CARS

Inheritance and Virtual Functions

In this project, 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

You many continue to work in your groups, but submit individually.

You are responsible to regularly backup your work.

MS VISUAL STUDIO

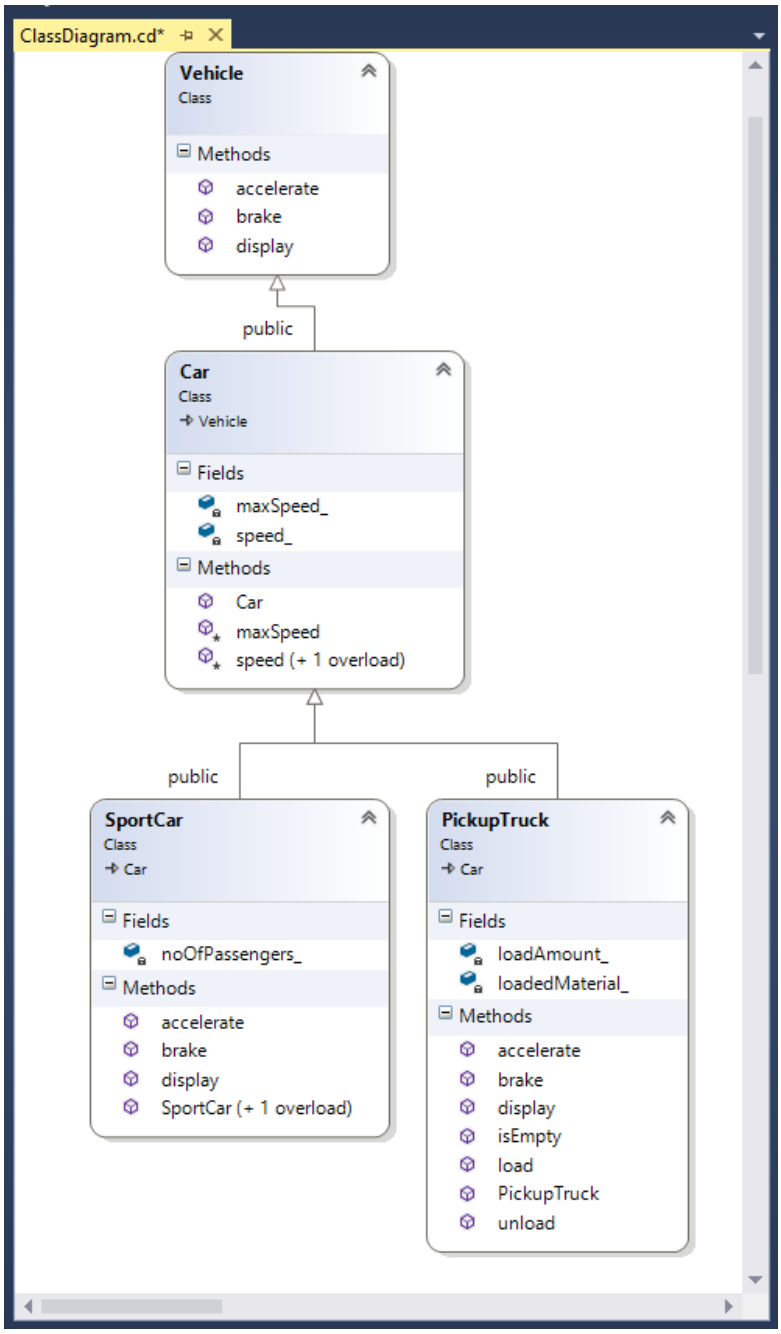
Watch the following video on how to fix the issues with strncpy_s
https://youtu.be/E6SUKCeL9Ic

PART 1:

For this section we are going to create an **interface** (an **abstract base** class with **only pure virtual methods** in it) called **Vehicle**.

Then we will ***inherit*** a class out of **Vehicle** called **Car**. A **Car** has general capabilities of a car but does not implement any of the **Vehicle's** pure virtual methods, therefore remaining abstract.

Finally, we will inherit two classed out of **Car**; **SportCar** and **PickupTruck** that will encapsulate the capabilities of a sport car and a pickup truck and also implement all the pure virtual methods of the **Vehicle** class.



You will see "//Complete Code" throughout the starter files. This is the code that you are responsible to complete

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 abstract base classes 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;

`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.

`int speed() const;`

Returns the speed_ attribute.

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]; // or  [MAX_MATERIAL + 1]
```

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.

PART 2 (20%)

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

Overload the operator<< for the Car class, so the Car class can 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.

DRIVER CLASS:

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) and then shows the Status of the driver (showStatus();).

```
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 main.cpp and make sure it works. It must produce the following output:

```
#include <iostream>
#include "SportCar.h"
#include "PickupTruck.h"
#include "Driver.h"

using namespace std;
using namespace cs;

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;
   cout << "Tom Marazzo, SN 123-456-789" << 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.
Tom Marazzo, SN 123-456-789
```

SUBMISSION

Create a Folder Called **CARS FINAL_FULL NAME_STUDENT NUMBER.ZIP** and upload to Blackboard or D2L.

1. **Include a screenshot of the working main.cpp output in your .zip file**. The LAST LINE OF YOUR OUTPUT must be your FULL name (***First and Last***) and student number.

Microsoft Visual Studio Debug Console

```
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.
Tom Marazzo, SN 123-456-789

G:\C++\Term Project\CarProject\CarProject\Debug\CarProject.exe (process 12108) exite
To automatically close the console when debugging stops, enable Tools->Options->Debu
le when debugging stops.
Press any key to close this window . . .
```

You many continue to work in your groups, <u>but submit individually</u>.

The **FINAL PROJECT** is **due on FRIDAY, DECEMBER 6, 2019, before 11:59:59pm.**

10% Late penalty, per day for each CALANDAR day (weekends count as a penalty day!)

This Project will serve as part of your final exam preparation.