Java I/O and Exceptions

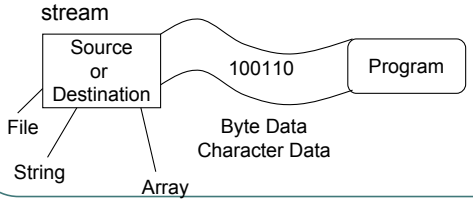
CS1316: Representing Structure and Behavior

Writing to a Text File

- We have to create a *stream* that allows us access to a *file*.
- We're going to want to write *strings* to it.
- We're going to have to handle things going wrong—***exception***al events like the filename being wrong or the disk failing.
- Here's how…

Input and Output Streams - java.io

- Java handles input and output through sequential streams of bits
- Programs can read from a stream or write to a stream



Standard Input and Output

- We have been using System.out.println to print output to a PrintStream (standard output).

```
System.out.println("First Name: " + firstName);
```

- There is also a System.err PrintStream that can be used to write to the standard error output.

```
System.err.println("Error: no file name given");
```

- You can use System.in to read a byte or bytes from an InputStream (standard input).

```
int numGrades = System.in.read();
```

Chaining Input and Output Classes

- Often input or output classes are chained
  - Passing one type of input/output class to the constructor for another
- One common thing is to chain a processing class with a data sink class
  - Like a BufferedReader or BufferedWriter and a FileReader or FileWriter

```
new BufferedReader(new
    FileReader(fileName));
```

Exceptions

- Exceptions are disruptions in the normal flow of a program. Exception is short for exceptional event.
- The programmer is *required* to handle *checked* exceptions in Java
  - like trying to read from a file that doesn't exist
- *Run-time* exceptions do not have to be handled by the programmer
  - like trying to invoke a method on a object reference that is **null**
  - Children of **RuntimeException**

Try and Catch

- Use a try & catch clause to catch an exception

```
try {
    code that can cause exceptions
} catch (ExceptionClassName varName) {
    code to handle the exception
} catch (ExceptionClassName varName) {
    code to handle the exception
}
```

- You can catch several exceptions
  - Make the most general one last
  - All exceptions are children of the class **Exception**

Try and Catch: If the file isn't there…

- What if you want to know if a file isn't found
  - That you are trying to read from
  - If this occurs you might want to use a **JFileChooser** to let the user pick the file
- You also want to handle any other error

```
try {
    code that can cause the exception
} catch (FileNotFoundException ex) {
    code to handle when the file isn't found
} catch (Exception ex) {
    code to handle the exception
}
```

Catching Exceptions

- A catch clause will catch the given **Exception** class and any subclasses of it.
- So to catch all exceptions use:

```
try {
    code that can throw the exception
} catch (Exception e) {
    System.err.println("Exception: " + e.getMessage());
    System.err.println("Stack Trace is:");
    e.printStackTrace();
}
```

You can print the error message and the *stack trace* (the list of all currently running methods)

- You can create your own exceptions by subclassing **Exception** or a child of **Exception**.

The optional finally clause

- A try and catch statement can have a finally clause
  - Which will always be executed
    - Will happen if no exceptions
    - Will happen even if exceptions occur

```
try {
    code that can cause the exception
} catch (FileNotFoundException ex) {
    code to handle when the file isn't found
} finally {
    code to always be executed
}
```

Writing to a File

- Use a try-catch clause to catch exceptions
  - Create a buffered writer from a file writer

    writer = new BufferedWriter(new FileWriter(fileName));
  - Write the data

    writer.write(*data*);
  - Close the buffered writer
    - writer.close();

Reading Lines of Character Data

- Enclose the code in a try and catch clause
  - Catch FileNotFoundException if the file doesn't exist
    - And you may want to give the user a chance to specify a new file
  - Catch **Exception** to handle all other errors
- Create a buffered reader from a file reader for more efficient reading
  - File names are relative to the current directory
- Loop reading lines from the buffered reader until the line is null
  - Do something with the data
- Close the buffered reader

Reading from File Example

```
BufferedReader reader = null;
String line = null;

// try to read the file
try {

  // create the buffered reader
  reader = new BufferedReader(new FileReader(fileName));

  // loop reading lines till the line is null (end of file)
  while ((line = reader.readLine()) != null)
  {
     // do something with the line
  }

  // close the buffered reader
  reader.close();

} catch (Exception ex) {
  // handle exception
}
```

Adding an Output File to WolfDeerSimulation

```
/* A BufferedWriter for writing to */
 public BufferedWriter output;

 /**
  * Constructor to set output to null
  **/
 public WolfDeerSimulation() {
   output = null;
 }
```

Opening the File

```
/**
  * Open the input file and set the BufferedWriter to speak to it.
  **/
 public void openFile(String filename){
  // Try to open the file
   try {

     // create a writer
     output = new BufferedWriter(new FileWriter(filename));

   } catch (Exception ex) {
     System.out.println("Trouble opening the file " + filename);
     // If any problem, make it null again
     output = null;
   }
 }
```

Changing the time loop

```
// Let's figure out where we stand...
     System.out.println(">>> Timestep: "+t);
     System.out.println("Wolves left: "+wolves.getNext().count());
     System.out.println("Deer left: "+deer.getNext().count());

     // If we have an open file, write the counts to it
     if (output != null) {
       // Try it
       try{
         output.write(wolves.getNext().count()+"\t"+deer.getNext().count());
         output.newLine();
       } catch (Exception ex) {
         System.out.println("Couldn't write the data!");
         System.out.println(ex.getMessage());
         // Make output null so that we don't keep trying
         output = null;
       }
     }
```

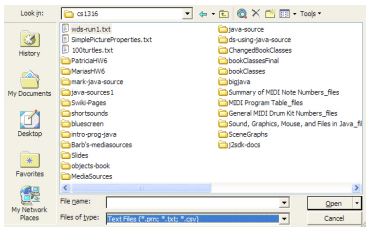
After the timing loop

```
// If we have an open file, close it and null the variable
  if (output != null){
    try{
      output.close();}
    catch (Exception ex)
    {System.out.println("Something went wrong closing the file");}
    finally {
      // No matter what, mark the file as not-there
      output = null;}
  }
```

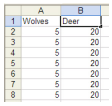
Running the Simulation with a File

```
Welcome to DrJava.
> WolfDeerSimulation wds = new WolfDeerSimulation();
> wds.openFile("D:/cs1316/wds-run1.txt")
> wds.run();
```

Finding the file in Excel



Adding Labels for the Chart



	A	B
1	Wolves	Deer
2	5	20
3	5	20
4	5	20
5	5	20
6	5	20
7	5	20
8	5	20