UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science
Faculty of Engineering, Built Environment & IT
University of Pretoria

COS110 - Program Design: Introduction

Practical 6 Specifications

Release Date: 09-09-2024 at 06:00

Due Date: 13-09-2024 at 23:59

Total Marks: 135

Contents

1 General Instructions

- ***Read the entire assignment thoroughly before you start coding.***
- This assignment should be completed individually, no group effort is allowed.
- **To prevent plagiarism, every submission will be inspected with the help of dedicated software.**
- Be ready to upload your assignment well before the deadline, as **no extension will be granted.**
- If your code does not compile, you will be awarded a mark of 0. The output of your program will be primarily considered for marks, although internal structure may also be tested (eg. the presence/absence of certain functions or classes).
- Failure of your program to successfully exit will result in a mark of 0.
- Note that plagiarism is considered a very serious offence. Plagiarism will not be tolerated, and disciplinary action will be taken against offending students. Please refer to the University of Pretoria's plagiarism page at `https://portal.cs.up.ac.za/files/departmental-guide/`.
- Unless otherwise stated, the usage of c++11 or additional libraries outside of those indicated in the assignment, will **not** be allowed. Some of the appropriate files that you have submit will be overwritten during marking to ensure compliance to these requirements. **Please ensure you use c++98**
- All functions should be implemented in the corresponding `cpp` file. No inline implementation in the header file apart from the provided functions.
- The usage of ChatGPT and other AI-Related software is strictly forbidden and will be considered as plagiarism.
- Note, UML notation for variable and function signatures are used in this assignment.

2 Overview

Inheritance in Object-Oriented Programming is a concept where a new class, called a derived class, is created based on an existing class, known as the base class. The derived class automatically inherits the attributes and methods of the parent class, enabling code reusability and the creation of a structured class hierarchy. This enables developers to write less redundant code and improve the maintainability of the code base. In this practical you will implement a simple class hierarchy to improve your understanding of inheritance.

3 Background

In this practical you will simulate a simple car rental service that allow customers to book a fleet of vehicles. The fleet can composed out of cars, buses and/or trucks.

4 Your Task:

You are required to implement the following class diagram illustrated in Figure 1. Pay close attention to the function signatures as the `h` files will be overwritten, thus failure to comply with the UML, will result in a mark of 0.

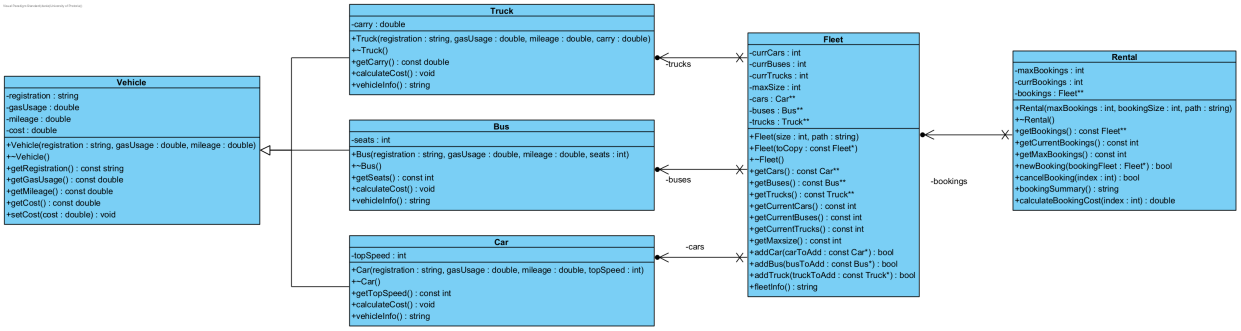

Figure 1: Class diagrams

Note, the importance of the arrows between the classes are not important for COS 110 and will be expanded on in COS 214.

4.1 Vehicle

This is the base class of the class hierarchy and has 3 derived classes.

4.1.1 Members

- - registration: string
  - This member variable represents the registration of the vehicle.
- - gasUsage: double
  - This member variable represents the gas usage (liters/100km) of the vehicle.
- - mileage: double
  - This member variable represents the total kilometers the vehicle has traveled.
- - cost: double
  - This member variable represents the cost to rent the vehicle.

4.1.2 Functions

- + Vehicle(registration : string, gasUsage : double, mileage : double)
  - This is the constructor for the Vehicle class.
  - It should set the member variables to the corresponding parameters.
  - Initialise the cost to -1.0.
- + ∼Vehicle()
  - This is the destructor for the Vehicle class.
  - There is no dynamic memory to deallocate, therefore it can be left blank.
- + getRegistration(): const string
  - This is an accessor method for the registration member variable.
  - This is a const function and does not return a const value.
  - It should return the registration member variable.
- + getGasUsage(): const double
  - This is an accessor method for the gas usage member variable.
  - This is a const function and does not return a const value.
  - It should return the gas usage member variable.
- + getMileage(): const double

  - This is an accessor method for the mileage member variable.
  - This is a const function and does not return a const value.
  - It should return the mileage member variable.
- + getCost (): const double
  - This is an accessor method for the cost member variable.
  - This is a const function and does not return a const value.
  - It should return the cost member variable.
- + setCost (cost : double): void
  - This method should set the cost member variable to the passed in parameter.

4.2 Car

The Car class is one of the Vehicle's derived classes. It inherits publicly from Vehicle. It has one additional member variable.

4.2.1 Members

- - topSpeed: int
  - This member variable represents the maximum speed the car can drive.

4.2.2 Functions

- + Car(registration : string, gasUsage : double, mileage : double, topSpeed: int)
  - This is the constructor for the Car class.
  - It should set the member variables to the corresponding parameters.
  - Remember to call the base class's constructor.
- + ∼Car()
  - This is the destructor for the Car class.
  - There is no dynamic memory to deallocate, therefore it can be left blank.
- + getTopSpeed() : const int
  - This is an accessor method for the top speed member variable.
  - This is a const function and does not return a const value.
  - It should return the top speed member variable.
- + calculateCost() : void

  - This method calculates the cost to rent the car based on the current mileage of the car and the total gas usage.
  - The following formula is used to calculate the cost based on the mileage: $costMileage = baseCost \times e^{-mileage \times decay}$.
  - The *decay* should be set to 0.015.
  - If $topSpeed > 150$ and $topSpeed <= 200$ the base cost should be 12000.
  - If $topSpeed <= 150$ the base cost should be 6000.
  - In all other cases the base cost should be 20000.
  - The following formula is used to calculate the cost based on the gas usage: $costGas = max(costMileage, baseCost) + \frac{multiplier}{gasUsage}$.
  - The *multiplier* should be set to 500.
  - Set the cost of the car to the result the *costGas* formula.

- + vehicleInfo() : string
  - This method return a string with all the info of the vehcile.
  - The string should be formatted as follows:

```
R: registration G: gasUsage M: mileage S: topSpeed
```

4.3 Bus

The Bus class is one of the Vehicle's derived classes. It inherits publicly from Vehicle. It has one additional member variable.

4.3.1 Members

- - seats: int
  - This member variable represents the number of seats in the bus.

4.3.2 Functions

- + Bus(registration : string, gasUsage : double, mileage : double, seats: int)
  - This is the constructor for the Bus class.
  - It should set the member variables to the corresponding parameters.
  - Remember to call the base class's constructor.

- + ~Bus()
  - This is the destructor for the Bus class.
  - There is no dynamic memory to deallocate, therefore it can be left blank.

- + getSeats() : const int
  - This is an accessor method for the seats member variable.
  - This is a const function and does not return a const value.
  - It should return the seats member variable.
- + calculateCost() : void
  - This method calculates the cost to rent the bus based on the current mileage of the bus and the total gas usage.
  - The following formula is used to calculate the cost based on the mileage: $costMileage = baseCost \times e^{-mileage \times decay}$.
  - The *decay* should be set to 0.025.
  - If $seats > 26$ and $seats <= 40$ the base cost should be 30000.
  - If $seats <= 26$ the base cost should be 20000.
  - In all other cases the base cost should be 40000.
  - The following formula is used to calculate the cost based on the gas usage: $costGas = max(costMileage, baseCost) + \frac{multiplier}{gasUsage}$.
  - The *multiplier* should be set to 1000.
  - Set the cost of the bus to the result the *costGas* formula.
- + vehicleInfo() : string
  - This method return a string with all the info of the vehcile.
  - The string should be formatted as follows:

```
R: registration G: gasUsage M: mileage S: seats
```

4.4 Truck

The Truck class is one of the Vehicle's derived classes. It inherits publicly from Vehicle. It has one additional member variable.

4.4.1 Members

- - carry: double
  - This member variable represents the weight the truck can carry in tons.

4.4.2 Functions

- + Truck(registration : string, gasUsage : double, mileage : double, seats: int)
  - This is the constructor for the Truck class.
  - It should set the member variables to the corresponding parameters.
  - Remember to call the base class's constructor.
- + ~Truck()
  - This is the destructor for the Truck class.
  - There is no dynamic memory to deallocate, therefore it can be left blank.
- + getCarry() : const double
  - This is an accessor method for the carry member variable.
  - This is a const function and does not return a const value.
  - It should return the carry member variable.
- + calculateCost() : void
  - This method calculates the cost to rent the truck based on the current mileage of the truck and the total gas usage.
  - The following formula is used to calculate the cost based on the mileage: $costMileage = baseCost \times e^{-mileage \times decay}$.
  - The *decay* should be set to 0.010.
  - If $carry > 2$ and $carry <= 5$ the base cost should be 52000.
  - If $carry <= 2$ the base cost should be 35000.
  - In all other cases the base cost should be 78000.
  - The following formula is used to calculate the cost based on the gas usage: $costGas = max(costMileage, baseCost) + \frac{multiplier}{gasUsage}$.
  - The *multiplier* should be set to 10000.
  - Set the cost of the truck to the result the *costGas* formula.
- + vehicleInfo() : string
  - This method return a string with all the info of the vehcile.
  - The string should be formatted as follows:

```
R: registration G: gasUsage M: mileage C: carry
```

4.5 Fleet

The Fleet class represent a fleet of vehicles that can be rented. To populate the vehicles in the fleet you will get input from a textfile of the following format:

```
Type#Registration#GasUsage#Mileage#Derived
```

Here is an example textfile:

```
Truck#PDC702#13.57#188838#6.45
Car#WPG076#21.39#42382#299
Truck#XYZ186#7.41#83905#2.58
Bus#CSB770#19.42#58982#31
Truck#EUB742#24.0#44491#13.04
Bus#TZF253#7.38#119930#70
Car#EVQ294#21.42#183372#251
Bus#UZO804#14.7#66463#67
Car#XHO584#10.22#23498#225
Truck#IPE879#22.72#149089#8.28
```

4.5.1 Members

- - currCars : int
  - This member variable keeps track of the total cars currently in the fleet.
- - currBuses : int
  - This member variable keeps track of the total buses currently in the fleet.
- - currTrucks : int
  - This member variable keeps track of the total trucks currently in the fleet.
- - maxSize : int
  - This member variable represents the maximum number of Cars, Buses and Trucks that can be in the fleet.
  - All 3 have the same maxSize.
- - cars : Car**
  - This member variable represents the cars in the fleet.
  - It is a one-dimensional array of dynamic car objects.
- - buses : Bus**
  - This member variable represents the buses in the fleet.
  - It is a one-dimensional array of dynamic bus objects.
- - trucks : Truck**

  - This member variable represents truck in the fleet.
  - It is a one-dimensional array of dynamic truck objects.

4.5.2 Functions

- + Fleet(size: int, path: string)
  - This is the constructor for the Fleet class.
  - Set the max size member variable to the passed in size parameter.
  - Initialize the 3 dynamic arrays with a size of maxSize.
  - Initialize each index in the arrays to NULL.
  - Set the current number of Cars, Buses and Truck to 0.
  - Populate the arrays with the vehicles in the textfile. The name of the textfile is given by path parameter. Each line correspond to a vehicle object that should be created.
  - The format of the textfile is:

```
Type#Registration#GasUsage#Mileage#Derived
```

  - If the Type is ”Car”, Derived represent the top speed.
  - If the Type is ”Bus”, Derived represent the seats.
  - If the Type is ”Truck”, Derived represent the carry weight.
  - Create the correct derived vehicle object based on the Type with the correct values.
  - Add the new object to the appropriate array and remember to increment the current member variable.
- + Fleet(toCopy: const Fleet*)
  - This is the copy constructor for the Fleet class.
  - Set the maxSize, and the current member variables to the corresponding variables of the toCopy Fleet object.
  - Create deep copies of the cars, buses and trucks arrays.
  - *Hint: Set all indexes to NULL before creating the deep copy.*
- + ∼Fleet()
  - This is the destructor for the Fleet class.
  - It should deallocate all the dynamic memory i.e. cars, buses and truck arrays.
- + getCars() : const Car**
  - This is an accessor method for the cars member variable.
  - This is a const function and does not return a const value.
  - It should return the cars member variable.

- + getBuses() : const Bus**
  - This is an accessor method for the buses member variable.
  - This is a const function and does not return a const value.
  - It should return the buses member variable.
- + getTruck() : const Truck**
  - This is an accessor method for the trucks member variable.
  - This is a const function and does not return a const value.
  - It should return the trucks member variable.
- + getCurrentCars() : const int
  - This is an accessor method for the current cars member variable.
  - This is a const function and does not return a const value.
  - It should return the current cars member variable.
- + getCurrentBuses() : const int
  - This is an accessor method for the current buses member variable.
  - This is a const function and does not return a const value.
  - It should return the current buses member variable.
- + getCurrentTrucks() : const int
  - This is an accessor method for the current trucks member variable.
  - This is a const function and does not return a const value.
  - It should return the current trucks member variable.
- + getMaxsize() : const int
  - This is an accessor method for the max size member variable.
  - This is a const function and does not return a const value.
  - It should return the max size member variable.
- + addCar(carToAdd : const Car*) : bool
  - This method adds a car to the fleet.
  - The passed in car should be added to the cars array.
  - Remember to increment the current cars and create a deep copy of the carToAdd.
  - If carToAdd is NULL or if the current cars is equal to the maximum allowed, return false. If not and the car was added, return true.
- + addBus(busToAdd : const Bus*) : bool

  - This method adds a bus to the fleet.
  - The passed in bus should be added to the buses array.
  - Remember to increment the current buses and create a deep copy of the busToAdd.
  - If busToAdd is NULL or if the current buses is equal to the maximum allowed, return false. If not and the bus was added, return true.

- + addTruck(truckToAdd : const Truck*) : bool
  - This method adds a truck to the fleet.
  - The passed in truck should be added to the trucks array.
  - Remember to increment the current truck and create a deep copy of the truckToAdd.
  - If truckToAdd is NULL or if the current trucks is equal to the maximum allowed, return false. If not and the truck was added, return true.

- + fleetInfo() : string
  - This method return the details of the fleet.
  - The info should be returned in the following format:

```
Cars:\n
Registration: registration Mileage: mileage Top Speed:
   topSpeed\n
\nBuses:\n
Registration: registration Mileage: mileage Seats: seats\n
\nTrucks:\n
Registration: registration Mileage: mileage Carry: carry\n
```

  - \n correspond to a newline character.
  - The corresponding information of each car, bus and truck should be printed under the corresponding Headers.

4.6 Rental

4.6.1 Members

- - maxBookings: int
  - This member variable represents the maximum number of bookings.

- - currBookings: int
  - This member variable represents the current number of bookings.

- - bookings: Fleet**
  - This member variable represents the fleet rented for each booking.
  - It is a dynamic one-dimensional array of dynamic Fleet objects.

4.6.2 Functions

- + Rental(maxBookings: int, bookingSize: int, path: string)
  - This is the constructor for the rental class.
  - Set the max bookings member variable to the corresponding parameter.
  - Initialize the bookings array with the size of max bookings.
  - Initialize the current bookings to 0.
  - Set all the indexes of the bookings array to NULL.
  - Add a new fleet object at the first index of the bookings array by calling the Fleet constructor and passing the other two parameters as the parameters for the Fleet constructor.
  - Since a new booking get added, increment the current bookings.
- + ∼Rental()
  - This is the destructor for the Rental class.
  - It should deallocate all the dynamic memory i.e. bookings array.
- + getBookings() : const Fleet**
  - This is an accessor method for the bookings member variable.
  - This is a const function and does not return a const value.
  - It should return the bookings member variable.
- + getCurrentBookings() : const int
  - This is an accessor method for the current bookings member variable.
  - This is a const function and does not return a const value.
  - It should return the current bookings member variable.
- + getMaxBookings() : const int
  - This is an accessor method for the max bookings member variable.
  - This is a const function and does not return a const value.
  - It should return the max bookings member variable.
- + newBooking(bookingFleet : const Fleet*) : bool
  - This method adds a fleet to the bookings.
  - The passed in fleet should be added to the bookings array.
  - Remember to increment the current bookings and create a deep copy of bookingFleet.
  - If bookingFleet is NULL or if the current bookings is equal to the maximum allowed, return false. If not and the fleet was added, return true.

- + cancelBooking(index : int) : bool
  - This method removes a fleet from the bookings.
  - If the passed in index is valid, deallocate the fleet object at the index and return true.
  - The rest of the fleets in the array need to be shifted down. Therefore the NULL fleets will be at the end of the array.
  - Remember to decrement the current bookings.
  - If the index is not valid, return false.
  - An index greater than the current number of bookings can not be removed.
- + bookingSummary(): string
  - This method return a summary of the bookings array.
  - The summary have the following format:

```
Booking: index Cars: numCars Buses: numBuses Trucks: trucks
   Total Cost: bookingCost\n
```

  - Index correpsond to the index of the fleet in the array.
  - numCars, numBuses and numTrucks correspond to the number of the current number of cars, buses and trucks in the fleet.
  - bookingCost correspond to the cost of the booking. Call the calculateBookingCost for each of the bookings to get the correct amount.
- + calculateBookingCost(index: int): double
  - This method calculates the cost to rent the fleet at the specified index.
  - If the index is not valid return -100.
  - If the index is valid, call the calculateCost method for each of the cars, buses and trucks in the fleet and add them together.
  - The calculateCost method is void, therefore you must make use of the getCost method as well.
  - Return the result from the above.

5 Testing

As testing is a vital skill that all software developers need to know and be able to perform daily, 10% of the assignment marks will be allocated to your testing skills. To do this, you will need to submit a testing main (inside the `main.cpp` file) that will be used to test an Instructor Provided solution. You may add any helper functions to the main.cpp file to aid your testing. In order to determine the coverage of your testing the gcov [1] tool, specifically the following version *gcov (Debian 8.3.0-6) 8.3.0*, will be used. The following set of commands will be used to run gcov:

[1] For more information on gcov please see `https://gcc.gnu.org/onlinedocs/gcc/Gcov.html`

```
g++ --coverage *.cpp -o main
./main
gcov -f -m -r -j ${files}
```

This will generate output which we will use to determine your testing coverage. The following coverage ratio will be used:

$$\frac{\text{number of lines executed}}{\text{number of source code lines}}$$

We will scale this ration based on class size.

The mark you will receive for the testing coverage task is determined using Table 1:

Coverage ratio range	% of testing mark
0%-5%	0%
5%-20%	20%
20%-40%	40%
40%-60%	60%
60%-80%	80%
80%-100%	100%

Table 1: Mark assignment for testing

Note the top boundary for the Coverage ratio range is not inclusive except for 100%. Also, note that only the functions stipulated in this specification will be considered to determine your testing mark. Remember that your main will be testing the Instructor Provided code and as such, it can only be assumed that the functions outlined in this specification are defined and implemented.

As you will be receiving marks for your testing main, we will also be doing plagiarism checks on your testing main.

6 Implementation Details

- You must implement the functions in the header files exactly as stipulated in this specification. Failure to do so will result in compilation errors on FitchFork.
- You may only use **c++98**.
- You may only utilize the specified libraries. Failure to do so will result in compilation errors on FitchFork.
- Do not include using `namespace std` in any of the files.
- You may only use the following libraries:
  - string

  - cmath
  - sstream
  - fstream
  - iostream
- You are supplied with a **trivial** main demonstrating the basic functionality of this assessment.

7 Upload Checklist

The following c++ files should be in a zip archive named uXXXXXXXX.zip where XXXXXXXX is your student number:

- vehicle.h
- vehicle.cpp
- car.h
- car.cpp
- bus.h
- bus.cpp
- truck.h
- truck.cpp
- fleet.h
- fleet.cpp
- rental.h
- rental.cpp
- `main.cpp`
- Any textfiles used by your `main.cpp`
- `testingFramework.h` and `testingFramework.cpp` if you used these files.

The files should be in the root directory of your zip file. In other words, when you open your zip file you should immediately see your files. They should not be inside another folder.

8 Submission

You need to submit your source files on the FitchFork website (`https://ff.cs.up.ac.za/`). All methods need to be implemented (or at least stubbed) before submission. Your code should be able to be compiled with the following command:

```
g++ *.cpp -o main
```

and run with the following command:

```
./main
```

Remember your `h` file will be overwritten, so ensure you do not alter the provided `h` files.
You have 10 submissions and your best mark will be your final mark. Upload your archive to the Practical 2 slot on the FitchFork website. Submit your work before the deadline. **No late submissions will be accepted!**

9 Accessibility

1 Show the class diagram of the six classes to be implemented in this practical:

- Vehicle
- Car
- Bus
- Truck
- Fleet
- Rental

The member variables and functions are discussed in Section 4. The following relationships exists between the classes:

- Car, Bus and Truck inherit from Vehicle.
- Fleet has a relationship with Car because of the cars member variable.
- Fleet has a relationship with Bus because of the bus member variable.
- Fleet has a relationship with Truck because of the truck member variable.
- Rental has a relationship with Fleet because of the bookings member variable.