CSE 1321L: Programming and Problem Solving I Lab

Lab 12

Introduction to Object-Oriented Programming Part I

What students will learn:

- Encapsulation and OOP
- Creating a class
- The purpose of constructors
- Creating an object in main
- Understand the difference between a class and an object
- Review of methods

Overview: What you've seen up to this point is called procedural programming. What this means is that we structured programs based on the concept of functions/methods (which are also called procedures). For this lab, you're going to start programming using Object-Oriented Programming. OOP is a completely different way of thinking about programming and may seem unusual at first. That being said, you should be well prepared for OOP, since you've already been exposed to the concepts of variables and methods. A class just puts those two things together, which is a technique called encapsulation. We're going to focus on the process of building simple classes. Next week's lab will be a bit more complex, so make sure you understand what's going on.

The main idea behind OOP starts with a class – which is a new data type, just like an int, string, float and so on. A class can be just about anything, but usually represents a thing in the real world, like a Dog, a Bank Account, or a Beverage. Then (very much like working with primitive data types), you have to a declare a variable of that type. That variable is an object – and hence the term Object Oriented Programming. The interesting part is that classes have things that they can do in the form of methods, which is why methods are sometimes called behaviors.

Just like the lecture slides state, designing and implementing a class usually occurs using the following steps:

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

Then, in main, you can create variables of the class type the same way you create other variables (e.g. int myInt = 0;) However, in OOP, you have to use the "new" keyword, which calls the constructor method of the class and "brings the object to life":

```
Dog d1;               // The dog is current "dead"
d1 = new Dog();       // The dog is alive
d1.bark();            // Call the bark method in the dog
```

Once you've written a few classes, you'll see the pattern. So let's get started.

As always, your filenames and class names should follow the conventions we've used all semester (i.e. filenames should be Lab12A(.java, .cs or .cpp)) and class names should be appropriate as well (e.g. Lab12A).

Lab12A: A Tale of Two Classes

When learning about objects, you may hear the example "a chair is an object". Well, for this lab, we are going to implement this example.

Create a class called "Chair", this is going to be a class which can create "Chair" objects. Also create a separate class to drive the function of Chair; call this driver class "Lab12A". The driver class will contain the main method you typically make while Chair will not have a main method.

Chair class:

- Variables (Attributes): - notice they are public. This isn't normal, but it's ok for this lab.
  - `public int numOfLegs //how many legs are on the chair`
  - `public bool/boolean rolling //does it roll or not`
  - `public String/string material //what is the chair made of`

Using these attributes you can describe most chairs. For example, you might have a wooden chair with 4 legs that does not roll, or you may have a rolling chair made of leather with 5 legs.
Now that you know the parameters and their data types, you can make a constructor - also called Chair. A constructor is a special method that shares its name with the class name. It is responsible for creating the object and starting it off with values for its attributes. It should take in user inputs to provide meaningful info for each attribute.

This class isn't going to have any methods, so it should only include a constructor and the attributes.

Driver class (Lab 12A):
This class is going to have a main method and take user input for the attributes for the Chair object. Once you have all that information, create a new object with those variables as parameters.

Once you have done that, print out the information about your chair. You can use the name of your Chair object and the dot operator (a period) to print out info about the chair you created.

Then, to show you really understand how the dot operator can work, change the attributes to:

```
numOfLegs = 4;
rolling = false;
material = wood;
```

Finally, repeat your print statement you wrote earlier to print out what the chair looks like.

Tip: When taking in parameters to a constructor, you may want to have the variables used in the constructor header be different words from the attributes in the Chair class. When dealing with rolling/not rolling you may want to use an IF statement with different print statements.

Remember, the driver class name should be Lab12A.
Both classes have to be in the same file, the file should be called Lab12A
The user input is indicated in bold.

Sample output:

```
You are about to create a chair.
How many legs does your chair have: 8
Is your chair rolling (true/false): true
What is your chair made of: plastic

Your chair has 8 legs, is rolling, and is made of plastic.

This program is going to change that.

Your chair has 4 legs, is not rolling, and is made of wood.
```

Lab12B: My dog can do tricks

For this lab we once again are going to create 2 classes, 1 called Dog and 1 called Lab12B.
Dog objects have a few attributes, but this time unlike chair objects they can also do some cool things too. Each action is represented by a method. Therefore, for any action our Dog can do or we do to the Dog, we are going to create a method. For example, if I want my Dog to bark, I can create a method to do that in the Dog class and call that method in the driver Lab12B (once I have created an object.)

Dog class:

- Variables (Attributes): - make these public, like the first exercise
  - `int age` `//current age of the dog`
  - `double weight` `//how much does it weight in lbs`
  - `String/string name` `//what is the name of the dog`
  - `String/string furColor` `//color of the dog's fur/hair`
  - `String/string breed` `//what breed is the dog`

- Behaviors (Methods): - these also should be public
  - `bark` `//prints "Woof! Woof!"`
  - `rename` `//take a string and change the name of the dog`
  - `eat` `//take a double and add that number to weight`

Keep in mind you need to have a return data type for each method and what parameters these take to carry out their function when creating the methods.

Driver class:

This class will have our typical main method. Inside of this method, create a new Dog object and prompt the user to input the attributes describing this Dog. Once done, print out all the details about the Dog.

Next, use the methods you created in the Dog class to have it bark, change the name (using the rename method, not the dot operator), and feed it.

Finally print out all the details about the Dog, the object should have changed because of your calls to the various methods.

Sample output:

```
You are about to create a dog.
How old is the dog: 5
How much does the dog weigh: 30.5
What is the dog's name: Patches
What color is the dog: chocolate
What breed is the dog: lab

Patches is a 5 year old chocolate lab that weighs 30.5 lbs.

Woof! Woof!
Patches is hungry, how much should he eat: 5000.3
Patches isn't a very good name. What should they be renamed to: Sparky

Sparky is a 5 year old chocolate lab that weighs 5030.8 lbs.
```

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.