1

**Summer 2016**

**BTP200 Final Project (v. 4)**

**Seneca Course Management Tool**

**(SCM)**

You are asked to develop a C++ application that manages Seneca courses taken by the students at the School of Information and Communications Technology (ICT). In this proof of concept application, the ICT students can take two types of courses, namely ICT-related courses (e.g. OOP244, BTP200) and general education courses (e.g. EAC150, BTC140). The application will provide a console-based menu system to manage these courses.

You will develop this application *incrementally* as new user requirements emerge during the project development process. As you work on the project, you will build up your conceptual and practical knowledge of object-oriented programming in C++. You will learn to apply three principles of object-oriented programming (i.e. encapsulation, inheritance and polymorphism). In particular, you will create five classes as new user requirements emerge. The application class will be the point of integrating four classes.

**A. THE USER INTERFACE**

Seneca Course Management Tool

1- List courses.

2- Display the details of a course.

3- Add a course.

4- Change the study load of a course.

5- Load courses from a file.

6- Save courses to a file.

0- Exit program.

> \_

**B. CLASSES TO BE DEVELOPED**

You will develop the following five classes for this application:

**Streamable**

This interface (a class with “only” pure virtual functions) enforces that the classes inherited from it are to be “*streamable”*. In the first half of the course we define streams to be sequence of characters. IO streams allow program data to be exchanged with peripherals such as computer consoles or keyboards. Any class derived from “Streamable” can read from or write to streams.

Using this class, a list of courses can be saved to and retrieved from a file.

Individual course details can be displayed on screen or read from keyboard.

**Course**

A class that encapsulates information about courses.

**As** the project develops, this class will acquire *streamable* characteristics as

new user requirements emerge.

**ICTCourse**

A class that encapsulates information about ICT-related courses (e.g. OOP244,

BTP200). It is derived from the **Course** class.

**GenEdCourse**

A class that encapsulates information about general education courses (e.g. EAC150, BTC140). It is derived the **Course** class.

**ScmApp**

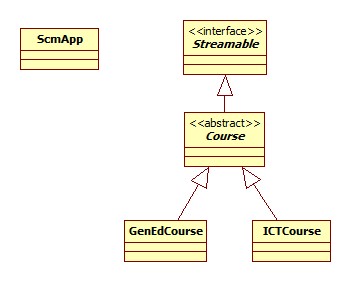
An application class that provides a console-based menu system to manage

The courses. It will use the other four classes in order to provide the system’s

functionality. You will develop this application *incrementally* as new user

requirements emerge.

**C. CLASS DIAGRAM**



**D. PROJECT DEVELOPMENT PROCESS**

You have **four weeks** to complete the project. The project is divided into 4 milestones and thus four deliverables. Each milestone has a clear learning focus and specific learning tasks. The approximate schedule for the deliverables is as follows:

• **Final Project Due date: Sunday August 7, 2016 by 11:59:00pm.**

• MS1: The Course and ScmApp classes. Due: July 20, 2016 by 11:59:00pm

• MS2: The ICTCourse, GenEdCourse and Due: July 27, 2016 by 11:59:00pm

ScmApp classes.

• MS3: The Streamable , Course, ITCourse Due: **August 7, 2016 by 11:59:00pm**

and GenEdCourse classes.

• MS4: Completion of the ScmApp class. Due: **August 7, 2016 by 11:59:00pm**

**E. FILE STRUCTURE OF THE PROJECT**

Each class will have its own header file and .cpp file. The names of these files should be the same as the class names.

Example: Class **Course** has two files: **Course.h** and **Course.cpp**

In addition to header files for each class, create a header file called **“general.h”** that will hold the general defined values for the project, such as:

MAX\_COURSECODE\_LEN (6) The maximum length of a course code.

DISPLAY\_LINES (10) The maximum number of lines used to display course details

before each pause.

MAX\_NO\_RECS (2000) The maximum number of records in the data file.

This header file should get included where these values are used.

**Notes**

1. All the code developed for this application should be in the ***sict***namespace.

2. All the code must be properly documented and indented. Ask your instructor for

details.

**MILESTONE 4: COMPLETION OF THE APPLICATION CLASSS.**

In this last milestone, you are going to implement two new functionalities of the application class (ScmApp). The application class will allow a user to save courses to a file and read courses from a file. In brief, you are going to implement selections #5 and #6 on the following menu system.

Seneca Course Management Tool

1- List courses.

2- Display the details of a course.

3- Add a course.

4- Change the study load of a course.

5- Load courses from a file.

6- Save courses to a file.

0- Exit program.

> \_

**A. User Requirements**

**The ScmApp Class.**

**1. Private Member Variables.**

**char filename\_ [256];**

It holds the name of the text file that stores information about the courses.

**fstream datafile\_;**

An fstream object used to create and access a file.

**2. The Constructor.**

It receives a const character string as an argument (**filename**) and does the following

initialization:

a) It copies the argument filename to the member variable **filename\_.**

b) It initializes other member variables as specified in Milestone 1.

**3. Private Member Functions.**

Note: You are free either to use your own logic or follow the pseudo-code or improve the pseudo-code.

**void loadRecs();**

It opens the file for reading the courses, If the file does not exist, it will create an empty

file and exit. Otherwise it loads the records (i.e. objects) from the file to initialize new

objects. These new objects will be pointed by the member variable **courseList\_**.

**Note: ~~In order to make sure that there is no memory leak~~ *~~before~~* ~~loading the records from the file, the function must release the memory storage that has been allocated for~~ *~~all~~* ~~the objects that are currently pointed by the member variable courseList\_.~~**

Finally, it closes the file

**Here is the pseudo-code:**

Set **readIndex** to zero.

Open the file for reading (use ios::in).

If the file is in fail state (i.e. the files does not exist on the disk),

then

clear the failure,

close the file.

Otherwise

Deallocate the memory for all objects pointed by **courseList\_.**

until reading fails

loop

read one char character into a variable (**objectType**) to identify **the**

**type of a course**.

If the value of **objectype** is ‘G’

Dynamically create a **GenEdCourse** object and store the object address at

**courseList\_[*readindex*]**

If the value of **objectype** is ‘I’

Dynamically create an **ICTCourse** object and store the object address at

**courseList\_[*readindex*]**

If neither ‘G’ nor ‘I’ is read

skip the comma in the file

load the record (i.e. the course object) from the file, using the load()

function

add one to read index continue the loop

Set the number of courses to **readIndex**.

Close the file.

**void saveRecs();**

- It opens the file for writing records (i.e. course objects).

- It loops through the array courseList\_ up to noOfCourses\_ and stores the

objects in the file.

- It closes the file.

**B. TESTING AND SUBMISSION REQUIREMENTS (MILESTONE 4).**

1. If not on matrix already, upload **general.h**, **Streamable.h, Course.h, ScmApp.h,**

**ICTCourse.h, GenEdCourse.h**, **Coures.cpp, ICTCourse.cpp, GenEdCourse.cpp,**

**ScmApp.cpp** and the tester to your matrix account. Compile and run your code and

make sure everything works properly.

Then run the following script from your account:

**~eden.burton/submit ms4 <ENTER> OR**

**~bradly.hoover/submit ms4 <ENTER> OR**

**~peter.liu/submit ms4 <ENTER>**

and follow the instructions.

2. Upload all your source files (zipped) to Blackboard. Do **manual testing** at the

computer lab.

**THE END OF MILESTONE 4**

**CONGRATULAIONS! YOU HAVE COMPLETED THE FINAL PROJECT**