EECS 2030 Winter 2020: Lab 4

Lassonde School of Engineering, York University

INTRODUCTION TO LAB-4

Students in the same lab section are allowed to work in groups of 2 or 3 students. This lab will be graded for style and correctness. This lab is due on **29th Feb 2020, 11pm**.

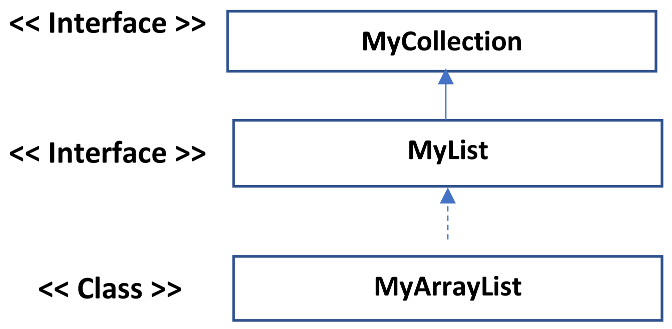
STYLE RULES FOR CODING

The style rules are not overly restrictive in EECS2030.
1) Your programs should use the normal Java conventions (class names begin with an up-percase letter, variable names begin with a lowercase letter, public static final constants should be in all caps, etc.).
2) In general, use short but descriptive variable names. There are exceptions to this rule; for example, traditional loop variables are often called i, j, k, etc. Avoid very long names; they are hard to read, take up too much screen space, and are easy to mistype.
3) Use a consistent indentation size. Beware of the TAB vs SPACE problem: Tabs have no fixed size; one editor might interpret a tab to be 4 spaces and another might use 8 spaces. If you mix tabs and spaces, you will have indenting errors when your code is viewed in different editors.
4) Use a consistent and aligned brace style.
5) Insert a space around operators (except the period ".")
6) Avoid using "magic numbers". A magic number is a number that appears in a program in place of a named constant.
7) A good IDE (integrated development environment) such as eclipse will correct many style errors for you. In eclipse, you can select the code that you want to format, right click to bring up a context menu, and choose Source → Format to automatically format code.

LAB EXERCISE 1. Collections, generics, interfaces, implementing interfaces

In this exercise you're going to implement a simplified version of **ArrayList**. The purpose of this exercise is to help you better understand the Java Collection framework, especially the List interface, and the behavior of **ArrayList**. You will also practice implementing generic classes.

In package **myUtil**, there is an (generic) interface named **MyCollection**, which mimics the **Collection** interface in Java Collection framework, defining some basic operations for a collection, e.g, add or remove an element from a container. There is also a (generic) sub-interface **MyList** which mimics the **List** interface, extending **MyCollection** interface and declaring some index-related operations on the list, e.g., add or remove an element at a specified index. You are also provided with a (generic) concrete class **MyArrayList** which mimics the **ArrayList** class, implementing **MyList** interface. The APIs of the interfaces are provided.



Implementation:

- Complete the implementation of the **MyArrayList** class. Note that similar to the **ArrayList** class, this class maintains an array as its internal storage, which has a default size 10. Once the array is full, it will be expanded by the default size 10.
- Obviously, you should not use the **ArrayList**, **Set** or **Map** in your implementation. The only data structure used is an array.

LAB EXERCISE 2. Collections, static fields, immutability, class invariant, aggregation and composition

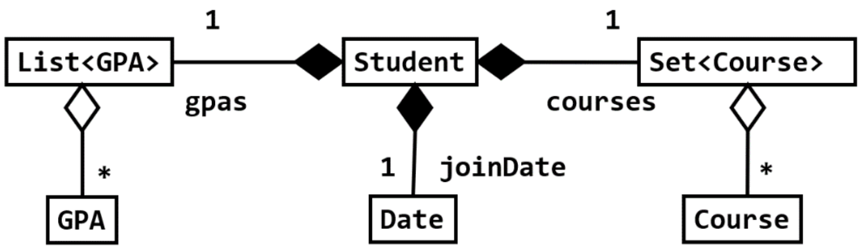
Complete the implementations of a student record system in a package **studentSystem**. The system maintains a **Student** class. A student has a name, a YorkID, a year level, a join date, a collection of courses taking, and a collection of GPAs for each year of study. That is, the **Student** class has the following instance attributes: String *name*, String *YorkID*, int *year*, **Date** object *joinDate*, a collection of **Course** objects and a collection of **GPA** objects.

- Each student has a unique YorkID in the format of *York-XX*, where XX is the serial number, starting from 01. That is, the first student object gets York ID *York-01*, the second object gets *York-02*. Assume no more than 99 instances can be created.
- The student **owns** the its join date information, i.e., the student and its *joinDate* field form a **composition** relation where the Student object has exclusive access to the Date

object, not sharing the reference with clients. This ensures no “privacy leak” (i.e., clients would not accidentally modify a student’s join date). The join date can be changed using mutator only.

- The student **owns** its collection of courses, but does not own the courses in the collection. That is, the Student has exclusive access to its collection of Course, not sharing the collection reference with clients (i.e., student and its collection form a **composition**), but the Course objects stored in the collection are shared. E.g., if a course’s instructor is changed outside the Student class, all the students taking that course will see the change. The collection can be modified using mutator only.
- The student **owns** its collection of GPAs, and also **owns** the GPA objects stored in the collection. That is, the Student class has exclusive access to the collection of GPAs as well as the GPA objects stored in the collection. The student does not share the reference of the collection, and also does not share the reference of GPA objects stored in the collection. This prevents “privacy leak” (i.e., clients would not accidentally modify a student’s GPA). The collection can be modified using mutator only.

The relations are shown in the UML diagram below. (The diagram does not show *name*, *YorkID* and *year,* as they are either primitive or immutable reference type).



Implementation:

- Create classes **Date, Course**, and **GPA,** following the provided APIs.
  The **Date** class has a class invariant that the year is non-negative, and the month is 1~12.
- Complete the default constructor of class **Student**.
  Modify the customer constructor, getters and setters when necessary.
  Hint: you need to consider when a simple reference aliasing is enough, when you should use a shallow copy, and when a deep copy is needed.

SUBMIT INSTRUCTIONS FOR STUDENTS NOT WORKING IN A GROUP

Submit your solutions using the submit command. Refer to lab0 (manual) webpage for instructions. Remember that you first need to find your workspace directory, then you need to find your project directory. In your project directory, your files will be located under different package directories.

- Your **MyArrayList.java** should be in the directory **EECS2030LAB4W20_stu/src/myUtil**. Once you are in the directory, issue **submit 2030 lab4 myArrayList.java**

- Your **Student.java** and others should be in the directory **EECS2030LAB4W20_stu/src/studentSystem**. Once you are in the directory, submit the files individually, or, submit together by issuing
  submit 2030 lab4 Student.java Date.java Course.java GPA.java

Alternatively, you can submit using the department's web submission system.

SUBMIT INSTRUCTIONS FOR STUDENTS WORKING IN A GROUP

If you are working in a group, in addition to submitting program files as shown above, one of the group members also need to submit a text file containing EECS login names of your group members. Create a plain text file named **group.txt**. You can do this in eclipse using the menu File → New → File, or, use any other text editors. Type your login names into the file with each login name on its own line. For example, if the students with EECS login names rey, finn, and dameronp, worked in a group the contents of **group.txt** would be:
rey
finn
dameronp

Submit your group information using submit command. In the directory where the text file is saved, issue **submit 2030 lab4 group.txt**

SUBMIT INSTRUCTIONS FOR STUDENTS OUTSIDE PRISM LAB

It is possible to submit work from outside the Prism lab, but the process is not trivial; do not attempt to do so at the last minute if the process is new to you. The process for submitting from outside of the Prism lab involves the following steps:
1) transfer the files from your computer to the undergraduate EECS server red.eecs.yorku.ca

2) remotely log in to your EECS account
3) submit your newly transferred files in your remote login session using the instructions for submitting from within the lab
4) repeat Steps 1 and 3 as required

Windows users will likely need to install additional software first. Mac users have all of the required software as part of MacOS.
Alternatively, you can submit using the department's web submission system.