COMP 2404 -- Assignment #5

Due: Friday, April 4, 2014 at 11:00 AM

Goal

You will modify your movie database program to a single process version, and evolve it to polymorphically support the management of multiple forms of media (movies, books, etc). You will also upgrade your list class to be templated.

Learning Objectives

- design and implement classes that encapsulate object creation and behaviour, using polymorphism
- practice implementing a templated class

Instructions:

1. Templating the `List` class

You will change the **`List`** class from Assignment #4, including all overloaded operators, to make it a templated class that can store any type of data. You will modify all your code to use the new templated **`List`** for all collections, including the permanent collection in storage.

2. Management of media objects

a. Scaling back to single process

You will be working from a single process implementation of your movie database program, just as you did before Assignment #4. It will be easier if you begin with your Assignment #3 and remove some classes. Your scaled-back program will have the following characteristics:

- There will be no **`Server`** façade or **`Serializer`** classes. Your control object will have the **`Storage`** object as a data member.
- Your **`Storage`** class will not have the **`handleRequest`** operation from Assignment #3. Instead, it will provide the **`update`** and **`retrieve`** functions as public member functions, with the same implementation as in Assignment #3.
- Your program will still use the **`List`** class for storing media object pointers in the permanent collection.

b. Implementing the media hierarchy

In this assignment, before the user sees the main menu, he/she will be prompted to select the type of media to be managed (movies or books) for the entire run of the program. From that point, your program will be working with **`Media`** objects and pointers, although these will all be of the same derived class (either all books or all movies, depending on the user selection). **Nowhere** in your program are you going to check the type of object being managed! Once the user selects the type of object, all processing must be done **polymorphically**. This will require the use of an interface class to gather user input, and a factory class to create **`Media`** derived objects.

You will:

- create an abstract class called **`Media`**, with title and year as data members
- change your **`Movie`** class to be derived from **`Media`**
- create a concrete class called **`Book`**, which derives from **`Media`**; this new class will have as data members an author and ISBN number, both strings

c. Update the control, UI and storage classes to manage media objects

All classes in your program must be changed to manage **Media** objects, rather than just movies. You will:

- change your control and UI classes to allow the user to select the type of media to be managed
- change all classes in the program to support **Media** objects instead of movies, including menu options and member function names; you can remove the print movies by genre menu option and associated functions
- modify the **Storage** class to store a permanent collection of **Media** pointers

3. Encapsulating data input behaviour

Once the user has selected the type of media to be managed, you need to initialize the interface object that will **polymorphically** prompt the user to enter the correct data. For example, for both movies and books, the user has to enter a title and year. However, for movies, the user must be prompted to enter a genre, and for books, he/she must enter an author name and ISBN number.

You will implement an abstract interface class called **InputBehaviour**, which will define only one function with the following prototype:

```
void getMediaData(vector<void*>& values)
```

This function will prompt the user for piece of data required, and add each one to the STL **vector** output parameter. For example, if the program is managing movies, the **values** vector will contain three elements: the title, year and genre, all represented as **void***. If the program is managing books, the **values** vector will contain four elements: the title, year, author and ISBN number. You will implement a concrete derived class for every type of **Media** to be managed, each with a concrete implementation of the **getMediaData** function.

4. Encapsulating media object creation

You will also need to initialize the factory object that will **polymorphically** create the correct **Media** derived objects (movies or books). You will implement an abstract factory class called **MediaFactory**, which will define only one function with the following prototype:

```
void createData(vector<void*>& values, Media** newMedia)
```

This function will dynamically allocate a new **Movie** or **Book** object, returned in the output parameter **newMedia**. The new object will have values for the title, year and additional data members as found in the elements of the **values** vector input parameter (the first element of the vector will be the title, the second the year, etc). You will implement a concrete derived class for every type of **Media** to be managed, each with a concrete implementation of the **createData** function.

Constraints

- your program must be written in C++, and **not** in C
- do not use any functions from the C standard library (e.g. printf, scanf, fgets, sscanf, malloc, free, string functions)
- do not use any classes or containers from the C++ standard template library (STL), unless otherwise specified
- do **not** use any global variables or any global functions other than **main**
- do **not** use structs, use classes instead
- objects should always be passed by reference, not by value

Submission

You will submit in *cuLearn*, before the due date and time, the following:

- a *detailed* UML class diagram (as a PDF file) that corresponds to your program design
- one `tar` file that includes:
  - all source and header files for your movie database program
  - a Makefile
  - a readme file that includes:
    - a preamble (program author, purpose, list of source/header/data files)
    - compilation, launching and operating instructions

Grading

- **Marking breakdown:**

Component	Marks
UML class diagram	10
Templated List class	20
Media hierarchy	10
Input behaviour class	30
Media factory class	30

- **Deductions:**
  - Up to 50 marks for any of the following:
    - the code does not compile using g++ in the VM provided for the course
    - the code cannot be tested
    - the design does not generally follow correct design principles (e.g. data abstraction)
    - prohibited library classes or functions are used
  - Up to 20 marks for any of the following:
    - the Makefile or readme file is missing
    - global variables, global functions, or structs are used
    - objects are passed by value
    - the code is not correctly separated into header and source files
  - Up to 10 marks for missing comments or other bad style (non-standard indentation, etc.)

- **Bonus marks:**
  - Up to 5 extra marks are available for fun and creative additional features