Lab 02 - OOP Fundamentals Review

Direction: Submit typed work in the Labs directory of your github repository and/or as an attachment on Google classroom under the accurate Lab02 assessment. All submissions should have their appropiate extensions.

Part A: In class

A scrollbar allows users to scroll up and down a page whenever the content of the page is too long to fit in a single window view. The scrollbar is a resizable bar on a ranged track as follows

You can move the bar directly up and down the track, or move the bar by clicking on the arrow buttons on either end.

Your objective is to write a header file that contains the class *Scrollbar* which must contain:

- ☐ A public default constructor that initializes the fields of the scrollbar so that it represents a bar of a length of 5 at the very top of the track.
- ☐ A public copy constructor.
- ☐ A public assignment operator.
- ☐ A public empty destructor.
- ☐ A public int constant getter method named `GetLength()` that takes no parameters. It returns the length of the bar.
- ☐ A public void setter method named `SetLength()` that takes an int parameter. If the parameter is a positive number at most 50 ($0 < n \leq 50$) , it makes the length of the bar equal to the parameter and repositions the bar to the top of the track; otherwise, it will do nothing.
- ☐ A public int constant getter method named `GetPosition()` that takes no parameters. It returns the position of the lower end of the bar.
- ☐ A public void setter method named `SetPosition()` that takes an int parameter. If the parameter is a nonnegative number less than 50 ($0n < 50$) and the new position of the top end of the bar will not exceed 50 when the lower end of the bar is repositioned to the parameter, the lower end of the bar will become the parameter; otherwise, it will do nothing.
- ☐ A public bool method named `MoveUp()` that takes no parameters. If the top end of the bar will not exceed 50, it moves the bar up 1 unit, and then, returns true; otherwise, it only returns false.
- ☐ A public bool method named `MoveDown()` that takes no parameters. If the lower end of the bar will not be less than 0, it moves the bar down 1 unit, and then, returns true; otherwise, it only returns false.
- ☐ A public bool method named `Drag()` that takes an int parameter. If the parameter is between -50 and 50 exclusively ($-50 < n < 50$) and the ends of the bar will not pass the borders of the track when it is repositioned, it moves the bar parameter units, and then, returns true; otherwise, it only returns false.
- ☐ A public string constant method named `ToString()` that takes no parameters. It returns a string of the scrollbar vertically similar to the following:

```
  ******************
- |                | +
  ******************
```

- ☐ A friend overloaded ostream operator that returns the display exactly like the return of the `ToString()` method.

Note: the track is 50 units long, the difference between the ends of the bar is 1 less than the length, and you need to define fields

Part B: Take home

A deck of any type of cards can be loaded, drawn from, and shuffled. Your objective is to write three header files namely **Card.h**, **AbstractDeck.h** and **Deck.h**. In the **Card.h** file, define the class *Card* that contains

- ☐ A private string field named *suit*.
- ☐ A private string field named *symbol*.
- ☐ A public default constructor that assigns the empty string to both *suit* and *symbol* fields.
- ☐ A public copy constructor.
- ☐ A public assignment operator.
- ☐ A public empty destructor.
- ☐ A public string constant getter method named `GetSuit()` that takes no parameters. It returns the value of the *suit* field.
- ☐ A public string constant getter method named `GetSymbol()` that takes no parameters. It returns the value of the *symbol* field.
- ☐ A public void setter method named `SetSuit()` that takes a string parameter. It assigns the parameter to the *suit* field.
- ☐ A public void setter method named `SetSymbol()` that takes a string parameter. It assigns the parameter to the *symbol* field.
- ☐ A public string constant method named `ToString()` that takes no parameters. It returns a string in the format `[`*x*`:`*y*`]` where *x* and *y* are the values of the fields *symbol* and *suit* respectively.
- ☐ A friend ostream operator. It returns the display exactly like the return of the `ToString()` method.

In the **AbstractDeck.h** file, define the class *DeckInterface* that contains

- ☐ A public pure virtual void method named `Shuffle()` that takes no parameters.
- ☐ A public pure virtual *Card* method named `Draw()` that takes no parameters.
- ☐ A public pure virtual bool method named `Load()` that takes a constant *Card* reference parameter.
- ☐ A public pure virtual bool constant method named `IsEmpty()` that takes no parameters.
- ☐ A public pure virtual string constant method named `ToString()` that takes no parameters.

In the **Deck.h** file, define the class *TarotDeck* that publicly inherits *DeckInterface* and contains

- ☐ A private *Card* array field named *deck* with a size of 22.
- ☐ A private int field named `size`.
- ☐ A public default constructor that assigns 0 to the *size* field.
- ☐ A public copy constructor.
- ☐ A public assignment operator.
- ☐ A public empty destructor.
- ☐ A public overridden `Draw()` method. It returns the top *Card* in *deck* if *deck* is not empty; otherwise, it throws an error message.
- ☐ A public overridden `Shuffle()` method. It shuffles the elements of *deck*.
- ☐ A public overridden `Load()` method. It add the parameter to the bottom of *deck* only if the parameter represents a valid missing card from *deck* (i.e. it is a valid tarot card that is not currently in the deck).
- ☐ A public overridden `IsEmpty()` method. It return true if *deck* is empty; otherwise, it returns false.
- ☐ A public overridden `ToString()` method. It returns a string of the elements of *deck* in a list.
- ☐ A friend ostream operator. It returns the display exactly like the return of the `ToString()` method.

Note: make sure to include the appropriate libraries and header file in each header file. Furthermore, the suit of a tarot deck are 22 unique numbers from 0 to 21 and there are 22 unique symbols. Do your research on the major arcana tarot cards for more details. Last, you can add any fields or methods need to the TarotDeck class to assist you.