PROG1385 – OBJECT-ORIENTED PROGRAMMING

A-02 : *THE DISNEYCHARACTER CLASS*

OVERVIEW

Believe it or not, you have all had the lessons necessary to begin writing your own classes! One of the scariest things is to do it for the first time … this exercise will ease you through it. In order to minimize the scary factor – you will be developing a class to model Disney Characters (Mickey, Minnie, etc.) – because hey – there's nothing scary about Mickey Mouse! (Right? ☺)

In this exercise, you will be

- Developing 3 source files to store in your repository
  - `DisneyCharacter.h` – the file that contains the class definition for this exercise
  - `DisneyCharacter.cpp` – the source file that contains the method bodies (the code) for the class
  - `testDisneyCharacter.cpp` – your *testharness* (main function) which you will use to test your class

The requirements description below give you details on the `DisneyCharacter` class in terms of the data-members of the class as well as the methods.

This is an individual assignment and must be completed by you alone

OBJECTIVES

This assignment supports and reinforces the following objectives:

- Understand and implement Object-Oriented language and design principles
- Use the C++ syntax to design and define classes and use them in assignments and projects

ACADEMIC INTEGRITY AND LATE PENALTIES

- Please refer to the SET Policies document regarding Academic Integrity Information
- Please refer to the SET Policies document regarding Late Policy

EVALUATION

- Please refer to the assignment weighting in the *Instructional Plan* for the course as well as the assignment's Rubric in the course shell.

PREPARATION

Review Module-04, Module-05 and Module-06 (lesson content, videos as well as the sample source code) for an understanding of how layout and code your first class definition.

REQUIREMENTS

This should be a simple one … I want you to write a class definition for a class to be known as **DisneyCharacter**. Here are the particulars of the class:

The Class Particulars

- The class has the following data members

  `name`
  - a *string*[1] of maximum 50 characters
  - if upon construction of the DisneyCharacter object or modification of the name data-member, the name is being set to a *string* longer than 50 character – you need to truncate the string at 46 characters and append the character sequence " …" to the end (that is a leading space before the "…"

  `creationDate`
  - a *string* of the format yyyy-mm-dd (example value "2012-01-30")
  - need to ensure that yyyy, mm and dd are numbers (does not matter what values) and also ensure that the dashes are in the correct location in the string

  `numMovies`
  - an integer representing the number of movies that the character has been in

  `whichPark`
  - a single character value indicating which of the DisneyWorld / DisneyLand parks the character can be found in – allowable values are:
    - 'M' for Magic Kingdom
    - 'S' for Disney Studios
    - 'A' for Animal Kingdom
    - 'E' for Epcot
    - 'C' for California Adventure
    - 'N' to indicate the character is not placed
- The class should have the following methods
  - a constructor which takes values for all 4 data-members and sets them

[1] You can choose to use a char[] for this data member or a string object … either way, you still need to be able to limit the maximum length to 50 characters …

        - need to ensure that the `name` *string* length is not exceeded – otherwise do as required
        - need to ensure that incoming value for `whichPark` data member is valid. If not, mark the character as *Not Placed*.
        - need to ensure that incoming value for `numMovies` is greater than or equal to zero. If not, set to zero.
        - need to ensure that incoming value for `creationDate` is in the required format. If not, leave blank
    - a constructor which takes only the **`name`** and **`creationDate`** data-members and defaults the **`numMovies`** data-member to a value of 0, and the **`whichPark`** data-member to 'N'
        - need to ensure that the `name` *string* length is not exceeded – otherwise do as required
        - need to ensure that incoming value for `creationDate` is in the required format. If not, leave blank
        - no other input validation is necessary in this constructor
    - You must create these 2 constructors – I want you to overload the constructors. Do not create 1 constructor with default values for the `numMovies` parameter and the `whichPark` parameter
    - You need to create a destructor
        - which prints out message to say "`<name data-member> destroyed.`" When invoked
            - e.g if the `DisneyCharacter` is "Tigger" then it states "`Tigger destroyed`"
    - accessors for *each* of the data-members
    - a mutator for each of the **`numMovies`** and **`whichPark`** data-members
        - your mutator's – like good mutators should perform validation on the incoming parameter value
        - and only set the data-member's value if the incoming value is valid
        - if the value is not valid – then the current value for the data-member is left alone
        - your mutator should also return a ***bool*** data-type to indicate whether the incoming value was valid or not
    - also declare the following public methods
        - `void ShowInfo(void)` – this method is called in order to print out the current value of all of the data members for the object
            - In presenting the information to the user … ***think like the user*** … what makes sense – what would be understandable to them?
            - This method would be similar to `void ShowDogInfo(void)` in Module-06's *The ClassicDog* example
        - `bool PlaceCharacter(char whichPark)` – this method is called in order to place or position the character at a particular park
        - `void SameMovies(DisneyCharacter& anotherCharacter)` – this method is called upon to set the number of movies that this Disney Character has been in to the same number of movies as the specified character (from the parameter list)
- Please follow and use *best practices* in your class design in terms of **encapsulation**
- Place only the class definition in the .H file and **place all of the method body code in the .CPP file**

- Commenting the source code is ***not required in this exercise.*** You can choose to comment the code (to practice) if you like, but the comments will not be marked.

The Test-Harness Particulars

- Your *testharness* source code module (i.e. your `main()` function) is called upon to use the `DisneyCharacter` class in the following manner. Please code only the following in your testharness:
    a. Instantiate a `DisneyCharacter` object called `mickey` (i.e. `name`="Mickey", `creationDate`="1929-01-01", `numMovies`=100, `whichPark`='M')
    b. Instantiate a `DisneyCharacter` object called `minnie` (i.e. `name`="Minnie", `creationDate`="1930-01-01")
    c. set the number of Minnie's movies equal to Mickey's by calling the `SameMovies()` method
        i. Question: Which object (Minnie or Mickey) is this method called on – and which object (Minnie or Mickey) is passed into it as a parameter?
    d. Show both Mickey's and Minnie's information by calling `ShowInfo()`
    e. Finally, place Minnie in the Epcot park

FILE NAMING REQUIREMENTS

You will create and submit **only the following files**:

- `testDisneyCharacter.cpp`
- `DisneyCharacter.cpp`
- `DisneyCharacter.h`

SUBMISSION REQUIREMENTS

Please ZIP up only the required files and submit to the appropriate eConestoga Dropbox by the deadline. Please give your ZIP submission the filename *lastName-firstInitial.zip*

- e.g. if you are Sally Jones – then your ZIP should be named `jones-s.zip`