CMIS 141 Hands-on Lab
Week 8

Overview

The week we continue our study of Java by studying exception handling and File Input/Output. In this lab, we will demonstrate how exceptions can be used to gracefully handle run-time errors where resources or data is not as expected. We will also use Java classes to read from and write to a file.

It is assumed the JDK 8 or higher programming environment is properly installed and the associated readings for this week have been completed.

Submission requirements

Hands-on labs are designed for you to complete each week as a form of self-assessment. You do not need to submit your lab work for grading. However; you should post questions in the weekly questions area if you have any trouble or questions related to completing the lab. These labs will help prepare you for the graded assignments, therefore; it is highly recommended you successfully complete these exercises.

Objectives

The following objectives will be covered by successfully completing each exercise:

1. Use try, catch and finally statements to catch and handle exceptions
2. Read data from a file using Java
3. Write to a file using Java

Exercise 1 – Use try, catch and finally statements to catch and handle exceptions

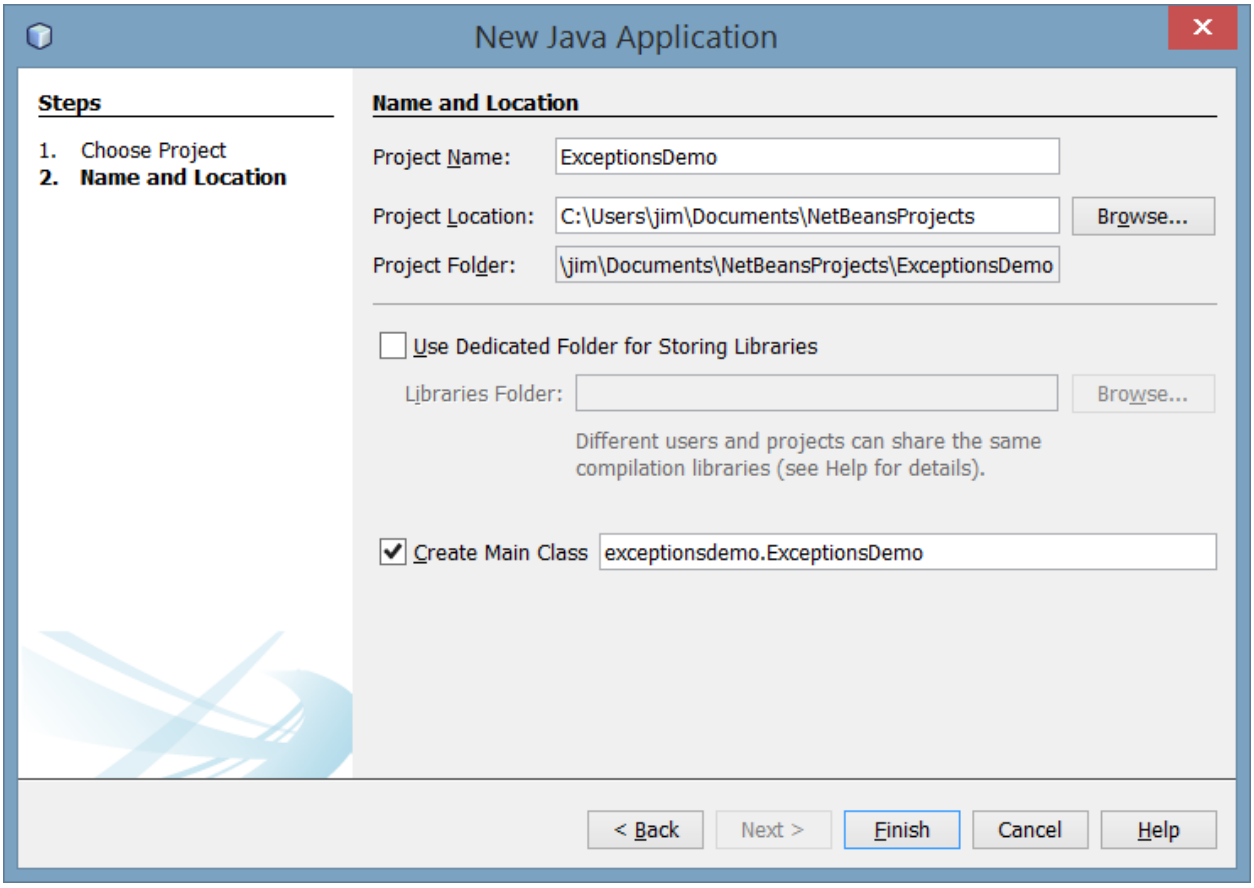
Run-time errors often occur due to missing resources or incomplete or incorrect data input. For example, if you were planning on reading data from a file and the file was not found on the system, a run-time error would result. Similarly, if the user needed to enter number and entered non-numeric values instead, when the program attempted to process the values, a run-time would result. Java, and most other modern languages, have mechanisms to catch, and if needed, handle exceptions.

As you can tell by the reading, exception handling can become very complex with dozens of exception classes to work with. This exercise will concentrate on using try, catch and finally statements to catch and handle simple and common exceptions. The following code snippet shows a line of code that could potentially cause a run-time error if the inputValue was not an integer value.

```
try {
  int age = Integer.parseInt(inputValue);
}
catch (NumberFormatException ne) {
  System.err.println("NumberFormatException: " + ne.getMessage());
  System.out.println("Please start again with a valid age");
finally {
    System.out.println("finally always executes");
}
```

If an exception does occur, the code inside the catch clause will be executed. The finally clause is always executed regardless of whether an exception occurred in the execution.

In this exercise, we will ask the user to enter several numeric values and use available Java exceptions to catch and report possible run-time errors. Launch your IDE and create a new project named ExceptionsDemo.

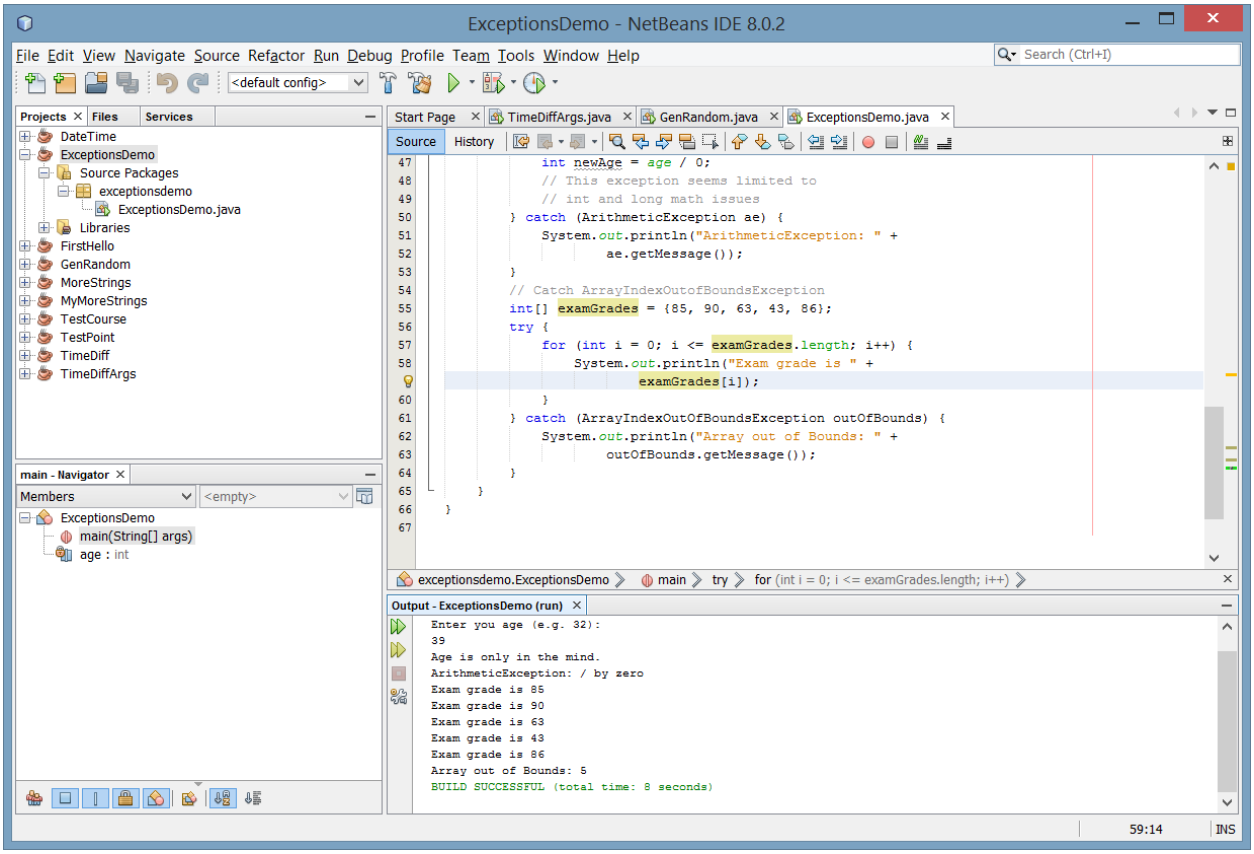


a. Type, or copy and paste the following java code into the ExceptionsDemo.java file in your IDE.

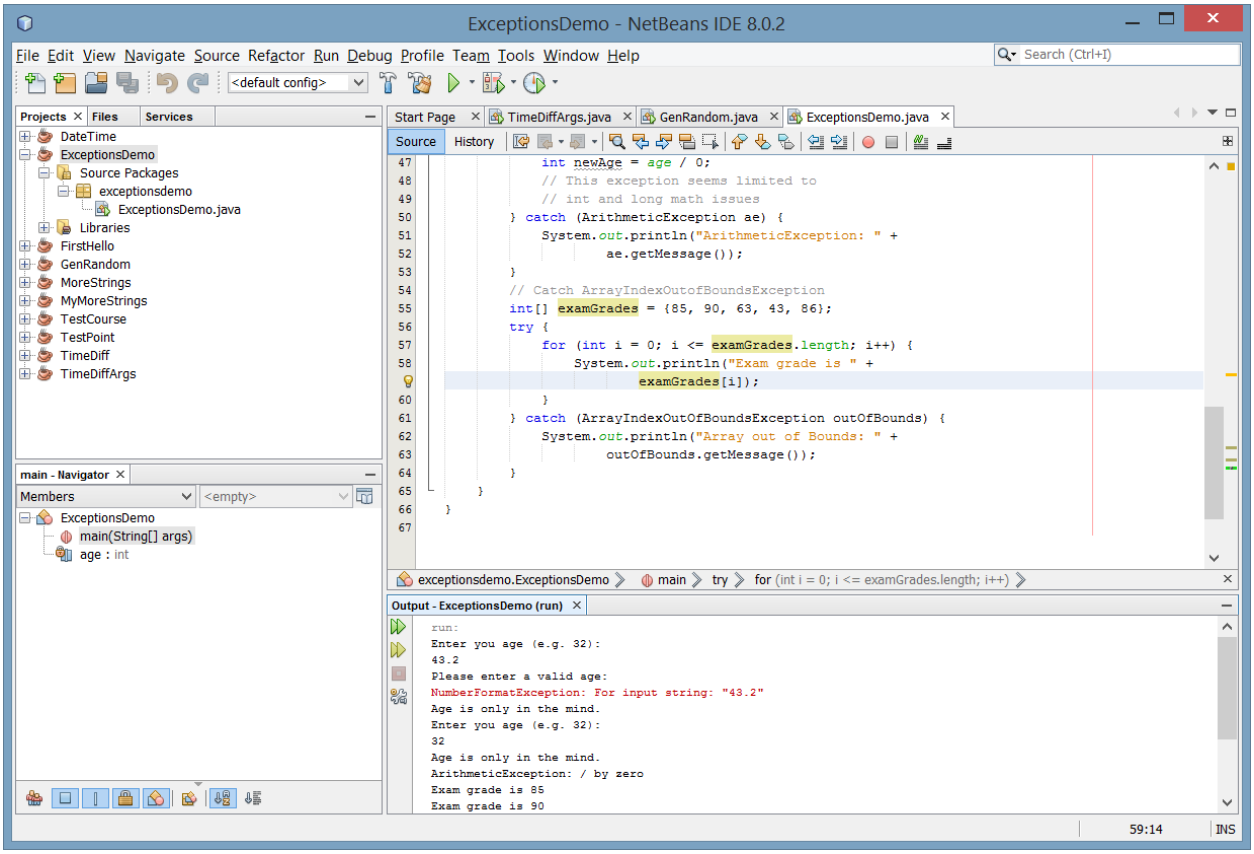
```
package exceptionsdemo;
import java.util.Scanner;


/*
 * File: ExceptionsDemo.java
 * Author: Dr. Robertson
 * Date: January 1, XXXX
 * Purpose: This program demonstrates
 * catching several Java run-time exceptions
 */
public class ExceptionsDemo {
```

```
 // Declare private variables
 private static int age = 32;

 /**
  * @param args the command line arguments
  */
 public static void main(String[] args) {

     // TODO code application logic here
     Scanner myScanner = new Scanner(System.in);

     // Prompt use to enter an int
     while (true) {
         try {
             System.out.println("Enter you age (e.g. 32): ");
             age = Integer.parseInt(myScanner.next());
             break;
         } catch (NumberFormatException ne) {
             System.err.println("NumberFormatException: " +
                     ne.getMessage());
             System.out.println("Please enter a valid age: ");
         } finally {
             System.out.println("Age is only in the mind.");
         }
     } // End while loop
     // Catch ArithmeticException
     try {
         int newAge = age / 0;
         // This exception seems limited to
         // int and long math issues
     } catch (ArithmeticException ae) {
         System.out.println("ArithmeticException: " +
                 ae.getMessage());
     }
     // Catch ArrayIndexOutofBoundsException
     int[] examGrades = {85, 90, 63, 43, 86};
     try {
         for (int i = 0; i <= examGrades.length; i++) {
             System.out.println("Exam grade is " +
                     examGrades[i]);
         }
     } catch (ArrayIndexOutOfBoundsException outOfBounds) {
         System.out.println("Array out of Bounds: " +
                 outOfBounds.getMessage());
     }
   }
}
```

b. Compile and run the code by clicking on the green arrow in the IDE.
   When prompted, enter an integer for the age.



On subsequent runs, type a decimal value, or String with any characters and notice the NumberFormatException is thrown. The program will continue to prompt you for a valid age.



As you analyze and experiment with the code, note the following:

1. A loop using a break; statement at the proper location, can allow the program to continue to prompt a user for a value until the format is acceptable. Here the break; is used to break out a potentially infinite loop when a valid value is entered.

```
while (true) {
      try {
          System.out.println("Enter you age (e.g. 32): ");
          age = Integer.parseInt(myScanner.next());
          break;
      } catch (NumberFormatException ne) {
          System.err.println("NumberFormatException: " +
                  ne.getMessage());
          System.out.println("Please enter a valid age: ");
      } finally {
          System.out.println("Age is only in the mind.");
      }
  } // End while loop
```

2. The finally clause is always executed, regardless of whether or not an exception is thrown.
3. You can use the getMessage() method of the exceptions to display useful information related to the exception that occurred.

As always you should experiment with the code by adding additional functionality to invoke and catch more exceptions. Use the Java 8 API to look up additional exceptions found in the java.lang package.

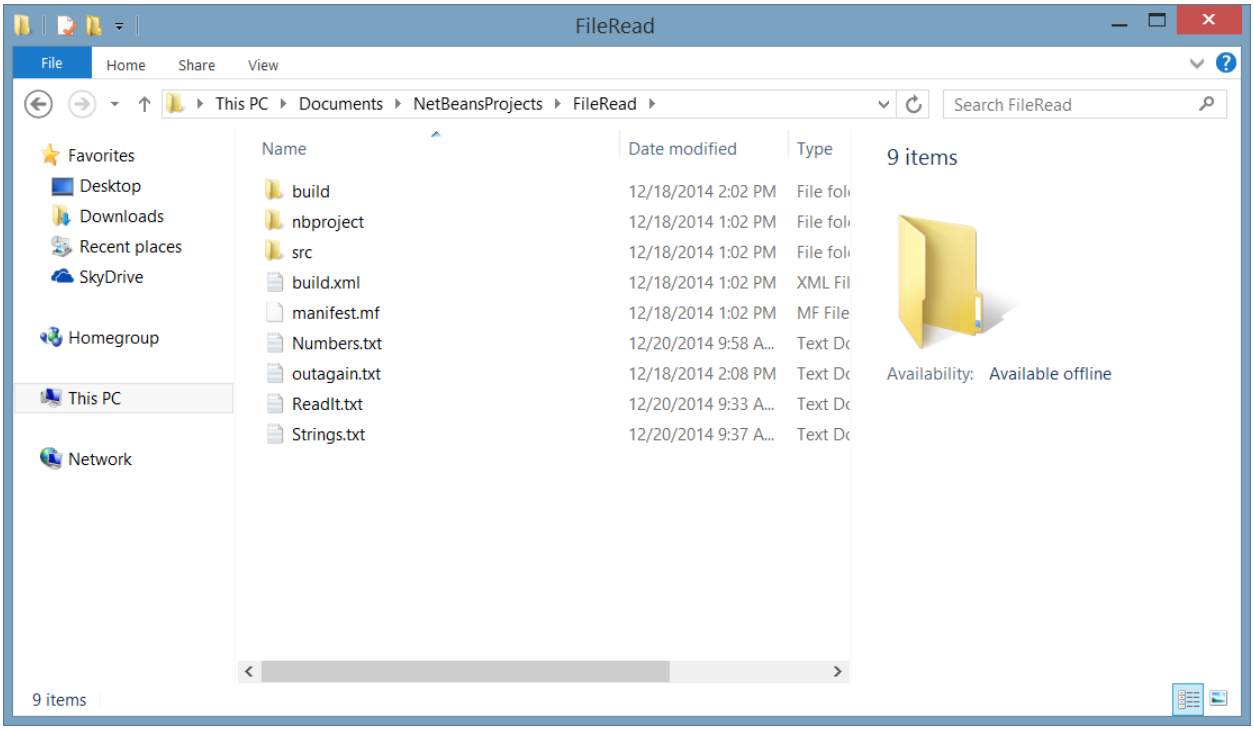
Now it is your turn. Try the following exercise:

Using your IDE, create a project named AddressBook. Prompt the user to enter their name, age and zip code. Use exceptions to catch values that are not numbers for the age and zip code. (Assume a 5-digit zip code). Continue to prompt the user for a valid values if entries are not valid.

Exercise 2 – Read data from a file using Java

We have used the Scanner class to read data from the System.in (standard input) in exercises in previous weeks. This exercise will demonstrate how to use the Scanner class, and other classes to read data from a file. When reading data file from a file, you need to create a text file to be read. You can do this in any text editor or even use Netbeans to create the file.

The critical part of the file creation is where you place the file to be read. If you are using Netbeans, you need to place the files in the Netbeans Projects/Project name location. For example, if I was going to read a file named "Strings.txt" from a Netbeans project named FileRead, I would place the Strings.txt file in the Netbeans Projects/FileRead folder.



Note your installation folders will be different. So find your Netbeans projects folder for correct placement. You will receive a FileNotFound Exception if you haven't placed your files in the correct location.

In this exercise, we will read data from three different files using three different, yet similar approaches. After reading the data, we will simply echo the data out to standard output. In the next exercise, we will write the data to a file.

Launch your IDE and create a new project named FileRead.

a. Type or copy and paste the following java code into the ExceptionsDemo.java file in your IDE.

```
package fileread;

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileReader;
import java.io.IOException;
import java.util.Scanner;

/*
 * File: FileRead.java
 * Author: Dr. Robertson
 * Date: January 1, XXXX
 * Purpose: This program demonstrates
 * reading data from files
*/

public class FileRead {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Scanner scannerIn = null;
        FileInputStream in = null;
        BufferedReader inputStream = null;

        // int equivalent of the char
        int fileChar;
        String fileLine;
        try {
            // File should be located
            // in Netbeans folder Project location
            // Netbeans Projects/FileRead/
            // Use of FileInputStream
             in = new FileInputStream("ReadIt.txt");

             System.out.println("ReadIt File Contents");
             // Read one char at a time
            while ((fileChar = in.read()) != -1) {
                // convert int to char
                System.out.print((char) fileChar);
            }
            // Separate the file output
            System.out.println("");

            System.out.println("Numbers.txt File Contents");
```

```
            // Use of Scanner and BufferedReader
           inputStream =  new BufferedReader(new FileReader("Numbers.txt"));
            scannerIn = new Scanner(inputStream);
            while (scannerIn.hasNext()) {
                if (scannerIn.hasNextInt()) {
                    System.out.println(scannerIn.nextInt());
                }
                if (scannerIn.hasNextDouble()) {
                    System.out.println(scannerIn.nextDouble());
                } else {
                    scannerIn.next();
                }
            }

            // Separate the file output
            System.out.println("");
            // Use of
            inputStream = new BufferedReader(new FileReader("Strings.txt"));

            System.out.println("Strings.txt Contents");
           // Read one Line using BufferedReader
            while ((fileLine = inputStream.readLine() ) != null) {
                System.out.println(fileLine);
            }
        }  catch (IOException io) {
            System.out.println("File IO exception" + io.getMessage());

        } finally {
            // Need another catch for closing
            // the streams
            try {
                // Close the streams
                if (in != null) {
                    in.close();
                }
                if (inputStream != null) {
                inputStream.close();
            }

            } catch (IOException io) {
                System.out.println("Issue closing the Files" +
io.getMessage());
            }
        }

    }
}
```

b. Create three files named ReadIt.txt, Numbers.txt and Strings.txt and place them in the Netbeans Projects/FileRead folder. (Note: adjust the folder to the location of your Netbeans projects)

The contents of the files can be the following:

ReadIt.txt:

```
12 30 90 1200
19 32 44 1900
20 32 22 1000
```

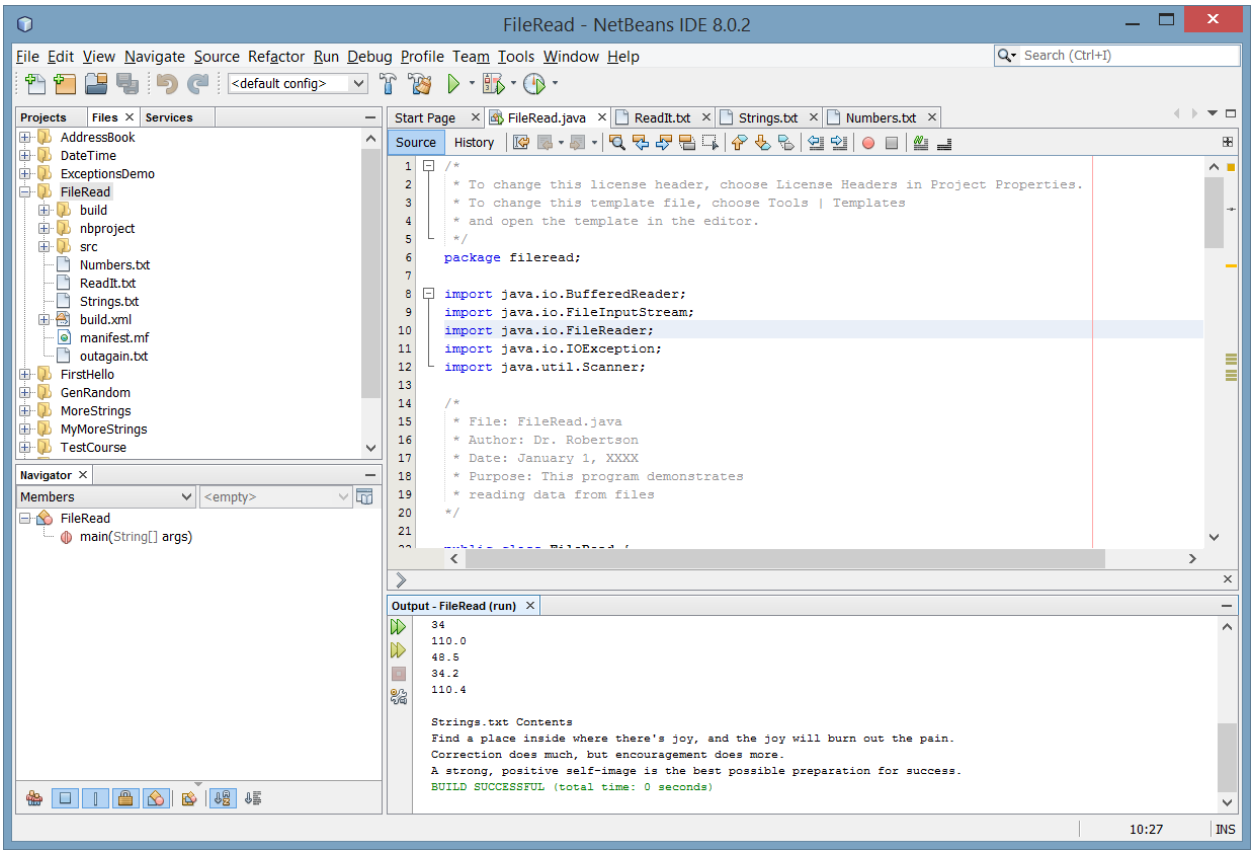
Numbers.txt:

```
4 12 30 100 43.2 43.2 90.2
3 10 34 110 48.5 34.2 110.4
```

Strings.txt:

```
Find a place inside where there's joy, and the joy will burn out the pain.
Correction does much, but encouragement does more.
A strong, positive self-image is the best possible preparation for success.
```

c. Compile and run the code by clicking on the green arrow in the IDE.



As you analyze and experiment with the code, note the following:

1. There are 2 separate instances of the try/catch statements. The first catches the file IO exceptions that may occur if any of the files were not available. The second set is used to check for exceptions when attempting to close the files streams within the finally clause:

```
finally {
   // Need another catch for closing
   // the streams
   try {
       // Close the streams
       if (in != null) {
           in.close();
       }
       if (inputStream != null) {
       inputStream.close();
   }
```

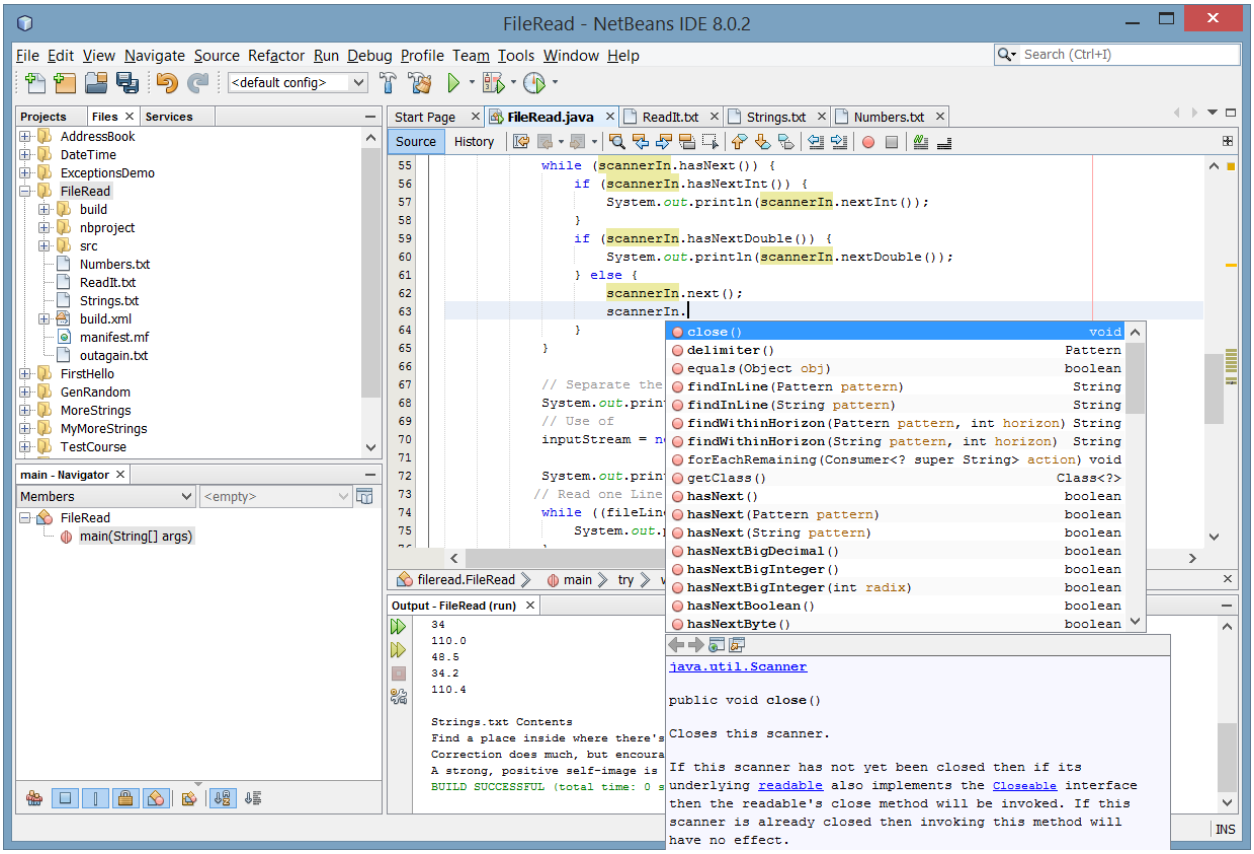
2. When we are reading from the FileInputStream, we must cast the output to characters or the results will be the ASCII code equivalent of the character

```
   while ((fileChar = in.read()) != -1) {
          // convert int to char
           System.out.print((char) fileChar);
   }
```

3. A BufferedReader is used to provide more methods when using both the Scanner class and the FileReader class. You can "daisy chain" multiple streams to provide more convenient processing.

```
   inputStream =  new BufferedReader(new FileReader("Numbers.txt"));
   scannerIn = new Scanner(inputStream);
```

4. Take advantage of the IDE for revealing the available methods in each of the streams. For example, after you type the period (.) after the reference all of the available methods are revealed.



5. Notice, the open streams are closed before exiting the application. Checking to make sure the stream is not null is needed to avoid possible run time exceptions if the stream was already closed.

```
if (in != null) {
        in.close();
 }
if (inputStream != null) {
       inputStream.close();
 }
```

Reading data from a file can be tricky since there are so many possible classes to support reading the data. Experiment with the code and create your own files to read to help better understand these different approaches.

Now it is your turn. Try the following exercise:

Using your IDE, create a project named ReadEmails. Create a text file to store email addresses and add at least 10 emails to the file. Save the file as EmailAddresses.txt. The Java application should open the EmailAddresses.txt file and read each email address echoing it to standard output.

Exercise 3 – Write data to a file using Java

Often, we need to write data to a file. In this exercise, we will read data from an existing file and then modify the data as appropriate and echo the resulting data to another file.

In this exercise, we will read up to 100 age values from a data file, copy the values to another file, and finally write the average age to another file.

Launch your IDE and create a new project named FileWrite.

a. Type, or copy and paste the following java code into the FileWrite.java file in your IDE.

```
package filewrite;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Scanner;

/*
 * File: FileWrite.java
 * Author: Dr. Robertson
 * Date: January 1, XXXX
 * Purpose: This program reads
 * ages from a file, copies the data
 * to another file write the average
 * age to another file.
 */
public class FileWrite {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Scanner scannerIn = null;
        FileInputStream in = null;
        BufferedReader inputStream = null;
        BufferedWriter outputStream = null;
        PrintWriter fileOut = null;
        // For summary data
        File summaryOut = new File("DataSummary.txt");
        // Use of Scanner and BufferedReader
        int[] ages = new int[100];
        // Counter for age
        int cnt = 0;
        int ageSum = 0;
        try {
            inputStream = new BufferedReader(new FileReader("Ages.txt"));
            // Open Copy Output file
```

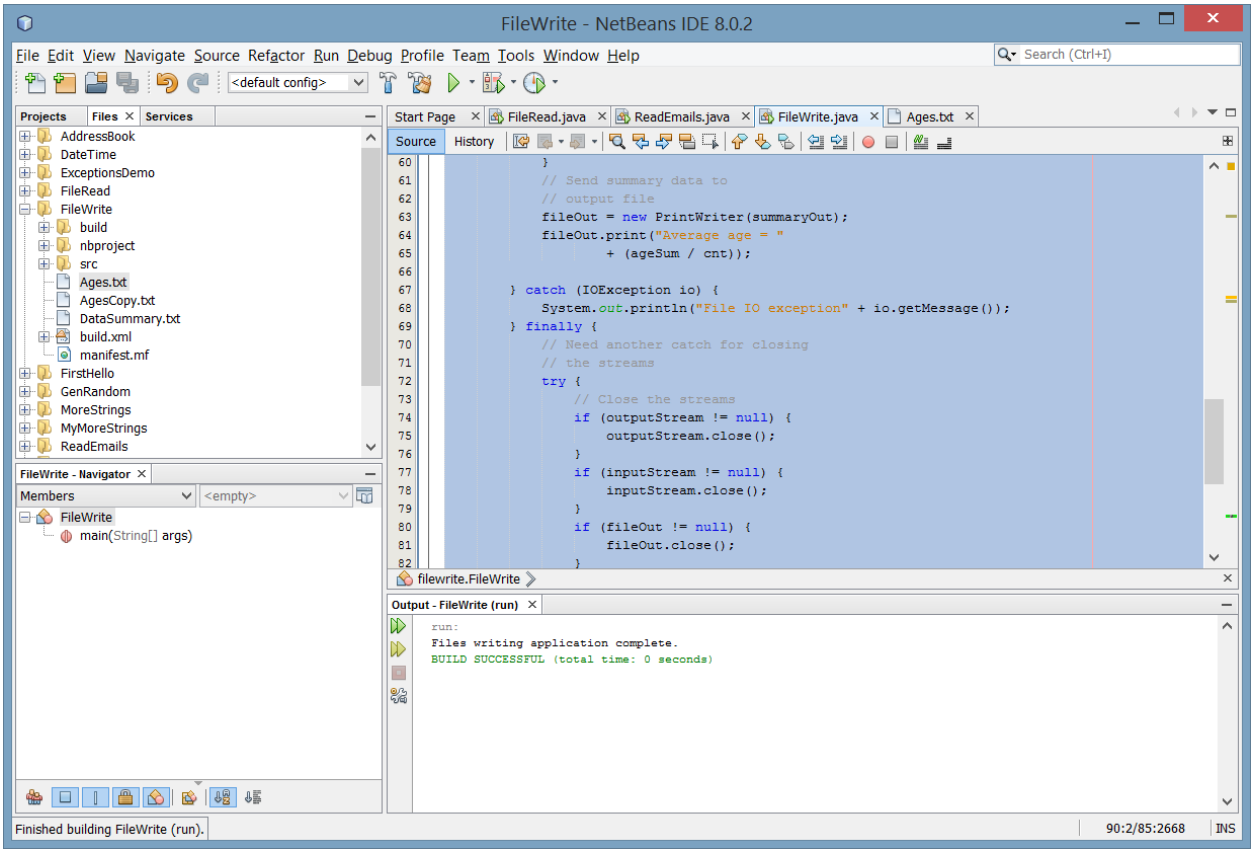
```
            outputStream = new BufferedWriter(new
FileWriter("AgesCopy.txt"));
            scannerIn = new Scanner(inputStream);
            while (scannerIn.hasNext()) {
                ages[cnt] = scannerIn.nextInt();
                // Write to output
                // \r\n provides string content and new line
                outputStream.write(ages[cnt] + "\r\n");
                // Increment ageSum
                ageSum += ages[cnt];
                // Increment counter
                cnt++;
            }
            // Send summary data to
            // output file
            fileOut = new PrintWriter(summaryOut);
            fileOut.print("Average age = "
                    + (ageSum / cnt));

        } catch (IOException io) {
            System.out.println("File IO exception" + io.getMessage());
        } finally {
            // Need another catch for closing
            // the streams
            try {
                // Close the streams
                if (outputStream != null) {
                    outputStream.close();
                }
                if (inputStream != null) {
                    inputStream.close();
                }
                if (fileOut != null) {
                    fileOut.close();
                }
            } catch (IOException io) {
                System.out.println("Issue closing the Files" +
io.getMessage());
            }
        }
        System.out.println("Files writing application complete.");

    }
}
```

b. Create a file named Ages.txt and place it in the Netbeans Projects/FileWrite folder. (Note: adjust the folder to the location of your Netbeans projects). Add up to 100 age values, 1 per line in the file. For example, this file contains 20 ages:

```
86
28
38
57
67
89
```

```
43
65
60
8
39
12
63
39
92
17
18
58
93
49
```

c. Compile and run the code by clicking on the green arrow in the IDE.



d. Verify that the AgesCopy.txt and DataSummary.txt contain a copy of the Ages.txt file and the average age, respectively.

As you analyze and experiment with the code, note the following:

1. We are using the Scanner object to read each age value in the file and OutputStream to write the age values to the CopyAge.txt file. Notice, we write a String the output file by concatenating the age value and carriage return , new line escape characters. We are also summing the ages and incrementing the number of ages in the file.

   ```
   while (scannerIn.hasNext()) {
   ```

```
            ages[cnt] = scannerIn.nextInt();
            // Write to output
            // \r\n provides string content and new line
            outputStream.write(ages[cnt] + "\r\n");
            // Increment ageSum
            ageSum += ages[cnt];
            // Increment counter
            cnt++;
        }
```

2. We use the PrintWriter class and a File class to write the summary data.

```
File summaryOut = new File("DataSummary.txt");
…
fileOut = new PrintWriter(summaryOut);
            fileOut.print("Average age = "
                    + (ageSum / cnt));
```

3. We close all of the file streams in the finally clause of the first try/catch statement

```
        try {
            // Close the streams
            if (outputStream != null) {
                outputStream.close();
            }
            if (inputStream != null) {
                inputStream.close();
            }
            if (fileOut != null) {
                fileOut.close();
            }
        }
```

Now it is your turn. Try the following exercise:

Using your IDE, create a project named WriteEmails. Create a text file to store email addresses and add at least 10 emails to the file. Save the file as EmailAddresses.txt. The Java application should open the EmailAddresses.txt file and read each email address echoing the addresses to a file named EmailCopy.txt