**Parking Management System**

Analysis & Assumptions :

1. The valet will use the application to perform parking related services.
2. There are 3 types of Parking a. Regular, Handicapped, Compact
3. Parking has parking spots. Parking spots can be of 3 types small, medium and large
4. So a parking spot has 2 categories parking type and size of parking.
5. 3 kinds of vehicles can be parked. two wheeler, car and bus
6. A two wheeler can be parked in all sizes of spot (small, medium and large). A car can be parked in medium and large spot and a bus can only be park in large spot
7. One spot can contain only one vehicle irrespective the size of vehicle and capacity of spot.

The system will provide below services

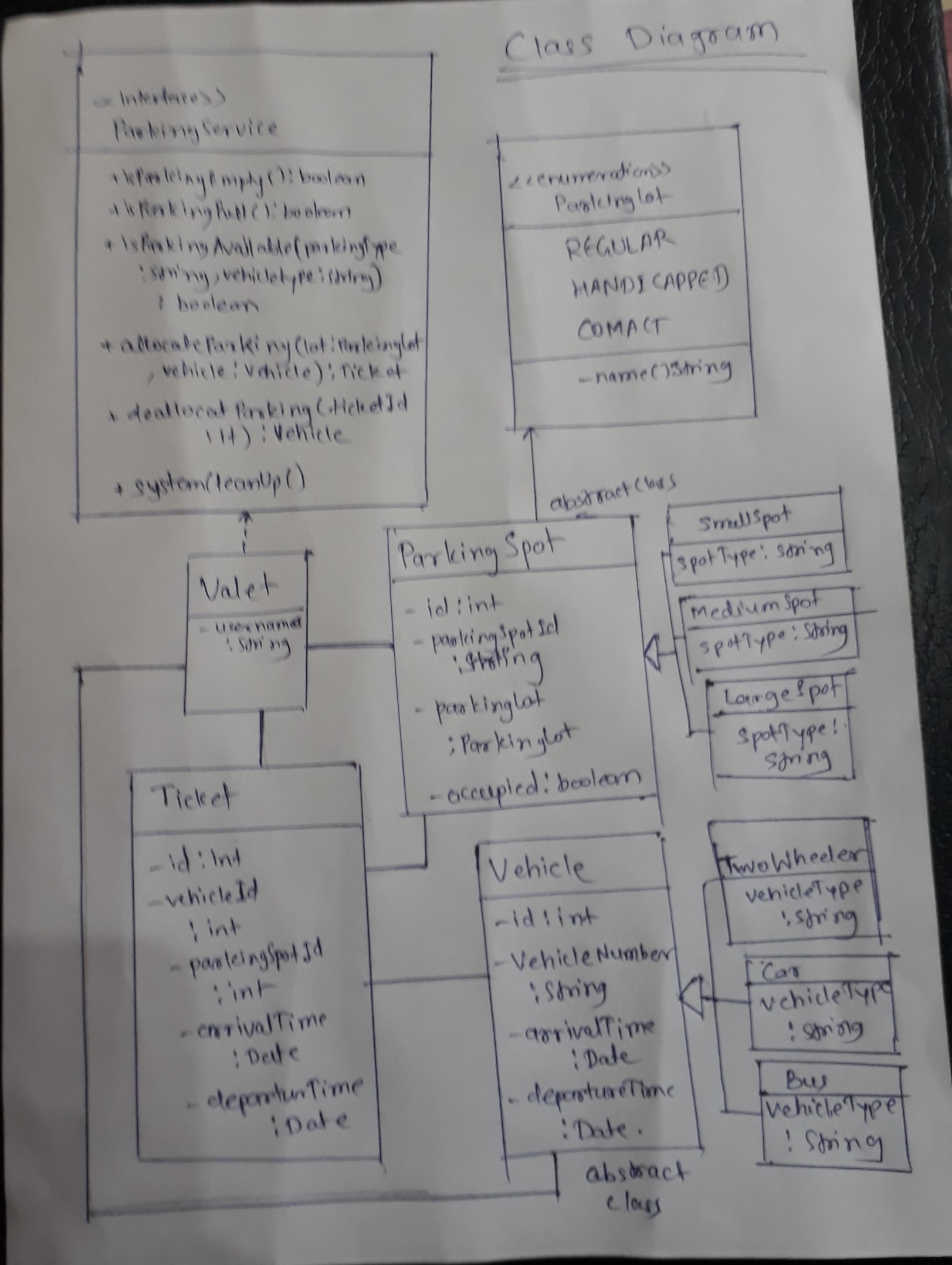
1. To check if the parking is totally empty
2. To check if parking is full and there is no spot for any parking
3. To check if a specific type of parking is available for a specific vehicle
4. To allocate the corresponding parking to the vehicle and provide (generate) ticket
5. To de-allocate the parking – Once user submits the ticket the vehicle will be handed over.

Development Approach :

**Class Diagram**:

Refer the class diagram in image below :

(Please zoom in the page if not visible)



Here ParkingLot is considered as a type and created as enumeration. It can have 3 values. The parking spot can be of any of these 3 types.

The parking spot is can further divided into 3 types Small, Medium or Large. A parking spot has to be any one of these 3 size. Created the ParkingSpot as abstract class so that an instance of it cannot be created directly. Vehicle class contains all the common properties of all the 3 child classes.

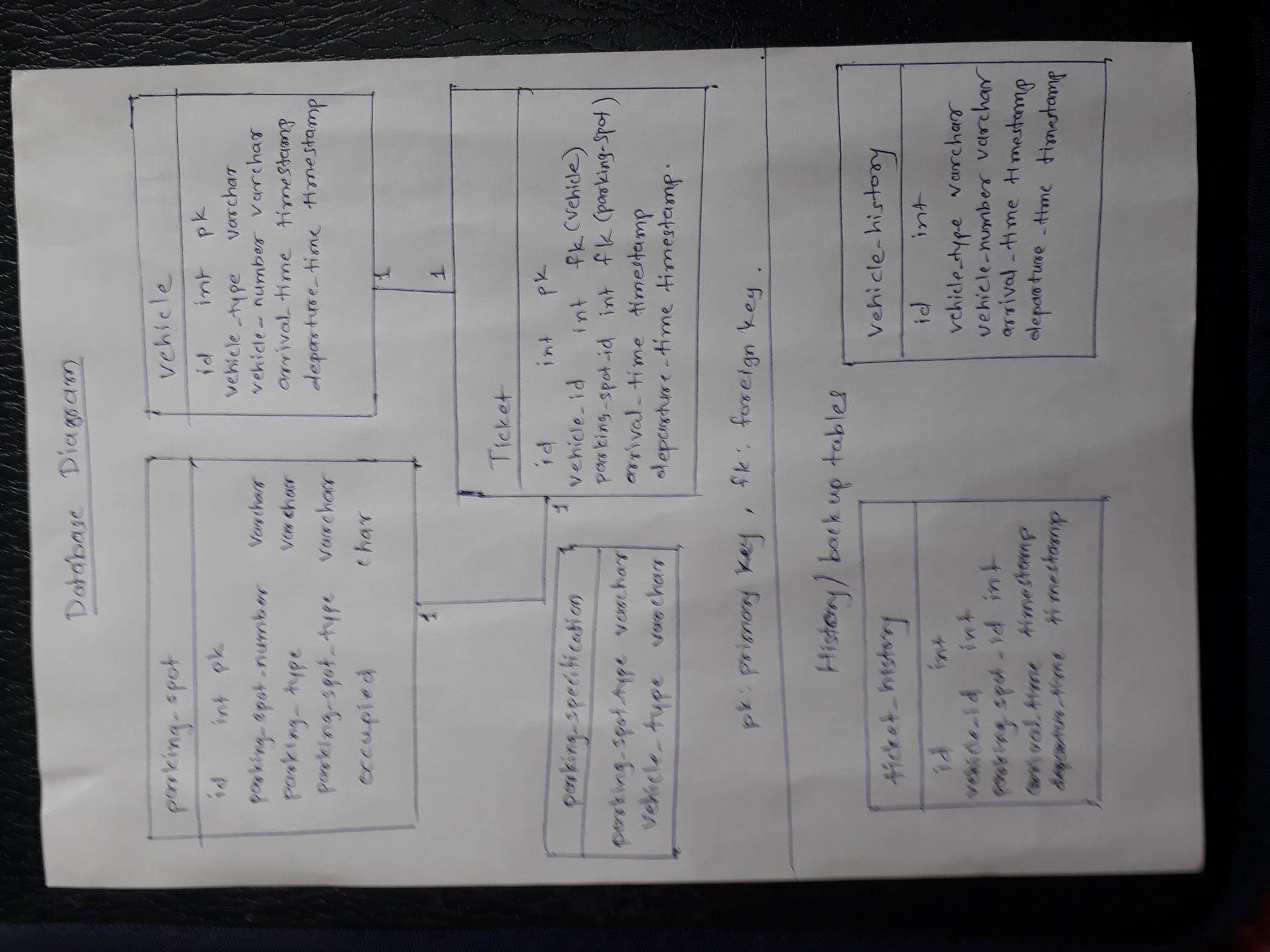
In case of vehicle It can be a two wheeler, Car and Bus. The vehicle class is also an abstract class. It has common vehicle properties which can be used by all the types of vehicle. But to create an instance of type vehicle any of one of the three classes can be used.

The ParkingService is an interface which mentions the services to be provided by Valet. So class Valet has implemented the interface parking service and provides all the services (functionalities) mentioned the ParkingSevice interface.

When a person is allocated parking a ticket is generated. To deallocate the parking the ticket is mandatory.

**Database** :

Please refer the image below –



The parking\_spot table contains the information of all the spots and its parking type and if the spot is allocated or not.

The vehicle and ticked tables will have lots of data. As soon as a vehicle is allocated a parking its information is shored in the system also a ticket will also be generated and handed over to driver.

The parking\_specification table is used to store the rules of parking. Using this configuration table the spots are allocated to the vehicles

e.x.

|  |  |
| --- | --- |
| Vehicle type | Parking spot Type |
| Two wheeler | Small |
| Two wheeler | Medium |
| Two wheeler | large |
| Car | Medium |
| Car | Large |
| Bus | large |

**History tables** :

As there could so many parking in whole day. One type of vehicle can also come and leave so many time. For every vehicle entered and parked in parking there will be entry in vehicle and ticket tables.

There won’t be any data added in parking\_spot and no change in parking\_specification.

But data in vehicle and table can grow so fast. To overcome this the history tables are created.

The system will keep the backup of vehicle and ticket table which will be available for some time and then can be deleted from history table as well. As this data can be used to trace any parking info. It would be better to put data in history table for some time.

So when Valet stops the parking system these data cleanup activity will take place. The vehicles left the parking and deallocated tickets will be moved to the history. So there will be less data in vehicle and parking table and it will not affect the performance of the system.

Ddl statemets :

CREATE TABLE parking\_spcification

(

id integer PRIMARY KEY,

spot\_type varchar(20),

vehicle\_type varchar(20)

);

CREATE TABLE parking\_spot

(

id integer PRIMARY KEY,

parking\_spot\_number varchar(20),

description varchar(30),

parking\_type varchar(20),

parking\_spot\_type varchar(20),

occupied char

);

CREATE TABLE vehicle

(

id integer PRIMARY KEY,

vehicle\_type varchar(20),

vehicle\_number varchar(20),

arrival\_time timeStamp,

departure\_time timeStamp

);

CREATE TABLE ticket

(

id integer PRIMARY KEY,

vehicle\_id integer,

parking\_spot\_id integer,

arrival\_time timeStamp,

departure\_time timeStamp,

FOREIGN KEY (parking\_spot\_id) REFERENCES parking\_spot(id),

FOREIGN KEY (vehicle\_id) REFERENCES vehicle(id)

);

CREATE TABLE vehicle\_history

(

id integer,

vehicle\_type varchar(20),

vehicle\_number varchar(20),

arrival\_time timeStamp,

departure\_time timeStamp

);

CREATE TABLE ticket\_history

(

id integer,

vehicle\_id integer,

parking\_spot\_id integer,

arrival\_time timeStamp,

departure\_time timeStamp

);

CREATE SEQUENCE spot\_id\_seq AS integer START WITH 1 INCREMENT BY 1 MAXVALUE 100000 CYCLE;

CREATE SEQUENCE ticket\_id\_seq AS integer START WITH 1 INCREMENT BY 1 MAXVALUE 100000 CYCLE;

CREATE SEQUENCE vehicle\_id\_seq AS integer START WITH 1 INCREMENT BY 1 MAXVALUE 100000 CYCLE;

Information about the Application :

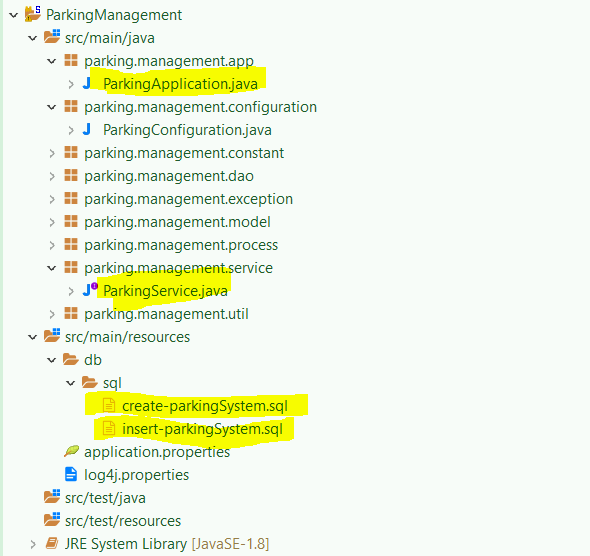
Name : Parking Management

Class to start the service is : parking.management.app.ParkingApplication

This process will continuously run and will accept the input from the valet and perform the corresponding operation

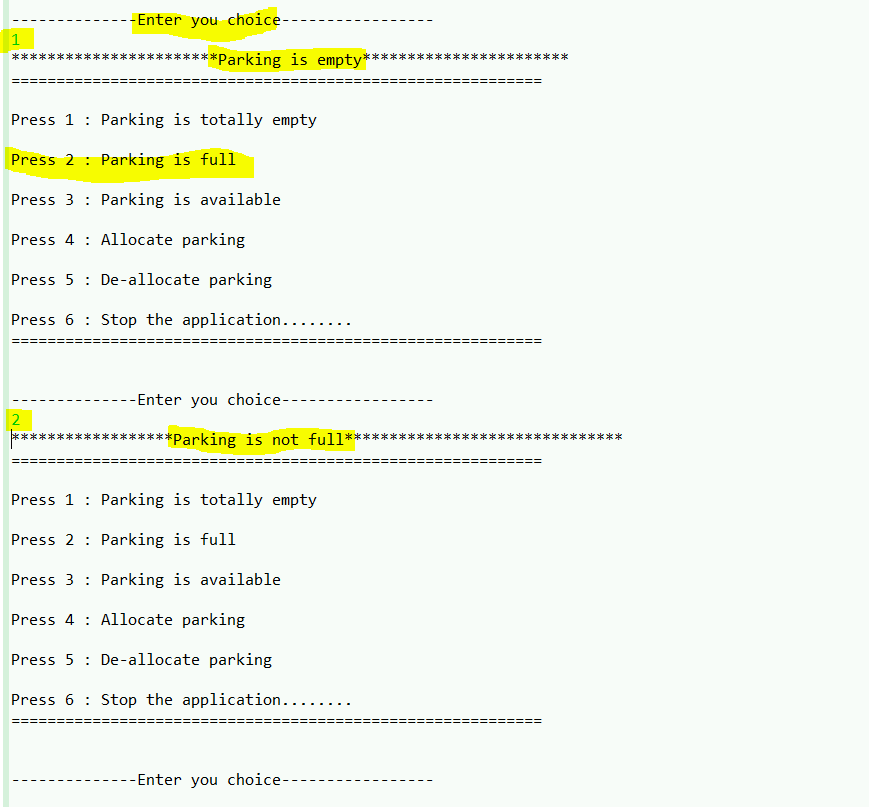
Logs will be stored in logs/ParkingManagement.log

While running the application, application used in memory database. In create and inserts data in table and uses the same to run the application. Database can be changed by changing the database configurations.

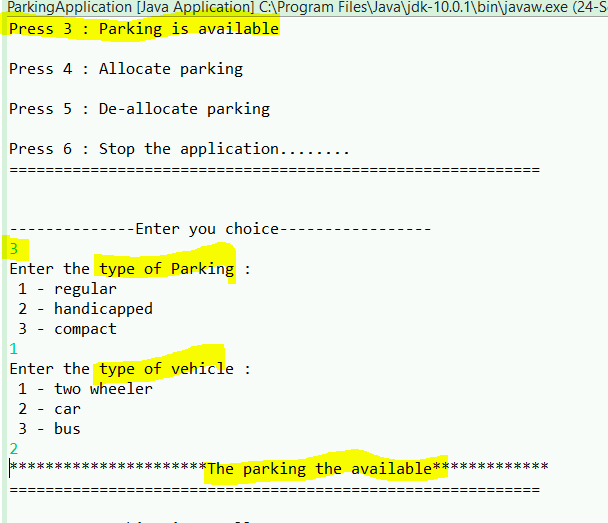


Application Screenshot –

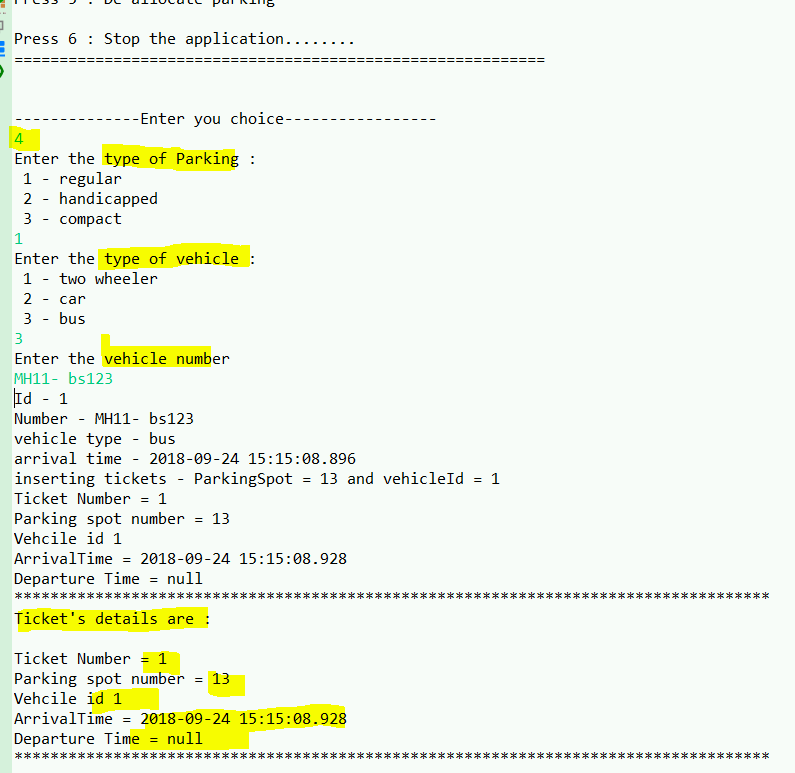
1. To check parking is full or empty



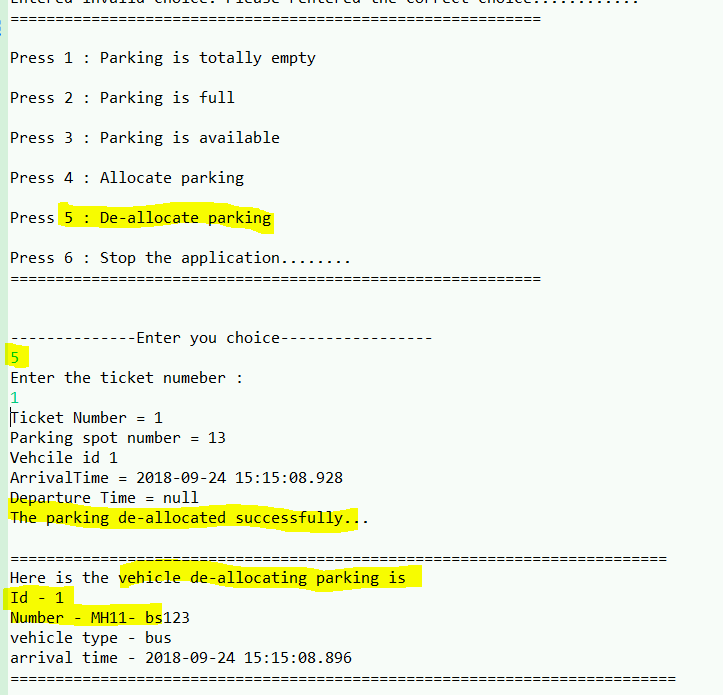
1. To check particular type of parking is available for specific type of vehicle



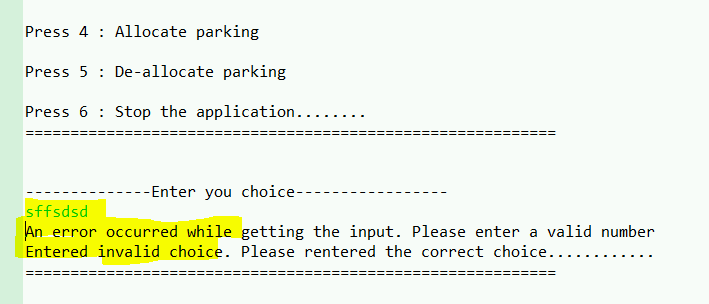
1. Allocate a parking to a vehicle and generate a ticket



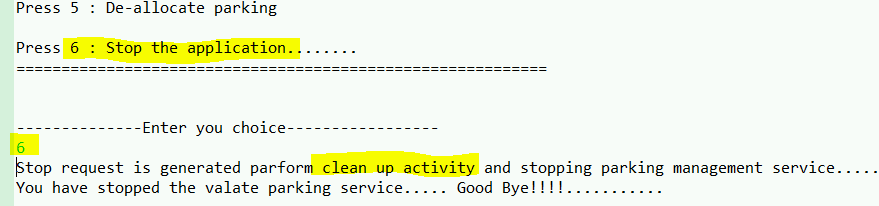
1. Deallocate the parking and gets an instance of vehicle



1. User entered wrong choice



1. Stopping the service



**Limitations :**

GUI is not present. A bit difficult to process.

The large and medium sizes of spots can allocate to only one small sized bike which is a waste of space.

There are not much validation

Only a desktop application.

**Enhancement :**

Parking can be made payable.

Allow access to multiple valets same time which will improve the services speed.

**Main Class : parking.management.app.ParkingApplication**