Programming Project 7 – Moon Study

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: MoonStudy.java

Martians want to improve their exploration of the moons and have created a text file of their moon data that we looked at in Project 4. You will write a program that will accept at the **command line** two filenames: one is a text file containing the moon data and the other is an output filename.

The data in the first text file consists of information about the moons that the Martians are still exploring. The data contains many lines of data, one for each moon. Each line of data contains information about the moon:

- The moon's name, a String
- The moon's radius, a double
- The moon's density, a double
- The moon's distance from the Mars, a double

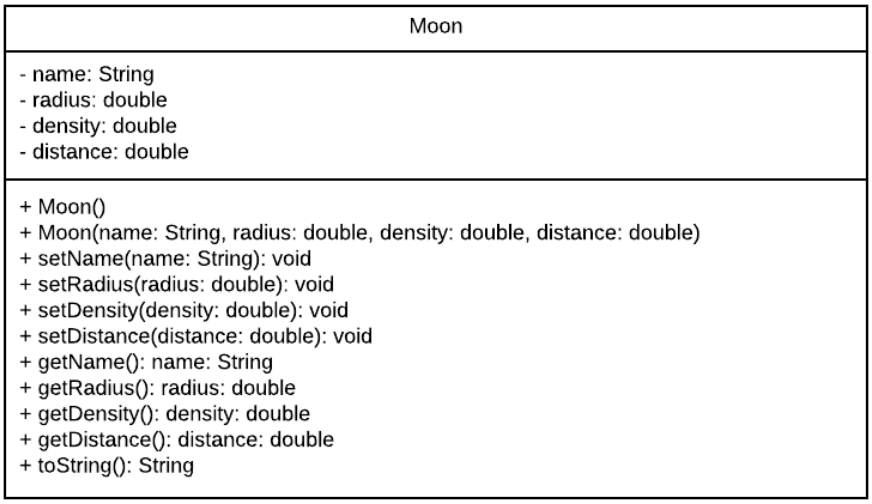
A sample text file is below:

Phobus	11.3	1.8	3.7
Deimos	6.2	1.4	23.4
Adrastea	68.9	14.2	550391.6
Aitne	33.6	33.3	227894.9
Amalthea	71.2	16.4	778893.6
Ananke	26.8	68.1	143323.5
Aoede	255.9	121.3	287223.5
Arche	47.4	38.2	449655.1

Before implementing MoonStudy.java, we will create an object to encapsulate the data values associated with a moon. Here are the specifications for the Moon.java class:

- You will have 4 instance variables: name: String, radius: double, density: double, distance: double
- You will have a default constructor that sets all string values to an empty String and numeric values to 0.0.
- You will have a parameterized constructor that passes in a name, radius, density, and distance to set to the instance variables
- You will have getters and setters for each instance variable

- You will have a toString() method that will return the name, radius, density, and distance separated by a space. Each double value will be rounded to two decimal places using **String.format()**.



You will also create a Java Enumerated Type in its own class file named MoonAttributes.java. The values of MoonAttributes will be: RADIUS, DENSITY, DISTANCE

MoonStudy.java will contain several methods. The first method is **openFile()**, which takes in a File object and will loop to read the data from the file, line by line. Each line will contain data for a Moon object as shown above. The lines of data read will be stored in an ArrayList of Strings. Here is the method header:

```
        public static ArrayList<String> openFile(File inputFile) throws
FileNotFoundException {
           //Your code here
        }
```

The second method is **createObjects()**, which will parse the data from each element of the ArrayList into an ArrayList containing Moon objects.

Within each row of String data, the data is separated by tabs. You can use the String.split("\t") method to split the data and temporarily store it in a String array.

Once the data is in this new String array, you can use this data to create a new Moon object. The first element is the name, the second is the radius, the third is the density, and the fourth is the distance. You will need to use the Double.parseDouble() method to convert your String input into the double value you need for the three double values for each of the numeric data attributes for Moon objects. You will want to make sure that your numeric data actually is numeric for each of the double values in the data. You will handle this possible conversion error by catching the exception that is thrown, the NumberFormatException. If the file has **incorrect data** or **negative values** for any of the numeric inputs, you will simply just place a 0.0 into the numeric instance variable instead of the incorrect value. However, you will keep all the good data for each Moon object, so only place 0.0 into the attributes that have bad data.

Here is the method header:

```
public static ArrayList<Moon>
                                createObjects(ArrayList<String> lines){
  //Your code here
}
```

The third method is the **findMean()** method which calculates the average for the given numeric attribute of the Moon objects. This method will return the mean/average of the Moon objects' given attribute as a double. Here is the method header:

```
public static double findMean(ArrayList<Moon> moons,
            MoonAttributes attribute){
  //Your code here
}
```

The fourth method is the **findHighValue()** method which calculates the highest value for the given numeric attribute within the Moon objects. This method will return the highest of the Moon objects' given attribute as a double. Here is the method header:

```
public static double findHighValue(ArrayList<Moon>
            moons, MoonAttributes attribute){
  //Your code here
}
```

The fifth method is the **findMeanMoon()** method which finds and returns the Moon object whose value for the given attribute within the given ArrayList of Moon objects is closest to the mean value of the data for that given attribute. This method will return the Moon object who's value of the Moon objects' given attribute is closest to the mean. The method will pass in the array list of Moon objects, an attribute which is being searched, and the mean value that is being searched. Here is the method header:

```
public static Moon findMeanMoon(ArrayList<Moon>
            moons, MoonAttributes attribute, double meanValue){
  //Your code here
}
```

The sixth method is the **findLowestMoons()** method which finds the moon objects below the value passed in for the given attribute. It will pass in the ArrayList, a value, and an attribute. It will return an array list containing the Moon objects that are lower than the value passed in for a given attribute. Here is the method header:

```
public static ArrayList<Moon> findLowestMoons(ArrayList<Moon>
            moons, double value, MoonAttributes attribute){
  //Your code here
}
```

The next three methods are the **writeOutData()** methods. There are three outputToFile

methods depending on the data you need to print. They will all pass in a String of text to describe the output and a **PrintWriter** object to print to. Each output will be on a new line, and each output will be separated by a blank line.

To print an array list of values:

```
public static void outputToFile(String outputMessage,
      ArrayList<Moon> moons, PrintWriter out){
  //Your code here
}
```

To print one Moon object:

```
public static void outputToFile(String outputMessage,
      Moon moon, PrintWriter out){
  //Your code here
}
```

Note: You will call the Moon toString() for both methods above.

To print a double value:

```
public static void outputToFile(String outputMessage, double
      value, PrintWriter out){
  //Your code here
}
```

Note: You will need to round the double output to two decimal places when you output it.

The tenth and final method is a main method. This main method will first read in the two command line arguments and use them to create File objects. The first command line argument is the input file, the second is the output file. The main method will then call the method **openFile().** You will wrap the call to the **openFile()** method in a try catch block, so that if there is a problem with the opening of the text file, you will catch it in main. If there is a problem, you will print to the console “Incorrect input filename”. If there is not a problem, then you will print to the console “Input file correct”.

The main method will then pass the ArrayList of Strings returned from **openFile()** to the **createObjects()** method. The return value of **createObjects()** can then be used as an argument when calling the **findMean(), findHighValue(), findMeanMoon(),** and **findLowestMoons()** methods.

The main method will then write the values for each output produced in the program to the given outputFile parameter by calling a **writeOutData** method. You will wrap the call to this method in a try catch block, so that if there is a problem with the writing to the text file, you will catch it in the main method. If there is a problem, you will print to the console “Incorrect output filename. If there is not a problem, then you will print to the console “Output file corrrect”. Regardless, you must close the PrintWriter file when you are done so that the file successfully written to.

Sample output for the moonGoodData.txt file is:

At the console line:

```
Incorrect input filename or Input file correct
Incorrect output filename or Output file correct
```

To the output file from moonGoodData.txt:

The mean of radii is: 65.16

The highest density value is: 121.30

The moon closest to the mean is: Adrastea 68.90 14.20 550391.60

The moons below the mean value for radii are: Phobus 11.30 1.80 3.70 Deimos 6.20 1.40 23.40 Aitne 33.60 33.30 227894.90 Ananke 26.80 68.10 143323.50 Arche 47.40 38.20 449655.10

Sample output for the moonBadData.txt file is:

To the output file from moonBadData.txt:

The mean of radii is: 24.28

The highest density value is: 68.10

The moon closest to the mean is: Ananke 26.80 68.10 143323.50

The moons below the mean value for radii are: Phobus 11.30 0.00 3.70 Deimos 6.20 1.40 0.00 Amalthea 0.00 16.40 778893.60 Aoede 0.00 0.00 287223.50

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 one week before the project 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 MoonStudy.java 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.

You will document the Moon.java file using Javadoc comments as we did in Project 6.

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 MoonStudy.java code running within IntelliJ showing the output from running the methods requested above.

You will submit your Java source code files (MoonStudy.java and Moon.java) to Gradescope. You will upload your two or three screenshots, one or two showing the JUnit tests running and one of the MoonStudy.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 7

Criteria	Points
Algorithm submitted in class on time	15
Comments used appropriately (including comment header information and fully documented algorithm)	10
Moon class is correctly defined	15
Reads from file and correctly populates ArrayList<Moon>	15
Correct other methods	20
Correct outputToFile methods & output file is correct	10
Input and output errors are correctly handled	5
Screenshot of JUnit tests passing & MoonStudy.java running	10
TOTAL	100