CS264SS Software Design

Department of Computer Science
National University of Ireland, Maynooth

Lab 5 assignments July 24th and 26th 2018

Welcome to your Fifth Lab for CS264SS.

Having completed this lab you should consolidate your understanding of C++ OO classes and how they encapsulate data and functions. You should be familiar with the concepts of information hiding and be able to use OO inheritance as well as describe inheritance using UML. (Are you becoming more familiar with the concepts in **Appendix B?**)

From a programming perspective, you will be able to implement simple C++ classes which are part of an inheritance hierarchy and use those classes to instantiate objects and use these objects (instance of those classes) within your programs.

NB: In order to complete question 3 Part C, you will need to have a grasp of "pointers to derived types" and "Virtual functions and Polymorphisms". These are covered in section 10.8, 10.9, and 10.10 in "The Beginners Guide to C++" by Herbert Schildt.

NB: Please use the code from the folder on moodle titled **"Module 10 corrected code"** instead of the official listings from the book as there problems with out of date code with the original examples.

Examine the inheritance examples at the website http://www.cplusplus.com/doc/tutorial/inheritance/ . Build the polygon class hierarchy example and make sure that you understand all aspects of this. Use this example to guide the structure of your programs for the questions in Part B.

For Lab assignments you should answer **Part A** question **1**, **Part B** questions **1-3** and **Part C** question **1**.

Part A

(1) Use the example of the Rectangle Class in the Lab 4 example code folder on Moodle to help you rewrite (refactor) the `Dog` Class question form (2) from the lab 4 assignments (**this question is reproduced in Appendix A**).
You should have the following separate files:

i) The class definition of the `Dog` Class
ii) The declaration of the functions in the `Dog` Class (remember the class scope operator ::)
iii) The program that creates the two `Dog` Class instances.

Notice that this folder contains two different ways of creating a Rectangle Class definition and using this to create functions.
The file `ARectangle.cpp` does everything in one file (this is the way you did the Day 4 assignments). Whereas the three files named a) `Rectangle.h` b) `Rectangle.cpp` and c) `RectangleTester.cpp` break the exercise up in a similar way to i) , ii) and iii) above in this question. *Each of the three files in this folder contains comments explaining how they are compiled.*
Use the sequence of command line instructions associated with the folder of Day 4 examples code in order to build your programs. Clearly document the significance of each compilation step in your **blue book.**

Hint: Try out the compilation steps for a), b) and c) first before you *refactor* question 2 from Day 4.
Note: Write the answer to these questions and other observations you may have, in your **blue book.**

Part B

(1) Develop a class hierarchy of Vehicles. Create two classes, `Taxi` and `Truck` that inherit publicly from class `Vehicle`. **Hint:** Draw out a UML inheritance diagram to represent this exercise before you start to code.

i) A `Taxi` should contain a data member (attribute) that indicates whether or not it is carrying passengers.

ii) A `Truck` should contain a data member (attribute) that indicates whether or not it is carrying cargo.

iii) Add the necessary member functions to manipulate and access class data (i.e. getters and setters)

iv) Write a driver that prints a `Truck` object and a `Taxi` object to the screen (using an overloaded stream-insertion operator).

To complete this problem you should use the **`template code`** in the **Exercise2** directory as a starting point. Comments in each of the file will guide you through the functionality and definitions that need to be added.
The **`driver.cpp`** file contains a main function that provides a partial implementation of the driver code mentioned above. The result of running this code should look something like the following:

```
Vehicle
Number of doors: 2
Number of cylinders: 6
Transmission type: 3
Colour: blue
Fuel level: 14.6
Taxi
Number of doors: 4
Number of cylinders: 6
Transmission type: 5
Colour: yellow
Fuel level: 3.3
The taxi currently has no customers
Truck
Number of doors: 2
Number of cylinders: 16
Transmission type: 8
Colour: black
Fuel level: 7.54

The truck is currently carrying cargo.
New colour is: red
New fuel level is: 4.9
```

(2) Write a `Racecar` class that inherits publicly from a class `Car`. Class `Car` represents a car using attributes for (i) its maximum speed, (ii) the number of engine valves, (iii) its colour and (iv) its name.
A `Racecar` Class is distinguished by attributes for (i) its gearbox (the number of gears it has), (ii) its sponsor and (iii) the presence of a parachute.

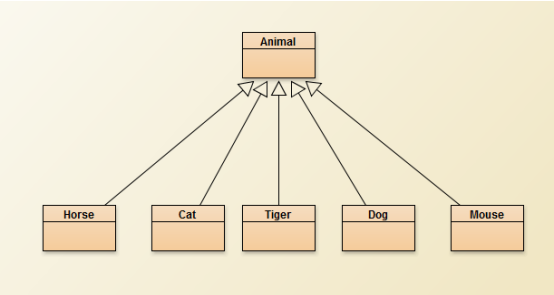
To complete this problem you should use the **`template code`** in the **Exercise1** directory as a starting point. Comments in each of the files will guide you through the functionality and definitions that need to be added.

The `driver.cpp` file contains a main function that you can use to test your solution.
The result of running this code should look something like the following:

chevy:
```
Car: Chevrolette is black and has a 4-valve engine. MAX SPEED = 95 mph.
```

f1:
```
Car: Ferrari is red and has a 40-valve engine. MAX SPEED = 220 mph.
Ferrari also has 7 gears and is sponsored by Bug2Bug.
Ferrari has used its parachute.
```

(3) Create a series of classes with an inheritance hierarchy for animals and which will look like the following diagram below when it is completed (UML Unified Modelling Language). **You may write in one single file, all the code for this class hierarchy as well as the main function that instantiates object of this hierarchy.**



1) Create a class called `Animal`.
   a. Add a string attribute `animalCategory`;
   b. Add a string attribute `animalSound`;
   c. Add a default constructor which in its body assigns the attributes `animalCategory` and animalSound to the value "undefined".
   d. Add a second constructor which takes two string parameters and sets the `animalCategory` and `animalSound` to these parameters.
   e. Add getter and a setter for `animalCategory` and `animalSound`
   f. Add a virtual function to the `Animal` class named `describeIdentity`, which does not return any value and does not take any parameter, but prints out the message "`I am a simple Animal`".
2) Create separate classes for each of the classes, `Horse`, `Mouse`, `Dog`, `Cat` and `Tiger`.
   Note: These will not have attributes defined in the class definition as they will inherit the attributes from the Animal class!
   a. Make all the other classes inherit from the Animal class and then take note of the diagram.
   b. In each of these classes create a default constructor which uses the base class (superclass) default constructor
   c. In each of these classes create a second constructor which takes two String parameters and uses the base class (super class) constructor passing it these values.
   d. Redefine the virtual function `describeIdentity` in each of the derived classes so that each of them prints out the identity of the derived animal. E.g. A Dog object would have a redefinition of the `describeIdentity` that would print out "I am a dog".
3) Open the `Dog` Class again
   a. In the `Dog` class create an int attribute called `numDogBiscuitsEaten`.
   b. Add code to both of the `Dog` constructors that sets `numDogBiscuitsEaten` to 0.
   c. Add a getter function for the attribute `numDogBiscuitsEaten`
   d. Add a function called `eatDogBiscuit( )` which will have no parameters and will not return anything but will add one to `numDogBiscuitsEaten`.

Note: There is no need to provide a setter for `numDogBiscuitsEaten`, not every attribute will have getters and setters in OO.

4) Create a tester class `GeneralTester` which is a simple program with only a main function.
   a. In this main function create the following objects: (instances of the classes) a `Cat` object, a `Dog` object and two `Tiger` objects using the appropriate class.
   b. Then following part (a), make calls on the getter functions for the `Cat` object and the 1st `Tiger` object to retrieve the sound these animals make.
   c. For the 2nd `Tiger` object use the getter functions to retrieve the sound and the category.
   d. Change the sound that the 2nd tiger makes to "purr" using the setter function and then use the getter function to verify that the sounds has been changed.
   e. Call the eatDogBiscuit( ) for your Dog Object.
   f. Can you call the `eatDogBiscuit( )` function on a `Cat`? Why?
   g. Call the getters for `Dog` Object created in part (a) to retrieve the `animalCategory` `animalSound` and `numDogBiscuitsEaten`.
   h. Create and array of Animal 5 pointers, then create 5 different animal objects and allow successive elements in the array of pointers, point to a different animal object. Start this exercise by create a single Animal pointer and letting it store the address (point to) a single Dog object. **(see the code in the folder on moodle "Module 10 corrected code")**
   i. Next use a loop to cycle through the array of Animal pointers and use the appropriate operators so that you call the `describeIdentity` function on each of the objects that they point to. In your **BlueBook**, describe what happens when you run this code.

Part C

(1) Draw the UML class diagrams or class icons for each of the classes from questions 1 to 3 in Part B.

Check out the following website http://www.codeproject.com/Articles/618/OOP-and-UML to see what a UML class diagram looks like. Don't worry if you do not get this completely correct, it is important that you try.
Also try : https://www.safaribooksonline.com/library/view/learning-uml-20/0596009828/pr02s06.html
Both of these links are on moodle.

Note: Write the answer to these questions and any other observations you may have, into your **bluebook**.

Appendix A Question 2 from Day 4 Lab, placed here only for convenience. See Part A question 1.

(2) Write a program, which contains your own class to represent a dog and name it the class **Dog**.The **Dog** class should have class variables (attributes) as follows:

```
string breed; // the breed of the dog e.g. terrier.
bool isDangerous; // to indicate whether or not the dog is dangerous.
int age;     // the age of the dog
```

The class should have one class constructor

1) A constructor which takes parameters to give values to each of the class attributes.
2) The class should have **getter** and **setter** functions for each of the class attributes
   i) Getters are: `string getBreed( ); bool getTemperament( ); in getAge( );`
   ii) Setters are: `void setBreed(string a); void setTemperament(bool a); void setAge(int x);`
3) The class should have a separate function named **bark**. This function should simply print out the sentence "Woof! The dog has barked" when the function bark is called. `void bark();`
4) Use this new class definition in your program to create two different dogs (objects) and get both of your dogs to bark for you. (choose the age, breed and temperament for each dog)
5) Use the getter functions to find out if your dogs are dangerous, what breed they are and how old they are.
6) Print the information obtained in part (5) to the screen with a meaningful message to the user.

Note: The file containing the Dog class definition and a main function, which uses this class will be called **MyDog.cpp**

Appendix B Basic principles and concepts used in object oriented style programming.

Class Scope	This is when a variable (attribute) is declared just inside the first line of a class after the first brace {. The scope of this variable covers the whole class which includes any functions inside the class.
Data encapsulation	This OO concept is achieved by **bundling** the functions (behaviour) and the variable (attributes) of a class representation together in the class.
private	This is an access modifier (a C++ keyword) which is put in front of your class variables declarations (variables declared just inside your class). Only functions defined inside a class definition can have access to or are able to change the value of variable (attributes) declared with the access modifier **private. The private access modifier can also be used for class functions in order to restrict "who" has access these functions.** .
Information hiding	Information hiding is the principle OO concept of **only allowing the functions defined within a class to have access to or change the values of the attributes of the class**. Achieved by declaring the class attribute variables to using the access modifier **private** in front of their declarations.
Instance / object	This is comes about when you use a class definition or specifically a call to the constructor

	in a class definition to create a concrete object in memory which has its own attribute data and its own functions encapsulated into a bundle (the object). **Remember** that you can use the class definition repeatedly to create many objects in the same way as you can use a template to create many cookies. **("Cutter is to cookie as class is to object"). Remember** also that you have to have links to these objects using object reference variables.
public	This is an access modifier (a C++ keyword) which is put in front of your functions defined within your class. This allows other code to call these functions using an instance of an object and the dot "." operator followed by the function name. (Calling functions in this way allows us to indirectly change or retrieve the values of private class variables). You allow the public to have a way of accessing your attributes; you provide a public interface (set of publicly available functions) to access your attributes (class variables).
Inheritance	When a class definition reuses functions and attributes from another class definition by inheriting that classes definition. C++ allows the programmer to **reuse** class definitions and extend the functionality in them by allowing one class (subclass) to **inherit** from another class (parent or superclass). This is a very important ability because it allows to programmer to develop and extend software which is dependent on previously proven super classes.
Overriding functions (functions in Java)	When a function is overridden in a subclass, and that function is invoked (called) on an instance (object) of the subclass, **it is the overridden version of the function that is executed**. The subclass function must have the same name and parameter types as the superclass function it overrides. The superclass function is still there but it is hidden or overridden by the subclass function.
Inheritance	This is when a class definition reuses functions and attributes from another class definition by inheriting that classes definition. C++ allows the programmer to **reuse** class definitions and extend the functionality in them by allowing one class (**child class**, **subclass**, **derived class**) to **inherit** from another class (**parent class**, **super class**, **base class**). This is a very important ability because it allows to programmer to develop and extend software which is dependent on previously proven super classes. Remember, a class is a template or blue print for creating objects, inheritance allows you to combine the parent and child templates to build an extended template, which can then be used to create enhanced objects.
Polymorphism	**Introduction: Poly (many) morphisms (forms). "many forms (identities) that an object can be, at the same time"** For now, note that any object that is created using an inheritance template can be categorised as both the subclass and the parent class. For example, if we wrote a `Rectangle` class that inherited from a general `Shape` class, and we created an instance of the `Rectangle` class, then this newly created object can be described as an object of type `Rectangle` but also an object of type `Shape`. The object is described as a polymorphic object. Also, the ability to overload functions is an important support mechanism for polymorphism, more to follow. According to Schildt, page 409, (module 10.9), "The foundation upon which C++ builds its support for polymorphism consists of **inheritance** and **base class pointers**. The specific feature that actually implements polymorphism is the **virtual function**".