Week 10 – Session 1: Inheritance & Polymorphism

In this laboratory session you are going to explore polymorphism and inheritance in OOP programming with C++.

Challenge 1: Student Record System

Download the lab solution from week 8 – Advanced class part I from GitHub. Please use this link:

https://github.com/eiramlan/StudentModule/tree/master/COM326W8L2/StudentModule

You can clone this solution (1) Manually, using download zip and click .sln on your local machine (2) Using VStudio clone new project option OR (3) Using Gitkraken.

Task 1 Person base class - duration 15 minutes

Add an abstract base class call Person to the project. The person class should contain the following:

Data members

- `Name (std::string name_)`
- `Email (std::string email_)`

Functions

- Default & custom constructors
- Getters & setters for all data members
- `virtual std::string ToString()` that returns a string containing the name and email of the person

Task 2 Refactor the student class - duration 20 minutes

Refactor the student class so that:

- It is derived from the Person class
- No longer contains a name data member
- Overrides the `ToString()` function

Task 3 task Add a lecturer class - duration 20 minutes

Add a lecturer class to the system. It should also derive from the person class and contain the following:

Data members

- `StaffID (std::string staffId_)`
- `subjectArea (std::string subjectArea)`
- `Vector of modules that they teach (vector <Module> modules_)`

Functions

- Default & custom constructors
- Getters & setters for all data members
- `virtual std::string ToString()` that returns a string containing the name and email of the lecturer, their subject area and the modules they teach.

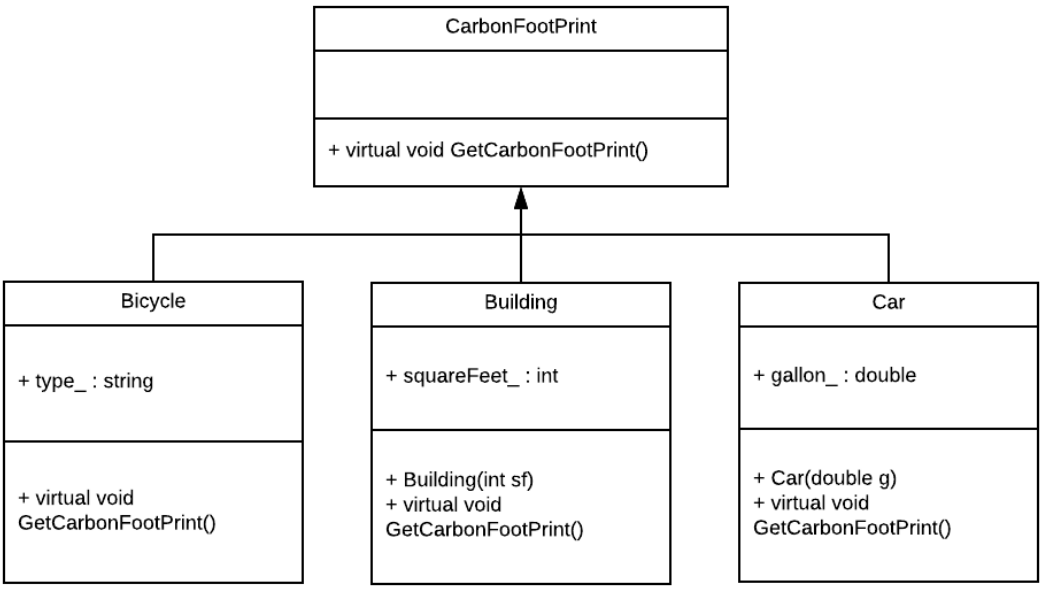
Task 4 task Polymorphic interaction - duration 10 minutes

Instantiate a number of student and lecturer objects. Call `ToString()` on the student and lecturer objects using a pointer of type Person.

Challenge 2: Carbon footprint

Using an abstract class with only pure virtual functions, you can specify similar behaviours for possibly disparate classes.

Create three small classes unrelated by inheritance – classes Building, Car and Bicycle. Give each class some unique appropriate attributes and behaviours that it does not have in common with the other classes. Use the following UML class diagram:



Write an abstract class `CarbonFootPrint()` with only a pure `virtual getCarbonFootPrint()` member function.

Have each of your classes inherit from that abstract class and implement the `GetCarbonFootPrint()` member function to calculate an appropriate carbon footprint for that class. Simple calculation for carbon footprint for each class:

- Bicycle: 0 – no consideration for the factory that actually make the bicycle

- Building: Multiply the square footage by 50 for the wood frame, by 20 for the basement, by 47 for the concrete, and 17 for the steel
- Car: One gallon of gas yields 20 pounds of CO2

In your main.cpp create objects of each of the three classes, place pointers to those objects in a vector of `CarbonFootPrint` pointers, then iterate through the vector, polymorphically invoking each object's `GetCarbonFootPrint()` member function.

For each object, print some identifying information and the object's carbon footprint.