CSE 1321L: Programming and Problem Solving I Lab

Lab 13

Introduction to Object-Oriented Programming Part II

What students will learn:

- Encapsulation and OOP
- Review of creating classes
- Constructor overloading
- Getters and setters

Overview: At this point in the course, you've written a few classes and created a few objects. For this lab, we're going to extend your knowledge of class design by developing classes with more functionality. As a review, remember that creating a class follows a pattern:

1. Declare the class (e.g., "class Dog")
2. Add attributes (variables) in the class (e.g., weight, name, hair color)
3. Add a constructor – a special method used to initialize the attributes in step #2
4. Add methods – these are behaviors of the class, or things it can do

We're going to expand on steps 2 and 4. Here's a quick explanation:

Attributes: You typically mark attributes with a special keyword called private. This means that you cannot directly access the attributes. For example, if we had a Dog class with weight, name, and hair color, and wrote the following code in main:

```
Dog d1 = new Dog();    // Create a dog
d1.weight = 15;        // Try to directly access an attribute
```

The second line above would throw an error if the attributes were marked private. Why would we want this? Because you don't want to let just any program modify Dog. That's why we have…

Getters/Setters: These are also called Accessors/Modifiers – which are simply two methods that are included per attribute (one to modify/set the attribute, one to read the attribute). For example, in the Dog class, you may have a modifier that sets the weight of the dog only if the value is positive:

```
void setWeight (int newWeight) {
   if (newWeight > 0) {
     weight = newWeight;
   }
}
```

This helps with encapsulation, meaning that our attributes are contained (and manipulated) from within the class itself; it's a way of protecting the integrity of the class.

As always, your filenames and class names should follow the conventions we've used all semester (i.e., filenames should be Lab13A(.java, .cs or .cpp)) and class names should be appropriate as well (e.g., Lab13A).

Lab13A: The Architect.

Buildings can be built in many ways. Usually, the architect of the building draws up maps and schematics of a building specifying the building's characteristics such as how tall it is, how many stories it has, etc. Then the actual building itself is built based on the schematics (also known as blueprints). Now it is safe to assume that the actual building is based off the blueprint but is not the blueprint itself (and vice versa).

The idea of a classes and objects follows a similar ideology. The class file can be considered the blueprint and the object is the building following the analogy mentioned above. The class file contains the details of the object i.e., the object's attributes (variables) and behavior (methods).

Please keep in mind that a class is a template of an eventual object. Although the class has variables, these variables lack an assigned value since each object will have a unique value for that variable. Think of a form that you may fill out, for example, for a bank; this form has many boxes such as ones for your first name, last name, etc. That form is analogous to a class; it's generic and does not have any unique information. Once someone picks up the form and fills it out it becomes unique to that person and is no longer a generic form; this is analogous to an object.

For the one and only exercise in Lab 13, you will need to design a class and create objects from this class in your main method. Please read and follow the instructions below carefully

To start, you need to create a class BuildingBlueprint, and it should have the following:

- Variables (Attributes):
  - A variable that represents the number of stories the building has. This should be an integer – must be private
  - A variable that represents the number of apartments the building has. This should be an integer – must be private
  - A variable that represents the occupancy rate of the building (It contain numbers between 0 and 1 representing the percentage of occupancy). This should be a float – must be private
  - A variable that indicates if the building is fully occupied (all the apartments have been purchased) which is only becomes true if the variable for the occupancy rate is 1 and changes back to false if the occupancy falls to any value below 1. This should be a Boolean – must be private
- Constructors:
  - A default constructor that when used creates a building object that has the following default values:
    - 10 Stories
    - 20 apartments
    - Occupancy variable set to 100% (1.0)
    - Fully Occupied variable set to true
  - Overloaded (Argument) Constructor that should accept any value for the number of stories, number of apartments and occupancy percentage. This constructor should only set the fully occupied variable to true if and only if the provided occupancy percentage is 1.0 .
- Methods (Behavior)
  - Getters and Setters for the occupancy rate variable
  - Getter for the number of stories variable
  - Getter for the number of apartments variable
  - Getter for the Boolean value of full

Next, create class Lab13A with a main method or driver which should do the following:

- Create two building objects:
  - `buildingOne` should be created using the default constructor
  - `buildingTwo` should be created using the overloaded (argument) constructor with the values provided:
    - 30 Stories
    - 30 apartments
    - Occupancy variable set to 75% (0.75) – you may have to cast to float
- You should then print out the information of `buildingOne` like the example shown below in the sample using the getters
- You should then print out the information of `buildingTwo` like the example shown below in the sample using the getters
- Use the Setter of the occupancy percentage of `buildingOne` to change its value to 0% (0.0)
- Use the Setter of the occupancy percentage of `buildingTwo` to change it to 100% (1.0)

Remember, the name of the class containing the main method should be Lab13A. Please make sure that your two classes (Lab13A and BuildingBlueprint) are in the one file

Please keep in mind that everyone's output should be the same and as shown below…

```
Sample (AND ONLY) output:
Year 2020:
Building 1 has 10 floors, 20 apartments, and is 100% occupied. Full? true
Building 2 has 30 floors, 30 apartments, and is 75% occupied. Full? false

Many years pass.

Year 2043:
Building 1 has 10 floors, 20 apartments, and is 0% occupied. Full? false
Building 2 has 30 floors, 30 apartments, and is 100% occupied. Full? true

Looks like people prefer taller buildings.
```

Instructions:

- Programs must be working correctly.
- Programs must be saved in files with the correct file name.
- If working in Java or C#, class names must be correct.
- Programs (only .java, .cs or .cpp files) must be uploaded to Gradescope by due date.