CS 112 - Week 14 Lab Exercise - 2022-12-02

Deadline

Due by the end of lab on 2022-12-02.

How to submit

Submit your `.cpp` and `.h` files for the problems below on https://canvas.humboldt.edu.

IF you prefer, you may instead compress your `.cpp` and `.h` files to be submitted into a single `.zip` file and submit that `.zip` file to Canvas.

(I'll also accept the `.zip` file created when one downloads a folder from the CS50 IDE, as long as it includes all of your lab's `.cpp` and `.h` files -- I suspect it will also contain your resulting executables, but that's OK.)

Purpose

To practice lightly with polymorphism and dynamic/late binding, a pure abstract class, and multiple inheritance.

Important notes

- Be sure to put BOTH of your names and today's date in each of the files you modify or create for this lab exercise.
- When you are done, or before you leave lab, the driver/whoever's account has the lab exercise files should e-mail a copy of all of the files to BOTH/ALL of you, and EACH of you should submit these files on Canvas.

Lab Exercise Setup

- FIRST: in the CS50 IDE, in your folder for today's lab exercise, create copies of the following:
  - `Point.h` from **Week 14 Lecture 2** and `Point.cpp` from **Week 13 Lecture 2**
  - `ColorPoint.h` and `ColorPoint.cpp` from **Week 13 Lecture 1**
  - One of your team's `ThreeDPoint.h` and `ThreeDPoint.cpp` classes from the **Week 13** Lab Exercise

Problem 1 - making a small style-based tweak to ThreeDPoint.h and ColorPoint.h

In the Week 13 Lecture 1 version of `ColorPoint.h`:

- add an `adapted by:` line in the opening comment listing your names
- add a line to the beginning of `last modified:` with today's date, and a note that you are adding syntax to make it clear that methods `display` and `to_string` are now going to be **overridden** instead of redefined
- follow the good-style suggestion from Savitch of adding `virtual` to the beginning of the `ColorPoint`

method headers `display` and `to_string`, to make it clear these are now overridden rather than redefined (because of the `virtual` added to the front of these methods in base class `Point`)

And, In your team's chosen version of `ThreeDPoint.h`:

- add an `adapted by:` line in the opening comment listing your names
- add a line to the beginning of `last modified:` with today's date, and a note that you are adding syntax to make it clear that methods `display` and `to_string` are now going to be **overridden** instead of redefined
- follow the good-style suggestion from Savitch of adding `virtual` to the beginning of the `ThreeDPoint` method headers `display` and `to_string`, to make it clear these are now overridden rather than redefined (because of the `virtual` added to the front of these methods in base class `Point`)

Problem 2 - seeing dynamic/late binding in action

In a `main` function in a file `dynamic-demo.cpp`:

- Create a vector or an array -- your choice! -- able to hold at least 6 elements, where each element is of type pointer to a `Point`.
- Initialize your vector or array so that:
  - at least two of its elements point to dynamically-allocated `Point` objects
  - at least two of its elements point to dynamically-allocated `ColorPoint` objects
  - at least two of its elements point to dynamically-allocated `ThreeDPoint` objects
- Loop through your vector or array, calling the `display` method for the object pointed to by each of its pointers, and also printing to the screen the result of calling the `to_string` method for the object pointed to by each of its pointers.
  - Because of the `virtual` added to this method header in the `Point` class, and because these are pointers to dynamically-allocated instances of `Point`, `ColorPoint`, or `ThreeDPoint` instances, dynamic binding *should* happen so that you see the color data field values for `ColorPoint` objects and the z data field values for the `ThreeDPoint` objects.
- (Add any other statements for things you'd like to do with your vector or array.)
- Loop through your vector or array, calling `delete` for each of the dynamically-allocated objects being pointed to.

Problem 3 - writing a pure abstract class

Recall:

- A **pure virtual method** is:
  - declared in a class definition with a method header that begins with the keyword **`virtual`** and ends with the so-called **pure specifier**, **`= 0`** (before the method header's closing semicolon)
  - and, there is NO implementation of this method written for this class.
- A **pure abstract class** has only pure virtual methods and no data fields or concrete methods.

Thinking about our class style this semester for separately-compiled classes, then, such a class would have no `.cpp` file, would it? It would just have a `.h` file! And I think it might only need a `public:` part, since

it does not have data fields, and I'm not sure how private methods would even work in a method with no concrete methods.

(And, because it has less baggage, if you will, it might make for some less-complicated multiple inheritance practice, I hope.)

Create a pure abstract class named `Rankable` in a file `Rankable.h` that:

- uses the usual `.h` file template
- includes just one pure abstract method, named `compute_rank`, that expects nothing and returns the computed rank of the calling `Rankable` instance
- FUN FACT: https://stackoverflow.com/questions/24316700/c-abstract-class-destructor suggests that we also want a virtual destructor with this pure abstract class, that does nothing!
  - SO: add this to the `public:` part of your `Rankable` class definition, also:

```
// because it turns out you want a virtual destructor that
//     does nothing for your pure virtual class, also!

virtual ~Rankable(){};
```

I believe your result here should be a single, quite short file named `Rankable.h`.

Problem 4 - trying out multiple inheritance

To practice lightly with multiple inheritance, choose either class `ColorPoint` or `ThreeDPoint`, and **modify** it into a derived class with **TWO** base class parents, `Point` and `Rankable`.

Since `Rankable` happens to be a pure abstract class, I *think* you will only need to:

- (Add an `adapted by:` comment-line to your chosen class' `.cpp` and `.h` files if not already there for today's lab, and in either case add to `last modified:` to include that you are having it now also be derived from `Rankable`)
- Add the appropriate `#include` to include `Rankable.h` in both your chosen class' `.h` file and `.cpp` file.
- Modify your chosen class' header in its class definition to also include `Rankable` as a base class/parent class, in addition to `Point`
- Add the method header for `compute_rank` to your chosen class' definition in its `.h` file, and add your chosen class' implementation of `compute_rank` to its `.cpp` file.
  - You can decide how rank is computed for your selected class! Make sure it is somehow computed based on your selected class' data fields -- for example, you might add or average or multiply its x and y (and z?) coordinates, let color influence its rank, etc. But as a practice problem, keeping this computation simple is fine!

Then, try out at least the following in a `main` function in a file `multi-demo.cpp`:

- Create at least two instances of your chosen class, and print to the screen the result of comparing calls to `compute_rank` for each of those instances to what they should return.
- And, to show that your class' instances are also of type `Rankable`:
  - create an array or vector of at least four pointer-to-a-**`Rankable`** instances

  - dynamically allocate that array or vector's pointers to point to objects of your selected class (which *should* be seen as being also of type `Rankable`)
  - loop through your vector or array, printing to the screen the result of calling method **`compute_rank`** for the object pointed to by each of its pointers
- (Add any other statements for things you'd like to do with your vector or array.)
- Loop through your vector or array, calling `delete` for each of the dynamically-allocated objects being pointed to.

- When you are done, or before you leave lab, use Gmail to
  - MAIL a copy of ALL of the resulting `.h` and `.cpp` files to BOTH of you, and
  - EACH of you should SUBMIT these files on Canvas