Exceptions

In the real world, programs are never perfect and never operate in a perfect environment. The types of things that can go wrong range from program bugs, such as having an array index out of bounds, to incorrect user actions, such as not putting data into a form TextField or not supplying a file to be read, to unforeseen circumstances such as network issues that don't allow the program to make a connection or a printer that runs out of paper during a printout. So it's a given that programs have to be able to deal with unexpected errors.

Java Exceptions are a way to systematically handle errors as much as possible by having error handling code put into programs that respond correctly to errors by doing the appropriate action, such as asking the user to put in the input or just making the program print error messages and quit.

Java has a system of very detailed Exceptions and they are arranged in a hierarchy so that you can choose to handle specific exceptions with a specific action or just generally handle any exception that can occur.

Some errors we might consider as internal Java errors and these are actually called Errors and do not have to be handled by the programmer. The ArrayIndexOutOfBounds error is an example of this.

The other errors are classed as Exceptions and should be handled. Examples of these are FileNotFound, and NumberFormatException. But FileNotFound and other exceptions that occur during input/output are all classed as IOException and can be handled at once. The most general exception is called Exception and can be used to handle everything.

Handling Exceptions

Exceptions are handled by what is usually called a "try-catch" block, which has the following form:

```
try {
<code containing methods that may cause an exception>
}
catch  (<exception type>  <exception variable name>) {   // may be 1 or more
<code to handle the exception>
}
finally {                                              // optional
<code to finish up whether there is an exception or not>
}
```

The order of execution is:
• Execute the code in the try block. If any method causes an exception, quit this block immediately and go to the catch block. If no exeption occurs, skip to the finally block.
• If an exception occurs, execute the catch block and then go to the finally block.

Example1:

Converting a String to a number inside an actionPerformed method:

```
      int number;
      try
  {
      number = Integer.parseInt(jTextField1.getText());
      jLabel3.setText("Result = " + number
  }
  catch ( NumberFormatException e)
  {
      // print out the Exception
      System.out.println(e.toString());
      // give the user a message
      javax.swing.JOptionPane.showMessageDialog(this,
            "Please enter a valid integer");
      // instead of quitting, return to the user to correct the textfields
      //   and try the button again
  }
```

Example2:

Reading lines from a file and echoing them:

```
import java.io.*;

public class IOTest  {
      public static void readAndPrintLines( )  {
            // input stream variable
            BufferedReader inputStream = null;

            try {
                  // open a file for reading
                  inputStream = new BufferedReader( new FileReader("xanadu.txt"));

                  String line;
                  // read the next line of the file until there are none left
                  while (( line = inputStream.readLine()) != null)  {
                        // print out the line
                        System.out.println(line);
                  }
            }
            catch ( IOException e)  {
                  // display information about the exception and exit the program
                  e.printStackTrace();
                  System.exit(-1);
```

```
            }
            finally  {
                    // close the file
                    inputStream.close();
            }
      }
}
```

For all exceptions, there are several methods to display information. The one shown above prints what is called the stack trace: it gives the error and a list of all the methods that were called before the error. There are also:
        e.getMessage()   // can print any message attached to the exception
        e.toString()   // gives an idea of what caused the error and which method it was in
        e.printStackTrace()   // prints out the sequence of programs where the error occurred
Of course, you can also put other messages either in the output using System.out.println, or giving information dialog boxes, using one of the methods like showMessageDialog.

In your program, there are some exceptions that you are required to catch (and the compiler will complain if you don't) and others you can leave if you wish and the Java runtime will print out a stack trace as it closes the program.

How can you tell if a method will cause an Exception?

Each method that can cause an exception is required to have a "throw" clause in the header of the method. For Java system methods, we can look in an online resource called the Java API, which lists all packages, classes and methods of Java.
        http://java.sun.com/j2se/1.5.0/docs/api/
For example, the parseInt method is in the class Integer, and its method header is:

        public static int **parseInt**(String s)
                    throws NumberFormatException

In serious Java programs, the program can create its own exceptions and can have methods that throw exceptions in their code, for example, if you find that there is an error in the data. But this topic is outside the scope of this class.