[**Click Here for Java Programming I Portfolio Help**](https://docs.google.com/document/d/16bY04kv2evDwSyr60l6W0PdsEiov1l7LeEgVWh8ub88/edit?usp=sharing)

**“The only way to learn a new programming language is by writing programs in it.”**

*- Dennis Ritchie, creator of C & UNIX*

To help you in creating your portfolios, I’ve included instructions for each portfolio and linked helpful resources and tips, where applicable. Please use the hyperlinked table of contents to navigate around this document easily.

Want to write **good code** and get a good grade on your programs? Read our [STYLE GUIDE](https://drive.google.com/open?id=1_p0202b1BkaWo-joUBfKrTCunAiJQach).

## 

## **Table of Contents**:

|  | \*\*these are hyperlinked - click on them to go to a particular portfolio!\*\*  [Table of Contents:](#_3y10z23ri8tz)  [Rubric](#_ptbz7iv8pevj)  [Lesson 6 Portfolio: Exception Handling & Text I/O](#_wtrywth48tql)  [Portfolio Overview](#_d3cxpw5a62zh)  [Portfolio Goal](#_b57kk7us1k5w)  [Portfolio Instructions](#_xc1bb6oth4rk)  [Getting started:](#_mar27ouqnv1s)  [DataWriter Class](#_9g7jwmz8i9il)  [DataReader Class](#_7bn71zipinpi)  [Submission Instructions](#_ob5ucao3eywi)  [Lesson 8 Portfolio: Abstract Classes and Interfaces](#_hgi0anqtohlv)  [Portfolio Overview](#_xpbwidpsz6ry)  [Portfolio Goal](#_ifyn1vgt7kxk)  [Portfolio Instructions](#_rdzft3ts1n6d)  [Submission Instructions](#_wuoct5pabckw)  [Lesson 13: Sets & Maps](#_32dqovgkyz0t)  [Portfolio Overview](#_p4nskg6w7dz5)  [Portfolio Goal](#_g3bjof498tvw)  [Portfolio Instructions](#_4lgcyiovvxsn)  [Importing Files into Eclipse](#_t6kfledvdykf)  [Portfolio Introduction & Algorithm Overview](#_x7gvkfl4p38q)  [CharCountingStudy Class](#_4rs5so54i0y5)  [FileExaminer Class](#_b26lwdifrhz6)  [CharCount Class](#_d3gy8cbzwvby)  [FileExaminer Class, revisited](#_y0qvxgajlu4p)  [Submission Instructions](#_ggxgf4t6gnvp) |
| --- | --- |

## 

## Rubric

We will use the same general rubric for all portfolios.

|  | Full credit | Partial credit | No Credit |
| --- | --- | --- | --- |
| Content | * Comments are used appropriately * Code executes as instructed * Correct number of functions are implemented * Output is as expected | * Sparse or unclear comments * Code executes but is missing some tasks in assignment * Some functions implemented/too few functions implemented * Output is mostly as expected | * No comments * Code does not execute what is asked in assignment * No functions implemented or run * Output not what’s expected or missing |
| Organization | * Code is well organized * Classes are appropriately organized into files * Files submitted correctly | * Code needs to be broken up into smaller subroutines * Classes & functions not in correct files | * Poorly organized functions * All code in one file when it should be in multiple separate files |
| Mechanics | * Code compiles * Program executes as expected * No syntax errors | * Code compiles but program does not execute * OR * Code does not compile due to a few minor syntax errors | * Code does not compile due to many major syntax errors |

# Lesson 6 Portfolio: Exception Handling & Text I/O

## Portfolio Overview

By completing this portfolio item, you will develop a greater understanding of file I/O using the **File**, **Scanner**, and **PrintWriter** classes.

## Portfolio Goal

You will be creating two small programs. One will write a text multiplication table to a file using a PrintWriter object. The second will verify that the newly created file exists and then read its contents using a Scanner object. You will need to demonstrate your knowledge of nested loops, exception handling, and various file I/O classes in order to complete this portfolio item.

## Portfolio Instructions

### Getting started:

1. In your Eclipse project, create a new package named lesson6.
2. Inside lesson6, create a new class named **DataWriter**. This class will contain code that will write data to a file.
3. Inside lesson6, create a new class named **DataReader**. This class will contain code that will read from the file.
4. Proceed with the instructions for each class.

#### DataWriter Class

1. Add the following import statements to your class. These imported classes will be needed in the code you will be writing.

import java.io.File;  
import java.io.FileNotFoundException;  
import java.io.PrintWriter;

1. Create a main() method inside the DataWriter class. All of the following code instructions for this class will be placed inside this method.
2. Create a File object using "data.txt" as the argument to its constructor. Store the reference to the new File object in a variable of type File.
3. Create a PrintWriter object using the reference to the File object you just created as an argument to its constructor. Store the reference to the new PrintWriter object in a variable of type PrintWriter. You might notice that Eclipse will indicate a problem if the code creating a PrintWriter object is not handled by a try-catch block. Supply the appropriate exception handling using either a normal try-catch block or a new try-with-resources statement. Supply a useful error message in the catch block using a System.out.println() statement.
4. Add a System.out.println() statement that displays the text, "Writing to file." This statement will provide visual output in Eclipse so that you can see that the program is actually running.
5. Use the PrintWriter object to write content to the file using print() and println()statements on the PrintWriter object instead of System.out. For example, to write “Hello, Fred!” to the file, if your PrintWriter reference is named p, the code would look like this:

p.println("Hello, Fred!");

1. Do not actually write “Hello, Fred!” to the file. Instead, you are going to write a multiplication table. The table spans from 1–10 in rows and columns. For the first row, write code that produce output that looks like this:

1x1=1 1x2=2 1x3=3 1x4=4...

1. Use a nested loop to loop through rows and columns to produce the desired output. Each element (such as 5x3=15) should be generated using the row number, column number, and their product. The finished output should look like this:

1x1=1 1x2=2 1x3=3 1x4=4 1x5=5 1x6=6 1x7=7 1x8=8 1x9=9 1x10=10   
2x1=2 2x2=4 2x3=6 2x4=8 2x5=10 2x6=12 2x7=14 2x8=16 2x9=18 2x10=20   
3x1=3 3x2=6 3x3=9 3x4=12 3x5=15 3x6=18 3x7=21 3x8=24 3x9=27 3x10=30   
4x1=4 4x2=8 4x3=12 4x4=16 4x5=20 4x6=24 4x7=28 4x8=32 4x9=36 4x10=40   
5x1=5 5x2=10 5x3=15 5x4=20 5x5=25 5x6=30 5x7=35 5x8=40 5x9=45 5x10=50   
6x1=6 6x2=12 6x3=18 6x4=24 6x5=30 6x6=36 6x7=42 6x8=48 6x9=54 6x10=60   
7x1=7 7x2=14 7x3=21 7x4=28 7x5=35 7x6=42 7x7=49 7x8=56 7x9=63 7x10=70   
8x1=8 8x2=16 8x3=24 8x4=32 8x5=40 8x6=48 8x7=56 8x8=64 8x9=72 8x10=80   
9x1=9 9x2=18 9x3=27 9x4=36 9x5=45 9x6=54 9x7=63 9x8=72 9x9=81 9x10=90   
10x1=10 10x2=20 10x3=30 10x4=40 10x5=50 10x6=60 10x7=70 10x8=80 10x9=90 10x10=100

1. Use your knowledge of nested loops, print(), and println() statements to write this output to the file. Be sure to add spaces between each element as shown above. ***Tip!*** *Check out the code example on* [*page 176*](https://media.pearsoncmg.com/pls/in/connections/1269657666/ebookCM84087197_2/1.0/htmls/page_176.html) *of your textbook (note that you can use normal print() or println() rather than printf()).*
2. Add a System.out.println() statement that displays the text, "Finished writing to file."
3. Right-click on the DataWriter class and run it as a Java application. Correct any errors that occur.
4. If your program ran without errors, it should have created and written to a file named “data.txt”. You can find this file in the root of your Eclipse project (not package). Select the project (above the src directory) and then press the F5 key. This will refresh the display and you should see the data.txt file at the bottom of the Package Explorer window. Open the file to confirm that it contains the multiplication table shown above.

#### DataReader Class

1. Add the following import statements to your class. These imported classes will be needed in the code you will be writing.

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

1. Create a main() method inside the DataReader class. All of the following code instructions for this class will be placed inside this method. This is a second, separate program that will read the contents of the file you just created.
2. Create a new File object like before using "data.txt" as the argument to its constructor.
3. Display the text, "File exists?" with a space after the question mark. On the same line, display the results of the exists() method on the File object.
4. Create an if-else statement. If the file exists, read the contents of the file (discussed in steps six and seven). If it does not exist, display the text, "Nothing to read."
5. Inside the if statement, create a new Scanner object using the reference to the Fileobject you just created as an argument to its constructor. Handle the exceptions of the Scanner in a try-catch block or a try-with-resources statement.
6. Use a while loop and the hasNext() and nextLine() methods of the Scanner class to loop through the content of the file and display its contents one line at a time.
7. Run the DataReader class as a Java application. If everything works, your output should look like the following text. You will be asked to copy and paste this output into a document as part of your assignment.

File exists? true  
1x1=1 1x2=2 1x3=3 1x4=4 1x5=5 1x6=6 1x7=7 1x8=8 1x9=9 1x10=10   
2x1=2 2x2=4 2x3=6 2x4=8 2x5=10 2x6=12 2x7=14 2x8=16 2x9=18 2x10=20   
3x1=3 3x2=6 3x3=9 3x4=12 3x5=15 3x6=18 3x7=21 3x8=24 3x9=27 3x10=30   
4x1=4 4x2=8 4x3=12 4x4=16 4x5=20 4x6=24 4x7=28 4x8=32 4x9=36 4x10=40   
5x1=5 5x2=10 5x3=15 5x4=20 5x5=25 5x6=30 5x7=35 5x8=40 5x9=45 5x10=50   
6x1=6 6x2=12 6x3=18 6x4=24 6x5=30 6x6=36 6x7=42 6x8=48 6x9=54 6x10=60   
7x1=7 7x2=14 7x3=21 7x4=28 7x5=35 7x6=42 7x7=49 7x8=56 7x9=63 7x10=70   
8x1=8 8x2=16 8x3=24 8x4=32 8x5=40 8x6=48 8x7=56 8x8=64 8x9=72 8x10=80   
9x1=9 9x2=18 9x3=27 9x4=36 9x5=45 9x6=54 9x7=63 9x8=72 9x9=81 9x10=90   
10x1=10 10x2=20 10x3=30 10x4=40 10x5=50 10x6=60 10x7=70 10x8=80 10x9=90 10x10=100

## Submission Instructions

For this portfolio you should include the following 3 files:

1. DataWriter.java
2. DataReader.java
3. data.txt

To submit your portfolio for grading, find your Java source files & text files in your Project folder within your Eclipse workspace (a folder you created for Eclipse to save projects when you first installed). Copy & paste your Java source files & text files to a new folder. Compress the folder so that it has the extension .zip. Upload this zip file to the portfolio dropbox.

If you need help with submitting per the above requirements, [please click here](https://docs.google.com/document/d/1stEEnJIoVx8pZSAZy-U8sLtvZylyzskrbgM6rCn7m08/edit?usp=sharing).

**\*\*\*Please note I will not be accepting a word document with code and your output for this portfolio. If you submit a word document, you will receive a zero. You will be able to resubmit properly for up to full credit if this happens.\*\*\***

# Lesson 8 Portfolio: Abstract Classes and Interfaces

## Portfolio Overview

\*\*\* Please note that you should **complete the activity in Lesson 7** before beginning this portfolio!! **Lesson 7 slides 5 - 9**. If you are not completing the activities in the lesson for practice, the portfolios will feel significantly more difficult.\*\*\*

By completing this portfolio item, you will develop a greater understanding of interfaces, particularly the Comparable interface. You will also gain experience with ArrayList, Collections.sort(), and the toString() method.

## Portfolio Goal

You will further improve upon the program from Lesson 4 and Lesson 7 by implementing the **Comparable** interface and allowing various Shape objects to be stored in a single data structure and sorted by relative area.

## Portfolio Instructions

1. Create a new package in your Eclipse project named lesson8.
2. Inside lesson8, create classes Shape, Rectangle, RightTriangle, and Square by copying the code in the hidden document. This code is identical to the solutions for Lesson 7 except for the package name. Select the link to access the starting code.

[Sorting Comparable Objects Starting Code](https://www.connexus.com/content/render.aspx?idDocument=1262878&idCourse=32934)

1. The next step will be to compare two shapes to determine which one is bigger or smaller than the other. Do this by comparing areas. A Square with an area of 10.0 is considered bigger than a RightTriangle with an area of 8.0.
2. Implement the Comparable interface in the Shape class. To compare a Shape to other Shapes, you will implement Comparable<Shape>.
3. Because the getArea() method is overridden in each child class, you can write a single compareTo()method in the Shape class that will work correctly when comparing any type of Shape or subclass object to any other. Implement the compareTo() method in the Shape class. Remember that you are comparing the area of the current object to the area of the Shape being supplied as an argument. The header of your method should look like the following:

public int compareTo(Shape s)

In the code for your method, if the area of the current object is less than the area of the incoming object, return -1. If the incoming object is smaller, return 1.

The String class also implements Comparable. If the areas of your two Shapes are the same, compare the current Shape’s name (getName()) to the incoming Shape’s name, and return the result of that comparison. The code for the comparison is shown below.

int k = getName().compareTo(s.getName());

The method getName() returns a String. The compareTo() method above compares two Strings, which might be “square” and “rectangle”, and returns a positive, negative, or zero result based on the comparison. Strings are compared lexicographically rather than alphabetically, meaning uppercase letters will have a lower value than lowercase letters. As long as all of your getName() values are lowercase, case will not be an issue. As a result, the words will sort alphabetically.

1. Override the toString() method inherited from Object in the Shape class. Have it return a String that includes the name of the current object and its area in the following format:

name: area

The values for name and area are produced by calls to getName() and getArea(). For example, the output of the toString() method from a Rectangle with height 4 and width 5 would look like this:

rectangle: 20.0

1. Create a Plus class. Plus is a twelve-sided shape that fits in a square and looks like this:



1. A Plus shape has a height and a width like all the other Shapes in this program. However, as the diagram depicts, the shape is formed by starting with a square with height *h* and width *w*, then removing squares of a third of the height and a third of the width from each corner. Note that height and width are equal (since the Plus should fit in a square), so in the diagram, we will just use width.
2. Since the plus shape is not conceptually a type of square, have it be a subclass of Shape. Because the height and width will always be the same, have its constructor accept one argument, then call its parent’s constructor to set the height and width values using this one argument.
3. Override any methods you believe are necessary. For the toString() method, return "plus" and the area. When performing any calculations involving division, be sure to use double values.
4. Create a ShapeTest class with a main() method. In the main() method, create an ArrayList of Shape references. Create the following objects and add their references to the list in the order shown. Do not change the arguments. Use the following values exactly.

new Square(4)

new Rectangle(3, 2)

new RightTriangle(5, 2)

new Square(2)

new RightTriangle(2, 4)

new Plus(3)

1. Display the contents of the ArrayList by sending its reference to a println() statement.
2. Import java.util.Collections. Use the Collections.sort() method to sort your list.
3. Display the contents of your list again.
4. Run ShapeTest as a Java application. Observe your results. The elements in the last line should be sorted. Those elements that have the same area should be in alphabetical order.

## Submission Instructions

For this portfolio you should include the following 6 files:

1. Square.java
2. Rectangle.java
3. RightTriangle.java
4. Plus.java
5. ShapeTester.java
6. Shape.java

To submit your portfolio for grading, find your Java source files & text files in your Project folder within your Eclipse workspace (a folder you created for Eclipse to save projects when you first installed). Copy & paste your Java source files & text files to a new folder. Compress the folder so that it has the extension .zip. Upload this zip file to the portfolio dropbox.

If you need help with submitting per the above requirements, [please click here](https://docs.google.com/document/d/1stEEnJIoVx8pZSAZy-U8sLtvZylyzskrbgM6rCn7m08/edit?usp=sharing).

**\*\*\*Please note I will not be accepting a word document with code and your output for this portfolio. If you submit a word document, you will receive a zero. You will be able to resubmit properly for up to full credit if this happens.\*\*\***

# Lesson 13: Sets & Maps

## Portfolio Overview

By completing this activity, you will develop a greater understanding of different types of sets and maps and how to store and retrieve information from each general type. In addition to normal set and map operations, a complex problem will be studied that will require the use of a custom class and more than one data structure to solve.

## Portfolio Goal

This program will read the contents of text files and provide methods that will show the text and count the number of times each character appears. One method will show the characters in alphabetical order with their associated counts. Another method will show the characters in descending numerical order according to their counts.

This portfolio involves five files. The main program, CharCountingStudy, will create two objects from the FileExaminer class. The FileExaminers will each read a text file and perform calculations on the data. A fifth class, CharCount, will be used to aid an operation beyond the capabilities of a standard map. You will be responsible for providing the code for the FileExaminer and CharCount classes. The CharCountingStudycode and the text files are provided for you.

## Portfolio Instructions

1. In Eclipse, create a new package named lesson13. All classes in this activity will be placed inside this package.
2. Two text files, raven.txt and jabberwocky.txt, have been provided for you. Download these files to a location you remember and them import them into the Java 2 project in Eclipse. In order to make the coding easier, you will want to rename the files to be raven.txt and jabberwocky.txt.
3. Click on the link to download the text files. They will download in a zip file which you will need to extract (right click > Extract All).

[Download files from here](https://drive.google.com/open?id=1IuSfDCIMfxh4cgqxh2lgq_Pmq3jvmNVs)

### Importing Files into Eclipse

1. Download files to your Downloads Directory or another directory of your choosing.
2. Right-click on the project you wish to import the files into and select Import.
3. Open the General folder, select File System, and select the Next button.
4. Browse to the directory where you saved your files.
5. Check the boxes next to the files you want to import.
6. Select the Finish button.

### Portfolio Introduction & Algorithm Overview

Watch the following video tutorial to get a better understanding of the prior knowledge needed for this portfolio and the steps you will take: [Intro & The Algorithm](https://www.loom.com/share/7068a19c418a4da0b92ff66d1a86d809?sharedAppSource=personal_library)

### CharCountingStudy Class

You can watch a video tutorial of the steps for the CharCountingStudy class and part 1 of the FileExaminer class here: [The Code (CharCountStudy, FileExaminer part 1)](https://www.loom.com/share/b5eb4ddaa3674befa74960299a620935?sharedAppSource=personal_library)

1. Create a class named CharCountingStudy by copying the following code exactly.

package lesson13;

import java.util.LinkedHashSet;

import java.util.Set;

public class CharCountingStudy {

public static void main(String[] args) {

Set<FileExaminer> examiners = new LinkedHashSet<>();

examiners.add(new FileExaminer("raven.txt"));

examiners.add(new FileExaminer("jabberwocky.txt"));

for (FileExaminer fe : examiners) {

System.out.println("--------------------");

fe.display(9);

System.out.println();

fe.displayCharCountAlpha();

fe.displayCharCountNumericDescending();

}

}

}

1. This program creates a number of FileExaminer objects and adds their references to a Set of FileExaminers. This makes it easy to loop through them. In the for loop, the display(), displayCharCountAlpha(), and displayCharCountNumericDescending()methods are called on each FileExaminer. You will be implementing these methods in the next few classes.

### FileExaminer Class

The FileExaminer class will read a text file, perform calculations on the data, and provide methods that will allow that information to be displayed.

You can watch a video tutorial of the steps for the CharCountingStudy class and part 1 of the FileExaminer class here: [The Code (CharCountStudy, FileExaminer part 1)](https://www.loom.com/share/b5eb4ddaa3674befa74960299a620935?sharedAppSource=personal_library)

1. Create a class named FileExaminer.
2. Inside FileExaminer, create three private instance variables:
   1. Create a Set of String named lines. This set will store the lines of the text file. **Choose a set that stores elements in the order in which they are added to the set.**
   2. Create a Map of Character keys and Integer values named charCountAlpha. This map will store every character found in the text file as a key and the number of times each character is found as a value. **Choose a map that automatically sorts the keys**.
   3. Create a Set of CharCount objects named charCountNumericDescending. The CharCountclass will be created later in this activity. **Choose a set that automatically sorts its elements**.
3. Create a constructor for FileExaminer that takes a String named filename as an argument. Use filename to create a File object. Use your experience from exception handling and file I/O from Lesson 6 to read the contents of this file. Store each line read into the lines set. Remember to catch any exceptions that occur. If the file is not found, display the text, "File not found: " plus the name of the file.
4. At the end of the constructor, call the method calculateCharCountAlpha(). You will write this method next. You want this method to be called as part of the construction process.
5. Create a private method named calculateCharCountAlpha(). This method will read through all the lines you just collected and count the characters in every line. The character keys will be sorted automatically if you chose the correct Map type.
   1. Loop through every element in the lines set.
   2. In each line (String), loop through all of the characters in the line. Get each character with charAt(). Use the charCountAlpha map to store the characters and their counts using this logic: if charCountAlpha contains your current character as a key, get the count associated with that key, then put the key back into the map with a value of count + 1. If the character was not found in the map, add it to the map with a value of 1.
6. After your loops, at the end of calculateCharCountAlpha(), call calculateCharCountDescendingByCount().
7. Create a private method named calculateCharCountDescendingByCount().This method will use the information gathered by calculateCharCountAlpha() to sort the information using map **values rather than keys**. You want to know which character occurred the most in the file, followed by the second most common, etc. While a Map will sort keys, there is no built-in functionality to sort by values. In order to do this, you will need to take extra steps. You will need to create a custom class that represents each Character-Integer pair. This class will implement Comparable. You will create objects for each pair, place all their references in a Set, and have the set automatically sort them for you.
8. For now, leave this method for later.

### CharCount Class

You can watch a video tutorial of the steps for the CharCount class here: [The Code (CharCount)](https://www.loom.com/share/bb174c76bf7f4e20b8646e3172ae2440?sharedAppSource=personal_library)

1. Create class CharCount and have it implement Comparable<CharCount>.
2. Inside CharCount, create two private instance variables of type Character and Integer named ch and count, respectively.
3. Create a constructor that will accept a Character and an Integer. Assign the incoming values to your instance variables.
4. Implement the compareTo() method. The argument to compareTo() will be a reference of type CharCount representing another object that the current object is being compared to. You want the results of this method to sort elements in descending order according to their counts.
5. First, get the results of a compareTo() between count and the other object’s count. Multiply the value received by –1 to change its sign. This will make a positive number negative or a negative number positive in order to get the opposite value. For example, suppose one CharCount object represents the letter “e” and the other represents the letter “a”. If “e” has a count of 15, and “a” has a count of 20, the result of 15 compared to 20 would normally be a negative value, but now it should be a positive value if you want to sort them in descending order.
6. If the result is not zero, return that value. Otherwise, return the results of a normal compareTo()comparison between ch and the other object’s ch. This is necessary, because if the letters “e” and “a” both occur 22 times in a text file, a compareTo() result of zero would mean they are equal, and one would get eliminated when added to a set. This extra step prevents that.
7. Override the toString() method by copying the following code. This method provides more descriptive output for tab and space characters when printing out the charCountNumericDescending set.

public String toString() {

if (ch.equals('\t')) return "(tab)=" + count;

else if (ch.equals(' ')) return "(spc)=" + count;

else return ch + "=" + count;

}

### FileExaminer Class, revisited

You can watch a video tutorial of the steps for the FileExaminer class (part 2) here: [The Code (FileExaminer part 2)](https://www.loom.com/share/b19696c974574f33ac98af96eb1c4a71)

1. Go back to the calculateCharCountDescendingByCount() method. Inside this method, retrieve and store a Set<Character> of keys by calling the appropriate Map method on charCountAlpha. Loop through the Characters in this set. For each one, create a new CharCount object and add it to the charCountNumericDescending set. Remember that when creating a CharCount object, pass to it a Character and the Integer associated with it. These values are both available from your charCountAlpha map.
2. Create a public method named displayCharCountAlpha() that displays charCountAlpha.
3. Create a public method named displayCharCountNumericDescending() that displays charCountNumericDescending.
4. Create a public method with the signature, display(int numberOfLines). In this method, find a way to display a number of stored lines from the text file specified by the incoming value numberOfLines. Even if the text file is 100 lines long, if the incoming value is 5, display only 5 lines.
5. Add any imports that are needed by this class and fix any errors that occur.

**Running the Program**

1. Run the CharCountingStudy class as a Java application. Observe the output generated by the two character counting techniques. Also observe that characters with the same count values have not been eliminated.

## Submission Instructions

For this portfolio you should include the following 5 files:

1. CharCountingStudy.java
2. FileExaminer.java
3. CharCount.java
4. raven.txt
5. jabberwocky.txt

To submit your portfolio for grading, find your Java source files & text files in your Project folder within your Eclipse workspace (a folder you created for Eclipse to save projects when you first installed). Copy & paste your Java source files & text files to a new folder. Compress the folder so that it has the extension .zip. Upload this zip file to the portfolio dropbox.

If you need help with submitting per the above requirements, [please click here](https://docs.google.com/document/d/1stEEnJIoVx8pZSAZy-U8sLtvZylyzskrbgM6rCn7m08/edit?usp=sharing).

**\*\*\*Please note I will not be accepting a word document with code and your output for this portfolio. If you submit a word document, you will receive a zero. You will be able to resubmit properly for up to full credit if this happens.\*\*\***

# 