Return to Module Overview Page (https://onlinelearning.berkeley.edu/courses/665040/pages/module-seven)

7.3: A Few Words about Style

1) The most important style suggestion regarding exceptions is that you not write this:

```
try
    {
       doSomething();
    }
catch(FirstException fe)
    {
       // handle error 1
    }

try
    {
       doSomethingElse();
    }
catch(SecondException se)
    {
      // handle error 2
    }

try
    {
       doSomethingMore();
    }
catch(ThirdException te)
    {
        // handle error 3
    }
```

when you could write this:

```
try
    {
       doSomething();
       doSomethingElse();
       doSomethingMore();
    }
catch(FirstException fe)
     {
        // handle error 1
     }
catch(SecondException se)
     {
        // handle error 2
     }
catch(ThirdException te)
    {
       // handle error 3
    }
```

The first example is much harder to read than the second, which is entirely equivalent. In the second case, the error-handling code is all together at the bottom of the method, with the functional code together at the top. This is a much better state of affairs than having them all interspersed.

2) A second important style suggestion is that you should *not* reduce the above to

```
try
    {
       doSomething();
       doSomethingElse();
       doSomethingMore();
    }
catch(Throwable t)
    {
       // handle all errors
    }
```

which is in fact legal since all exception classes must inherit from Throwable. Getting into the "catching Throwable" habit is bad because the compiler won't then be able to tell you about specific exception types you should be catching. Such information is revealing; there may be specific problems you can recover from in a special way, for example, or ones so severe you cannot ignore them. If you catch all exceptions generically, you lose this valuable information.

3) Third, you should use exceptions only to indicate errors, *never* as part of normal program execution. throwing and catching exceptions is computationally expensive, compared to other language constructs. Never write code like this:

```
int tweakElementsInArray(Thing [] t)
{
    int i = 0;
    try
       {
         while (i < t.length)
         {
            t[i].tweak(); i++;
         }
       }
    catch (NullPointerException npe)
       {
         return i;
       }
}
```

This code counts on a NullPointerException being thrown when we try to access a blank slot in the array (the assumption being that there are fewer Things in t than t.length). You could just as easily write this:

```
int tweakElementsInArray2(Thing [] t)
{
    int i = 0;
    while (i < t.length && t[i] != null)
```

```
    {
       t[i++].tweak();
    }
    return i;
}
```

Some people write the first version expecting that it will be faster since it avoids a test on every loop iteration. For large arrays, that may be true, but for small ones it won't be. In either case, tweakElementsInArray is just bad style. Exceptions should be used for error handling alone. Any other use just makes code harder to follow, and easier to break through modification.

4) One last piece of advice: if you define your own exception types, they can contain arbitrary data that you can use to transmit extra information about an error.

```
public class TooColdException
    extends Exception
   {
    private int m_temp;
    public TooColdException(int t){ m_temp = t; }
    public int temp(){ return m_temp; }
   }

   ...

   try
    {
       goOutside();}
       play();
    }
   catch(TooColdException tce)
    {
       System.out.println("Can't go out today;
                          it's " + tce.temp() +
                          " degrees!");
    }
```