

INFORMATICS INSTITUTE OF TECHNOLOGY

In collaboration with

UNIVERSITY OF WESTMINSTER

Object Oriented Principles

5COSC007C

Coursework – Phase 4

Vehicle Rental System

Module Leader’s Name – Mr. Guhanathan Poravi

Dinuka Piyadigama

UoW ID – 17421047

IIT ID – 2018373

Contents

[Design 2](#_Toc26105354)

[1) Use Case Diagram 2](#_Toc26105355)

[2) Class Diagram 3](#_Toc26105356)

[All Code + Screenshots of GUIs 4](#_Toc26105357)

[ConsoleApp 4](#_Toc26105358)

[Controller – Package 7](#_Toc26105359)

[API 7](#_Toc26105360)

[DatabaseController 11](#_Toc26105361)

[GUIController 16](#_Toc26105362)

[WestminsterRentalVehicleManager 22](#_Toc26105363)

[Model 33](#_Toc26105364)

[RentalVehicleManager 33](#_Toc26105365)

[Vehicle 34](#_Toc26105366)

[Schedule 37](#_Toc26105367)

[Car 39](#_Toc26105368)

[Motorbike 41](#_Toc26105369)

[View 43](#_Toc26105370)

[GUI 43](#_Toc26105371)

[Screenshots – JavaFX GUI 53](#_Toc26105372)

[Angular GUI 63](#_Toc26105373)

[app.component.html 63](#_Toc26105374)

[app.component.ts 66](#_Toc26105375)

[vehicle.service.ts 73](#_Toc26105376)

[styles.scss 74](#_Toc26105377)

[app.component.scss 75](#_Toc26105378)

[Screenshots – Angular GUI 77](#_Toc26105379)

[Testing 81](#_Toc26105380)

[Test Plan 81](#_Toc26105381)

[Automated testing with Junit 84](#_Toc26105382)

[Code – Junit testing 84](#_Toc26105383)

[Screenshots – Junit testing 99](#_Toc26105384)

# Design

## 1) Use Case Diagram

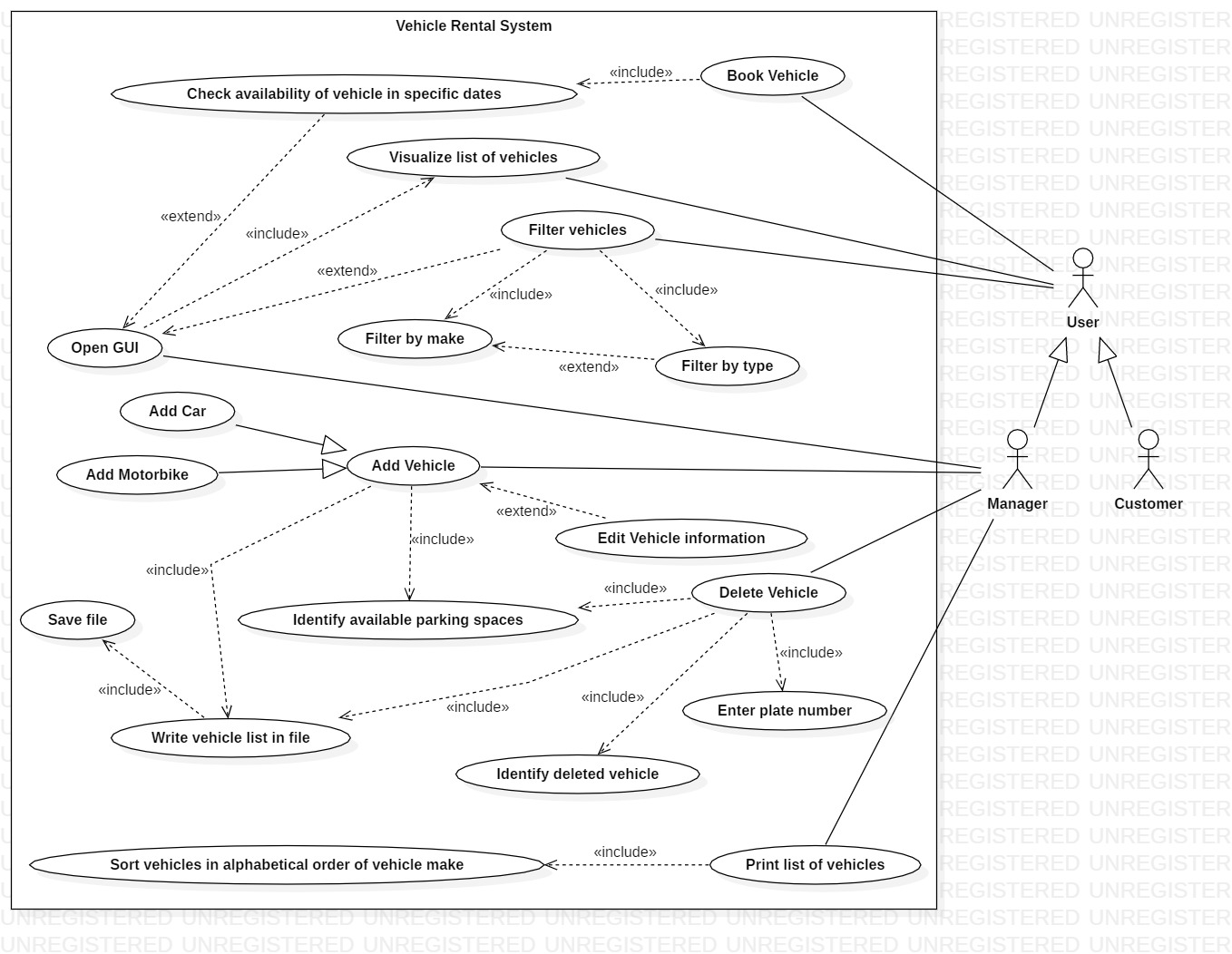
*“You are* *required to develop a program that implements a basic vehicle rental system.”*

I have included the GUI section and the Console section in the same use case diagram as the assignments says that both of these are part of a single system.

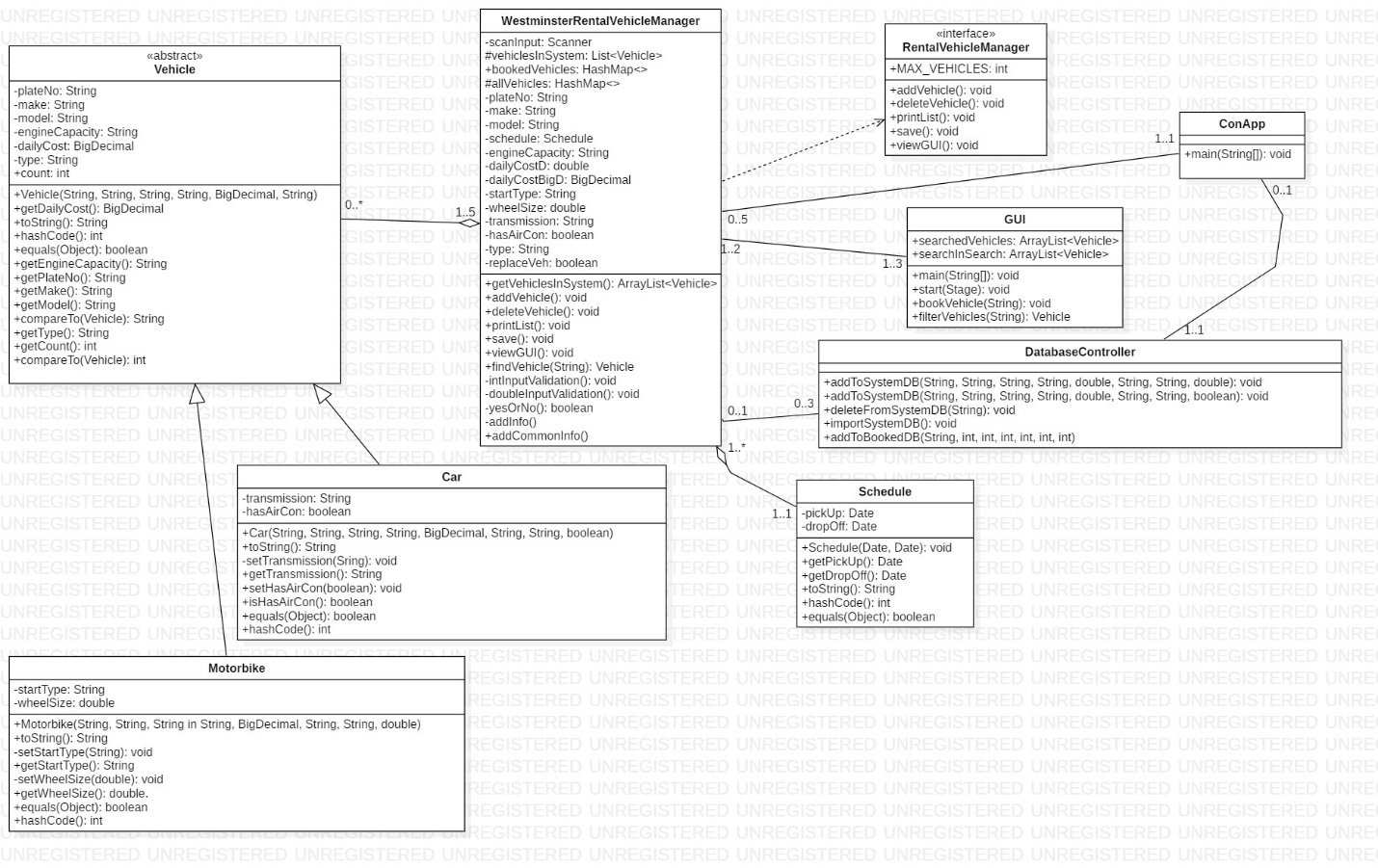
***“Create******a******Graphical******User******Interface*** *(GUI) that can be opened selecting an option from the menu console”*

My use case diagram satisfies this condition as well.

But I have ensured that the customers can’t change the information in the system by using specialization, which clearly shows that the Manager is only allowed to perform managerial operations.



## 2) Class Diagram



# All Code + Screenshots of GUIs

## ConsoleApp

package lk.dinuka.VehicleRentalSystem;

import lk.dinuka.VehicleRentalSystem.Controller.API;

import lk.dinuka.VehicleRentalSystem.Controller.DatabaseController;

import lk.dinuka.VehicleRentalSystem.Controller.WestminsterRentalVehicleManager;

import java.util.HashMap;

import java.util.Scanner;

public class ConApp {

private static HashMap<String, String> accessCredentials = new HashMap<>(); //used to store for the user name & password to access the system functions

//A hashMap is used to allow multiple user access credentials

public static void main(String[] args) {

accessCredentials.put("PrimaryAdmin", "welcome123"); //valid user name and password

Scanner sc = new Scanner(System.in);

System.out.println("Enter Login Credentials to access system");

System.out.printf("UserName: ");

String username = sc.nextLine();

System.out.printf("Password: ");

String password = sc.nextLine();

sc.reset(); //clearing the cache of the scanner to secure username and password

if (accessCredentials.containsKey(username) && password.equals(accessCredentials.get(username))) {

int chooseOption;

API.allowHeaders(); //allow headers in multiple responses

DatabaseController.importSystemDB(); //importing Vehicles and Bookings saved in database

System.out.println("\n----All vehicles and bookings retrieved from database.----");

System.out.println("```````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````");

do {

System.out.println("\n\t\\~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~/");

System.out.println("\t||````` ~~\tVehicle Rental System\t~~ `````||");

System.out.println("\t/~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\\");

//display main menu

System.out.println("\n1) Add item");

System.out.println("2) Delete item");

System.out.println("3) Print list of items");

System.out.println("4) Open GUI");

System.out.println("5) Exit program");

// Scanner sc = new Scanner(System.in);

System.out.print("\nEnter Option:\n>>");

while (!sc.hasNextInt()) { //validation for integer input

System.out.println("Only integer numbers are allowed! Please provide a valid input"); //error handling message for characters other than integers

sc.next(); //removing incorrect input entered

}

chooseOption = sc.nextInt();

WestminsterRentalVehicleManager managementAction = new WestminsterRentalVehicleManager(); //new object

switch (chooseOption) {

case 1: //add vehicle

managementAction.addVehicle();

break;

case 2: //delete vehicle

managementAction.deleteVehicle();

break;

case 3: //print list of vehicles

managementAction.printList();

break;

case 4: //open GUI

managementAction.viewGUI();

break;

case 5: //display exit message

System.out.println("\n\n\*\*\*\* +----------------------+ \*\*\*\*");

System.out.println(" Thank you for using the Vehicle Management System");

System.out.println("\tLooking forward to assist you in the future.");

System.out.println("\t\t\tExiting Program...");

System.out.println(" +-------+ ");

System.exit(0);

default:

System.out.println("Invalid input. Please try again");

}

} while (chooseOption != 5);

} else {

System.out.println("> Incorrect Access Credentials were entered! <");

}

}

}

/\*Reference:

https://stackoverflow.com/questions/7421612/slf4j-failed-to-load-class-org-slf4j-impl-staticloggerbinder

\*/

## Controller – Package

### API

package lk.dinuka.VehicleRentalSystem.Controller;

import com.google.gson.Gson;

import com.google.gson.GsonBuilder;

import lk.dinuka.VehicleRentalSystem.Model.Schedule;

import spark.Spark;

import java.time.LocalDate;

import static spark.Spark.\*;

public class API {

public static void getAllVehiclesToFront() {

Gson prettyGson = new GsonBuilder().setPrettyPrinting().create();

String vehiclesPrettyJson = prettyGson.toJson(WestminsterRentalVehicleManager.getVehiclesInSystem());

// System.out.println("Cars in Json Format: " + vehiclesPrettyJson);

//GET - used to load data into GUI

get("/hello", "application/json", (request, response) -> {

return vehiclesPrettyJson;

});

}

public static void postBookingsFromFront() {

//get plateNo of vehicle and days, book vehicle if available and let the front end know it's availability

//POST - used to book

post("/books","application/json", (request, response) -> {

String responsePrettyJson;

String plateNo = request.queryParams("plateNo");

int yearPickUp = Integer.parseInt(request.queryParams("yearPickUp"));

int monthPickUp = Integer.parseInt(request.queryParams("monthPickUp"));

int dayPickUp = Integer.parseInt(request.queryParams("dayPickUp"));

int yearDropOff = Integer.parseInt(request.queryParams("yearDropOff"));

int monthDropOff = Integer.parseInt(request.queryParams("monthDropOff"));

int dayDropOff = Integer.parseInt(request.queryParams("dayDropOff"));

LocalDate pickUpDate = LocalDate.of(yearPickUp,monthPickUp,dayPickUp);

LocalDate dropOffDate = LocalDate.of(yearDropOff,monthDropOff,dayDropOff);

Schedule newBooking = new Schedule(pickUpDate,dropOffDate);

// System.out.println(newBooking);

boolean created = GUIController.createBooking(plateNo,newBooking);

response.status(201); // 201 Created

if (created) { //if booking was created

Gson prettyGson = new GsonBuilder().setPrettyPrinting().create();

responsePrettyJson = prettyGson.toJson("successful");

// adding new booking to the database

DatabaseController.addToBookedDB(plateNo, yearPickUp, monthPickUp, dayPickUp,

yearDropOff, monthDropOff, dayDropOff);

}else{ //if booking wasn't created (already booked)

Gson prettyGson = new GsonBuilder().setPrettyPrinting().create();

responsePrettyJson = prettyGson.toJson("unsuccessful");

}

// return created; //true if successful

return responsePrettyJson;

});

}

public static void postAvailabilityFromFront() {

//get plateNo of vehicle and days. check whether vehicle is available and let the front end know it's availability

//POST - used to book

post("/checks","application/json", (request, response) -> {

String responsePrettyJson;

String plateNo = request.queryParams("plateNo");

int yearPickUp = Integer.parseInt(request.queryParams("yearPickUp"));

int monthPickUp = Integer.parseInt(request.queryParams("monthPickUp"));

int dayPickUp = Integer.parseInt(request.queryParams("dayPickUp"));

int yearDropOff = Integer.parseInt(request.queryParams("yearDropOff"));

int monthDropOff = Integer.parseInt(request.queryParams("monthDropOff"));

int dayDropOff = Integer.parseInt(request.queryParams("dayDropOff"));

LocalDate pickUpDate = LocalDate.of(yearPickUp,monthPickUp,dayPickUp);

LocalDate dropOffDate = LocalDate.of(yearDropOff,monthDropOff,dayDropOff);

Schedule newBooking = new Schedule(pickUpDate,dropOffDate);

// System.out.println(newBooking);

boolean created = GUIController.checkAvailabilityOfVeh(plateNo,newBooking);

response.status(201); // 201 Created

if (created) { //if booking was created

Gson prettyGson = new GsonBuilder().setPrettyPrinting().create();

responsePrettyJson = prettyGson.toJson("successful");

}else{ //if booking wasn't created (already booked)

Gson prettyGson = new GsonBuilder().setPrettyPrinting().create();

responsePrettyJson = prettyGson.toJson("unsuccessful");

}

// return created; //true if successful

return responsePrettyJson;

});

}

public static void allowHeaders() {

Spark.staticFiles.location("/assets");

Spark.staticFiles.header("Access-Control-Allow-Origin", "\*");

options("/\*", (req, res) -> {

String accessControlRequestHeaders = req.headers("Access-Control-Request-Headers");

if (accessControlRequestHeaders != null) {

res.header("Access-Control-Allow-Headers", accessControlRequestHeaders);

}

String accessControlRequestMethod = req.headers("Access-Control-Request-Method");

if (accessControlRequestMethod != null) {

res.header("Access-Control-Allow-Methods", accessControlRequestMethod);

}

return "OK";

});

before((req, res) -> {

res.header("Access-Control-Allow-Origin", "\*");

res.header("Access-Control-Allow-Headers", "\*");

res.type("application/json");

});

}

}

/\*

References:

https://github.com/perwendel/spark

http://sparkjava.com/documentation#response-transformer

JSON and java objects

https://github.com/google/gson

https://www.baeldung.com/spark-framework-rest-api

https://gist.github.com/saeidzebardast/e375b7d17be3e0f4dddf#gistcomment-2704256

https://technology.finra.org/code/serialize-deserialize-interfaces-in-java.html

https://crunchify.com/in-java-how-to-convert-arraylist-to-jsonobject/

Make http requests from browser

https://github.com/axios/axios

https://gist.github.com/akexorcist/ea93ee47d39cf94e77802bc39c46589b

\*/

### DatabaseController

package lk.dinuka.VehicleRentalSystem.Controller;

import com.mongodb.MongoClientURI;

import com.mongodb.client.MongoCollection;

import com.mongodb.client.MongoDatabase;

import com.mongodb.client.model.Filters;

import lk.dinuka.VehicleRentalSystem.Model.Car;

import lk.dinuka.VehicleRentalSystem.Model.Motorbike;

import lk.dinuka.VehicleRentalSystem.Model.Schedule;

import lk.dinuka.VehicleRentalSystem.Model.Vehicle;

import org.bson.Document;

import java.math.BigDecimal;

import java.time.LocalDate;

import java.util.ArrayList;

public class DatabaseController {

public static void addToSystemDB(String plateNo, String make, String model, String engineCapacity, double dailyCost, String type, String startType, double wheelSize) {

//Adding a Motorbike to the Collection

MongoClientURI uri = new MongoClientURI(

"mongodb+srv://cw\_user:123098@cluster0-gxfyy.gcp.mongodb.net/test?retryWrites=true&w=majority");

com.mongodb.MongoClient mongoClient = new com.mongodb.MongoClient(uri);

MongoDatabase database = mongoClient.getDatabase("VehicleRentalSystem");

//Access collection

MongoCollection<Document> collection = database.getCollection("VehiclesInSystem");

//create a document

Document newVehicle = new Document("Plate No", plateNo)

.append("Make", make)

.append("Model", model)

.append("Engine Capacity", engineCapacity)

.append("Daily Cost", dailyCost)

.append("Type", type)

.append("Start Type", startType)

.append("Wheel Size", wheelSize);

//insert the document

collection.insertOne(newVehicle);

}

public static void addToSystemDB(String plateNo, String make, String model, String engineCapacity, double dailyCost, String type, String transmission, boolean hasAirCon) {

//Adding a car to the Collection

MongoClientURI uri = new MongoClientURI(

"mongodb+srv://cw\_user:123098@cluster0-gxfyy.gcp.mongodb.net/test?retryWrites=true&w=majority");

com.mongodb.MongoClient mongoClient = new com.mongodb.MongoClient(uri);

MongoDatabase database = mongoClient.getDatabase("VehicleRentalSystem");

//Access collection

MongoCollection<Document> collection = database.getCollection("VehiclesInSystem");

//create a document

Document newVehicle = new Document("Plate No", plateNo)

.append("Make", make)

.append("Model", model)

.append("Engine Capacity", engineCapacity)

.append("Daily Cost", dailyCost)

.append("Type", type)

.append("Transmission", transmission)

.append("Air Con", hasAirCon);

//insert the document

collection.insertOne(newVehicle);

}

public static void deleteFromSystemDB(String plateNo) { //Deleting an item from the Collection

MongoClientURI uri = new MongoClientURI(

"mongodb+srv://cw\_user:123098@cluster0-gxfyy.gcp.mongodb.net/test?retryWrites=true&w=majority");

com.mongodb.MongoClient mongoClient = new com.mongodb.MongoClient(uri);

MongoDatabase database = mongoClient.getDatabase("VehicleRentalSystem");

//Access collection

MongoCollection<Document> collection = database.getCollection("VehiclesInSystem");

collection.deleteOne(Filters.eq("Plate No", plateNo));

}

public static void importSystemDB() { //Importing stored data in db to application (From VehiclesInSystem & BookedVehicles)

MongoClientURI uri = new MongoClientURI(

"mongodb+srv://cw\_user:123098@cluster0-gxfyy.gcp.mongodb.net/test?retryWrites=true&w=majority");

com.mongodb.MongoClient mongoClient = new com.mongodb.MongoClient(uri);

MongoDatabase database = mongoClient.getDatabase("VehicleRentalSystem");

//importing from VehiclesInSystem collection (For allVehicles HashMap & vehiclesInSystem ArrayList)

//Access collection

MongoCollection<Document> savedCollection = database.getCollection("VehiclesInSystem");

for(Document selectedDoc : savedCollection.find()){

String plateNo = (String)selectedDoc.get("Plate No");

String make = (String) selectedDoc.get("Make");

String model = (String) selectedDoc.get("Model");

String engineCapacity = (String) selectedDoc.get("Engine Capacity");

double dailyCostD = (double) selectedDoc.get("Daily Cost");

String type = (String) selectedDoc.get("Type");

BigDecimal dailyCostBigD = BigDecimal.valueOf(dailyCostD); //converting double to BigDecimal, to use for calculations

if(type.equals("Car")){

String transmission = (String) selectedDoc.get("Transmission");

boolean hasAirCon = (boolean) selectedDoc.get("Air Con");

Vehicle storedCar = new Car(plateNo,make,model,engineCapacity,dailyCostBigD,type,transmission,hasAirCon);

WestminsterRentalVehicleManager.allVehicles.put(plateNo,storedCar);

WestminsterRentalVehicleManager.vehiclesInSystem.add(storedCar);

// System.out.println(storedCar); //to check whether Car was added

}else if(type.equals("Motorbike")){

String startType = (String) selectedDoc.get("Start Type");

double wheelSize = (double) selectedDoc.get("Wheel Size");

Vehicle storedBike = new Motorbike(plateNo,make,model,engineCapacity,dailyCostBigD,type,startType,wheelSize);

WestminsterRentalVehicleManager.allVehicles.put(plateNo,storedBike);

WestminsterRentalVehicleManager.vehiclesInSystem.add(storedBike);

// System.out.println(storedBike); //to check whether Motorbike was added

}

}

//================

//importing from BookedVehicles collection (For bookedVehicles HashMap)

MongoCollection<Document> bookedCollection = database.getCollection("BookedVehicles");

for(Document selectedDoc : bookedCollection.find()){

String plateNo = (String)selectedDoc.get("Plate No");

Document pickUpObject = (Document) selectedDoc.get("pick up");

Document dropOffObject = (Document) selectedDoc.get("drop off");

//breaking down date document to create date using Schedule constructor

//pick up date

int yearUp = pickUpObject.getInteger("year");

int monthUp = pickUpObject.getInteger("month");

int dayUp = pickUpObject.getInteger("day");

//drop off date

int yearDown = dropOffObject.getInteger("year");

int monthDown = dropOffObject.getInteger("month");

int dayDown = dropOffObject.getInteger("day");

LocalDate pickUpDate = LocalDate.of(yearUp,monthUp,dayUp);

LocalDate dropOffDate = LocalDate.of(yearDown,monthDown,dayDown);

Schedule bookedSchedule = new Schedule(pickUpDate,dropOffDate);

if (WestminsterRentalVehicleManager.bookedVehicles.containsKey(plateNo)){

ArrayList bookedDates = WestminsterRentalVehicleManager.bookedVehicles.get(plateNo);

bookedDates.add(bookedSchedule);

WestminsterRentalVehicleManager.bookedVehicles.put(plateNo,bookedDates);

}else{

ArrayList bookedDate = new ArrayList();

bookedDate.add(bookedSchedule);

WestminsterRentalVehicleManager.bookedVehicles.put(plateNo,bookedDate);

}

}

}

public static void addToBookedDB(String plateNo, int yearUp, int monthUp, int dayUp, int yearDown, int monthDown, int dayDown) {

MongoClientURI uri = new MongoClientURI(

"mongodb+srv://cw\_user:123098@cluster0-gxfyy.gcp.mongodb.net/test?retryWrites=true&w=majority");

com.mongodb.MongoClient mongoClient = new com.mongodb.MongoClient(uri);

MongoDatabase database = mongoClient.getDatabase("VehicleRentalSystem");

//Access collection

MongoCollection<Document> collection = database.getCollection("BookedVehicles");

//if already existing, delete document and add new document------------

//create a document

Document newSchedule = new Document("Plate No", plateNo)

.append("pick up", new Document("year", yearUp) //document inside document

.append("month", monthUp)

.append("day", dayUp))

.append("drop off", new Document("year", yearDown) //document inside document

.append("month", monthDown)

.append("day", dayDown));

//insert the document

collection.insertOne(newSchedule);

}

}

/\*

References:

https://www.tutorialspoint.com/mongodb/mongodb\_java

https://mongodb.github.io/mongo-java-driver/3.4/driver/getting-started/quick-start/

https://mongodb.github.io/mongo-java-driver/3.4/driver/getting-started/installation/

https://mongodb.github.io/mongo-java-driver/

https://github.com/mongodb/mongo-java-driver/tree/master

Importing MongoDB documents to Java ArrayList

https://stackoverflow.com/questions/19435621/extract-field-value-from-mongodb-basicdbobject?rq=1

\*/

### GUIController

package lk.dinuka.VehicleRentalSystem.Controller;

import lk.dinuka.VehicleRentalSystem.Model.Schedule;

import lk.dinuka.VehicleRentalSystem.Model.Vehicle;

import java.math.BigDecimal;

import java.time.LocalDate;

import java.time.Period;

import java.util.ArrayList;

import java.util.List;

import static lk.dinuka.VehicleRentalSystem.Controller.WestminsterRentalVehicleManager.bookedVehicles;

public class GUIController {

public static boolean createBooking(Vehicle chosenVeh, Schedule newBooking ) {

//used to create a booking as required and add booking info into the system

List<Schedule> bookedVehicleDates = new ArrayList<>(); //used to record pick up & drop off dates of a vehicle

//Only used to store the dates into the bookedVehicles HashMap

boolean availability = checkAvailabilityOfVeh(chosenVeh, newBooking); //checking whether vehicle is available for booking

System.out.println();

System.out.println("---checked availability---");

System.out.println();

if (availability) {

System.out.println("Vehicle is available for booking");

if (bookedVehicles.containsKey(chosenVeh.getPlateNo())) {

bookedVehicleDates = bookedVehicles.get(chosenVeh.getPlateNo()); //getting recorded bookings into temporary list

}

bookedVehicleDates.add(newBooking); //adding the newly booked dates to the list of bookings.

WestminsterRentalVehicleManager.bookedVehicles.put(chosenVeh.getPlateNo(), (ArrayList) bookedVehicleDates); //adding all booked vehicles to bookedVehicles HashMap

System.out.println(WestminsterRentalVehicleManager.bookedVehicles); //checking whether required booking was entered into the system

return true;

} else {

System.out.println("Vehicle isn't available for booking during the requested time period.");

//vehicle isn't available to be book

return false;

}

}

//``````~~~~~~~~~~~~~~~~~~~``````

public static boolean checkAvailabilityOfVeh(Vehicle chosenVeh, Schedule newBooking ) {

//used to check for the availability of a chosen vehicle

String plateNoOfChosen = chosenVeh.getPlateNo(); //The plate number of the chosen vehicle

if (!WestminsterRentalVehicleManager.bookedVehicles.containsKey(plateNoOfChosen)) {

return true; //vehicle is not booked

} else {

List<Schedule> bookedVehicleDates = new ArrayList<>(); //used to record pick up & drop off dates of a vehicle

bookedVehicleDates = bookedVehicles.get(chosenVeh.getPlateNo()); //getting recorded bookings into temporary list

//Only used to get each of the dates from the bookedVehicles HashMap Values

int totalBookings = bookedVehicles.get(plateNoOfChosen).size();

int passedChecks = 0;

for (int i = 0; i < totalBookings; i++) {

boolean checkPickUpBefore = LocalDate.from(newBooking.getPickUp()).isBefore( //pick up before booked pickup

bookedVehicleDates.get(i).getPickUp());

boolean checkDropOffBefore = LocalDate.from(newBooking.getDropOff()).isBefore( //drop off before booked pick up

bookedVehicleDates.get(i).getPickUp());

boolean checkPickUpAfter = LocalDate.from(newBooking.getPickUp()).isAfter( //pick up after booked drop off

bookedVehicleDates.get(i).getDropOff());

boolean checkDropOffAfter = LocalDate.from(newBooking.getDropOff()).isAfter( //drop off after booked drop off

bookedVehicleDates.get(i).getDropOff());

if ((checkPickUpBefore && checkDropOffBefore) || (checkPickUpAfter && checkDropOffAfter)) {

// if both requested pick up and drop off are either before the booked pick up date or after the

// booked drop off date, the vehicle is available for requested period

passedChecks += 1;

}

//if false for at least one, can't book

}

//-------------------

// if (totalBookings>0){

// return passedChecks == totalBookings; //if all the bookings don't interfere with the requested time -> true

//

// } else{

// return true;

//since this else block will run only if there has been at least one previous entry, the above verification isn't required

return passedChecks == totalBookings; //if all the bookings don't interfere with the requested time -> true

}

}

public static BigDecimal getCalculatedRent(BigDecimal dailyCost, Schedule newBooking) {

// have calculation of total cost here

BigDecimal totalCost = BigDecimal.valueOf(0);

Period period = Period.between(newBooking.getPickUp(),newBooking.getDropOff());//difference between the number of days

int noOfDays = period.getDays();

if (noOfDays > 0) {

return dailyCost.multiply(BigDecimal.valueOf(noOfDays)); //dailyCost\*noOfDays

}

return totalCost;

}

//----------------------------->>>>

//Booking methods for Angular GUI

public static boolean createBooking(String plateNo, Schedule newBooking ) {

//used to create a booking as required and add booking info into the system

List<Schedule> bookedVehicleDates = new ArrayList<>(); //used to record pick up & drop off dates of a vehicle

//Only used to store the dates into the bookedVehicles HashMap

boolean availability = checkAvailabilityOfVeh(plateNo, newBooking); //checking whether vehicle is available for booking

if (availability) {

// System.out.println("Vehicle is available for booking");

if (bookedVehicles.containsKey(plateNo)) {

bookedVehicleDates = bookedVehicles.get(plateNo); //getting recorded bookings into temporary list

}

bookedVehicleDates.add(newBooking); //adding the newly booked dates to the list of bookings.

WestminsterRentalVehicleManager.bookedVehicles.put(plateNo, (ArrayList) bookedVehicleDates); //adding all booked vehicles to bookedVehicles HashMap

return true;

} else {

//vehicle isn't available to be book

return false;

}

}

//``````~~~~~~~~~~~~~~~~~~~``````

public static boolean checkAvailabilityOfVeh(String plateNo, Schedule newBooking ) {

//used to check for the availability of a chosen vehicle

String plateNoOfChosen = plateNo; //The plate number of the chosen vehicle

if (!WestminsterRentalVehicleManager.bookedVehicles.containsKey(plateNoOfChosen)) {

return true; //vehicle is not booked

} else {

List<Schedule> bookedVehicleDates = new ArrayList<>(); //used to record pick up & drop off dates of a vehicle

bookedVehicleDates = bookedVehicles.get(plateNoOfChosen); //getting recorded bookings into temporary list

//Only used to get each of the dates from the bookedVehicles HashMap Values

int totalBookings = bookedVehicles.get(plateNoOfChosen).size();

int passedChecks = 0;

for (int i = 0; i < totalBookings; i++) {

boolean checkPickUpBefore = LocalDate.from(newBooking.getPickUp()).isBefore( //pick up before booked pickup

bookedVehicleDates.get(i).getPickUp());

boolean checkDropOffBefore = LocalDate.from(newBooking.getDropOff()).isBefore( //drop off before booked pick up

bookedVehicleDates.get(i).getPickUp());

boolean checkPickUpAfter = LocalDate.from(newBooking.getPickUp()).isAfter( //pick up after booked drop off

bookedVehicleDates.get(i).getDropOff());

boolean checkDropOffAfter = LocalDate.from(newBooking.getDropOff()).isAfter( //drop off after booked drop off

bookedVehicleDates.get(i).getDropOff());

if ((checkPickUpBefore && checkDropOffBefore) || (checkPickUpAfter && checkDropOffAfter)) {

// if both requested pick up and drop off are either before the booked pick up date or after the

// booked drop off date, the vehicle is available for requested period

passedChecks += 1;

}

//if false for at least one, can't book

}

//since this else block will run only if there has been at least one previous entry, the above verification isn't required

return passedChecks == totalBookings; //if all the bookings don't interfere with the requested time -> true

}

}

}

/\*

References:

Current Date & Time

https://stackoverflow.com/questions/833768/java-code-for-getting-current-time

https://docs.oracle.com/javase/1.5.0/docs/api/java/text/SimpleDateFormat.html#month

https://www.javatpoint.com/java-get-current-date

Java8DateTimeExamples.java

https://gist.github.com/mscharhag/9195718

\*/

### WestminsterRentalVehicleManager

package lk.dinuka.VehicleRentalSystem.Controller;

import lk.dinuka.VehicleRentalSystem.Model.Car;

import lk.dinuka.VehicleRentalSystem.Model.Motorbike;

import lk.dinuka.VehicleRentalSystem.Model.RentalVehicleManager;

import lk.dinuka.VehicleRentalSystem.Model.Vehicle;

import lk.dinuka.VehicleRentalSystem.View.GUI;

import java.io.\*;

import java.math.BigDecimal;

import java.nio.file.DirectoryNotEmptyException;

import java.nio.file.Files;

import java.nio.file.NoSuchFileException;

import java.nio.file.Paths;

import java.util.\*;

public class WestminsterRentalVehicleManager implements RentalVehicleManager {

private static Scanner scanInput = new Scanner(System.in);

protected static HashMap<String, Vehicle> allVehicles = new HashMap<>(); //used to check whether the plate No already exists in the system

protected static List<Vehicle> vehiclesInSystem = new ArrayList<>(); //used for sorting and printing. protected: making sure that customers can't modify the vehicles in the system

public static HashMap<String, ArrayList> bookedVehicles = new HashMap<>(); //used to record booked vehicles (plateNo, ArrayList of Schedules)

public static List<Vehicle> getVehiclesInSystem() { //accessed in GUI

return vehiclesInSystem;

}

public static HashMap<String, Vehicle> getAllVehicles() {

return allVehicles;

}

private static String plateNo;

private static String make;

private static String model;

private static String engineCapacity;

private static double dailyCostD;

private static BigDecimal dailyCostBigD;

private static String startType;

private static double wheelSize;

private static String transmission;

private static boolean hasAirCon;

private static String type;

private static boolean replaceVeh; //used to check whether vehicle data is being added or edited

@Override

public void addVehicle() { //add vehicle into system

//Pointless the getting the inputs in the console because edit option is in the add option (will have to repeat code)

if (Vehicle.getCount() <= MAX\_VEHICLES) { //checking whether the vehicles existing in the system has occupied all the available parking lots

int typeSelection;

do {

System.out.println("\nChoose the type of Vehicle to be added:");

System.out.println("1) Car\n2) Motorbike");

System.out.print(">");

intInputValidation();

typeSelection = scanInput.nextInt();

scanInput.nextLine(); //to consume the rest of the line

} while (!(typeSelection == 1 || typeSelection == 2));

System.out.println("\nEnter Plate No:");

System.out.print(">");

plateNo = scanInput.nextLine();

if (allVehicles.containsKey(plateNo)) {

System.out.println("This Plate No. exists in the system.");

System.out.println(); //to keep space for clarity

replaceVeh = false;

printListForEdit(); //display information of vehicle

System.out.println(); //to keep space for clarity

System.out.println("Do u want to edit information related to this vehicle?");

System.out.print(">");

boolean edit = yesOrNo();

if (edit) {

replaceVeh = true;

//remove vehicle from db

DatabaseController.deleteFromSystemDB(plateNo);

addInfo(typeSelection); //add information related to a Vehicle of identified plateNo.

deleteFile(); //deleting existing file

save(); //saving info in file

API.getAllVehiclesToFront(); //update vehicles in front end

} else {

System.out.println(); //keeps space and goes back to main menu

}

} else {

addInfo(typeSelection); //add information related to a Vehicle of identified plateNo.

save(); //saving info in file

API.getAllVehiclesToFront(); //update vehicles in front end

}

} else {

System.out.println("There are no available spaces. 50 vehicles have been added!");

}

}

@Override

public void deleteVehicle() { //delete vehicle by entering plate no. of vehicle

System.out.println("Enter the plate number of the vehicle that u desire to delete:");

System.out.print(">"); //get plateNo from user to choose vehicle to be deleted

String searchNo = scanInput.nextLine();

if (allVehicles.containsKey(searchNo)) {

Vehicle vehicleToBeDeleted = allVehicles.get(searchNo);

type = vehicleToBeDeleted.getType();

System.out.println("\nA " + type + " has been deleted from the system.");

System.out.println("The details of the vehicle that was deleted: " + vehicleToBeDeleted.toString()); //displaying information of deleted vehicle

vehiclesInSystem.remove(vehicleToBeDeleted);

allVehicles.remove(searchNo);

Vehicle.count -= 1; //decreasing the number of vehicles from the system by one

//Deleting from noSQL Database

DatabaseController.deleteFromSystemDB(searchNo);

System.out.println("There are " + (MAX\_VEHICLES - Vehicle.getCount()) + " parking lots left in the garage.");

save(); //save changes to file

API.getAllVehiclesToFront(); //update vehicles in front end

} else {

System.out.println("There's no vehicle related to the Plate No: " + searchNo);

}

}

@Override

public void printList() { //prints list of vehicles in the system

Collections.sort(vehiclesInSystem); //sort vehicles alphabetically, according to make

// print the plate number, the type of vehicle (car/ van/ motorbike).

String leftAlignFormat = "| %-15s | %-12s |%n";

System.out.format("+-----------------+--------------+%n");

System.out.format("| Plate ID | Type |%n");

System.out.format("+-----------------+--------------+%n");

for (Vehicle item : vehiclesInSystem) {

if (item instanceof Car) {

System.out.format(leftAlignFormat, item.getPlateNo(), "Car");

} else if (item instanceof Motorbike) {

System.out.format(leftAlignFormat, item.getPlateNo(), "Motorbike");

}

}

System.out.println("+--------------------------------+");

}

@Override

public void save() { //saves the information of vehicles entered into the system

//Rewrite the file every time a change is made.

deleteFile(); //delete existing file

try { //creating the file

File myFile = new File("allVehicles.txt");

myFile.createNewFile();

// System.out.println("\nFile created: " + myFile.getName());

FileWriter soldFile = new FileWriter("allVehicles.txt", true);

soldFile.write(String.format("+-----------------+---------------+--------------+----------------+---------------+-----------+--------------+--------+-----------------+------------+%n"));

soldFile.write(String.format("| Plate ID | Make | Model | Engine Capacity| Daily Cost(£) | Type | transmission | AirCon | Start type | Wheel Size |%n"));

soldFile.write(String.format("+-----------------+---------------+--------------+----------------+---------------+-----------+--------------+--------+-----------------+------------+%n"));

// soldFile.write(System.getProperty("line.separator")); //line break

String leftAlignFormat2 = "| %-15s | %-13s | %-12s | %-14s | %-13s | %-9s | %-12s | %-6s | %-15s | %-10s |%n";

//writing into the file

for (Vehicle veh : vehiclesInSystem) {

if (veh instanceof Motorbike) {

soldFile.write(String.format(leftAlignFormat2, veh.getPlateNo(), veh.getMake(), veh.getModel(), veh.getEngineCapacity(),

veh.getDailyCost(), veh.getType(), " - ", " - ", ((Motorbike) veh).getStartType(), ((Motorbike) veh).getWheelSize()));

} else {

soldFile.write(String.format(leftAlignFormat2, veh.getPlateNo(), veh.getMake(), veh.getModel(), veh.getEngineCapacity(),

veh.getDailyCost(), veh.getType(), ((Car) veh).getTransmission(), ((Car) veh).isHasAirCon(), " - ", " - "));

}

soldFile.write(System.getProperty("line.separator")); //line break

}

soldFile.write(String.format("+-----------------+---------------+--------------+----------------+---------------+-----------+--------------+--------+-----------------+------------+%n"));

soldFile.close();

} catch (IOException e) {

System.out.println("\nAn error occurred.");

e.printStackTrace();

}

}

@Override

public void viewGUI() { //opens a chosen GUI

int guiSelection;

do {

System.out.println("\nChoose the required GUI:");

System.out.println("1) Angular\n2) JavaFX");

System.out.print(">");

intInputValidation();

guiSelection = scanInput.nextInt();

scanInput.nextLine(); //to consume the rest of the line

} while (!(guiSelection == 1 || guiSelection == 2));

if (guiSelection == 1) { // Angular GUI

API.getAllVehiclesToFront(); //send vehicles to front end

API.postBookingsFromFront(); //handle booking

API.postAvailabilityFromFront(); //handle availability

//Open Angular GUI in browser

ProcessBuilder builder = new ProcessBuilder("explorer.exe", "http://localhost:4200/");

builder.redirectErrorStream(true);

Process p = null;

try {

p = builder.start();

} catch (IOException e) {

e.printStackTrace();

}

BufferedReader r = new BufferedReader(new InputStreamReader(p.getInputStream()));

String line;

while (true) {

try {

line = r.readLine();

if (line == null) {

break;

}

System.out.println(line);

} catch (IOException e) {

e.printStackTrace();

}

}

} else { //JavaFX GUI

GUI.main(null); //used to open javafx application

}

}

// ---- repeated methods ----

private static void addInfo(int typeSelection) { //method to add information related to a Vehicle of identified plateNo.

if (replaceVeh) {

vehiclesInSystem.remove(allVehicles.get(plateNo)); //removing vehicle from ArrayList, if editing it's information

}

if (typeSelection == 1) { //new Car chosen

addCommonInfo();

type = "Car";

System.out.println("\nEnter the type of transmission:");

System.out.print(">");

transmission = scanInput.nextLine();

System.out.println("\nDoes this car have A/C?");

System.out.print(">");

hasAirCon = yesOrNo();

Vehicle newCar = new Car(plateNo, make, model, engineCapacity, dailyCostBigD, type, transmission, hasAirCon);

allVehicles.put(plateNo, newCar); //adding a car into the allVehicles hashMap

vehiclesInSystem.add(newCar);

//adding new Car to noSQL database

DatabaseController.addToSystemDB(plateNo, make, model, engineCapacity, dailyCostD, type, transmission, hasAirCon);

System.out.println(newCar); //displaying added vehicle

} else if (typeSelection == 2) { //new Motorbike chosen

addCommonInfo();

type = "Motorbike";

System.out.println("\nEnter start type:");

System.out.print(">");

startType = scanInput.nextLine();

System.out.println("\nEnter wheel size:");

System.out.print(">");

doubleInputValidation();

wheelSize = scanInput.nextDouble();

scanInput.nextLine(); //to consume the rest of the line

Vehicle newBike = new Motorbike(plateNo, make, model, engineCapacity, dailyCostBigD, type, startType, wheelSize);

allVehicles.put(plateNo, newBike); //adding a motorbike into the allVehicles hashMap

vehiclesInSystem.add(newBike);

//adding new Bike to noSQL database

DatabaseController.addToSystemDB(plateNo, make, model, engineCapacity, dailyCostD, type, startType, wheelSize);

System.out.println(newBike); //displaying added vehicle

}

System.out.println("\nThere are " + (MAX\_VEHICLES - Vehicle.getCount()) + " parking lots left, to park vehicles.");

}

private static void addCommonInfo() { //common information related to Car & Motorbike in addVehicle

System.out.println("\nEnter Make:");

System.out.print(">");

make = scanInput.nextLine();

System.out.println("\nEnter Model:");

System.out.print(">");

model = scanInput.nextLine();

System.out.println("\nEnter Engine Capacity (in CC):");

System.out.print(">");

engineCapacity = scanInput.nextLine();

System.out.println("\nEnter Daily cost (in £):");

System.out.print(">$");

doubleInputValidation();

dailyCostD = scanInput.nextDouble();

dailyCostBigD = BigDecimal.valueOf(dailyCostD); //converting double to BigDecimal, to use for calculations

scanInput.nextLine(); //to consume the rest of the line

}

public static void printListForEdit() {

//print information of vehicle when asked whether to edit

System.out.println("Make: " + allVehicles.get(plateNo).getMake());

System.out.println("Model: " + allVehicles.get(plateNo).getModel());

System.out.println("Engine Capacity: " + allVehicles.get(plateNo).getEngineCapacity());

System.out.println("Daily Cost (in £): " + allVehicles.get(plateNo).getDailyCost());

System.out.println("Type: " + allVehicles.get(plateNo).getType());

if (allVehicles.get(plateNo) instanceof Car) {

System.out.println("Transmission: " + ((Car) allVehicles.get(plateNo)).getTransmission());

System.out.println("Has Air Conditioning: " + ((Car) allVehicles.get(plateNo)).isHasAirCon());

} else {

System.out.println("Start Type: " + ((Motorbike) allVehicles.get(plateNo)).getStartType());

System.out.println("Wheel Size: " + ((Motorbike) allVehicles.get(plateNo)).getWheelSize());

}

}

private static boolean yesOrNo() { //gets yes/ no input

while (!scanInput.hasNextBoolean()) { //check whether this works as expected!!!!!!!!!!!

String inputYN = scanInput.nextLine().toLowerCase();

if (inputYN.equals("y") || inputYN.equals("yes")) {

return true;

} else if (inputYN.equals("n") || inputYN.equals("no")) {

return false;

} else {

System.out.println("Invalid input. Please try again.");

System.out.print(">");

}

}

return false; //won't reach this point (added to get rid of the missing return statement error)

}

private static void intInputValidation() { //validating integer input

while (!scanInput.hasNextInt()) {

System.out.println("Only integer numbers are allowed! Please provide a valid input"); //error handling message for characters other than integers

scanInput.next(); //removing incorrect input entered

}

}

private static void doubleInputValidation() { //validating double input

while (!scanInput.hasNextDouble()) {

System.out.println("Only numbers are allowed! Please provide a valid input"); //error handling message for characters other than integers

scanInput.next(); //removing incorrect input entered

}

}

private static void deleteFile() { //deleting file, if exists (When vehicle is added/ deleted/ edited)

try {

Files.deleteIfExists(Paths.get("C:\\Users\\Dell XPS15\\Documents\\IIT Work\\L5\\OOP\\Coursework 01\\OOP-CW\\OOP-CW+\\allVehicles.txt"));

} catch (NoSuchFileException e) {

System.out.println("No such file/directory exists");

} catch (DirectoryNotEmptyException e) {

System.out.println("Directory is not empty.");

} catch (IOException e) {

System.out.println("Invalid permissions.");

}

}

}

/\*

References:

Open URL in browser (Angular GUI)

https://alvinalexander.com/blog/post/java/how-open-read-url-java-url-class-example-code

Java Big Decimal

https://www.geeksforgeeks.org/bigdecimal-class-java/

https://stackoverflow.com/questions/27409718/java-reading-multiple-objects-from-a-file-as-they-were-in-an-array

replacing hashMap value

https://stackoverflow.com/questions/35297537/difference-between-replace-and-put-for-hashmap

-------

https://stackoverflow.com/questions/13102045/scanner-is-skipping-nextline-after-using-next-or-nextfoo

https://www.callicoder.com/java-arraylist/

https://stackoverflow.com/questions/48720936/java-enhanced-for-loop-for-arraylist-with-custom-object

To open GUI from console

https://stackoverflow.com/questions/2550310/can-a-main-method-of-class-be-invoked-from-another-class-in-java

File handling

https://www.w3schools.com/java/java\_files.asp

Next line in file handling

https://stackoverflow.com/questions/17716192/insert-line-break-when-writing-to-file

File handling - table format

https://stackoverflow.com/questions/26229140/writing-data-to-text-file-in-table-format

Delete file

https://www.geeksforgeeks.org/delete-file-using-java/

Table display format for print list

https://stackoverflow.com/questions/15215326/how-can-i-create-table-using-ascii-in-a-console

Selling date/time

https://www.javatpoint.com/java-get-current-date

Search for object in ArrayList

https://stackoverflow.com/questions/17526608/how-to-find-an-object-in-an-arraylist-by-property

\*/

## Model

### RentalVehicleManager

package lk.dinuka.VehicleRentalSystem.Model;

public interface RentalVehicleManager {

//constants

int MAX\_VEHICLES = 50;

//methods

void addVehicle();

void deleteVehicle();

void printList();

void save();

void viewGUI();

}

### Vehicle

package lk.dinuka.VehicleRentalSystem.Model;

import java.math.BigDecimal;

import java.util.Objects;

public abstract class Vehicle implements Comparable<Vehicle> {

private String plateNo;

private String make;

private String model;

private String engineCapacity;

private BigDecimal dailyCost;

private String type;

public static int count = 0;

public Vehicle(String plateNo, String make, String model, String engineCapacity, BigDecimal dailyCost, String type) {

this.plateNo = plateNo;

this.make = make;

this.model = model;

this.engineCapacity = engineCapacity;

this.dailyCost = dailyCost;

this.type = type;

count++;

}

@Override

public String toString() {

return "Vehicle{" +

"plateNo='" + plateNo + '\'' +

", make='" + make + '\'' +

", model='" + model + '\'' +

", engineCapacity='" + engineCapacity + '\'' +

", dailyCost=" + dailyCost +

", type='" + type + '\'' +

'}';

}

public static int getCount() {

return count;

}

public String getPlateNo() {

return plateNo;

}

public String getMake() {

return make;

}

public String getModel() {

return model;

}

public String getEngineCapacity() {

return engineCapacity;

}

public BigDecimal getDailyCost() {

return dailyCost;

}

public String getType() {

return type;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

Vehicle vehicle = (Vehicle) o;

return Objects.equals(plateNo, vehicle.plateNo) &&

Objects.equals(make, vehicle.make) &&

Objects.equals(model, vehicle.model) &&

Objects.equals(engineCapacity, vehicle.engineCapacity) &&

Objects.equals(dailyCost, vehicle.dailyCost) &&

Objects.equals(type, vehicle.type);

}

@Override

public int hashCode() {

return Objects.hash(plateNo, make, model, engineCapacity, dailyCost, type);

}

@Override

public int compareTo(Vehicle obj) {

return this.make.compareTo(obj.getMake()); //used for sorting vehicle alphabetically according to make

}

}

/\*

References:

https://www.geeksforgeeks.org/comparable-vs-comparator-in-java/

https://beginnersbook.com/2013/12/java-arraylist-of-object-sort-example-comparable-and-comparator/

\*/

### Schedule

package lk.dinuka.VehicleRentalSystem.Model;

import java.time.LocalDate;

import java.util.Objects;

public class Schedule {

private LocalDate pickUp;

private LocalDate dropOff;

public Schedule(LocalDate pick, LocalDate drop) {

this.pickUp = pick;

this.dropOff = drop;

}

public LocalDate getPickUp() {

return pickUp;

}

public LocalDate getDropOff() {

return dropOff;

}

// public String getTime() {

// return time;

// }

// public void setTime() { //getting time at which the booking was made

//

// Calendar cal = Calendar.getInstance();

// SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");

//// String h1 = sdf.format(cal.getTime()) ;

//

// this.time = sdf.format(cal.getTime());

// }

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

Schedule schedule = (Schedule) o;

return Objects.equals(pickUp, schedule.pickUp) &&

Objects.equals(dropOff, schedule.dropOff);

}

@Override

public int hashCode() {

return Objects.hash(pickUp, dropOff);

}

@Override

public String toString() {

return "Schedule{" +

"pickUp=" + pickUp +

", dropOff=" + dropOff +

'}';

}

}

/\*

References:

Java 8 DateTime

https://gist.github.com/mscharhag/9195718

Current time

https://stackoverflow.com/questions/833768/java-code-for-getting-current-time

https://docs.oracle.com/javase/1.5.0/docs/api/java/text/SimpleDateFormat.html#month

https://www.javatpoint.com/java-get-current-date

\*/

### Car

package lk.dinuka.VehicleRentalSystem.Model;

import java.math.BigDecimal;

import java.util.Objects;

public class Car extends Vehicle {

private String transmission;

private boolean hasAirCon;

public Car(String plateNo, String make, String model, String engineCapacity, BigDecimal dailyCost, String type, String transmission, boolean hasAirCon) {

super(plateNo, make, model, engineCapacity, dailyCost, type);

this.transmission = transmission; //making sure that this extra info is added when creating a new Car object

this.hasAirCon = hasAirCon;

}

public String getTransmission() {

return transmission;

}

public void setTransmission(String transmission) {

this.transmission = transmission;

}

public boolean isHasAirCon() {

return hasAirCon;

}

public void setHasAirCon(boolean hasAirCon) {

this.hasAirCon = hasAirCon;

}

@Override

public String toString() {

return super.toString() + " {" +

"transmission='" + transmission + '\'' +

", hasAirCon=" + hasAirCon +

'}';

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

if (!super.equals(o)) return false;

Car car = (Car) o;

return hasAirCon == car.hasAirCon &&

Objects.equals(transmission, car.transmission);

}

@Override

public int hashCode() {

return Objects.hash(super.hashCode(), transmission, hasAirCon);

}

}

### Motorbike

package lk.dinuka.VehicleRentalSystem.Model;

import java.math.BigDecimal;

import java.util.Objects;

public class Motorbike extends Vehicle {

private String startType; //Kick Start or Electric Start

private double wheelSize;

public Motorbike(String plateNo, String make, String model, String engineCapacity, BigDecimal dailyCost, String type, String startType, double wheelSize) {

super(plateNo, make, model, engineCapacity, dailyCost,type);

this.startType = startType; //making sure that this extra info is added when creating a new Motorbike object

this.wheelSize = wheelSize;

}

public String getStartType() {

return startType;

}

public void setStartType(String startType) {

this.startType = startType;

}

public double getWheelSize() {

return wheelSize;

}

public void setWheelSize(double wheelSize) {

this.wheelSize = wheelSize;

}

@Override

public String toString() {

return super.toString() + " " +

"startType='" + startType + '\'' +

", wheelSize=" + wheelSize +

'}';

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

if (!super.equals(o)) return false;

Motorbike motorbike = (Motorbike) o;

return Double.compare(motorbike.wheelSize, wheelSize) == 0 &&

Objects.equals(startType, motorbike.startType);

}

@Override

public int hashCode() {

return Objects.hash(super.hashCode(), startType, wheelSize);

}

}

## View

### GUI

package lk.dinuka.VehicleRentalSystem.View;

import javafx.application.Application;

import javafx.event.ActionEvent;

import javafx.event.EventHandler;

import javafx.geometry.Insets;

import javafx.scene.Scene;

import javafx.scene.control.\*;

import javafx.scene.control.cell.PropertyValueFactory;

import javafx.scene.layout.HBox;

import javafx.scene.layout.VBox;

import javafx.scene.paint.Color;

import javafx.scene.text.Text;

import javafx.stage.Stage;

import lk.dinuka.VehicleRentalSystem.Controller.DatabaseController;

import lk.dinuka.VehicleRentalSystem.Controller.GUIController;

import lk.dinuka.VehicleRentalSystem.Controller.WestminsterRentalVehicleManager;

import lk.dinuka.VehicleRentalSystem.Model.\*;

import java.util.ArrayList;

public class GUI extends Application {

public static void main(String[] args) {

launch(args);

}

private static ArrayList<Vehicle> searchedVehicles = new ArrayList<>(); //used to pass in searched vehicles into the table

private static ArrayList<Vehicle> searchInSearch = new ArrayList<>(); //used to filter search by Vehicle type

//-----------------------------------------------------//

@Override

public void start(Stage primaryStage) throws Exception {

// Platform.setImplicitExit(false);

primaryStage.setTitle("List of vehicles in system");

TableView tableOfVehicles = new TableView();

//Creating columns to be added to the table

TableColumn<String, Vehicle> plateNoColumn = new TableColumn<>("Plate No");

plateNoColumn.setCellValueFactory(new PropertyValueFactory<>("plateNo"));

TableColumn<String, Vehicle> makeColumn = new TableColumn<>("Make");

makeColumn.setCellValueFactory(new PropertyValueFactory<>("make"));

TableColumn<String, Vehicle> modelColumn = new TableColumn<>("Model");

modelColumn.setCellValueFactory(new PropertyValueFactory<>("model"));

TableColumn<String, Vehicle> engineCapacityColumn = new TableColumn<>("Engine Capacity(CC)");

engineCapacityColumn.setCellValueFactory(new PropertyValueFactory<>("engineCapacity"));

engineCapacityColumn.setMinWidth(130);

TableColumn<String, Vehicle> dailyCostColumn = new TableColumn<>("Daily Cost(£)");

dailyCostColumn.setCellValueFactory(new PropertyValueFactory<>("dailyCost"));

dailyCostColumn.setMinWidth(110);

TableColumn<String, Vehicle> typeColumn = new TableColumn<>("Type");

typeColumn.setCellValueFactory(new PropertyValueFactory<>("type"));

TableColumn<String, Vehicle> transmissionColumn = new TableColumn<>("Transmission");

transmissionColumn.setCellValueFactory(new PropertyValueFactory<>("transmission"));

transmissionColumn.setMinWidth(130);

TableColumn<String, Vehicle> hasAirConColumn = new TableColumn<>("Has Air Conditioning");

hasAirConColumn.setCellValueFactory(new PropertyValueFactory<>("hasAirCon"));

hasAirConColumn.setMinWidth(180);

TableColumn<String, Vehicle> startTypeColumn = new TableColumn<>("Start Type");

startTypeColumn.setCellValueFactory(new PropertyValueFactory<>("startType"));

startTypeColumn.setMinWidth(120);

TableColumn<String, Vehicle> wheelSizeColumn = new TableColumn<>("Wheel Size");

wheelSizeColumn.setCellValueFactory(new PropertyValueFactory<>("wheelSize"));

wheelSizeColumn.setMinWidth(130);

tableOfVehicles.getColumns().addAll(plateNoColumn, makeColumn, modelColumn, engineCapacityColumn, dailyCostColumn,

typeColumn, transmissionColumn, hasAirConColumn, startTypeColumn, wheelSizeColumn); //adding all the columns to the table

tableOfVehicles.getItems().addAll(WestminsterRentalVehicleManager.getVehiclesInSystem()); //adding all the vehicles in the available

// in the vehiclesInSystem ArrayList

searchedVehicles.addAll(WestminsterRentalVehicleManager.getVehiclesInSystem()); //to get filter by vehicle type to work before searching for a Make

//---------------------------------------------------

HBox searchSection = new HBox();

searchSection.setMinWidth(220);

searchSection.getChildren().add(new Label("Search Make:"));

TextField makeSearch = new TextField();

searchSection.getChildren().add(makeSearch);

Button searchClick = new Button("Search");

searchSection.getChildren().add(searchClick);

Button resetClick = new Button("Reset");

searchSection.getChildren().add(resetClick);

// VBox filterSection = new VBox(new Label("Filter By"));

HBox filterType = new HBox(new Label("Filter Type:"));

Button filterCarClick = new Button("Cars");

filterType.getChildren().add(filterCarClick);

Button filterBikeClick = new Button("Motorbikes");

filterType.getChildren().add(filterBikeClick);

filterType.setPadding(new Insets(10, 0, 0, 0));

// HBox filterEngineCap = new HBox(new Label("Engine Capacity:"));

filterType.setMinWidth(200);

// filterSection.getChildren().addAll(filterType);

VBox allSearchFilter = new VBox(searchSection, filterType);

allSearchFilter.setPadding(new Insets(20, 0, 20, 20));

//---------------------------------------------------

VBox bookingSection = new VBox();

HBox allDates = new HBox();

//pick up date entry section

HBox pickUpDateSec = new HBox(new Label("Pick Up:"));

// ---------------->>>>>>>

DatePicker pickDatePicker = new DatePicker();

// pickUpDateSec.getChildren().addAll(dayPickUp, monthPickUp, yearPickUp);

pickUpDateSec.getChildren().addAll(pickDatePicker);

//drop off date entry section

HBox dropOffDateSec = new HBox();

Label dropOffLabel = new Label("Drop Off:");

// ---------------->>>>>>>

DatePicker dropDatePicker = new DatePicker();

// dropOffDateSec.getChildren().addAll(dropOffLabel, dayDropOff, monthDropOff, yearDropOff);

dropOffDateSec.getChildren().addAll(dropOffLabel, dropDatePicker);

Button availabilityCheck = new Button("Check Availability");

allDates.setSpacing(10.0);

Button bookOnClick = new Button("Book");

// bookOnClick.setAlignment(right);

Text checkBookedStatus = new Text();

Text bookStatusText = new Text();

Text displayTotalCost = new Text();

VBox buttonsForBooking = new VBox();

buttonsForBooking.getChildren().addAll(availabilityCheck, bookOnClick);

buttonsForBooking.setSpacing(5.0);

allDates.getChildren().addAll(pickUpDateSec, dropOffDateSec, buttonsForBooking);

bookingSection.getChildren().addAll(allDates, checkBookedStatus, bookStatusText, displayTotalCost);

bookingSection.setPadding(new Insets(20, 0, 20, 20));

//---------------------------------------------------

VBox parent = new VBox(allSearchFilter, tableOfVehicles, bookingSection);

Scene newScene = new Scene(parent);

primaryStage.setScene(newScene);

primaryStage.show();

primaryStage.setAlwaysOnTop(true); //open the application on top of intelliJ

//---------------//------------------//---------------------//-----------------------//

//Button actions

searchClick.setOnAction(new EventHandler<ActionEvent>() { //actions when search button is clicked

@Override

public void handle(ActionEvent event) {

String vehMakeSearch = makeSearch.getText(); //getting Make to be searched

searchedVehicles.clear(); //clearing previous search results from ArrayList

for (Vehicle searchVeh : WestminsterRentalVehicleManager.getVehiclesInSystem()) {

if (searchVeh.getMake().equals(vehMakeSearch)) {

searchedVehicles.add(searchVeh); //adding vehicles that have matching makes as searched into ArrayList

}

}

// System.out.println(searchedVehicles); //to check

tableOfVehicles.getItems().clear(); //clearing table

tableOfVehicles.getItems().addAll(searchedVehicles);

}

});

resetClick.setOnAction(new EventHandler<ActionEvent>() { //actions when reset button is clicked

@Override

public void handle(ActionEvent event) {

searchedVehicles.clear(); //resetting search to all Vehicles

searchedVehicles.addAll(WestminsterRentalVehicleManager.getVehiclesInSystem());

tableOfVehicles.getItems().clear(); //reseting display to all Vehicles

tableOfVehicles.getItems().addAll(WestminsterRentalVehicleManager.getVehiclesInSystem());

makeSearch.setText("");

}

});

filterCarClick.setOnAction(new EventHandler<ActionEvent>() { //actions when Filter Cars button is clicked

@Override

public void handle(ActionEvent event) {

searchInSearch.clear();

for (Vehicle searchVeh : searchedVehicles) {

if (searchVeh instanceof Car) {

searchInSearch.add(searchVeh); //adding vehicles that are of Type Car into ArrayList

}

}

// System.out.println(searchInSearch); //to check

tableOfVehicles.getItems().clear(); //clearing table

tableOfVehicles.getItems().addAll(searchInSearch);

}

});

filterBikeClick.setOnAction(new EventHandler<ActionEvent>() { //actions when Filter Motorbikes button is clicked

@Override

public void handle(ActionEvent event) {

searchInSearch.clear();

for (Vehicle searchVeh : searchedVehicles) {

if (searchVeh instanceof Motorbike) {

searchInSearch.add(searchVeh); //adding vehicles that are of Type Car into ArrayList

}

}

// System.out.println(searchInSearch); //to check

tableOfVehicles.getItems().clear(); //clearing table

tableOfVehicles.getItems().addAll(searchInSearch);

}

});

//---------------------------------------------------

availabilityCheck.setOnAction(new EventHandler<ActionEvent>() { //actions when Availability check button is clicked

@Override

public void handle(ActionEvent event) {

try {

if (tableOfVehicles.getSelectionModel().getSelectedItem() != null) {

Vehicle chosenVeh = (Vehicle) tableOfVehicles.getSelectionModel().getSelectedItem(); //selected vehicle's information

// System.out.println(chosenVeh); //to check whether expected vehicle was chosen

Schedule newBooking = new Schedule(pickDatePicker.getValue(), dropDatePicker.getValue());

boolean availability = GUIController.checkAvailabilityOfVeh(chosenVeh, newBooking);

if (availability) { //vehicle available

checkBookedStatus.setFill(Color.GREEN);

// System.out.println("Vehicle is available for booking.");

checkBookedStatus.setText(chosenVeh.getPlateNo() + " is available for booking.");

} else {

checkBookedStatus.setFill(Color.RED);

// System.out.println("Vehicle isn't available for booking during requested time period.");

checkBookedStatus.setText(chosenVeh.getPlateNo() + " isn't available for booking during requested time period.");

}

} else {

checkBookedStatus.setFill(Color.DARKGRAY);

checkBookedStatus.setText("Please select a vehicle to book.");

bookStatusText.setText(""); //clearing old booking details

displayTotalCost.setText("");

}

} catch (NumberFormatException e) {

checkBookedStatus.setFill(Color.DARKGRAY);

checkBookedStatus.setText("Please enter a valid date in Integer Numbers.");

}

}

});

bookOnClick.setOnAction(new EventHandler<ActionEvent>() { //actions when Book button is clicked

@Override

public void handle(ActionEvent event) {

try {

if (tableOfVehicles.getSelectionModel().getSelectedItem() != null) {

//getting selected vehicle's information

Vehicle chosenVeh = (Vehicle) tableOfVehicles.getSelectionModel().getSelectedItem(); //selected vehicle's information

//down-casted from Object type to Vehicle type

System.out.println(chosenVeh); //to check whether expected vehicle was chosen

Schedule newBooking = new Schedule(pickDatePicker.getValue(), dropDatePicker.getValue());

boolean booked = GUIController.createBooking(chosenVeh, newBooking);

if (booked) {

checkBookedStatus.setFill(Color.GREEN);

// System.out.println("Vehicle is available for booking.");

checkBookedStatus.setText(chosenVeh.getPlateNo() + " is available for booking.");

bookStatusText.setText("Booked vehicle with Plate No: " + chosenVeh.getPlateNo() + " from " +

newBooking.getPickUp() + " to " + newBooking.getDropOff());

displayTotalCost.setText("Total Cost: £" + GUIController.getCalculatedRent(chosenVeh.getDailyCost(), newBooking));

int yearPickUpInput = pickDatePicker.getValue().getYear();

int monthPickUpInput = pickDatePicker.getValue().getMonthValue();

int dayPickUpInput = pickDatePicker.getValue().getDayOfMonth();

int yearDropOffInput = pickDatePicker.getValue().getYear();

int monthDropOffInput = pickDatePicker.getValue().getMonthValue();

int dayDropOffInput = pickDatePicker.getValue().getDayOfMonth();

//addToBookedDB here

DatabaseController.addToBookedDB(chosenVeh.getPlateNo(), yearPickUpInput, monthPickUpInput, dayPickUpInput,

yearDropOffInput, monthDropOffInput, dayDropOffInput);

} else {

//notify the user that the vehicle isn't available for rent during the chosen time period.

checkBookedStatus.setFill(Color.RED);

// System.out.println("Vehicle isn't available for booking during requested time period.");

checkBookedStatus.setText(chosenVeh.getPlateNo() + " isn't available for booking during requested time period.");

bookStatusText.setText("");

displayTotalCost.setText("");

}

} else {

checkBookedStatus.setFill(Color.DARKGRAY);

checkBookedStatus.setText("Please select a vehicle to book.");

}

} catch (NumberFormatException e) {

checkBookedStatus.setFill(Color.DARKGRAY);

checkBookedStatus.setText("Please enter a valid date in Integer Numbers.");

}

}

});

}

}

/\*

References:

https://stackoverflow.com/questions/14169240/getting-integer-values-from-textfield

How to get information of selected row in javafx tableview

https://stackoverflow.com/questions/17388866/getting-selected-item-from-a-javafx-tableview

javafx Datepicker

http://tutorials.jenkov.com/javafx/datepicker.html

Error handling in GUI

https://stackoverflow.com/questions/18711896/how-can-i-prevent-java-lang-numberformatexception-for-input-string-n-a

https://docs.oracle.com/javafx/2/layout/size\_align.htm

Multithreading for GUI

https://code-examples.net/en/q/173180e

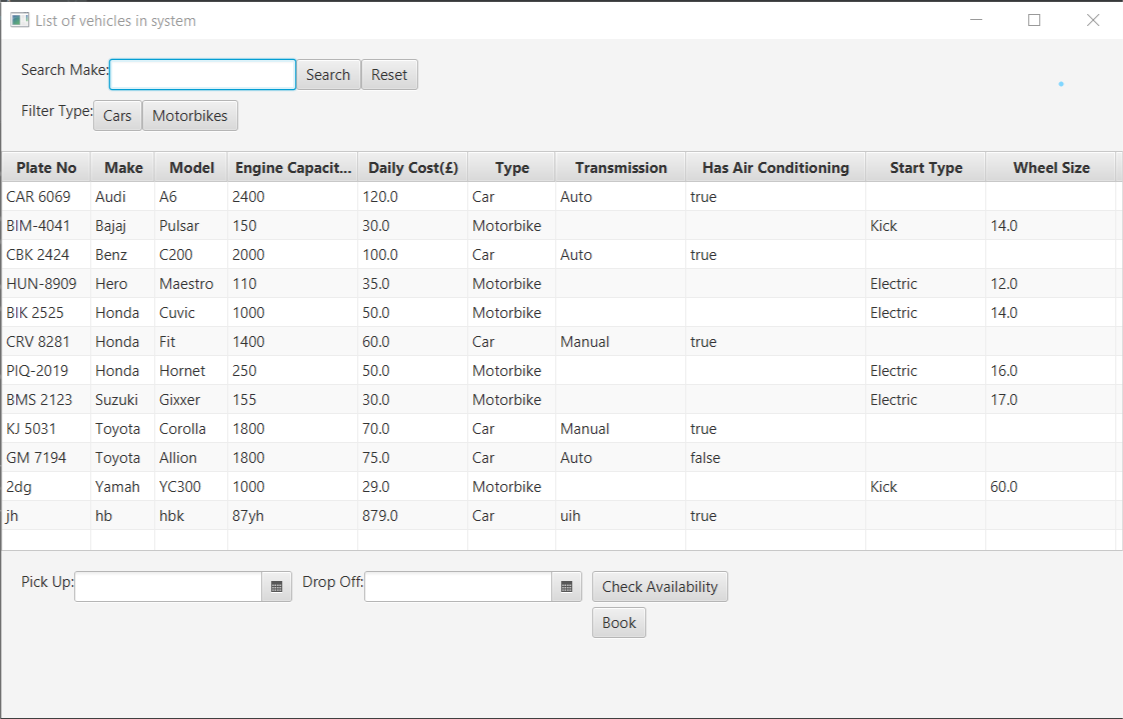
https://stackoverflow.com/questions/24320014/how-to-call-launch-more-than-once-in-java?noredirect=1&lq=1

https://stackoverflow.com/questions/32355335/on-javafx-how-to-hide-stage-without-disposing-it-and-closing-the-application/32356741

\*/

### Screenshots – JavaFX GUI

#### Visualize the list of vehicles



##### Code

@Override

public void start(Stage primaryStage) throws Exception {

// Platform.setImplicitExit(false);

primaryStage.setTitle("List of vehicles in system");

TableView tableOfVehicles = new TableView();

//Creating columns to be added to the table

TableColumn<String, Vehicle> plateNoColumn = new TableColumn<>("Plate No");

plateNoColumn.setCellValueFactory(new PropertyValueFactory<>("plateNo"));

TableColumn<String, Vehicle> makeColumn = new TableColumn<>("Make");

makeColumn.setCellValueFactory(new PropertyValueFactory<>("make"));

TableColumn<String, Vehicle> modelColumn = new TableColumn<>("Model");

modelColumn.setCellValueFactory(new PropertyValueFactory<>("model"));

TableColumn<String, Vehicle> availabilityColumn = new TableColumn<>("Availability");

availabilityColumn.setCellValueFactory(new PropertyValueFactory<>("availability"));

availabilityColumn.setMinWidth(100);

TableColumn<String, Vehicle> engineCapacityColumn = new TableColumn<>("Engine Capacity");

engineCapacityColumn.setCellValueFactory(new PropertyValueFactory<>("engineCapacity"));

engineCapacityColumn.setMinWidth(130);

TableColumn<String, Vehicle> dailyCostColumn = new TableColumn<>("Daily Cost");

dailyCostColumn.setCellValueFactory(new PropertyValueFactory<>("dailyCost"));

dailyCostColumn.setMinWidth(110);

TableColumn<String, Vehicle> typeColumn = new TableColumn<>("Type");

typeColumn.setCellValueFactory(new PropertyValueFactory<>("type"));

TableColumn<String, Vehicle> transmissionColumn = new TableColumn<>("Transmission");

transmissionColumn.setCellValueFactory(new PropertyValueFactory<>("transmission"));

transmissionColumn.setMinWidth(130);

TableColumn<String, Vehicle> hasAirConColumn = new TableColumn<>("Has Air Conditioning");

hasAirConColumn.setCellValueFactory(new PropertyValueFactory<>("hasAirCon"));

hasAirConColumn.setMinWidth(180);

TableColumn<String, Vehicle> startTypeColumn = new TableColumn<>("Start Type");

startTypeColumn.setCellValueFactory(new PropertyValueFactory<>("startType"));

startTypeColumn.setMinWidth(120);

TableColumn<String, Vehicle> wheelSizeColumn = new TableColumn<>("Wheel Size");

wheelSizeColumn.setCellValueFactory(new PropertyValueFactory<>("wheelSize"));

wheelSizeColumn.setMinWidth(130);

tableOfVehicles.getColumns().addAll(plateNoColumn, makeColumn, modelColumn, availabilityColumn, engineCapacityColumn, dailyCostColumn,

typeColumn, transmissionColumn, hasAirConColumn, startTypeColumn, wheelSizeColumn); //adding all the columns to the table

tableOfVehicles.getItems().addAll(WestminsterRentalVehicleManager.getVehiclesInSystem()); //adding all the vehicles in the available

// in the vehiclesInSystem ArrayList

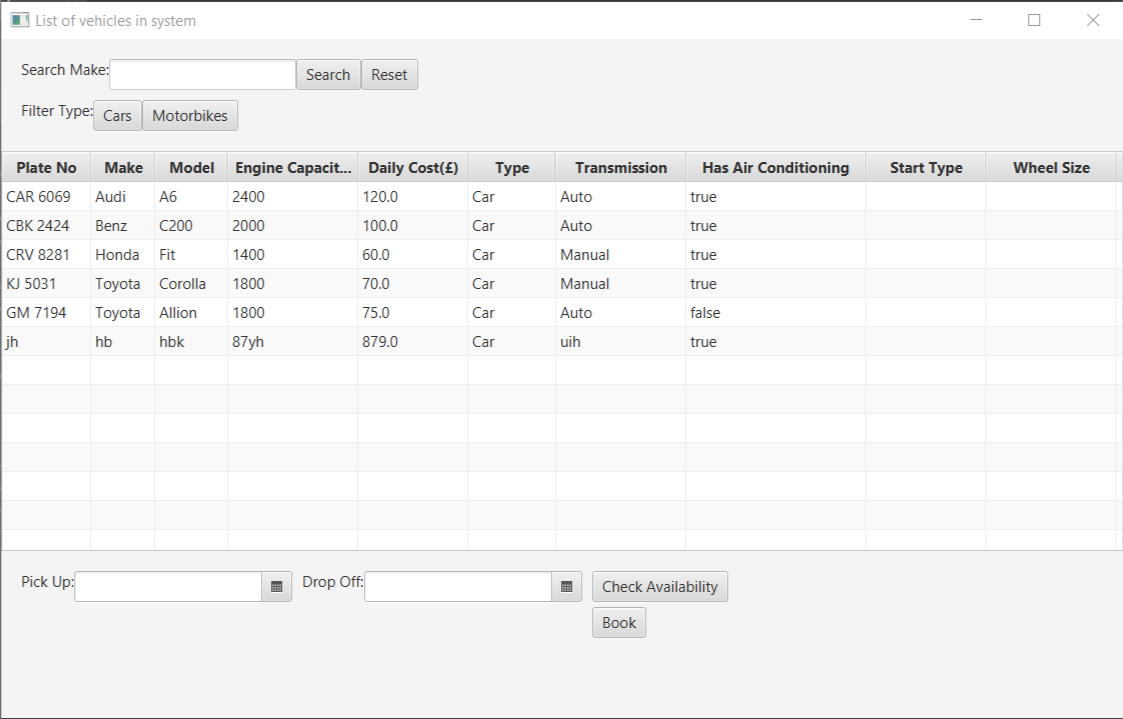
searchedVehicles.addAll(WestminsterRentalVehicleManager.getVehiclesInSystem()); //to get filter by vehicle type to work before searching for a Make

##### Explanation of the functionality

When the GUI is opened, the list of all vehicles is displayed as shown above.

#### Filter vehicles by type

#### Filtered by cars



##### Code

filterCarClick.setOnAction(new EventHandler<ActionEvent>() { //actions when Filter Cars button is clicked

@Override

public void handle(ActionEvent event) {

searchInSearch.clear();

for (Vehicle searchVeh : searchedVehicles) {

if (searchVeh instanceof Car) {

searchInSearch.add(searchVeh); //adding vehicles that are of Type Car into ArrayList

}

}

// System.out.println(searchInSearch); //to check

tableOfVehicles.getItems().clear(); //clearing table

tableOfVehicles.getItems().addAll(searchInSearch);

}

});

filterBikeClick.setOnAction(new EventHandler<ActionEvent>() { //actions when Filter Motorbikes button is clicked

@Override

public void handle(ActionEvent event) {

searchInSearch.clear();

for (Vehicle searchVeh : searchedVehicles) {

if (searchVeh instanceof Motorbike) {

searchInSearch.add(searchVeh); //adding vehicles that are of Type Car into ArrayList

}

}

// System.out.println(searchInSearch); //to check

tableOfVehicles.getItems().clear(); //clearing table

tableOfVehicles.getItems().addAll(searchInSearch);

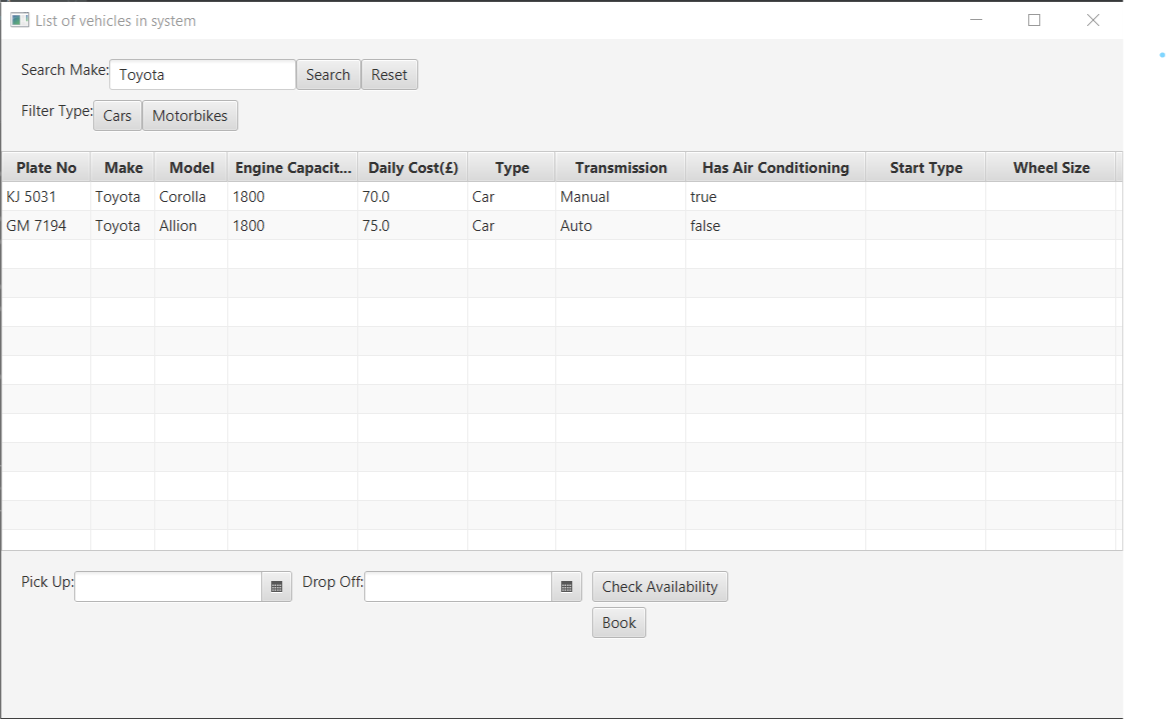
}

});

##### Explanation of the functionality

When the “Filter Type: Cars” button is clicked, all the cars in the system are displayed in the table. Similarly, it works for Motorbikes as well.

#### Filter vehicles by make



##### Code

searchClick.setOnAction(new EventHandler<ActionEvent>() { //actions when search button is clicked

@Override

public void handle(ActionEvent event) {

String vehMakeSearch = makeSearch.getText(); //getting Make to be searched

searchedVehicles.clear(); //clearing previous search results from ArrayList

for (Vehicle searchVeh : WestminsterRentalVehicleManager.getVehiclesInSystem()) {

if (searchVeh.getMake().equals(vehMakeSearch)) {

searchedVehicles.add(searchVeh); //adding vehicles that have matching makes as searched into ArrayList

}

}

// System.out.println(searchedVehicles); //to check

tableOfVehicles.getItems().clear(); //clearing table

tableOfVehicles.getItems().addAll(searchedVehicles);

}

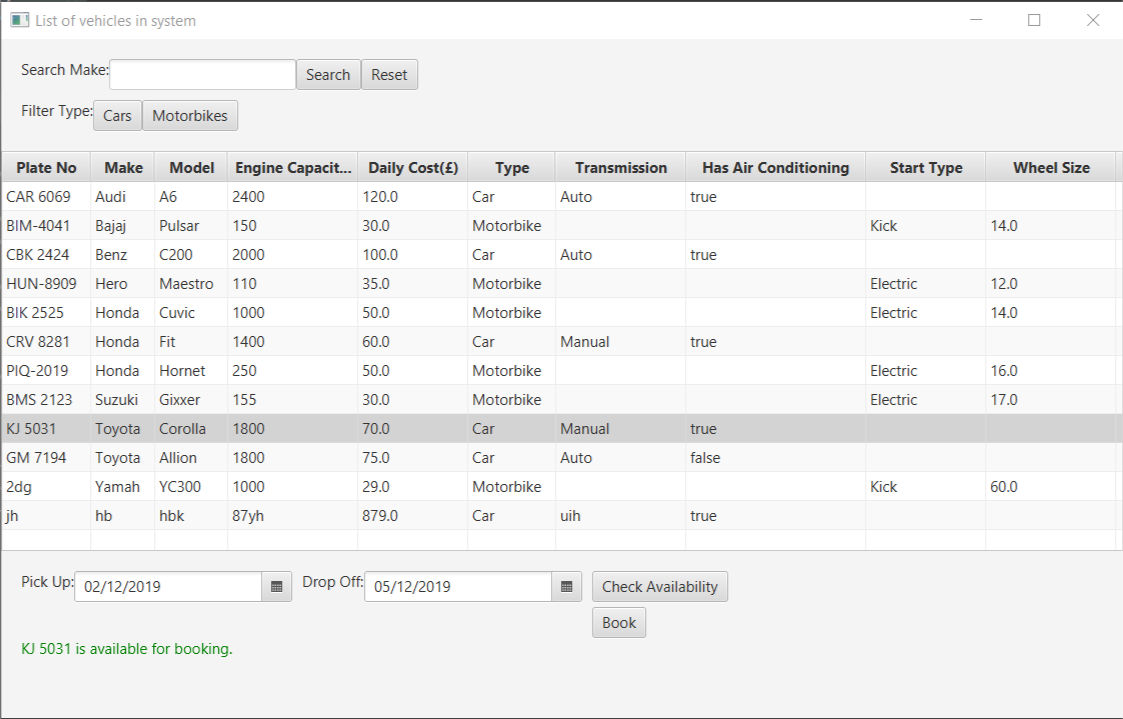
});

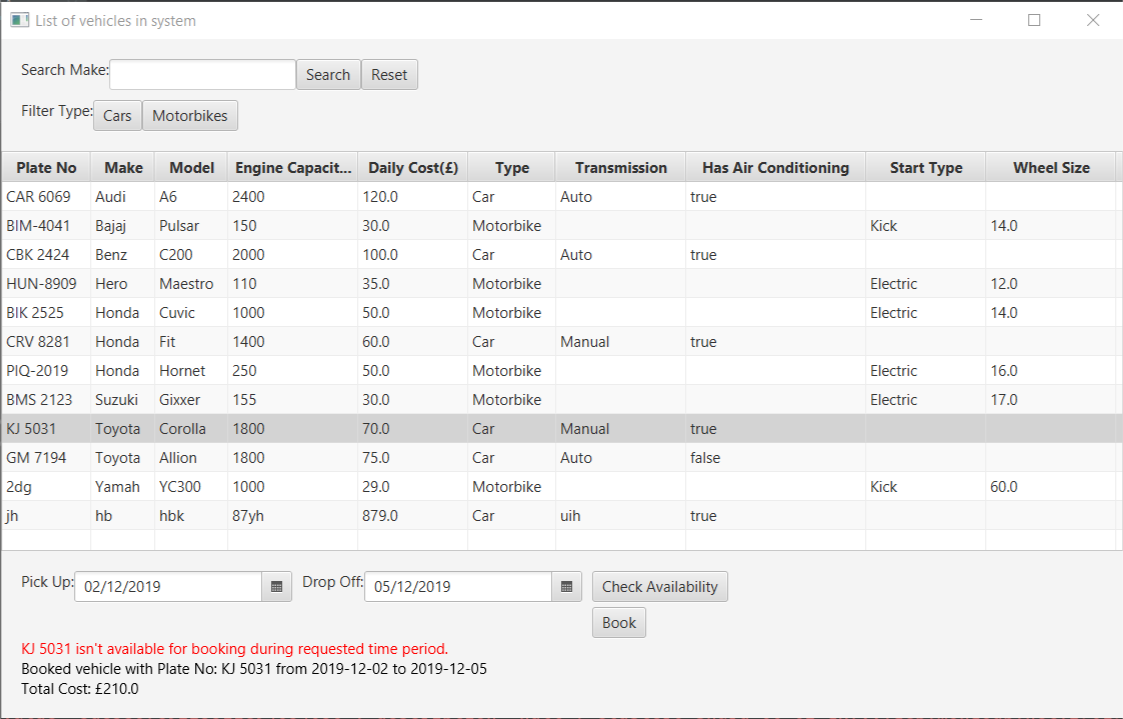
##### Explanation of the functionality

When the required make is typed in and search using the search box, all vehicles related to the searched Make is displayed.

When researched is clicked all Filters and search results are removed, displaying all the vehicles available in the system.

#### Check availability on specific dates





##### Code

availabilityCheck.setOnAction(new EventHandler<ActionEvent>() { //actions when Availability check button is clicked

@Override

public void handle(ActionEvent event) {

try {

if (tableOfVehicles.getSelectionModel().getSelectedItem() != null &&

pickDatePicker.getValue() != null &&

dropDatePicker.getValue() != null){

Vehicle chosenVeh = (Vehicle) tableOfVehicles.getSelectionModel().getSelectedItem(); //selected vehicle's information

// System.out.println(chosenVeh); //to check whether expected vehicle was chosen

Schedule newBooking = new Schedule(pickDatePicker.getValue(), dropDatePicker.getValue());

boolean availability = GUIController.checkAvailabilityOfVeh(chosenVeh, newBooking);

if (availability) { //vehicle available

checkBookedStatus.setFill(Color.GREEN);

// System.out.println("Vehicle is available for booking.");

checkBookedStatus.setText(chosenVeh.getPlateNo() + " is available for booking.");

bookStatusText.setText(""); //clearing old booking details

displayTotalCost.setText("");

} else {

checkBookedStatus.setFill(Color.RED);

// System.out.println("Vehicle isn't available for booking during requested time period.");

checkBookedStatus.setText(chosenVeh.getPlateNo() + " isn't available for booking during requested time period.");

bookStatusText.setText(""); //clearing old booking details

displayTotalCost.setText("");

}

} else{

checkBookedStatus.setFill(Color.DARKGRAY);

checkBookedStatus.setText("Please select a vehicle to book and enter a valid date range.");

bookStatusText.setText(""); //clearing old booking details

displayTotalCost.setText("");

}

} catch (NullPointerException e) {

checkBookedStatus.setFill(Color.DARKGRAY);

checkBookedStatus.setText("Please select a vehicle to book and enter a valid date range.");

}

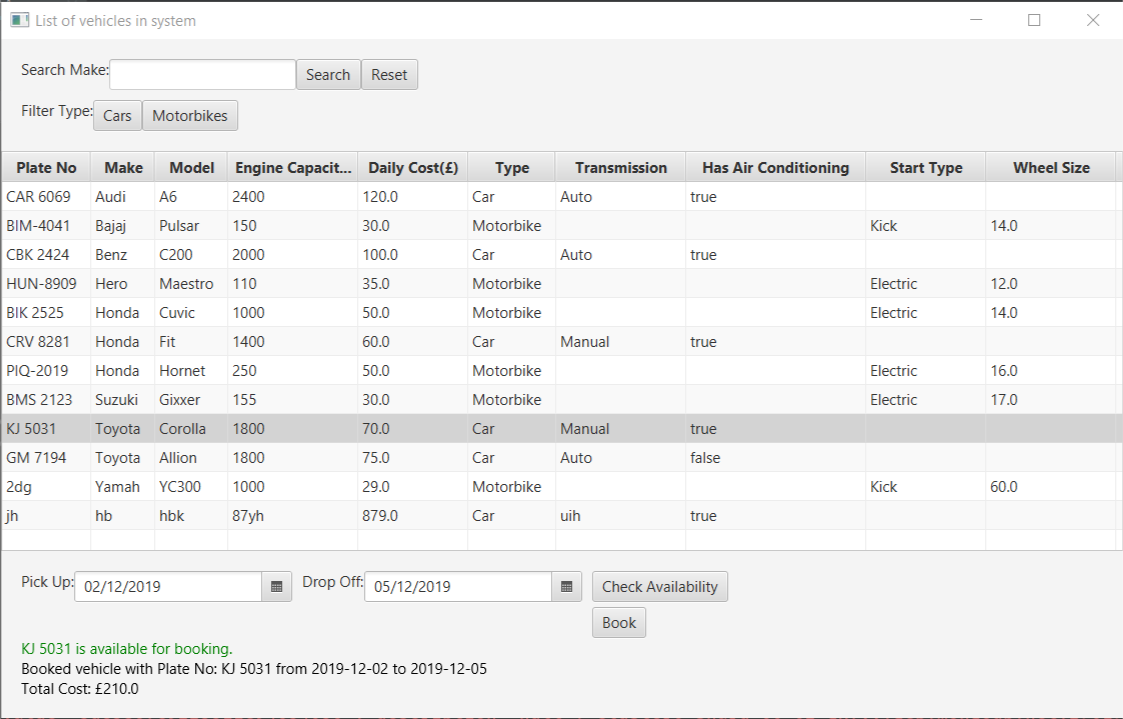
}

});

##### Explanation of the functionality

When “Check Availability” is clicked after a vehicle is selected and pick up & drop off dates are specified, the system will let the user know whether the chosen vehicles is available or not during the requested time period. This is checked when “Book Vehicle” is clicked as well.

#### Book vehicle



##### Code

bookOnClick.setOnAction(new EventHandler<ActionEvent>() { //actions when Book button is clicked

@Override

public void handle(ActionEvent event) {

try {

if (tableOfVehicles.getSelectionModel().getSelectedItem() != null &&

pickDatePicker.getValue() != null &&

dropDatePicker.getValue() != null) {

//getting selected vehicle's information

Vehicle chosenVeh = (Vehicle) tableOfVehicles.getSelectionModel().getSelectedItem(); //selected vehicle's information

//down-casted from Object type to Vehicle type

System.out.println(chosenVeh); //to check whether expected vehicle was chosen

Schedule newBooking = new Schedule(pickDatePicker.getValue(), dropDatePicker.getValue());

boolean booked = GUIController.createBooking(chosenVeh, newBooking);

if (booked) {

checkBookedStatus.setFill(Color.GREEN);

// System.out.println("Vehicle is available for booking.");

checkBookedStatus.setText(chosenVeh.getPlateNo() + " is available for booking.");

bookStatusText.setText("Booked vehicle with Plate No: " + chosenVeh.getPlateNo() + " from " +

newBooking.getPickUp() + " to " + newBooking.getDropOff());

displayTotalCost.setText("Total Cost: £" + GUIController.getCalculatedRent(chosenVeh.getDailyCost(), newBooking));

int yearPickUpInput = pickDatePicker.getValue().getYear();

int monthPickUpInput = pickDatePicker.getValue().getMonthValue();

int dayPickUpInput = pickDatePicker.getValue().getDayOfMonth();

int yearDropOffInput = pickDatePicker.getValue().getYear();

int monthDropOffInput = pickDatePicker.getValue().getMonthValue();

int dayDropOffInput = pickDatePicker.getValue().getDayOfMonth();

//addToBookedDB here

DatabaseController.addToBookedDB(chosenVeh.getPlateNo(), yearPickUpInput, monthPickUpInput, dayPickUpInput,

yearDropOffInput, monthDropOffInput, dayDropOffInput);

} else {

//notify the user that the vehicle isn't available for rent during the chosen time period.

checkBookedStatus.setFill(Color.RED);

// System.out.println("Vehicle isn't available for booking during requested time period.");

checkBookedStatus.setText(chosenVeh.getPlateNo() + " isn't available for booking during requested time period.");

bookStatusText.setText("");

displayTotalCost.setText("");

}

} else {

checkBookedStatus.setFill(Color.DARKGRAY);

checkBookedStatus.setText("Please select a vehicle to book and enter a valid date range.");

}

} catch (NullPointerException e) {

checkBookedStatus.setFill(Color.DARKGRAY);

checkBookedStatus.setText("Please select a vehicle to book and enter a valid date range.");

}

}

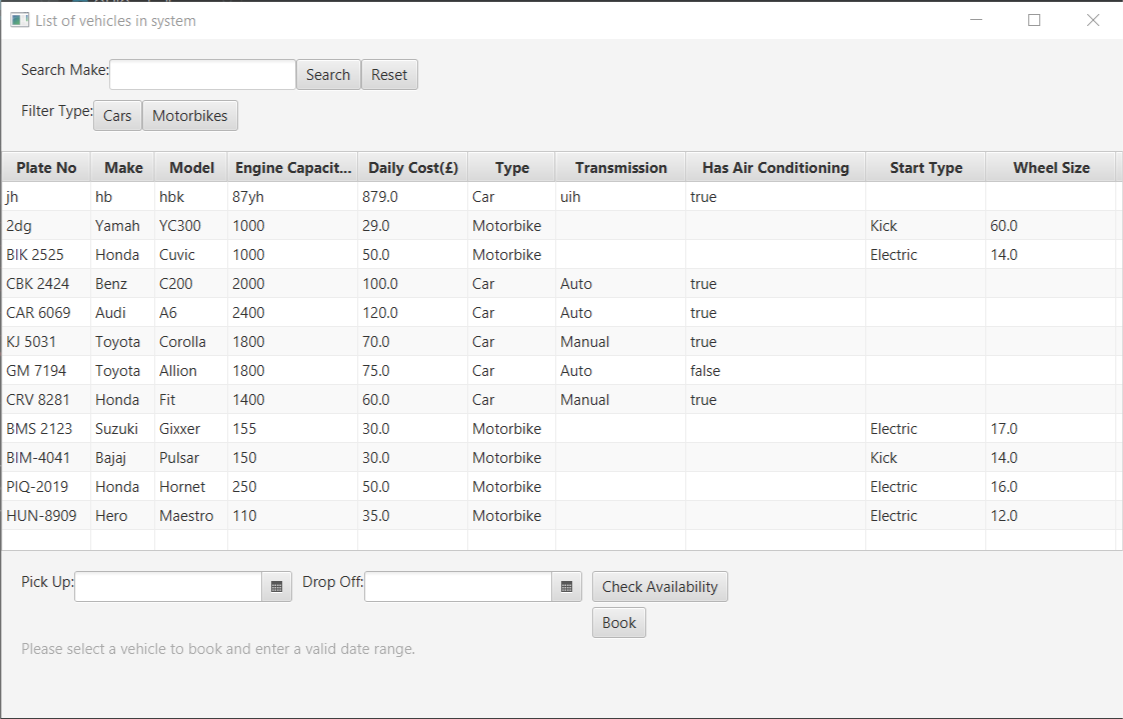
});

##### Explanation of the functionality

When ‘Book’ button is clicked, the system will perform a similar check like “Check Availability” and let the user know that the vehicle was booked for the requested time period.

The total cost will also be displayed below.

#### Required inputs not given – Error Handling



## Angular GUI

### app.component.html

<mat-toolbar color="primary">

<mat-toolbar-row class="header">

<span>Westminster Vehicle Rental Store</span>

</mat-toolbar-row>

</mat-toolbar>

<div class="fullContainer">

<div class = "subHeader">

<mat-form-field>

<input matInput (keyup)="applyFilter($event.target.value)" placeholder="Filter">

</mat-form-field>

<div class="bookingSection">

<mat-form-field>

<input matInput [matDatepicker]="picker1" placeholder="Pick-Up Date" [(ngModel)]="pickUpDate" [min]="today">

<mat-datepicker-toggle matSuffix [for]="picker1"></mat-datepicker-toggle>

<mat-datepicker #picker1></mat-datepicker>

</mat-form-field>

<mat-form-field>

<input matInput [matDatepicker]="picker2" placeholder="Drop-Off Date" [(ngModel)]="dropOffDate" [min]="pickUpDate">

<mat-datepicker-toggle matSuffix [for]="picker2"></mat-datepicker-toggle>

<mat-datepicker #picker2></mat-datepicker>

</mat-form-field>

<button mat-button (click)="checkAvailability()">Check Availability</button>

<button mat-raised-button (click)="bookVehicle()">Book Vehicle</button>

</div>

</div>

<div class="tableContainer mat-elevation-z8">

<table mat-table [dataSource]="dataSource">

<!--- Note that these columns can be defined in any order.

The actual rendered columns are set as a property on the row definition" -->

<ng-container matColumnDef="plateNo">

<th mat-header-cell \*matHeaderCellDef> Plate No. </th>

<td mat-cell \*matCellDef="let element"> {{element.plateNo}} </td>

</ng-container>

<ng-container matColumnDef="make">

<th mat-header-cell \*matHeaderCellDef> Make </th>

<td mat-cell \*matCellDef="let element"> {{element.make}} </td>

</ng-container>

<ng-container matColumnDef="model">

<th mat-header-cell \*matHeaderCellDef> Model </th>

<td mat-cell \*matCellDef="let element"> {{element.model}} </td>

</ng-container>

<ng-container matColumnDef="engineCapacity">

<th mat-header-cell \*matHeaderCellDef> Engine Capacity(CC) </th>

<td mat-cell \*matCellDef="let element"> {{element.engineCapacity}} </td>

</ng-container>

<ng-container matColumnDef="dailyCost">

<th mat-header-cell \*matHeaderCellDef> Daily Cost(£) </th>

<td mat-cell \*matCellDef="let element"> {{element.dailyCost}} </td>

</ng-container>

<ng-container matColumnDef="type">

<th mat-header-cell \*matHeaderCellDef> Type </th>

<td mat-cell \*matCellDef="let element"> {{element.type}} </td>

</ng-container>

<ng-container matColumnDef="transmission">

<th mat-header-cell \*matHeaderCellDef> Transmission </th>

<td mat-cell \*matCellDef="let element"> {{element.transmission}} </td>

</ng-container>

<ng-container matColumnDef="hasAirCon">

<th mat-header-cell \*matHeaderCellDef> Has Air Conditioning </th>

<td mat-cell \*matCellDef="let element"> {{element.hasAirCon}} </td>

</ng-container>

<ng-container matColumnDef="startType">

<th mat-header-cell \*matHeaderCellDef> Start Type </th>

<td mat-cell \*matCellDef="let element"> {{element.startType}} </td>

</ng-container>

<ng-container matColumnDef="wheelSize">

<th mat-header-cell \*matHeaderCellDef> Wheel Size </th>

<td mat-cell \*matCellDef="let element"> {{element.wheelSize}} </td>

</ng-container>

<tr mat-header-row \*matHeaderRowDef="displayedColumns sticky: true"></tr>

<tr mat-row \*matRowDef="let row; columns: displayedColumns;" (click) = "rowClicked(row)" [ngClass]="{'highlight': selectedRowIndex == row.plateNo}">

<!-- if equal, the colour of the row will be changed -->

</tr>

</table>

</div>

</div>

<router-outlet></router-outlet>

### app.component.ts

import { Component, OnInit } from '@angular/core';

import {VehicleService} from '../app/services/vehicle.service';

import {MatTableDataSource} from '@angular/material';

import axios from 'axios';

import {stringify} from 'query-string';

import {MatSnackBar} from '@angular/material/snack-bar';

// --------------------------------------

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.scss']

})

export class AppComponent implements OnInit {

title = 'ng-GUI';

heading: string;

getAllVehicles: any;

allVehicles: any[];

selectedRowIndex = -1;

chosenPlateNo: string;

today = new Date(); // used to get current date

pickUpDate: any;

dropOffDate: any;

displayPickUpDate: any;

displayDropOffDate: any;

responseBook: any;

responseCheck: any = '';

completeMessage: any;

bookingURL: any = 'http://localhost:4567/books';

checkingURL: any = 'http://localhost:4567/checks';

// ---------------

// tslint:disable-next-line: max-line-length

displayedColumns: string[] = ['plateNo', 'make', 'model', 'engineCapacity', 'dailyCost', 'type', 'transmission', 'hasAirCon', 'startType', 'wheelSize'];

dataSource;

// ---------------

constructor(private vehicleService: VehicleService,

private snackBar: MatSnackBar) { } // creating an instance of the service

ngOnInit() {

this.heading = 'Vehicle List';

this.getAllVehicles = this.getServiceData();

// console.log(this.getAllVehicles);

}

getServiceData() { // get data

this.vehicleService.getData().subscribe( // requesting service for information received from the backend

data => {

// console.log(data); // what to do with the received data

// const jsonInfo = JSON.parse(data);

this.allVehicles = data;

this.dataSource = new MatTableDataSource(this.allVehicles);

console.log(this.allVehicles);

}

);

}

postBookingData() { // post plate no & booking data to backend, to book vehicle

const data = { plateNo: this.chosenPlateNo,

yearPickUp: this.pickUpDate.getFullYear(),

monthPickUp: this.pickUpDate.getMonth() + 1, // months are from 0-11

dayPickUp: this.pickUpDate.getDate(),

yearDropOff: this.dropOffDate.getFullYear(),

monthDropOff: this.dropOffDate.getMonth() + 1, // months are from 0-11

dayDropOff: this.dropOffDate.getDate()

};

axios.post(this.bookingURL, stringify(data) , {

headers: {

'Content-Type': 'application/x-www-form-urlencoded'

}

}

)

.then((response) => {

// console.log(response.data);

this.responseBook = response.data;

this.openSnackBarBooking('Close');

})

.catch((error) => {

console.log(error);

});

// const url = this.bookingURL;

// const data = { plateNo: this.chosenPlateNo,

// yearPickUp: this.pickUpDate.getFullYear(),

// monthPickUp: this.pickUpDate.getMonth(),

// dayPickUp: this.pickUpDate.getDate(),

// yearDropOff: this.dropOffDate.getFullYear(),

// monthDropOff: this.dropOffDate.getMonth(),

// dayDropOff: this.dropOffDate.getDate()

// };

// console.log(data); // all the data is getting added into data here

// try {

// const response = await fetch(url, {

// method: 'POST', // or 'PUT'

// body: qs.JSON.stringify(data), // data can be `string` or {object}!

// headers: {

// 'Content-Type': 'application/x-www-form-urlencoded'

// }

// });

// const json = await response.json(); // not receiving response at front end

// console.log(json);

// console.log('Success:', JSON.stringify(json));

// } catch (error) {

// console.error('Error:', error);

// }

// this.vehicleService

// .addBooking(this.chosenPlateNo);

// .subscribe(booking => this.bookings.push(booking));

// }

}

postCheckingData() { // post plate no & booking data to backend, to check availability

const data = { plateNo: this.chosenPlateNo,

yearPickUp: this.pickUpDate.getFullYear(),

monthPickUp: this.pickUpDate.getMonth() + 1, // months are from 0-11

dayPickUp: this.pickUpDate.getDate(),

yearDropOff: this.dropOffDate.getFullYear(),

monthDropOff: this.dropOffDate.getMonth() + 1, // months are from 0-11

dayDropOff: this.dropOffDate.getDate()

};

axios.post(this.checkingURL, stringify(data) , {

headers: {

'Content-Type': 'application/x-www-form-urlencoded'

}

}

)

.then((response) => {

console.log(response.data);

this.responseCheck = response.data;

this.openSnackBarAvailability('Close');

})

.catch((error) => {

console.log(error);

});

}

bookVehicle() {

console.log('book vehicle');

try {

this.postBookingData(); // call post method to book vehicle

} catch {

this.snackBar.open('Make sure that you have chosen the required vehicle and entered the pick up & drop off dates!', 'Close', {

duration: 10000,

panelClass: ['error-snackbar']

});

}

}

checkAvailability() {

console.log('check availability of vehicle');

try {

this.postCheckingData(); // call post method to check availability

} catch {

this.snackBar.open('Make sure that you have chosen the required vehicle and entered the pick up & drop off dates!', 'Close', {

duration: 10000,

panelClass: ['error-snackbar']

});

}

}

// ------------- for table filter function

applyFilter(filterValue: string) {

this.dataSource.filter = filterValue.trim().toLowerCase();

}

rowClicked(row: any): void {

console.log(row);

this.chosenPlateNo = row.plateNo;

console.log(this.chosenPlateNo);

this.selectedRowIndex = row.plateNo;

}

openSnackBarBooking(action: string) { // content to display when a vehicle is requested to be booked

if (this.responseCheck === 'successful') {

const options = { weekday: 'long', year: 'numeric', month: 'long', day: 'numeric' };

this.displayPickUpDate = this.pickUpDate.toLocaleDateString('en-US', options);

this.displayDropOffDate = this.dropOffDate.toLocaleDateString('en-US', options);

const dateRange = ' was booked from '.concat(this.displayPickUpDate).concat(' to ', this.displayDropOffDate);

this.completeMessage = 'The vehicle with Plate No: '.concat(this.chosenPlateNo).concat(dateRange);

console.log(this.completeMessage);

this.snackBar.open(this.completeMessage, action, {

duration: 15000,

panelClass: ['success-snackbar']

});

} else {

const chosenVehicle = 'The vehicle with Plate No: '.concat(this.chosenPlateNo);

this.completeMessage = chosenVehicle.concat(' isn\'t available for booking during the requested time period.');

this.snackBar.open(this.completeMessage, action, {

duration: 10000,

});

}

}

openSnackBarAvailability(action: string) { // content to display when the availability of a vehicle is checked

if (this.responseCheck === 'successful') {

const options = { weekday: 'long', year: 'numeric', month: 'long', day: 'numeric' };

this.displayPickUpDate = this.pickUpDate.toLocaleDateString('en-US', options);

this.displayDropOffDate = this.dropOffDate.toLocaleDateString('en-US', options);

const dateRange = ' is available for booking from '.concat(this.displayPickUpDate).concat(' to ', this.displayDropOffDate);

this.completeMessage = 'The vehicle with Plate No: '.concat(this.chosenPlateNo).concat(dateRange);

console.log(this.completeMessage);

this.snackBar.open(this.completeMessage, action, {

duration: 15000,

panelClass: ['success-snackbar']

});

} else {

const chosenVehicle = 'The vehicle with Plate No: '.concat(this.chosenPlateNo);

this.completeMessage = chosenVehicle.concat(' isn\'t available for booking during the requested time period.');

this.snackBar.open(this.completeMessage, action, {

duration: 10000,

});

}

}

}

/\*References:

https://www.w3schools.com/angular/angular\_tables.asp

https://stackoverflow.com/questions/34973654/angularjs-create-a-table-from-an-array

https://stackoverflow.com/questions/22209117/create-table-from-json-data-with-angularjs-and-ng-repeat/22209337

https://material.angular.io/components/table/overview

Booking confirmation - Snack bar

https://material.angular.io/components/snack-bar/overview

https://stackoverflow.com/questions/56389290/angular-material-snackbar-change-color

Calendar Date input

https://material.angular.io/components/datepicker/overview

Button

https://material.angular.io/components/button/overview

mat table selection

https://stackblitz.com/edit/mat-table-row-click-event

https://stackoverflow.com/questions/52759637/how-to-get-the-selected-row-values-in-table-in-angular

matDatePicker

https://angular-material-nw1brd.stackblitz.io/

https://stackoverflow.com/questions/54828459/angular-material-date-picker-min-and-max-date-validation-messages

https://www.devglan.com/angular/angular-data-table-example

JS Date

https://www.w3schools.com/js/js\_date\_methods.asp

https://stackoverflow.com/questions/3552461/how-to-format-a-javascript-date

https://www.tutorialspoint.com/typescript/typescript\_string\_concat.htm

Typescript passing functions

https://stackoverflow.com/questions/47813442/could-not-able-to-access-property-of-angular-component-inside-it

\*/

### vehicle.service.ts

import { Injectable } from '@angular/core';

import {HttpClient, HttpEvent, HttpRequest} from '@angular/common/http';

import {Observable} from 'rxjs';

// import {catchError} from 'rxjs/operators';

import axios from 'axios';

@Injectable({

providedIn: 'root'

})

export class VehicleService {

// get endpoint

endpointURL: any = 'http://localhost:4567/hello'; // back-end URL

// service makes an HTTP call to the backend

// bookingURL: any = 'http://localhost:4567/books';

constructor(private http: HttpClient) { }

getData(): Observable<any> {

return this.http.get(this.endpointURL); // getting JSON from the backend

}

// POST

}

/\*Reference:

https://angular.io/guide/http

\*/

### styles.scss

html, body { height: 100%; }

body {

margin: 0;

font-family: Roboto, "Helvetica Neue", sans-serif;

background: lavender;

}

.error-snackbar {

color: lightcoral;

}

.success-snackbar{

color: lightgreen;

}

### app.component.scss

.header{

font-family: Roboto,"Helvetica Neue Light","Helvetica Neue",Helvetica,Arial,"Lucida Grande",sans-serif;

font-weight: 300;

// margin: 0;

padding: 28px 15px;

font-size: 20px;

color: #fff;

}

.subHeader{

background: linear-gradient( to top,lavender,white 40%);

}

table {

width: 100%;

// margin: auto;

cursor:pointer;

}

.tableContainer{

width: 95%;

margin: auto;

max-height: 80vh;

overflow:auto;

}

/\* width \*/

.tableContainer::-webkit-scrollbar {

width: 8px;

}

/\* Track \*/

.tableContainer::-webkit-scrollbar-track {

background: #f1f1f1;

}

/\* Handle \*/

.tableContainer::-webkit-scrollbar-thumb {

background: rgb(172, 172, 172);

border-radius: 15px;

}

/\* Handle on hover \*/

::-webkit-scrollbar-thumb:hover {

background: rgba(85, 85, 85, 0.767);

}

.bookingSection{

float: right;

}

.fullContainer{

width:97%;

margin:auto;

}

.highlight{

background: rgb(197, 229, 231);

transition: 0.25s;

}

/\*Reference:

CSS custom scrollbar

https://www.w3schools.com/howto/howto\_css\_custom\_scrollbar.asp

Change colour of table row on selection

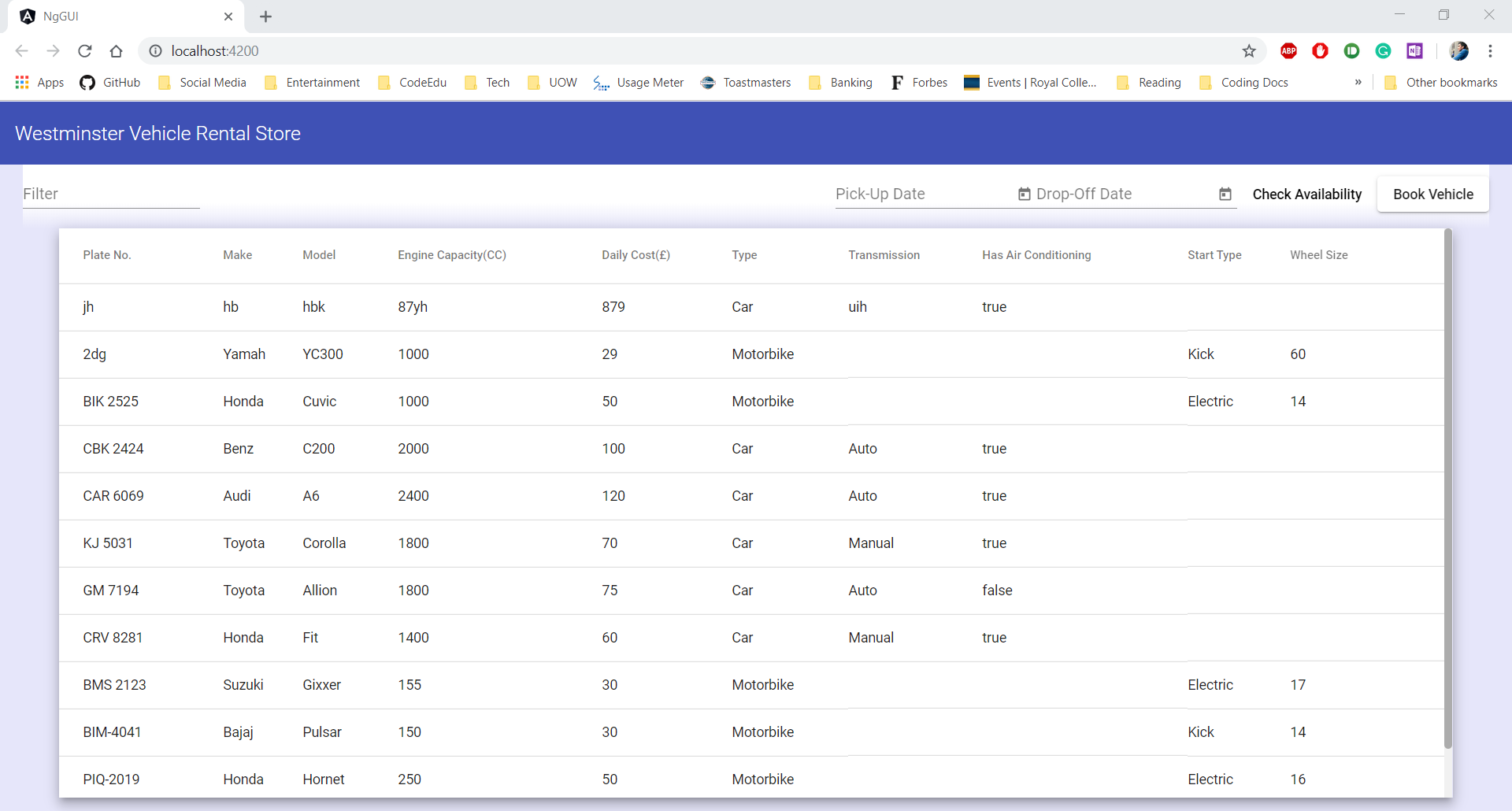
https://stackoverflow.com/questions/45417248/angular-4-material-table-highlight-a-row

Toolbar/ Header

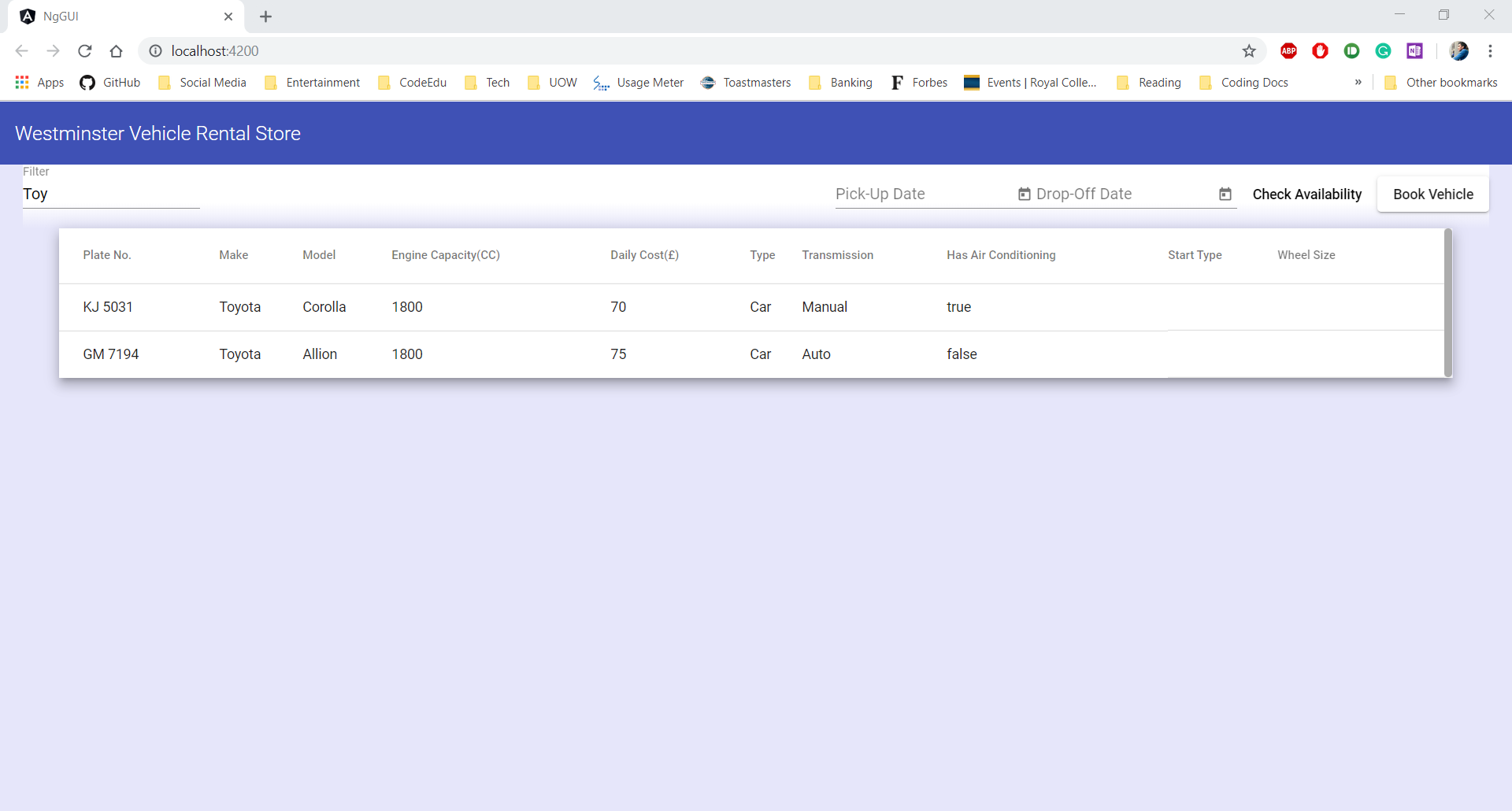
https://material.angular.io/components/toolbar/examples

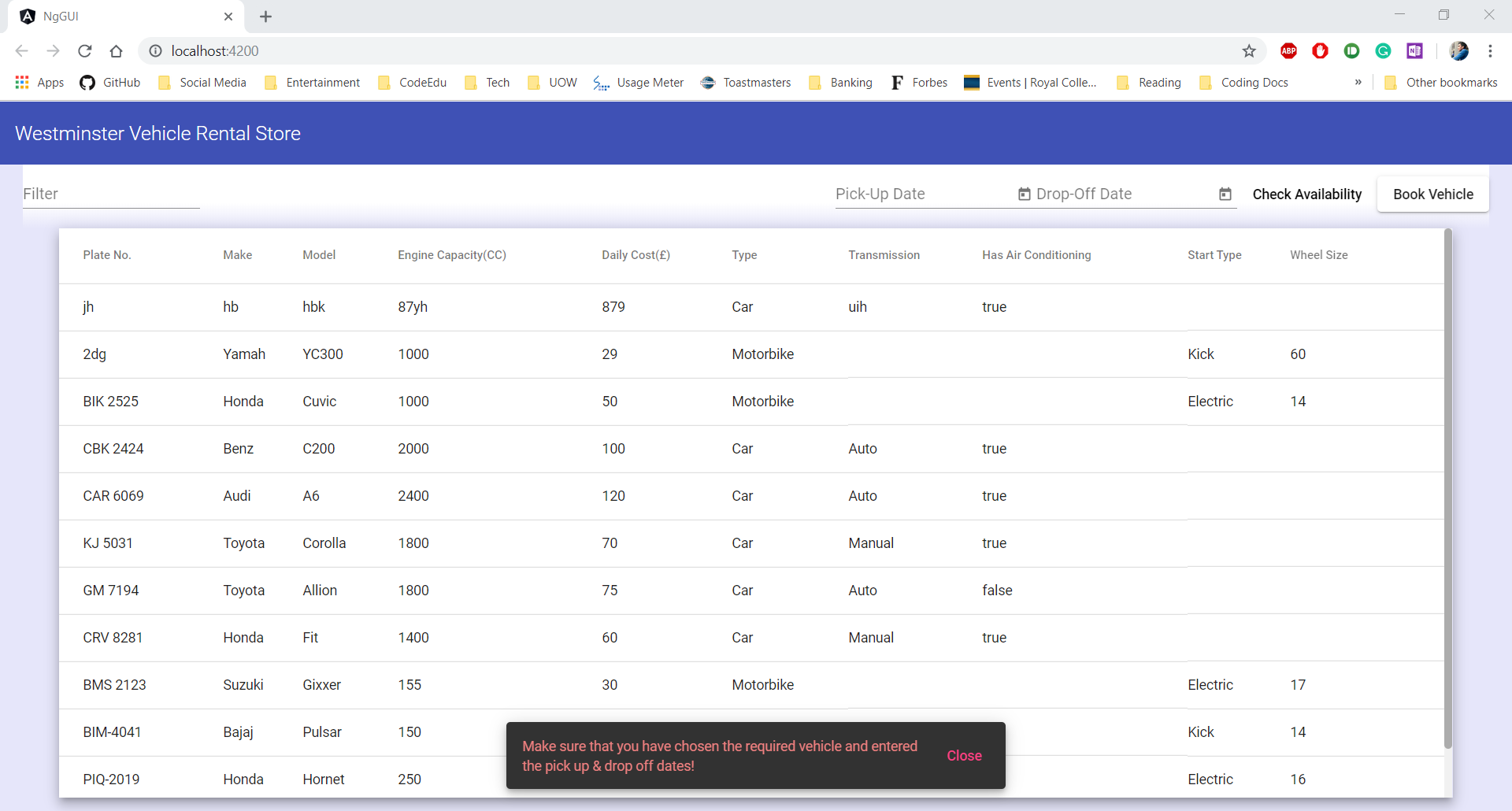
\*/

### Screenshots – Angular GUI

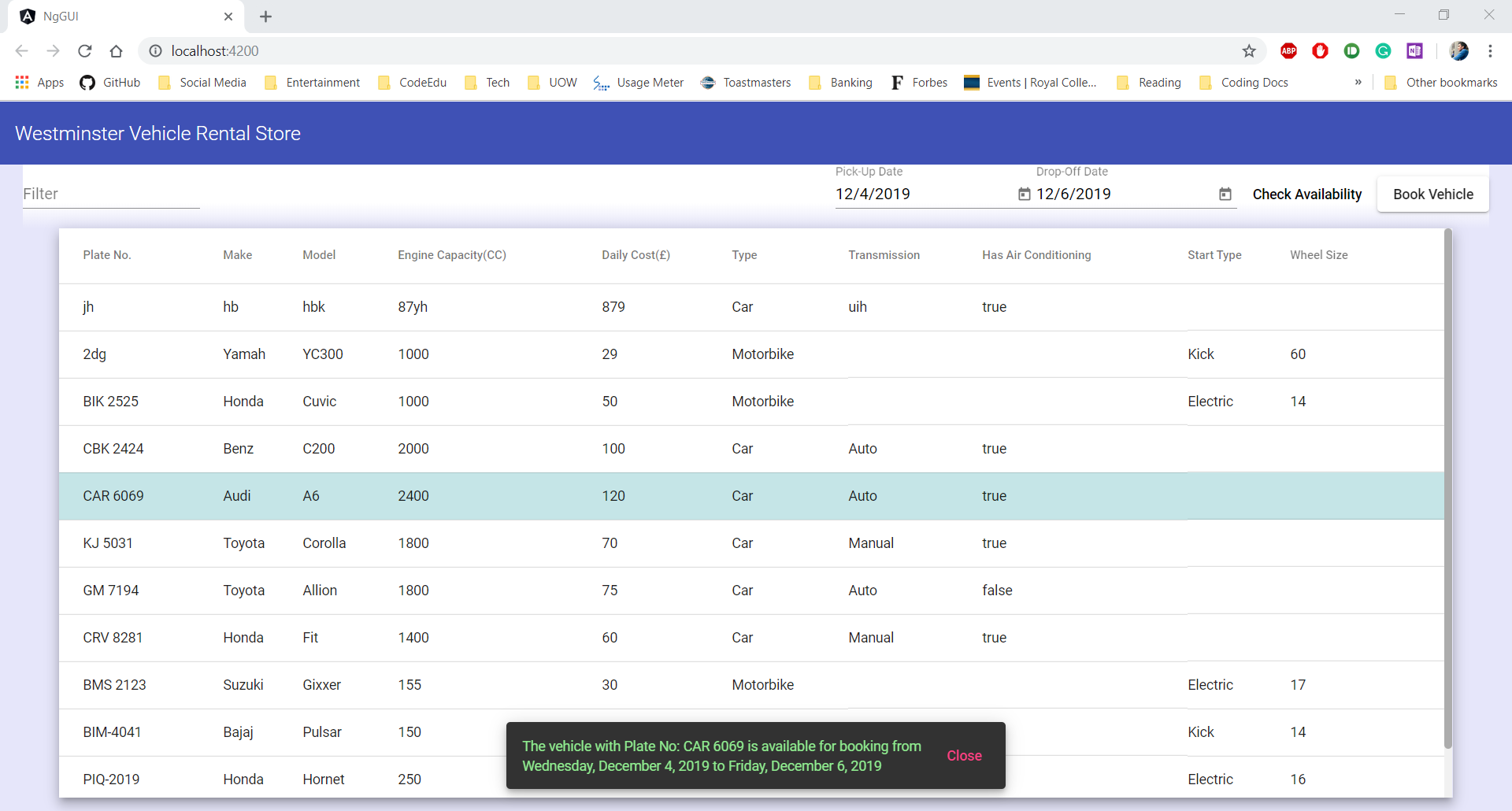


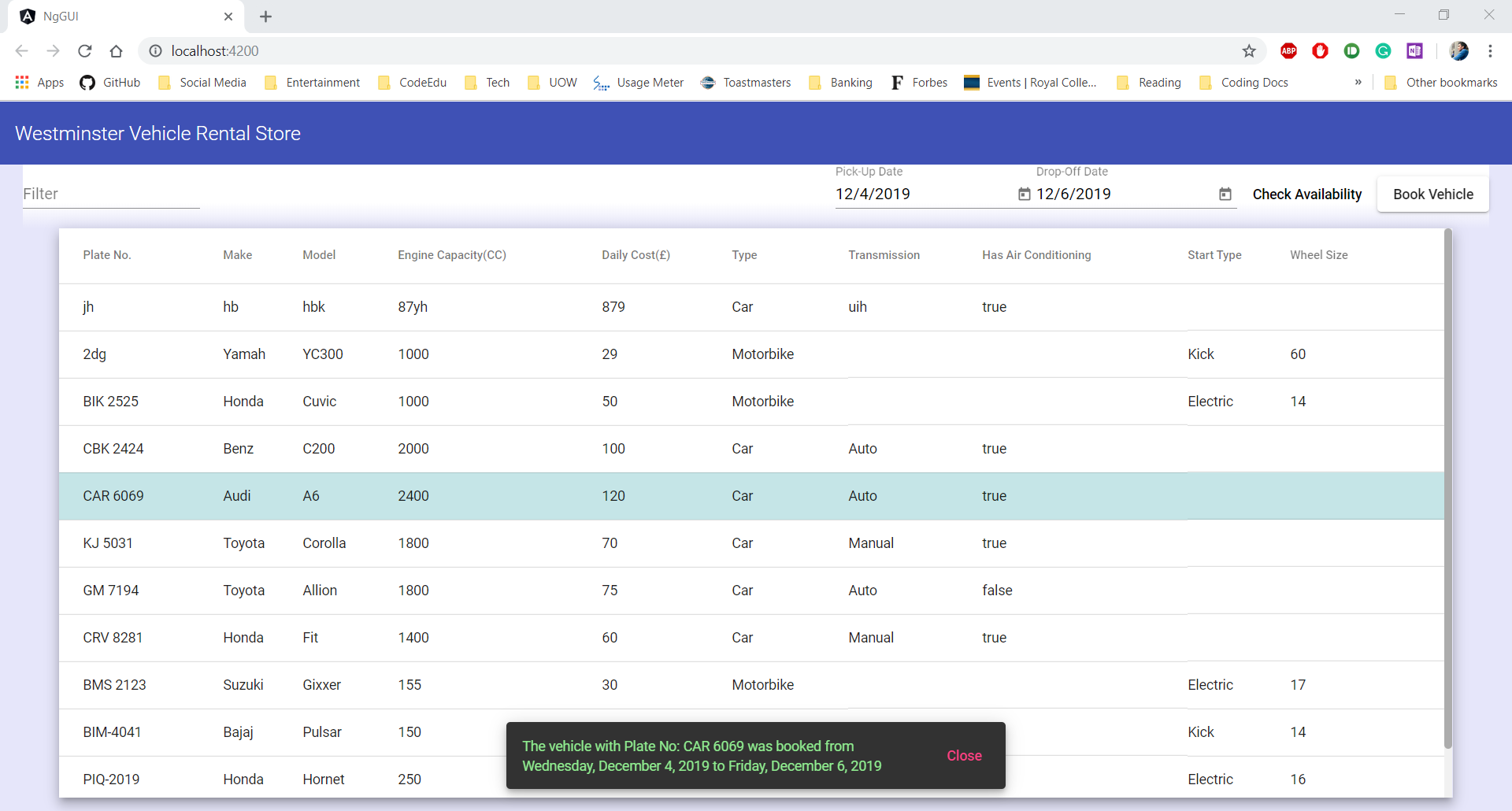
Filtering Function – Works for all fields

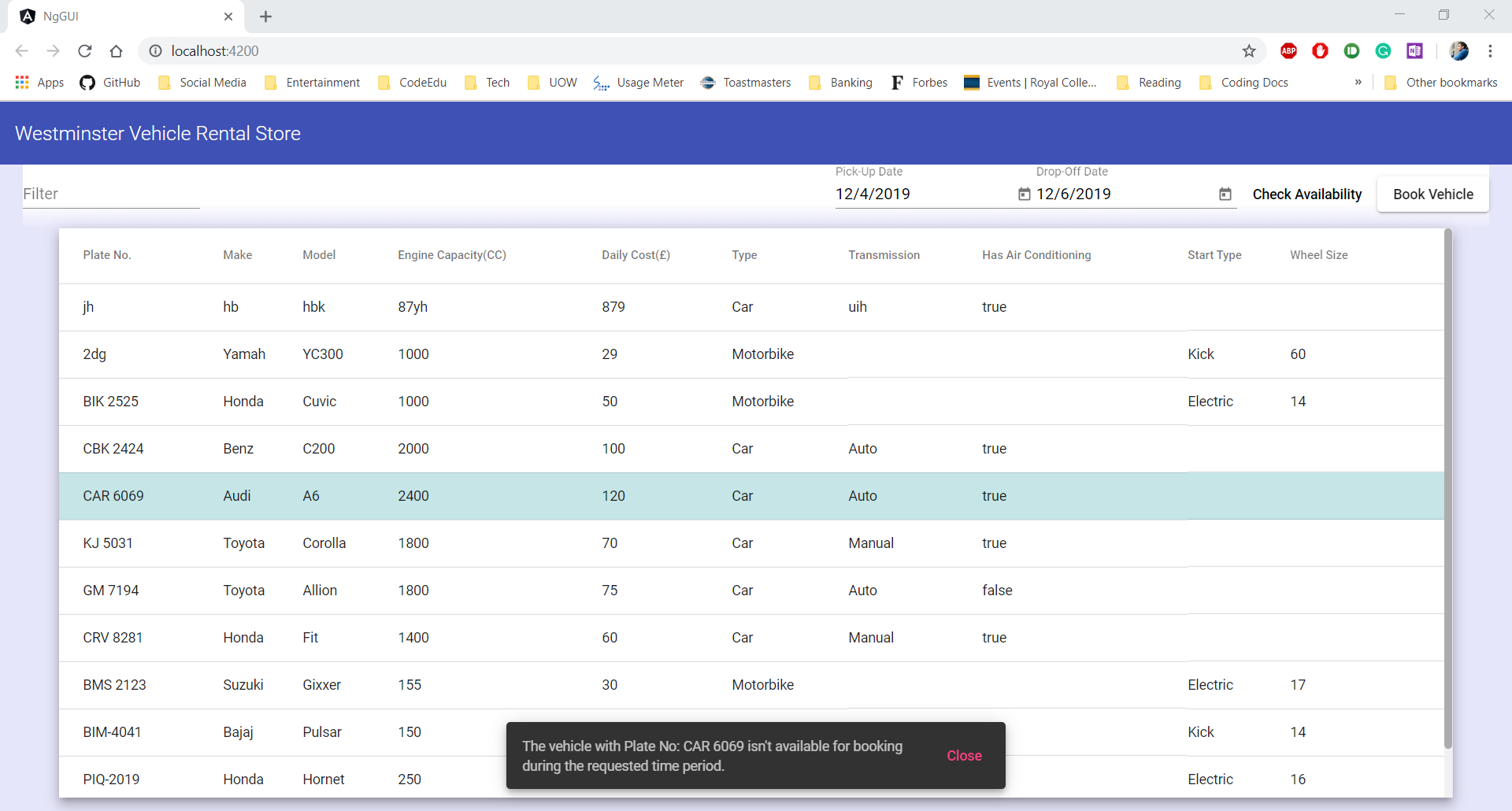


Check whether all required information has been added before making a request to the API.

Checking for availability to book a vehicle during a required time period (When “Check Availability” button is clicked).



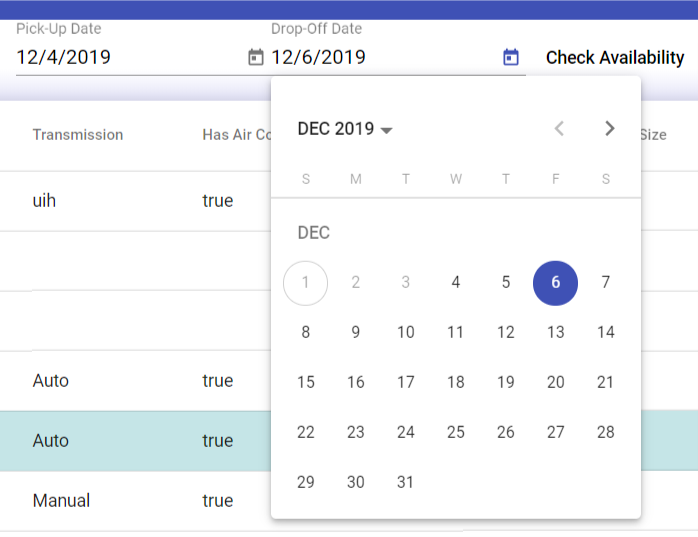
Booking a vehicle and letting the user know the exact booking details. (When “Book Vehicle” button is clicked).

Letting the user know that the vehicle is unavailable for booking during the requested time period (When “Book Vehicle”/ “Check Availability” button is clicked).

Restrictions for date input, to get valid date ranges

(Pick-Up date is available only from the current date, drop-off date is available on from the pick-up date onwards).

Ease of date selection using a date picker.



# Testing

## Test Plan

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID | Test Case | Input | Expected Output |
| 1 | Add vehicle (Car) | 1 | Request user to choose whether to add a car or a motorbike |
|  |  | 1 | Prompt user with information relevant for a car |
|  |  | Enter all prompted information | Add vehicle to system. |
|  |  |  |  |
| 2 | Add vehicle (Motorbike) | 1 | Request user to choose whether to add a car or a motorbike |
|  |  | 2 | Prompt user with information relevant for a motorbike |
|  |  | Enter all prompted information | Add vehicle to system. |
|  |  |  |  |
| 3 | Edit vehicle (Car/ Motorbike) | 1 | Prompt user with information relevant for a motorbike |
|  |  | 1 OR 2 | Prompt user for Plate No. of the vehicle |
|  |  | Enter plate number that’s already in the system (BIK 2525) | Display message to user that, “This Plate No exists in the system.”  Display the information of the vehicle with the entered Plate No.  Prompt user, “Do u want to edit information related to this vehicle?” |
|  |  | Y OR yes OR Yes OR y | Prompt all information relevant to vehicle type chosen |
|  |  | Enter all prompted information | Remove old vehicle information from the system.  Add new vehicle information to the system. |
|  |  |  |  |
| 4 | Edit vehicle (Car/ Motorbike) | 1 | Prompt user with information relevant for a motorbike |
|  |  | 1 OR 2 | Prompt user for Plate No. of the vehicle |
|  |  | Enter plate number that’s already in the system (BIK 2525) | Display message to user that, “This Plate No exists in the system.”  Display the information of the vehicle with the entered Plate No.  Prompt user, “Do u want to edit information related to this vehicle?” |
|  |  | N OR No or no OR n | Display the main menu with its options.  Prompt for selection of option |
|  |  |  |  |
| 5 | Delete vehicle - available | 2 | Prompt user for the Plate No. of the vehicle required to be deleted. |
|  |  | Enter plate number that’s already in the system (BIK 2525) | Display the type of vehicle that was deleted (car/ motorbike)  Print the details of the vehicle that was deleted.  Display the remaining parking slots left in the garage.  Remove vehicle from system. |
|  |  |  |  |
| 6 | Delete vehicle - unavailable | 2 | Prompt user for the Plate No. of the vehicle required to be deleted. |
|  |  | Enter plate number that’s not in the system (fg) | Display, “There's no vehicle related to the Plate No: fg”  Re prompt main menu. |
|  |  |  |  |
|  |  |  |  |
| 7 | Print List of vehicles | 3 | Sort the list of vehicles in alphabetical order of make.  Print all the plateIDs and types of all the vehicles in the system. |
|  |  |  |  |
| 8 | Open GUI | 4 | Prompt user to enter required GUI (Angular / JavaFX) |
|  |  | 1 | Open Angular GUI in the default web browser. |
|  |  |  |  |
| 9 | Open GUI | 4 | Prompt user to enter required GUI (Angular / JavaFX) |
|  |  | 2 | Open JavaFX GUI in a new window. |
|  |  |  |  |
| 10 | Exit program with exit message | 5 | Display exit message and exit console application. |
|  |  | 20 | Display message for invalid input & re prompt menu |
|  |  | -5 |
|  |  | f | Display, "Only integer numbers are allowed! Please provide a valid input".  Re prompt menu |
|  |  | @ |
|  |  |  |  |
| 11 | Write/ Save vehicle stock list into a file after any changes. | Add vehicle | Write vehicle information into a file. |
|  |  | Edit vehicle information | Edit vehicle information in the file. |
|  |  | Delete vehicle | Delete vehicle information from the file. |
|  |  |  |  |
| 12 | Validate integer input | 4 | Get out of validation loop. Continue with rest of the program functions. |
|  |  | d | Display, "Only integer numbers are allowed! Please provide a valid input".  Re prompt for input. |
|  |  | # |
|  |  |  |  |
| 13 | Validate double input | 3 | Get out of validation loop. Continue with rest of the program functions. |
|  |  | 4.7 |
|  |  | f | Display, "Only numbers are allowed! Please provide a valid input".  Re prompt for input. |
|  |  | % |

## Automated testing with Junit

### Code – Junit testing

package lk.dinuka.VehicleRentalSystem.Controller;

import lk.dinuka.VehicleRentalSystem.Model.Car;

import lk.dinuka.VehicleRentalSystem.Model.Motorbike;

import lk.dinuka.VehicleRentalSystem.Model.Vehicle;

import org.junit.Rule;

import org.junit.Test;

import org.junit.rules.TemporaryFolder;

import java.io.\*;

import java.math.BigDecimal;

import java.util.\*;

import static lk.dinuka.VehicleRentalSystem.Model.RentalVehicleManager.MAX\_VEHICLES;

import static lk.dinuka.VehicleRentalSystem.Model.Vehicle.count;

import static org.junit.Assert.assertEquals;

import static org.junit.Assert.assertTrue;

public class WestminsterRentalVehicleManagerTest {

@Test

public void addVehicleCar() { //testing whether a car can be added into the system

//test HashMap

HashMap<String, Vehicle> vehiclesMap = new HashMap<>(); //used to check whether the plate No already exists in the system

//test arrayList

List<Vehicle> vehiclesArrayList = new ArrayList<>(); //temporary arrayList

Vehicle newCar = new Car("CAR-123", "Honda", "Grace", "1300", BigDecimal.valueOf(70), "Car", "Auto", true);

int initialNumOfVehicles = vehiclesArrayList.size();

if (initialNumOfVehicles <= MAX\_VEHICLES) { //checking whether the vehicles existing in the system has occupied all the available parking lots

vehiclesMap.put("CAR-123", newCar); //adding new car into vehiclesMap

vehiclesArrayList.add(newCar);

// System.out.println(vehiclesArrayList);

// System.out.println(vehiclesMap);

// assertTrue("New Car wasn't added into the system", vehiclesArrayList.add(newCar)); //checking whether the car was added to the arrayList

assertEquals(initialNumOfVehicles + 1, vehiclesArrayList.size()); //??

assertEquals(initialNumOfVehicles + 1, vehiclesMap.size());

System.out.println("\nThere are " + (MAX\_VEHICLES - Vehicle.getCount()) + " parking lots left, to park vehicles.");

assertTrue("The new car hasn't been added to the system arrayList", vehiclesArrayList.contains(newCar));

assertTrue("The new car hasn't been added to the system hashMap", vehiclesMap.containsKey("CAR-123"));

} else {

System.out.println("There are no available spaces. 50 vehicles have been added!");

}

System.out.println("~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~"); //used to separate test outputs

}

@Test

public void addVehicleBike() { //testing whether a motorbike can be added into the system

//test HashMap

HashMap<String, Vehicle> vehiclesMap = new HashMap<>(); //used to check whether the plate No already exists in the system

//test arrayList

List<Vehicle> vehiclesArrayList = new ArrayList<>(); //temporary arrayList

Vehicle newBike = new Motorbike("BIK-123", "Hero", "Honda", "800", BigDecimal.valueOf(40), "Motorbike", "Push", 15);

int initialNumOfVehicles = vehiclesArrayList.size();

System.out.println(vehiclesArrayList);

System.out.println(vehiclesMap);

if (initialNumOfVehicles <= MAX\_VEHICLES) { //checking whether the vehicles existing in the system has occupied all the available parking lots

vehiclesMap.put("BIK-123", newBike); //adding new car into vehicles arrayList

vehiclesArrayList.add(newBike);

// System.out.println(vehiclesArrayList);

// System.out.println(vehiclesMap);

// assertTrue("New Motorbike wasn't added into the system", vehiclesArrayList.add(newBike)); //checking whether the motorbike was added to the arrayList

assertEquals(initialNumOfVehicles + 1, vehiclesArrayList.size()); //??

assertEquals(initialNumOfVehicles + 1, vehiclesMap.size());

System.out.println("\nThere are " + (MAX\_VEHICLES - Vehicle.getCount()) + " parking lots left, to park vehicles.");

assertTrue("The new motorbike hasn't been added to the system arrayList", vehiclesArrayList.contains(newBike));

assertTrue("The new motorbike hasn't been added to the system hashMap", vehiclesMap.containsKey("BIK-123"));

} else {

System.out.println("There are no available spaces. 50 vehicles have been added!");

}

System.out.println("~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~"); //used to separate test outputs

}

@Test

public void testEditCar() {

//test HashMap

HashMap<String, Vehicle> vehiclesMap = new HashMap<>(); //used to check whether the plate No already exists in the system

//test arrayList

List<Vehicle> vehiclesArrayList = new ArrayList<>(); //temporary arrayList

Vehicle newCar = new Car("CAR-123", "Honda", "Grace", "1300", BigDecimal.valueOf(70), "Car", "Auto", true);

String enteredPlateNo = newCar.getPlateNo();

vehiclesArrayList.add((newCar));

vehiclesMap.put(enteredPlateNo, newCar);

if (vehiclesMap.containsKey(enteredPlateNo)) {

System.out.println("This Plate No exists in the system.");

System.out.println(); //to keep space for clarity

//print information of vehicle when asked whether to edit

System.out.println("Make: " + vehiclesMap.get(enteredPlateNo).getMake());

System.out.println("Model: " + vehiclesMap.get(enteredPlateNo).getModel());

System.out.println("Engine Capacity: " + vehiclesMap.get(enteredPlateNo).getEngineCapacity());

System.out.println("Daily Cost (in £): " + vehiclesMap.get(enteredPlateNo).getDailyCost());

System.out.println("Type: " + vehiclesMap.get(enteredPlateNo).getType());

if (vehiclesMap.get(enteredPlateNo) instanceof Car) {

System.out.println("Transmission: " + ((Car) vehiclesMap.get(enteredPlateNo)).getTransmission());

System.out.println("Has Air Conditioning: " + ((Car) vehiclesMap.get(enteredPlateNo)).isHasAirCon());

} else {

System.out.println("Start Type: " + ((Motorbike) vehiclesMap.get(enteredPlateNo)).getStartType());

System.out.println("Wheel Size: " + ((Motorbike) vehiclesMap.get(enteredPlateNo)).getWheelSize());

}

boolean edit = true;

if (edit) {

System.out.println("\nMake required changes to vehicle information.");

} else {

System.out.println(); //keeps space and goes back to main menu

}

}

// vehiclesMap.clear(); //clearing to make sure that other unit tests aren't affected by this unit test

// vehiclesArrayList.clear();

System.out.println("~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~"); //used to separate test outputs

}

@Test

public void testEditBike() {

//test HashMap

HashMap<String, Vehicle> vehiclesMap = new HashMap<>(); //used to check whether the plate No already exists in the system

//test arrayList

List<Vehicle> vehiclesArrayList = new ArrayList<>(); //temporary arrayList

Vehicle newBike = new Motorbike("BIK-123", "Hero", "Honda", "800", BigDecimal.valueOf(40), "Motorbike", "Push", 15);

String enteredPlateNo = newBike.getPlateNo();

vehiclesArrayList.add(newBike);

vehiclesMap.put(enteredPlateNo, newBike);

if (vehiclesMap.containsKey(enteredPlateNo)) {

System.out.println("This Plate No exists in the system.");

System.out.println(); //to keep space for clarity

//print information of vehicle when asked whether to edit

System.out.println("Make: " + vehiclesMap.get(enteredPlateNo).getMake());

System.out.println("Model: " + vehiclesMap.get(enteredPlateNo).getModel());

System.out.println("Engine Capacity: " + vehiclesMap.get(enteredPlateNo).getEngineCapacity());

System.out.println("Daily Cost (in £): " + vehiclesMap.get(enteredPlateNo).getDailyCost());

System.out.println("Type: " + vehiclesMap.get(enteredPlateNo).getType());

if (vehiclesMap.get(enteredPlateNo) instanceof Car) {

System.out.println("Transmission: " + ((Car) vehiclesMap.get(enteredPlateNo)).getTransmission());

System.out.println("Has Air Conditioning: " + ((Car) vehiclesMap.get(enteredPlateNo)).isHasAirCon());

} else {

System.out.println("Start Type: " + ((Motorbike) vehiclesMap.get(enteredPlateNo)).getStartType());

System.out.println("Wheel Size: " + ((Motorbike) vehiclesMap.get(enteredPlateNo)).getWheelSize());

}

boolean edit = true;

if (edit) {

System.out.println("\nMake required changes to vehicle information.");

} else {

System.out.println(); //keeps space and goes back to main menu

}

}

// vehiclesMap.clear(); //clearing to make sure that other unit tests aren't affected by this unit test

// vehiclesArrayList.clear();

System.out.println("~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~"); //used to separate test outputs

}

@Test

public void testDeleteCarAvailable() { //testing the result when a car that is in the system is requested to be deleted

//test HashMap

HashMap<String, Vehicle> vehiclesMap = new HashMap<>(); //used to check whether the plate No already exists in the system

//test arrayList

List<Vehicle> vehiclesArrayList = new ArrayList<>(); //temporary arrayList

Vehicle newCar = new Car("CAR-123", "Honda", "Grace", "1300", BigDecimal.valueOf(70), "Car", "Auto", true);

String carPlateNo = "CAR-123";

vehiclesMap.put(carPlateNo, newCar);

vehiclesArrayList.add(newCar);

int initialNumOfVehicles = vehiclesArrayList.size();

// System.out.println(vehiclesArrayList);

// System.out.println(vehiclesMap);

if (vehiclesMap.containsKey(carPlateNo)) {

Vehicle vehicleToBeDeleted = vehiclesMap.get(carPlateNo);

vehiclesArrayList.remove(vehicleToBeDeleted);

// assertTrue(vehiclesArrayList.remove(vehicleToBeDeleted));

vehiclesMap.remove(carPlateNo);

count -= 1; //decreasing the number of vehicles from the system by one

String type = vehicleToBeDeleted.getType();

System.out.println("\nA " + type + " has been deleted from the system.");

System.out.println("The details of the vehicle that was deleted: " + vehicleToBeDeleted.toString()); //displaying information of deleted vehicle

// System.out.println(initialNumOfVehicles);

assertEquals(initialNumOfVehicles - 1, vehiclesArrayList.size());

// System.out.println(initialNumOfVehicles);

// assertEquals(initialNumOfVehicles - 1, vehiclesMap.size());

} else {

System.out.println("There's no vehicle related to the Plate No: " + carPlateNo);

}

System.out.println("~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~"); //used to separate test outputs

}

@Test

public void testDeleteCarUnavailable() { // testing the result when cars that are not

// in the system are requested to be deleted

//test HashMap

HashMap<String, Vehicle> vehiclesMap = new HashMap<>(); //used to check whether the plate No already exists in the system

//test arrayList

List<Vehicle> vehiclesArrayList = new ArrayList<>(); //temporary arrayList

String carPlateNo = "CAR-123";

int initialNumOfVehicles = vehiclesArrayList.size();

// System.out.println(vehiclesArrayList);

// System.out.println(vehiclesMap);

if (vehiclesMap.containsKey(carPlateNo)) {

Vehicle vehicleToBeDeleted = vehiclesMap.get(carPlateNo);

vehiclesArrayList.remove(vehicleToBeDeleted);

// assertTrue(vehiclesArrayList.remove(vehicleToBeDeleted));

vehiclesMap.remove(carPlateNo);

count -= 1; //decreasing the number of vehicles from the system by one

String type = vehicleToBeDeleted.getType();

System.out.println("\nA " + type + " has been deleted from the system.");

System.out.println("The details of the vehicle that was deleted: " + vehicleToBeDeleted.toString()); //displaying information of deleted vehicle

assertEquals(initialNumOfVehicles - 1, vehiclesArrayList.size());

// assertEquals(initialNumOfVehicles - 1, vehiclesMap.size());

} else {

System.out.println("There's no vehicle related to the Plate No: " + carPlateNo);

}

System.out.println("~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~"); //used to separate test outputs

}

@Test

public void testDeleteBikeAvailable() { //testing the result when a motorbike that is in the system is requested to be deleted

//test HashMap

HashMap<String, Vehicle> vehiclesMap = new HashMap<>(); //used to check whether the plate No already exists in the system

//test arrayList

List<Vehicle> vehiclesArrayList = new ArrayList<>(); //temporary arrayList

Vehicle newBike = new Motorbike("BIK-123", "Hero", "Honda", "800", BigDecimal.valueOf(40), "Motorbike", "Push", 15);

String bikePlateNo = "BIK-123";

vehiclesMap.put(bikePlateNo, newBike);

vehiclesArrayList.add(newBike);

int initialNumOfVehicles = vehiclesArrayList.size();

// System.out.println(vehiclesArrayList);

// System.out.println(vehiclesMap);

if (vehiclesMap.containsKey(bikePlateNo)) {

Vehicle vehicleToBeDeleted = vehiclesMap.get(bikePlateNo);

vehiclesArrayList.remove(vehicleToBeDeleted);

// assertTrue(vehiclesArrayList.remove(vehicleToBeDeleted));

vehiclesMap.remove(bikePlateNo);

count -= 1; //decreasing the number of vehicles from the system by one

String type = vehicleToBeDeleted.getType();

System.out.println("\nA " + type + " has been deleted from the system.");

System.out.println("The details of the vehicle that was deleted: " + vehicleToBeDeleted.toString()); //displaying information of deleted vehicle

assertEquals(initialNumOfVehicles - 1, vehiclesArrayList.size());

// assertEquals(initialNumOfVehicles - 1, vehiclesMap.size());

} else {

System.out.println("There's no vehicle related to the Plate No: " + bikePlateNo);

}

System.out.println("~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~"); //used to separate test outputs

}

@Test

public void testDeleteBikeUnavailable() { // testing the result when motorbikes that are not

// in the system are requested to be deleted

//test HashMap

HashMap<String, Vehicle> vehiclesMap = new HashMap<>(); //used to check whether the plate No already exists in the system

//test arrayList

List<Vehicle> vehiclesArrayList = new ArrayList<>(); //temporary arrayList

String bikePlateNo = "BIK-123";

int initialNumOfVehicles = vehiclesMap.size();

// System.out.println(vehiclesArrayList);

// System.out.println(vehiclesMap);

if (vehiclesMap.containsKey(bikePlateNo)) {

Vehicle vehicleToBeDeleted = vehiclesMap.get(bikePlateNo);

vehiclesArrayList.remove(vehicleToBeDeleted);

// assertTrue(vehiclesArrayList.remove(vehicleToBeDeleted));

vehiclesMap.remove(bikePlateNo);

count -= 1; //decreasing the number of vehicles from the system by one

String type = vehicleToBeDeleted.getType();

System.out.println("\nA " + type + " has been deleted from the system.");

System.out.println("The details of the vehicle that was deleted: " + vehicleToBeDeleted.toString()); //displaying information of deleted vehicle

assertEquals(initialNumOfVehicles - 1, vehiclesArrayList.size());

// assertEquals(initialNumOfVehicles - 1, vehiclesMap.size());

} else {

System.out.println("There's no vehicle related to the Plate No: " + bikePlateNo);

}

System.out.println("~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~"); //used to separate test outputs

}

@Test

public void testPrintList() {

Vehicle newCar = new Car("CAR-123", "Honda", "Grace", "1300", BigDecimal.valueOf(70), "Car", "Auto", true);

Vehicle newBike = new Motorbike("BIK-123", "Hero", "Honda", "800", BigDecimal.valueOf(40), "Motorbike", "Push", 15);

//test arrayList

List<Vehicle> vehiclesArrayList = new ArrayList<>(); //temporary arrayList

vehiclesArrayList.add(newCar);

vehiclesArrayList.add(newBike);

Collections.sort(vehiclesArrayList); //sort vehicles alphabetically, according to make

// print the plate number, the type of vehicle (car/ motorbike).

String leftAlignFormat = "| %-15s | %-12s |%n";

System.out.format("+-----------------+--------------+%n");

System.out.format("| Plate ID | Type |%n");

System.out.format("+-----------------+--------------+%n");

for (Vehicle item : vehiclesArrayList) {

if (item instanceof Car) {

System.out.format(leftAlignFormat, item.getPlateNo(), "Car");

} else if (item instanceof Motorbike) {

System.out.format(leftAlignFormat, item.getPlateNo(), "Motorbike");

}

}

System.out.println("+--------------------------------+");

vehiclesArrayList.clear(); //emptying arrayList so that other unit tests can run smoothly

count -= 2;

System.out.println("~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~"); //used to separate test outputs

}

//testing write/ save file ---------------

@Rule

public TemporaryFolder tempFolder = new TemporaryFolder();

@Test

public void testSaveFile() throws IOException {

//test arrayList

List<Vehicle> vehiclesArrayList = new ArrayList<>(); //temporary arrayList

File file = tempFolder.newFile("test.txt");

FileWriter soldFile = new FileWriter("test.txt", true);

soldFile.write(String.format("+-----------------+---------------+--------------+----------------+---------------+-----------+--------------+--------+------------+------------+%n"));

soldFile.write(String.format("| Plate ID | Make | Model | Engine Capacity| Daily Cost(£) | Type | transmission | AirCon | Start type | Wheel Size |%n"));

soldFile.write(String.format("+-----------------+---------------+--------------+----------------+---------------+-----------+--------------+--------+------------+------------+%n"));

// soldFile.write(System.getProperty("line.separator")); //line break

String leftAlignFormat2 = "| %-15s | %-13s | %-12s | %-14s | %-13s | %-9s | %-12s | %-6s | %-10s | %-10s |%n";

//writing into the file

for (Vehicle veh : vehiclesArrayList) {

if (veh instanceof Motorbike) {

soldFile.write(String.format(leftAlignFormat2, veh.getPlateNo(), veh.getMake(), veh.getModel(), veh.getEngineCapacity(),

veh.getDailyCost(), veh.getType(), " - ", " - ", ((Motorbike) veh).getStartType(), ((Motorbike) veh).getWheelSize()));

} else {

soldFile.write(String.format(leftAlignFormat2, veh.getPlateNo(), veh.getMake(), veh.getModel(), veh.getEngineCapacity(),

veh.getDailyCost(), veh.getType(), ((Car) veh).getTransmission(), ((Car) veh).isHasAirCon(), " - ", " - "));

}

soldFile.write(System.getProperty("line.separator")); //line break

}

soldFile.write(String.format("+-----------------+---------------+--------------+----------------+---------------+-----------+--------------+--------+------------+------------+%n"));

assertTrue(file.exists());

}

// ----------------

@Test

public void testViewGUIFX() {

// GUI.main(null); //used to open javafx application

//test won't complete until javaFX application is closed

}

@Test

public void testViewGUIAngular() {

// API.getAllVehiclesToFront(); //send vehicles to front end

// API.postBookingsFromFront(); //handle booking

// API.postAvailabilityFromFront(); //handle availability

//

//

// //Open Angular GUI in browser

// ProcessBuilder builder = new ProcessBuilder("explorer.exe", "http://localhost:4200/");

//

// builder.redirectErrorStream(true);

//

// Process p = null;

// try {

// p = builder.start();

// } catch (IOException e) {

// e.printStackTrace();

// }

// BufferedReader r = new BufferedReader(new InputStreamReader(p.getInputStream()));

// String line;

// while (true) {

// try {

// line = r.readLine();

// if (line == null) {

// break;

// }

// System.out.println(line);

//

// } catch (IOException e) {

// e.printStackTrace();

// }

// }

}

@Test

public void testIntInputValidationString() {

String data = "Hello Human\r\n";

InputStream stdin = System.in;

try {

System.setIn(new ByteArrayInputStream(data.getBytes()));

Scanner scanInput = new Scanner(System.in);

System.out.println("Input entered: " + scanInput.nextLine());

// while (!scanInput.hasNextInt()) {

if (!scanInput.hasNextInt()) {

System.out.println("Only integer numbers are allowed! Please provide a valid input"); //error handling message for characters other than integers

// scanInput.next(); //removing incorrect input entered

}

} finally {

System.setIn(stdin);

}

}

@Test

public void testIntInputValidationCharacter() {

String data = "@\r\n";

InputStream stdin = System.in;

try {

System.setIn(new ByteArrayInputStream(data.getBytes()));

Scanner scanInput = new Scanner(System.in);

System.out.println("Input entered: " + scanInput.nextLine());

// while (!scanInput.hasNextInt()) {

if (!scanInput.hasNextInt()) {

System.out.println("Only integer numbers are allowed! Please provide a valid input"); //error handling message for characters other than integers

// scanInput.next(); //removing incorrect input entered

}

} finally {

System.setIn(stdin);

}

}

@Test

public void testIntInputValidationInteger() {

int scanInput = 4;

assertTrue("Only integer numbers are allowed! Please provide a valid input", scanInput == (int) scanInput);

}

@Test

public void testDoubleInputValidationString() {

String data = "Hello Humans\r\n";

InputStream stdin = System.in;

try {

System.setIn(new ByteArrayInputStream(data.getBytes()));

Scanner scanInput = new Scanner(System.in);

System.out.println("Input entered: " + scanInput.nextLine());

// while (!scanInput.hasNextInt()) {

if (!scanInput.hasNextInt()) {

System.out.println("Only numbers are allowed! Please provide a valid input"); //error handling message for characters other than integers

// scanInput.next(); //removing incorrect input entered

}

} finally {

System.setIn(stdin);

}

}

@Test

public void testDoubleInputValidationCharacter() {

String data = "$\r\n";

InputStream stdin = System.in;

try {

System.setIn(new ByteArrayInputStream(data.getBytes()));

Scanner scanInput = new Scanner(System.in);

System.out.println("Input entered: " + scanInput.nextLine());

// while (!scanInput.hasNextInt()) {

if (!scanInput.hasNextInt()) {

System.out.println("Only numbers are allowed! Please provide a valid input"); //error handling message for characters other than integers

// scanInput.next(); //removing incorrect input entered

}

} finally {

System.setIn(stdin);

}

}

@Test

public void testDoubleInputValidation() {

double scanInput = 4.02;

assertTrue("Only numbers are allowed! Please provide a valid input", scanInput == (double) scanInput);

}

}

/\*Reference:

https://stackoverflow.com/questions/156503/how-do-you-assert-that-a-certain-exception-is-thrown-in-junit-4-tests

https://www.mkyong.com/unittest/junit-4-tutorial-2-expected-exception-test/

https://stackoverflow.com/questions/12558206/how-can-i-check-if-a-value-is-of-type-integer

JUnit: How to simulate System.in testing

http://www.javased.com/?post=1647907

\*/

### Screenshots – Junit testing

