Case Study-1
Parking Lot

System Requirements (Excepted and can be added more)

1. The parking lot should have multiple floors where customers can park their vehicles.

2. The parking lot should have multiple entry and exit points.

3. Customers can collect a parking ticket from the entry points and can pay the parking fee at the exit points on their way out.

4. Customers can pay the tickets at the automated exit panel or to the parking attendant.

5. Customers can pay via both cash and credit cards.

6. Customers should also be able to pay the parking fee at the customer's info portal on each floor. If the customer has paid at the info portal, they don't have to pay at the exit.

7. The system should not allow more vehicles than the maximum capacity of the parking lot. If the parking is full, the system should be able to show a message at the entrance panel and on the parking display board on the ground floor.

8. Each parking floor will have many parking spots. The system should support multiple types of parking spots such as Compact, Large, Handicapped, Motorcycle, etc.

9. The Parking lot should have some parking spots specified for electric cars. These spots should have an electric panel through which customers can pay and charge their vehicles.

10. The system should support parking for different types of vehicles like car, truck, van, motorcycle, etc.

11. Each parking floor should have a display board showing any free parking spot for each spot type.
12. The system should support a per-hour parking fee model. For example, customers have to pay Rs 20 for the first hour, Rs 10 for the second and third hours, and Rs 5 for all the remaining hours.

Using Object Oriented thinking to develop a ParkingLot Software

If you are looking for a Software for your ParkingLot, here is why you should choose to be our client:

- ➢ **We have used Object Oriented design which gives you incredible flexibility and clarity about the interaction between the various components**
- ➢ **We provide for easy modification as and when required in the future**
- ➢ **We provide integration with a simple and clean SQLite Database that keeps your data well-organised**
- ➢ **We bring you a different level of User-Friendliness with simple Graphic User Interfaces that are easy to use, but provide a very high level of flexibility and convenience**
- ➢

Features

- We model interactions of the System with 3 types of users, Admin, Employees and Customers
- Admin-
    - The Admin is allowed to Log In (Using GUI where password is hidden while being typed on the screen).
    - The Admin is allowed complete flexibility in choosing the number of Floors and the number of Spots of each type in each Floor.
    - The Admin is also allowed to choose different price rates for 1st hour, 2nd and 3rd hour, and remaining hours, for each type of Spot independently
    - Admin can also alter the price of electricity to be charged per hour at the electric panels
    - Admin can add or remove Employees from the record
    - Admin can add or remove Checkpoints from the record
    -
- Employee-
    - Employees can LogIn, Change their password if they wish(Again using a GUI so password is hidden from sight of others), and record which Checkpoint(Entry point, exit point or Info Portal) that they are going to work at on that day
    - Employees can also park their car when not on duty as a regular customer, which is why we have used the OOPs concept of Inheritance and decided that an Employee 'is a' Customer
    - Checkpoints may be manned or automated. Hence some Employee may or may not be assigned to a particular Checkpoint.
- Customer -

    - A typical Customer can ‘Go To’ and particular Checkpoint by entering the ID of the Checkpoint
    - At Entry Point he can enter his type of Spot required depending on the Vehicle. Our system finds the closest available Spot of the suitable type, even if it is on a different Floor, generates a unique Ticket and allocates the Spot. It records the Entry time using System Time and will be used later to calculate the bill
    - The Customer can use the Electric Panel GUI to interact with the automated Charging Panel, available at Electric Type Spots. He can pay directly at the panel which notes the time difference between start and stop times and calculates the bill amount.
    - The Customer can go for the prepaid option of paying Ticket by going to Info Portal. Bill is computed for the Customer based on type of Spot and time. If bill is paid at Info Portal, the Ticket status is changed to Paid
    - At the exit point if the Ticket is not paid, Customer must pay. If the Ticket is already paid, the customer need not pay.
    - We also have a Graphic user Interface to Display the Number of available Spots of each type on any given Floor

<u>Discussions around OOPs design decisions</u>

1. We considered making Checkpoint an interface or abstract class and implementing its features as required in the individual classes Entry Point, Exit Point and Info-Portal. However we then decided against that because (we cannot instantiate abstract classes and interfaces) and we wanted to instantiate all Checkpoints and store them together because all Checkpoints have very similar attributes, although they have different functions.
2. **Inheritance.** We decided that Admin inherits Employee which inherits Customer because an Employee ‘is a’ Customer on some days and an Admin ‘is an’ Employee with additional powers
3. We used **Encapsulation** combining the attributes and functions of each abstract Entity into multiple classes, each storing its individual attributes and functions. All entities interact with each other by passing objects and all required information to each other through appropriate setter and getter methods. This provides a safety check and disallows direct modification of instance variables.
4. **Polymorphism.** We have used **method overloading** wherever it could be elegantly used. For example, in the method computeBill(), the calculation is the same but the argument passed is different; for prepaid, it is the number of hours chosen by the user and for postpaid it is the exit time, recorded by the System.
5. **Polymorphism**. We have used **method overriding** in the GUIs during the implementation of JButtons and on-click events.

6. **Abstraction** has been used in GUIs that implement inbuilt interfaces such as ActionListener provided by Java. We also attempted to try to implement the Callback mechanism for Synchronous using an interface and anonymous class, so that we can integrate the GUIs with the normal flow of execution, to some extent, however we were unable to understand it completely and implement it successfully.
7.

Our Internal Representation of Data in the Database

Employee Records Table

Employee ID(Primary ID)	Username of employee	Password	Amount Due by the Employee	Login Status

Internally we can represent the single admin as an employee with ID 1.
Floors Information Table

Floor Number	Total number of Compact Spots	Number of Available Compact Spots	Total number of Large Spots	Number of Available Large Spots	Total number of HandicappedSpots	Number of Available handicappedSpots	Total number of 2-wheeler Spots	Number of Available 2-wheeler Spots	Total number of Electric Spots	Number of Available Electric Spots
1										
2										
3										

Tickets Records Table

Ticket ID	Start Time	Spot ID	Vehicle Number Plate	Is Paid?

Checkpoints Table

Checkpoint ID	Name	Type(Entry/Exit/Info)	Floor no	Employee assigned(by ID)(int)
1	WestWing	Entry point	3	
2				

Spot Prices table

Spot Type	1st hour	2nd&3rd hour	Remaining hours
Compact			
Large			
Handicapped			
2-Wheeler			
Electric			

Price of Electricity Per Hour
380.7

Division of Tasks among team members and methodology followed

- We had 2 group discussions of a total of 3 hours, discussing design possibilities.
- During the discussion we laid out the structure.
- Debeshee roughly divided the pseudocode structure into 5 parts and then we voluntarily picked our choices.
- Aashrith opted to implement the Database component
- Debeshee volunteered to design and implement GUIs wherever possible and all user input/output facets.
- Anand volunteered to implement the important classes, and related functionalities, of Customer, Admin, Employee, Exit Point, Info Portal.
- Jaswanth volunteered to implement the other crucial set of classes, Spot, Floor, Ticket, EntryPoint and Checkpoint
- Navjoth took the task of implementing the central class, ParkingLot, which is crucial for all other components to work.

- ❖ After we began implementing, we discovered new problems and solutions so we had 3 more group discussions, which amounted to a total duration of 8 hours.
- ❖ After we all wrote some amount of code expressing a basic structure, Debeshee did the first few rounds of integrations and testing.
- ❖ Followed by which Aashrith integrated and tested the Database with the code and then we all individually tested the code as much as possible.

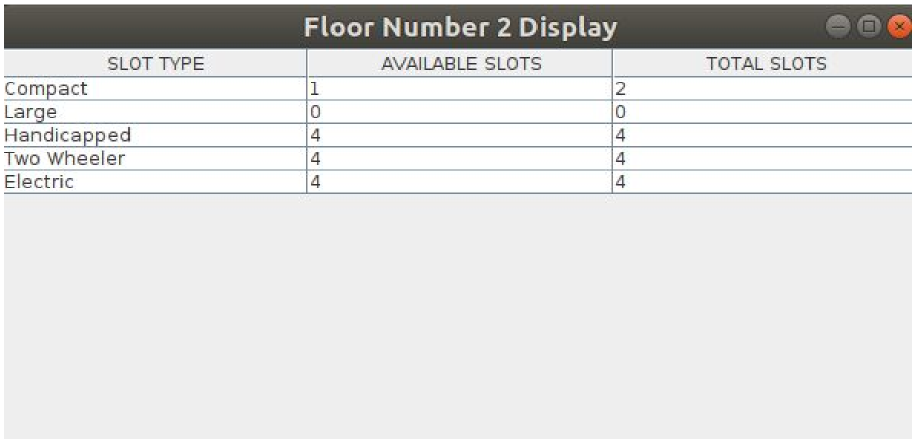
Other points to note:

We used Figma to design the user interfaces as well as the use case diagrams. They were drawn up by Debeshee and Anand based on the inputs of the entire team.

We taught each other many useful things in this learning process. Most of us developed some confidence using Git and GitHub as we were using it as a collaboration platform for the first time. (Aashrith gave us a crash course on Git). We learnt many interesting things about parameterized constructors with inheritance, we learnt to appreciate problems associated with Databases, and storing related objects in Databases and that Swing classes in Java are very slow!

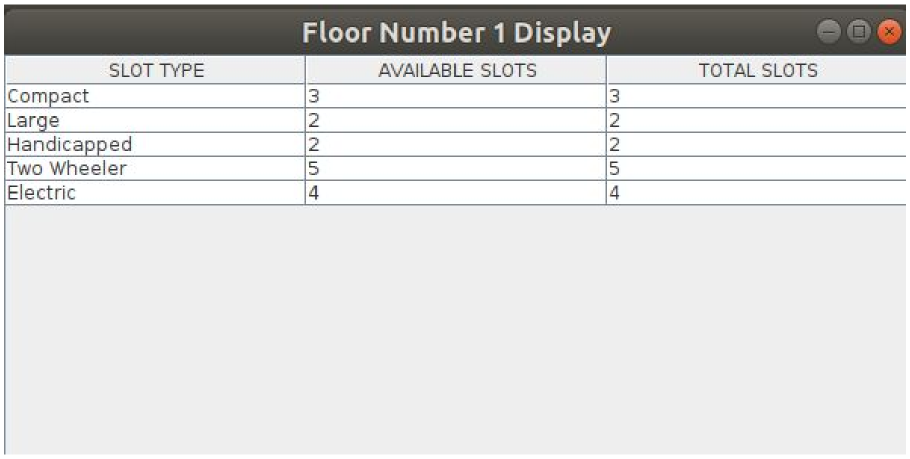
Some Screenshots of our GUI

Link to Figma file containing all diagrams



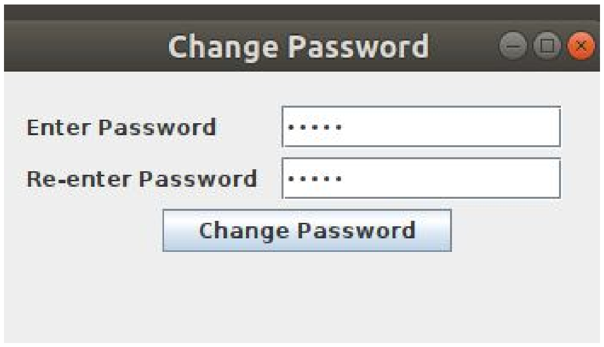
Floor Number 2 Display

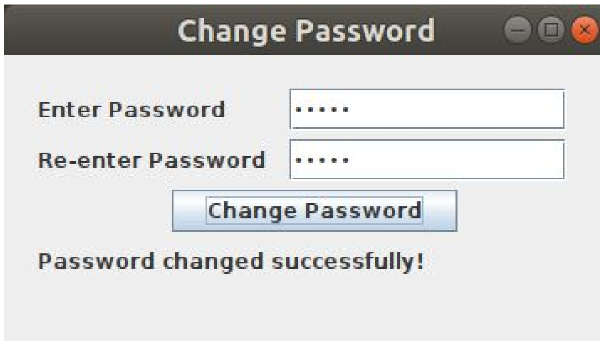
SLOT TYPE	AVAILABLE SLOTS	TOTAL SLOTS
Compact	1	2
Large	0	0
Handicapped	4	4
Two Wheeler	4	4
Electric	4	4



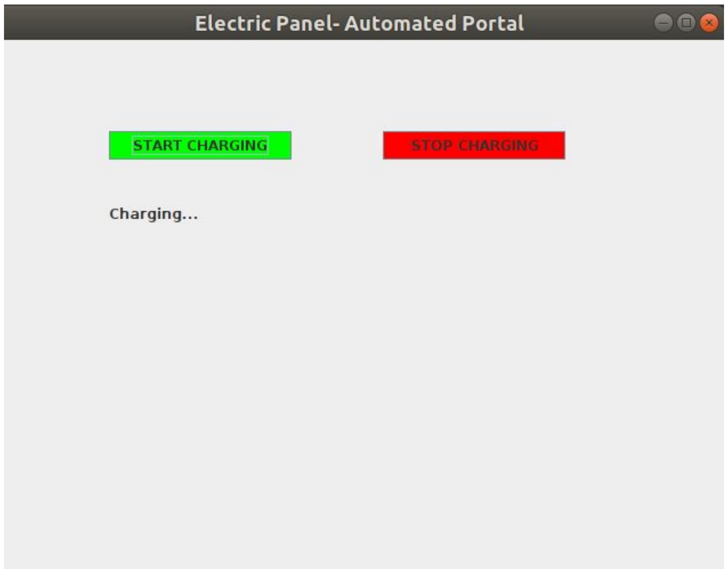
Floor Number 1 Display

SLOT TYPE	AVAILABLE SLOTS	TOTAL SLOTS
Compact	3	3
Large	2	2
Handicapped	2	2
Two Wheeler	5	5
Electric	4	4

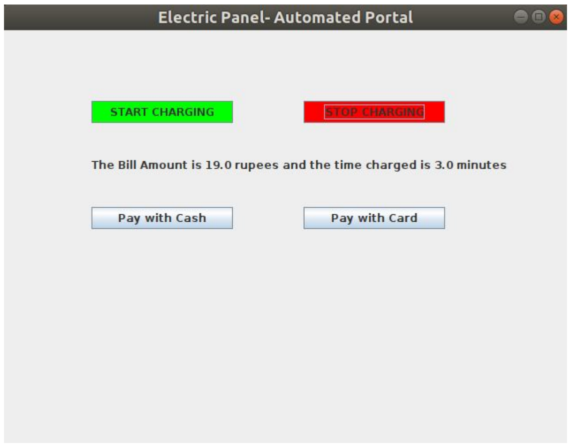




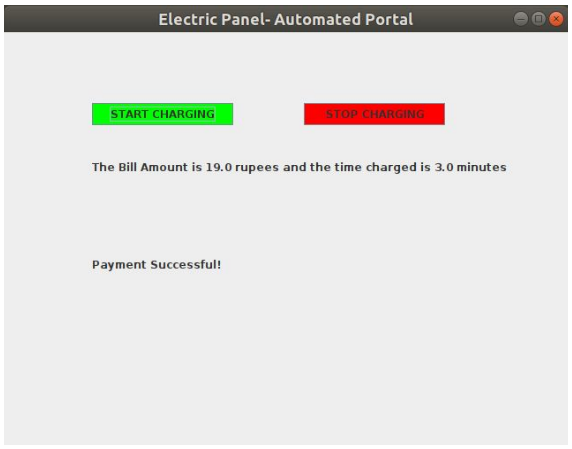
Password changed successfully!



Charging...

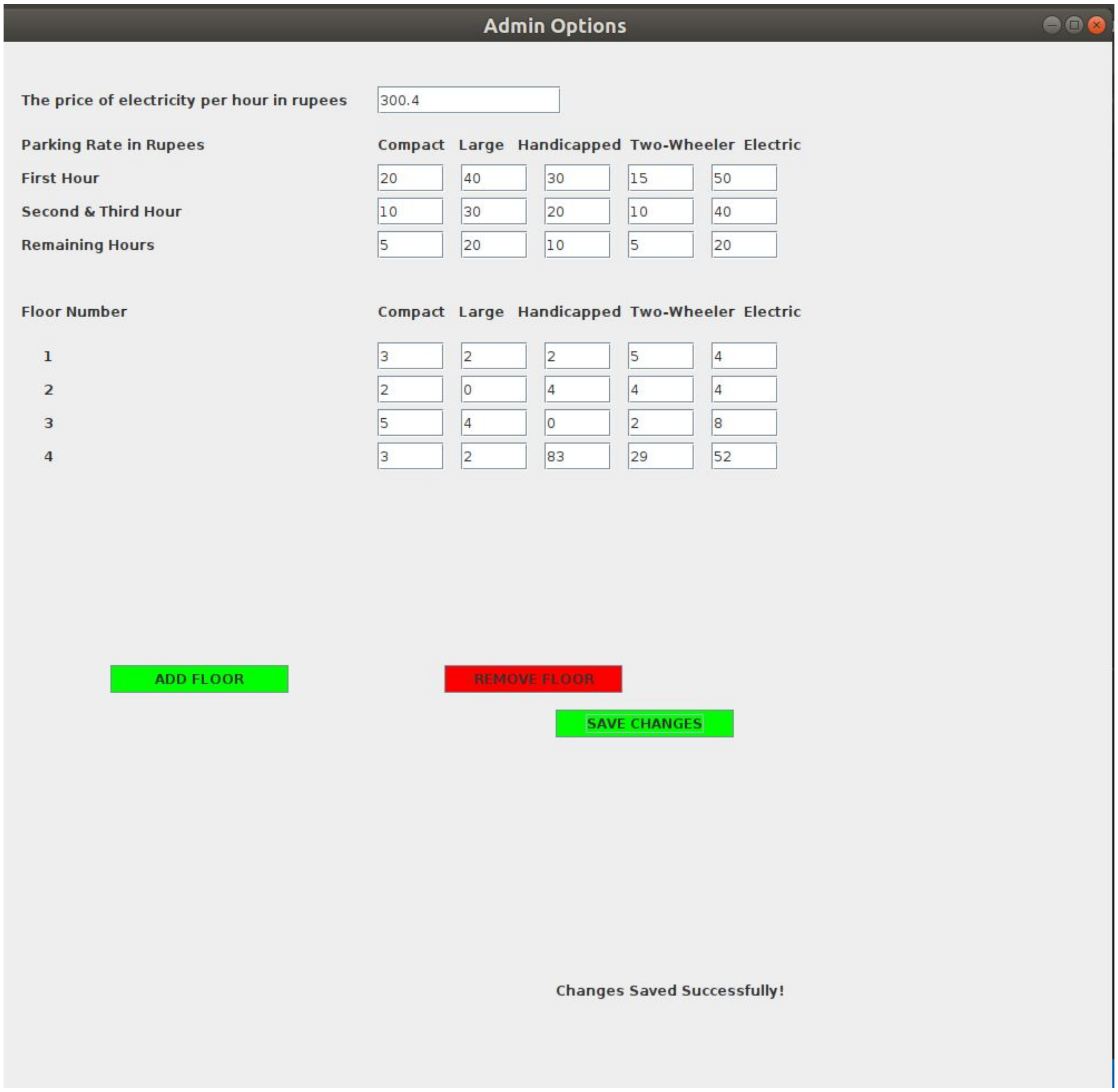


The Bill Amount is 19.0 rupees and the time charged is 3.0 minutes



The Bill Amount is 19.0 rupees and the time charged is 3.0 minutes

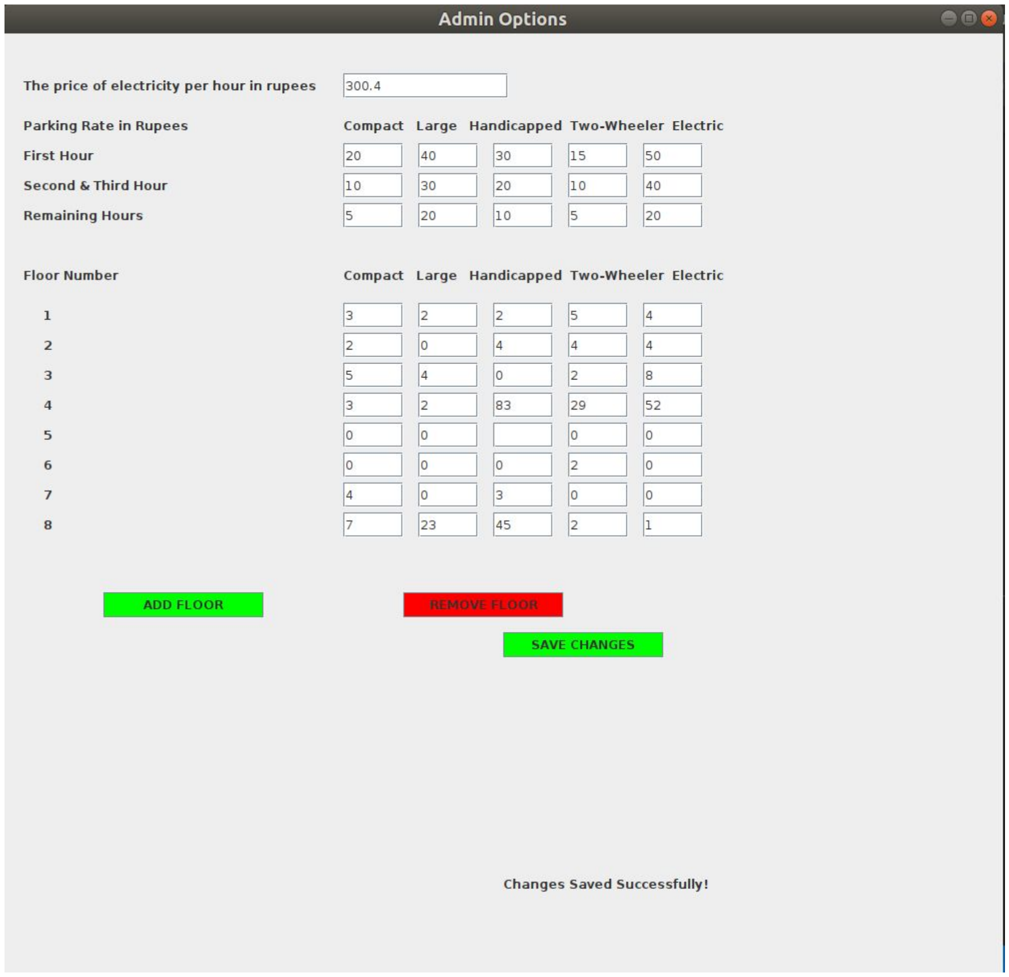
Payment Successful!


Admin Options

The price of electricity per hour in rupees	300.4				
Parking Rate in Rupees	Compact	Large	Handicapped	Two-Wheeler	Electric
First Hour	20	40	30	15	50
Second & Third Hour	10	30	20	10	40
Remaining Hours	5	20	10	5	20

Floor Number	Compact	Large	Handicapped	Two-Wheeler	Electric
1	3	2	2	5	4
2	2	0	4	4	4
3	5	4	0	2	8
4	3	2	83	29	52

Changes Saved Successfully!

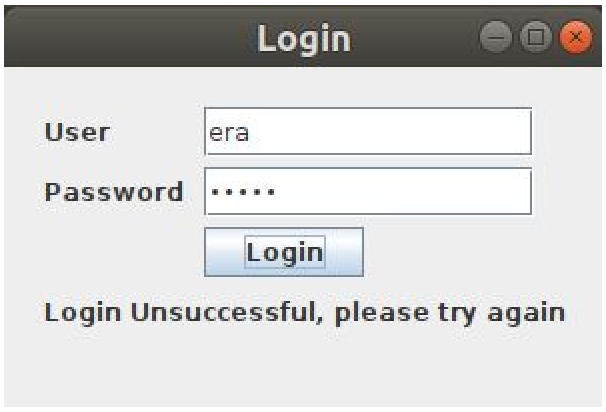

Admin Options

The price of electricity per hour in rupees 300.4

Parking Rate in Rupees	Compact	Large	Handicapped	Two-Wheeler	Electric
First Hour	20	40	30	15	50
Second & Third Hour	10	30	20	10	40
Remaining Hours	5	20	10	5	20

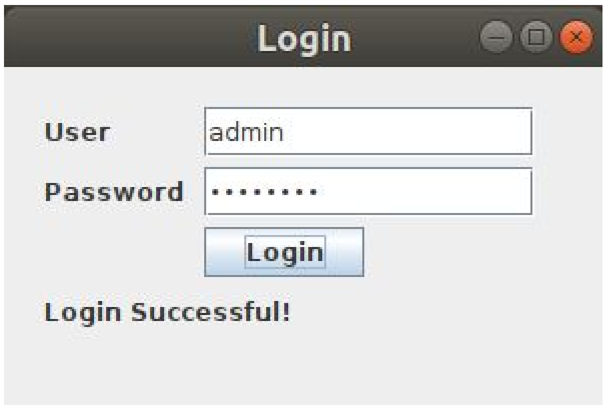
Floor Number	Compact	Large	Handicapped	Two-Wheeler	Electric
1	3	2	2	5	4
2	2	0	4	4	4
3	5	4	0	2	8
4	3	2	83	29	52
5	0	0		0	0
6	0	0	0	2	0
7	4	0	3	0	0
8	7	23	45	2	1

Changes Saved Successfully!


Login

User era
Password •••••

Login Unsuccessful, please try again


Login

User admin
Password ••••••••

Login Successful!

Rough work from our

Discussion from 1 - 8-10-2020

Handicapped – elevator
Admin – options
Types – of customers
Motorcycle – top floor –
Trucks lower floor-   Spot{

```
                    ○ Unique number – alphanumeric – info on both floor and type
                    ○ Which floor?
                    ○ What type?
                    ○ Has an electric panel or no?
                    ○ Is vacant or not?
}
Public class
{
Void main()
{         ParkingLot obj = new ParkingLot();
          obj.run();
}
}
-    class Parking lot
{
Void run()
{//fetch data from existing database and model into object
//check if admin or employee or customer

}
void setup()
{
}

}
-    {
                    ○ Floors{
                              § How many spots of each type – constant to be chose by admin
                              § Entry points – (set by admin)
                              § Exit points
                              Info portal
                              § Spots
                              §
                    ○ }
                    ○
 Class point
{
          Manned or automated

}
Class entry point extends point
{         public void run()
          {         //run all the private functions in order of whatever is required
          }
void display()
          {
          }
          Void customerInput(Object Customer)
          {
                    What kind of vehicle do you have?
                    //menu driven - Compact, Large, motorcycle, handicapped (think more)
                    //in the menu which is displayed when asking customer to choose we should large  means trucks,
buses, ...and compact means - small cars,
                    Is it electric or not?
                    if(electric)
                    {Do you want to charge or not?}
          }
          Void customerInput(Object Employee)
          {         What kind of vehicle do you have?
                    //menu driven - Compact, Large, motorcycle, handicapped (think more)
                    Is it electric or not?
                    if(electric)
                    {Do you want to charge or not?}
          }


}
Class exit point extends point
Class customer
{
          Type of vehicle
          Time of entry
```

```
        Time of exit
        Spot allotted
        Bill amount

}
        Class electric vehicle extend customer
        {
                Charge vehicle or not?
                Bill amount for charging= time charged+price of electricity(to be set by admin)
                Time charged
        }

        Class Employee extends Customer{
                 is_employee
        }

        Class Admin implements Employee{
                Private String password;
                Change password;
                Boolean loginstatus;
                boolean signIn()
                {       //take input
                        //check password is correct or not
                        Return true or false based on if login is successful
                }

        }

        Class Vehicle{

                Owner;
                Type;
                Number plate:

        }

        Class Transactions
        {

        }
```

*Optional idea - consider possibility of allotting one car space to 2 bikes if no space for bikes