CS1110 15 November 2011

Exceptions in Java. Read chapter 10.

HUMOR FOR LEXOPHILES (LOVERS OF WORDS):

Police were called to a day care; a three-year-old was resisting a rest.

Did you hear about the guy whose whole left side was cut off? He's all right now.

The butcher backed into the meat grinder and got a little behind in his work.

When fish are in schools they sometimes take debate.

A thief fell and broke his leg in wet cement. He became a hardened criminal.

Thieves who steal corn from a garden could be charged with stalking.

When the smog lifts in Los Angeles, U.C.L.A.

Exceptional circumstances

```
/** = the decimal number represented by s. */
int parseInt(String s) { ... }
```

…but what if s is "bubble gum"?

```
/** = the decimal number represented by s, or -1 if s
    does not contain a decimal number. */
```

…but what if s is "−1"?

```
/** = the decimal number represented by s
    Precondition: s contains a decimal number. */
```

…but what if s might not, sometimes?

Somehow, we have to be able to deal with the unexpected case.

Dealing with exceptional circumstances

```
/** = the decimal number represented by s.
    Pre: s contains a number. */
int parseInt(String s) { ... }

/** = "s contains a decimal number." */
boolean parseableAsInt(String s) { ... }
```

Now we have to write:

```
if (parseableAsInt(someString))
    i= parseInt(someString);
else {
    // do something about the error
}
```

We could roll this into one call, but it doesn't really change things:

```
/** = "s contains a decimal number."
    If yes, update the value of result */
boolean parseInt(String s, Integer result) { ... }
```

How to read a number from a file, in fourteen easy steps:
1. Open the file
2. If the file doesn't exist, …
3. If there was a disk error, …
4. Read a line from the file.
5. If the file was empty, …
6. If there was a disk error, …
7. Convert the string to a number.
8. If the string is not a number, …
9. If we have run out of memory, …
10. Close the file.
11. If there was a disk error
12. If
13. If
14. If

Common outcome: weary programmers write code that ignores errors.

There has to be a better way!

Exception Handling

```
/** Parse s as a signed decimal integer and return
    the integer. If s does not contain a signed decimal
    integer, throw a NumberFormatException. */
public static int parseInt(String s) ...
```

parseInt, when it finds an error, does not know what caused the error and hence cannot do anything intelligent about it. So it "throws the exception" to the calling method. The normal execution sequence stops!

With this definition we can write, e.g.:

```
try {
    i= Integer.parseInt(someString);   ← this might throw a NumberFormatException
    System.out.println("The number is: " + i);
} catch (NumberFormatException nfe) {  ← this tells Java we want to handle N.F.E.s here
    System.out.println("Hey! That is not a number!")
}
```

this executes if the exception happens

but we can also just write:

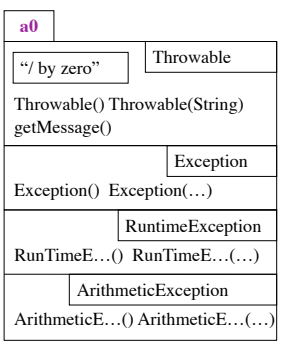
```
i= Integer.parseInt(someString);
```

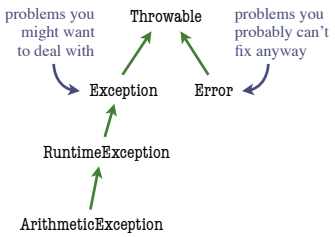
thereby letting **our** caller handle the exception instead.

Exceptions in Java

Exceptions are represented by instances of class Throwable.

Making exceptions instances of classes lets them be organized in a hierarchy.





Class

```
02 /** Illustrate exception handling */
03 public class Ex {
04     public static void first() {
05         second();
06     }
07
08     public static void second() {
09         third();
10     }
11
12     public static void third() {
13         int x= 5 / 0;
14     }
15 }
```

Call:

```
Ex.first();
```

Output:

```
ArithmeticException: / by zero
  at Ex.third(Ex.java:13)
  at Ex.second(Ex.java:9)
  at Ex.first(Ex.java:5)
```

Class →

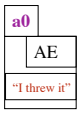
```
02 /** Illustrate exception handling */
03 public class Ex {
04    public static void first() {
05       second();
06    }
07
08    public static void second() {
09       third();
10    }
11
12    public static void third() {
13       throw new ArithmeticException ("I threw it");
14    }
15 }
```

Call:

```
Ex.first();
```

Output:

```
ArithmeticException: I threw it
  at Ex.third(Ex.java:13)
  at Ex.second(Ex.java:9)
  at Ex.first(Ex.java:5)
```




```
/** An instance is an exception */
public class OurException extends Exception {

   /** Constructor: an instance with message m*/
   public OurException(String m) {
      super(m);
   }

   /** Constructor: an instance with no message */
   public OurException() {
      super();
   }
}
```

Class →

```
02 /** Illustrate exception handling */
03 public class Ex {
04    public static void first() throws OurException {
05       second();
06    }
07
08    public static void second() throws OurException{
09       third();
10    }
11
12    public static void third() throws OurException{
13       throw new OurException ("Whoa!");
14    }
15 }
```

"This method sometimes throws OurException"

Won't compile yet!

Call:

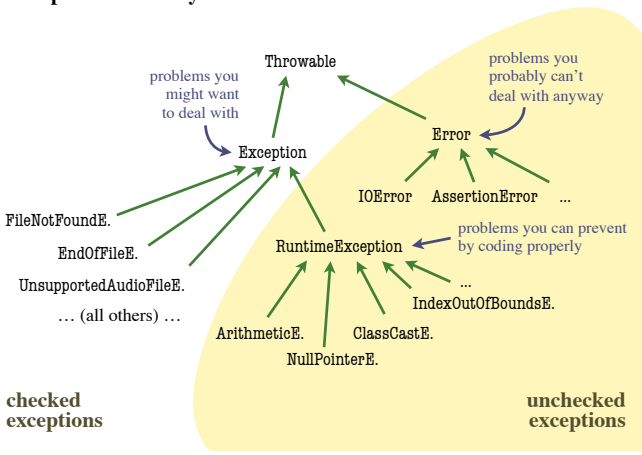
```
Ex.first();
```

Output:

```
OurException: Whoa!
  at Ex.third(Ex.java:13)
  at Ex.second(Ex.java:9)
  at Ex.first(Ex.java:5)
```

throws clauses are required because OurException, unlike ArithmeticException, is a "checked exception."

Exception Hierarchy



Catching a thrown exception

```
public class Ex1 {
   public static void first() {
      try {
         second();
      }
      catch (MyException ae) {
         System.out.println
            ("Caught MyException: " + ae);
      }
      System.out.println
         ("Procedure first is done.");
   }
   public static void second() throws MyException {
      third();
   }
   public static void third() throws MyException {
      throw new MyException("yours");
   }
}
```

Execute the try-block. If it finishes without throwing anything, fine.

If it throws a MyException object, catch it (execute the catch block); else throw it out further.

```
/** Input line supposed to contain one int, maybe whitespace on either
    side. Read line, return the int. If line doesn't contain int, keep asking
    until it does. */
public static int readLineInt() {
   String input= readString().trim();

   // inv: input contains last input line read; previous
   // lines did not contain a recognizable integer.

   while (true) {
      try {
         return Integer.valueOf(input).intValue();
      } catch (NumberFormatException e) {
         System.out.println("Input not int. Must be an int like");
         System.out.println("43 or -20. Try again: enter an int:");
         input= readString().trim();
      }
   }
}
```