State Pattern

In order to complete this worksheet you will need the files located in the folder `State` in CoCalc.

1. According to the GoF, the intent of the strategy pattern is to *"Allow an object to alter its behavior when its internal state changes. The object will appear to change its class."*

   (a) (5pts) Draw the class diagram for the State pattern with a class `Context` along with an abstract class `State` having two concrete subclasses `ConcreteStateA` and `ConcreteStateB`. This is the same class diagram from the GoF using our class diagram conventions. Leave some space for an extra method in `State`.

   (b) (1pt) Write the pseudocode the GoF gives for the implementation of `request()` in the space provided below.

   (c) (2pts) What is the difference between the State and Strategy patterns?

   (d) (3pts) Modify the class diagram above so that each `State` is responsible for keeping track of an additional `State` called `nextState`.

   (e) (3pts) The pseudocode from part (b) should also be modified so that the Context's state is updated when `request()` is called. Decorate the diagram above with the new pseudocode.

   (f) (1pt) Implementations of the State pattern often involve heap memory allocation (i.e. the operator `new` is used). What do good C++ programmers worry about when using the `new` operator?

2. For this problem you are meant to look at the program in the CoCalc folder `State/Context`. In that file you'll see an example of implementation for the class diagram on the front of this worksheet. You will be asked to give line numbers of the code, so don't add any new lines of code (at least, wait until you are done answering the questions). When I ask for a line number, your answer should be something like "line 3 in State.h".

   (a) (2pts) When a Context object is instantiated, what is its initial state? What line number determines the initial state?

   (b) (1pt) If at some point a Context object is in state A, what will its state be after `request()` is called?

   (c) (1pt) Suppose that you want the following to be true: if a Context object is in state B, then it will remain in state B after `request()` is called. What line number would you need to modify?

   (d) (2pts) Why is the data member `_nextState` in `State` protected as opposed to private? What line(s) of code in what file(s) would lead to compile time errors if it was private?

   (e) (2pts) Which class is cleaning up memory leaks? Which specific lines are removing memory leaks? You can check if these lines are really handling memory leaks by commenting them out, compiling the code[1], and then running the program with valgrind[2].

[1] Compile command: `c++ -std=c++11 -Wall -o context *.cpp`
[2] Then run with: `valgrind ./context`

3. For this problem you will need to look inside the folder in CoCalc called `State/Mario`.

   (a) (5pts) Draw a class diagram for the program in the Mario folder. For each class, leave enough space for one extra method to be added later. You'll also want to leave enough space to add one more concrete state later.

   (b) (5pts) The program uses the State pattern. What are playing the roles of the following:

      i. `Context`:

      ii. The abstract state:

      iii. Concrete states:

      iv. `request()`:

      v. `handle()`:

   (c) (5pts) Include a new method called `powerDown()` in all the classes in your class diagram (are any abstract?). Calling the power down method on Mario should change his state from Big to Small, and Small to Dead. That means you'll need to add a new concrete Mario state to your diagram too.

   (d) (15pts) Implement the power down method in the code provided in CoCalc. To do so you should add `powerDown()` methods to multiple files[3]. You'll also need to add a new .cpp file and .h file for the new concrete Dead Mario state. I'll leave it to you to choose how a dead Mario powers up or down.

[3]To keep things organized you should put implementations of all the `powerDown()` methods in the .cpp files, and just declare those methods in the .h files

4. Consider the following program that uses the State strategy:

   - There is a class `H2O` that has a data member called `temperature` and a (setter) method called `setTemperature()`.
   - `H2O` has a `H2OState` which can be one of `SolidH2OState`, `LiquidH2OState`, or `GasH2OState`.
   - Each state has a method called `handleTemperatureChange()` that handles temperature change (i.e. handles the setter in `H2O`).
   - `SolidH2OState` has a method `melt()`.
   - `LiquidH2OState` has methods `freeze()` and `evaporate()`.
   - `GasH2OState` has a method `condense()`.

   (a) (10pts) Draw a class diagram for the program.

(b) (2pts) Which of the methods should have parameters and what parameters should they have?

(c) (3pts) Write pseudocode for the implementation of `setTemperature()` in the space below.

(d) (5pts each) Write pseudocode for the implementation of `handleTemperatureChange()` in each of the concrete states. In addition to indicating how to set the nextState, your pseudocode should indicate when methods like `melt()`, `freeze()`, etc. will be called.

i. `SolidH2OState`:

ii. `LiquidH2OState`:

iii. `GasH2OState`:

(e) If you have extra time, you should try to implement your design in C++.