CO320 Assignment 2 (Version 2018.11.16)

Introduction

This assignment is designed to allow you to practice writing class definitions that make use of collections and loops. It includes an element that should be considered challenging. You will hand the assignment via the CO320 Moodle page.

Download the starting project from:

https://www.cs.kent.ac.uk/~djb/co320/LOCAL-ONLY/assign2.zip

Date set: Thu 8th November 2018
Deadline: 23:55 Thu 22nd November 2018
Weighting: 15% of the module's coursework mark.

Please make an effort to read through the assignment as soon as possible so that you know what you need to understand to complete it, and can think about how best to organise your time.

Plagiarism and Duplication of Material

The work you submit must be your own. We will run checks on all submitted work in an effort to identify possible plagiarism, and take disciplinary action against anyone found to have committed plagiarism.

Some guidelines on avoiding plagiarism:

- One of the most common reasons for programming plagiarism is leaving work until the last minute. Avoid this by making sure that you know what you have to do (that is not necessarily the same as how to do it) as soon as an assessment is set. Then decide what you will need to do in order to complete the assignment. This will typically involve doing some background Taxi and programming practice. If in doubt about what is required, ask a member of the course team.

- Another common reason is working too closely with one or more other students on the course. *Do not* program together with someone else, by which I mean do not work together at a single PC, or side by side, typing in more or less the same code. By all means *discuss* parts of an assignment, but do not thereby end up submitting the same code.

- It is not acceptable to submit code that differs only in the comments and variable names, for instance. It is very easy for us to detect when this has been done and we will check for it.

- **Never** let someone else have a copy of your code, no matter how desperate they are. Always advise someone in this position to seek help from their class supervisor or lecturer. Otherwise they will never properly learn for themselves.
- It is not acceptable to post assignments on sites such as Freelancer and we treat such actions as evidence of attempted plagiarism, regardless of whether or not work is payed for.

Further advice on plagiarism and collaboration is also available.

You are reminded of the rules about plagiarism that can be found in the Stage I Handbook. These rules apply to programming assignments. We reserve the right to apply checks to programs submitted for assignment in order to guard against plagiarism and to use programs submitted to test and refine our plagiarism detection methods both during the course and in the future.

The Task

Imagine that you have been asked to write part of some software that will be used to manage voting for a Sports Personality competition. You have been provided with the `Personality` class as a starting point and your task is to write the `Competition` class:

- The `Personality` class represents details of a sporting personality. It has two fields of type `String` called `name` and `sport` and one field of type `int` called `votes`. Further details are given below. You can use the 'test fixture to object bench' menu option of the `CompetitionTest` class to put a sample of `Personality` objects onto the object bench for help with testing your code.
- The `Competition` class remains to be written. It must be able to store an `ArrayList` of `Personality` objects and provide a range of functionality on the list. Full details are given below.

To complete this task you will need to be familiar with collection classes and iteration, both of which are covered in chapter 4 of the course text. In particular, you will need to make use of the `ArrayList` class that is available in the `java.util` package, and understand how to write *for-each loops* and *while loops*. You will find it instructive to study the source code of the `music-organizer` project, which we have discussed in the course and is explored in detail in chapter 4 of the course text book.

You will also find it useful to look at the Java API documentation, which can be found here:

https://docs.oracle.com/javase/8/docs/api/

In particular, select the link to `java.util` in the right-hand pane, and then scroll down to the link to `ArrayList<E>` to read about the methods of the `ArrayList` class.

In addition, there is a challenge element that will require you to seek out additional information should you choose to attempt it.

The CompetitionTest and PersonalityTest classes

Two additional classes are included with the `Personality` and `Competition` classes. These are called `PersonalityTest` and `CompetitionTest`. They are not part of your assessment and you do not have to add any further code to them. Rather, they are placeholders that can later be used to help you test the accurate implementation of the `Competition` class.

Towards the end of assessment period, an extended version of `CompetitionTest` will be made available to help you test your implementation of Competition. However, as noted above, you can already use the 'test fixture to object bench' menu option of the `CompetitionTest` class to put a sample of `Personality` objects onto the object bench to help with testing as you develop your program.

The Personality class

This provided class is fairly simple and should not require any change. It consists of:

- One constructor that takes two parameters of type `String` that are used as the initial values of its `name` and `sport` fields. The `votes` field is initialised to zero.

- An accessor method for each field.

- A method called `increaseVotes` that takes a single `int` parameter and returns nothing. The method adds the value of the parameter to the value of the `votes` field if the parameter's value is greater than zero.

- A method called `getDetails` that takes no parameters and returns a `String`. It returns a `String` containing the values of the fields in the following format:

  ```
  "Lewis Hamilton takes part in F1 and has 500 votes."
  ```

  where `"Lewis Hamilton"` is an example of the value of the `name` field, `"F1"` is an example of the value of the `sport` field and `500` is an example of the value of the `votes` field. Different `Personality` objects will have different values for these fields.

Requirements

Your task is to write the `Competition` class as described below. The work will be marked out of 100 and provisional marks are indicated against each element of the assessment. Pay careful attention to the exact details of the requirements; such as the names of methods and the parameters required.

Commenting and style [10 marks]

All the code you write must be well laid out and the fields, methods and constructors must be properly documented, using the style that is used in this course. If in doubt, a style guide is available:

https://www.bluej.org/objects-first/styleguide.html

Adherance to these will be included in the marking scheme.

The Competition class – basic methods [30 marks]

The class must use an `ArrayList` to store any number of `Personality` objects. In the class write:

- A method called `addPersonality` that takes a `Personality` object as a parameter and adds it to the `ArrayList`. You may assume that two personalities with the same name will never be added to the same list and there is no need to check this.

- A method called `getSize` that returns the number of `Personality` objects in the `ArrayList`.

- A method called `list` that prints out the details of all the `Personality` objects stored in the `ArrayList`, one per line. Use the `Personality` objects' `getDetails` method to format the details. The printed list must **not** include the index of each `Personality` in the list.

The Competition class – iterative methods [30 marks]

In the `Competition` class write:

- A method called `voteFor` that takes a single parameter of type `String` called `name` and does not return anything. This method locates the single `Personality` object in the `ArrayList` whose `name` field *exactly* matches the contents of the method's `name` parameter, and then adds 1 to the number of votes for that personality.

  Note that the match required is a character-by-character match of two strings. You must assume that at most one `Personality` will have the matching name but there might be no matching `Personality`. If no matching `Personality` is found then the method must print an error message.

- A method called `shortlist` that takes a single `int` parameter called `minimumVotes` and does not return anything. This must remove from the list all `Personality` objects that have *fewer* votes than the value of the parameter. There could be zero or more objects removed by this method.

The Competition class – challenge method [30 marks]

Important note: This element is considerably harder than the others and might require you to seek additional information to complete it. You should consider carefully whether the time it could take you to complete it is justified given the percentage of marks available for it.

In the `Competition` class write:

- A method called `getMost` that takes a single integer parameter representing the number of Personality objects to return. The method returns an `ArrayList` of those `Personality` objects. (However, see the examples below for cases where the number of objects in the returned list might be more or less than the parameter's value.) You may assume that the parameter's value will be greater-than or equal-to 0 but it might be greater than the number of items in the `Competition` object's list.

  This method must return an `ArrayList` containing the `Personality` objects of the `Competition` with the most votes. The `ArrayList` returned must be a different one from the one in the `Competition` object but the `Personality` objects will be the same ones. For instance, if the parameter is 3 then the method must create a new `ArrayList`, store within it the three `Personality` objects that have the highest number of votes, and return that list.

Hint: In order to complete this requirement you will probably want to sort the list of personalities by number of votes. However, you should try to avoid writing your own sorting code because there are library methods to do it for you. Investigate how to sort an `ArrayList`. The list returned by `getMost` does not have to be sorted.

When writing this method you should consider the following special cases:

- If the parameter's value is greater-than or equal-to the number of personalities in the list then all of the personalities will be returned.

- There might be a tie for votes, which could mean that the returned list might contain more objects than the parameter value. For instance, if the top 3 are required and the top five personalities have 100, 98, 50, 50 and 42 votes, then four personalities must be returned. On the other hand, if the top 3 are required and the top five have 100, 100, 98, 50, 50 and 42 votes then exactly three would be returned.

Confidence in the completely correct implementation of this method will only be possible through thorough testing.

Starting the assignment

This section contains some advice on how to get started with the assignment. In the first assignment, we encouraged you to keep checking whether your code still compiles after each change. It will be important to adopt the same approach with this assignment. We won't keep repeating this advice from now on.

The Competition class

Start with an initially empty class. Add your name and date to the opening class comment.

Add a field

Add a field to `Competition` for storing the collection of `Personality` objects. You will need to use an import statement at the top of the file. (See the `MusicOrganizer` class in chapter 4 if you are uncertain how to do this.)

Constructor

Complete the constructor for the class. The constructor must take no parameters. The constructor must initialise the collection field.

Methods

The order of the methods listed in the requirements section is a reasonable guide to the order in which you should try to implement them. For instance, start with a method to add to the collection, then a method that tells you how many objects are in the collection. After adding each method, compile the code, fix any errors and test that it works. Use inspectors and the debugger to give yourself confidence that the code is working as you think it should.

Getting help

If you need help with the assignment, there are various places you can go. You will not lose marks if you ask for help.

- You can ask questions on the module's anonymous Q&A page and it is ok to post code there because I will edit it before publishing.
- You can ask for assistance from your class supervisor.
- You can ask the module's lecturer.
- You can turn up at the homework club: https://www.cs.kent.ac.uk/teaching/student/support/homework/

Be very careful about asking for help from other students on the course, or students who have taken the course before, other than as described above. If in doubt about what is appropriate, check the plagiarism guidelines above and/or ask the module's lecturer.

Finally

Before you submit your assessment, thoroughly test the whole class to make sure that nothing you added later has broken anything added earlier. If you are unable to complete the class – even if you cannot get it to compile – still submit what you have done because it is likely that you will get at least some credit for it.

David Barnes, 7th November 2018
Updated 16th November to clarify the requirements for the getMost method.