Nicholas Ang

CS 202 - 1101

Project 6 Documentation

**Purpose:** The purpose of this project is to design classes and methods that correspond to the proj6.cpp test driver. The project is used to understand how inheritance works and how to overload and override class methods. The project also tests understanding of constructors and what values need to be instantiated to get the program to function properly. It also expands on prior knowledge such as pointers, arrays, static members, const qualifications, and other aspects. The project also requires aspects of polymorphism in C++ using virtual methods and pure virtual methods. The project also includes the use of CMake to create libraries and makefiles to easily compile the project.

**Design:** The project is organized into four folders and a make file. Within the src folder is the proj6.cpp main source file and another folder holding all my other source files. The include file holds all my header files. The build folder holds the executable along with the cmake folders and libraries created by cmake. The main source file proj6.cpp is a test driver for my classes in Car.cpp and Vehicle.cpp. The header file Vehicle.h holds my class declaration and methods. The Vehicle source file holds the implementation and defines the methods. It holds getters and setters for where the vehicle “is at”. It also holds the pure virtual method move which is implemented in the Car.cpp and Car.h files. The Vehicle class holds the virtual method serialize to print out the information of the class. The Car class also has the virtual method serialize and when it is called, prints out the specialized version of the information compared to the base class Vehicle serialize. The Vehicle source file overloads the operator = to be able to assign vehicles to each other. It also overloads the operator << to call the serialize method from the Car class and vehicle class to print out its specialized information depending on which class is called. The Vehicle class holds a default constructor, a parameterized constructor, a copy constructor, and a destructor. The constructors take in values based on the test driver and at the end of the program, all the constructors destruct. The Car.h header file holds the derived class declaration of Car. It uses inheritance to be able to use the aspects of class Vehicle in its methods. Car.cpp source file is similar to Vehicle.cpp but overrides some of its methods to function for cars. It has getters and setters for the throttle of the car. The Car source file overloads the operators = to be able to assign Car to each other. To print out it’s information, it has its own serialize function which is called by the operator overload in the Vehicle base class. The drive method assigns the car a new throttle and the move method is a virtual method and modifies the car information to “move” them to new coordinates. The Car class holds a default constructor, a parameterized constructor, a copy constructor, and a destructor. The constructors take in values based on the test driver and at the end of the program, all the constructors destruct.

**Problems/Challenges:** Some difficulties I had were with the serialize function not calling out the right information. I had realized that I was not overriding the function properly and that it was essentially overloaded instead. I also had a little trouble with const correctness as the input kept providing errors but I figured out which parts needed to be constant. I also had to change all my double methods to floats because the proj6 test driver used floats instead of doubles.

**Possible Changes:** If I had more time to change stuff, I would try to clean up the code and the program. I would also try to make the code better and simpler. I also would try to organize the folders more thoroughly and create multiple libraries instead of one library.

////////////////////////////////

///// Constructor Tests /////

////////////////////////////////

Testing Derived Default ctor

Vehicle: Default-ctor

Car: Default-ctor

Testing Derived Parametrized ctor

Vehicle: Parameterized-ctor

Car: Parameterized-ctor

Testing Derived Copy ctor

Vehicle: Parameterized-ctor

Car: Copy-ctor

Testing Derived Assignment operator

Car: Assignment

/////////////////////////////////

///// Polymorphism Tests /////

/////////////////////////////////

Testing VIRTUAL Move Function for DERIVED Class Objects

Car: DRIVE to destination with throttle @ 75

Testing Insertion operator<< Overload for BASE Class Objects

Car: Throttle: 0@ [39.54, 119.82, 4500]

///////////////////////////////////////////////////

///// Polymorphic Base Class Pointer Tests /////

///////////////////////////////////////////////////

Testing VIRTUAL Move Function on Base Class Pointers

Car: DRIVE to destination with throttle @ 75

Car: DRIVE to destination with throttle @ 75

Car: DRIVE to destination with throttle @ 75

Testing Insertion operator<< Overload for Base Class Pointers

Car: Throttle: 75@ [37.77, 122.42, 52]

Car: Throttle: 75@ [37.77, 122.42, 52]

Car: Throttle: 75@ [37.77, 122.42, 52]

////////////////////////////

///// Tests Done /////

////////////////////////////

Car: Dtor

Vehicle: Dtor

Car: Dtor

Vehicle: Dtor

Car: Dtor

Vehicle: Dtor

**Line By Line Analysis:** The first three lines are a prompt indicating that the constructor tests were starting. Lines 4, 5, and 6 use the default constructor to construct a default car and creates a vehicle with default parameters. The vehicle that is constructed is not an object as it holds a pure virtual method which causes the class to be abstract. Lines 7, 8, and 9 use the parameterized constructor to create a car with the given parameters and set the vehicle with the given coordinates. Lines 10, 11, and 12 use the copy constructor in class car and the parameterized constructor in class vehicle due to being given values of a car so vehicle holds the lla using the parameterized constructor. Lines 13 and 14 call the assignment operator of the class car to assign the values of a car to another car. Lines 15, 16, and 17 are a prompt indicating that the polymorphism tests are starting. Lines 18 and 19 call the move function that was declared as pure virtual in class Vehicle and implemented in class Car to set the new throttle and new coordinates of the car. Then, in lines 20 and 21, it calls the operator overload << in class Vehicle and uses the serialize function from class Car because serialize is virtual and adjusts according to function. This prints out the information of the car to the terminal. The next three lines to line 24 indicate that the polymorphic base class pointer tests are starting. Lines 25, 26, 27, and 28 call the move function that was declared as pure virtual in class Vehicle and implemented in class Car to set the new throttle and new coordinates of each car in the Vehicle pointer array. Then lines 29, 30, 31, and 32 call the serialize function from class Car and prints out the information of the car to the terminal. Lines 33, 34, and 35 are a prompt that indicates that the tests are done. The last 6 lines of the program call the destructors for each car and its associated vehicle.