The purpose of this lab is to get more practice with setting up java classes. And to get practice in writing methods and using encapsulation.

**Note**: You can use the same java project and package within the project as you did last week – or set up a *new* package within the same project. Or set up a new project and package. Up to you…

**Part 1 – Java class**

Create a new class called Animal. Give it the following class members:

**5 attributes in your Animal class** - (1) name, (2) breed, (3) age, (4) domesticAnimal (or not), (5) colour;

For now, DON’T mark them as private.

**2 constructors in your Animal class**

* One constructor that sets up just the animal name;
* Another constructor that sets up all 5 attributes.

**a toString() method in your Animal class**

As we did last week in the lab, add a toString() method to your Animal class that returns a String that contains the object data in a readable way e.g. “This animal is called Jerry, is a dog that is white in colour and has 4 legs and is domestic”..

Create another java class (call it Control) and put a main method in it. In this “main” method, instantiate the following Animal objects, including:

A domestic 4-legged brown dog called Spot

An animal called “Leo”;

… more of your own choice.

From the main method, print out each object your create (Using System.out.println(objectname).

**PART 2 – Encapsulation - hide**

The concept of encapsulation is enclosing or “hiding” things. OO programming uses this concept to protest the data (attribute values) of objects so that they can only be changed in a controlled way. Let’s illustrate this:

In your “Main” method of your control class, try to print out the attribute values directly of any object you have created e.g.

System.out.println(*whateveryourobjectnameis*.name);

System.out.println(*whateveryourobjectnameis*.domestic);

System.out.println(*whateveryourobjectnameis*.numberOfLegs);

UNLESS your attributes in your Animal class have been marked private, you’ll be allowed to access these attributes of your object from another class, and to see their values. See if you can update them to new values. i.e. animal.name = “.. whatever”.

Now mark each attribute in your Animal class as private – e.g.

Eg. Private String name ;… The compiler will then prevent you from access the attributes directly from another class.

Marking the attributes as “private” in the class is the first step to *encapsulating* them.

**PART 3 – getters and setters methods to support Encapsulation**

Once you have your attributes all set to private from Part 2, you can then add methods to your class that act as “gateways” to these attributes- for “setting” and “getting” the values.

**Getter and Setter methods for the name attribute**

* Write a method public void setName (String name) that allows the name attribute to be changed to the name value being sent in.
* Write a method called getName that returns the value of the name attribute as a String

From your main method, using an animal object you set up earlier – call your new getter method and setter methods to see and change the name attribute. e.g. set the name to “Sam”, now check it is changed etc.. .

**Other Getters and Setters**

**Eclipse has way to generate getter and setters methods for your attributes automatically. Use the “Refactor/ Encapsulate” function to generate getter and setter methods for the other four attributes.**

From your main method, using an animal object you set up earlier – call your new getter and setter methods to get and set each of the attributes values.

**PART 4 – Using methods**

In your animal class, add a method called makeNoise();

In it, just print out a noise… put in an if condition to do this. If the breed is a dog, print out a Bark, if it’s a cat, “miaow” etc.. for whatever breeds you have.

A sample syntax for “if-else” in java is shown below (or use a “switch” statement if you prefer)

if (someeString.equals(“somevalue”))

{

// do whatever

} else if (…..// some check/// )

{

// do whatever

}

etc

Test your makeNoise() method by calling it for your animal objects in the main method.

**PART 5 – Overloading methods**

Overloading methods means having more than one “version” of a behaviour ..i.e. having more than one method in a class with the same name but different method signatures.

Add a second makeNoise method – but this times it takes in a boolean parameter old. If old, then the noise will be “Quiet animal” otherwise, the original makeNoise behaviour happens.

Try out both versions (makeNoise()and makeNoise(String volume) methods on your animal objects from the main method in the Control class.

**PART 6 – Prompting the user for keyboard input (optional)**

If you have finished the other parts, get the main method in the control class to prompt the user for keyboard input e.g. “Enter the breed” etc.. in order to get attributes values for setting up a new object – instead of hard coding. The code for keyboard input is:

/\*\*

\* Code to read an input string from the keyboaard

\*/

Scanner input = **new** Scanner(System.***in***);

System.***out***.print("Enter something > ");

String inputString = input.nextLine();

System.***out***.print("You entered : ");

System.***out***.println(inputString);