COE 528 Design Project: Parking Lot

Summer 2020

Uzair Ahmed
Alireza Golband
Syed Wadood

The Problem

Multi Floor application consisting of:

1 manager that can modify, update, delete tickets

0 or more customer accessing the application

One file per ticket information

Ticket number is a random 10 digit number

The Problem

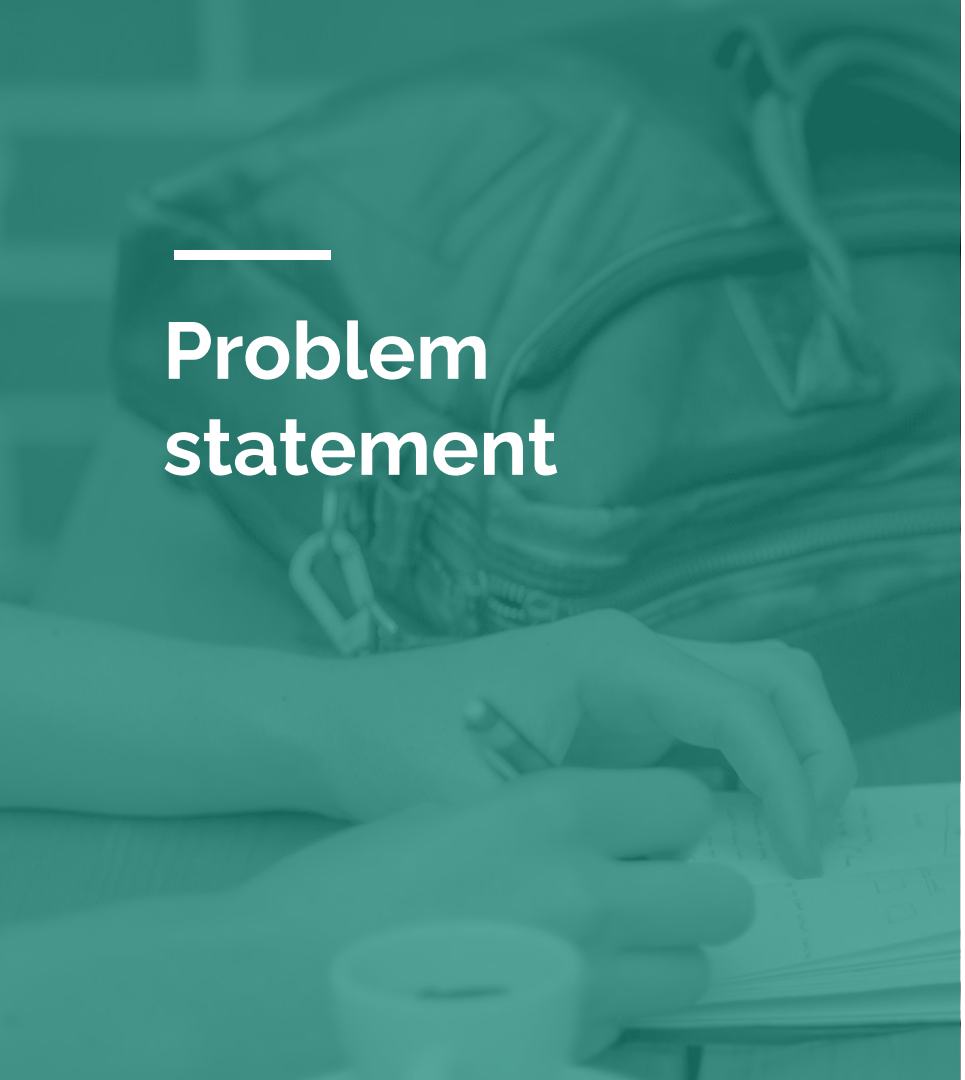
Multi Floor application consisting of:

Manager is verified upon login

Only manager can add, update and modify tickets

Purchased ticket will be stored in its file

No 2 tickets have same numbers



Problem statement

Modelling a real world problem using the approach of object oriented analysis and design approach

Creation of a multi floor parking space application and using the specific thinking and modelling.

Parking Lot Class

```java
public class ParkingLot {
    Lot mainLot;
    public ParkingLot() {
        this.mainLot = Lot.getInstance();
    }

    public static void main(String args[]){
        ParkingLot parkinglot = new ParkingLot();

        parkinglot.mainLot.createFloor(30, 20, 10, 10, 5);
        parkinglot.mainLot.createFloor(10, 50, 10, 0, 5);
        parkinglot.mainLot.createFloor(10, 20, 20, 10, 15);

        parkinglot.mainLot.getCapacity();
        parkinglot.mainLot.getAvailableCapacity();

        Space tempSpace = parkinglot.mainLot.entrance1.chooseAnEmptySpace("compact");
        Ticket t = parkinglot.mainLot.entrance1.createNewTicket(tempSpace, "John", 0);

    }
}
```

This class does not impact the Interface but it allows for testing of class functions in order to guarantee execution.

Lot Class

```java
public class Lot {
    Entrance entrance1;
    Exit exit1;
    Exit exit2;
    public int totalNumTickets;
    public int numFloors;
    public ArrayList<Floor> floors = new ArrayList();

    private static Lot instance = null;

    private Lot() {
        entrance1 = new Entrance();
        exit1 = new Exit();
        exit2 = new Exit();

        this.totalNumTickets = 0;
        this.numFloors = 0;
    }
}
```

This class is a Singleton, it is only made once and is essential to represent the whole parking lot. It handles all the floors/spaces and keeps track of the quantity of each floor. For our purposes, we made 3 floors with 5 spaces on each floor.

Floor Class

```java
public class Floor {
    public ArrayList<ArrayList <Space>> spaces;
    public int id ;
    public int[] floorCapacity;
    public int[] availableFloorCapacity;
    public Map<String, Integer> map = new HashMap<>();

    public Floor(int compactSlots, int largeSlots, int handicapSlots, int
motorcycleSlots, int EVSlots) {
        this.id = Lot.getInstance().numFloors+1;
        Lot.getInstance().numFloors++;
        Lot.getInstance().floors.add(this);

        this.spaces = new ArrayList<>();
        this.spaces.add(new ArrayList<>());
        this.spaces.add(new ArrayList<>());
        this.spaces.add(new ArrayList<>());
        this.spaces.add(new ArrayList<>());
        this.spaces.add(new ArrayList<>());

        this.floorCapacity = new int[5];
        this.availableFloorCapacity = new int[5];

        map.put("compact", 0);
        map.put("large", 1);
        map.put("handicap", 2);
        map.put("motorcycle", 3);
        map.put("electric", 4);
    }
}
```

This class manages the parking spaces on a floor-by-floor basis.

Space Class

```
public class Space {
    public Floor floor;
    public int id;
    public String type;
    public boolean full;
    public Ticket user;

    public Space(String type, Floor floor) {
        this.floor = floor;
        this.type = type;
        this.full = false;

        this.id = floor.getTotalFloorCapacity()+1;

        floor.spaces.get(floor.map.get(this.type)).add(this);
        floor.floorCapacity[floor.map.get(this.type)]++;
        floor.availableFloorCapacity[floor.map.get(this.type)]++;
    }
}
```

This class manages a single Parking Space and whether or not it is currently being used.

EVSpace class

```
public class EVSpace extends Space{

    public EVSpace(String type, Floor floor) {
        super(type, floor);
    }

    public void charge(){
        this.user.extraCosts+=10.00;
    }
}
```

This class is a Child of Space, it adds a function to apply extra charging costs for Electric Vehicles.

Ticket Class

```java
public class Ticket {
    public long identifier;
    public Space space;
    public String customerName;
    public int timeStarted;
    public double cost = 0;
    public double extraCosts = 0;
    public boolean paid;

    public Ticket(Space space, String customerName, int timeStarted) {
        this.identifier = (long) Math.floor(Math.random() * 9_000_000_000L) +
1_000_000_000L;
        this.space = space;
        Lot.getInstance().totalNumTickets++;
        this.customerName = customerName;
        this.timeStarted = timeStarted;
        this.space.setFull(true, this);

        System.out.println("Made a ticket for "
                + customerName
                + " in "
                + String.valueOf(space.floor.id)
                + "-"
                + String.valueOf(space.id));
        createTicketFile();
    }
}
```

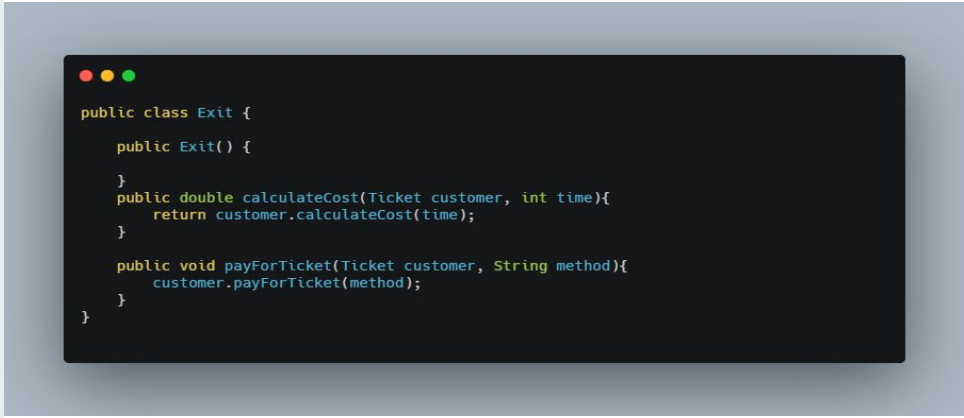
This class manages a single Ticket, and records the time elapsed and charges incurred for a specific parking.

Entrance Class

```java
public class Entrance {
    public Entrance( ) {
    }

    public Space chooseAnEmptySpace(String type){
        for (Floor f : Lot.getInstance().floors){
            if (f.getAvailableFloorCapacity()[f.map.get(type)] > 0){
                System.out.println("got "
                        + f.getSpaces(type).size()
                        + " free spaces in floor "
                        + f.id
                        + " of type "
                        + type
                );
                return f.getSpaces(type).get(0);
            }
        }
        return null;
    }


    public Ticket createNewTicket(Space space, String customer, int time){
        if (space != null){
            Ticket ticket = new Ticket(space, customer, time);
            return ticket;
        }
        else return null;
    }
}
```

This class handles handles parking space selection, and Ticket creation upon entering.

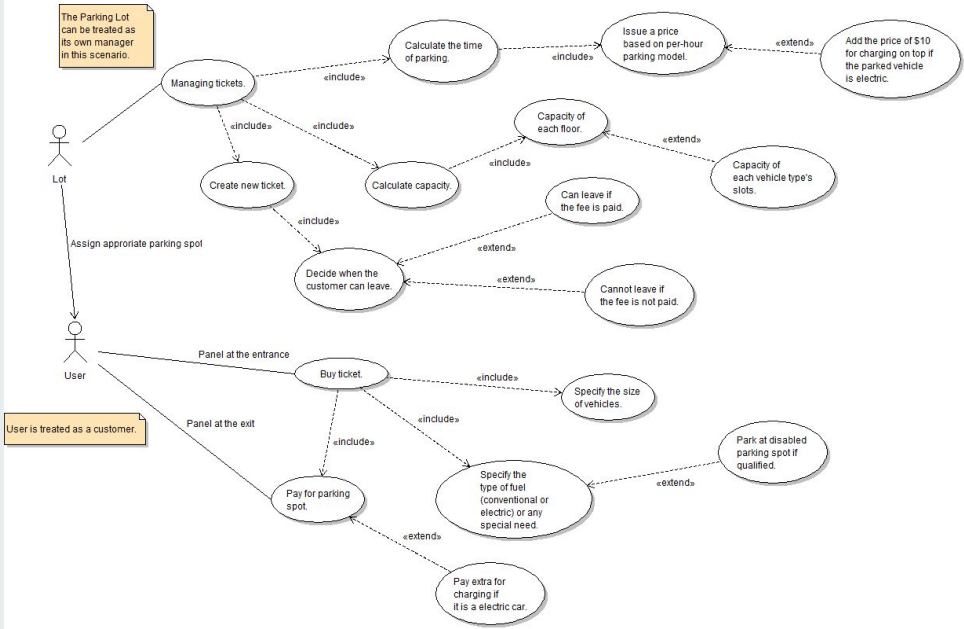
Exit Class



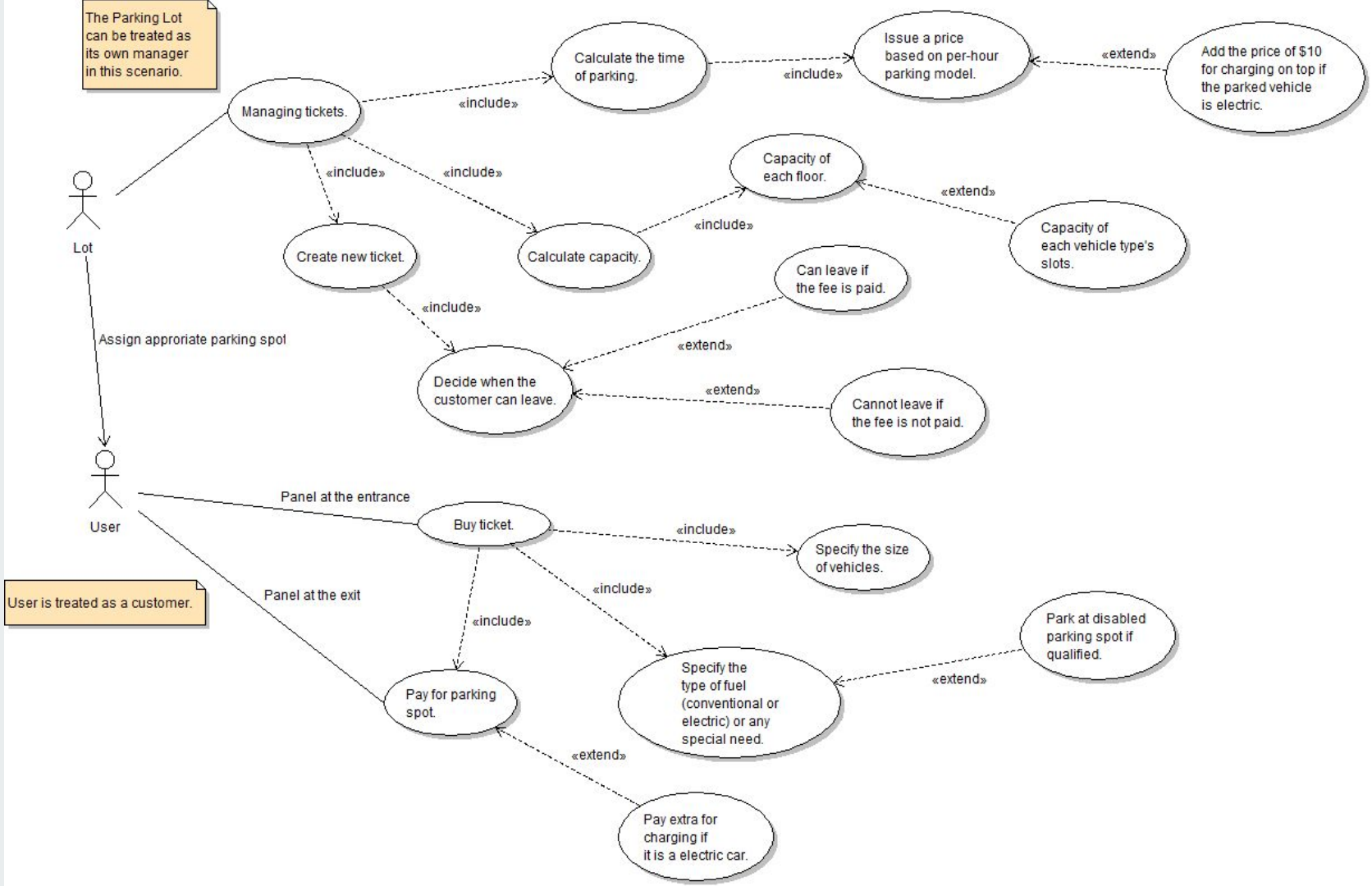
```java
public class Exit {

    public Exit() {

    }
    public double calculateCost(Ticket customer, int time){
        return customer.calculateCost(time);
    }

    public void payForTicket(Ticket customer, String method){
        customer.payForTicket(method);
    }
}
```

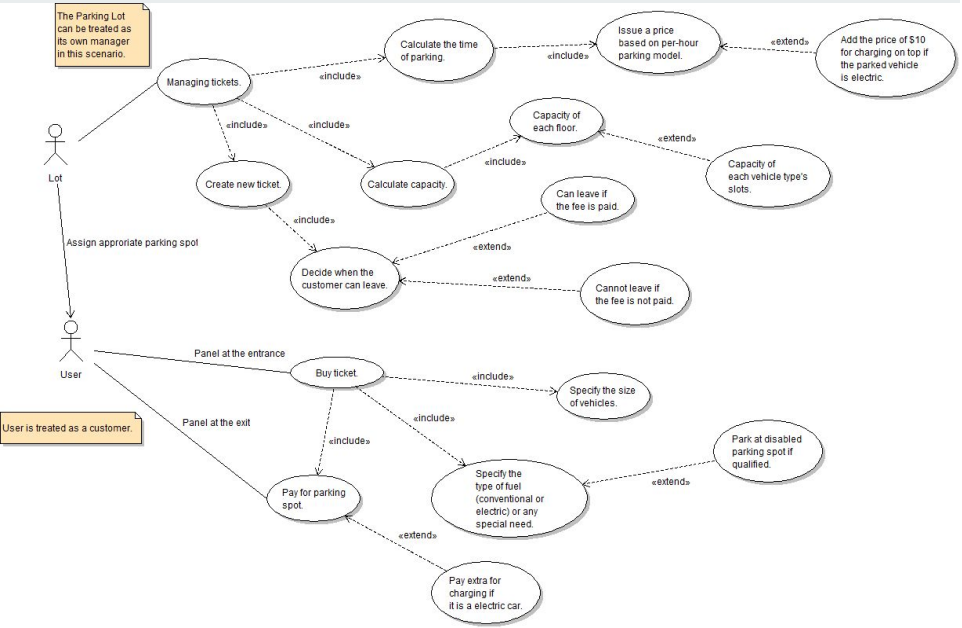
This class handles handles Ticket deletion, and Customer payment upon exit.

Use Case



Larger picture is included in next slide


The Parking Lot can be treated as its own manager in this scenario.
Managing tickets.
Calculate the time of parking.
Issue a price based on per-hour parking model.
Add the price of $10 for charging on top if the parked vehicle is electric.
«include»
«include»
«extend»
Lot
«include»
«include»
Create new ticket.
Calculate capacity.
Capacity of each floor.
Capacity of each vehicle type's slots.
«include»
«extend»
Assign approriate parking spot
«include»
Can leave if the fee is paid.
«extend»
Decide when the customer can leave.
«extend»
Cannot leave if the fee is not paid.
User
Panel at the entrance
Buy ticket.
«include»
Specify the size of vehicles.
User is treated as a customer.
Panel at the exit
«include»
«include»
Park at disabled parking spot if qualified.
Specify the type of fuel (conventional or electric) or any special need.
«extend»
Pay for parking spot.
«extend»
Pay extra for charging if it is a electric car.



Division into multiple parts

Actors - Lot and the user/customer

Relations are specifically stated with include and extend so the reader understands clearly

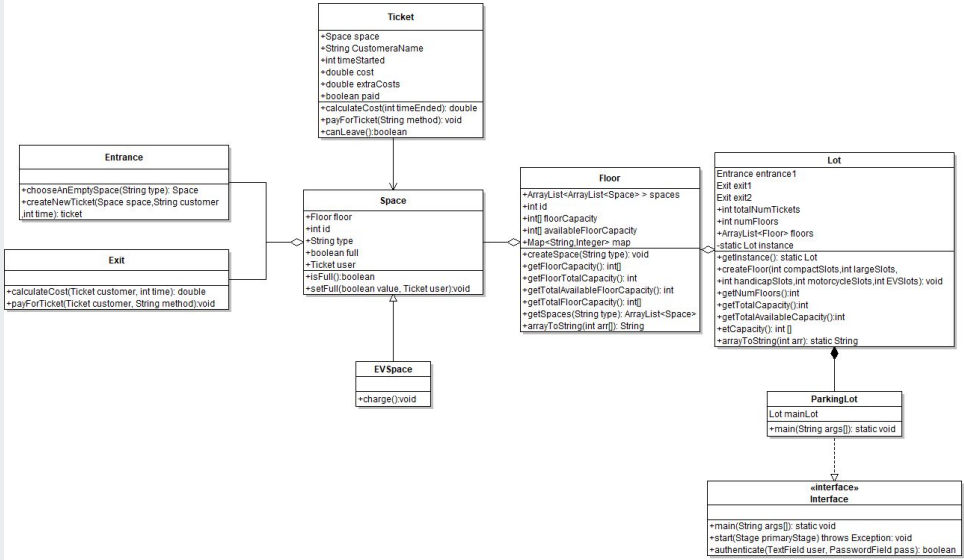
Relationships expand for Lot - managing tickets to calculating the time of parking and price being issued

New paths from managing tickets- creation of new tickets and when a customer can leave and capacity calculation

User relationship to buy ticket - size of vehicle, the type of car and method of payment

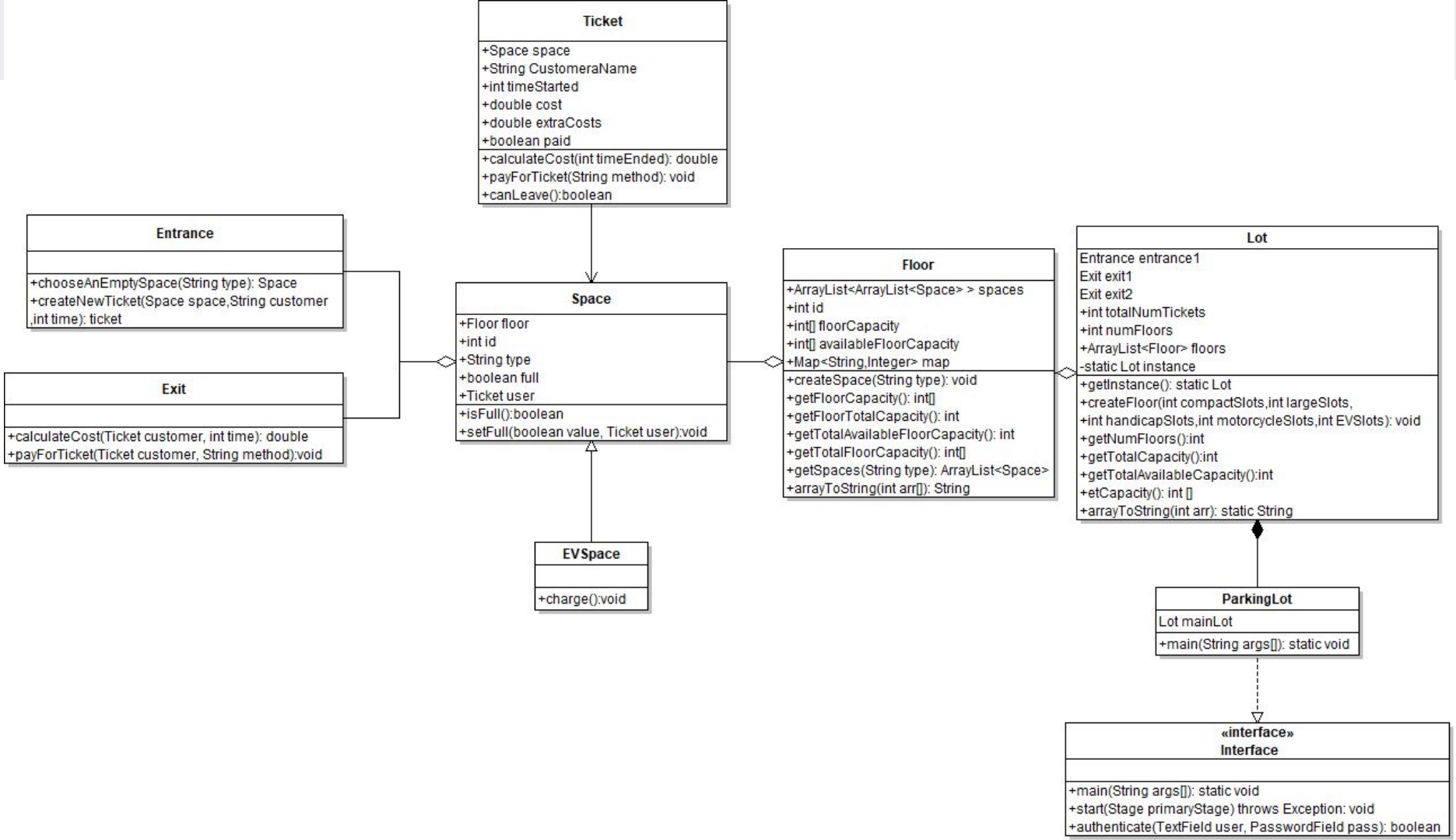
Class Diagram

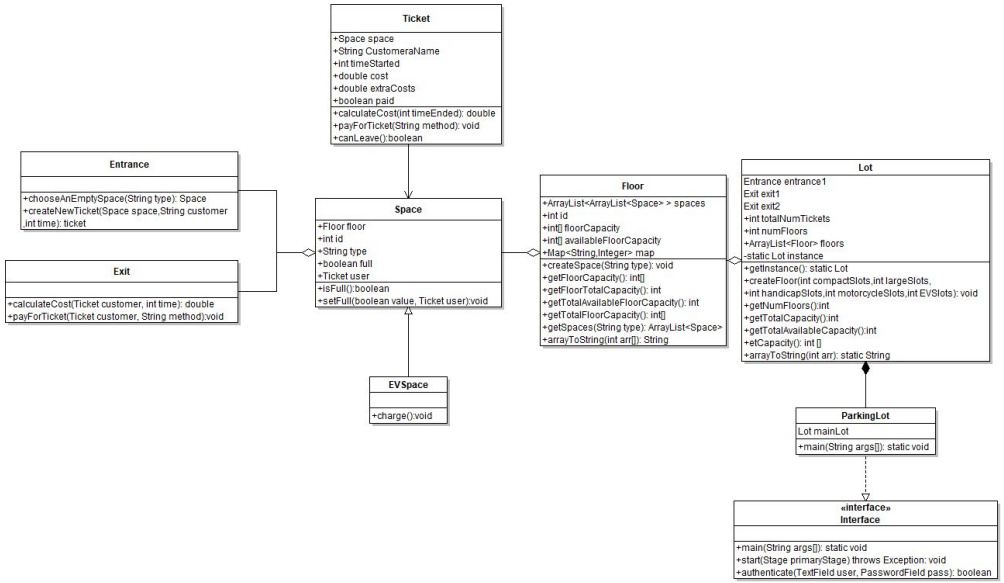
02



Larger picture is included in next slide

- Visualize the effect of classes
- Understanding the System
- Structure and Relationship of classes


Ticket
+Space space
+String CustomeraName
+int timeStarted
+double cost
+double extraCosts
+boolean paid
+calculateCost(int timeEnded): double
+payForTicket(String method): void
+canLeave():boolean
Entrance
+chooseAnEmptySpace(String type): Space
+createNewTicket(Space space,String customer
,int time): ticket
Exit
+calculateCost(Ticket customer, int time): double
+payForTicket(Ticket customer, String method):void
Space
+Floor floor
+int id
+String type
+boolean full
+Ticket user
+isFull():boolean
+setFull(boolean value, Ticket user):void
EVSpace
+charge():void
Floor
+ArrayList<ArrayList<Space> > spaces
+int id
+int[] floorCapacity
+int[] availableFloorCapacity
+Map<String,Integer> map
+createSpace(String type): void
+getFloorCapacity(): int[]
+getFloorTotalCapacity(): int
+getTotalAvailableFloorCapacity(): int
+getTotalFloorCapacity(): int[]
+getSpaces(String type): ArrayList<Space>
+arrayToString(int arr[]): String
Lot
Entrance entrance1
Exit exit1
Exit exit2
+int totalNumTickets
+int numFloors
+ArrayList<Floor> floors
-static Lot instance
+getInstance(): static Lot
+createFloor(int compactSlots,int largeSlots,
+int handicapSlots,int motorcycleSlots,int EVSlots): void
+getNumFloors():int
+getTotalCapacity():int
+getTotalAvailableCapacity():int
+etCapacity(): int []
+arrayToString(int arr): static String
ParkingLot
Lot mainLot
+main(String args[]): static void
«interface»
Interface
+main(String args[]): static void
+start(Stage primaryStage) throws Exception: void
+authenticate(TextField user, PasswordField pass): boolean



Elements represented using the “+” sign as the public members of classes

Inheritance Relationship- Space class from EVSpace and Ticket Class (Arrow)

Composition Relationship- Space class and Exit class, Floor class and Space class (Full diamonds)

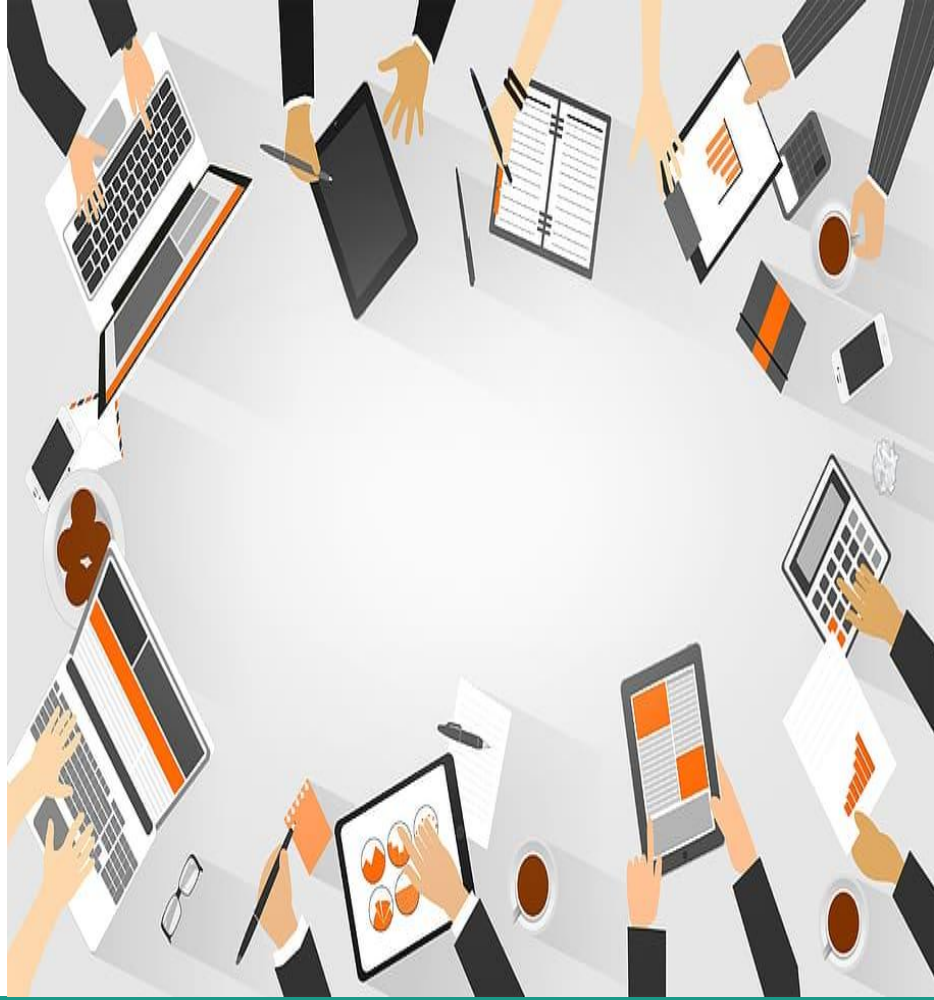
Aggregation Relationship- Lot class and Parking lot Class

Responsibilities

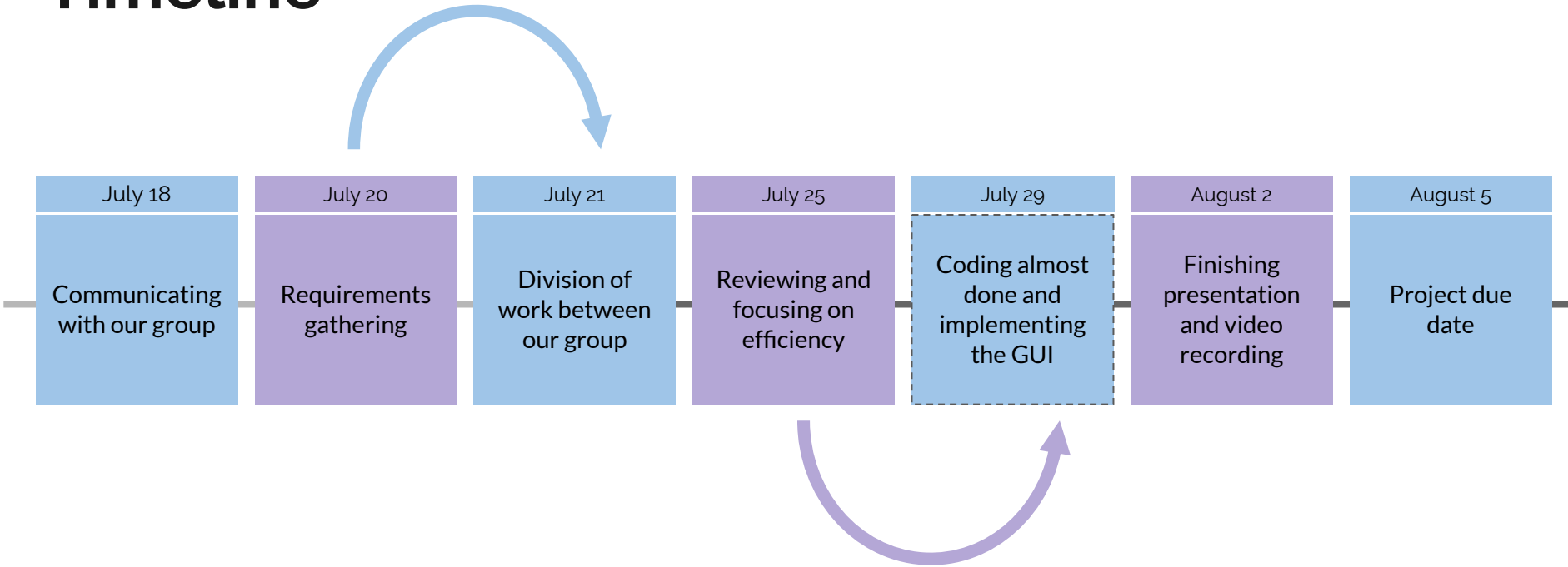
Abdul- Design and Interface, Exception Handling

Uzair- Classes, Debugging, JavaDoc

Alireza - UML, Report, Presentation



Timeline


July 18
Communicating with our group
July 20
Requirements gathering
July 21
Division of work between our group
July 25
Reviewing and focusing on efficiency
July 29
Coding almost done and implementing the GUI
August 2
Finishing presentation and video recording
August 5
Project due date

References

What Is Unified Modeling Language (UML)? Accessed July 5, 2020. https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/.

UML Use Case Diagram Tutorial. (n.d.). Retrieved July 15, 2020, from https://www.lucidchart.com/pages/uml-use-case-diagram