# ITSC 1213 - File I/O

## Introduction

In this lab you will experiment with the Java Input/Output components by implementing a program that reads input from a file and write output to a file. You will also practice handling exceptions (e.g., IOException ) with try-catch.

## Concepts covered in this lab:

- Reading in file contents
- Creating objects from file input
- “throws” keyword
- Writing out text to a file
- try/catch blocks

## Required files or links

- Code snippets linked in document.

## Preliminary

In previous labs we already have created some classes that represent different roles people can play in a university system. In those labs we also had a class with a main method that we used to test different aspects of our program (e.g., instantiating objects, ArrayLists, sorting,…). In this lab we will use a class with a main method to practice with reading data from a file that we will use to instantiate objects and then writing to a file.

For this lab you can use any of the previous labs where we have defined Student and Professor classes or you can create a new project and add the Student and Professor classes.

## Part A: Reading from File

For this lab we will be inputting data with the use of File and Scanner Objects. We’ve worked with Scanner objects before and File objects are not any different in that they are made just like any other object that we have seen before. The constructor for the File object takes a String that represents a path to a file that it wants to do input with. The path of the file can be **relative** or **absolute**. An example of a relative path would be, if a file resided in a project folder in NetBeans. Since this file is local to the project, the path name for this file would be the file name itself. However, if the file resides somewhere else, the absolute path would need to be given. An important note about the file path is that if the file does not exist/cannot be found, Java will throw an exception (we will deal with this later). Below is an example between local and absolute path for a file named Example.txt.

**Relative Path:**
```
    File fileObj = new File("Example.txt");
```

**Absolute Path:**
```
    File fileObj = new File("/Users/username/Documents/Example.txt");
```

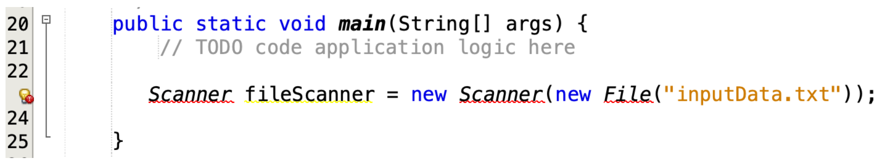
Once a File Object is initialized, methods can be used from its class to do input manipulation. However, we will not be using those methods in this lab, instead we will be using a Scanner. To learn more about File Objects review the Java Doc.

File and Scanner Objects work well together. As we have seen before, we can use a Scanner to parse input from a keyboard and save that input into variables. Scanners can also be used to parse through files. In order to do this, a Scanner object must be made with an initialized File or FileInputStream object in its constructor. By having this, the Scanner's methods such as next() and nextInt() can be used for parsing a file or the input stream. Shown below is two ways to initialize a File and Scanner object to parse a file. Both have the same result; one just has less lines of code.

**Option 1:**
```
    File fileObj = new File("Example.txt");
    Scanner fileScanner = new Scanner(fileObj);
```

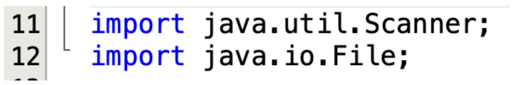
**Option 2:**
```
  Scanner fileScanner = new Scanner(new File("Example.txt"));
```

1) Comment out or delete the contents of the main method of the Module9.java file and add code to initialize a File and Scanner object from a file called inputData.txt.



```
public static void main(String[] args) {
    // TODO code application logic here

   Scanner fileScanner = new Scanner(new File("inputData.txt"));

}
```

2) Notice that we will need to add import statements for the Scanner and File classes. Go ahead and add them:

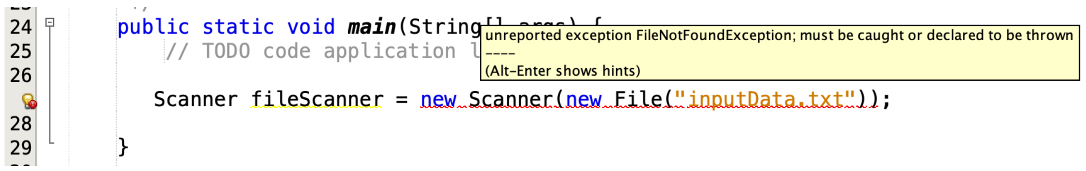


```
import java.util.Scanner;
import java.io.File;
```

3) When using File I/O there is some overhead in exception handling. When reading or writing to a file, exceptions can be thrown by Java. We've seen this before with exceptions like "NullPointerException" or "ArrayIndexOutOfBounds". This happens in Java when the program runs into a problem that disrupts the normal flow of the program. When this happens, Java does not have a direct way to fix it and therefore the program terminates followed by a message to the console of what went wrong. In File I/O, we

have to take this into account. This is mainly due to the exception “FileNotFoundException” can occur especially with File Objects.

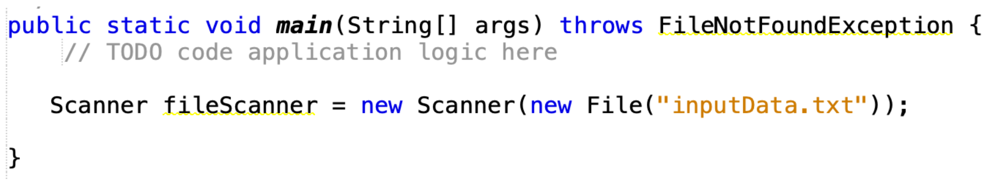
If you hover over the statements where we tried to initialize the Scanner object you will see that NetBeans is alerting us to this.



```
public static void main(String[] args) {
    // TODO code application l
    
    Scanner fileScanner = new Scanner(new File("inputData.txt"));
    
}
```

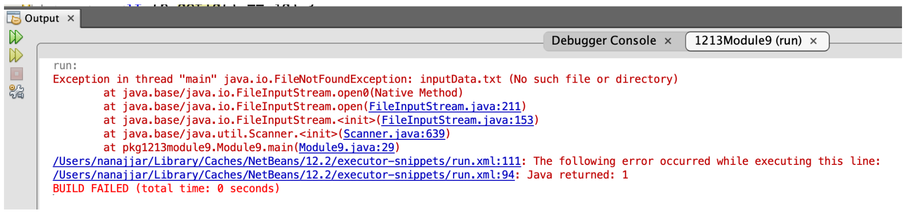
unreported exception FileNotFoundException; must be caught or declared to be thrown
----
(Alt-Enter shows hints)

4) Let’s go ahead and have the main method throw the exception by adding the throws keyword followed by the type of exception here, FileNotFoundException to method declaration. Notice where this is added; between the parameter (…) list and the body{…} of the method.



```
public static void main(String[] args) throws FileNotFoundException {
    // TODO code application logic here
    
    Scanner fileScanner = new Scanner(new File("inputData.txt"));
    
}
```

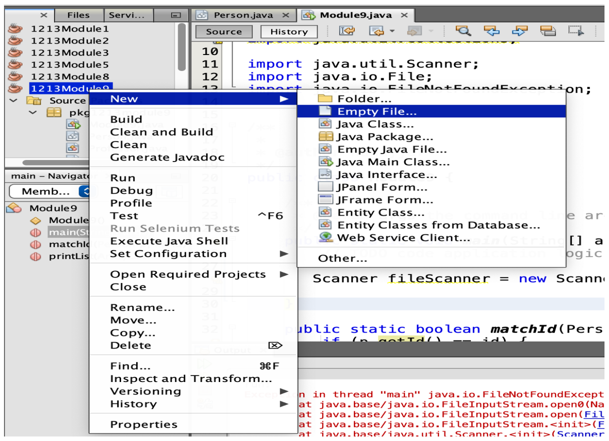
With this change what we are essentially saying is “this method is not prepared to handle the exception, go back please!”. When this is the case, the runtime system will search the call stack in reverse in search of a suitable exception handler. For example, if methodA calls methodB which calls methodC and an exception occurs during the execution of methodC, the runtime system will search methodC first, then methodB, and lastly methodA for an exception handler. When an appropriate exception handler is found, the runtime system turns over the exception object to the handler. The selected exception handler is said to ‘catch’ the exception. Here this is happening inside the main method so we are already at the top of the call stack. If the exception occurs the program will terminate. Run the program to see what happens. What can we do to fix this?



```
run:
Exception in thread "main" java.io.FileNotFoundException: inputData.txt (No such file or directory)
	at java.base/java.io.FileInputStream.open0(Native Method)
	at java.base/java.io.FileInputStream.open(FileInputStream.java:211)
	at java.base/java.io.FileInputStream.<init>(FileInputStream.java:153)
	at java.base/java.util.Scanner.<init>(Scanner.java:639)
	at pkg1213module9.Module9.main(Module9.java:29)
/Users/nanajjar/Library/Caches/NetBeans/12.2/executor-snippets/run.xml:111: The following error occurred while executing this line:
/Users/nanajjar/Library/Caches/NetBeans/12.2/executor-snippets/run.xml:94: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

5) There are actually a few things that should be done to handle this situation. To start we will make sure the file exists by adding the file to our project. You can do this just like what you do when you create a new file on your computer and saving it inside a particular folder. For example, you can use any text editor (e.g., Notepad, TextEdit, …) and save the file inside the project directory. We can also use NetBeans to create this file:

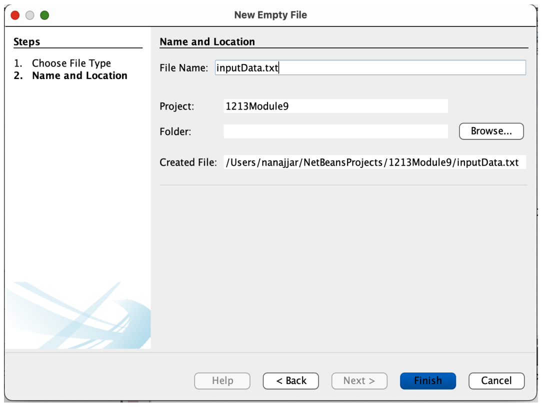
Right click the project and select New → Empty File. Insert the name of the file inputData.txt and click Finish



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

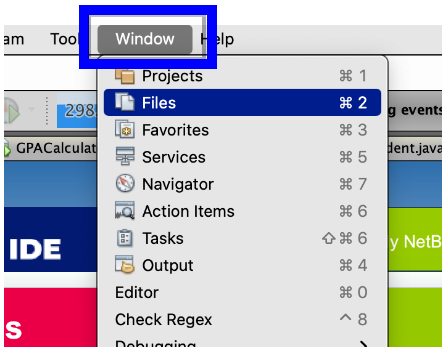
```
Scanner fileScanner = new Scann
```

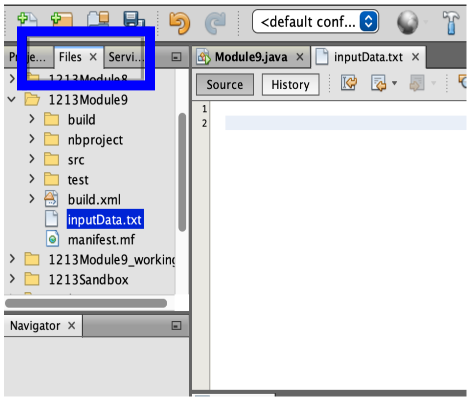
```
n in thread "main" java.io.FileNotFoundExcept
at java.base/java.io.FileInputStream.open0(Na
at java.base/java.io.FileInputStream.open(Fil
at java.base/java.io.FileInputStream.<init>(F
```



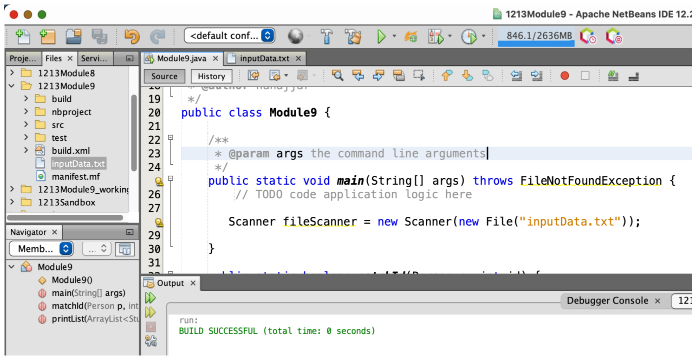
File Name: inputData.txt
Project: 1213Module9
Folder:
Created File: /Users/nanajjar/NetBeansProjects/1213Module9/inputData.txt

You should see the file listed under the project files tab as well as open in the editor tab. Make sure the file is saved in the main project folder not the src folder.  Open the Files tab to verify the file exists in the correct folder. If you do not see the Files tab you can open it from the Window menu





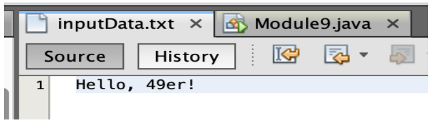
Now that the file exists, we don’t see any exceptions being thrown when we run the program.



```
 */
public class Module9 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws FileNotFoundException {
        // TODO code application logic here

        Scanner fileScanner = new Scanner(new File("inputData.txt"));

    }
```

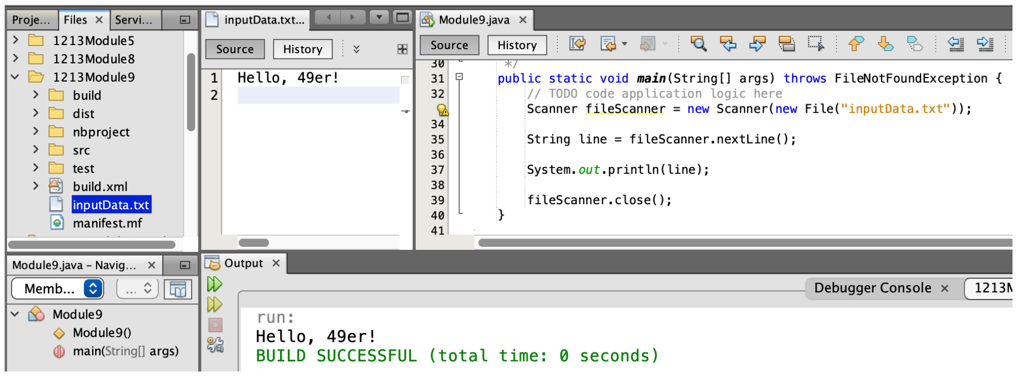
```
run:
BUILD SUCCESSFUL (total time: 0 seconds)
```

6)  Add one line of text to the inputData.txt file and save it.



```
Hello, 49er!
```

7) Now we can use the fileScanner object and its methods to read the contents of this file. We can iterate through the file line by line using the nextLine() method which returns a String type object of that line. Add statements to the main method to read the first line, then print it out and close it once we're done.



```
Hello, 49er!
```

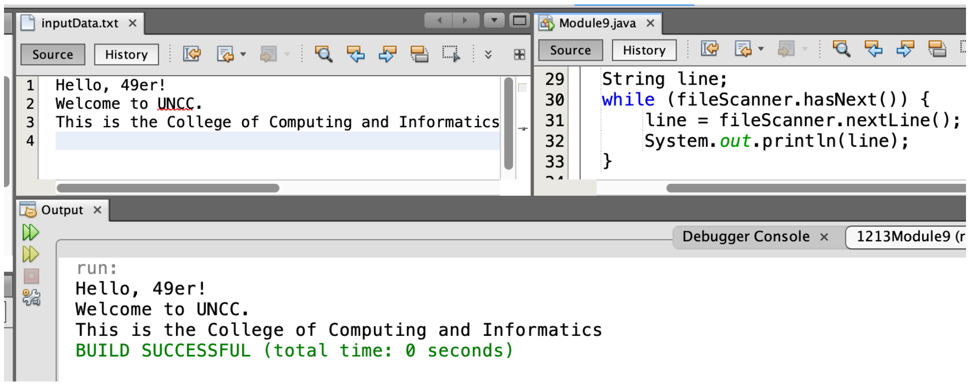
```java
public static void main(String[] args) throws FileNotFoundException {
    // TODO code application logic here
    Scanner fileScanner = new Scanner(new File("inputData.txt"));

    String line = fileScanner.nextLine();

    System.out.println(line);

    fileScanner.close();
}
```

```
run:
Hello, 49er!
BUILD SUCCESSFUL (total time: 0 seconds)
```

8) What about if we had more lines in the file to read? What if don't know how many lines the file has? Looking back at the prepwork we've seen how the hasNext() method which returns true if the scanner has more input can be used with a while loop to iterate and read the file line by line.

9) Let's go ahead and add one or more lines to our file and update the code to read the file line by line (iterate) and print it out.

10) Run your project to verify that everything works.



```
Hello, 49er!
Welcome to UNCC.
This is the College of Computing and Informatics
```

```java
String line;
while (fileScanner.hasNext()) {
    line = fileScanner.nextLine();
    System.out.println(line);
}
```

```
run:
Hello, 49er!
Welcome to UNCC.
This is the College of Computing and Informatics
BUILD SUCCESSFUL (total time: 0 seconds)
```

11) Now that we know we can access our file and read its lines we are ready to use this data in our program.

12) When your program works and you are satisfied with the result, show your work to the instructional team to get checked off and proceed to Part B.

## Part B: Creating Objects Using Input from Files

1) We now want to use input to read from a file to instantiate objects. Let's start by copying the following lines that represent the data to create Student and Professor objects into this new file. Make sure you save the file when you're done copying.

```
student,Xavier,Cato,900111222,CS,3.5,75
student,Kathrine,Johnson,900,CS,4.0,100
student,Roy,Clay,901,Biology,3.2,85
student,Kimberly,Bryant,902,Electrical Engineering,3.0,80
professor,Mary,Castro,300,CS,80000.0
professor,Frank,Black,801,Math,85000.0
student,Grace,Maxeem,903,Psychology,3.4,95
professor,Kayla,Wilson,301,CS,95000.0
```

2) Now in the main method you need to
   a. Create a list (ArrayList) to store Student objects.

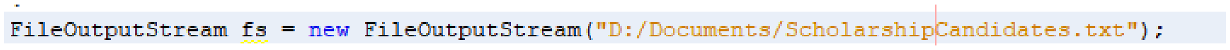
   b. Read the lines of this file and if the line represents student data create a new Student object and add it to the list. (Hint! You may want to review String methods to help you separate the values within each line and wrapper classes to convert Strings to different data types)

   c. Sort the list of students based on their GPA. (Hint! We did this in a previous lab (module 9)
   d. Print the list of students to verify that your logic and program works. (Hint! see the previous hint!)

When your program works and you are satisfied with the result, show your work to the instructional team to get checked off and proceed to Part C.

## Part C: Save to file

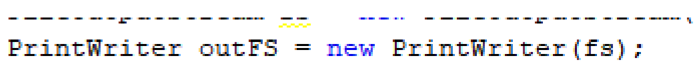
1) Now that we have a sorted list of Students from Part B, step 4, let's practice writing this information back out to another file. Let's assume that this list of students are candidates for a scholarship, and it is our job to report out a list of candidates sorted into three categories, Top, Middle, and Ineligible. Let's assume that our top candidates have a GPA >= 3.5, our Middle candidates have a GPA less than 3.5 but greater or equal to 3.2, and our ineligible students have a GPA less than 3.2
2) In order to write out to a file, we need to first create two new objects, a FileOutputStream and a PrintWriter. The FileOutputStream is what will create a file and direct the output, and the PrintWriter will allow us to print Strings into our newly created file.
3) The FileOutputStream requires one String argument: a filename (and optionally the path where we want to put that file). If you don't specify an absolute path the file will be created in the project folder (the same folder inputData.txt is saved under). If you choose to provide an absolute path, pick an easy to locate path on your computer, like the Desktop or Documents folder to create our new text file in. (Note, the path will likely look different for you, so don't attempt to copy this exactly)



```java
FileOutputStream fs = new FileOutputStream("D:/Documents/ScholarshipCandidates.txt");
```

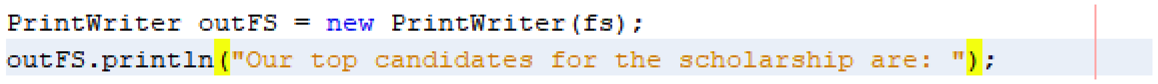
4) Next, we need to create our PrintWriter. This takes one argument, our FileOutputStream

reference.



```
PrintWriter outFS = new PrintWriter(fs);
```

5) Now that we have these two components, we can start by constructing the lines we want to print into our new file. This is very similar to how we used System.out.print() before, only now we are printing into a file instead of to the system.



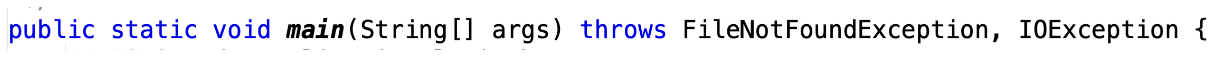
```
PrintWriter outFS = new PrintWriter(fs);
outFS.println("Our top candidates for the scholarship are: ");
```

6) Using our sorted list of students, construct a loop that will traverse the list and print out relevant student information, such as their name, GPA, and major into the appropriate category. This will likely take some trial and error to get your formatting right, so once you have your first attempt at print statements done, continue to the next step and we'll revisit your formatting in a moment.

7) To tell Java that we are finished writing, we must close our FileOutputStream and PrintWriters. Your text may not be saved to the file until you use the close method. After your code for writing out to the file is finished, add the following two lines:

```
outFS.close();
fs.close();
```

Notice that NetBeans is now warning us about an exception (IOException) the added code introduced. IO stands for input-output, and an IOException is thrown whenever some sort of input-output error happens. For now, we will just throw this exception similar to what we did with the FileNotFoundException. We will catch them in Part D.



```
public static void main(String[] args) throws FileNotFoundException, IOException {
```

8) You can now run your project and see what your formatting looks like. Remember our goal here is to create a file that reports candidates for a scholarship, so ask yourself if that task is accomplished well. Is the formatting clean? Is each section (Top, Middle, Ineligible) clearly defined? Have I included all of the necessary information? If you're happy with your formatting, congratulations! You're done! If not, make the necessary corrections and try again.

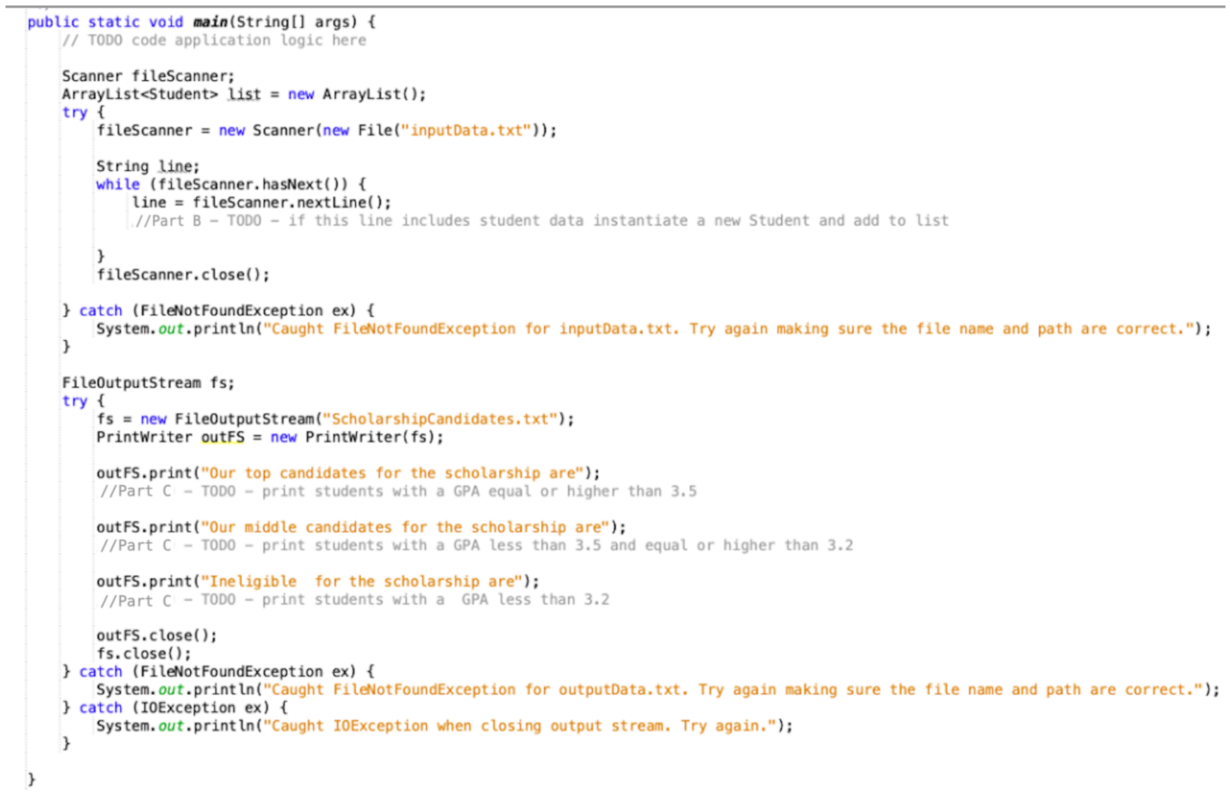
9) When your program works and you are satisfied with the result, show your work to the instructional team to get checked off and proceed to Part D.

## Part D: Exception Handling

1) In part A we mentioned that there are several things that need to be done to handle the FileNotFoundException that was thrown. Currently our program will still give an error if we try and use an input file that is not there (does not exist!). When we write a program one of our goals should be to ensure that all possible errors and exceptions are handled somewhere not just thrown.

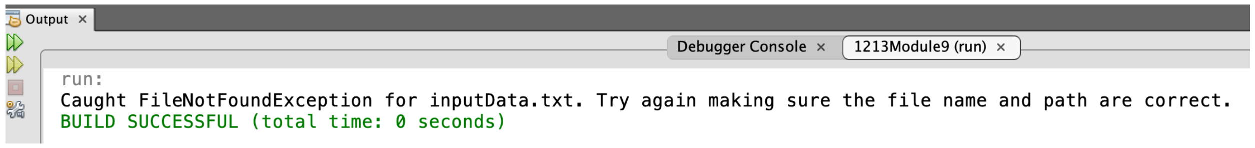
Remove the “throws” statement you added earlier. Notice NetBeans will now once again warn you about uncaught exceptions.

2) Add in try and catch blocks to handle these exceptions, making sure to catch the specific type of exception. Have the catch block print a helpful message explaining what happened.



```
public static void main(String[] args) {
    // TODO code application logic here

    Scanner fileScanner;
    ArrayList<Student> list = new ArrayList();
    try {
        fileScanner = new Scanner(new File("inputData.txt"));

        String line;
        while (fileScanner.hasNext()) {
            line = fileScanner.nextLine();
            //Part B - TODO - if this line includes student data instantiate a new Student and add to list

        }
        fileScanner.close();

    } catch (FileNotFoundException ex) {
        System.out.println("Caught FileNotFoundException for inputData.txt. Try again making sure the file name and path are correct.");
    }

    FileOutputStream fs;
    try {
        fs = new FileOutputStream("ScholarshipCandidates.txt");
        PrintWriter outFS = new PrintWriter(fs);

        outFS.print("Our top candidates for the scholarship are");
         //Part C - TODO - print students with a GPA equal or higher than 3.5

        outFS.print("Our middle candidates for the scholarship are");
         //Part C - TODO - print students with a GPA less than 3.5 and equal or higher than 3.2

        outFS.print("Ineligible  for the scholarship are");
         //Part C - TODO - print students with a  GPA less than 3.2

        outFS.close();
        fs.close();
    } catch (FileNotFoundException ex) {
        System.out.println("Caught FileNotFoundException for outputData.txt. Try again making sure the file name and path are correct.");
    } catch (IOException ex) {
        System.out.println("Caught IOException when closing output stream. Try again.");
    }

}
```

3) To test how your program behaves with the try-catch blocks try using a file name and path for the input or output that does not exist (e.g., wrongFile.txt).



```
run:
Caught FileNotFoundException for inputData.txt. Try again making sure the file name and path are correct.
BUILD SUCCESSFUL (total time: 0 seconds)
```

4) When your program works and you are satisfied with the result, show your work to the instructional team to get checked off.

You’re done with this lab!

## Bonus: More Exception Handling

Depending on how you implemented the logic for creating the Student objects in Part C it is likely you used an array reference when you split the line to access the values needed for each student. What if we add a line to our input file that does not include the same information as the other students we have in there? For example, a line with just the first and last name (student,Sarah,Shay).

Try it and see if any exceptions are thrown. If so, add in try and catch blocks to handle these exceptions, making sure to catch the specific type of exception.
Include a note with your submission indicating that you handled this exception.

**You're done!**

**So, what did you learn in this lab?**

- **Reading in file contents**
- **Creating objects from file input**
- **"throws" keyword**
- **Writing out text to a file**
- **try/catch blocks**

## <u>Lab Canvas submission</u>

In this week's module, you will find the link to the lab completion submission. Submit the following files:

a. Screenshots for parts A, B, D
b. NetBeans project export
c. The output text file from Part C
d. Did you complete the bonus? Let us know.