**Instructions: Please read!**

**All programming assignments are to Java source files you create using Eclipse.**

* **Source code documentation in the java source is mandatory**.
  + These include a **clearly stated** class description using at least 2 sentences and the information needed about author and version using j**avadoc**format.
  + **Every method must be accurately described in javadocLinks to an external site. format and placed on the line immediately above the method declaration**. See examples from classroom discussions uploaded to module.
  + Mismatched or incomplete javadoc **will be highlighted** by CheckStyle if you have configured Eclipse per the instructions given at the beginning of the term.
* Please ensure that you have **formatted** (**whitespace**) the file to make it readable. In the Eclipse editor: Ctrl Shift F on Windows or Cmd Shift F on macOS will format your opened java source.
* **Please remove all inline comments if you have added any** in your file as over-commenting can make the source unreadable.
  + This refers to Comments beginning with **// or block comments /\*... \*/ (these will be color coded green in Eclipse)**
* Variable names must be **relevant to the solution and intuitive**. Single letter variable names or ambiguous names will result in reduction of points.
* The **class** names have been provided to you. Use appropriately named methods that are relevant to the questions below.
* **User interaction must include clear instructions to the user**. Even though your textbook avoids this, it is important to make your application user friendly.
* All variable and method names must use **[camelCase](https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html" \t "_blank)**[Links to an external site.](https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html" \t "_blank). <--See link
* Use **exception handling** where necessary to ensure your logic functions correctly. ***As an option, you may also create your own exception classes. If you do, then those exception classes must also be submitted***
* All resources **must be closed** before the execution terminates.
* **Please read Submission and Grading details at the bottom of the page**

**Answer the Question from Part A - 50 points**

**Optional: Answer ANY ONE from PART B - 25 points**

**PART A**

Lyrics in some genres of music contain words that repeat a number of times, a simple way to compress a lyric file is to create a map that stores each word once along with the positions of each word in the file.

The objective is to read a file containing the lyrics of a song and create the following:

* A HashMap that maps a lyric word to a list of integers representing the locations of the word in the lyrics. (Assume the first word is at location 0)
* The total number of distinct words (Case sensitive)
* The most frequent word(s) used. If there are more than one word that has the **same maximum** **frequency**, list all of them in alphabetical order
* Save the reconstructed lyrics from the HashMap to a new file. The format of the new file will be each word on a different line in the correct order. See the example below.

You are given an interface called [**TextAnalyzer.java**](https://learn.vccs.edu/courses/631696/files/170182076?wrap=1)

[**Download TextAnalyzer.java**](https://learn.vccs.edu/courses/631696/files/170182076/download?download_frd=1) with the list of methods. You must implement this interface in your solution. **Please do NOT modify the interface**

**Please download this file to your Eclipse project. You may have to update the package statement based on your project setup.**

For example, if the text was the lyrics of the song Yellow Submarine by The Beatles and consists of the lines:

We all live in a yellow submarine  
Yellow submarine, yellow submarine

The output would be

Words and Positions:  
  
We [0]  
Yellow [7]  
a [4]  
all [1]  
in [3]  
live [2]  
submarine [6, 10]  
**submarine,** [8] <-- note the comma that separated this word from the other occurrences of submarine  
yellow [5, 9]  
  
Reconstructed Lyrics:  
  
We  
all  
live  
in  
a  
yellow  
submarine  
Yellow  
submarine,  
yellow  
submarine  
  
Total Unique Words: 9 <-- Update: this is effectively the number of key value pairs. If you have already submitted your work - no change needed.  
  
Most Frequent Word(s): submarine, yellow  
  
Saved the reconstructed song to a new file. [newsubmarine.txt](https://learn.vccs.edu/courses/631696/files/170223689?wrap=1)

[Download newsubmarine.txt](https://learn.vccs.edu/courses/631696/files/170223689/download?download_frd=1)

**Use the following text files for testing:**

* [yellowsubmarine.txt](https://learn.vccs.edu/courses/631696/files/170187939?wrap=1)

 [Download yellowsubmarine.txt](https://learn.vccs.edu/courses/631696/files/170187939/download?download_frd=1)

 [dirtywater.txt](https://learn.vccs.edu/courses/631696/files/170187942?wrap=1)

 [Download dirtywater.txt](https://learn.vccs.edu/courses/631696/files/170187942/download?download_frd=1)

 [anotherbrick.txt](https://learn.vccs.edu/courses/631696/files/170187976?wrap=1)

* [Download anotherbrick.txt](https://learn.vccs.edu/courses/631696/files/170187976/download?download_frd=1)

|  |
| --- |
| **Note:**   * The text in the file must be read as is - there is no need to remove any punctuation. * However leading and trailing spaces may have to be trimmed for each word added to the HashMap. * You may reduce the text to a more manageable amount of lines for testing. |

**OPTIONAL for a 10 point bonus:**

* Create an algorithm for a simple hash function for a String key
* Use pseudocode to explain your algorithm
* Provide one example where the algorithm fails because of collisions
* this does not require any implementation in Java.
* Submit your response via a pdf or word doc.

***Files to be submitted:***

* ***LyricAnalyzer.java***
* ***LyricsTester.java***
* ***Optional part to be submitted as a word doc or pdf- if you are answering the bonus section***
* ***Please don't submit the interface. The file provided in the question will be used to test your solution.***

**Part B**

**Question 1**

Given the Cartesian coordinates of a point, the objective is to find all the points from an array of points that shares either the x or the y value. Using the resulting set of points, find the closest point by calculating the Manhattan Distance (also known as Taxicab Metric)

The Manhattan distance between two points

and

is

More information on Manhattan Distance is available at <https://en.wikipedia.org/wiki/Taxicab_geometry>

[Links to an external site.](https://en.wikipedia.org/wiki/Taxicab_geometry)

Create a Java class called **TaxicabGeometry** that includes the following attributes and behaviors

* Attributes
  + int[] origin (this must have exactly two values)
  + int[][] pointsArray  (The number of rows must be at least 1 and have exactly 2 columns for each row)
* Behaviors
  + Constructor that accepts the values for origin and pointsArray. **Throw an exception if either of the arrays are null or of length 0**
  + Accessor methods for each attribute
  + Mutator methods for each attribute. **Throw an exception if the array is null or of length 0**
  + **toString** - this should return a String that includes
    - the cartesian coordinates of the origin,
    - the point referenced by the return value from **findNearestPoint** method. If the returned value is -1, substitute with appropriate text.
    - the value returned by **calcTaxicabMetric** method. If the distance is returned as -1, substitute with appropriate text (see an example below)
  + **calcTaxicabMetric**: this is a **private** method that calculates the taxicab metric using the formula stated above and returns an int value. This method calls the findNearestPoint method to find the nearest point from the origin. If the distance cannot be calculated, return **-1**
  + **findNearestPoint**: this is a **private** method that finds the nearest point that shares either x or the y coordinate with the origin. **This method must use a HashMap.**This method must return the index of the point in the pointsArray. If no point is found, then return -1.

**Example 1:**

Calculating Taxicab Metric:  
Input Origin: 3,4  
  
Input array of points (Done to end input):   
  
1,5  
2,1  
10,3  
9,4  
12,4  
Done  
  
The nearest point to the origin (3, 4) is (9, 4) and the taxicab metric is: 6

**Example 2:**

Calculating Taxicab Metric:  
Input Origin: 1,2  
  
Input array of points (Done to end input):   
  
3,5  
4,7  
10,8  
9,6  
12,5  
Done  
  
The nearest point to the origin (1, 2) does exist in the data provided and the taxicab metric cannot be calculated

Create a Tester class called **TaxicabMetricTester**

**Organize your logic into multiple methods**

Get input from the user for both the origin and the array of values. Validate the input so that all values are between 0 and 15

Use exception handling when getting input from user.

Create an object of type TaxicabGeometry and display the output

Handle the exception if any, when returned by the constructor or the mutator methods

***Files to be submitted:***

* ***TaxicabGeometry.java***
* ***TaxicabMetricTester.java***

**Part B**

**Question 2**

Create a Java class called **PixelFlip**.

The objective is to convert the entire row and column to a given hex value if there is at least one value equal to the given hex value in that row or column

Assume the 2D array is a square matrix (# rows = # columns)

The hex values are generated by the Random number generator.

Set the upper bound to 0xffffff + 1 for the random number generator. This will allow for all colors in the spectrum to be used.

For every instance of the color in the matrix update the entire row and column that the cell belongs.

Note: You must use a **HashSet** in the flipColor method

If the color to be used in the update does not exist in the matrix, throw an IllegalArgumentException

The following UML is the design that must be used

**Example:**

Change color values for row and column...  
  
Input array size: 5  
  
Generated array:  
  
#1b4ed9 #0d5498 #06a3ee #148e15 #13ba06   
#23198f #0f1272 #36446c #0899ff #1d864c   
#30b1dc #104ebe #1582c4 #11d295 #347722   
#1423dd #18c9f6 #3132e7 #1816ca #220428   
#2a18e2 #04da23 #2e3a94 #1cbbd4 #261e32  
  
Input color: #347722  
  
The changed array:  
  
#1b4ed9 #0d5498 #06a3ee #148e15 #347722   
#23198f #0f1272 #36446c #0899ff #347722   
#347722 #347722 #347722 #347722 #347722   
#1423dd #18c9f6 #3132e7 #1816ca #347722   
#2a18e2 #04da23 #2e3a94 #1cbbd4 #347722

***Note: the green is just to show you what changed.***

Create a Tester class called **PixelFlipTester**

**Organize your logic into multiple methods**

Get input from the user for the array size

Generate the array

and pass it to the constructor of the **PixelFlip** class

Display the array - use the toString method to return the array using String formatting

Get input for the color to be flipped along the row and column. Validate the input color as a valid Hex Color value.

Call the **flipColor** method to change the color. Handle the exception if thrown by the flipColor method.

Display the array after the flip - use the toString method to return the array using String formatting

***Files to be submitted:***

* ***PixelFlip.java***
* ***PixelFlipTester.java***

**Submission**

See the list of files/documents for each problem

**All solutions must be your own** - See [**Syllabus**](https://learn.vccs.edu/courses/631696/assignments/syllabus) for details .

**Follow the programming instructions stated at the top of the page**

Significantly incomplete or missing javadoc will result in a reduction of your score by **10 points**.**This is non-negotiable.**

**Line comments above or below the method signature are not a substitute for Javadoc.**

**Remove unused logic/methods/variables/import statements. Eclipse will underline these in yellow.**

**Once again to sound extra tedious :-) and for the last time in this course: Source code documentation is mandatory.**

**Grading**

Your solutions will be graded based on the following:

* **logic**
* **structure**
* **style**
* **following Java conventions in your implementation**
* **output**
* **completeness of source code documentation**
* **meeting the stated requirements.**