Part E - Polymorphism

Inheritance and Virtual Functions

Workshop 8

In this workshop, you are to create an abstract base class out of an interface class and inherit it into two derived classes.

LEARNING OUTCOMES

Upon successful completion of this workshop, you will have demonstrated the abilities to

- Inherit from a class
- Define a virtual base class
- Call derived class functions through a virtual base class call, demonstrating inclusion polymorphism
- Reflect on the concepts learned in this workshop

SUBMISSION POLICY

The “in-lab” section is to be completed **during your assigned lab section**. It is to be completed and submitted by the end of the workshop. If you do not attend the workshop, you can submit the “in-lab” section along with your “at-home” section (a 20% late deduction will be assessed). The “at-home” portion of the lab is **due the day before your next scheduled workshop**

All your work (all the files you create or modify) must contain your name, Seneca email and student number.

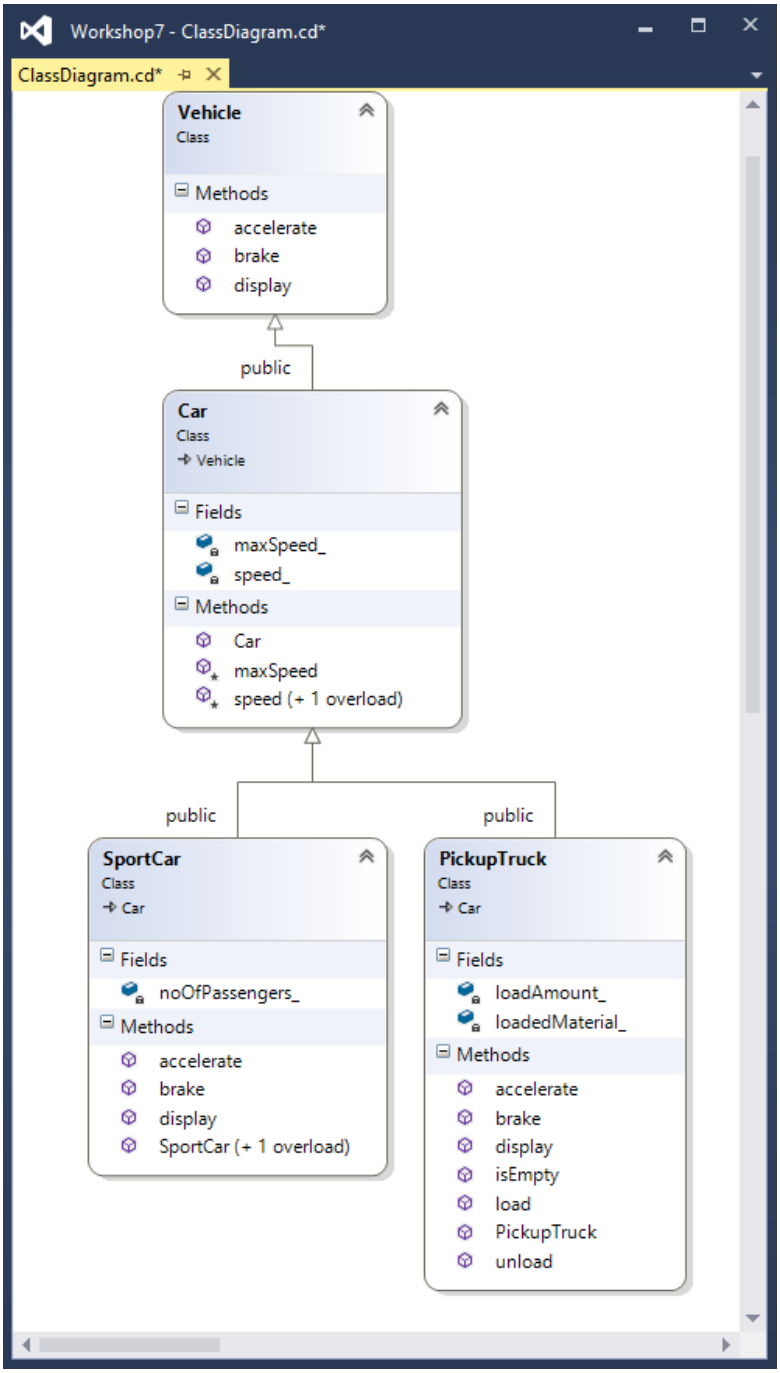
You are responsible for regularly backup your work.

Both submissions assume your work is included in the **sict** namespace.

IN LAB:

Download or clone workshop 8 from https://github.com/Seneca-244200/OOP-Workshop8/EF

For the in lab section we are going to create an ***interface*** (an **abstract base** class with **only pure virtual methods** in it) called **Vehicle**. We then create another abstract class called **Car** which inherits from **Vehicle** but does not implement any of its pure virtual methods. Finally we create two concrete subclasses of Car; **SportCar** and **PickupTruck** that will encapsulate the capabilities of a sport car and a pickup truck and implement all the pure virtual methods in the **Vehicle** class.



VEHICLE CLASS:

In Vehicle.h:
Add the following three member functions to the Vehicle class as pure virtual methods. By adding a virtual to the type and "= 0" to the end of prototype:
virtual type function(type) = 0;

```
void accelerate();
void brake();
std::ostream& display(std::ostream& ostr) const;
```

Note that interfaces do not have "cpp" files since all their methods are pure virtual.

CAR CLASS:

In Car.h and Car.cpp;
Complete the code of the class named **`Car`** that holds general information about a car.

Car is inherited from Vehicle.

Private member variables (attributes):

```
int speed_;
int maxSpeed_;
```

Protected member functions:

```
void speed(int value);
```
Sets the speed_ attribute to the incoming value.
If the value is greater than maxSpeed_ attribute or less than 0, then values are corrected to maxSpeed_ and 0 respectively.

```
int maxSpeed()const;
```
Returns the maxSpeed_ attribute.

Public member function and constructor;

```
int speed() const;
```
Returns the speed_ attribute.

```
Car constructor:
```

Receives one argument to set the maxSpeed attribute. If this argument is not provided, it will set the maximum Speed to 100. It also sets the speed attribute to 0.

SPORTCAR CLASS:

In SportCar.h and SportCar.cpp;
Complete the code of the class named **`SportCar`** to inherit a Car and fully implement a sport car.

Private member variables (attributes):

```
int noOfPassengers_;
```

Public member functions and constructors;

```
No argument Constructor:
```
Sets the number of passengers to 1.

```
Two integer argument Constructor:
```
Receives maximum speed and number of passengers; it passes the maximum speed value to its Base's (Car) constructor and sets the number of passengers to the incoming value.

```
// implementations of Vehicle's pure virtual methods
void accelerate();
```
Adds 40 kilometers to the speed.

```
void brake();
```
Reduces the speed by 10 kilometers

```
std::ostream& display(std::ostream& ostr) const;
```
Using the ostr (cout refrence) prints one of following two messages:
If the speed is greater than zero:

`This sport car is carrying Pnum passengers and is traveling at a speed of Snum km/h.`

If the speed is zero:

`This sport car is carrying Pnum passengers and is parked.`

Where Pnum is number of passengers and Snum is the speed.

PICKUPTRUCK CLASS:

In PickupTruck.h and PickupTruck.cpp;
Complete the code of the class named **`PickupTruck`** to inherit a Car and fully implement a pickup truck.

Private member variables (attributes):

```
int loadAmount_;
```

The load amount in kilograms

```
char loadedMaterial_[31];
```

The loaded material name.

No argument constructor and Public member functions;

```
PickupTruck();
```

Sets the loadAmount_ attribute to zero and the loadecMaterial_ to an empty C-style string.

```
Void load(const char* loadedMaterial, int loadAmount);
```

Sets the two corresponding attributes to the incoming values through the argument list.

```
void unload();
```

Sets the loadAmmount_ attribute to zero.

```
bool isEmpty()const;
```

Returns true if the loadAmmount_ attribute is zero.

```
// implementations of Vehicle's pure virtual methods
void accelerate();
```

Adds 20 kilometers to the speed.

```
void brake();
```

Reduces the speed by 5 kilometers

```
std::ostream& display(std::ostream& ostr) const;
```

Using the ostr (cout refrence) prints one of following two messages:
If the truck is not carrying any load, (isEmpty() is true)

This pickup truck is not carrying any load

Otherwise:

This pickup truck is carrying Lnum kgs of Lname

And Then:
If the speed is greater than zero:

, traveling at the speed of Snum km/h.

If the speed is zero:

and is parked.

Where Lnum is load amount , Lname is loaded material and Snum is the speed.

W8_IN_LAB.CPP:

```
#include <iostream>
#include "SportCar.h"
#include "PickupTruck.h"
using namespace std;
using namespace sict;

int main()
{
  SportCar Tesla(240, 2);
  PickupTruck Ford;
  Tesla.display(cout) << endl;
  Ford.display(cout) << endl;
  Ford.load("Bricks", 3500);
  drive(Tesla);
  drive(Ford);
  stop(Tesla);
  stop(Ford);
  Ford.unload();
  Tesla.display(cout) << endl;
  Ford.display(cout) << endl;
  return 0;

}
```

Add two stand-alone, helper functions to this file called drive and stop that accept a reference to a Car as an argument:

drive:
Accelerates and then brakes the Car. Then it will display the car.
stop:
Keeps breaking until the speed of the car is zero. Then it will display the car.

Compile your code four classes with w8_in_lab.cpp and the output should be exactly as follows.

```
This sport car is carrying 2 passengers and is parked.
This pickup truck is not carrying any load and is parked.
This sport car is carrying 2 passengers and is traveling at a speed of 30 km/h.
This pickup truck is carrying 3500 kgs of Bricks, traveling at the speed of 15 km/h.
This sport car is carrying 2 passengers and is parked.
This pickup truck is carrying 3500 kgs of Bricks and is parked.
This sport car is carrying 2 passengers and is parked.
This pickup truck is not carrying any load and is parked.
```

Note how in drive and stop, the latest versions of accelerate, brake and display are called.

IN-LAB SUBMISSION (40%)

To submit the in-lab section demonstrate execution of your program with the exact output as example above.

If not on matrix already, upload **the four classes** and **w8_in_lab.cpp** to your matrix account. Compile and run your code and make sure everything works properly.

Then run the following script from your account:

```
Sections SEF:
~eden.burton/submit w8_in_lab <ENTER>
```

and follow the instructions.

AT-HOME

OVERLOADING OPERATORS FOR ABSTRACT BASE CLASSES CREATING DRIVER CLASS TO USE A CAR (VIRTUALS)

To begin the at home section, first copy all your in_lab files except w8_in_lab.cpp to the at home project directory.

Then overload the operator<< for the Car class, so the Car class cat be printed with cout.

In the implementation of operator overload for ostream, Call and return the display method inherited from the Vehicle, passing the ostream argument through it.

Create a Driver class to drive a Car.

In Driver.h and Driver.cpp Create a class called Driver with following specs:

Private Member Variables (Attributes):

```
char name_[31];
     C-style character string to hold the drivers name.
Car& car_;
     A reference to a Car that driver is going to drive.
```

Public Constructor and Member Functions (Methods):

Driver's constructor receives two arguments; a c-style character string to set the name of the driver to, and a reference to a Car to INITIALIZE the car_ reference attribute with.

```
Driver(const char* name, Car& cRef);
```

Note that car_ must be initialized with cRef and not "set to". In fact this is the only possible way and any other attempt to set the car_ reference to cRef, will cause compile error.

```
void drive();
```

Accelerates, brakes and then shows the Status of the driver (showStatus());

```
void stop();
```

Keeps braking until car_ comes to a complete stop. (speed() becomes zero).

```
void showStatus();
```

Frist displays this massage:

Dname is driving this car.<newline>

then it prints the car_ attribute using the overloaded operator<< and goes to new line.

Where Dname is the name of the Driver.

Test your class the w8_at_home.cpp and make sure it works.

Your Driver class together with w8_at_home.cpp must produce the following output:

```
#include <iostream>
#include "SportCar.h"
#include "PickupTruck.h"
#include "Driver.h"
using namespace std;
using namespace sict;
int main()
{
  SportCar Tesla(240, 2);
  PickupTruck Ford;
  Driver J("John", Tesla);
  Driver K("Kim", Ford);
  cout << Tesla << endl;
  cout << Ford << endl;
  Ford.load("Bricks", 3500);
  J.drive();
  K.drive();
  J.stop();
  K.stop();
  cout << Tesla << endl;
  cout << Ford << endl;
  return 0;
```

```
}
```

```
This sport car is carrying 2 passengers and is parked.
This pickup truck is not carrying any load and is parked.
John is driving this car.
This sport car is carrying 2 passengers and is traveling at a speed of 30 km/h.
Kim is driving this car.
This pickup truck is carrying 3500 kgs of Bricks, traveling at the speed of 15 km/h.
John is driving this car.
This sport car is carrying 2 passengers and is parked.
Kim is driving this car.
This pickup truck is carrying 3500 kgs of Bricks and is parked.
This sport car is carrying 2 passengers and is parked.
This pickup truck is carrying 3500 kgs of Bricks and is parked.
```

AT-HOME SUBMISSION (30%) AND REFLECTION (30%)

Please provide brief answers to the following questions in a text file named **`reflect.txt.`** Use examples from the workshop in your answers.

1. Explain what virtual functions are.
2. What is the difference between virtual methods and pure virtual methods?
3. What are abstract classes and why were they useful in the application we created for this workshop?
4. What are interfaces and why were they useful in the application we created for this workshop?
5. Is an abstract class necessarily an interface? Is an interface necessarily an abstract class? Explain your answers.

SUBMISSION

To test and demonstrate execution of your program use w8_at_home.cpp.

If not on matrix already, upload **all the classes** and **w8_at_home.cpp** to your matrix account. Compile and run your code and make sure everything works properly.

Then run the following script from your account:

Sections SEF:
~eden.burton/submit w8_at_home <ENTER>

and follow the instructions.