INFO5125

Project 1

You will be creating a program that evaluates stacks of heavy boxes. Stacks of boxes can be evaluated in a variety of ways, such as how pyramid-like they appear, or how stable they are (with heaviest boxes on the bottom). Your goal is to implement both the box stacks and the evaluation methods for them as a C# program.

Details

HeavyObject

- The starter project contains a defined class for HeavyObject.
- This object has member variables for its dimensions (length, width, height) as well as density.
- Handy formulas:
    - Volume of a box = Length * Width * Height
    - Mass = Density * Volume
- Volume and Mass are given to you in the HeavyObject class as accessors.
- You don't have to use all of these properties, but they might come in handy!

Iterator

- You should make a new Aggregate class that can create Iterators. This will be your list of heavy objects.
- Each Aggregate class should hide the underlying C# collection (like List) from the public.
- Aggregates should support methods to Add a HeavyObject, get a HeavyObject at an index, and get the length of the list.
- Aggregates need to support a CreateIterator() method that returns an iterator as described in class. The iterator should have all the methods described in class as well as whatever methods you feel are necessary (for example, GetPreviousItem might be a good function to implement).

Strategy

- Your evaluation metrics should take the form of a Strategy.
- Strategies need to take as input a concrete Aggregate class and output a floating-point number.

- This number represents the score of the list (higher is better).
- You should evaluate stacks based on three criteria:
    - **BottomWeight** - rewards stacks where weight is concentrated near the bottom of the stack
    - **Pyramid** - rewards stacks where it is shaped like a pyramid (boxes with larger dimensions on bottom)
    - **Topple** - rewards stacks that are likely to topple (fall over)
- These strategies don't have to be perfect mathematically correct but should resemble what they're trying to do!
    - No need to match my output! But it's nice if you can figure out how I did it.
- **NOTE: Assume that the list of HeavyObjects starts at index 0 on the bottom of the stack and each subsequent HeavyObject would be placed ontop of the previous.**

Flyweight

- You should create a Flyweight Factory for your Strategies.
- Because a Strategy likely doesn't have any state at all, you probably don't have to modify your Strategy classes to make them flyweights.
- Your FlyweightFactory is a concrete class. As such it's fine to reference the concrete classes of Strategy inside its GetFlyweight method.

Starter Content

You are provided with a starter project with a main method and a HeavyObject class.

Uncomment the main method as you implement the above patterns and objects to test your progress.

The starter package contains:

- Assi1.csproj - the project file for the solution. Open this if you're using Visual Studio.
- HeavyObject.cs - the HeavyObject class
- Program.cs - the main entry point/main method
- Folders for Aggregates, Flyweights, and Strategies

You will also receive a built .exe of the solution.

The starter project was written using .NET Core 3.0. If you want to deviate from this, feel free to create a new project in Visual Studio and copy over the files.

Rubric

Detail	**Points (%)**
Aggregate ● Implements necessary list operations ● Hides underlying List ● Creates Iterator Iterator ● Implements operations (First, Next, Current, IsDone)	 15 5 5 15
Strategy ● Three strategies implemented ● Strategies accomplish their stated goal	 15 10
Flyweight ● Strategies are only retrieved from Flyweight Factory ● Flyweight Factory returns flyweights/singletons	 5 10
Misc. ● Program runs correctly without errors or warnings ● Program adequately commented	 15 5
Total	**100**