# Programming Project 5 – Moon Samples

*Note: When you turn in an assignment to be graded in this class you are making the claim that you neither gave nor received assistance on the work you turned in (except, of course, assistance from the instructor).*

**Program Name: MoonSamples.java**

Martians wants a program to help them analyze the soil on Mars. They want to determine the best areas to grow crops. You will write a program that will accept, at the **command line,** two Strings, one containing various soil composition elements and another containing the values found of these elements within the soil samples. A sample of the two Strings are below:

carbon-dioxide,magnesium,sodium,potassium,chloride,water

8.3,4.5,6.7,2.3,12.5,4.5<>3.9,1.8,34.7,23.5,1.2,14.3<>6.7,7.4,1.5,18.4,7.2,23.7<>23.4,5.6,2.9,18.5,39.5,18.2<>15.4,5.3,27.4,9.8,3.8,27.4

The elements in the first string are separated by a "," character. The compositions found in the second string are separated into a two-dimensional grid with the rows separated by the "<>" characters and each column separated by a "," character. MoonSamples.java will contain several methods.

The first method is the getElements() method. This method takes in a String of elements and returns a one-dimensional String array containing the list of elements. You can use the String method <String>.split(",") to split your string based on the "," character separating the categories. Here is the method header.

public static String[] getElements(String inputElementString){

//Your code here

}

The second method is the getSamples() method. This method takes in a String of sample values and returns a two-dimensional double array containing the elemental composition of each sample. Again, you can use the String method <String>.split("String") to split your string based on the char character you specify. You will need to split this string on the "," and "<>" symbols. Here is the method header.

public static double[][] getSamples(String inputSamplesString){

//Your code here

}

Here is an example of the two arrays that are created from the first two methods. Note the two-dimensional array will just contain the sample values. This is just an example of the data that might be used to test your code. The elements and their order will not change. However, the samples will change, but will always be numerically labeled starting with the number 1.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | carbon dioxide | magnesium | sodium | potassium | chloride | water |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | carbon dioxide | magnesium | sodium | potassium | chloride | water |
| Sample 1 | 8.3 | 4.5 | 6.7 | 2.3 | 12.5 | 4.5 |
| Sample 2 | 3.9 | 1.8 | 34.7 | 23.5 | 1.2 | 14.3 |
| Sample 3 | 6.7 | 7.4 | 1.5 | 18.4 | 7.2 | 23.7 |
| Sample 4 | 23.4 | 5.6 | 2.9 | 18.5 | 39.5 | 18.2 |
| Sample 5 | 15.4 | 5.3 | 27.4 | 9.8 | 3.8 | 27.4 |

The third method is searchForLife(). This method takes in a 2D double array of sample values. It will search all samples for the ones that could support life. The formula for supporting life is as follows:

The criteria value that needs to be attained is: 300. If this criteria value is attained or surpassed, then the sample number is returned in an int array. If the first row of data attains the score of 350 or more, the number 1 will be inserted into the first position in the int array. Since not all samples may meet the criteria, the array returned should be resized so that it only contains the values that meet the criteria with no extra data.

public static int[] searchForLife(double [][] samples){

//Your code here

}

The fourth method is searchHighestElements(). This method will take in a String array of elements, a 2D double array of sample values, and a sample to search. It will search that sample and will return a string containing the two elements that are found in the highest amount for that sample in the order of highest first, then second highest element.

public static String searchHighestElements(double [][]

samples, String[] elements, int sampleNum){

//Your code here

}

The fifth method is searchHighestSample(). This method will take in a String array of elements, a 2D double array of sample values, and an element to search. It will search for and return the sample number that contains the highest amount of that element.

public static int searchHighestSample(double [][]

samples, String[] elements, String element){

//Your code here

}

Lastly, you will define a main method. The main method will do the following:

* Accept, at the **command line,** two Strings, one containing various soil composition elements and another containing the values found of these elements within the soil samples
* Pass the first command line argument to the getElements() method and store the result
* Pass the second command line argument to the getSamples() method and store the result
* Call searchForLife(), passing in the result of getSamples() and print the result
* Call searchHighestElements(), passing in the results of getSamples() and getElements() and print the result
* Call searchHighestSample(), passing in the results of getSamples() and getElements() and print the result

The main method will not be tested by the test cases. You should use your own output statements to test the output of your code based on the inputs given.

Below is a sample run:

**The samples that contain life are: 4 5**

**The highest elements for sample 1 are chloride and carbon-dioxide**

**The sample with the highest value of the element water is 5**

You can assume all arguments passed in on the command line are valid Strings for this program.

Before beginning this project, you will document your algorithm as a list of steps to take you from your inputs to your outputs. This algorithm will be due on paper at the beginning of class on the day it is due. This will be graded and returned to you. It will be your responsibility to understand and correct any errors you might have with your algorithm.

Each step will be added as a comment block within your code. You will have the comment block right above the code that performs the actions specified. For example, before your lines of code that ask the user for inputs, you would have a comment block that states what inputs you are requesting from the user.

This and all program files in this course must include a comment block at the beginning (top) of the source code file that contains:

* the Java program name
* project description
* your name
* the date created
* the course number and section

The comment block for the header as well as all the comment blocks should look like this:

/\*

\* Java program name

\*

\* Project description

\*

\* Your name

\* The version date

\* The course number and section

\*/

You will test your code using the provided JUnit tests. You should use these tests within IntelliJ to make sure that your code is running correctly. You will take a screenshot of your code passing all JUnit tests. Make sure that the screenshot has your header comment block in it so that we know it is your code running. Once it is, then you will submit your code to Gradescope. You will only have 4 submissions to Gradescope, so please make sure your code is running correctly in IntelliJ before submitting to Gradescope.

You will also take a screenshot of your MoonSamples.java code running within IntelliJ showing the output from running the methods requested above.

You will submit your Java source code file (MoonSamples.java) to Gradescope. You will upload your two screenshots, one showing the JUnit tests passing and one of the MoonSamples.java code running, by uploading the files to the Assignment link in Canvas. Please do not submit your files in a zipped folder.

Ask questions about any part of the programming project that is not clear!

## Rubric for Programming Project 5

|  |  |
| --- | --- |
| **Criteria** | **Points** |
| Algorithm submitted in class on time | 15 |
| Comments used appropriately (including comment header information and fully documented algorithm) | 15 |
| Appropriate choice of variable names | 5 |
| Correctly populates elements array | 10 |
| Correctly populates samples array | 10 |
| Separate methods are used as specified and using good program design | 20 |
| Output is correct | 10 |
| Program layout and appearance (Coding style is clear and easily understood) | 5 |
| Screenshot of JUnit tests passing & MoonSamples.java running | 10 |
| TOTAL | 100 |