Part E - Polymorphism

Inheritance and Virtual Functions

Workshop 9

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"* portion 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"* portion 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.

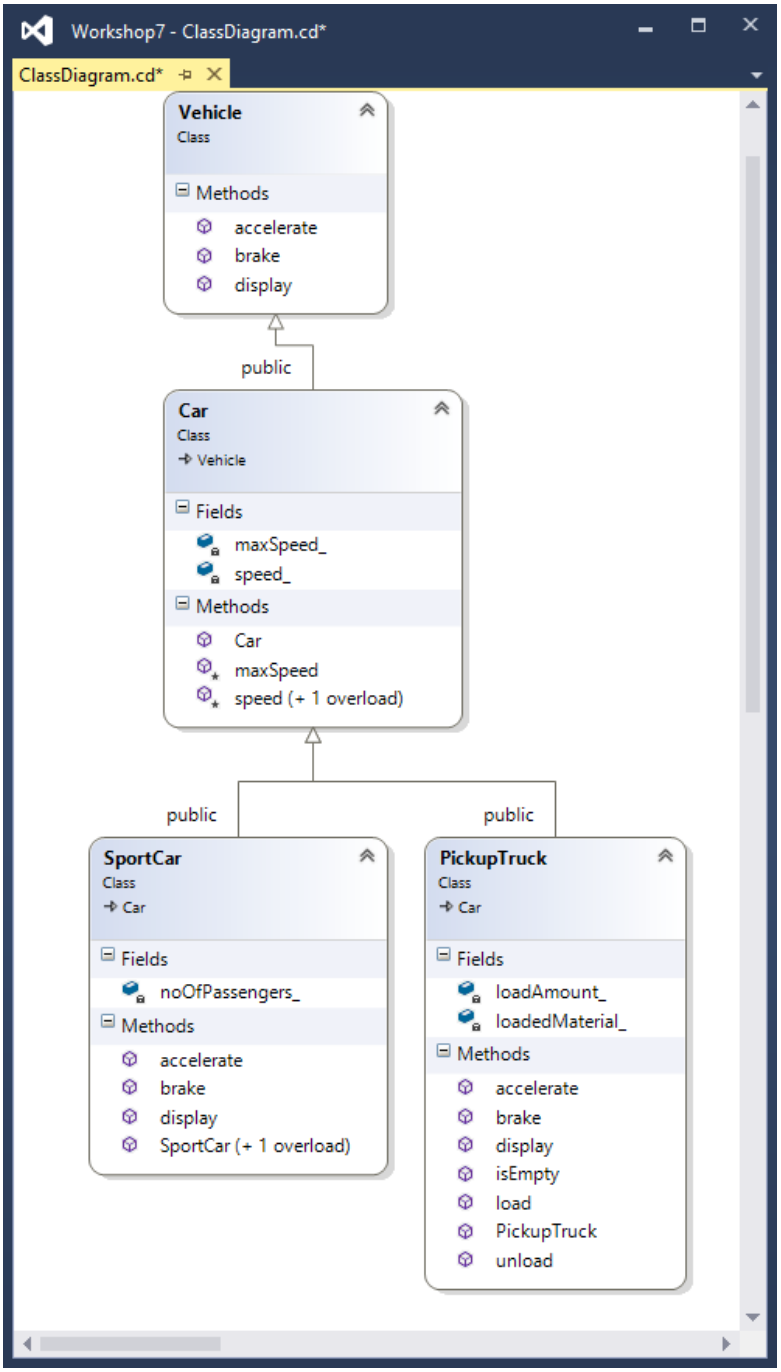
You are responsible to regularly backup your work.

In-Lab:

For the *in-lab* portion, 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 (see the class diagram below).



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 functionName(...) = 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 members:

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

Protected members:

`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;`

- Getter that returns the `maxSpeed_` attribute.

Public members:

`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:

Complete the code for the class named `SportCar` to inherit from class `Car` and fully implement a sport car (in `SportCar.h` and `SportCar.cpp`).

Private members:

```
int noOfPassengers_;
```

Public members:

`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 class (`Car`) constructor and sets the number of passengers to the incoming value.

```
void accelerate();
```

- Implementation of `Vehicle`'s pure virtual method.
- Adds 40 kilometers to the speed.

```
void brake();
```

- Reduces the speed by 10 kilometers

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

- Prints to `ostr` one of the following messages: "`This sport car is carrying Pnum passengers and is traveling at a speed of Snum km/h.`" if the speed is greater than zero OR "`This sport car is carrying Pnum passengers and is parked.`" if the speed is zero (where `Pnum` is number of passengers and `Snum` is the speed).

PICKUPTRUCK CLASS:

Complete the code of the class named `PickupTruk` (in files `PickupTruck.h` and `PickupTruck.cpp`) to inherit a `Car` and fully implement a pickup truck.

Private members:

```
int loadAmount_;
```

- The load amount in kilograms

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

- The loaded material name.

Public members:

```
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.

```
void accelerate();
```

- Adds 20 kilometers to the speed.

```
void brake();
```

- Reduces the speed by 5 kilometers

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

- If the truck is not carrying any load, (`isEmpty()` is true), it prints to `ostr` "`This pickup truck is not carrying any load`", otherwise it prints "`This pickup truck is carrying` Lnum `kgs of` Lname". Then, if the speed is greater than zero it prints "`, traveling at the speed of` Snum `km/h.`"; if the speed is zero, it prints "`and is parked.`" (Where Lnum is load amount, Lname is loaded material and Snum is the speed.)

W9_IN_LAB.CPP:

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.

```
#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;
}
```

```
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.
```

Compile the `four classes` with `w9_in_lab.cpp` and run the resulting program. The output should be exactly as above.

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

IN-LAB SUBMISSION (70%)

To submit the *in-lab* section, demonstrate execution of your program with the exact output as example above. Upload `the four classes` and `w9_in_lab.cpp` to your matrix account. Compile and run your code and make sure everything works properly. To submit, run the following script from your account (and follow the instructions):

~profname.proflastname/submit w9_in_lab <ENTER>

AT HOME(30%)

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

To begin the *at-home* section, copy all your *in-lab* files, except for `w9_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 `ostream`. 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 representing a driver of a car (in `Driver.h` and `Driver.cpp`). The `Driver` class should have the following specs:

Private members:

```
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 members:

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

- Two integer argument constructor: 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.
- Note that `car_` must be initialized with `theCar` and not "set to". In fact this is the only possible way and any other attempt to set the `car_` reference to `theCar`, 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()` returns `0`).

```
void showStatus();
```

- Frist, displays the massage "Dname is driving this car.<newline>" and 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 using the w9_at_home.cpp and make sure it works. Your Driver class together with w9_at_home.cpp must produce the output below:

```
#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 john("John", tesla);
    Driver kim("Kim", ford);
    cout << tesla << endl;
    cout << ford << endl;

    ford.load("Bricks", 3500);
    john.drive();
    kim.drive();
    john.stop();
    kim.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.
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.
```

AT-HOME SUBMISSION (20%) AND REFLECTION (10%)

Please provide brief answers to the following questions in a text file named **reflect.txt**.

1. Explain what virtual functions are.
2. What is the difference between virtual and pure virtual?
3. What are abstract classes and interfaces?
   Use this workshop for you examples.

SUBMISSION

To test and demonstrate execution of your program use w9_at_home.cpp.

If not on matrix already, upload **all the classes** and **w9_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 (and follow the instructions):

```
~profname.proflastname/submit w9_at_home <ENTER>
```