## horizontal line

Car & Bike Rental System

# Abstract

This project is a sample application that tracks and manages the details of the rentals of cars and bikes. Using this application in a working environment can reduce the manual work for registering and managing the information about the car and bike rentals.

Ankhbayar Enkhlkhagva COS 221a Fundamental Data Structures

ID: 100132497 Spring 2020

Mail address: ANE171@aubg.edu Instructor: Professor Vladimir Georgiev

# Introduction

Car and Bike Rental System is an application-based project. It is designed for people to **rent** a car or a bike, check their **rental status,** and **delete** records (for admins). Rather than going through the pages of their diaries and copies to search a rental information, a person can simply use this software to view any of his stored information about all the rentals. Admins see the whole database and delete the records according to his need. Big organizations need to have this software in order to keep the records of thousands of rentals at one place without any paperwork.

**Project Goals**

Key goals of the Car and Bike Rental System:

* **Provide an easy interface for user. Example:**

Welcome to “ABC” car and bike rental system.

1. Rent a car/bike
2. Check a rental information
3. Print records (admins only)
4. Delete a record (admins only)
5. Quit

Enter your selection: \_

* **Easy to store records. (OPTION 1)**

When the user *chooses option 1*, the program will ask user to *choose what to rent (car or bike)*. Depending on the choice, the program will *print all available types of car/bike (diesel/petrol and electric/traditional)* in the company. When the user chooses the type, the program will *print all available cars/bikes (Petrol/Diesel and Electric/Traditional)* that belongs to the chosen type. After this process, the *program will ask the user to enter their information* *(name, phone number, etc.)*, followed by duration of the rental.

After the user finish giving their information, the program *will print Invoice letter* for the user. Invoice letter contains their Invoice ID, rented car/bike info, and total price for the rental. (Every rental has its unique Invoice ID, which will be assigned randomly during the rental process.)

All records *will be saved in a vector ,Rentals*. /**vector<Rental\*>Rentals**/

* **Users can check their rental information (OPTION 2)**

When the user *chooses option 2*, the program *will ask the user to enter their Invoice ID*. Using this data, the program *will traverse through the vector*, Rentals, to *find matching records*. The program uses **KMP algorithm** to search matching IDs. If there is a record matching to the ID, the program *will print the user’s Invoice letter*.

* **Admins can see all the records (OPTION 3)**

When the user *chooses option 3*, the program will *ask the user to enter password*, which only admins own. If the password is correct, the program will *print all the records* in the Rentals vector. Records will be sorted using their duration of rentals. The program will use **QuickSort algorithm**.

* **Admins can delete a record (OPTION 4)**

When the user *chooses option 4,*  the program will *ask the user to enter password*, which only admins own. If the password is correct, the program will *ask the admin to enter Invoice ID* of the record to be deleted. The program will search the record, using KMP algorithm, and deletes the record.

# Hierarchy of classes

This program consists of 7 classes, shown below.

Rental (base class)

*Rental* is a base class and it is abstract. It consists of pure virtual functions, such as, getTotalPrice(), getName(), etc. Definitions of this functions are overridden in the derived classes (Car, Bike, Diesel, Petrol, EBike, and TBike).

Car and Bike (Inherits Rental class)

*Car and Bike* are derived classes, which inherits class, Rental. These classes have its own constructor, getters, and setters, followed by their member functions. The definition of pure virtual function, getTotalPrice(), will be overridden. It will calculate total price of the rent as follows:

**Pseudocode for Car:**

|  |
| --- |
| double getTotalPrice { return price \* duration } |

Diesel/Petrol and Ebike/TBike (Inherits Car and Bike classes)

These classes have its own constructor, getters, and setters, followed by their member functions.

getTotalPrice() will be overridden, because each class has its own discount percentage to be applied. If the duration of rent is equal or more than a week, the user gets the discount. It will calculate total price of the rent, after the discount, as follows:

**Pseudocode for Diesel:**

|  |
| --- |
| double getTotalDiscount() {  if ( duration >= 7 ) { return discount \* duration } //discount is member variable of its class  else { return 0 }  }  double getTotalPrice { Car::getTotalPrice() - getTotalDiscount() } |