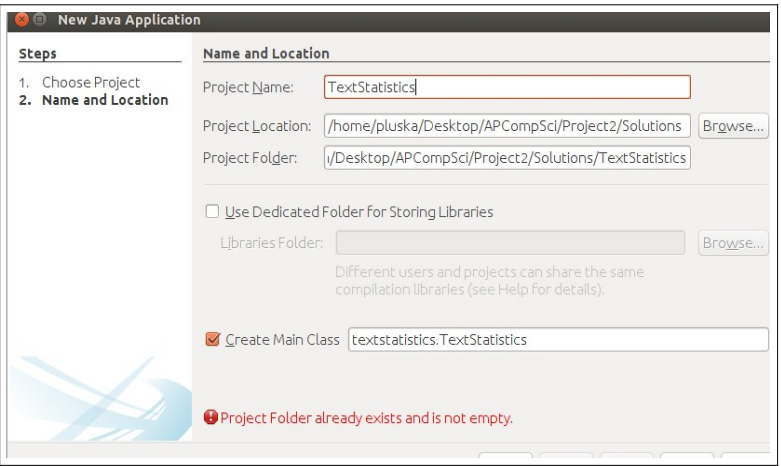
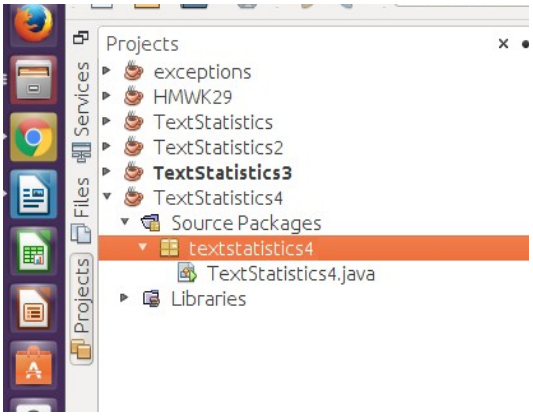
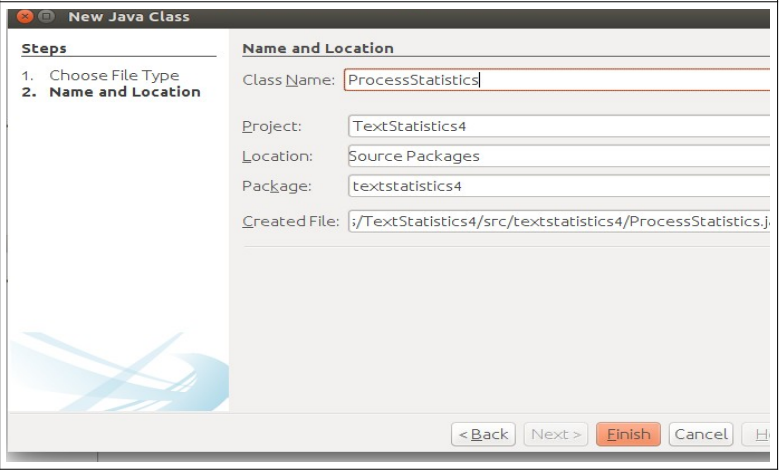
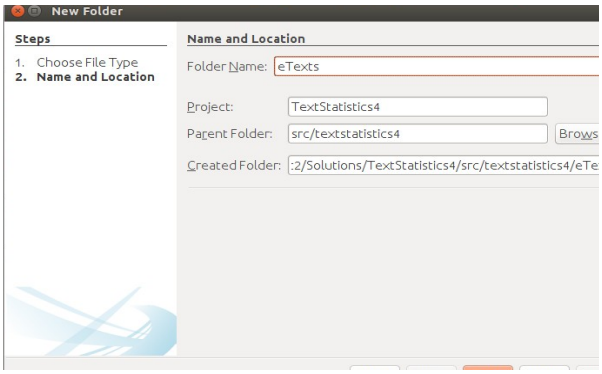
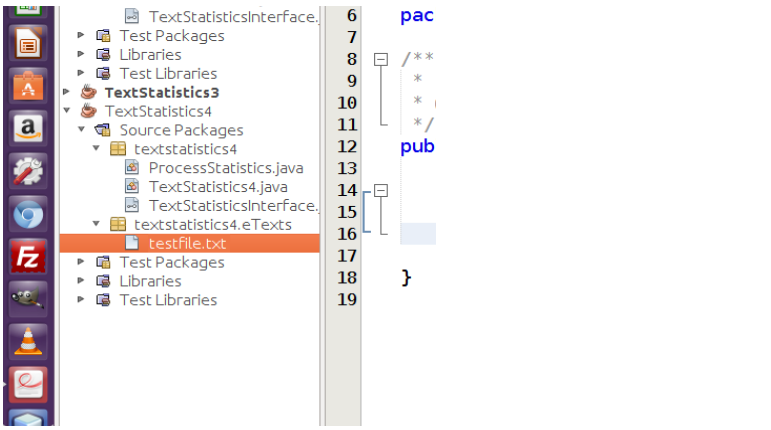
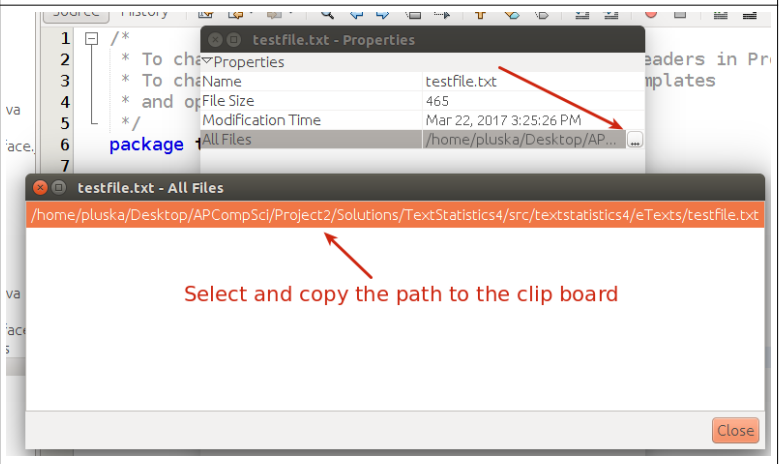
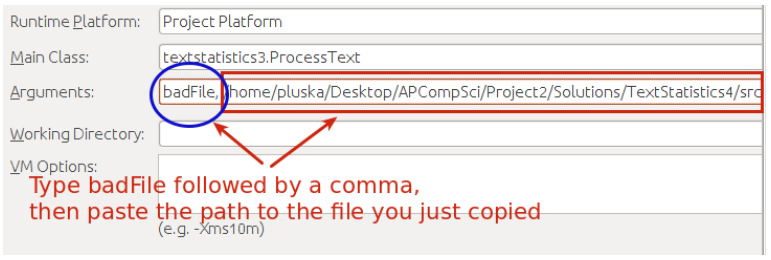
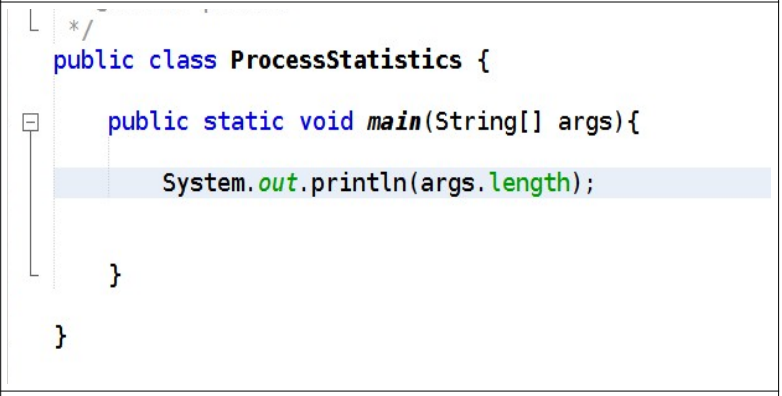
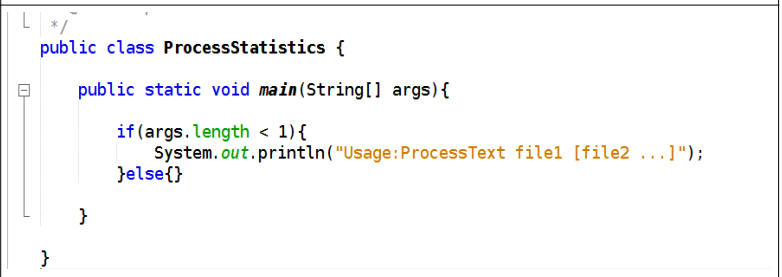
## Project 2: Getting Started

| | |
|---|---|
| Open NetBeans. Create a java project called “TextStatistics” |  |
| Navigate to “TextStatisticsInterface.java” in your downloads directory. Copy this file. Then paste it in the TextStatistics project. |  |
| Right click on your project package “textStatistics”. Select New → Java class.<br><br>Call the class “ProcessStatistics” |  |

## Project 2: Getting Started

---

| | |
|---|---|
| The arguments your program will accept are text files. Create a folder for the text files. Right click on the project package and select new → folder<br><br>Title the folder eTexts. |  |
| Place the required text files in this folder |  |
| Right click on the testfile and select properties. Click on the icon to the far right of the "All files" option. Select the file path in the popup window and copy this to your clip board. |  |
| To configure your project to accept arguments, go to Run → set project configuration → customize<br><br>In the arguments text box we need to add an array of arguments for our program to accept. The first argument will be "badFile" because we want to make sure our program tests for bad files. The |  |

| | |
|---|---|
| second argument will be the file path you copied to our clip board.<br><br>Type “badFile,” then after the comma paste the contents of your clipboard. | |
| To test whether or not everything is working, return to the processStatistics class and add<br><br>System.out.println(args.length);<br><br>to the main method.<br><br>Run your program. The number “2” should display. |  <br>*/<br>public class ProcessStatistics {<br>public static void main(String[] args){<br>System.out.println(args.length);<br>}<br>} |
| The first thing your project should test for is whether or not the user has passed in an argument. This can be done by checking the length of the args array. If the length is less than 1, no arguments are present and we need to print the usage statement.<br><br>Modify the main method to include this check by replacing the<br><br>System.out.println(args.length);<br><br>with,<br><br>if(args.length < 1){<br><br>System.out.println("Usage:ProcessText file1 [file2 ...]");<br><br>}else{} |  <br>*/<br>public class ProcessStatistics {<br>public static void main(String[] args){<br>if(args.length < 1){<br>System.out.println("Usage:ProcessText file1 [file2 ...]");<br>}else{}<br>}<br>} |

## Project 2: Getting Started

---

If an argument has been passed, we can create a File object out of the argument.

For this project we need to create File objects out of all the arguments, then process the associated statistics. To do this we will need a loop.

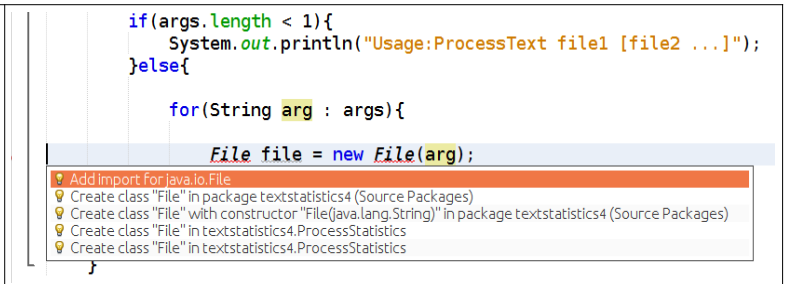
Add the following loop to the else clause. This loop will iterate through all the arguments passed (args) and will turn each into a File object.

Notice you have an error, we will fix that next.

```
 */
public class ProcessStatistics {

    public static void main(String[] args){

        if(args.length < 1){
            System.out.println("Usage:ProcessText file1 [file2 ...]");
        }else{

            for(String arg : args){

                File file = new File(arg);

            }
```

space

To fix the error, click on the light bulb and import the required File library



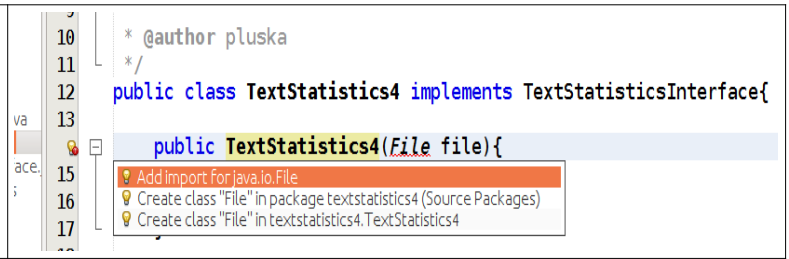

The bulk of the code for this project will take place in the TextStatistics constructor. Go to the TextStatistics class. Create a constructor which accepts a File object as a parameter.

Add the File library import just as you did before.

```
 * @author pluska
 */
public class TextStatistics4 implements TextStatisticsInterface{

    public TextStatistics4(File file){


    }
```

## Project 2: Getting Started

---




With our TextStatistics constructor written, we now can create objects with our new Files. Return to the ProcessStatistics class and add the following to the else clause,

TextStatistics ts =
new TextStatistics(file);

```
if(args.length < 1){
    System.out.println("Usage:ProcessText file1 [file2 ...]");
}else{

    for(String arg : args){

        File file = new File(arg);
        TextStatistics4 ts = new TextStatistics4(file);

    }
```

Now return to the TextStatistics class.

In order to process our File we need to retrieve the file that was passed from the TextStatistics class and create a Scanner to scan its contents.

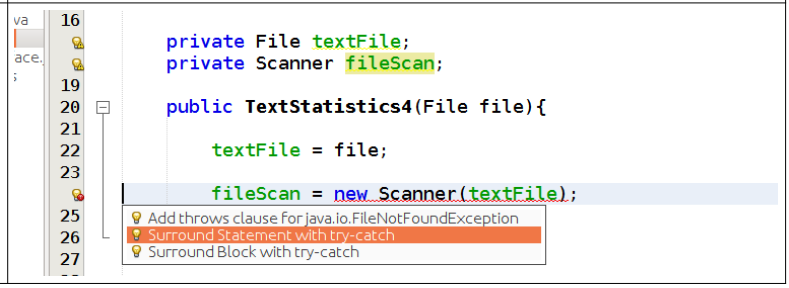
Create a File instance variable called "textFile" and a Scanner instance variable called "fileScan". These should be declared as private as shown right.

In the constructor, assign the parameter file to textFile and create a new Scanner with the textFile.

Notice we have an error...

```
 * @author pluska
 */
public class TextStatistics4 implements TextStatisticsInterface{

    private File textFile;
    private Scanner fileScan;

    public TextStatistics4(File file){

        textFile = file;

        fileScan = new Scanner(textFile);

    }
```

The error occurs, because java needs you to check whether or not the file to be scanned is valid. You can do this with a try-catch. Click on the light bulb next to the error and select "Surround statement with try-catch"



## Project 2: Getting Started

---

In the catch clause, delete the default error message and replace with your own as shown right.

```
try {
    fileScan = new Scanner(textFile);
} catch (FileNotFoundException ex) {
    //Logger.getLogger(TextStatistics4.class.getNa
    System.out.println("File cannot be located");
}
```

Now that we have our file loaded in our scanner we can scan it for information. For example, the number of lines of text.

Create a new state variable called lineCount.

In the try clause, add the following,

```
while(fileScan.hasNextLine()){
        fileScan.nextLine();
        lineCount++;
}
```

In the above code, "hasNextLine()" is a boolean. It checks whether or not there is another line of code in the document. If there is we increment lineCount and go to the next line (fileScan.nextLine()). We continue this process until there are no more lines.

```
private File textFile;
private Scanner fileScan;
private int lineCount;

public TextStatistics4(File file){

    textFile = file;

    try {
        fileScan = new Scanner(textFile);

        while(fileScan.hasNextLine()){
            fileScan.nextLine();
            lineCount++;
        }

    } catch (FileNotFoundException ex) {
```

---

Now that we have counted all the lines, we need a method which allows the main driver method access to the value. Notice that we declared lineCount as private. As a private variable the main method cannot access the value.

Go to the method "getLineCount()". This was one of the abstract methods we implemented from the interface. Delete the default code and add the return statement, "return lineCount". Because this is a public method, the main method can access it.

```
@Override
public int getLineCount() {
    //throw new UnsupportedOpe
    return lineCount;
}
```

Delete the default code
add a return value

## Project 2: Getting Started

---

Return to the ProcessStatistics class.

The project description requires that we print out the stats for the valid files only. To do this, we must first check whether or not the file of interest exists. If it does we then can show the stats. In this tutorial, you learned how to count the total lines in a text document. To print out the total lines associated with our valid file, add the following lines of code to the the for loop,

if(file.exists())
{
System.out.println(ts.getLineCount());
}

```
for(String arg : args){

    File file = new File(arg);
    TextStatistics4 ts = new TextStatistics4(file);

    if(file.exists()){
        System.out.println(ts.getLineCount());
    }
```

Now, run your program...

The first file in are arguments, “badFile”, cannot be located, but the second file was analyzed. It has 11 lines of code. To check if you are correct, click on the file in your project view.