**Home Work 5**

 (This lab was originally designed by Professor Paul Thor)

**Objective -** The objective of this lab assignment is to write and test a complete OOP. This program will implement a class inheritance hierarchy and **virtual** functions. It will give you practice in combining two important aspects of OOP – namely, **inheritance** and **polymorphism** – in a single program.

**Specifications -** This assignment will require you to use the C++ language to implement a simulation problem -- an area that OOP is especially adept at handling. Specifically, you are to simulate an Entertainment Media Collection that contains four different kinds of media: 1) AudioCassette, 2) AudioCD, 3) VideoVHS and, 4) VideoDVD. The user of the program will first give the media collection a name, and then will be able to, 1) Add a new media item to the collection, 2) Delete a media item from the collection, 3) Play a media item, 4) Fast-forward a media item to its end, 5) Rewind a media item to its beginning, 6) List the detailed information of a selected media item in the collection, 7) List all media titles, 8) Calculate the total investment in the media collection, 9) Calculate total playing time of the collection and 10) Terminate the simulation.The simulation continues until the user terminates it.

***This program will be interactive and menu driven.***

Your menu should be displayed as shown below:

Welcome to <the interactively-entered media collection name> Media Collection Menu

Select from the following choices:

1. Add Media Item to Collection
2. Delete Media Item from Collection
3. Play a Selection
4. Fast Forward a Selection
5. Rewind a Selection
6. List a Specific Media’s Details
7. List All Media Titles
8. Calculate Media Collection Investment
9. Calculate Total Playing Time of Collection
10. Terminate the Simulation

Enter your Selection:

**Preliminary Design:**

The diagram below shows the class relationships for the classes declared on the previous pages.

**Preliminary Design Diagram (a UML-like class model of the solution)**

The class declarations that follow are intended to give you a starting point for your solution. Consider them as a ***draft* *design*** given to you by an experienced software engineer. As previously mentioned, you may deviate from this draft design as you see fit as long as you do not deviate from the main intent of this exercise – namely, the use of both inheritance and polymorphism (via **virtual** functions).

**Legend:**

Contains

Inheritance

 *Italics* Abstract Base Class

Time

AudioCD

AudioCassette

VideoDVD

VideoVHS

*Media*

MediaCollection

// A C++ version of the polymorphic entertainment Media Collection program originally by Dr Paul Thor.

// Updated by Dr Bob Johnson, 2003, 2004, 2005, 2008, 2014 and 2015.

**class** Time

{

**friend** ostream & operator<<(ostream & Out, Time & RHS);

**public**:

 Time(**int** InMinutes = 0);

 **void** SetTime( **int** NewMinutes);

 **int** GetTime() **const**;

 **private**:

 **int** Minutes;

};

**+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++**

**A Media Abstract Base Class**

**+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++**

**class** Media // An Abstract Base Class

{

 **public**:

 Media();

 Media(**const** string &InTitle, **const** string & InArtist, **const** string & InWherePurchased,

 **Float** InCost, **const** Time & InRunningTime);

 string GetTitle() **const**;

 string GetArtist() **const**;

 string GetWherePurchased() **const**;

 **float** GetCost() **const**;

 Time GetRunningTime() **const**;

 **bool** IsMediaAtBeginning() **const**;

 **bool** IsMediaAtEnd() **const**;

 **virtual** ~Media() { } // virtual destructor so polymorphic-destruction works properly

 **virtual void** PrintMedia(ostream **&** Out) **const**; // Prints common data members to output stream

 **virtual void** Play() = 0; // Can only play a media item if it is NOT at its end **virtual void** FastForward() = 0; // Can only fast-forward a media item if it is NOT at its end

 **virtual void** Rewind() = 0; // Can only rewind a media item if it is NOT at its beginning

 **protected**:

 string Title;

 string Artist;

 string WherePurchased;

 **float** Cost;

 **bool** MediaAtBeginning;

 **bool** MediaAtEnd;

 Time RunningTime;

};

**++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++**

**Now the Derived Classes**

**++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++**

**class** AudioCassette : **public** Media

{

**public**:

 AudioCassette();

 AudioCassette(// all the needed parameters);

 **int** GetNumberOfSelections() **const**;

 **int** GetCurrentSelection() **const**;

 **virtual void** Play();

 **virtual void** FastForward();

 **virtual void** Rewind();

 **virtual void** PrintMedia(ostream **&** Out) **const**;

 // Displays media type, calls parent to display inherited members, then

 // displays NumberOfSelections

**protected**:

 **int** NumberOfSelections;

 **int** CurrentSelection:

};

**class** AudioCD: **public** Media

{

**public**:

AudioCD();

AudioCD(// all the needed parameters);

**int** GetNumberOfTracks() **const**;

**int** GetCurrentTrack() **const**;

 **virtual void** Play();

 **virtual void** FastForward();

 **virtual void** Rewind();

 **virtual void** PrintMedia(ostream **&** Out) **const**;

 // Displays media type, calls parent to display inherited members, then

 // displays NumberOfTracks

**protected:**

 **int** NumberOfTracks;

 **int** CurrentTrack;

};

**class** VideoVHS: **public** Media

{

**public**:

 VideoVHS();

 VideoVHS(// all the needed parameters);

**int** GetNumberOfTracks() **const**;

**int** GetCurrentTrack() **const**;

 **virtual void** Play();

 **virtual void** FastForward();

 **virtual void** Rewind();

 **virtual void** PrintMedia(ostream **&** Out) **const**;

 // Displays media type, calls parent to display inherited members, then

 // displays NumberOfTracks

**protected**:

**int** NumberOfTracks;

 **int** CurrentTrack;

};

**class** VideoDVD: **public** Media

{

**public**:

 VideoDVD();

 VideoDVD(// all the needed parameters);

**int** GetNumberOfScenes() **const**;

**int** GetCurrentScene() **const**;

**virtual void** Play();

 **virtual void** FastForward();

 **virtual void** Rewind();

 **virtual void** PrintMedia(ostream **&** Out) **const**;

 // Displays media type, calls parent to display inherited members, then

 // displays NumberOfScenes

**protected:**

**int** NumberOfScenes;

**int** CurrentScene;

};

**const int** MEDIA\_COLLECTION\_SIZE = 5;

**enum** MediaType {AUDIO\_CASSETTE = 1, AUDIO\_CD, VIDEO\_VHS, VIDEO\_DVD};

**class** MediaCollection

{

**public**:

 MediaCollection();

 //~MediaCollection(); // not needed if we use container of “smart” pointers

 **void** AddMedia();

 **void** ListMediaTitles() **const**;

 **void** DeleteMedia();

 **void** Play();

 **void** FastForward();

 **void** Rewind();

**void** PrintMediaDetails() **const**;

 **float** CalculateMediaCollectionInvestment() **const**;

 Time CalculateTotalCollectionPlayingTime() **const**;

 **string** GetCollectionName() **const**;

 **int** GetCurrentMCSize() **const**;

**protected**:

 **void** DisplayMediaTypeMenu() **const**;

 MediaType GetMediaTypeSelection() **const**;

**private**:

 **string** CollectionName;

 array<unique\_ptr<Media>**,** MEDIA\_COLLECTION\_SIZE > TheCollection;

 **int** CurrentMCSize;

};

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

The above class declarations are available via e-mail or on disk - check with your instructor if you desire a copy. **Remember: this class design is merely a suggested beginning point.** If you can justify additions (NO REMOVALS PLEASE) to these class abstractions to your instructor feel free to do so. Also remember that you must correctly implement AND TEST any enhancements you make to these classes.

**Discussion --** As the class design diagram shows, this exercise involves both an inheritance (Is-A) relationship (i.e., between the Media Abstract Base Class and its derived classes) as well as a containment (Has-A) relationship. Containment means that object(s) of the class are inside or are part of an outer object. The above may be read as: A MediaCollection object contains (i.e., “has”) pointers to Media objects.

According to the design above, to add a media item to a media collection, a pointer to a dynamically-allocated media of the appropriate class is added to the array of media pointers in the MediaCollection. For example, if the user of the simulation chooses menu item, 1 (Add Media Item to TheCollection), then the, AddMedia(), member function, 1) calls the protected member function, DisplayMediaTypeMenu() which displays a menu of the four media types, then 2) calls protected member function, GetMediaTypeSelection(), which returns the enumeration type value for the chosen media type and finally, 3) dynamically allocates a media object of the chosen type and places the pointer to this object in the MediaCollection array data member, TheCollection. If, for instance, the user chose to add a new Audio Cassette it would be allocated and added to the media collection. Shown below is a partial implementation of the AddMedia() member function:

 **. . .**

 MediaType Selection; // 🡨 NOTE the use of the enumerated type in this declaration

 **if**(CurrentMCSize < MEDIA\_COLLECTION\_SIZE)

 {

 DisplayMediaTypeMenu(); // Calls to protected-access support member

 Selection = GetMediaTypeSelection(); // functions of MediaCollection class

 **switch**(Selection)

{

 **case** AUDIO\_CASSETTE: TheCollection[CurrentMCSize] =

 unique\_ptr<Media>(**new** AudioCassette);

 **break**;

 **case** AUDIO\_CD: TheCollection[CurrentMCSize] =

 unique\_ptr<Media>(**new** AudioCD);

 **break**;

 // similar **case** selections for the other media types

} // end of switch statement

 CurrentMCSize++;

 }

 **else**

 cout << “Media collection is full – cannot add another media item at this time\n”;

 **. . .**

You may style the output in any manner you desire as long as the desired information is shown to the user. **Be careful that displayed information doesn’t scroll off your screen before the user can read all of it. To make these displays user-friendly you may have to place “pauses” in your displayed output (e.g., create a Pause() function and call it where needed).**

**Debugging Tips:** Additional interim information may also be displayed at the programmer’s discretion to assist in debugging your program. These are often called strategic output statements and are usually in the form of, “I am here” messages. These messages should not, however, remain in the final, handed-in, version of your solution (either comment them out or remove them altogether).

**Test driver** – Inside a main() test driver, declare a MediaCollection object and then test each of the menu items and exercise the program long enough to completely test all of its capabilities. You may assume that the user does not make entry mistakes (i.e., follows any instructions you provide). Error checking is encouraged to the extent you have time.

NOTE: To reduce the size of the main() driver, you may also want to implement a couple of global/free support functions, DisplayMainMenu() and GetMenuSelection(). DisplayMainMenu()’s job would be to display the menu of options shown on page 31. GetMenuSelection()’s job would be to output the prompt, “Enter your Selection:”, also shown on page 31, loop until the user inputs a valid selection number, and then return this selection. In **main()**, these could be used as shown below:

 // #includes of needed header (.h) files

 . . .

 **void** main()

{

 . . . // any other declarations you feel that you need

 MediaCollection MyMediaCollection;

 **int** MenuChoice = 0;

 **do**

 {

 cout << “Welcome to “ << MyMediaCollection**.**GetCollectionName()

 << “ Media Collection Menu” << endl;

 DisplayMainMenu(); // Displays menu shown on page 31

 MenuChoice = GetMenuSelection(); // Prompts for, inputs, validates and returns

 // user’s choice

 **switch** (MenuChoice)

 {

 **case** 1: MyMediaCollection**.**AddMedia();

 **break**;

 **case** 2: MyMediaCollection**.**DeleteMedia();

 **break**;

 **case 3**: MyMediaCollection.Play();

 **break;**

 // similar for cases 4 through 9

 **case** 10: cout << “Terminating simulation – thanks for playing!\n”;

 **break;**

 **default:** cout << “Invalid selection\n”;

 } // end of switch statement

 } **while**(MenuChoice != 10);

 **. . .**

 } // end of main()

**Development Methodology -** There are many ways to develop a program of this type. One method that starts small and builds on previous coding is to first code the menu and menu-selection functions, and ensure you can exit the simulation. Then take the menu selections one at a time, and completely develop all the member functions for each class necessary to support that menu selection (e.g., Add a new Audio Cassette). This incremental development will not only build on past development work, but also insures maximum partial credit. Your instructor may also discuss alternative development strategies. ***Remember to follow the recommendation -- “design-a-little, code-a-little, test-a-little, repeat until done” -- for this lab exercise!***

**Deliverable(s):**

A (**virus-free**) magnetic-media (e.g., Thumb Drive or DVD) with, **1)** all your properly documented source code, **2)** a README.TXT file that contains any comments, restrictions, limitations, etc., and **3)** an **executable** (e.g., Lab5.exe) version of the program. NOTE: Your executable must be able to run on an Intel-based PC hosting a Windows operating system! This executable file is very important since your instructor will not have the time to get your program compiled and linked on his/her system. **IF A README.TXT FILE IS NOT INCLUDED ON YOUR MAGNETIC MEDIA, YOUR LAB WILL BE RETURNED TO YOU UNGRADED!** If you do not complete all the specified functionality, submit what you have completed with the README.TXT file explaining what works and what does not, and any explanations about the program you wish to share with your instructor that could have a positive bearing on your grade.