Virginia Tech
Bradley Department of Electrical and Computer Engineering
ECE-3574: Applied Software Design Fall 2014

Homework 2

Submission Details

You must submit the solutions for this homework as an electronic submissions using Scholar (under ECE3574 → Assignments → Homework 2). The submission must be a gzipped tar file (.tar.gz) with your source code. Include all necessary project files, but no binary or compiled files. Your program will be run to evaluate its correctness, and the source code will be reviewed for adherence to the Qt programming style. Your program must run on Ubuntu 14.04.1 and compile/build using the GNU C/C++ compiler and the qmake/make tools. The following information must be included at the top of each of your source files as comments: your full name, your student ID number, your email address, class (ECE 3574), and the title of the assignment (Homework 1). The submitted file must be given a name in the following form: ***LAST_FIRST_hw2.tar.gz*** where LAST is your last or family name and FIRST is your first or given name. You are only allowed to make one submission. Paper, email or Drop Box submissions will not be accepted. All work must be submitted by the announced due date/time. **Late submissions will not be accepted!** (Don't do it! You have been warned!)

Questions

Use the Homework 2 forum in the Discussion Board area of the class web site to ask questions about this assignment. Do not post questions that contain specific information about the solution.

Honor Code

As stated in the syllabus, in working on homework and projects, discussion and cooperative learning are allowed. However, copying or otherwise using another person's detailed solutions to assigned problems is an honor code violation. See syllabus for details.

Learning Objectives:

The primary learning objectives of this homework include learning how to program using inheritance, polymorphism for the purposes of generic programming and use of Qt container classes. Additional objectives include learning to modify and integrate new code into legacy applications and experimenting with varying levels of encapsulation.

Exercise:

Required section

(R1) Do Exercise #2a, #2b, and #2c ("6.10.1 Exercises: Containers of Pointers") in Chapter 6 of Ezust (Page 225-226). Instead of using the UML diagram in the book use Figure 1 at the end of this document.

(R2) Do Exercise #3 ("6.10.1 Exercises: Containers of Pointers") in Chapter 6 of Ezust (Page 226-227)

(R3) Do Exercise #4 ("6.10.1 Exercises: Containers of Pointers") in Chapter 6 of Ezust (Page 228)

Optional section

(O1) Do Exercise #5 ("6.10.1 Exercises: Containers of Pointers") in Chapter 6 of Ezust (Page 228)

Grading

The grading for this assignment would be done in two parts:

- 90% for your application's compliance to the homework spec. It should be able to compile without errors and do all the required functions.
- 10% for your source code's compliance to the QT Style guidelines and naming conventions which are given in section 3.1 in the book.

Additional Information

1. Problem R1
   a) exercise #2c requires you to write a test client to test all the functions that you implement in #2a and #2b. Create a console based menu which asks the user to do the following
      i. Add a film, internally calls addFilm()
      ii. Remove a film, internally calls removeFilm()
      iii. Get films' IDs based on a title, internally calls getID(). Print IDs for all records with the matching title.
      iv. Search for a film, internally calls findFilm(). This should print all the details of the film with the matching ID.
      v. Exit the application
   b) Do standard error checking. Wherever a user performs an illegal action a message should be printed noting why it was not valid.
   c) The same film should not be added twice. A record may only match another record if it is the exact same type (so a Film object and an Educational object will not interfere with the other being added). You should use dynamic_cast and typeid for this purpose.

d) The input method is an interactive input method at the terminal that prompts the user for information and allows the user to exercise all the functionality of the code.

e) You may define Grade, FilmTypes and MPAARatings however you like but you are required to use at least the suggested labels in #2a.

f) The method functions defined in the UML are required but you may add other method functions. However, you may not add or omit member variables to their respective classes. You may only create one instance of FilmList.

2. Problem R2 is based on the Simple Library example discussed in Section 6.10 (pages 212-224). The complete source code of this Simple Library is available at:

   https://www.ics.com/designpatterns/dist/src.tar.gz.
   Look under src/pointer-container. There are 4 files of interest:
   - library.cpp and library.h contains the complete definition of the library "as is"
   - libraryClient.cpp (a portion of which is given in example 6.42; page 224)
   - libraryClient-v2.cpp -- this is the driver code (given in example 6.43; page 227) that you will be testing your solution for problem R2.
   - The interactive interface for exercises 3, 4 and 5 is used in libraryClient-v2.cpp and defined (with minor modification required) in libraryClient.cpp.

3. When submitting your code separate it into multiple parts that can compile individually. The code for problem R1 should be in a folder in your main archive named “R1” and so on.

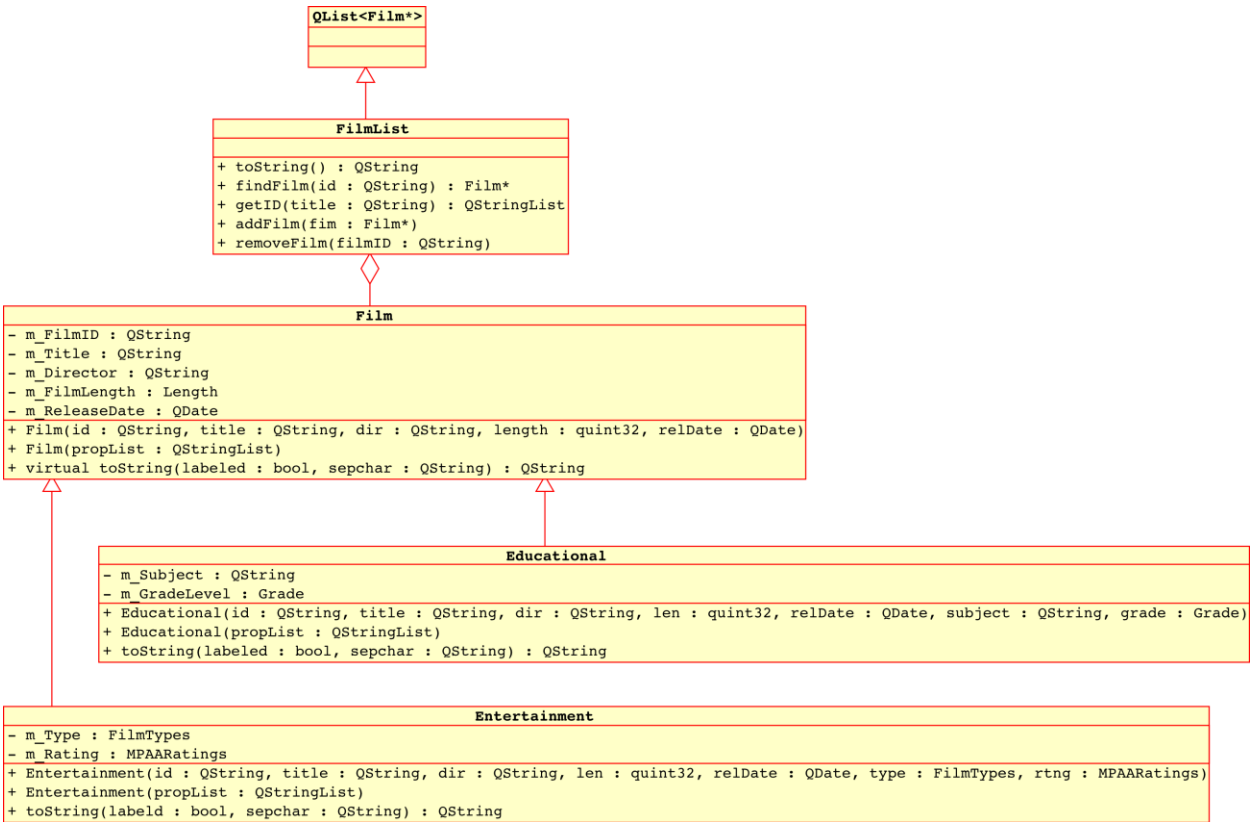

Figure 1 – UML diagram for R1