# Input & Output in Java
# Using Text Files

# Recall: Exception Handling

- when a program is asked to perform an action, it generally assumes such an action is possible
- if a situation occurs where the action is not possible, the program will throw an exception
- for example
  - dividing by zero
  - array index out of bounds
  - reading from an empty buffer
  - working with a file that does not exist
- an unhandled exception will crash a program

# File Output

- FileWriter: open a named file for output
  - if the file does not exist, create it
  - if the file does exist, any data will be overwritten

```
FileWriter fw = new FileWriter("dataFile.txt");
```

- PrintWriter: send output to an opened file

```
PrintWriter pw = new PrintWriter(fw);
pw.println("Hello");
pw.println("World");
pw.close();
```

- once writing is complete, close the file

# File Output

- import Java input/output libraries
- enclose operation in try-catch in case of failure

```
import java.io.*;
class FileOutDemo
{
  public static void main(String[] args)
  {
    try
    {
      FileWriter fw = new FileWriter("dataFile.txt");
      PrintWriter pw = new PrintWriter(fw);
      pw.println("Hello");  // will appear on 1st line
      pw.println("World");  // will appear on 2nd line
      pw.close();           // close file for writing
    }
    catch(IOException e){}
  }
}
```

# Input Stream from File

- start by creating a FileReader for the data file

```
// open a text file for reading
FileReader fr = new FileReader("dataFile.txt");
```

- like a keyboard, the text from a file creates a stream of data (characters)
- these characters will be directed to a buffer in system memory (RAM)

```
// direct contents of text file to buffer
BufferedReader br = new BufferedReader(fr);
```

# Reading File Data as Strings

- all buffered data is initially a collection of characters assembled into one string per line
- an empty line from the file will be interpreted as the 'null' string, signaling the end of the data

```
FileReader fr = new FileReader("dataFile.txt");
BufferedReader br = new BufferedReader(fr);
String line;

line = br.readLine();  // read first line
while (line != null)
{
  System.out.println(line);  // read next line
  line = br.readLine();
}
br.close();
```

```
import java.io.*;
class FileInDemo
{
  public static void main(String[] args)
  {
    try
    {
      FileReader fr = new FileReader("dataFile.txt");
      BufferedReader br = new BufferedReader(fr);
      String line;

      line = br.readLine(); // read first line
      while (line != null)
      {
        System.out.println(line);  // read next line
        line = br.readLine();
      }
      br.close();
    }
    catch(IOException e){}
  }
}
```

# Recall: Parsing Data

- once data has been temporarily stored in a string, you may wish to store some data in a more meaningful form (e.g., integers, doubles)
- use the same techniques as used for standard input from the keyboard
- must include a "catch" in case a conversion between string (from text file) and variable (e.g., integer) fails

```
import java.io.*;
class FileParseInputDemo
{
  public static void main(String[] args)
  {
    try
    {
      FileReader fr = new FileReader("dataFile.txt");
      BufferedReader br = new BufferedReader(fr);
      String line;

      line = br.readLine();                          // read first line
      String name = line;                            // store name

      line = br.readLine();
      int age = Integer.parseInt(line);              // store age

      line = br.readLine();
      double bankBal = Double.parseDouble(line); // store balance

      br.close();
    }
    catch(IOException e){
      System.out.println("Error Reading from File");
    }
    catch(NumberFormatException err) {
      System.out.println("Error Converting Number");
    }
  }
}
```