School of Computing Sciences

Module: CMP-4008Y Programming 1
Assignment: Coursework 1 - Messier Catalogue

Set by: Dr Gavin Cawley (g.cawley@uea.ac.uk)
Date set: 8th November 2022
Value: 35%

Date due: 9th December 2022 3pm
Returned by: 16th January 2023
Submission: Blackboard
Checked by: Dr Jason Lines J.Lines@uea.ac.uk

Learning outcomes

The aim of this assignment is for the student to gain experience in the design and implementation of a relatively small program using an object oriented approach. This will further reinforce the student's grasp of fundamental concepts such as classes, objects, instance variables and methods, but will also introduce the use of multiple classes, each of which defines a different type of object. During the assignment, the student will also begin to gain object-oriented design skills by deciding how the state and behaviours of the objects should be implemented and how the objects should interact.

On successful completion of this exercise, the student will have reinforced a basic understanding of the concepts of classes and objects, will be familiar with the basic syntax of Java programming constructs used to implement classes, instance variables and methods and will be able to select the instance variables and methods required to represent the state and behaviours of simple types of real-world objects. The student will have gained familiarity with some of the classes in the Java standard libraries used for file based I/O and collections. The student will also have an improved ability to use the ideas of cohesion and coupling to guide the design of programs comprised of more than a single class definition.

Specification

Overview

Charles Messier was a French astronomer of the 18th and early 19th century. He was best known for the Messier Catalogue a list of permanent diffuse objects in the night sky of particular interest to astronomers. The list contains objects of various different types, including star clusters, nebulae and galaxies. You are required to implement a Java program to load information about a collection of *Messier objects* from file, and answer a set of queries about them.

Description

The program must consist of *at least* the following classes:

1. `MessierObject` is a class used to store all of the information about a *Messier object*, including:

   - Messier number;
   - NGC/IC number;
   - Common name for the object (where one exists);
   - Type, e.g., planetary nebula, supernova remnant, globular cluster etc.
   - Distance in thousands of light years (kly);
   - Constellation in which it is found;
   - Apparent magnitude - an indication of how bright it appears in the night sky;
   - Right ascension and declination - the two angles that astronomers use to locate the object. Note that these are represented in different formats in the file.

   The right ascension and declination must be stored as `double` precision floating point numbers to facilitate efficient astronomical calculations. A suitable set of constructors and accessor methods should be provided. The class must implement the *Comparable* interface, to allow `MessierObjects` to be sorted in order of apparent magnitude. The class must have a `toString` method that produces a `String` representing the object in the same format used in the data file.

2. `MessierCatalogue` is a class used to store information about a number of *Messier objects*. It should have a member variable that is a collection of `MessierObject` objects. It should have methods that can be used to answer general queries regarding the content of the catalogue (so that the program is easily extensible to answer other queries, not just those given below in the current specification). It should also have a suitable set of constructors and accessor methods and a `toString` method that outputs a `String` containing the entire catalogue in the format used in the datafile (so that input and output formats are compatible).

3. `MessierProgram` which contains the `main` function, that reads in a `MessierCatalogue` from the file `messier.txt` (available on BlackBoard) and gives answers to the following five queries:

   (a) Display the catalogue, arranged in order of ascending apparent magnitude (brightest object first).

   (b) Display the average apparent magnitude of all open clusters.

   (c) Display the details of the most distant globular cluster.

   (d) Display the details of the object in the constellation Sagittarius with the lowest declination.

   (e) Display the details of the object that is closest in the sky to M45 (in the sense of having the smallest angular distance). Hint: test your code using M42 (the Great Orion Nebula) as M42 and M43 (De Mairan's Nebula) are both are part of the Orion molecular cloud complex, and so they should have a small angular separation (approximately 8 arc minutes).

The program should be written such that the main function should give a clear top-down overview of what the program does, without being overly cluttered with details of *how* it is done (demonstrating the advantages of *abstraction* and *encapsulation*). Each class should have a separate `.java` file for ease of compilation.

Hints:

- Tackle the problem one class at a time. I would start with `MessierObject`; have it fully implemented and tested before moving on to `MessierCatalogue`. Only once you have implemented and tested classes representing each type of object, should you try to implement the `MessierProgram` itself. Note this is an example of "top-down design, bottom-up implementation" discussed in the lectures.

- The science fiction writer Robert A. Heinlein wrote *"... when faced with a problem you do not understand, do any part of it you do understand, then look at it again."*, which sometimes can be a good approach to programming. For instance, the program will require the ability to convert a string containing a declination expressed in degrees, arcminutes and arcseconds, e.g. `+22° 00' 52.2''`, into an angle in radians. This is a relatively straightforward mathematical task that can be tackled in isolation, without needing to think about the rest of the program. So implementing a static method to perform this conversion would be an identifiable sub-problem you could solve without needing to understand the overall structure of the program. Once it has been implemented and tested, you can get on with implementing the rest of the program without ever having to think how this conversion is implemented again. This approach really helps deal with the complexity of writing a non-trivial program.

- For each class, it is a good idea to work out what methods the class should provide in order to provide the services required to implement the program, *before* starting to code.

- Make sure you have a good idea of what you have to do *before* you start writing code. This is especially true for aspects of the project that require some mathematical research, such as the representation of angles for the right ascension and declination, and computing the angular distance.

- It is important to test each class in isolation (don't write all the code for all three classes and expect them to work first time). `MessierObject` and `MessierCatalogue` could have a `main` method, used as a *test harness*.

- The implementation of `MessierCatalogue` requires the use of arrays, or more advanced types of collection, which we will cover in later lectures, However `MessierObject` can be implemented and tested based on the material already covered. Reading in the file should be left until last as it will be covered after arrays and `ArrayLists`

- It simplifies the design if `MessierObject` has a constructor that takes a `String` argument containing a line from the file.

- If you have done some programming before in a language other than Java, do not try to implement the program in the style used in a different language. Programming languages have different purposes and different programming styles. If you try and write in another style (particularly that used in Python), the task will be more difficult, and will not meet the learning objectives (and hence will attract lower marks).

All questions regarding the specifications **must** be asked via the appropriate discussion board on BlackBoard. Note that `messier.txt` may change before the due date for the assignment, and the answers to the queries must be based on the final version of that file.

Relationship to formative assessment

This assignment builds on the skills gained from the laboratory exercises for cmp-4008Y that have already been completed, for which formative feedback can be obtained from the teaching assistants during your scheduled laboratory session.

Deliverables

Your solution **must** be submitted via blackboard. The submission **must** be a single `.pdf` file generated using PASS, using the PASS server at `http://pass.cmp.uea.ac.uk/` . The PASS program formats your source code, and compiles and runs your program, appending any compiler messages and the output of your program to the `.pdf` file. If there is a problem with the output of PASS, contact me (gcc@uea.ac.uk). Do not leave it until the last moment to generate the `.pdf` file, there is a limit to the amount of help I am able to give if there is little time left before the submission deadline.

PASS is the target environment for the assignment. If your program doesn't operate correctly using PASS, it doesn't work, even if gives the correct answers on your own computer:

- The PASS program is not able to provide input to your program via the keyboard, so programs with a menu system, or which expect user input of some kind are not compatible with PASS. Design your program to operate correctly without any user input from the keyboard.
- Do not use absolute path names for files as the PASS server is unable to access files on your machine. If your program is required to load data from a file, or save data to a file, the file is expected to be in the current working directory. If the program is required to load data from a file called `rhubarb.txt` then the appropriate path name would be just `"rhubarb.txt"` rather than `"C:some\long\chain\of\directories\rhubarb.txt"`.
- If you develop your solution on a computer other than the laboratory machines, make sure that you leave adequate time to test it properly with PASS, in case of any unforeseen portability issues.
- If your program works correctly under InteliJ on the laboratory machines, but does not operate correctly using PASS, then it is likely that the data file you are using has become corrupted in someway. PASS downloads a fresh version of the file to test your submission, so this is the most likely explanation

Resources

- `https://stackoverflow.com/` - An excellent site for finding information about specific issues relating to various programming languages, including Java. It is important however not to become too reliant on sites such as `StackOverflow`, which are great for details, but don't give the "big picture", so it is difficult to get a good understanding of programming in this way. Note that if you re-use or modify code found on-line, then you must provide a comment giving the URL and a brief explanation of what modifications have been made. Re-using code found on-line without proper attribution would constitute plagiarism.
- `https://docs.oracle.com/javase/tutorial/` - A set of tutorials, provided by Oracle, who now develop the Java programming language.

- https://docs.oracle.com/en/java/javase/17/docs/api/index.html - Java Application Programming Interface (API) documents. Very useful for finding out what methods a class from the Java standard libraries provide.

Marking Scheme

Marks will be awarded according to the proportion of specifications successfully implemented, programming style (indentation, good choice of identifiers, commenting etc.), and appropriate use of programming constructs. It is not sufficient that the program generates the correct output, professional programmers are required to produce maintainable code that is easy to understand, easy to debug when (rather than if) bug reports are received and easy to extend. The code needs to be modular, where each module (e.g. class) has a well-defined interface to the rest of the program. The function of modules should be made as generic as possible, so that it not only solves the problem specified today, but can easily be modified or extended to implement future requirements without undue cost. The code should also be reasonably efficient. Marks may also be awarded for correct use of more advanced programming constructs not covered in the lectures.

The marking scheme is as follows (out of 35 marks):

- Class `MessierObject` - 10 marks;
- Class `MessierCollection` - 10 Marks;
- Class `MessierProgram` - 5 Marks;
- Answers to the five queries - 2 marks each (total of 10 marks).

Plagiarism, collusion, and contract cheating

The University takes academic integrity very seriously. You must not commit plagiarism, collusion, or contract cheating in your submitted work. Our Policy on Plagiarism, Collusion, and Contract Cheating explains:

- what is meant by the terms 'plagiarism', 'collusion', and 'contract cheating'
- how to avoid plagiarism, collusion, and contract cheating
- using a proof reader
- what will happen if we suspect that you have breached the policy.

It is essential that you read this policy and you undertake (or refresh your memory of) our school's training on this. You can find the policy and related guidance here:

https://my.uea.ac.uk/departments/learning-and-teaching/students/academic-cycle/regulations-and-discipline/plagiarism-awareness

The policy allows us to make some rules specific to this assessment. Note that:

> In this assessment, you are permitted to work within groups proscribed by the assessment setter. Discussing solutions to tasks *between* groups, or groups otherwise working together to perform the assessed tasks will be considered as a breach of university regulations. Please pay careful attention to the definitions of contract cheating, plagiarism and collusion in the policy and ask your module organiser if you are unsure about anything.