Aleks Atanasov

COS 221

Fall Semester 2022

Vladimir Georgiev

10.22.2022

Course Project ADT

Rent a Vehicle



The idea of the project is to calculate how much a user has to pay for the rent and the consumption of different types of vehicles. This will be achieved by creating a three-level hierarchy of classes that represented different types of Vehicles, that will inherit properties and methods of the base class Vehicle and add multiple more attributes that the different types of vehicles have.

Vehicle

Karts

Scooter

Bike

Car

Double

RX250

RX7

Luxury

Van

Electric

There would be a base class, called Vehicle which has the following abstract methods:

getTotalPrice() = 0; calculate the total price by adding together the rent and the consumption of fuel/electricity prices.

getTotalRentPrice() = 0; calculates the price for the rent services based on the price and the duration of the rent.

getRefilTankPrice() = 0; calculates the price for the consumption of fuel/electricity that needs to be paid

These will be inherited and implemented in different ways in the child classes (type of vehicles). These polymorphic methods will be overridden in the child classes based on the different uses/ways of calculating.

**class Vehicle** { //creating an abstract class Vehicle

**public:**

**virtual double GetTotalPrice() = getTotalRentPrice() + getRefilTankPrice();**

**virtual double getTotalRentPrice() = 0;**

**virtual double getRefillTankPrice() = 0;** // declaring the pure virtual function in the base class, that is later overriden in the derived classes because an abstract method has no definition in the base class and has to be defined in a derived class

};

The first subclass of Vehicle is going to be Car. The following attributes would be added and the following methods will be overridden:

**class Car: class Vehicle {**

**private:**

**string name;**

**string model;**

**int yearOfProduction;**

**int horsePower;**

**string transmission;**

**FuelType fuelType;**

**double dailyRentPrice;**

**int rentDuration;**

**string color;**

**double tankAmount;**

**double tankAmountAfterRent;**

**bool paidProtectionPlus;**

**double protectionPlus;**

**Car();**

**Car(string name, string model, int yearOfProduction, int horsepower, string transmission, FuelType typeOfFuel, double dailyRentPrice, int rentDuration, string color, double tankAmount, double tankAmountAfterRent, bool paidProtectionPlus, double protectionPlus);**

**double getTotalPrice();**

**double getTotalRentPrice();**

**double getRefillTankPrice();**

**};**

getTotalRentPrice() will be calculated based on the duration and the daily price for renting the car. Also, the user of the car has a choice if he wants to pay additional protection warranty in case something happens to the vehicle (collision, lost keys, damage) there would be a boolean variable paidProtectionPlus, if boolean variable equals true: additional protectionPlus cost will be added to the getTotalRentPrice().

**double Car::getTotalRentPrice()**

**{**

**double totalPrice;**

**if (paidProtectionPlus == true)**

**totalPrice = rentDuration \* dailyRentPrice + protectionPlus;**

**else**

**totalPrice = rentDuration \* dailyRentPrice;**

**return totalPrice;**

**}**

The idea of the second method is that when a car is provided by the supplier it has a full tank. After usage the user has to pay for filling up the tank again - paying for his consumption. The method will be based around the type of fuel the car requires and the amount the user has used:

**const double petrolPrice = 2.94;**

**const double dieselPrice = 3.14;**

**const double lpgPrice = 1.44;**

**enum fuelType {petrol, diesel, lpg}**

**double Car::getRefillTankPrice()**

**{**

**double refillPrice;**

**if (fuelType == petrol)**

**refillPrice = ((tankAmount - tankAmountAfterRent) \* petrolPrice);**

**else if (fuelType == diesel)**

**refillPrice = (tankAmount - tankAmountAfterRent) \* dieselPrice;**

**else if (fuelType == lpg)**

**refillPrice = (tankAmount - tankAmountAfterRent) \* lpgPrice;**

**else if (fuelType == electric)**

**refillPrice = (tankAmount - tankAmountAfterRent) \* electrictyKWHPrice;**

**return refillPrice;**

**}**

The derived classes from class Car will be Electric, Luxury, Van. All these subclasses will have different attributes.There will be a difference regarding the electric Car class, where we will override the getRefillTankPrice() to calculate based on the used electricity, not fuel.

Electric car will have the following additional attributes:

**Class Electric: public Car {**

**Electric();**

**Electric(string name, string model, int yearOfProduction, int horsepower, string transmission, FuelType typeOfFuel, double dailyRentPrice, int rentDuration, string color, double tankAmount, double tankAmountAfterRent, bool paidProtectionPlus, double protectionPlus);**

**getRefillTankPrice()**

**};**

Electric car will have the getRefillTankPrice() method overriden:

**const double electrictyKWHPrice = 0.16;**

**double Electric::getRefillTankPrice() class**

**{**

**return Car::getRefillTankPrice();**

**}**

The Van subclass will have these additional attributes:

**enum VanType {cargo,passenger}**

**class Van: public Car {**

**private:**

**int numOfSeats;**

**VanType vanType;**

**numOfDoors;**

**Van();**

**Van(int numOfSeats, VanType vanType, numOfDoors, string name, string model, int yearOfProduction, int horsepower, string transmission, FuelType typeOfFuel, double dailyRentPrice, int rentDuration, string color, double tankAmount, double tankAmountAfterRent, bool paidProtectionPlus, double protectionPlus);**

**getTotalRentPrice();**

The rent calculating method will be overridden based on what are you going to use the Van for – transporting cargo or passengers.

**double Van::getTotalRentPrice()**

**{**

**double totalPrice;**

**if (vanType == cargo)**

**totalPrice = Car::getTotalRentPrice() \* 1.5;**

**else if (vanType == passenger)**

**{**

**if (numOfSeats > 7)**

**totalPrice = Car::getTotalRentPrice() + 100;**

**else**

**totalPrice = Car::getTotalRentPrice();**

**}**

**return totalPrice;**

**}**

For Luxury car class paidProtection is mandatory, as they are very expensive cars!

**class Luxury: public Car {**

**private:**

**bool sportsMode;**

**bool cabrio;**

**Luxury();**

**Luxury(bool sportsMode, bool cabrio, string name, string model, int yearOfProduction, int horsepower, string transmission, FuelType typeOfFuel, double dailyRentPrice, int rentDuration, string color, double tankAmount, double tankAmountAfterRent, bool paidProtectionPlus, double protectionPlus);**

**getTotalRentPrice();**

**};**

Overriding the method for calculating the rent price of luxury car:

**double Luxury::getTotalRentPrice()**

**{**

**double totalPrice;**

**if ( Car::getHorsePower() > 500 && ( sportsMode == true || cabrio == true))**

**totalPrice = Car::getTotalRentPrice() + 2.5 \* Car::getProtectionPlus();**

**else**

**totalPrice = Car::getTotalRentPrice() + 2 \* Car::getProtectionPlus();**

**return totalPrice;**

**}**

The classes scooter and bike will inherit class Vehicle and will have different attributes than Car. Both classes will not override the getRefilTankPrice() because there is no need to calculate that. So, we will just override the getTotalRentPrice():

For the scooters:

**class Scooter: public Vehicle {**

**private:**

**string name;**

**double startingFee;**

**double durationMinutes;**

**double pricePerMinute;**

**int topSpeed;**

**double weigth;**

**Scooter();**

**Scooter(string name, double startingFee, double durationMinutes, double pricePerMinute, int topSpeed, double weigth);**

**double getTotalRentPrice(); };**

**class Bike: public Vehicle {**

**private:**

**int numOfWheels;**

**double pricePerHour;**

**double durationHours;**

**int gears;**

**string color;**

**int wheelsSize;**

**Bike();**

**Bike(int numOfWheels, double pricePerHour, double durationHours, int gears, string color, int wheelsSize);**

**getTotalRentPrice();**

overriding the method for the scooter:

**double Scooter::getTotalRentPrice()**

**{**

**return (startingFee + durationMinutes \* pricePerMinute);**

**}**

and for the bikes:

**double Bike::getTotalRentPrice()**

**{**

**return durationHours \* pricePerHour;**

**}**

The final subclass of Vehicle will be Kart. Kart will have the following attributes and methods:

**class Kart: public Vehicle {**

**string name;**

**int numberOfLaps;**

**double pricePerLap;**

**int number;**

**int horsePower;**

**int maxSpeed;**

**Kart();**

**Kart(string name, int numberOfLaps, double pricePerLap, int number, int horsepower, int maxSpeed);**

**getTotalRentPrice();**

**};**

**double Kart::getTotalRentPrice()**

**{**

**return numberOfLaps \* pricePerLap;**

**}**

There will be three subclasses of Kart: Double, RX7 and RX250. The difference is that the double Kart can accommodate two people and has a weight limit, while other can accomodate maximum one. Therefore, the price of renting the double kart will be x1.5 times more expensive. The RX7 is the default Kart, while RX250 has more horsepower, bigger topSpeed and additional features. Therefore, the faster Kart will have bigger Price. The rent of the Kart will be calculated based on the type of the Kart and how many laps you want to rent it for. The getTotalRentPrice() method will be overridden for every type of Kart.

**class RX7: public Kart {**

**RX7();**

**RX7(string name, int numberOfLaps, double pricePerLap, int horsePower, int maxSpeed)**

**getTotalRentPrice(); };**

Since the RX250 is a racing Kart – a way better performing Kart (involving more horse power and bigger fuel consumption), the user has to pay starting fee for using the improved features of the racing Kart. A difference is that you can choose to drive the faster Kart with paddleShifters that help you change gears by yourself or automated.

**class RX250: public Kart {**

**private:**

**double kartStartingFee;**

**bool paddleShifters;**

**RX250();**

**RX250(paddleShifters, string name, int numberOfLaps, double pricePerLap, int horsePower, int maxSpeed)**

**getTotalRentPrice(); };**

**double RX250::getTotalRentPrice()**

**{**

**return kartStartingFee + (1.2\* (Kart::getTotalRentPrice()));**

**}**

**class Double: public Kart {**

**private:**

**int capacity;**

**double weightLimit;**

**Ddouble();**

**Double(int capacity, double weightLimit, string name, int numberOfLaps, double pricePerLap, int horsePower, int maxSpeed)**

**getTotalRentPrice(); };**

**double Double::getTotalRentPrice()**

**{**

**return 1.5\* Kart::getTotalRentPrice();**

**}**

The algorithms that will be used will be KMP algorithm for pattern matching and the Quicksort algorithm:

**KMP algorithm:**

This will enable the search of a vehicle that you want to rent – search by its name/part of its name. After that, all of the results that have a match will be returned with all the respective attributes of the vehicle.

* Time complexity of O (n/m) is the best scenario, n is the size of the input text and m is the size of the pattern and O (n + m) in the worst scenario.

**A Quicksort algorithm** that is going to sort all Vehicles by their total price.

* We pick a pivot element. After the partitioning phase we have the correct position of the pivot and all smaller elements are on the left side and all bigger are on the right side – then we sort the two subarrays
* Time complexity of O(N\*logN) in the best scenario and O(N2) in the worst scenario.

**Main method:**

We will define multiple vectors for storing the data of the Vehicles rented. After reading the data from an external input file. We call the quicksort algorithm and the KMP for the vectors that we have created and then display the results through two for loops: the matches of the search and in the second: names and total prices of all Vehicles - sorted by total price.