Coe 528 Course Project

Summer 2020

Syed Wadood
Uzair Ahmed
Alireza Golband

Ryerson University

Department of Electrical & Computer Engineering
Faculty of Engineering & Architectural Science

Course Title:	**COE**
Course Number:	**528**
Semester/Year (e.g.F2016):	**S2020**

Instructor:	**Ahmed Ibrahim**

Assignment/Lab Number:	Final Project
Assignment/Lab Title:	Multi Floor Parking Application

Submission Date:	Aug 5,2020
Due Date:	Aug 5,2020

Student LAST Name	**Student FIRST Name**	**Student Number**	**Section**	**Signature**
Syed	Abdul Wadood	500892009		Syed Wadood
Golband	Alireza	500895725	02	Alireza Golband
Ahmed	Uzair	500905189	02	Uzair Ahmed

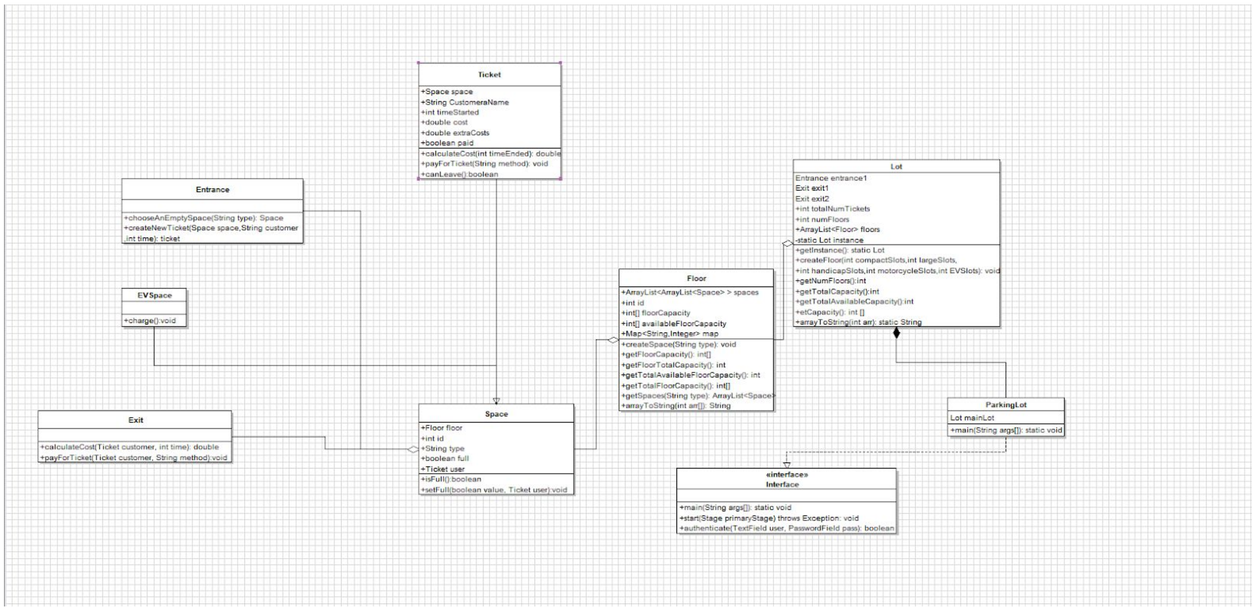
Overview

This report consists of sections explaining different parts of the final project for the COE 528 course. In an overview, topics such as Use Case Diagram, Class Diagram, UML, and explanation of multiple implementations is discussed in detail.

Goals

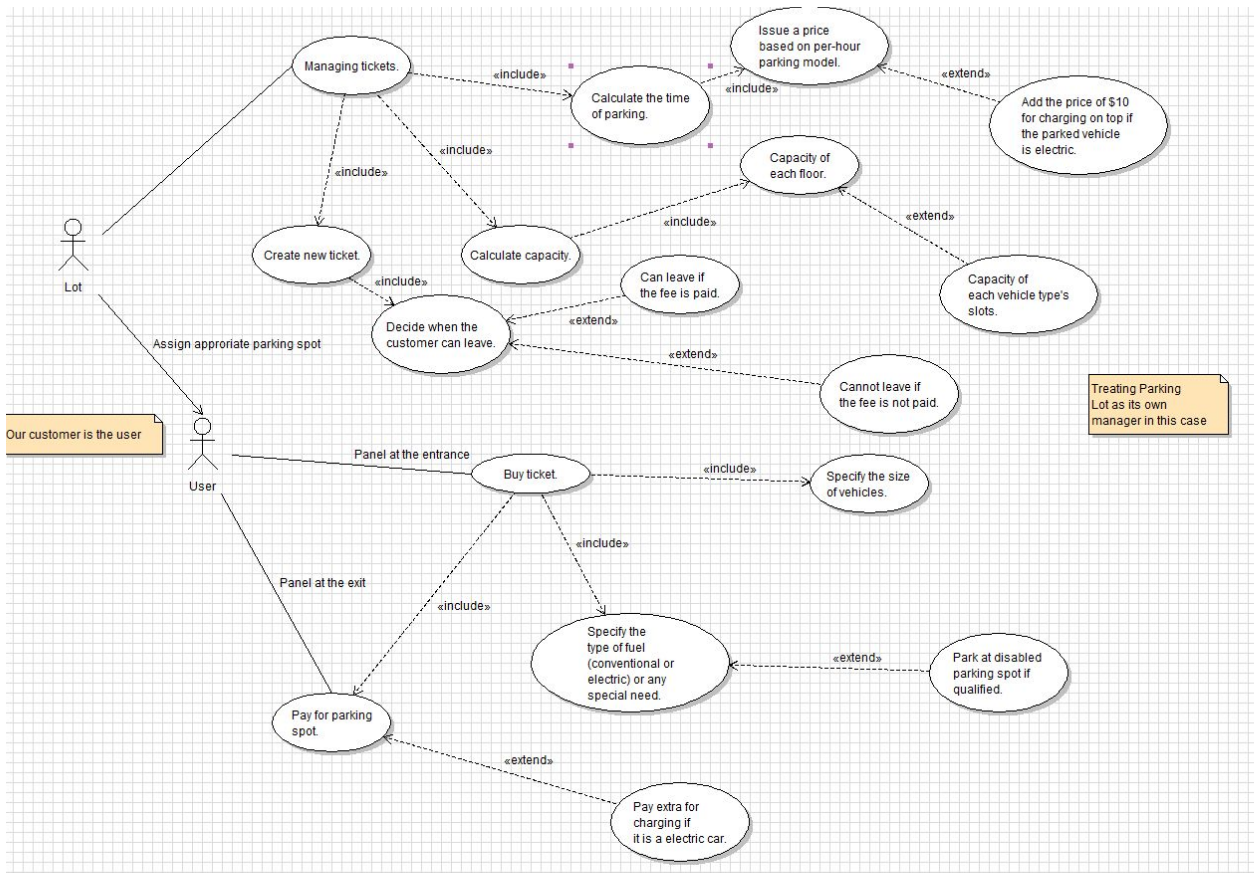
Modeling a real-world problem by the use of object-oriented analysis and design approach is our goal and this project is designed as an application for a multi floor parking space which requires multiple requirements for the two parties involved: a manager and any number of customers within each floor.

Class Diagram Explained


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
EVSpace
+charge():void
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
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

The Class Diagram is included as a picture, however the Violet file is submitted through D2L. It is very common to use class diagrams in order to visualize the effect of classes and this class diagram enables us to understand the system and the parking lot application in general much better as it illustrates the structure and the relationships of our classes specified. The elements represented in our class boxes are represented using the "+" sign as they are public members of the class. The inheritance relationship is showcased towards the Space class by EVSpace and Ticket Class as the arrow is used. Composition relation is showcased for the Space class with in regards to Exit class and Floor class in regards to Space class as shown by the full diamonds. The aggregation relation is also shown for the Lot class with regards to the parking lot class on the right side of the diagram. Lastly the Uses relation is also shown at the interface class with regards to the ParkingLot class.

Use Case Explained



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

From our technical knowledge, the Use Case is provided to represent the external behavior during the requirement elicitation and analysis. Even more, this part of the project represents the boundaries of the system as it captures the main functionality of the system from an important point of view which is the actor's or user perspective. This Use Case was implemented to act as a vehicle in order to divide our system into multiple parts and that division enabled us to implement each single part separately for this project. The lot and the user are the dedicated actors on the left of the diagram and furthermore, the relationship begins to expand the cases for Lot from managing tickets to calculating the time of parking and issuing a price based on per-hour parking model. From the managing tickets diagram two new paths are also emerged where new tickets are created which then decides when the customer can leave and also the capacity is calculated. Similarly for the user, the relationship is connected to Buy Ticket which will lead to specifying the size of vehicle, what type of car they are(electric or normal or disabled) and how they want to pay for parking spots. The relationship is also explained in detail in the diagram with the usage of include and extend to furtherly communicate with the audience.

Addressing point #2

The chosen class for point two was the Ticket class, which manages the Customers parking ticket, and makes sure that text files are stored to the computer. All functions inside Ticket include their REQUIRES/EFFECTS/MODIFIES clauses to overview what each method does.

UML & Design Patterns

Design patterns are popular among software projects such as this application and they are defined as a well described solution which solves a common software problem. The usage of the design pattern is beneficial as they are already defined and more importantly they allow the code to be standardized and allows us to approach solving the problems in the software by saving time. The design patterns promote the reusability and allow us to have maintainable code. They also benefited the code as it made it easier to understand problems and debugging our code. Similarly the usage of class diagram and combination with design patterns are the reason that the code is separated but each has a specific task to do, from allowing the customer to pay with different payment option, and calculating the amount of money and car, to enabling them to input the result and designing an interactive solution.