CS 1302 – HW 6

*Text Files*

Contents

[1 Problem 1 – Martians 2](#_Toc167883027)

[1.1 Overview 2](#_Toc167883028)

[1.2 Setup Eclipse Project 3](#_Toc167883029)

[1.3 Text File Format 4](#_Toc167883030)

[1.4 Requirement: *writeMartiansFile* Method 4](#_Toc167883031)

[1.5 Requirement: *readMartiansFile* Method 4](#_Toc167883032)

[1.6 Hints 5](#_Toc167883033)

[1.6.1 Algorithm 5](#_Toc167883034)

[1.6.2 Detecting Invalid Data 6](#_Toc167883035)

[1.6.3 Detecting Duplicates 6](#_Toc167883036)

[1.6.4 An Approach to Implementation 6](#_Toc167883037)

[1.7 Test Cases – Input Files 7](#_Toc167883038)

[1.8 Output for Test Cases 8](#_Toc167883039)

[1.9 Extra Credit 9](#_Toc167883040)

[1.10 Grading Criteria 9](#_Toc167883041)

[2 Problem 2 10](#_Toc167883042)

[2.1 Overview 10](#_Toc167883043)

[2.2 Details 10](#_Toc167883044)

[2.3 Problem 2 – Suggested Steps 11](#_Toc167883045)

[3 Submission Requirements 12](#_Toc167883046)

General Comments:

* Type this Academic Honesty statement followed by your full name, as a comment at the top of the *MartianManagerIO* and *NumberAdder* classes:

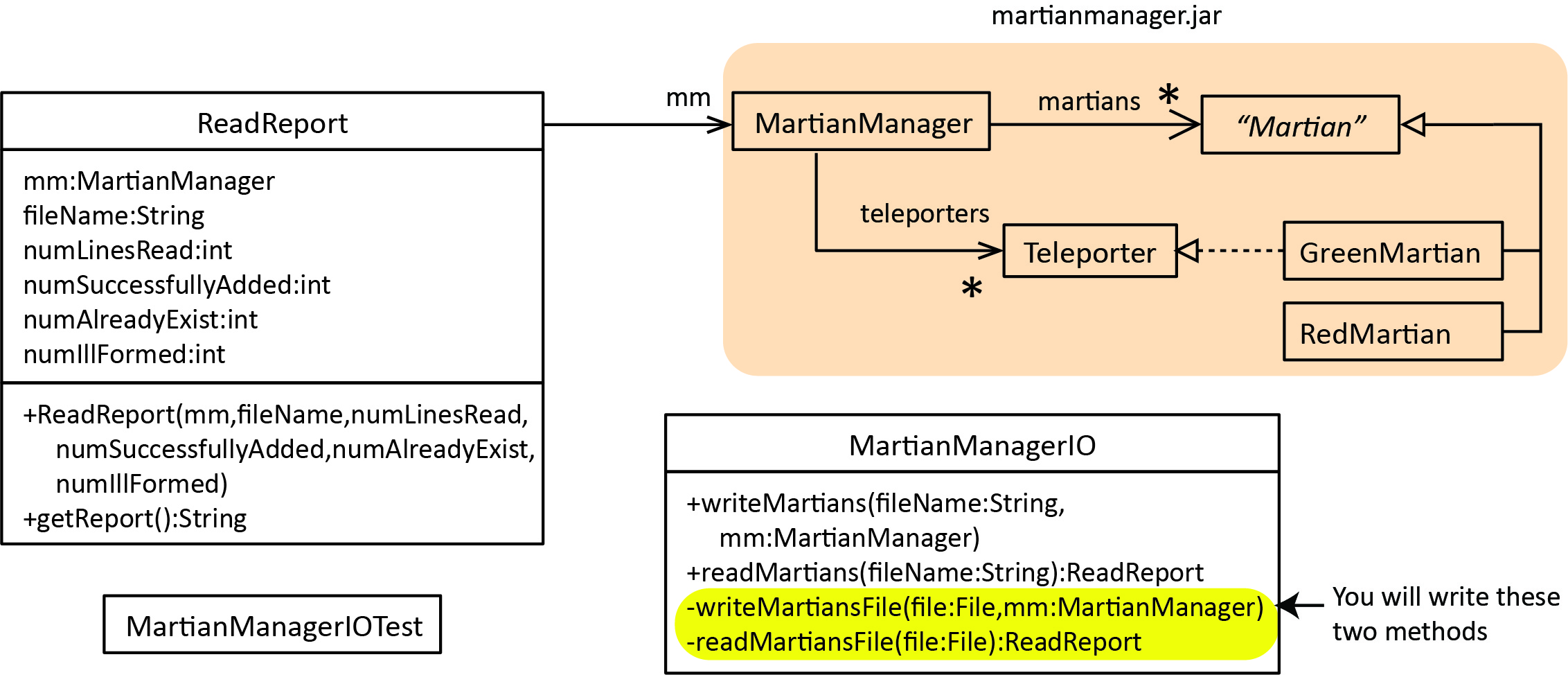
“This homework represents my own work. I understand that I may receive help, but I did not copy any portion of this assignment from anywhere. I understand that a violation of this will result in a Report of Academic Dishonesty.—YOUR FULL NAME HERE”

* There is no *HW06CompileTest* for this assignment.

# Problem 1 – Martians

## Overview

(60 Points) The problem builds on HW 5 where you created a *MartianManager* and related classes as shown in the highlighted (tan) region below. For this assignment, these classes are provided to you in a Jar file*.* In addition, you are provided the other three classes shown below. For this assignment, you will write the two methods highlighted (yellow) in the *MartianManagerIO* class.



Overview of the classes provided:

1. *martianmanager.jar* – Contains the *MartianManager* and related classes.
2. *MartianManagerIO* – You will write these two methods:
3. **private** **static** **void** writeMartiansFile(File file, MartianManager mm) – Accepts a file object and a *MartianManager* object and writes the martians in the manager into the file in the format [described below](#_Text_File_Format_1).
4. **private** **static** ReadReport readMartiansFile(File file) – Accepts a file object, reads the martians in the file, create the appropriate type of *Martian*, add them to a *MartianManager.* However, some *Martians* in the file may not be in the correct format, or some *Martians* may be in the correct format, but are duplicates and will not be added to the *MartianManager.* As the reading takes place, various statistics about the read will be tallied. At the conclusion of the method, the *MartianManager* and the statistics will be packaged inside a *ReadReport* object which is the returned by the method. Details about the statistics are found in another [section](#_Requirement:_readMartiansFile_Metho).

|  |
| --- |
| Note: There are two public methods that are already written, that are called by the test code and which call the methods you will write. You can ignore this description below, but is provided in case you are interested.   1. **public** **static** ReadReport readMartians(String fileName) – This method is called by the test code (*MartianManagerIOTest*) and simply calls the helper method, *readMartiansFile* to do the actual read. **Do not alter this method.** 2. **public** **static void** writeMartians(String filename, MartianManager mm) – This method is called by the test code (*MartianManagerIOTest*) and simply calls the helper method, *writeMartiansFile* to do the actual writing. **Do not alter this method.** |

1. *ReadReport* – Contains a *MartianManager* and a few statistics about the read. A *getReport* method is provided that returns a string with information about the *MartianManager* and the statistics, which is used by the test class. This class is completely written. Example output from the *getReport* method:

\*\*\* Read Report \*\*\*

File: inMartians6.txt

Martian Manager:

Martians:

Green Martian - id=3 vol=5

Red Martian - id=1 vol=4, ten=2

Green Martian - id=4 vol=5

Red Martian - id=2 vol=8, ten=7

Green Martian - id=6 vol=6

Teleporters:

Green Martian - id=3 vol=5

Green Martian - id=4 vol=5

Green Martian - id=6 vol=6

Martian Manager:

Num lines read :11

Num Martians added :5

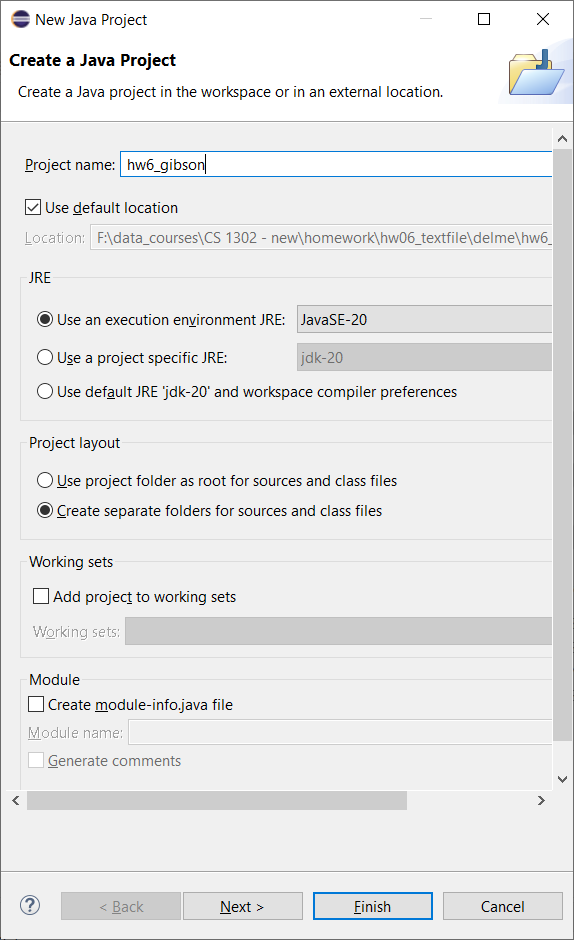
Num ill-formed lines :5

Num already exist (not added):1

1. *MartianManagerIOTest* – Contains test methods for testing the methods above. This class is completely written.

## Setup Eclipse Project

Follow these steps to get Eclipse setup with the provided code:

1. Download and unzip: *hw6\_code.zip*. You will find a *prob1* and *prob2* folders that correspond to the two problems for this assignment.
2. In Eclipse, choose: File, New, Java Project
   1. Supply the Project name: *hw6\_lastname*
   2. **Make sure *“Create module-info.java file”* is unchecked as shown in the figure on the right. The *Module* section is towards the bottom of the *Crate a Java Project* dialog.**
   3. Choose: Finish
3. Drag the *prob1* and *prob2* folders from the zip file into the *src* node in Eclipse.
4. Expand the *prob1* folder and right-click *martianmanager.jar* and choose: Build Path, Add to Build Path

Note: The Jar file classes are in a package named *mm.* The import statement has already been added to the appropriate classes: **import** mm.\*;

## Text File Format

The format for the martians text file is shown below (left column). Thus, a *GreenMartian* is represented with a ‘G’ followed by the *Id* and the *Volume.* A *RedMartian* is represented by a ‘R’ followed by the *Id, Volume,* and *Tenacity.* An example file is shown below (right column). So, we can see that each *Martian* is on a separate line. For example, the first line in the example is a *GreenMartian* with id=3 and volume=5. The second line is a *RedMartian* with id=1, volume=4, and tenacity=2.

|  |  |  |
| --- | --- | --- |
| **Format, either of these is valid** | **Key** | **Example** |
| G I V  R I V T | |  |  |  | | --- | --- | --- | | **Code** | **Type** | **Meaning** | | G | Character | Green Martian | | R | Character | Red Martian | | I | Integer | Id | | V | Integer | Volume | | T | Integer | Tenacity | | G 3 5  R 1 4 2  R 9 3 4  G 7 7  R 2 9 3  G 8 2  G 4 5 |

## Requirement: *writeMartiansFile* Method

1. Open the *MartianManagerIO* class and find:

**private** **static** **void** writeMartiansFile(File file, MartianManager mm)

Write this method. As described above, this method accepts a *MartianManager* and should loop through the martians in the *MartianManager* and write the martians tothe *file* object that is passed as an argument, using the format [described above](#_Text_File_Format_1).

Hint: What *MartianManager* methods will you need to loop over the *Martians* there? Open the class in the *Referenced Libraries* node and look carefully at the methods or reference the Javadoc for HW 5 (where the *MartianManager* and related classes were originally written).

1. Open the *MartainManagerIOTest* class, comment out all the *testReadMartians…* methods, and run the *testWriteMartians* method. Verify that your output is correct.

## Requirement: *readMartiansFile* Method

1. Open the *MartianManagerIO* class and find:

**private** **static** ReadReport readMartiansFile(File file)

You will write this method shortly. As described above, this method accepts a file object, reads the martians in the file and returns a *ReadReport* object. As martians are read, you will determine if they are valid, and if valid, attempt to add them to a *MartianManager*, collecting various statistics about the reading along the way. At the conclusion, you will package the *MartianManager* and the statistics in a *ReadReport* objectand return it.

1. To write the *readMartiansFile* method in the *MartainManagerIO* class, note the following:
2. A *valid* line in the file is one of the two formats [described above](#_Text_File_Format_1). If a line does not meet either of these two formats then it is considered *ill-formed*.
3. When you read a valid line, you can use it to create either a *RedMartian* or a *GreenMartian*. Then, you will attempt to add the martian to the *MartianManager.* The *MartianManager*’s *addMartian* method will only add a martian, if there doesn’t already exist a martian with the same *id.* The *addMartian* method returns *true* if the martian was added and *false* otherwise. Your code will keep track of how many martians were successfully added and how many were not added because they already exist.
4. As lines are being read, martians created, and attempting to add them to the *MartianManager*, your code should keep track of:
5. *numLinesRead* – the total number of lines read (valid or ill-formed)
6. *numSuccessfullyAdded* – the total number of martians successfully added to the *MartianManager*
7. *numAlreadyExist* – the total number of martians that could not be added to the *MartianManager* because they are duplicates, *i.e. already exist.*
8. *numIllFormed* – the total number of lines read that were ill-formed.

Note: *numLinesRead* = *numSuccessfullyAdded* + *numAlreadyExist* + *numIllFormed*

1. To conclude this method, you will build a *ReadReport* object and return it. The constructor for that class is shown below. As you can see, to create a *ReadReport,* simply pass the *MartianManager,* the *fileName,* and the various statistics to the constructor:

**public** ReadReport(

MartianManager mm,

String fileName,

**int** numLinesRead,

**int** numSuccessfullyAdded,

**int** numAlreadyExist,

**int** numIllFormed)

The *fileName* is found by: file.getName()

1. You have been provided six test files: *inMartians1.txt, …, inMartians6.text* which are described in the [Section 1.7](#_Test_Cases_–_1). You should look at these carefully before coding.
2. The output for all the six test cases is shown in [Section 1.8](#_Output_for_Test_1). You should look at each output and correlate with the test cases to make sure you understand what the *readMartians* method is supposed to do.
3. **Now, write the *readMartiansFile* method in the *MartainManagerIO* class. If you are not clear on how to get started, see the** [**next section**](#_Hints)**.**
4. Open the *MartainManagerIOTest* class, comment out all the *testReadMartians…* methods, except the first. Run the method and verify the output. Then, run the other *testReadMartians…* methods incrementally.

## Hints

### Algorithm

Consider the general read algorithm in Ch 7, Section 7.3. Instead of a list, you have a *MartianManager*, thus, you’ll need to create that. Once you split the line into tokens, the general idea is:

If the data is valid

Create the appropriate type of martian

If you can add it to the martian manager

Increment appropriate statistic

Else

Increment appropriate statistic

Else

Increment the appropriate statistic

However, I’m not sure I would code it that way. However, I think that this is a useful way to understand the problem.

### Detecting Invalid Data

Next, how can data be invalid?

* Invalid code – check the first token to see if it is “G” or “R”
* Too few or many parameters – Depending on whether code is “G” or “R”, check the size of the tokens array
* Non-numeric id, volume, or tenacity – Use a helper method as shown in Ch 7, Sec 8.

### Detecting Duplicates

How do you know if a valid martian has already been added (*i.e.* a duplicate)? The *MartianManager’s addMartian* method returns *true* if the martian was successfully added and *false* if it is a duplicate.

### An Approach to Implementation

1. Write code that doesn’t detect any invalid situations – it simply reads valid martians, creates them, and puts them in the *MartianManager*. When you need to create the *ReadReport* to return, simply put in 0 for all the statistics. The test files: *inMartians1.text* and *inMartians2.txt* have only valid martians. Test and debug as necessary.
2. Add code to keep track of: *numLinesRead*, *numSuccessfullyAdded, numAlreadyExist* (there won’t be any of those in the first two text files). Be sure an put these values into the *ReadReport* at the end.
3. Write code that handles an invalid code, but everything else is OK. Be sure and collect the *numIllFormed* statistic. Test against: *inMartians3.txt* has. Test and debug as necessary.
4. Write code that also handles too few or too many parameter, but the parameters themselves are correct (integers) as test file: *inMartians4.txt* has. Test and debug as necessary.
5. Write code that also handles invalid parameters (*i.e.* non-integers) as test file: *inMartians5.txt* has. Test file: *inMartians6.txt* all the different types of errors and is the only test file that has a duplicate. Test and debug as necessary. Note, your method is going to be getting big now, perhaps with some duplicated code. Now would be the time to think how to use a helper method for this requirement, and possibly the others. Sometimes, I will just write it all out, with duplication, get it to work, then refactor. Refactor means to make it easier to read and usually employing helper methods with descriptive names.

## Test Cases – Input Files

You have been provided 6 martian text files that are used in *MartianManagerIOTest.* This is a summary of the files.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | Title | Small-All Correct | | File | *inMartians1.txt* | | Data | G 3 5  R 1 4 2 | | |  |  | | --- | --- | | Title | Large-All Correct | | File | *inMartians2.txt* | | Data | G 3 5  R 1 4 2  R 9 3 4  G 7 7  R 2 9 3  G 8 2  G 4 5 | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | Title | Invalid Martian Code | | File | *inMartians3.txt* | | Data | G 3 5  V 1 4 2 // Invalid code  R 9 3 4  G 7 7  R 2 9 3  GREEN 8 2 // Invalid code  G 4 5 | | |  |  | | --- | --- | | Title | Invalid Num Parameters | | File | *inMartians4.txt* | | Data | G 3 5  R 1 2 // too few  R 9 3 4  G 7 7 9 // too many  R 2 9 3 8 // too many  G 8 2  R 5 6 // too few  G 4 5 | |

|  |  |
| --- | --- |
| Title | Invalid Parameters |
| File | *inMartians5.txt* |
| Data | G 3 5  R 1 4j 2 7 // non-integer parameter & to many  R 4 5 6  G 5 6  G six two // non-integer parameter |

|  |  |
| --- | --- |
| Title | Mixed Problems |
| File | *inMartians6.txt* |
| Data | G 3 5  R 1 4 2  R 3 4 9 // Duplicate  R 9 2 // Too few parameters  G 4 5  R 2 8 7  G 99 // Too few parameters  Green 88 99 // Invalid code  How // Invalid  R 6 6 6 6 6 6 // Too many parameters  G 6 6 |

## Output for Test Cases

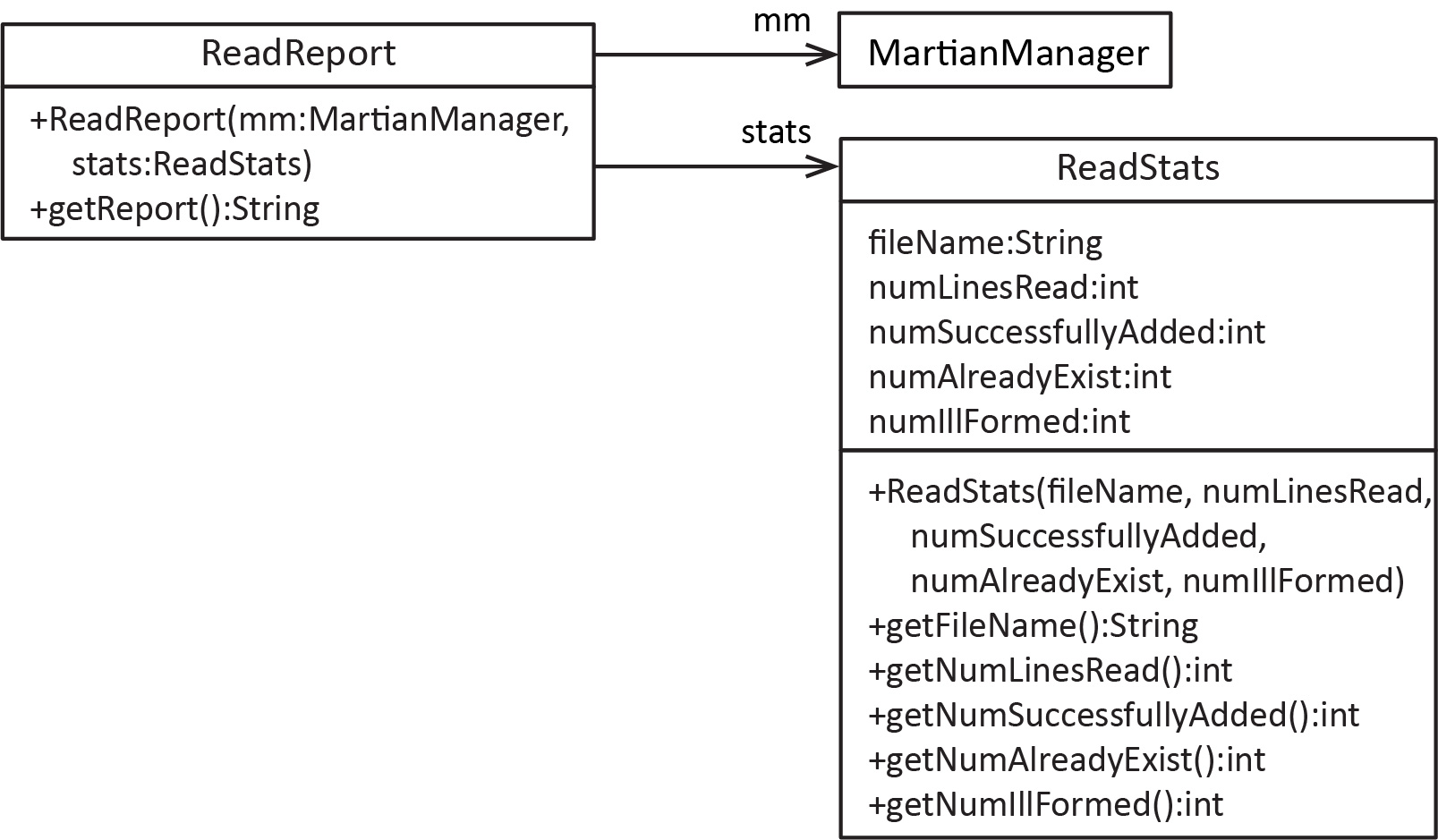
The correct output is shown below. Your code should produce these exactly results when the test code is run.

|  |  |  |
| --- | --- | --- |
| \*\*\* Read Report \*\*\*  File: inMartians1.txt  Martian Manager:  Martians:  Green Martian - id=3 vol=5  Red Martian - id=1 vol=4, ten=2  Teleporters:  Green Martian - id=3 vol=5  Martian Manager:  Num lines read :2  Num Martians added :2  Num ill-formed lines :0  Num already exist (not added):0  \*\*\* Read Report \*\*\*  File: inMartians2.txt  Martian Manager:  Martians:  Green Martian - id=3 vol=5  Red Martian - id=1 vol=4, ten=2  Red Martian - id=9 vol=3, ten=4  Green Martian - id=7 vol=7  Red Martian - id=2 vol=9, ten=3  Green Martian - id=8 vol=2  Green Martian - id=4 vol=5  Teleporters:  Green Martian - id=3 vol=5  Green Martian - id=7 vol=7  Green Martian - id=8 vol=2  Green Martian - id=4 vol=5  Martian Manager:  Num lines read :7  Num Martians added :7  Num ill-formed lines :0  Num already exist (not added):0  \*\*\* Read Report \*\*\*  File: inMartians3.txt  Martian Manager:  Martians:  Green Martian - id=3 vol=5  Red Martian - id=9 vol=3, ten=4  Green Martian - id=7 vol=7  Red Martian - id=2 vol=9, ten=3  Green Martian - id=4 vol=5  Teleporters:  Green Martian - id=3 vol=5  Green Martian - id=7 vol=7  Green Martian - id=4 vol=5  Martian Manager:  Num lines read :7  Num Martians added :5  Num ill-formed lines :2  Num already exist (not added):0 | \*\*\* Read Report \*\*\*  File: inMartians4.txt  Martian Manager:  Martians:  Green Martian - id=3 vol=5  Red Martian - id=9 vol=3, ten=4  Green Martian - id=8 vol=2  Green Martian - id=4 vol=5  Teleporters:  Green Martian - id=3 vol=5  Green Martian - id=8 vol=2  Green Martian - id=4 vol=5  Martian Manager:  Num lines read :8  Num Martians added :4  Num ill-formed lines :4  Num already exist (not added):0  \*\*\* Read Report \*\*\*  File: inMartians5.txt  Martian Manager:  Martians:  Green Martian - id=3 vol=5  Red Martian - id=4 vol=5, ten=6  Green Martian - id=5 vol=6  Teleporters:  Green Martian - id=3 vol=5  Green Martian - id=5 vol=6  Martian Manager:  Num lines read :5  Num Martians added :3  Num ill-formed lines :2  Num already exist (not added):0  \*\*\* Read Report \*\*\*  File: inMartians6.txt  Martian Manager:  Martians:  Green Martian - id=3 vol=5  Red Martian - id=1 vol=4, ten=2  Green Martian - id=4 vol=5  Red Martian - id=2 vol=8, ten=7  Green Martian - id=6 vol=6  Teleporters:  Green Martian - id=3 vol=5  Green Martian - id=4 vol=5  Green Martian - id=6 vol=6  Martian Manager:  Num lines read :11  Num Martians added :5  Num ill-formed lines :5  Num already exist (not added):1 | \*\*\* Write Report \*\*\*  G 1 2  R 2 3 4  R 3 4 5  G 4 5  R 5 6 7  \*\*\* Read Report \*\*\*  File: outMartians.txt  Martian Manager:  Martians:  Green Martian - id=1 vol=2  Red Martian - id=2 vol=3, ten=4  Red Martian - id=3 vol=4, ten=5  Green Martian - id=4 vol=5  Red Martian - id=5 vol=6, ten=7  Teleporters:  Green Martian - id=1 vol=2  Green Martian - id=4 vol=5  Martian Manager:  Num lines read :5  Num Martians added :5  Num ill-formed lines :0  Num already exist (not added):0 |

## Extra Credit

To earn up to 10 extra points on this assignment:

1. Extract the read statistics from the *ReadReport* class into its own class, *ReadStats* and compose these two classes as shown below.
2. Modify *ReadReport* to make the *stats* instance variables *protected.*
3. Modify *getReport* to use the *stats* instance variable.



## Grading Criteria

|  |  |
| --- | --- |
| **Weight** | **Description** |
| 9% | Read: Test file 1 |
| 9% | Read: Test file 2 |
| 8% | Read: Test file 3 |
| 8% | Read: Test file 4 |
| 8% | Read: Test file 5 |
| 8% | Read: Test file 6 |
| 10% | Write |
| **60%** | **Total** |

Plus, up to 10 extra credit points, if attempted.

# Problem 2

## Overview

(40 points) For this problem you will read a file of numbers adding them as you go, but skipping some of the number as indicated by a code in the file.

In the *NumberAdder* class, you will write the *getSum* method, which returns the sum of the integers read from a file. However, some numbers are skipped. When you encounter a line whose first character is ‘s’, then an integer will follow it. The integer tells you how many of the next consecutive numbers should be skipped. For Example 1 below, the yellow numbers should be added to produce 60, skipping the values as indicated. As shown in Example 3, a skip of zero doesn’t skip anything, effectively ignoring the skip. Example 4 shows a skip beyond the length of the file which should just skip to the end of the file, ignoring the invalid skips. Finaly, Example 5 shows overlapping (or embedded) skips. The embedded skip, s5, is effectively ignored, meaning it is treated as any other skipped value.

|  |  |  |  |
| --- | --- | --- | --- |
| Example 1 – Sum=60 | Example 2 – Sum=90 | Example 3 – Sum=10 | Example 5 – Sum=10 |
| E:\Data-Classes\CS 1302 - Programming 2\homework\Fall, 2015\hw7\a.jpg | s5  2  43  3  6  9  12  4  56  s2  10  4  8  s2  1  5  s2  4  2  10 | 2  3  s0  4  1 | 2  3  s3  4  s5  2  4  1 |
| Example 4 – Sum=5 |
| 2  3  s7  4  1 |

## Details

In the *prob2* package you will find a *NumberAdder* class as shown below. Write the *getSum* method (stub provided). This is also the test class as *main* calls *getSum* 10 times, passing it a different text file each time.

**public** **class** NumberAdder {

**private** **static** **final** String ***IN\_FILE\_PATH*** = "src/prob2/";

// I have supplied test files: t1.txt, t2.txt, ..., t10.txt

// You will need to verify the expected output by hand to ensure

// that your code gives the correct value.

**static** String[] *inFileNames* = {"t1.txt", "t2.txt", "t3.txt", "t4.txt", "t5.txt",

"t6.txt", "t7.txt", "t8.txt", "t9.txt", "t10.txt"};

**public** **static** **void** main(String[] args) **throws** FileNotFoundException {

// Comment out most of these as you test/debug.

File file;

file = **new** File(***IN\_FILE\_PATH*** + *inFileNames*[0]);

System.***out***.println("Sum=" + *getSum*(file));

file = **new** File(***IN\_FILE\_PATH*** + *inFileNames*[1]);

System.***out***.println("Sum=" + *getSum*(file));

file = **new** File(***IN\_FILE\_PATH*** + *inFileNames*[2]);

...

}

/\*\*

\* YOU WRITE THIS METHOD

\*/

**public** **static** **int** getSum(File file) {

**return** Integer.***MAX\_VALUE***;

}

...

}

## Problem 2 – Suggested Steps

If you need help getting started, I’d use baby steps:

1. Create a simpler text file to start with:
   1. Write a text file named: *test.txt* with these values: 1 2 a 3 4 5
   2. Comment out all code in *main.*
   3. Add these lines to *main*:

File file;

file = **new** File(***IN\_FILE\_PATH*** + *test.txt*);

System.***out***.println("Sum=" + *getSum*(file));

1. Write *getSum* so that it simply adds up all the numbers in the file. Thus, you need to read each token as a string, and then ask if it is an integer. If it is, add it. If it is not, then skip it. See Chapter 7, Section 8 to see how to determine if a string is an inteter.
2. Run, and verify that the output is 15. If not, debug and fix your code.
3. Modify *test.txt* so that the values are: 1 2 a b 3 c 4 5.
4. Run and verify that the output is 15. If not, debug and fix your code.
5. Add a skip to *test.txt*: 1 2 s2 3 4 5. Thus, the expected result is 8 once you modify *getSum.*
6. Modify *getSum* to detect and obey the skip:
   1. When you detect that a token is not an integer, then you can assume it is a skip, *i.e.* the first character is an ‘s’. So, write code to strip off the strip length. See Chapter 1, Appendix 1.4 to see how to use the substring method.
   2. Put an inner *for* loop (*i.e.* inside the while(scanner.hasNext()) loop) that goes from 0 to less than the strip length. Each time it should simply read the next token and do nothing with it.
   3. Run and verify that the output is 8. If not, debug and fix your code.
7. Comment out the two lines in *main* and then uncomment the first test case:

File file;

file = **new** File(***IN\_FILE\_PATH*** + *inFileNames*[0]);

System.***out***.println("Sum=" + *getSum*(file));

1. Run and verify the output. If incorrect, then debug and fix your code.
2. Repeat steps 8 and 9 with the second test case, then the third, *etc.*

# Submission Requirements

Checklist:

|  |  |  |
| --- | --- | --- |
|  | **Complete?** | **Requirement** |
| 1. |  | The Academic Honesty statement followed by your full name, appears as a comment at the top of the *MartianManagerIO* and *NumberAdder* classes. |
| 2. |  | All Java files (including test classes and textfiles) are in the *prob1* and *prob2* packages, respectively. |
| 3. |  | Your *prob1* and *prob2* folders are zipped into a file name: *hw6\_yourLastName.zip.*   * See Lab 2, Stage 9 for exact instructions. * Do not zip your workspace folder * Do not zip your *src* folder. * Do not zip just the java files * Do zip just your *prob1* folder |
| 4. |  | Submit in the *hw6* dropbox on Blazeview by the deadline. |