Department of Computer Science
University of Pretoria

Programming Languages
COS 333

Practical 7:

Object-Oriented Programming

October 24, 2023

1 Objectives

This practical aims to achieve the following general learning objectives:

- To gain and consolidate some experience writing object-oriented programs in Ruby;
- To consolidate a variety of basic concepts related to object-oriented programming languages, as presented in the prescribed textbook for this course.

2 Plagiarism Policy

Plagiarism is a serious form of academic misconduct. It involves both appropriating someone else's work and passing it off as one's own work afterwards. Thus, you commit plagiarism when you present someone else's written or creative work (words, images, ideas, opinions, discoveries, artwork, music, recordings, computer-generated work, etc.) as your own. Note that using material produced in whole or part by an AI-based tool (such as ChatGPT) also constitutes plagiarism. Only hand in your own original work. Indicate precisely and accurately when you have used information provided by someone else. Referencing must be done in accordance with a recognised system. Indicate whether you have downloaded information from the Internet. For more details, visit the library's website: `http://www.library.up.ac.za/plagiarism/`.

3 Submission Instructions

Upload your practical-related source code file to the appropriate assignment upload slot on the ClickUP course page. You must implement and submit your entire Ruby program in a single source file named `s99999999.rb`, where `99999999` is your student number. Multiple uploads are allowed, but only the last one will be marked. The submission deadline is **Tuesday, 31 October 2023, at 12:00**.

4 Background Information

For this practical, you will be writing a program in Ruby 2.5. The course website contains documentation related to Ruby [1]. You will be implementing all classes in a single source file.

In order to complete this practical, you will have to research various concepts in Ruby, which are relatively different to other object-oriented programming languages you will be used to. In particular, you will have to find out how constructors work in Ruby. You will also need to research accessor methods (attribute readers and writers), which are shorthand approaches for writing getters and setters. You will also have to investigate arrays in Ruby.

5 Practical Task

This practical requires you to build a simple system that represents students and courses. The following details must be included in your implementations:

A `Student` has a unique student number, a number of tests written, and an array of floating point marks. Each `Student` should provide only the following behaviour and any necessary constructors:

- `addTestScore(score)` which adds `score` to the list of marks and increments the number of tests written by the student.
- `getCurrentAverageMark()` which should be empty (it will be overridden in subclasses of `Student`).
- An attribute writer accessor method for the student number, which sets the student number of the student.
- An attribute reader accessor method for the student number, which returns the student number of the student.

Write a `Course` class. A `Course` object is composed of an array of `Student` objects. Additionally, the `Course` class should provide only the following behaviour:

- `addStudent(studentNumber, type)` which adds a student object with the given student number to the course. The second parameter, namely `type`, is a character indicating the type of the student (either full time or part time). You may use any characters to differentiate between the two types (although "`f`" and "`p`" can be used for consistency with the testing code described below).
- `addMark(studentNumber, mark)` which adds a test score to the student indicated by `studentNumber`.
- `getCurrentAverageMark(studentNumber)` which returns the current average mark of the student with the given student number.

Write two new classes, each of which is a subclass of `Student`. These classes should be called `PartTimeStudent` and `FullTimeStudent`. In the case of a `FullTimeStudent`, the worst mark is ignored when calculating the average mark. In the case of a `PartTimeStudent`, the best mark is ignored when calculating the average mark. Override the `getCurrentMark` method to accomplish this. Be sure to utilise as much functionality from parent classes as possible.

Finally, write test code that allows the user to type in the details for two students who are added to a course. These details include each student's student number, type of student (full time or part time), and three test scores. For the type of student, have the user type in either an "`f`" to indicate a full time student, or a "`p`" to indicate a part time student. The program should then repeatedly prompt the user for a student number, and print out the average mark for the specified student (or an error message if the student number is not present in the class). If the user types in a student number of -1, the program should exit. All this behaviour should be achieved through the `Course` class, instead of working directly with objects of the `Student` class (or any of its derived classes).

6 Marking

Submit the Ruby implementation to the appropriate assignment upload slot. Do not upload any additional files other than your source code. Both the implementation and the correct execution of the program will be taken into account. Your program code will be assessed during the practical session in the week of **Monday, 30 October 2023**.

References

[1] Huw Collingbourne. The book of Ruby. 2009.