11. Exceptions

- Objectives - when we have completed this set of notes, you should be familiar with:
  - the purpose of exceptions
  - exception messages
  - the try-catch statement
  - propagating exceptions
  - checked and unchecked exceptions
  - reading and writing text files
  - try-catch and exceptions for files
  - exception messages
  - opening files in the default web browser

Exceptions

- An ***exception*** is an object that describes an unusual or erroneous situation
- Exceptions are ***thrown*** by a program during execution; they may be ***caught*** and ***handled***, or they may be ignored (as we've been doing)
- A program can be separated into a normal execution flow and an ***exception execution flow***
- An ***error*** is also represented as an object in Java, but usually represents an unrecoverable situation and should not be caught

Exception Handling

- If an exception is ignored by the program, a run-time error will occur. *An example you've likely seen:*

```
----jGRASP exec: java Test

Exception in thread "main" java.lang.NullPointerException
       at Test.main(Test.java:6)

----jGRASP wedge2: exit code for process is 1.
----jGRASP: operation complete.
```

- A *call stack trace* (where the exception occurred) is included
- The call stack trace shows the method call trail that led to the attempted execution of the offending line
- See `Zero1.java`

Exception Handling

- Java has a predefined set of exceptions and errors that can occur during execution. Examples:
  - ArrayIndexOutOfBoundsException in the java.lang package
  - NullPointerException in the java.lang package
- A program can deal with an exception in one of three ways:
  - ignore it
  - handle it where it occurs
  - handle it an another place in the program

The try Statement

- To process an exception when it occurs, the line that throws the exception is executed within a *try block*
- A try block is followed by one or more *catch* clauses, which contain code that is run if the exception is thrown
- When an exception occurs, processing continues at the first catch clause that matches the exception type
- See `Zero2.java`
  `AbsoluteValue1.java`
  `AbsoluteValue2.java`

The finally Clause

- A try statement can have a `finally` clause
- Once a program enters the try block, the statements in the finally clause are always executed [unless System.exit() is called]
  - If no exception is generated, the statements in the **finally** clause are executed **after the statements in the try block complete**
  - If an exception occurs, control jumps to the matching **catch** clause, if any, and its statements are executed, and then the statements in the **finally** clause are executed.
- See `Zero3.java`
  `GuessNumber1.java` `GuessNumber2.java`

Exception Propagation

- An exception can be handled at a higher level if it is not appropriate to handle it where it occurs
- Exceptions *propagate* up through the method calling hierarchy until they are caught and handled or until they reach the level of the `main` method
- A try block that contains a call to a method in which an exception is thrown can be used to catch that exception
- See `Propagation.java` `ExceptionScope.java`

The throw Statement

- You may want to throw an exception in a method
  - Often better than just ignoring incorrect input / actions
- Exceptions are thrown using the *throw* statement
- Usually an if statement evaluates the condition to see if the exception should be thrown
- You can create your own exceptions if there is not an appropriate exception in the Java API
  See `PolygonCreator.java`

Checked Exceptions

- An exception is either *checked* or *unchecked*
- A *checked exception* either must be caught by a method, or must be listed in the *throws clause* of any method that may throw or propagate it
- A throws clause is appended to the method header
- The compiler will issue an error if a checked exception is not handled appropriately

Unchecked Exceptions

- An unchecked exception does not require you to handle it (recall, you have used Double.parseDouble without a try-catch or throws clause)
- The only unchecked exceptions in Java are objects of type `RuntimeException` or any of its descendants
- Error objects are similar to `RuntimeException` objects in that they are unchecked
  - Errors do not require a throws clause
  - Errors should not be caught (e.g., OutOfMemoryError)

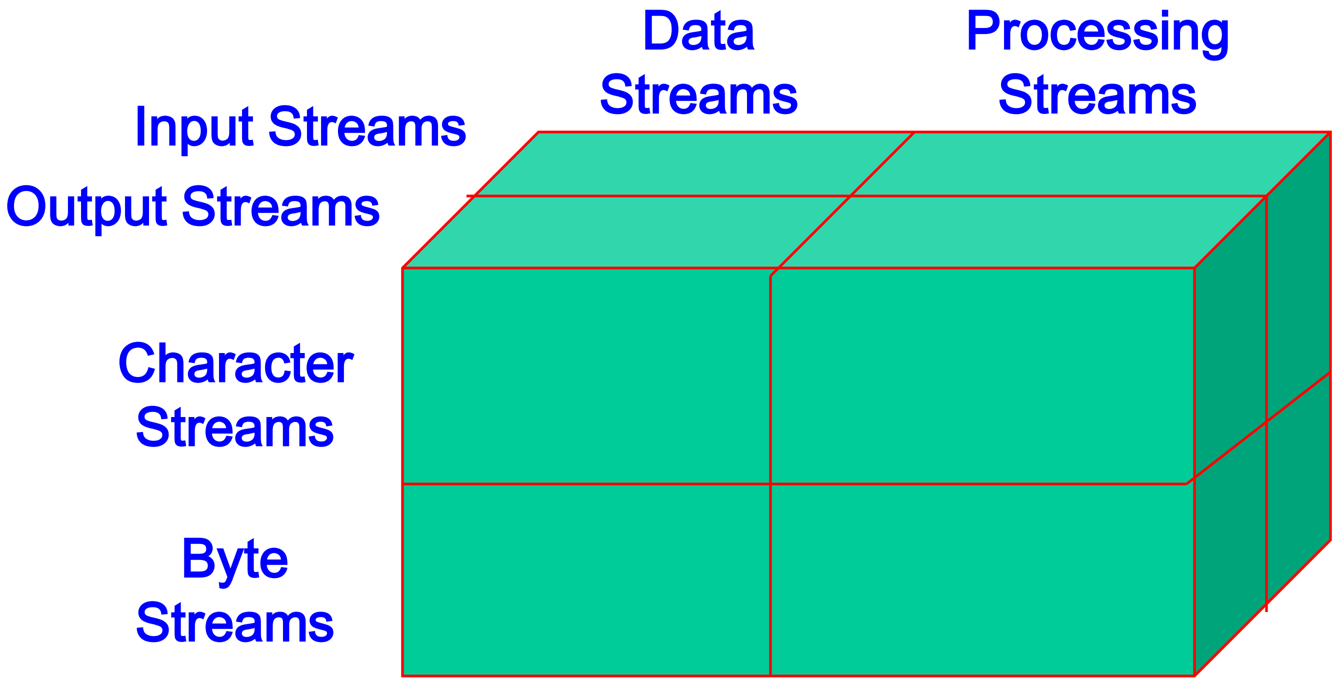
I/O Streams

- A *stream* is a sequence of bytes that flow from a source to a destination
- In a program, we read information from an input stream and write information to an output stream
- A program can manage multiple streams simultaneously

I/O Streams

- The `java.io` package contains many classes that allow us to define various streams with particular characteristics
- Some classes assume that the data consists of characters
- Others assume that the data consists of raw bytes of binary information
- Streams can be further subdivided as follows:
  - *data stream*, which acts as either a source or destination
  - *processing stream*, which alters or manipulates the basic data in the stream

I/O Streams


Data Streams
Processing Streams
Input Streams
Output Streams
Character Streams
Byte Streams

Standard I/O

- There are three standard I/O streams:
  - *standard input* – defined by `System.in`
  - *standard output* – defined by `System.out`
  - *standard error* – defined by `System.err`
- `System.in` is typically keyboard input
  - We've been using the Scanner class to read from System.in
- `System.out` and `System.err` are typically shown in a particular window on the screen
  - We use `System.out` when we execute `println` statements

File I / O

- Objectives - when we have completed this set of notes, you should be familiar with:
  - reading and writing text files
  - try-catch and exceptions for files
  - exception messages
  - opening files in the default web browser

I/O Streams

- The `java.io` package contains many classes that allow us to define various I/O streams
- You know about standard input and output. Now let's consider the details of reading and writing to files
- For <u>reading from a file</u>, we use the following:
  - java.io.File and java.util.Scanner
- For <u>writing to a file</u>, we use the following:
  - java.io.PrintWriter

Files

- The extension of a file specifies what program is used by the operating system to open the file.
  - input.txt
  - input.dat
  - input.xyzabc
- If a file contains text and does not have extension .txt, you'll have to specify what program to use to view the file's contents
- See input.xyzabc

Reading from a File

- In order to read from a file, you will have to create an instance of the File class in java.io
- You can then instantiate a Scanner object using the File object that you created.
- At that point you can use any of the methods that you have been using in Scanner to read the file:
  - The next method reads a "token"
  - The nextLine method reads a whole line
  - The hasNext and hasNextLine are also useful (see API documentation for more information and methods)

Reading from a File

- Look at the Java API documentation for Scanner. The constructor that accepts a File object as a parameter throws a FileNotFoundException.
- FileNotFoundException is a checked exception; you have to do one of two things …
  - Handle the exception with a try-catch
  - Specify throws FileNotFoundException in the method header
- Your program doesn't care what the extension of the file is.
- See ReadLines.java

Writing to a File

- Instantiate a PrintWriter object using the file name (a String).
  - The PrintWriter constructor throws FileNotFoundException.
- PrintWriter has methods similar to System.out
  - print: writes a specified String to a file
  - println: writes a specified String and a new line to a file
- **Do not forget to invoke the close() method on the PrintWriter object; otherwise nothing may be written to the file!**
- See WriteLines.java

Writing HTML to a File

- Similar to writing plain text
- Use PrintWriter
- Add HTML tags to the text (here are a few)
  - Heading <h1> . . . </h1>
  - Paragraph <p> . . . </p>
  - Line break <br>
  - Bold <b> . . . </b>
  - Font color <font color='blue'> . . . </font>
- Opening HTML file in default browser
- WriteLinesHTML.java
- WriteReadRandom.java WriteRandomHTML.java