Java Practical Exercise 1 – Spring 2021

This is a description of the first Java practical exercise for CSCU9T4 in Spring 2021. It is expected that you will work on this over approximately 2 weeks, on your own and in groups during weekly live sessions.

The aim of this practical sessions is to exercise your skills with structured data, use of inheritance, recognising good and bad coding practices, and working with a version control system (Git).

You'll do that by enhancing a reduced version of a Training record program. This program is to support a sports trainer who wants to keep a record of the times and distances her athletes run over the season. You are given a simple program with the ability to let you add and look up records over the years. Dates should be entered using DD MM YYYY format. Times in hh:mm:ss format.

The existing program is a Netbeans Maven project which makes use of Java library classes such as ArrayList and processes the contents of a list using a ListIterator. Your extensions must also use iterators and parameterised types. We assume that you will be using the Java API and online sources to gain more information about iterators and their use. Your extensions should be coded as elegantly and efficiently as possible. Using Netbeans is not required but a GitHub account will be.

While support on this exercise between students is encouraged, copying of code between class members is still an offence and considered as plagiarism. In order to avoid plagiarising (even inadvertently) we recommend you adapt the following approach:

1. Attempt each task on your own first
2. Use the group sessions to share general issues and solutions
    a. **Do**, share screens to demonstrate solutions you're proud of or to get fresh pair of eyes on a particular code you are struggling with.
    b. **Do not**, copy/paste code segments or share source files

The practical exercise contains 3 main parts:

1. Git version control
2. Programming
3. Code review

1. Git: Setting up the exercise.

This part is just about accessing the initial reduced version of the Training program which you will extend.

a) If you haven't already, sign up for a GitHub account. Preferably using your Stirling student email address so that we can be sure it is your account.
b) Go to this repository: https://github.com/saemundur-haraldsson/CSCU9T4Practical1
c) Fork the repository to your own account.
d) [Recommended] Create a new branch
e) Checkout a copy of the new branch to your machine and start working on the implementation

2. Programming (Submit by 5. February)

Introduction

Assuming you have already downloaded the program, run and test it. Consider the edge cases, and how the program performs with no data in the Training record, with just one entry, with lots of entries. Be thorough.
The program is deliberately rather basic and not very robust to incorrect data entries. The Add button is used to add a new entry, represented as Name, Month, Day, Year, Hours, Minutes, Seconds and Distance to the list. The LookUpByDate button is used to find a single entry on a given day/month/year combination. The final version of the program should be able to keep record of triathletes' training sessions: run/sprints, long and short distance cycles, swims. For example:

- a run/sprints session might consist of, for example:
    a. 4 * 40m sprints (with 3 minutes recovery between),
    b. 4 * 300m sprints (with 6 minutes recovery between), or
    c. 1 * 5km run
- a cycle session might include information on distance, terrain, time and tempo.
- a swim session should record similar information to that on a run (distance, time), but you might also want to know if it was in the pool or outdoors.

The tasks

Following is a list of extensions you will be asked to implement over the next 2 weeks. Most of them are required while some are optional/recommended, those are a bit more advanced, but I urge you to attempt them as they will provide you with very helpful experience for the assignment. The extensions are not described in detail as part of the exercise is to design the structure and implement your own approach. Any detail that is described is required. Add your own tests to the test suite as you see fit.
I expect you to be working from a Git repository and thus I recommend you commit after each extension. Push those commits to remote a bit less frequently.

1. Add a new button FindAllByDate to the GUI. Add statements to the actionPerformed method to handle this button. Use a place holder, e.g., when it is pressed the string 'Not implemented yet' is displayed. (We're suggesting this, so you don't try to implement everything at once and get confused over what's working and what's not. Incremental development means that you make small changes and test they work as you go.
2. Consider lookupEntry. If you have more than one entry on a given day, which one does lookupEntry return? Implement a new method which returns all the entries on a given date in a single string. (The advantage of this is that all the work of constructing the output string is in the TrainingRecord class (rather than the TrainingRecordGUI class).
3. Use that new method to implement the functionality of the FindAllByDate button. Now when you enter a Day and Month and Year and click FindAllByDate, the program will display the names and times of all runs for that day (not just the first one). The output should be shown in the output area with one run entry per line.
4. **(optional challenge)** Experiment with the program to see what kinds of input it will accept. It is not very robust, is it? It fails, with an exception, if the value entered for Day, Month or Year is not an integer. And it accepts the empty string as a Name. It also allows you to add the same entries over and over. Improve the program so that

it handles bad input in a sensible way, for example, by displaying a suitable error message in the TextArea. You may assume that name, day, month, year is a unique key for running records (i.e., your athletes do at most one run per day).

5. **Here would be a great idea to push your commits to the** remote
6. Implement the additional classes needed to represent the different types of training sessions (run/sprints, cycle, and swim). You need to replace Entry by the new superclass and add new subclasses. Consider carefully the use of *redefinition* of methods to make preparing output very straightforward for each class. This is one of the benefits of using superclasses and subclasses.
7. Add tests for these
8. Amend the GUI to handle the new entry types. How will you add new entries? Do you need three buttons or is there another way to deal with this? Think about your reasons for making the choice the way you have. What are the design considerations? The practical considerations? How will you justify your choice?
9. **(Optional challenge)** Add a button to find all entries for a given name. How are you going to handle string matching? Do you require an exact match? A case-independent match? A partial match?
10. **Here would be a great idea to push your commits to the** remote
11. Add a Remove button to the GUI and implement its functionality. Now when you enter a Name, Month, Day and Year and click Remove, the Entry on that date for that person should be removed from the running record. (If you haven't already done so, you need to make adding a new record more robust, ensuring that you can't add a record with the same Name, Month, Day and Year as one already present. This is a rather simplistic way of ensuring unique entries. Normally you would consider a more realistic scenario in which a person can have multiple training sessions per day and you'd either add a unique entry number to distinguish between entries or use the time of day to distinguish between entries. This is not required for the purposes of this practical.)
12. Test your work, add tests to the test suite. Consider the edge cases, the weird pathological cases, the incorrect output, how the program performs with no data in the Training record, with just one entry, with lots of entries. Be thorough.
13. Improve the appearance of the GUI. This should include using the setEnabled method with the buttons to ensure that they can only be used when appropriate. For example, at the start, the only enabled button should be Add.
14. **(Optional extra – for more of a challenge)**. Add a new button Weekly Distance? When you click the button, the program should display all records for the named person for the last seven days (including today). You will need to investigate the Calendar class more fully to implement this part.
    Now you have all the right records, you just need to use the distance to calculate the total number distance for each class of exercise (swim, cycle, run) for that person for this week. Add to the Weekly Distance? output the distance this week. Be careful to add the right things together.

3. Code Review

Submitting your code for review

On or before 5. February 2021, you should commit all of your extensions to the program, push them to your remote repository on GitHub and merge into the Master branch.
Then you enter the URL to your repository in the submission
https://canvas.stir.ac.uk/courses/8826/assignments/55353
Preferably, make sure your repository is public at this point. If you do not feel comfortable in doing so, then let me (Saemi) know and we will make other arrangements.

Reviewing code

After 5. February 2021, Canvas will automatically assign you another student's submission to review. You are required to **submit your code before** you are allowed to review, and your code **must contain genuine attempts** for more than half of the requirements listed above.
Submit your review by 15. February 2021 (@23:59)
Navigate to your assigned fellow student's GitHub repository and inspect their code. You are not required to download, run, and/or test their code. However, it might make your task quicker. Compile a short list of comments on the code and submit that to the review on Canvas. Your notes may include for example:

- If there are any requirements missing
- Praise for any optional (advanced) extensions added
- Comments on code consistency (e.g., mixing camel case with other styles, etc.)
- Comments on structural choices (e.g., the use (or lack) of overriding/overloading, mixing responsibilities between classes, etc.)
- Comments on documentation (e.g., are there clear and concise comments? Too little or too much? Consistency in comments, all methods have Javadoc strings or none? Btw. none would be bad.)
- And anything else you can think of

Your review must contain **at least 2 praises and 2 constructive criticisms** (give advice on how to fix what you feel is lacking). There are no wrong reviews, but you must provide some justifications.
Reviews are anonymous