Assignment 3: Simple Classes / Objects

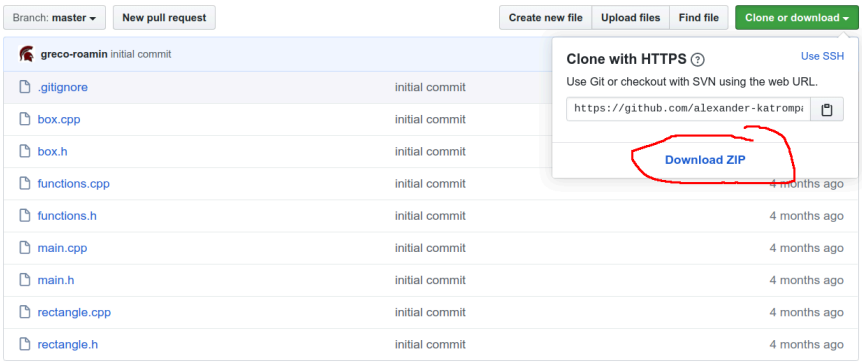
Description and purpose: To demonstrate (and advance) your knowledge of classes, header files, objects, compiling, version control, and GitHub. At the completion of this assignment, you should be ready to begin working on real data structures code. This assignment contains all the techniques and processes you will use throughout the semester. It should be just a review of Programming II, but if you are missing any of this information, this is your opportunity to get up to speed.

Background

- Understanding of all best practices notes, and lectures.
- Understanding commenting, paying particular attention to how to comment class header files.
- Watch the video detailing this assignment.

Instructions:

- Download the code from this repo. Place the code in an empty directory.



- Compile the code using the command: **g++ -I ./ *.cpp**
- Run the program by typing **a.out** or **./a.out**
- **Take a screenshot of your compilation and run results.** It will look something like this…

```
alex@alex-laptop:~/Dropbox/Sites/accgrading.local$ g++ -I ./ *.cpp -o out
alex@alex-laptop:~/Dropbox/Sites/accgrading.local$ out
0
0
0

5
4
20

0
0
0
0
0

5
4
6
120
148
```

- Study the Classes Starter code and make sure you understand the structure of the files, how header files work, how inheritance works, how pass by value and pass by reference work, etc. This is all material you should have taken away from Programming II, so this should be just a review.

Now it's time to code your own work....

- The GitHub Classroom invite: https://classroom.github.com/a/7V7n51wM
- Follow the *assignment-specific instructions* on the GitHub Classroom page.
- When you have reached the step, "*Now you are ready to begin working,*" you will have only a **README.md** and **.gitignore** in your directory. Fix the **README.md** according to the README guidelines. You can leave the **.gitignore** alone; it's complete and correct.
- Commit that change with the message "*updated README with the application and code descriptions.*"
- Push the changes and view them in your repo so you know everything is working correctly.
- Now create your own complete and working console application, which will be similar to the Classes Starter you downloaded. Follow these guidelines:
    - Make your classes as Circle and Sphere.
    - When you make your files, all file names should be **lowercase**.
    - Figure out all the methods you will need that make your classes analogous to the Classes Starter example. You'll have to look up the appropriate formulas.
    - Use a functions module as shown in the Classes Starter to make functions that act on the new objects the same way.
    - Comment everything correctly. Do not forget to look up how to comment class header files and do it correctly. You must include EVERY comment block as shown in the example, even if nothing goes in the block.
    - Structure your files properly (**this applies to all code you will write for this class**).
        - Prototypes and directives go in header files.
        - Code **never** goes in header files (this includes initialization of variables).
        - Prototypes and directives **never** go in cpp files (except the **one-and-only-one** directive to **#include** that file's header).
        - If you use cout, you may **not** use **using namespace std;** but you may (for example) use either:
            - this form: **using std::cout;**
            - or this form: **std::cout << "hello world";**

          Note: The global directive **using namespace std;** will **not** be allowed for any assignment in the course; it is considered bad practice in professional code.
    - Make sure your setters have protections in them so the object cannot be put in an invalid state. For example, in the Classes Starter code, you cannot set height or width below zero because that makes no sense.
    - All properties (aka attributes, aka variables) of classes must be private or protected. This goes for all attributes in all assignments and in the professional world as well.
    - All methods that are public should **need** to be public, and all others should be private or protected as appropriate.
    - Use your own setters inside your objects (especially in the constructor).
    - No getter should ever modify the object; getters are strictly for reading data.
    - Have at least one example of a method override. For example, in the Classes Starter code, **getArea()** is overridden in the box class from the rectangle class.
    - Have at least one example of more functionality in the child class; for example, in the Classes Starter code, the box class has a **getVolume()** method, which the rectangle does not have because it makes no sense for a rectangle to have volume.

- Keep your code and application simple. It does not need to be any more sophisticated or interesting than the Classes Starter code.
- Make a **main()** that tests your application in an automated fashion (do not use user input).

Theory Discussion: In this assignment, you are making an **"is a"** relationship between the base (parent) class and derived (child) class. An "is a" relationship means that the child is a kind of the parent. For example, a Dog is a kind of Canine. A Husky is a kind of Dog. Do not confuse this relationship with the other two object-oriented relationships, **"uses a"** and **"has a."** The relationship "uses a" means one class uses another, but they do not share attributes/methods (usually). The relationship "has a" means one class is an attribute of another class, but they still do not share their attributes/methods (usually).

Examples:

- **is a** : a Husky **is** a kind of Dog.
- **has a** : a DogSledTeam **has** one or more Husky in it.
- **uses a** : a Driver **uses** a DogSledTeam.

In each example above, the thing in blue is a class (i.e. it's a noun). What you are doing in this assignment is the first case. Later in the semester, we will do the other two examples.

Grading: The grading process is detailed on the course website. Read that page carefully! **Pay particular attention to the "automatic zero" section.** Do not lose points or get a zero for failure to follow specifications or best practices. Use that section like a checklist, and go over it **carefully**.

Use this checklist to double-check your code. 👈 IMPORTANT!

Submission: When you are ready for grading, use the write submission feature in Blackboard and submit your repo's link. Unless you have an important note to include about your submission, **just submit the link—no additional information**. If you do have important information to convey, that's fine, but otherwise—just the link. If you need to fix/change something **after** you submit but **before** it's graded, just fix/change it and push again. **Do not resubmit the assignment before getting a grade**. Only re-submit after you get a grade and want a re-grading.