CSC 7437

J Paul Gibson

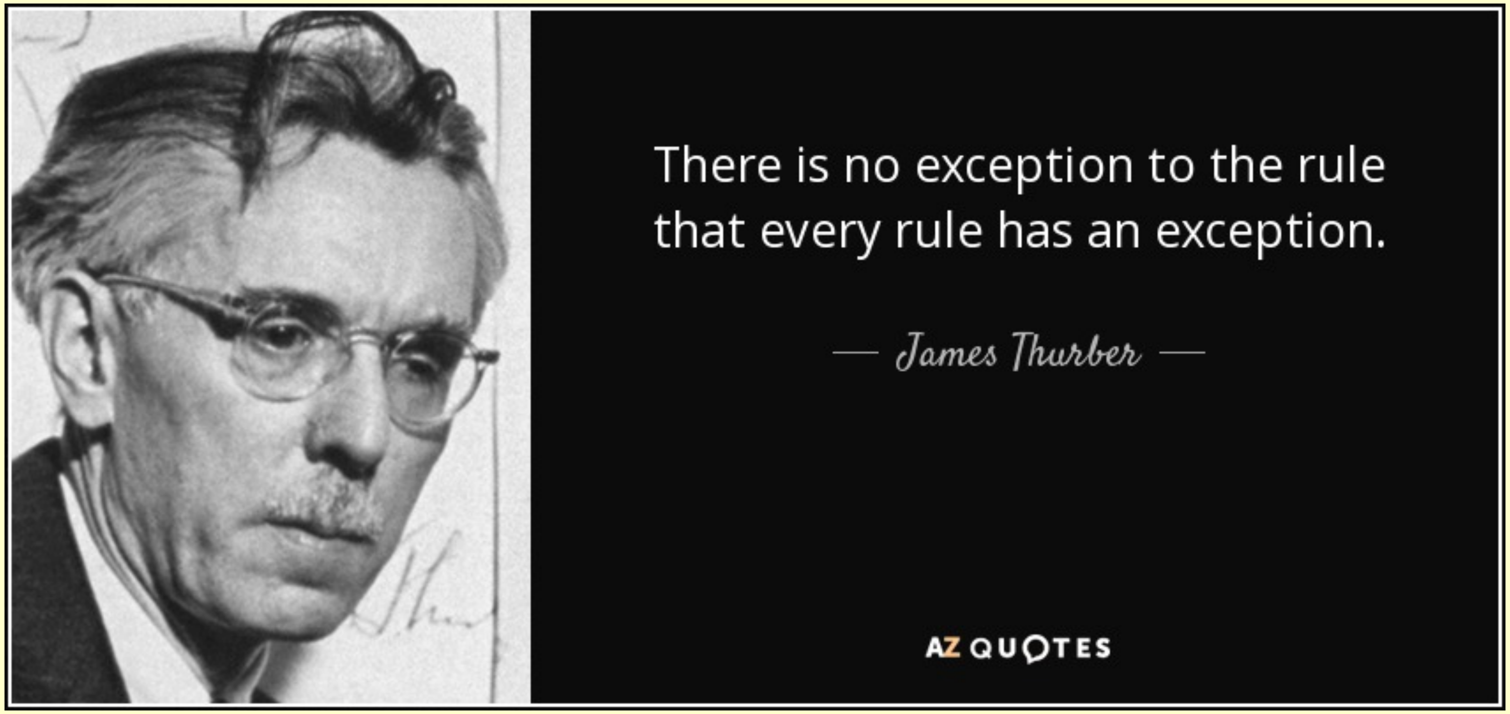
paul.gibson@telecom-sudparis.eu

Exceptions(in Java)

http://jpaulgibson.synology.me/~jpaulgibson/TSP/Teaching/CSC7437/

Exceptions

https://www.azquotes.com/quote/294495


There is no exception to the rule that every rule has an exception.

— James Thurber —

AZ QUOTES

Except this one?

“*When certain concepts of TeX are introduced informally, general rules will be stated; afterwards you will find that the rules aren't strictly true. In general, the later chapters contain more reliable information than the earlier ones do. The author feels that this technique of deliberate lying will actually make it easier for you to learn the ideas. Once you understand a simple but false rule, it will not be hard to supplement that rule with its exceptions.*”, Donald Knuth

Exceptions – General History

Software exception handling developed in Lisp in the 60s, where exceptions were caught by the ERRSET keyword, which returned NIL in case of an error - *LISP 1.5 programmer's manual*, **McCarthy, John, MIT press, 1965**

The control of exceptional conditions in PL/I object programs
JM Noble - Proc. IFIP Congress, 1968.

Exception Handling: Issues and a Proposed Notation
John B. Goodenough Commun. ACM, 1975

Software reliability: The role of programmed exception handling, **Melliar-Smith, P. M. and Randell, B, SIGSOFT Softw. Eng. Notes, 1977**

Exception Handling in CLU, **Liskov, B.H.; Snyder, A.; Software Engineering, IEEE Transactions, 1979**

A modular verifiable exception handling mechanism, **Shaula Yemini and Daniel M. Berry. 1985.. ACM Trans. Program. Lang. Syst**

2. Exceptions – Further Reading (C++ and Java)

Exception Handling for C++, **A. R. Koenig and B. Stroustrup: Journal of Object Oriented Programming, 1990**

Analyzing exception flow in Java programs. **Martin P. Robillard and Gail C. Murphy. In Proceedings of the 7th European software engineering conference held jointly with the 7th ACM SIGSOFT international symposium on Foundations of software engineering. 1999**

Analysis and testing of programs with exception handling constructs , **Sinha, S.; Harrold, M.J.; Software Engineering, IEEE Transactions Sep 2000**

A comparative study of exception handling mechanisms for building dependable object-oriented software, **Alessandro F. Garcia, Cecilia M. F. Rubira, Alexander Romanovsky, Jie Xu, Journal of Systems and Software, Volume 59, Issue 2, 15 November 2001**

Exceptions in Java

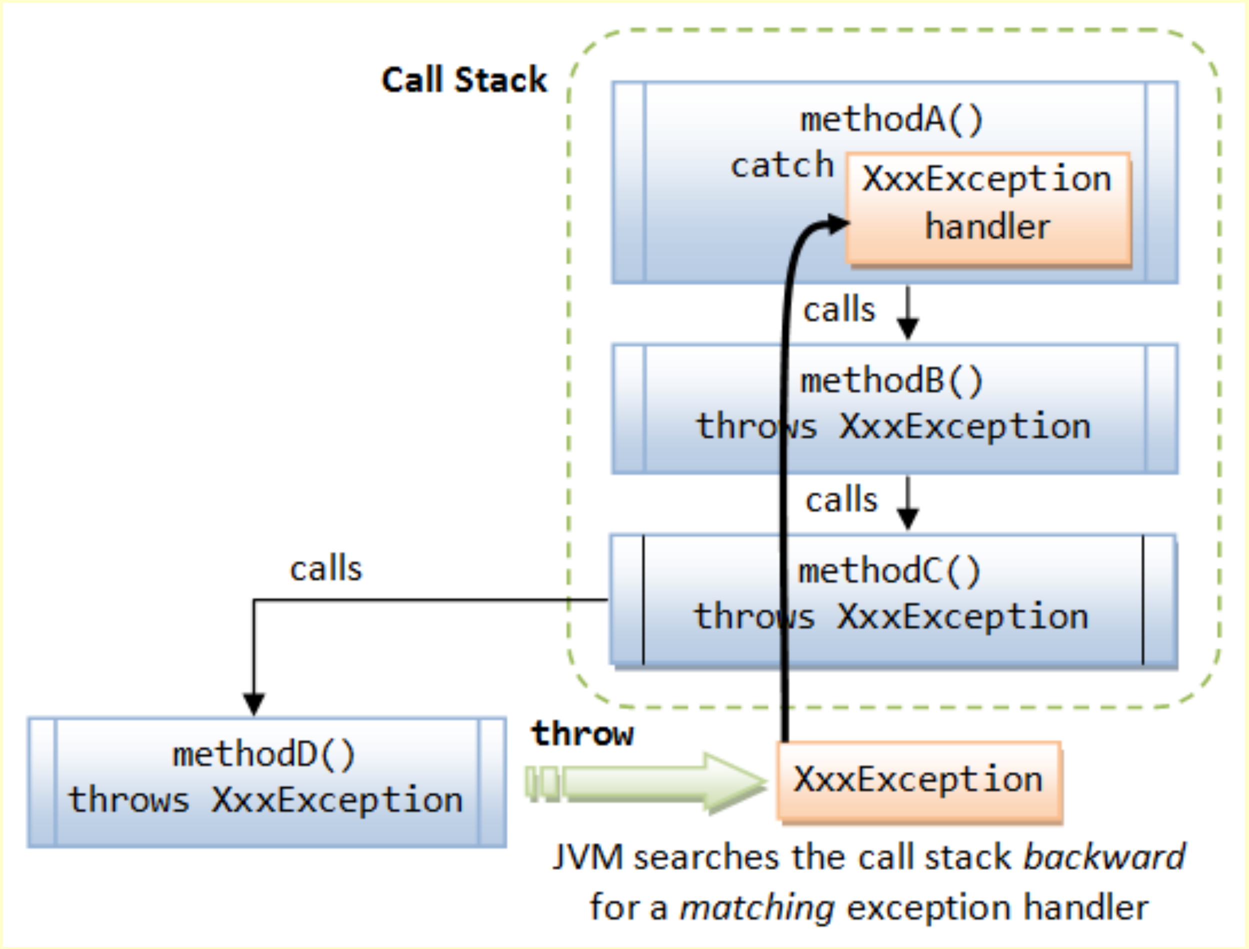
When a method encounters an abnormal condition (an *exception condition*) that it can't handle itself, it may *throw* an exception.

Exceptions are *caught* by handlers positioned along the thread's method invocation stack. If the calling method isn't prepared to catch the exception, it throws the exception up to *its* calling method, and so on.

When you program in Java, you must position catchers (the exception handlers) strategically, so your program will catch and handle all exceptions from which you want your program to recover.

NOTE: If one of the threads of your program throws an exception that isn't caught by any method along the method invocation stack, that thread will expire. (We will come back to this when we look at threads)

Exceptions in Java


Call Stack
methodA()
catch
XxxException handler
calls
methodB() throws XxxException
calls
methodC() throws XxxException
calls
methodD() throws XxxException
throw
XxxException
JVM searches the call stack *backward* for a *matching* exception handler

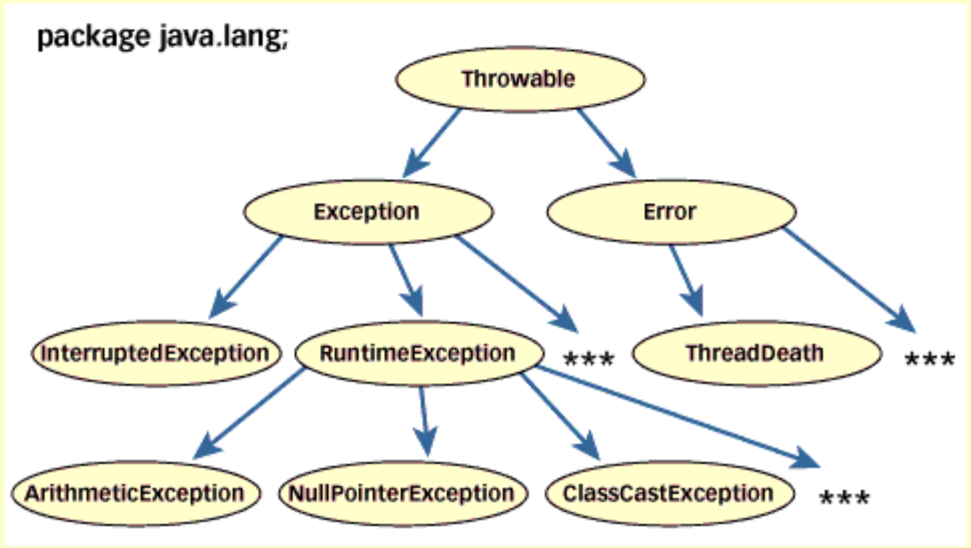
http://www.ntu.edu.sg/home/ehchua/programming/java/J5a_ExceptionAssert.html

Exceptions in Java

In Java, exceptions are objects. When you throw an exception, you throw an object.

You can't throw just any object as an exception, however -- only those objects whose classes descend from Throwable.

Throwable serves as the base class for an entire family of classes, declared in java.lang, that your program can instantiate and throw.



QUESTION: Have you seen Errors?

Exceptions (and errors) in Java

Exceptions (members of the Exception family) are thrown to signal abnormal conditions that can often be handled by some catcher, though it's possible they may not be caught and therefore could result in a dead thread.

Errors (members of the Error family) are usually thrown for more serious problems, such as OutOfMemoryError, that may not be so easy to handle.

In general, code you write should throw only exceptions, not errors. Errors are usually thrown by the methods of the Java API, or by the Java virtual machine itself.

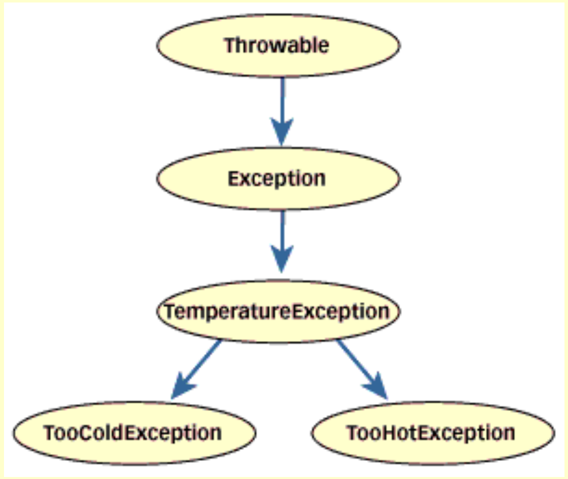
In addition to throwing objects whose classes are declared in java.lang, you can throw objects of your own design. To create your own class of throwable objects, you need only declare it as a subclass of some member of the Throwable family. In general, however, the throwable classes you define should extend class Exception.

Exceptions in Java

Whether you use an existing exception class from java.lang or create one of your own depends upon the situation. In some cases, a class from java.lang will do just fine.

For example, if one of your methods is invoked with an invalid argument, you could throw IllegalArgumentException, a subclass of RuntimeException in java.lang.

Sometimes you will want to indicate that a method encountered an abnormal condition that isn't represented by a class in the Throwable family of java.lang.
For example, in a coffee machine:



NOTE: Exceptional conditions are not necessarily rare, just outside the normal flow of events.

Exceptions in Java: Example Coffee Cup

```
public void drinkCoffee(CoffeeCup cup) throws TooColdException,
TooHotException {

 int temperature = cup.getTemperature();

 if (temperature <= TOOCOLD) throw new TooColdException();
 else if (temperature >= TOOHOT) throw new TooHotException();
 else cup.sip();

}


…


try {
customer.drinkCoffee(cup); System.out.println("Coffee is just right.");
}
catch (TooColdException e) { System.out.println("Coffee is too cold.");
catch (TooHotException e) { System.out.println("Coffee is too hot."); }
 }
```

Exceptions in Java: Example Coffee Cup

You can also group catches:

```
try {
customer.drinkCoffee(cup);
System.out.println("Coffee is just right.");
}
catch (TemperatureException e) {
System.out.println("Coffee is too cold or too hot.");
}
```

```
try {
customer.drinkCoffee(cup);
System.out.println("Coffee is just right.");
}
catch (TooHotException | TooColdException e) {
System.out.println("Coffee is too cold or too hot.");
}
```

Exceptions in Java: Example Coffee Cup

You can also group catches:

```
try {
customer.drinkCoffee(cup);
System.out.println("Coffee is just right.");
}
catch (TemperatureException e) {
System.out.println("Coffee is too cold or too hot.");
}
```

QUESTION: What about throwing exceptions inside the catch?
Does Java allow this? What are the semantics/rules?

TO DO: Write some experimental code to find the answers to these questions.

Exceptions in Java: Embedding information in an exception object

When you throw an exception, you are performing a kind of structured go-to from the place in your program where an abnormal condition was detected to a place where it can be handled.

The Java virtual machine uses the class of the exception object you throw to decide which catch clause, if any, should be allowed to handle the exception.

But an exception doesn't just transfer control from one part of your program to another, it also transmits information. Because the exception is a full-fledged object that you can define yourself, you can embed information about the abnormal condition in the object before you throw it. The catch clause can then get the information by querying the exception object directly.

Exceptions in Java: Example Coffee Cup

```
class UnusualTasteException extends Exception
{ UnusualTasteException() { }
UnusualTasteException(String msg) { super(msg);}
}

new UnusualTasteException("This coffee tastes like tea.")


try {
      cust.drinkCoffee(cup);
      System.out.println("Coffee ok.");
}
catch (UnusualTasteException e) {
System.out.println( "Customer is complaining of unusual taste.");
String s = e.getMessage();
if (s != null) System.out.println(s);
}
```

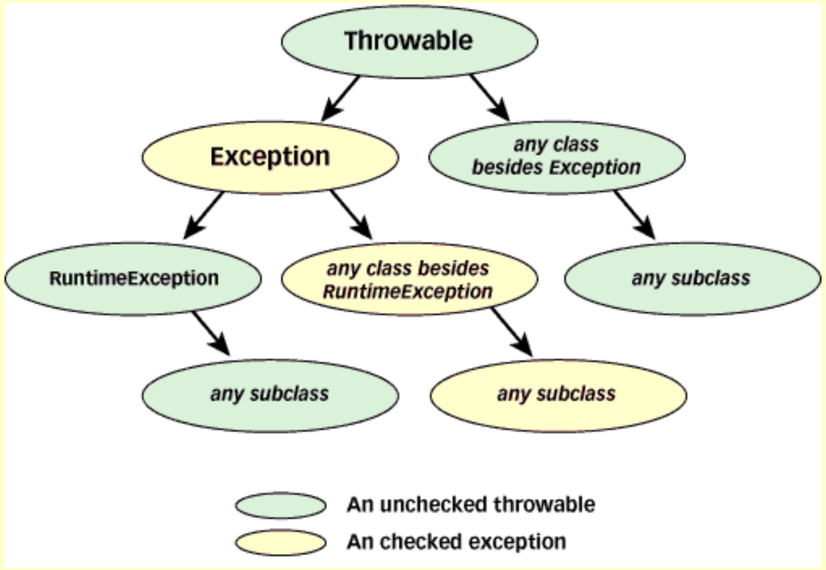
NOTE: here the info passed is a String explaining the strange taste, for TOOHOT or TOOCOLD we could pass the temperature value

Exceptions in Java: Checked vs. unchecked exceptions

There are two kinds of exceptions in Java, *checked* and *unchecked*, and only checked exceptions need appear in throws clauses.

The general rule is: Any checked exceptions that may be thrown in a method must either be caught or declared in the method's throws clause.

Checked exceptions are so called because both the Java compiler and the Java virtual machine check to make sure this rule is obeyed.



Exceptions in Java: finally block

Once a Java virtual machine has begun to execute a block of code -- the statements between two matching curly braces -- it can exit that block in any of several ways.

It could, for example, simply execute past the closing curly brace. It could encounter a break, continue, or return statement that causes it to jump out of the block from somewhere in the middle. Or, if an exception is thrown that isn't caught inside the block, it could exit the block while searching for a catch clause.

Given that a block can be exited in many ways, it is important to be able to ensure that something happens upon exiting a block, no matter how the block is exited. For example, if you open a file in a method, you may want to ensure the file gets closed no matter how the method completes. In Java, you express such a desire with a finally clause.

```
try { // Block of code with multiple exit points }
 finally {
 /* Block of code that must always be executed when the try
block exited, no matter how the try block is exited */
}
```

Exceptions in Java: good habits

Document exceptions that are thrown/handled by your own code

throw **IllegalArgumentException** when an argument to a method call is never valid

throw **IllegalStateException** when a method call (with the given argument values) is not valid in the current state

throw “InvariantBrokenException” when a method call (with the given argument values) breaks the invariant of the object executing the method

catch **IllegalAccessException** (when using reflexion to access private/protected attributes/methods)