Department of Computer Science  
University of Pretoria  
Programming Languages  
COS 333  

Practical Lab Experience 3: Expressions, Statements and Subprograms  

August 31, 2015

1 Objectives

This practical lab experience aims to achieve the following general learning objectives:

- To gain and consolidate some experience writing programs and scripts in several different object oriented languages, including: Ada and Ruby;
- To practically compare the above-mentioned object oriented languages in terms of expressions, assignment statements, control structures and subprograms.

2 Plagiarism Policy

The Department of Computer Science considers plagiarism as a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone else’s work without consent, copying a friend’s work (even with consent) and copying material (such as text or program code) from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to http://www.ais.up.ac.za/plagiarism/index.htm (from the main page of the University of Pretoria site, follow the Library quick link, and then click the Plagiarism link). If you have any form of question regarding this, please ask one of the lecturers, to avoid any misunderstanding. Also note that the OOP principle of code re-use does not mean that you should copy and adapt code to suit your solution.

3 Submission Instructions

Upload all practical-related files as a single tar-gzip or zip archive (named s?????????.tgz or s?????????.zip, where ????????? is your student number), to the appropriate assignment upload on the course website. Multiple uploads are allowed, but only the last one will be marked. The deadline is Monday, 14 September 2015, at 16:00. The archive must include all the program source files that you have written.

4 Background Information

For this practical, you will be writing programs in Ada and Ruby. You will have to compare these languages in terms of their support for different concepts related to expression, statements, control flow and subprograms. To do this, you will have to write short programs to demonstrate how each language handles the concept under consideration. These programs do not need to be long, but must demonstrate the concept adequately, and allow you to describe the language’s support (or lack of support) for the feature.

Ada is supported by GNAT, the GNU Ada Compiler extension for the GNU Compiler Collection Both Ada and Ruby are available in the labs. If you decide to use another compiler, please make sure that you test

---

1 A variety of official Ruby installations are available, for several different operating systems, from the download page of the official Ruby language home page, at http://www.ruby-lang.org/en/downloads/
your programs using the compilers installed in the labs. The course website contains documentation related to GNAT and Ada [1, 2], and the Ruby programming language [3].

5 Practical Task [10 marks]

This practical consists of three sections: Section 5.1 contains a program that must be implemented in both Ada and Ruby. Section 5.2 contains additional features that must be added to the Ada program to illustrate features specific to Ada. Section 5.3 contains additional features that must be added to the Ruby program to illustrate features specific to Ruby.

5.1 Common functionality

Write a Course class. A Course object is composed of an array of Student objects. A Student has a student name (represented by a first and last name), a unique student number, a number of tests written, and a list of marks. Each Student should provide the following behaviour:

• addTestScore(score) which adds score to the list of marks and increments the number of tests written by one.
• removeTestScore(score) which removes score from the list of marks and decrements the number of tests written by one.
• getCurrentMark() which should be empty (it will be overridden in subclasses of Student)
• setName(studentName) which sets the student’s name to studentName.
• getName() which returns the student’s name.

Additionally, the Course should provide the following behaviour:

• addStudent(student) which adds the student object to the course.
• removeStudent(studentNumber) which removes the student with the given student number from the course.
• addMark(studentNumber, mark) which adds a test score to the student indicated by studentNumber
• removeMark(studentNumber, mark) which removes the specified test score from the student indicated by studentNumber. If the score is not present, no score should be removed from the list of marks.
• getCurrentMark(studentNumber) which returns the current average mark of the student with the given student number.
• setName(studentNumber, name) which sets the account holder of the account with the given studentNumber to name.
• getName(studentNumber) which returns the account holder of the account with the given student number.

Write two new classes, each of which is a subclass of Student. These classes should be called PartTime and FullTime. In the case of a FullTime student, the two worst marks are ignored when calculating the average mark. In the case of a PartTime student, the two best marks are ignored when calculating the average mark. Override the getCurrentMark method to accomplish this.

5.2 Ada specifics

• Be sure that all courses and students are properly initialised. Ada does not implicitly call a constructor, so you will have to do this initialisation explicitly. Write and call constructors on your objects.

• Write the student class as a child package of Course. This has a similar effect to friend classes in C++, and will enable you to do without getters and setters in the account class. You must still ensure that unique IDs are maintained throughout all account objects.
5.3 Ruby specifics

- Ruby supplies shorthand for getters and setters, called accessor methods. Investigate this shorthand and use it to implement the get and set student functionality.

- Ruby allows operator overloading for its classes. Define the + operator for Student so that it adds a mark indicated by the right operand to the student’s list of test scores. For example, if myStudent is of type Student, it should be possible to replace the call addTestScore(50) with student + 50.

- Ruby’s class methods are first class objects which allows them to be dynamically added to and removed from class instances at run-time. By adding a method to a class instance which already has a method of that name, the method is overridden. A student, called Bud “Bare Knuckles” Pucket, made the lecturer of COS444 an offer he couldn’t refuse. Add Bud to the course after overriding his getCurrentMark method. This new method must return a random number between 55 and 65.

6 Marking

Both the implementation and the correct execution of the programs will be taken into account. Marking will take place offline (i.e. not in the practical sessions). Therefore, please make sure that you include adequate comments in each of your submissions, indicating how the program is to be executed, and any deviations from the above specifications. Also ensure that you upload your implementations into the correct assignment upload slot. The marks for this practical lab experience will be allocated as follows:

<table>
<thead>
<tr>
<th>Category</th>
<th>Mark Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ada</td>
<td>5 marks</td>
</tr>
<tr>
<td>Ruby</td>
<td>5 marks</td>
</tr>
</tbody>
</table>

References