# CECS 277 – Lecture 5 – Exceptions & File Input/Output

**Exceptions** – are Java's way of telling you that something bad happened, usually something so bad that the program crashes. Java gives us the ability to handle these exceptions if we plan for them ahead of time.

**Examples** – Here are a few examples of run time errors you may have seen:

1. InputMismatchException – occurs when the user inputs a value that is not of the expected data type. `int x = in.nextInt(); //user inputs an 'a'`
2. ArithmeticException – occurs when you divide by 0.
3. ArrayIndexOutOfBoundsException – occurs when you attempt to access an element of an array that does not exist.

Whenever one of these situations arises, an exception is thrown. If it is caught by your code, then your program can continue on. If it isn't caught, then your program will crash.

**Try/Catch Block** – the try section allows us to attempt code that may pose a risk of crashing the program. If an exception isn't thrown, it just continues on with the rest of the code. If an exception is thrown, then we need to catch it with a catch block by identifying the type of exception it will expect and then executing the solution.

Multiple catch blocks may be used after a try block if you expect more than one possible exception to be thrown. You can use as many catch blocks as you need.

**Example** – InputMismatchException and ArrayIndexOutOfBounds

```
import java.util.*;//needed for scanner and input exception
import java.lang.*;//needed for array exception
public class InputTest {
     public static void main(String [] args) {
          Scanner in = new Scanner(System.in);
          int [] array = {0, 1, 2, 3, 4};
          boolean testInput = true;
          while ( testInput ) {
               System.out.print("Enter a Number: ");
               try {
                    int num = in.nextInt(); //exceptions?
                    System.out.println( array[num] );
                    testInput = false;
               } catch ( InputMismatchException im ) {
                    in.next(); //clear buffer
                    System.out.println("Invalid Input");
               } catch (ArrayIndexOutOfBoundsException ab){
                    System.out.println("Invalid Index");
               }
          }
     }
}
```

**File I/O** – Information can be read from and written to files. To do this, two libraries need to be imported.

```
import java.io.*;
import java.util.Scanner;
```

**Reading from a File –** the file to be read should be inside of the project folder unless an exact path to the location of the file is given. Note: if you are specifying an exact pathname with a \ in it, you will have to include the escape character for it, which is \\.

**Declare the Scanner, File Reader, and Input File** – Java requires that a FileNotFoundException be handled, so a try/catch block is needed.

```
try {
     Scanner read=new Scanner(new File("input.txt"));
```

**Read from the File** – the Scanner works just like it normally does, you can use nextLine(), next(), nextInt(), or nextDouble() to retrieve data from the file.

```
     do {
          String line = read.nextLine();
          // do something with the string
```

**Checking for the End of a File**:

```
     } while( read.hasNext() );
```

**Close the File**:

```
     read.close();
} catch( FileNotFoundException fnf ) {
     System.out.println("File was not found");
}
```

**Breaking up a String –** often times an entire record is stored on a single line that requires you to break up the string into the individual fields that you need. Using a string and a delimiter, you can break up a string into an array of individual strings by using the String's split() method.

Given a string with a person's information separated by commas:

```
String line = "Mary,Smith,123 Fake St.,Phoenix,AZ";
```

You can split up the contents into an array of strings, each element containing the different fields using:

```
String [] tokens = line.split(",");
```

The split method breaks up the string line into the five smaller strings that are separated by commas (the commas are removed from the strings).

The delimiter uses regular expressions, so for more complicated delimitating, refer to the following: http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html. You can use http://regexpal.com/ to practice and to check your regular expressions.

**Writing to a File –** write text to a new file by creating a PrintWriter. Be aware, PrintWriter does not append to a file when created this way. It completely overwrites the contents of the file every time you open it. Print the content to a file the same way you would print to the screen using print(), printf(), and println(), but instead of using System.out, use your PrintWriter object. Close the file when you are finished writing to save the changes.

**Declare the Writer**:

```
try{
     PrintWriter writer = new PrintWriter("output.txt");
```

**Write to the File**:

```
     for ( int i = 1; i <= 10; i++ ) {
          writer.println( i );
     }
```

**Close the file**:

```
     writer.close();
} catch( FileNotFoundException fnf ) {
     System.out.println("File was not found");
}
```