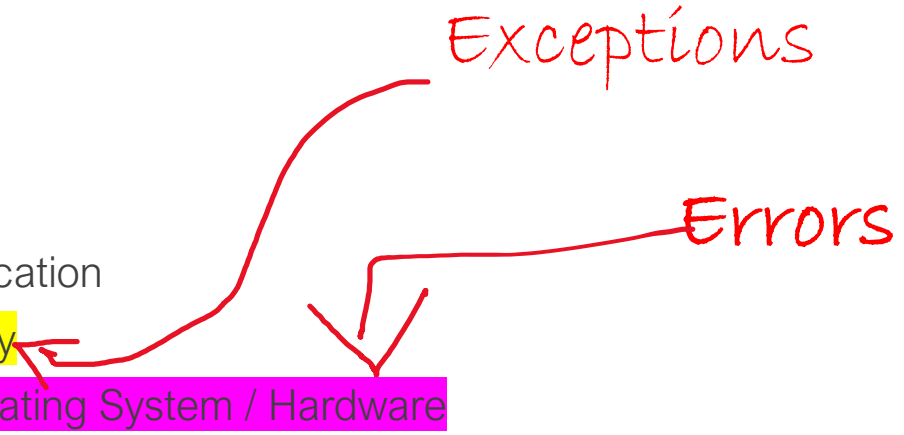
But something else can happen:

Compile Time

Link Time

Run Time

- Detected by the application
- Detected by the library
- Detected by the Operating System / Hardware



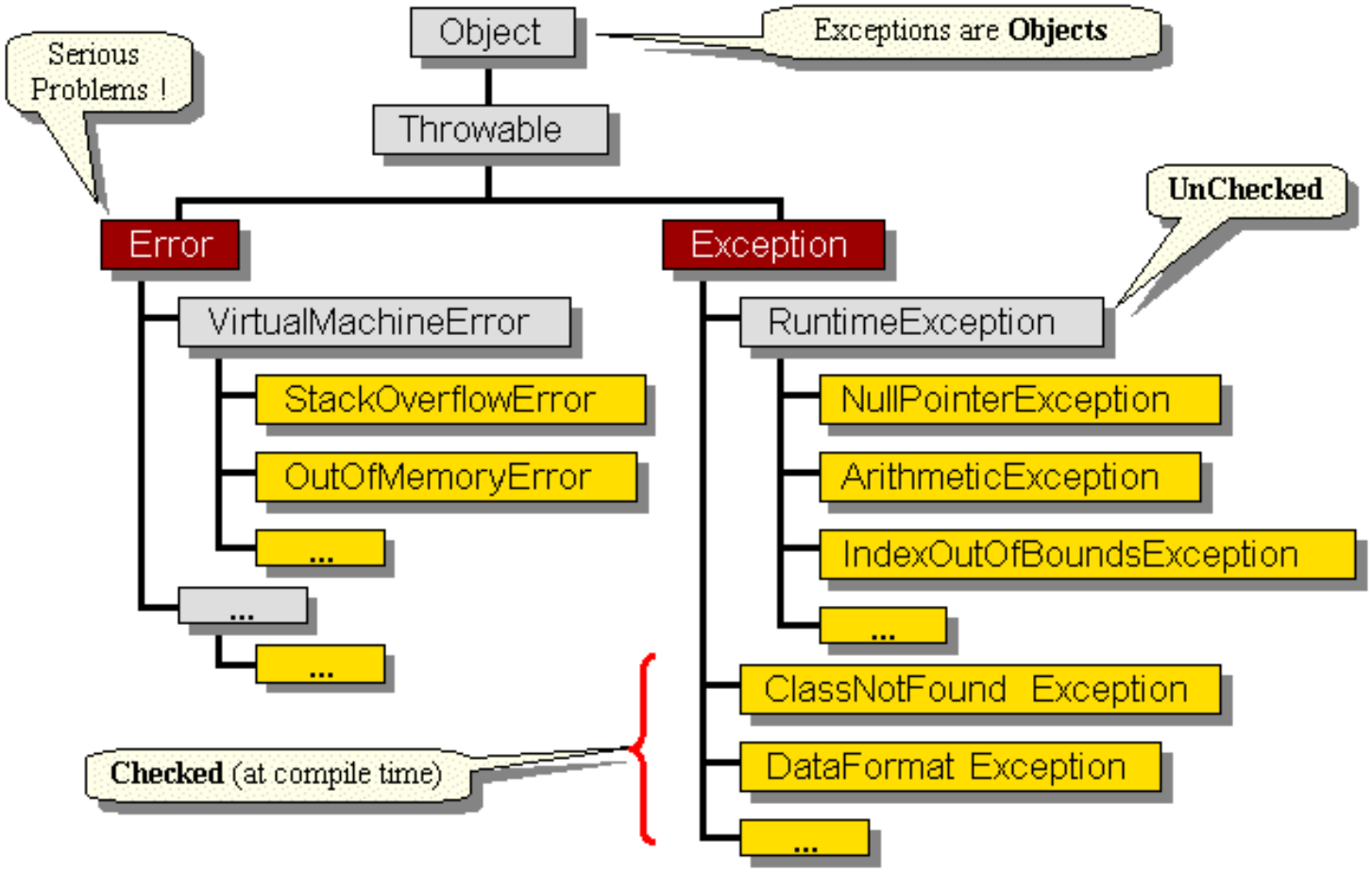
For example when you read a file, it may exist but be corrupted. **java.io.IOError**

Exceptions VS Errors

Exceptions are abnormal conditions that occur in your program or java telling you something has gone wrong. When an exception occurs java forces us to handle it (need to know when and what to do) or declare we want someone else to handle it.

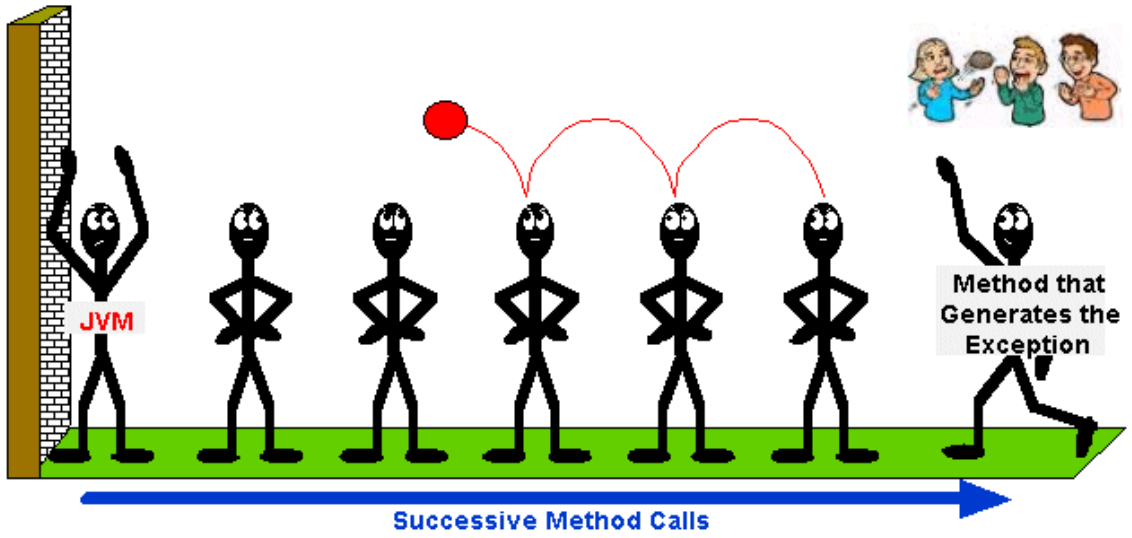
Exception handling provides a way to handle and recover from errors or a way to quit the program in a graceful way.

Errors represent serious represent serious unrecoverable problems for example VirtualMachineError.OutOfMemoryError



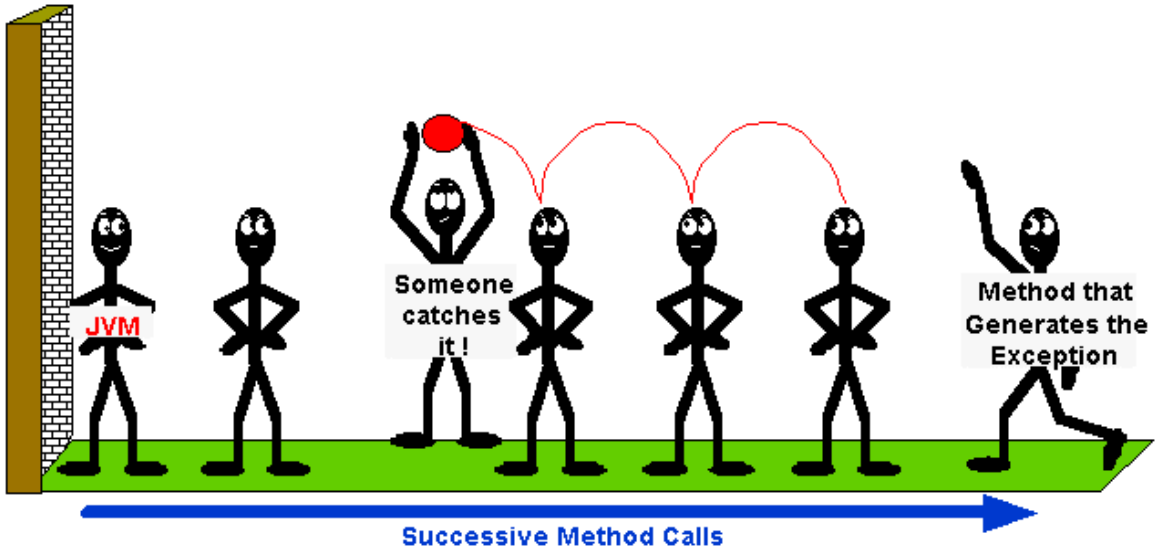
Exceptions VS Errors

Throwing Exceptions




```
…

public void getInformation()
throws MyException {

…
```

Throwing Exceptions




```
…

public void getInformation() throws MyException {

…

}

private void doStuff(){

try {

  getInformation();

catch (MyException ex){

  //handle here

}
```