CS 245
Assign 4: Strategies — Wednesday Mar 27

Objective:

- Apply the Strategy Pattern to a practical application.

Problem:

Previously we have discussed the need to have bounded integers in our programs. It is a very useful construct in dealing with data that needs to be limited in order to be properly understood:

- When entering grades, I want to be restricted to values between 0 and 100. If I accidently enter too large of a number, I need to be prompted to re-enter it. (likewise too small, but it is less likely that I am going to accidently type a negative sign).
- Timers have bounds – after each tick, the number of seconds exceeds the limit, I want to increment the number of minutes, then wrap the timer to the beginning.
- Candy.com is having a sale on Gobstoppers… but it is such a great bargain, they have a limit of ten. When I try to put 500, in my online cart – it enforces the limit and only adds 10 to my order.

Each of these tasks can be handled by custom code that addresses the logic specific to the problem, but all of these problems are essentially the same --- enforce a boundary on "legal" integer values. Do we need to reinvent the wheel through the lens of each specific problem?

You should also note that while the problem is the same – the strategy for dealing with the situation varies. Sometimes I want to force the client to handle it (i.e. make them re-enter); sometimes I wrap the value (timer); sometimes I will accept the cap.

Activities:

1. Re-implement or refactor the BoundedInteger project to demonstrate the Strategy Pattern.
   - As previously defined, BoundedInteger should support the following methods

```
BoundedInteger(int value, int lower, int upper)
void setValue(int)
int getValue()
void addWith(int)
```

   - When constructed – the Bounded Integer should verify that the bounds are legal; when the lower bound is larger than the upper bound – the class should throw an IllegalArgumentException.

- The project should support three distinct Strategies
    - Exception: Whenever the value is out of bounds, an OutOfBoundsExeception is thrown.
    - Wrap: Values are kept within the bounds by using modulo arithmetic to wrap values that external to the range. (i.e. the upper +1 = lower; lower -1 = upper)
    - Cap: Extreme values are assigned to the bounds: Extreme low values = lower

2. Create a main class/method that can exercise these strategies. This class should
    - Prompt the user for a filename and be able to read the file.
    - Create an output file name based on the original filename prepended with "out-". For example, if the filename was "nums.txt" the output would be "out-nums.txt"
    - Files are anticipated to be of the following format:
        - Line1 contains a string that has the name of the strategy ("Exception", "Wrap" or "Cap")
        - Line 2 contains two integers representing the lower and upper bound
        - Line 3:N contain a single integer value
    - After reading the first two lines of the file, the program should:
        - Create a new BoundedInteger and assign the value to the middle of the range (i.e (upper+lower)/2
        - Print the value of the BoundedInteger to the output file – (Just the number)
    - After reading each additional line, the program should:
        - Create a new bounded integer using the number read as the value
        - Print the value of the BoundedInteger to the output file – (Just the number)
        - Add the number read to the initial (midpoint) bounded integer
        - Print the value of the BoundedInteger to the output file – (Just the number)
    - Note: in the event that your program throws an exception you should NOT handle it. However, you should confirm that it is an anticipated exception and that it was thrown under the specified conditions. (i.e. if your program is throwing a DividebyZeroException, you have some fixing to do… If you are getting an OutOfBoundsException --- that *might* be ok… if the conditions are correct).

Your code will be evaluated on functionality ***and*** adherence to good OO design principles