Duck Simulator

CSC-490, Winter 2020

For this worksheet you will need the files located in the CoCalc folder `Quack/DuckSimulator`. This exercise is adapted for C++ from the very similar java program described in "Head First Design Patterns" by Freeman & Robson.

The goal of this worksheet is to practice working with several of the design patterns that we have seen this term. To do so, you will be asked to complete five steps, each of which involves adding a new design pattern to the Duck Simulator program. For each step you need to draw the relevant class diagram and implement the associated code. Each step is worth 20pts (10 for the class diagram and 10 for the implementation).

Step 1. Use the Adapter pattern to allow a `Goose` object to act like a `Quackable` object. After doing so, modify the definition of the variable `goose` in the file `DuckSimulator.cpp` and include the `goose` in the `flock`. When you're finished with this step, you should see the goose honk when you run the program.

Step 2. Create a new class called `Flock` that can hold a collection of `Quackable`s. You should be able to add or remove quackers to each flock. Moreover, `Flock` should have method `quack()`. When `quack()` is called on a flock, all the quackers in that flock should quack. Adjust the file `DuckSimulator.cpp` so that `flock` has type `Flock` or `Flock*`, whichever you think is best. Running the program should not appear any different after this step as long as you include all the same quackers in the flock.

Step 3. Create a class called `QuackCountDecorator` that is a wrapper for `Quackable`. The purpose of this class is to count the total number of times any wrapped quackers quack. This new decorator class should have a *static* data member `_quackCount` and a *static* method `getQuackCount()`.[1] The quack count should be updated every time the wrapped quacker quacks.

Modify the file `DuckSimulator.cpp` so that the mallardDuck, redheadDuck, rubberDuck, and duckCall (but not the goose) are all wrapped with a `QuackCountDecorator`. Add a line to the end of that file that uses the static method in order to print the quack count. You should see the total number of times any wrapped duck quacked (not counting the goose). In other words, if all 5 quackers quack (including the goose), your program should print something like "4 quacks were counted".

Step 4. Create an `AbstractDuckFactory` with methods `createMallardDuck()`, `createRedheadDuck()`, `createRubberDuck()`, `createDuckCall()`, and `createGoose()`. Create concrete factories called `DuckFactory` and `CountingDuckFactory`. The `DuckFactory` should create unwrapped quackers, and the `CountingDuckFactory` should create wrapped quackers (wrapping all but the goose).

Add a new data member to `DuckSimulator` of type `AbstractDuckFactory*`. Use this factory to create all the quackers in the implementation of `simulate()`. Get rid of all the unnecessary headers in `DuckSimulator.cpp`. If your `DuckSimulator` is pointing to a `CountingDuckFactory` then running the program should look just like it did after step 3. If it's pointing to the `DuckFactory` then the quack count should show up 0.

last updated: January 23, 2020

[1]Making the method and data member static will allow us to count the total number of quacks from *all* the wrapped quackers (as opposed to keeping track of each quackers count individually).

Step 5. Create a new class called `Quackologist`. This class should have only one method called `update(description)`. When `update(description)` is called the following message should be printed to the terminal:

```
Quackologist: description just quacked.
```

Use the Observer pattern with `Quackologist` playing the role of the concrete observer and `Quackable` playing the role of the concrete subject (even though `Quackable` is not concrete!). The Quakologist should be notified whenever any individual quacker quacks.

Add a new data member to `DuckSimulator` of type `Quackologist` or `Quackologist*`, whichever you think is better. Be sure to attach the quackologist as an observer to all the individual quackers. When you run your program you should see something like this:

```
quack
Quackologist: mallard duck just quacked.
quack
Quackologist: redhead duck just quacked.
squeak
Quackologist: rubber duck just quacked.
kwak
Quackologist: duck call just quacked.
honk
Quackologist: goose pretending to be a duck just quacked.

4 quacks were counted.
```

Bonus: If your program is complete and has no memory leaks you will get an extra 5 points. Remember that you can check for memory leaks using `valgrind`.