CIT 249: Java II
Chapter 19 Lab 2

In this program we will complete #3 under Programming exercises on page 733. As in the first lab we will both upload and download to a file. Instead of writing text to a text file, the output will be in binary format. In the input it will read the number of numbers read and display the total.

1. Open a new document window in Textpad and save the file as Ch19Lab2.java. First well provide some comments as to the purpose of the program. Type:

```
/* Purpose:  Data input and output in binary form.
*/
```

2. Next we import all the classes in the java.io package by typing:

```
import java.io.*;
```

3. Type the class header, opening brace, main method header, and opening brace for that method.
4. We declare our input, output objects and a string that is given the value of the name of the file to be read. Type:

```
// Declare data input and output streams
 DataInputStream dis = null;
DataOutputStream output = null;

int count = 0;
```

   - The DataInputStream class handles the stream by reading contents of a file.
   - The DataOutputStream does the reverse by writing to a file. The DataInputStream and DataOutputStream are primarily for binary content.

5. We declare and construct an integer that will keep track of the numbers counted. Type:

```
int count = 0;
```

6. Through a try/catch statement we will designate the file to be read, and read each number. This is accomplished through a try statement so that we can allow it to handle errors and error messages. Type:

```
try {
// Create data input stream
dis = new DataInputStream(new FileInputStream("Ch19Lab2.dat"));
int total = 0;
while (dis.available() > 0) {
  int temp = dis.readInt();
  total += temp;
  count++;
  System.out.print(temp + " ");
```

```
  }

  System.out.println("\nCount is " + count);
  System.out.println("\nTotal is " + total);
 }
```

- The *DataInputStream* has the arguments of a *FileInputStream*. This class handles the reading of the file, while the DataInputStream handles the actual data.
- The available() method of the DataInputStream allows us to go through each integer in the file and will stop once the end of the file is reached. As it reads the integer it is added to a integer named *temp*.
- The integer *total* receives the value of the *temp* plus its current value using the operator +=.
- The integer *count* is increased by one, and it prints out the value of *temp* + a space.
- After the while loop is closed the values of the variables *count* and *total* displayed.hat
- Note that there must be a space between the double quotes in the System.out.print(), otherwise the numbers will run together and be unreadable.

7. Next we create two catch statements. Type:

```
  catch (FileNotFoundException ex) {
  System.out.println("File not found");
 }
 catch (IOException ex) {
  System.out.println(ex.getMessage());
 }
```

- The first is for when the designated file is not found. In this case a simply message will be displayed.
- However if the file is found and the data being inputted is corrupt the default message for the IOException will be displayed.

8. Next we create a finally statement. Whereas catch statements are run only if the error occurs, finally statements are always run but they are optional, whereas the catch statement is not. Type:

```
  finally {
  try {
   // Close files
   if (dis != null) dis.close();
  }
  catch (IOException ex) {
   System.out.println(ex);
  }
 }
```

- This finally statement includes its own try/catch statement and is used to the close the stream.

9. Our next step is to use the DataOutputStream to write integers to a file, but in binary format. Type:

```
//output to a different file using the FileOutputStream, and DataOutputStream classes
try
{
    output = new DataOutputStream(new FileOutputStream("Out.dat"));

    for(int i = 0; i< 100; i++)
        output.writeInt((int)(Math.random() * 100000));

}
catch(IOException e)
{
    System.out.println("Unable to create file");
}
```

- Only one catch statement was used here, in case there was a problem creating the file.

10. Close the main method and class.
11. Compile the program, fix any errors if necessary and test it.
12. Compress all files, including the .dat file, into a single zip or rar file and submit to the drop box for this assignment.