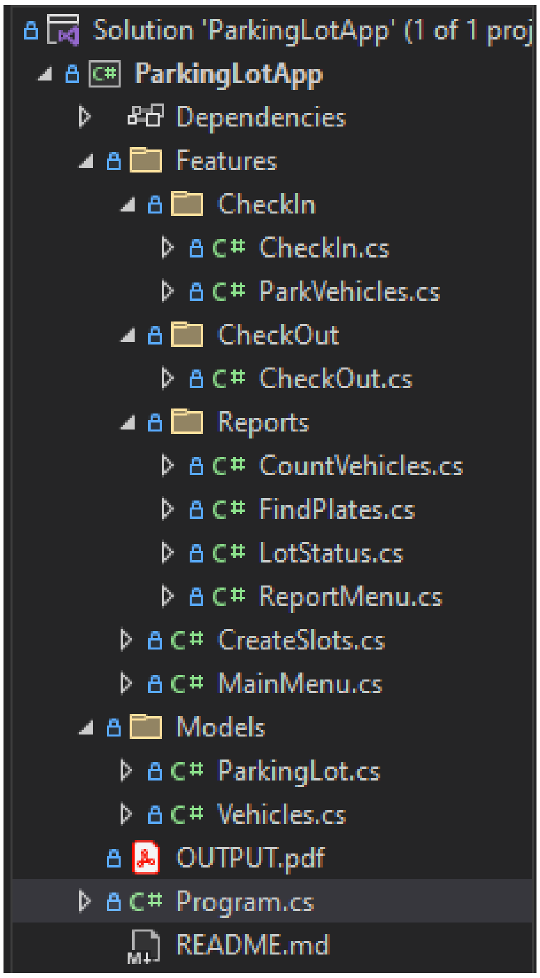
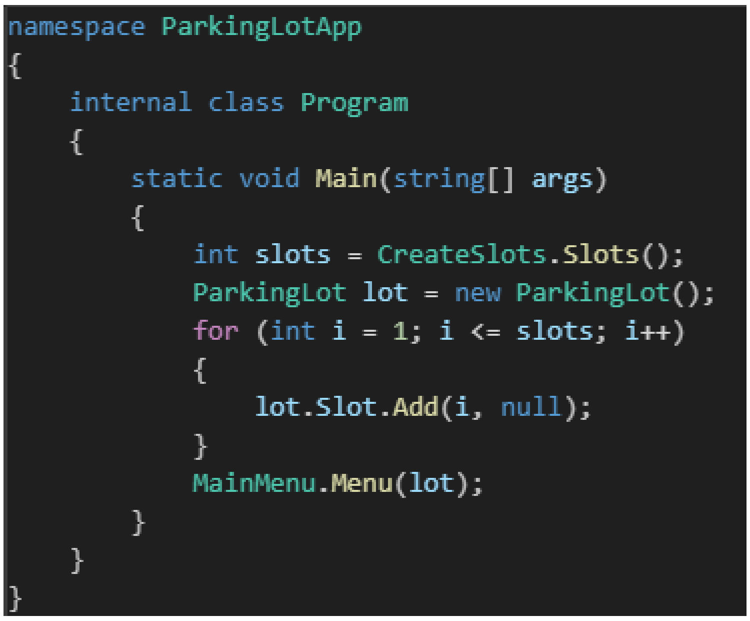
Folder Directories



```
Solution 'ParkingLotApp' (1 of 1 proj
  ParkingLotApp
    Dependencies
    Features
      CheckIn
        CheckIn.cs
        ParkVehicles.cs
      CheckOut
        CheckOut.cs
      Reports
        CountVehicles.cs
        FindPlates.cs
        LotStatus.cs
        ReportMenu.cs
      CreateSlots.cs
      MainMenu.cs
    Models
      ParkingLot.cs
      Vehicles.cs
    OUTPUT.pdf
    Program.cs
    README.md
```

Program / Main Class

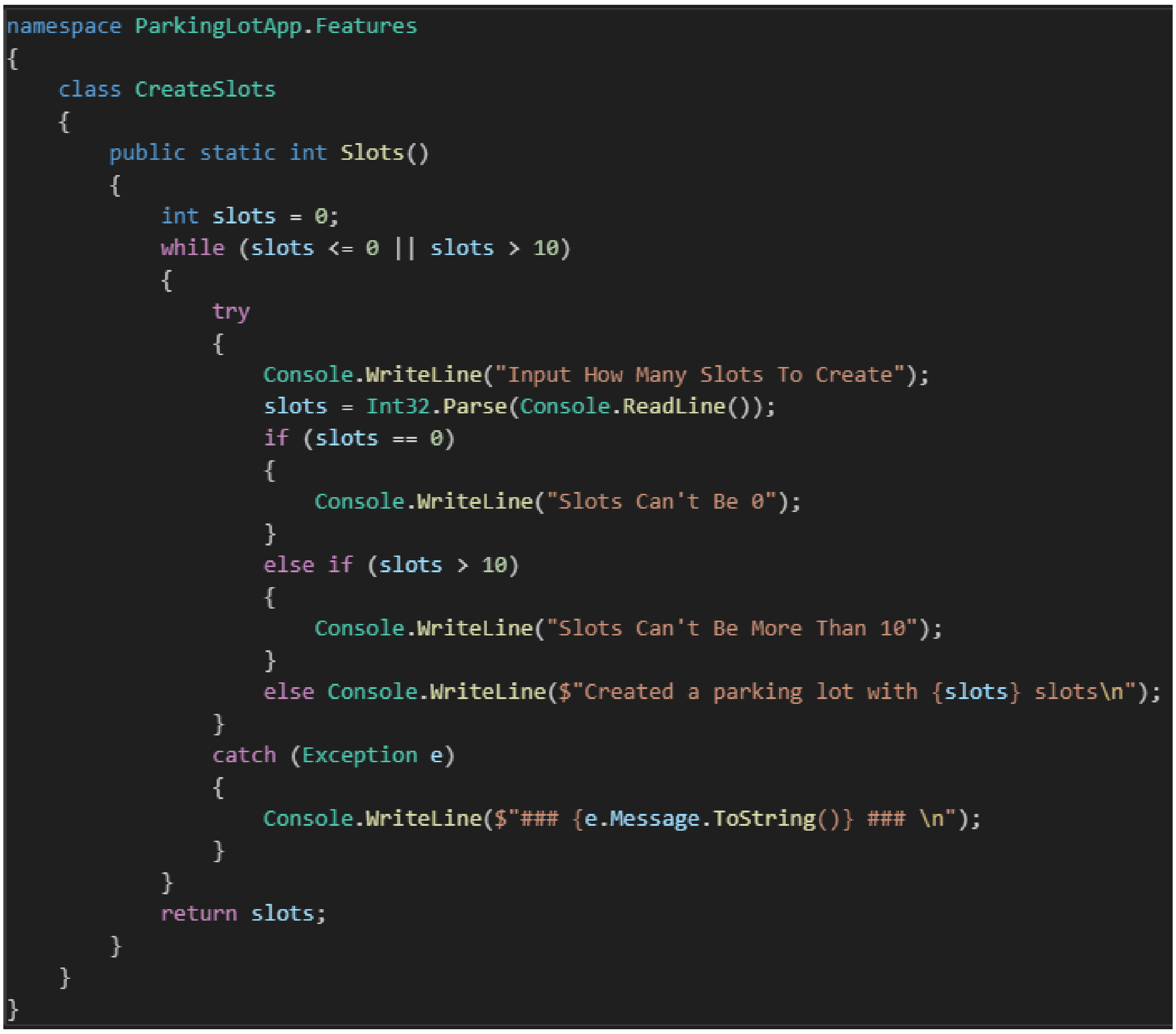


```
namespace ParkingLotApp
{
    internal class Program
    {
        static void Main(string[] args)
        {
            int slots = CreateSlots.Slots();
            ParkingLot lot = new ParkingLot();
            for (int i = 1; i <= slots; i++)
            {
                lot.Slot.Add(i, null);
            }
            MainMenu.Menu(lot);
        }
    }
}
```

Implementing OOP concepts, I tried my best to make everything separated as specific objects, so my Main would be look minimalist and also implementing OOP concepts.

The for loops is to make / generate a Slot No for ParkingLot, as Parking Lot actually have Dictionary collection type in it where I will show you in the next.

CreateSlots Class

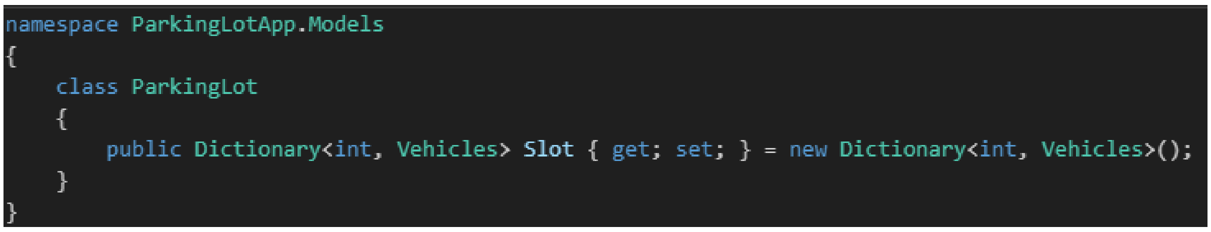


```
namespace ParkingLotApp.Features
{
    class CreateSlots
    {
        public static int Slots()
        {
            int slots = 0;
            while (slots <= 0 || slots > 10)
            {
                try
                {
                    Console.WriteLine("Input How Many Slots To Create");
                    slots = Int32.Parse(Console.ReadLine());
                    if (slots == 0)
                    {
                        Console.WriteLine("Slots Can't Be 0");
                    }
                    else if (slots > 10)
                    {
                        Console.WriteLine("Slots Can't Be More Than 10");
                    }
                    else Console.WriteLine($"Created a parking lot with {slots} slots\n");
                }
                catch (Exception e)
                {
                    Console.WriteLine($"### {e.Message.ToString()} ### \n");
                }
            }
            return slots;
        }
    }
}
```

This is the code that makes the Parking lot's Slots, I tried to limit the slots number to not more than 10 just because I want to limit user's input just like how normal programs works.

And after this code successfully running, this method will return an integer slots that will be the value of how many Slot of Parking Lot Class slots gonna have in the future of the program.

ParkingLot Class

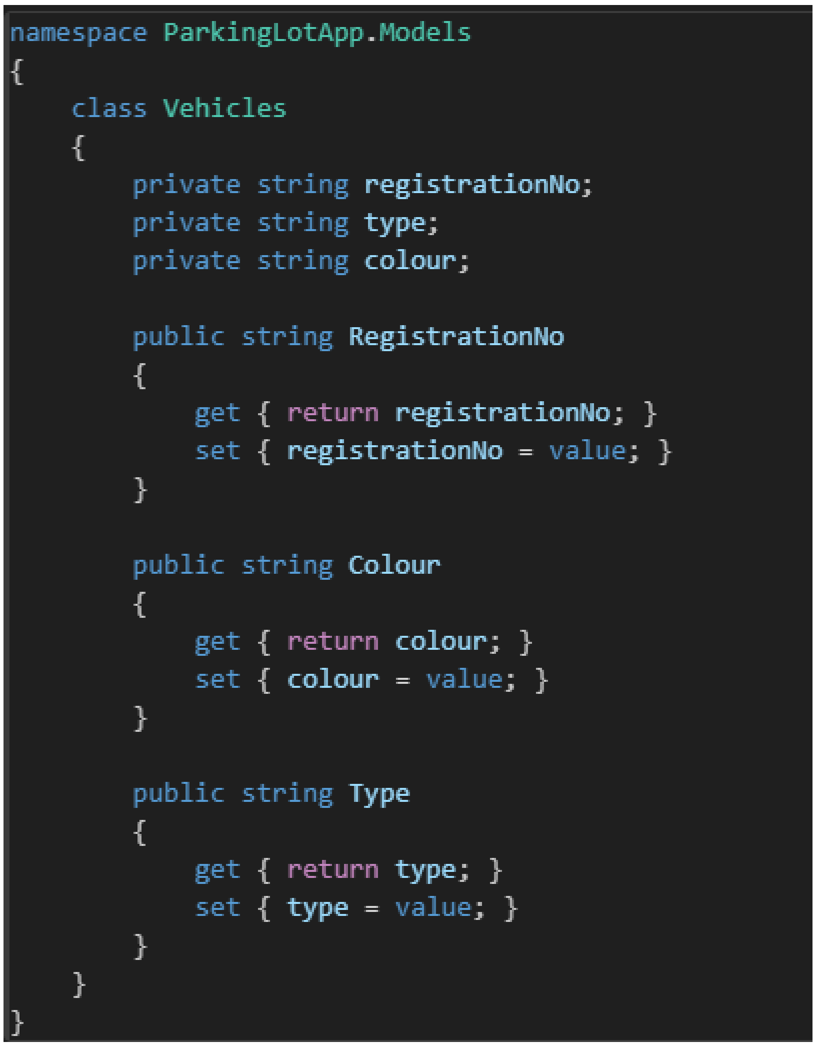


```
namespace ParkingLotApp.Models
{
    class ParkingLot
    {
        public Dictionary<int, Vehicles> Slot { get; set; } = new Dictionary<int, Vehicles>();
    }
}
```

I choose to set a ParkingLot as a Model with Dictionary collection type as its attribute, as I thought Dictionary have Key and Value, so it would be easier to manipulate things that only have 1 key and 1 value just like in this case, where 1 Slot contains 1 Vehicle.

Where the total int of Slot dictionary / key of the dictionary will be added by Create Slots method previously.

Vehicles Class



```
namespace ParkingLotApp.Models
{
    class Vehicles
    {
        private string registrationNo;
        private string type;
        private string colour;

        public string RegistrationNo
        {
            get { return registrationNo; }
            set { registrationNo = value; }
        }

        public string Colour
        {
            get { return colour; }
            set { colour = value; }
        }

        public string Type
        {
            get { return type; }
            set { type = value; }
        }
    }
}
```

This is the Vehicles object Class, that is going to be added into ParkingLot dictionary along with the Slot No. I use private access modifier for this Object / Class and then create a Getter & Setter to implement another OOP concepts about Encapsulation.

MainMenu Class

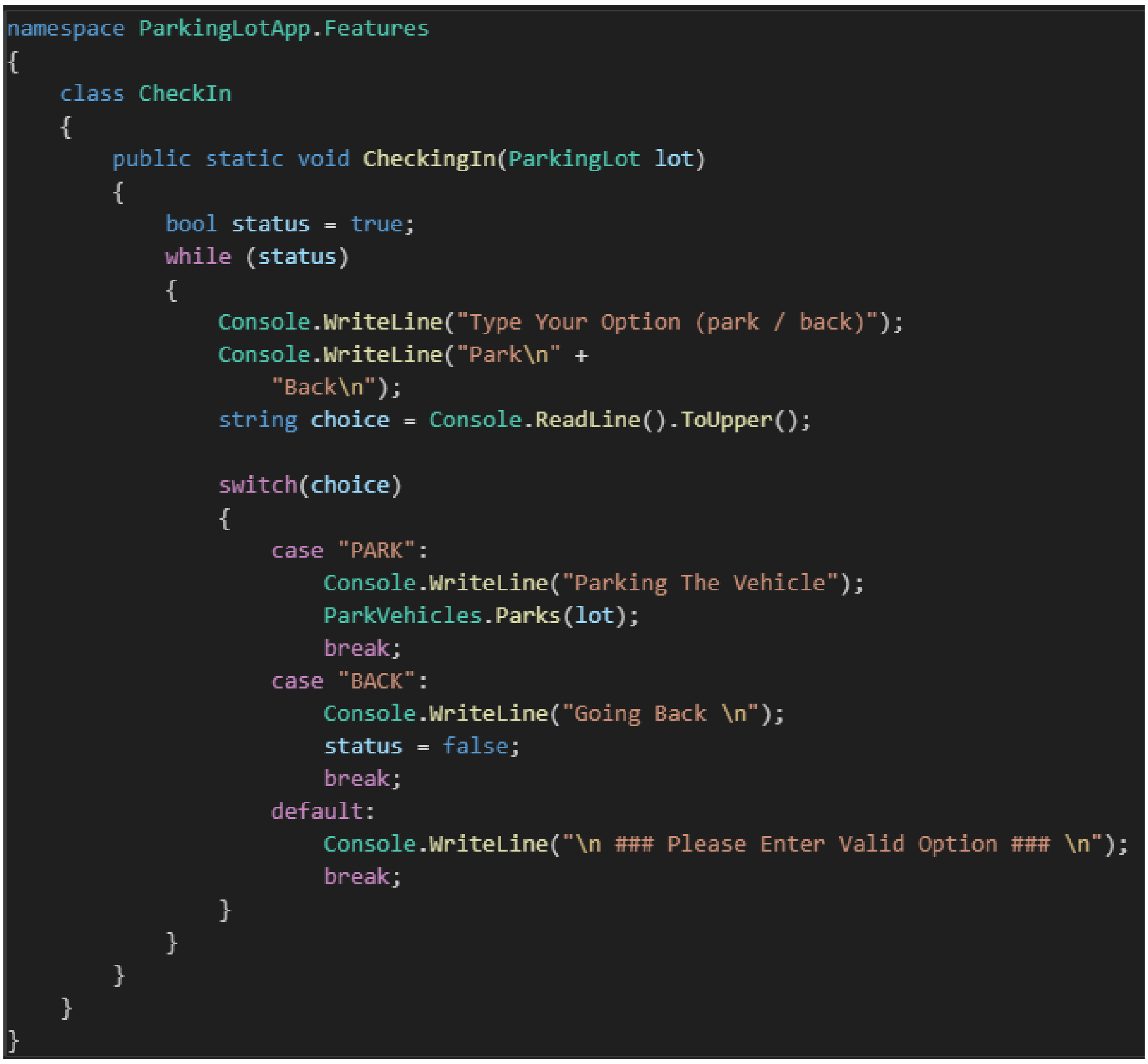
```
namespace ParkingLotApp.Features
{
    class MainMenu
    {
        public static void Menu(ParkingLot lot)
        {
            bool status = true;
            while (status)
            {
                Console.WriteLine("--- Welcome to WaterFall Parking ---");

                int availableSlots = lot.Slot.Count(s => s.Value == null);
                Console.WriteLine($"-- Available Slots : {availableSlots} --\n");

                Console.WriteLine("Please Pick Your Choices (1 - 4)");
                Console.WriteLine("1. Check-In \n" +
                    "2. Check-Out \n" +
                    "3. Reports \n" +
                    "4. Exit \n");
                try
                {
                    string choice = Console.ReadLine();
                    switch (choice)
                    {
                        case "1":
                            {
                                Console.WriteLine("Checking In... \n");
                                CheckIn.CheckingIn(lot);
                                break;
                            }
                        case "2":
                            {
                                Console.WriteLine("Checking Out.. \n");
                                CheckOut.CheckingOut(lot);
                                break;
                            }
                        case "3":
                            {
                                Console.WriteLine("Reports.. \n");
                                ReportMenu.Menu(lot);
                                break;
                            }
                        case "4":
                            {
                                Console.WriteLine("Exit The Program...");
                                status = false;
                                break;
                            }
                        default:
                            Console.WriteLine("\n ### Please Enter Valid Choice ### \n");
                            break;
                    }
                }
                catch (Exception e)
                {
                    Console.WriteLine(e.Message.ToString());
                }
            }
        }
    }
}
```

The MainMenu Class consist a features to list an options of what User wants to do, every choice will use another Class to run the features, by doing so the codes inside MainMenu Class are not too long.

The int availableSlots are code to iterate through ParkingLot's Dictionary, where it will count Keys where the value is null, so it would be represent as Available Parking Slot.

I put Try Catch whenever I need the user's input, because user's inputs are unexpected, I also put global Exception class so I will return any error by using e.Message.ToString().

Checking In Class

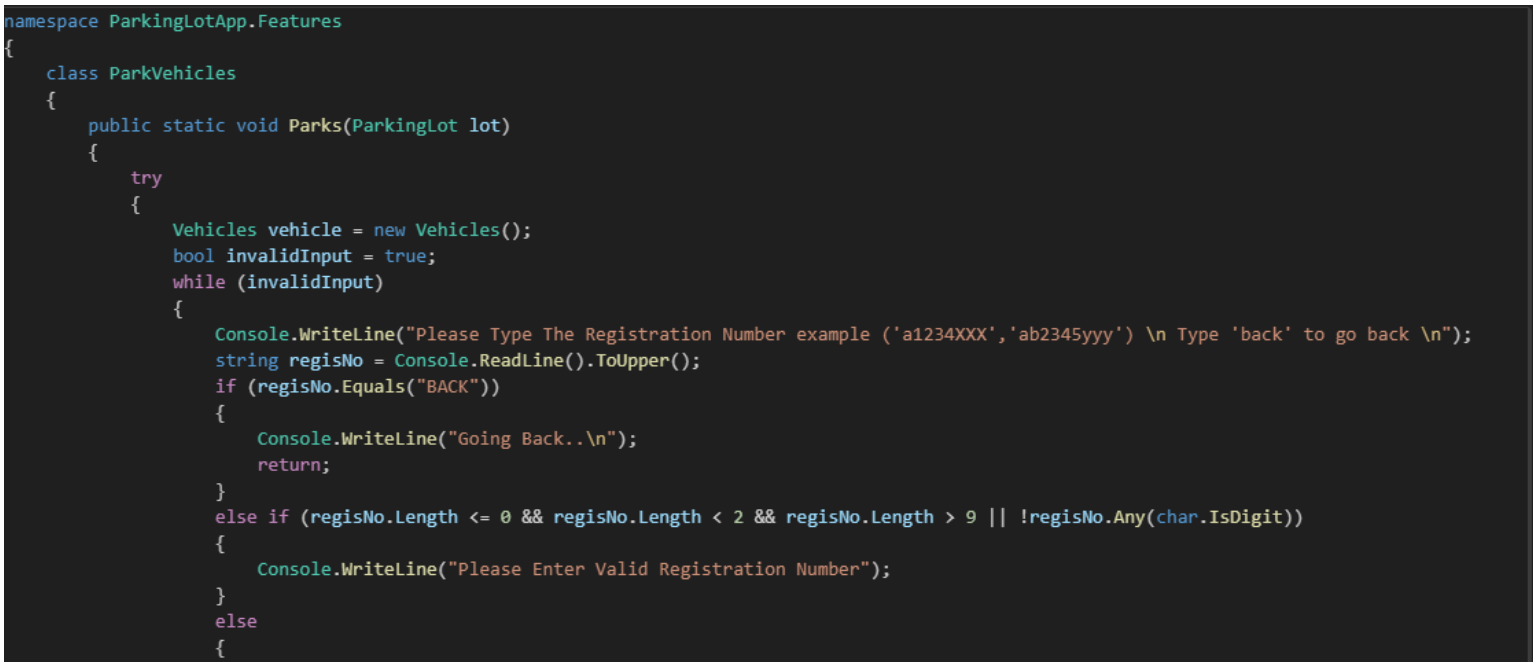


```
namespace ParkingLotApp.Features
{
    class CheckIn
    {
        public static void CheckingIn(ParkingLot lot)
        {
            bool status = true;
            while (status)
            {
                Console.WriteLine("Type Your Option (park / back)");
                Console.WriteLine("Park\n" +
                    "Back\n");
                string choice = Console.ReadLine().ToUpper();

                switch(choice)
                {
                    case "PARK":
                        Console.WriteLine("Parking The Vehicle");
                        ParkVehicles.Parks(lot);
                        break;
                    case "BACK":
                        Console.WriteLine("Going Back \n");
                        status = false;
                        break;
                    default:
                        Console.WriteLine("\n ### Please Enter Valid Option ### \n");
                        break;
                }
            }
        }
    }
}
```

Similar to Main Menu Class, in this CheckingIn Method() I ask for user input but I convert it to UPPERCASE so it would make user's case will always be uppercase letters, I don't use try catch because in this method, the inputs are string / letters type so It should be no matter what kind of input user try to do, it will always return to default: code program

ParkingVehicles Class

Since this code is kinda long, so will make cut the codes and explained it separately.

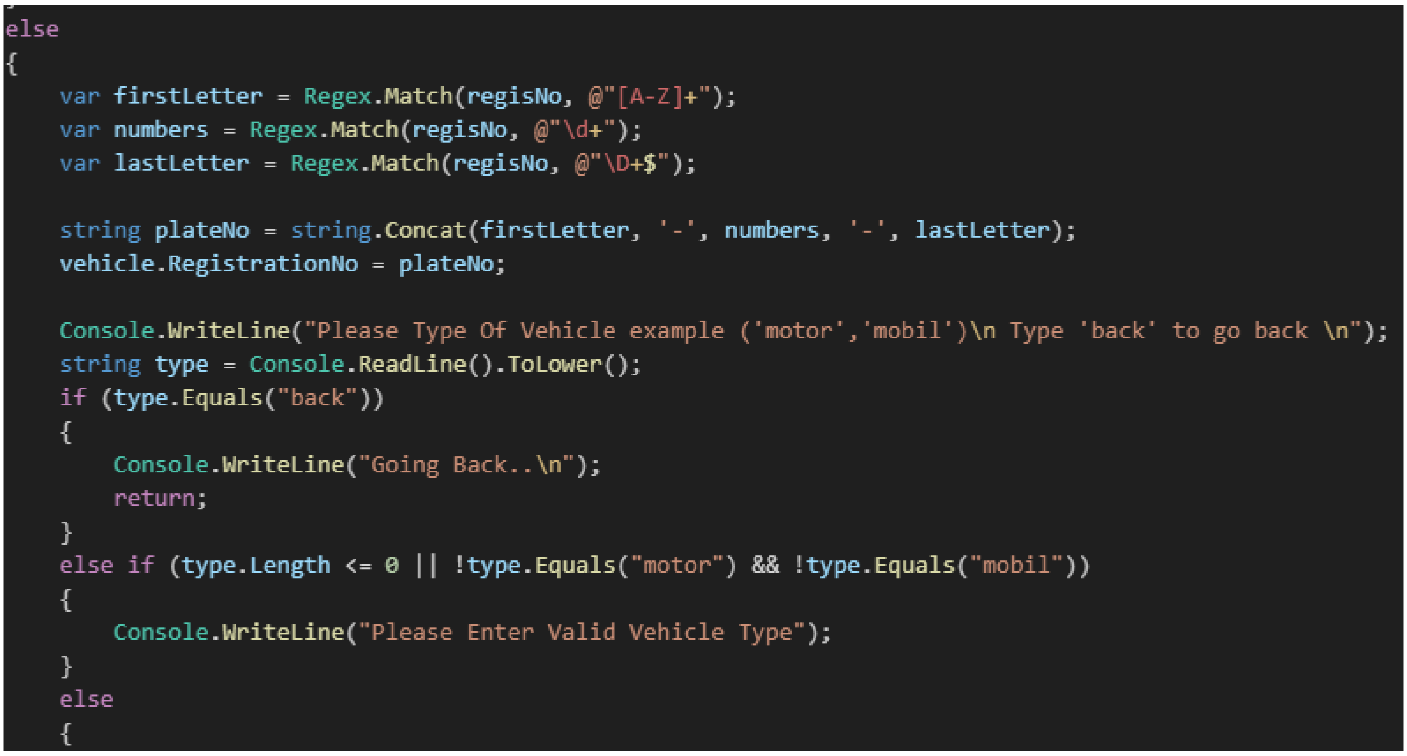


```
namespace ParkingLotApp.Features
{
    class ParkVehicles
    {
        public static void Parks(ParkingLot lot)
        {
            try
            {
                Vehicles vehicle = new Vehicles();
                bool invalidInput = true;
                while (invalidInput)
                {
                    Console.WriteLine("Please Type The Registration Number example ('a1234XXX','ab2345yyy') \n Type 'back' to go back \n");
                    string regisNo = Console.ReadLine().ToUpper();
                    if (regisNo.Equals("BACK"))
                    {
                        Console.WriteLine("Going Back..\n");
                        return;
                    }
                    else if (regisNo.Length <= 0 && regisNo.Length < 2 && regisNo.Length > 9 || !regisNo.Any(char.IsDigit))
                    {
                        Console.WriteLine("Please Enter Valid Registration Number");
                    }
                    else
                    {
```

From the beginning, Parks() methods are actually trying to records a vehicles, where it will ask user to input a plate/registration number, but if user type back or BACK (case insensitive) because I put it ToUpper() for user's input after they inputted it.
if user type back, the program will return and go back to Main Menu, I was hoping I could just make it return to previous menu which is Checking-In but since Checking-in Menu doesn't have many options I just stick by going to main menu.

The bool invalidInput variables used to make the while loops keep going until the inputs are valid according to my program, I will explain later when the while loop is going to end.

The method have several conditions while asking user's input for plate number, where if the length is less than or equals to 0, less than 2 and also if the number is longer than 9 digits, or if the number does not contain any Digits, it will show Message to "Please Enter Valid Registration Number" and will asking the user to reinput.

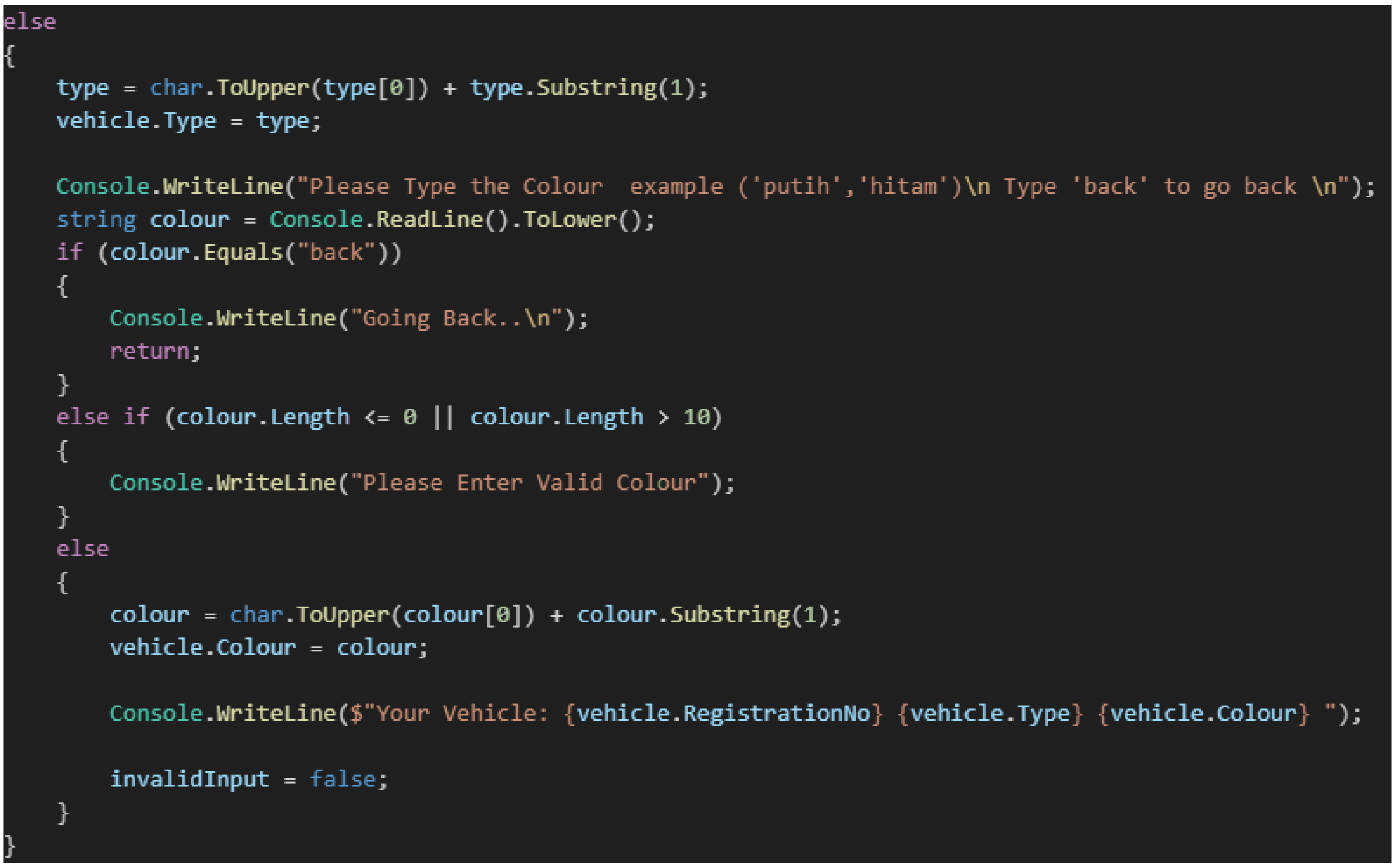


```
else
{
    var firstLetter = Regex.Match(regisNo, @"[A-Z]+");
    var numbers = Regex.Match(regisNo, @"\d+");
    var lastLetter = Regex.Match(regisNo, @"\D+$");

    string plateNo = string.Concat(firstLetter, '-', numbers, '-', lastLetter);
    vehicle.RegistrationNo = plateNo;

    Console.WriteLine("Please Type Of Vehicle example ('motor','mobil')\n Type 'back' to go back \n");
    string type = Console.ReadLine().ToLower();
    if (type.Equals("back"))
    {
        Console.WriteLine("Going Back..\n");
        return;
    }
    else if (type.Length <= 0 || !type.Equals("motor") && !type.Equals("mobil"))
    {
        Console.WriteLine("Please Enter Valid Vehicle Type");
    }
    else
    {
```

Continuing from previous code, after the user successfully enter a plate number, program will manipulate previous inputted variables to insert in for Vehicle object, where:

- the firstLetter is trying to match any letter from the beginning of the string that inpputed by user.
- the numbers are match any digits inputted after firstLetter, and
- lastLetter are matching any letters from the end of the input.

Then it will concat the 3 variables into 1 string named plateNo with additional literal "-" in between thems so it would appear as normal Plate Number just like "B-1234-XYZ". The program will continue asking the user to input type of vehicle, the program will only accept "mobil" or "motor" case insesitive because I convert the user's input to lowercase.



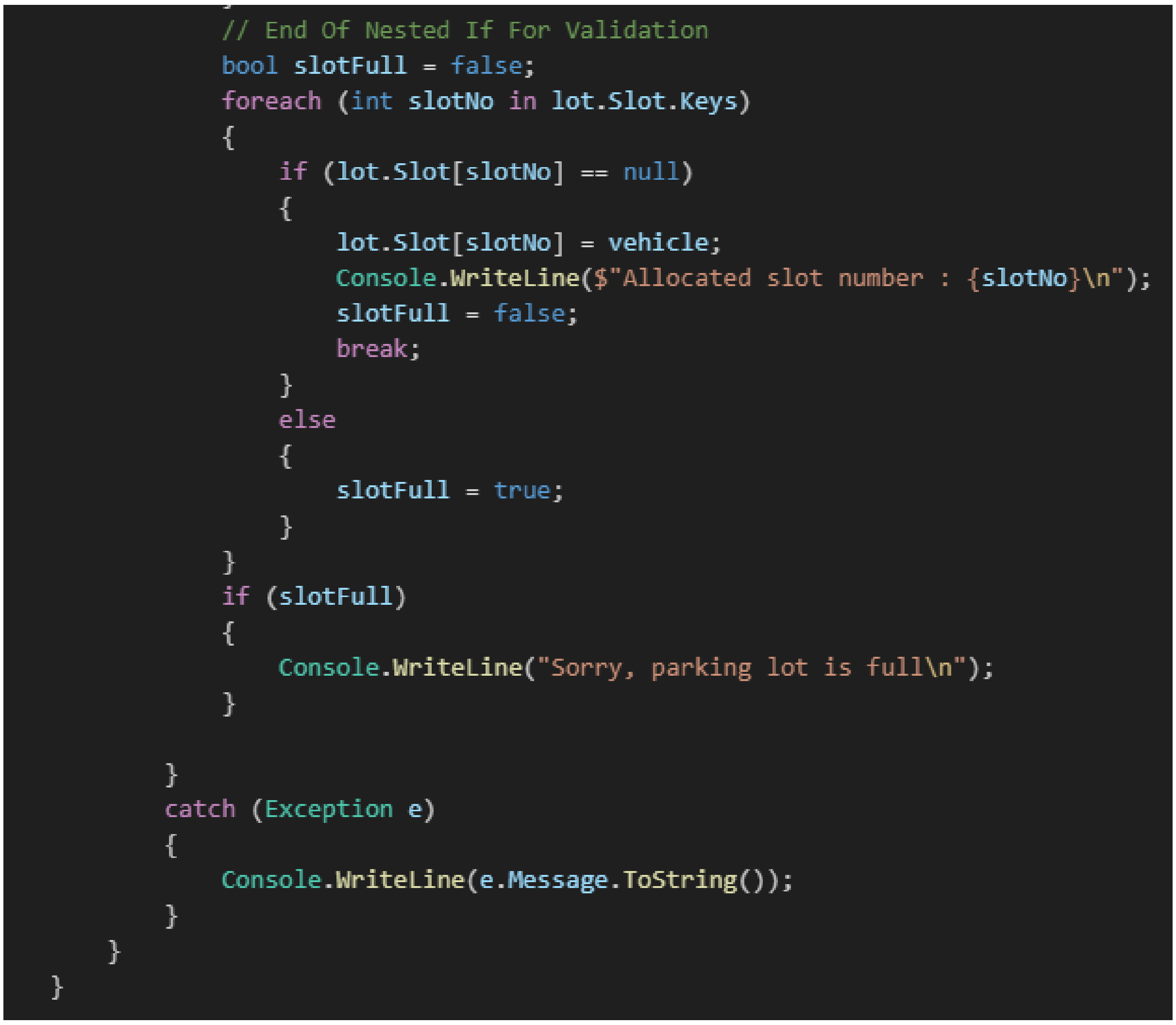
```
else
{
    type = char.ToUpper(type[0]) + type.Substring(1);
    vehicle.Type = type;

    Console.WriteLine("Please Type the Colour  example ('putih','hitam')\n Type 'back' to go back \n");
    string colour = Console.ReadLine().ToLower();
    if (colour.Equals("back"))
    {
        Console.WriteLine("Going Back..\n");
        return;
    }
    else if (colour.Length <= 0 || colour.Length > 10)
    {
        Console.WriteLine("Please Enter Valid Colour");
    }
    else
    {
        colour = char.ToUpper(colour[0]) + colour.Substring(1);
        vehicle.Colour = colour;

        Console.WriteLine($"Your Vehicle: {vehicle.RegistrationNo} {vehicle.Type} {vehicle.Colour} ");

        invalidInput = false;
    }
}
```

Then, to make the Type of Vehicle have 1 char Uppercase , I use char.ToUpper() method in C# to make only the first letter of previous input which is either "motor" or "mobil", before adding it into Vehicle object, I convert it so it will be either "Motor" or "Mobil".

The Program now will ask the User to input the colour of the vehicle,
I don't put any specific validation, I just put if the length is less than or equal to 0 or if the length is longer than 10 letters, so it will return message and will asking the user to re-input it.

After it done, the program will do the same like vehicle Type in previous code, where it will convert first Character of the input to became an Uppercase, and then will added the colour to the vehicle object.

And the program will show the details of the registered vehicles example
" Your Vehicle: B-1234-XYZ Mobil Putih", then set invalidInput to false so it would stop the while loop.

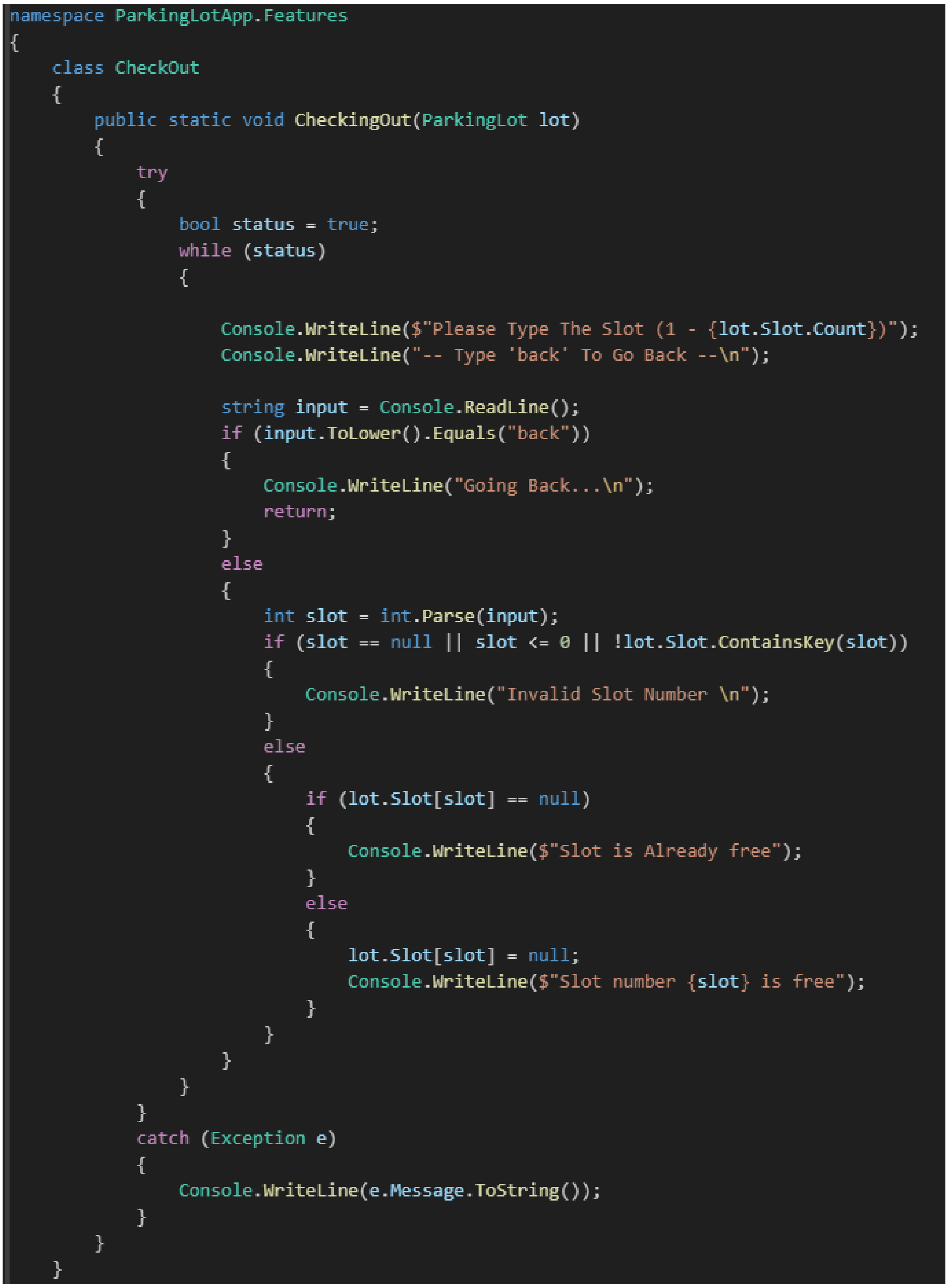


```
            // End Of Nested If For Validation
            bool slotFull = false;
            foreach (int slotNo in lot.Slot.Keys)
            {
                if (lot.Slot[slotNo] == null)
                {
                    lot.Slot[slotNo] = vehicle;
                    Console.WriteLine($"Allocated slot number : {slotNo}\n");
                    slotFull = false;
                    break;
                }
                else
                {
                    slotFull = true;
                }
            }
            if (slotFull)
            {
                Console.WriteLine("Sorry, parking lot is full\n");
            }

        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message.ToString());
        }
    }
}
```

After the registration of vehicle is done, declaring variable boolean slotFull as false for next conditional, then the program now will check the parking slots by using loops, where it will loops from the beginning of the key which should be sequential by 1, 2,3... and so on, so the vehicles will be always put to the first Key or Slots in this parking term, where it has no vehicles in it,

Then, after the the loops is done, and the vehicle didn't find any empty slots, it will set the status as slotFull = true, then the program will reach the conditional if the slotFull == true, return message "Sorry, Parking Lot Is Full" then the program for parking will stop and going back to previous menu.

After Checking-In, we need to leave / Check-Out eventually, so I will show the Check-Out Class codes.



```
namespace ParkingLotApp.Features
{
    class CheckOut
    {
        public static void CheckingOut(ParkingLot lot)
        {
            try
            {
                bool status = true;
                while (status)
                {

                    Console.WriteLine($"Please Type The Slot (1 - {lot.Slot.Count})");
                    Console.WriteLine("-- Type 'back' To Go Back --\n");

                    string input = Console.ReadLine();
                    if (input.ToLower().Equals("back"))
                    {
                        Console.WriteLine("Going Back...\n");
                        return;
                    }
                    else
                    {
                        int slot = int.Parse(input);
                        if (slot == null || slot <= 0 || !lot.Slot.ContainsKey(slot))
                        {
                            Console.WriteLine("Invalid Slot Number \n");
                        }
                        else
                        {
                            if (lot.Slot[slot] == null)
                            {
                                Console.WriteLine($"Slot is Already free");
                            }
                            else
                            {
                                lot.Slot[slot] = null;
                                Console.WriteLine($"Slot number {slot} is free");
                            }
                        }
                    }
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message.ToString());
            }
        }
    }
```

In the method of CheckingOut(), the program will try to loops and make the user to input what number of the slots that they want to leave / free.
the variable lot.Slot.Count is just my additional variable code so the Menu Would be dynamically based on how many Slots are parking lot have.

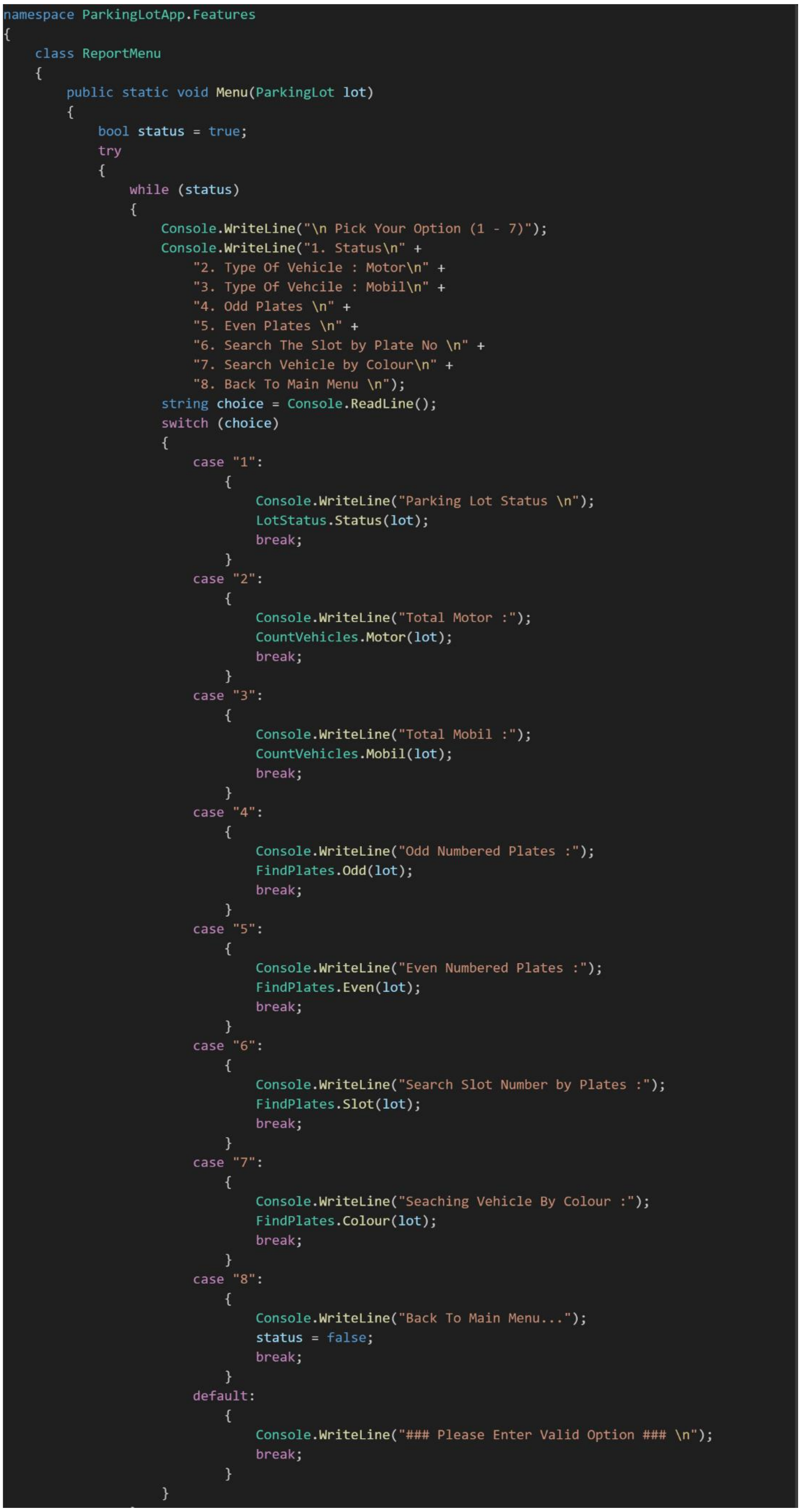
I put conditional if the input isn't valid like null or less than 0, or if the input isn't valid like if there are only 6 slots, but the user try to input 7, it will show error message "Invalid slot Number".

Then if user input correctly, the program will empty the specific Key / Slots based on Slots number they input,

the message will be displayed differently based on conditions where if the slots is already empty but the user try to empty it anyway it will show "Slot Is Already Free" otherwise it will make the specific slot as empty and show message "Slot Number {slot} is free" with {slot} is the user's input.

Records Class

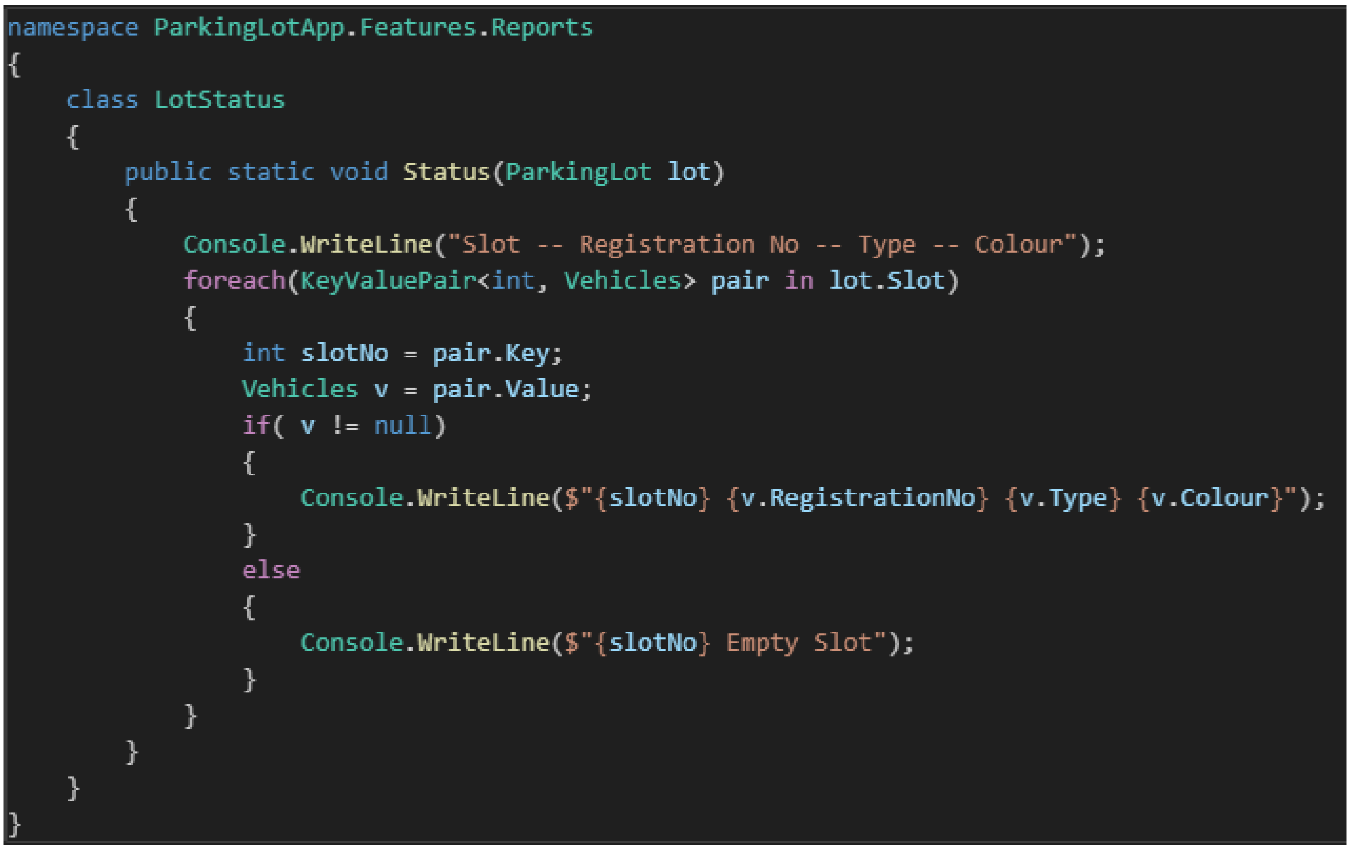
This class has a long code, since it consists of many methods and other classes that related to it.



```csharp
namespace ParkingLotApp.Features
{
    class ReportMenu
    {
        public static void Menu(ParkingLot lot)
        {
            bool status = true;
            try
            {
                while (status)
                {
                    Console.WriteLine("\n Pick Your Option (1 - 7)");
                    Console.WriteLine("1. Status\n" +
                        "2. Type Of Vehicle : Motor\n" +
                        "3. Type Of Vehcile : Mobil\n" +
                        "4. Odd Plates \n" +
                        "5. Even Plates \n" +
                        "6. Search The Slot by Plate No \n" +
                        "7. Search Vehicle by Colour\n" +
                        "8. Back To Main Menu \n");
                    string choice = Console.ReadLine();
                    switch (choice)
                    {
                        case "1":
                            {
                                Console.WriteLine("Parking Lot Status \n");
                                LotStatus.Status(lot);
                                break;
                            }
                        case "2":
                            {
                                Console.WriteLine("Total Motor :");
                                CountVehicles.Motor(lot);
                                break;
                            }
                        case "3":
                            {
                                Console.WriteLine("Total Mobil :");
                                CountVehicles.Mobil(lot);
                                break;
                            }
                        case "4":
                            {
                                Console.WriteLine("Odd Numbered Plates :");
                                FindPlates.Odd(lot);
                                break;
                            }
                        case "5":
                            {
                                Console.WriteLine("Even Numbered Plates :");
                                FindPlates.Even(lot);
                                break;
                            }
                        case "6":
                            {
                                Console.WriteLine("Search Slot Number by Plates :");
                                FindPlates.Slot(lot);
                                break;
                            }
                        case "7":
                            {
                                Console.WriteLine("Seaching Vehicle By Colour :");
                                FindPlates.Colour(lot);
                                break;
                            }
                        case "8":
                            {
                                Console.WriteLine("Back To Main Menu...");
                                status = false;
                                break;
                            }
                        default:
                            {
                                Console.WriteLine("### Please Enter Valid Option ### \n");
                                break;
                            }
                    }
```

The Record Class have the methods to display menu similar to Checking-In and Main Menu, but this class have more options like displaying Current Parking Lot Status, Type of Vehicles that are currently being parked, Odd & Even Numbered Plates, Search the Slot Position of Parked Vehicle by its Plate, show a list of vehicles by its Colour. And lastly the option to back to main menu.

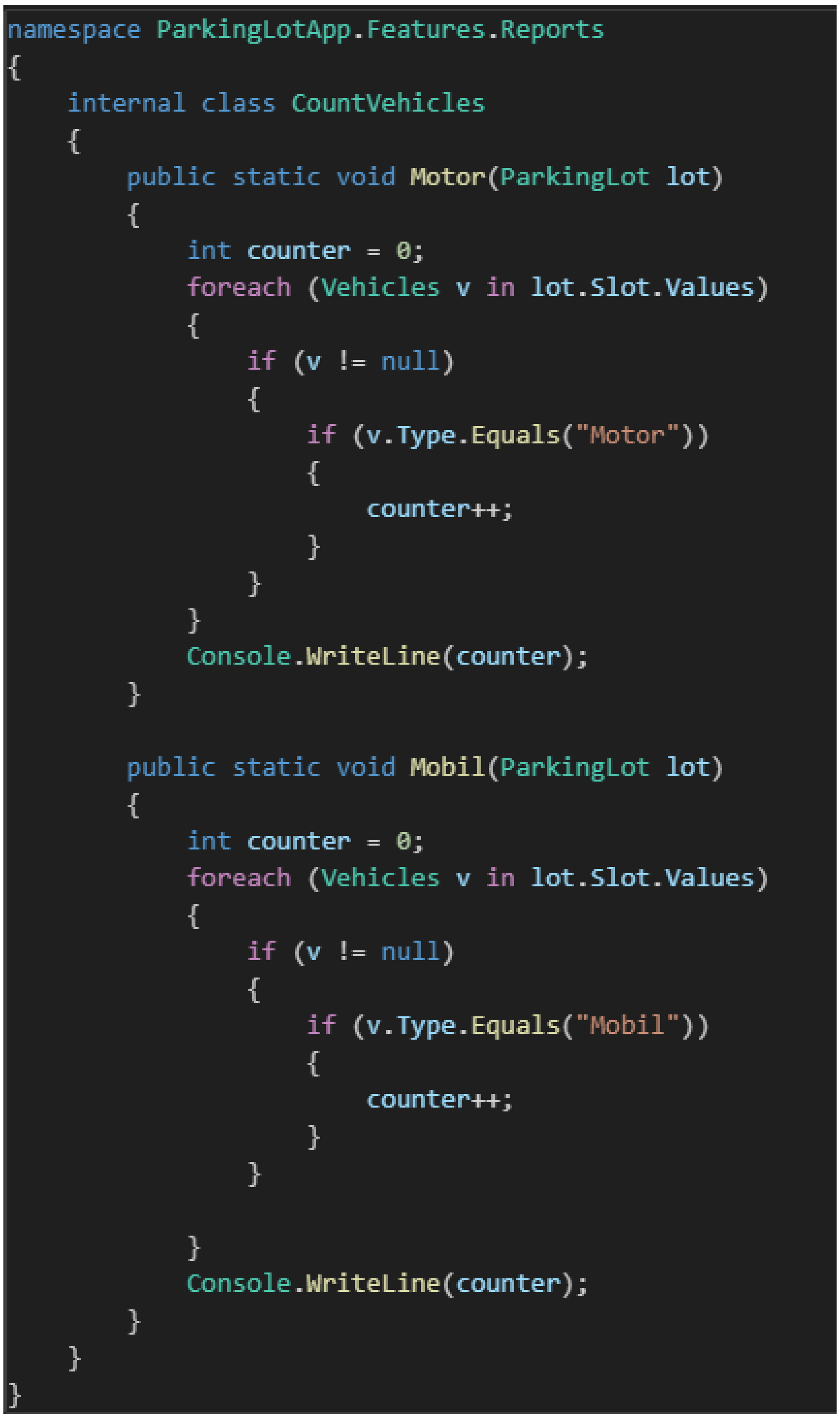
LotStatus Class



```
namespace ParkingLotApp.Features.Reports
{
    class LotStatus
    {
        public static void Status(ParkingLot lot)
        {
            Console.WriteLine("Slot -- Registration No -- Type -- Colour");
            foreach(KeyValuePair<int, Vehicles> pair in lot.Slot)
            {
                int slotNo = pair.Key;
                Vehicles v = pair.Value;
                if( v != null)
                {
                    Console.WriteLine($"{slotNo} {v.RegistrationNo} {v.Type} {v.Colour}");
                }
                else
                {
                    Console.WriteLine($"{slotNo} Empty Slot");
                }
            }
        }
    }
}
```

The method Status() in this class is to display every slots of the parking lot and show the vehicles details such as its RegistrationNo, Type, and Colour,
and if the Slot doesn't have any vehicles in it, it will show as example"No. 5 Empty Slot".

CountVehicles class



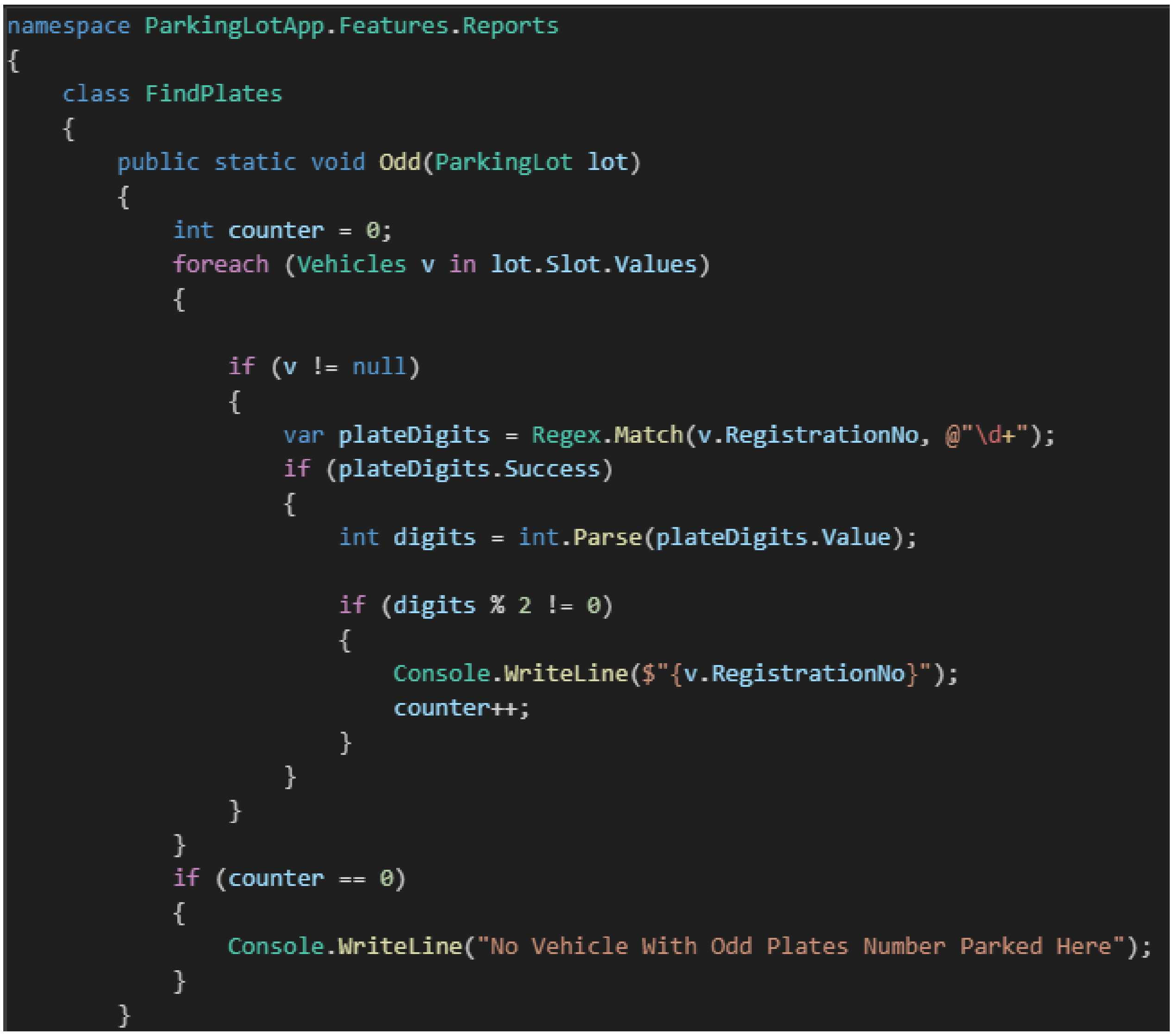
```
namespace ParkingLotApp.Features.Reports
{
    internal class CountVehicles
    {
        public static void Motor(ParkingLot lot)
        {
            int counter = 0;
            foreach (Vehicles v in lot.Slot.Values)
            {
                if (v != null)
                {
                    if (v.Type.Equals("Motor"))
                    {
                        counter++;
                    }
                }
            }
            Console.WriteLine(counter);
        }

        public static void Mobil(ParkingLot lot)
        {
            int counter = 0;
            foreach (Vehicles v in lot.Slot.Values)
            {
                if (v != null)
                {
                    if (v.Type.Equals("Mobil"))
                    {
                        counter++;
                    }
                }

            }
            Console.WriteLine(counter);
        }
    }
}
```

This class have 2 methods, Motor() and Mobil() which are both of them work similarly by searching for the in ParkingLot Slots where the type of vehicles are either Motor or Mobil, it will return a counter which will increase by 1 everytime the program found Motor or Mobil in their own method.

FindPlate Class

I Choose to make this class have many methods since it logically corrects to me if I put FindPlates.Colour() it would mean Find Plate Number by Colours, etc.

Find Odd Number Plates Method

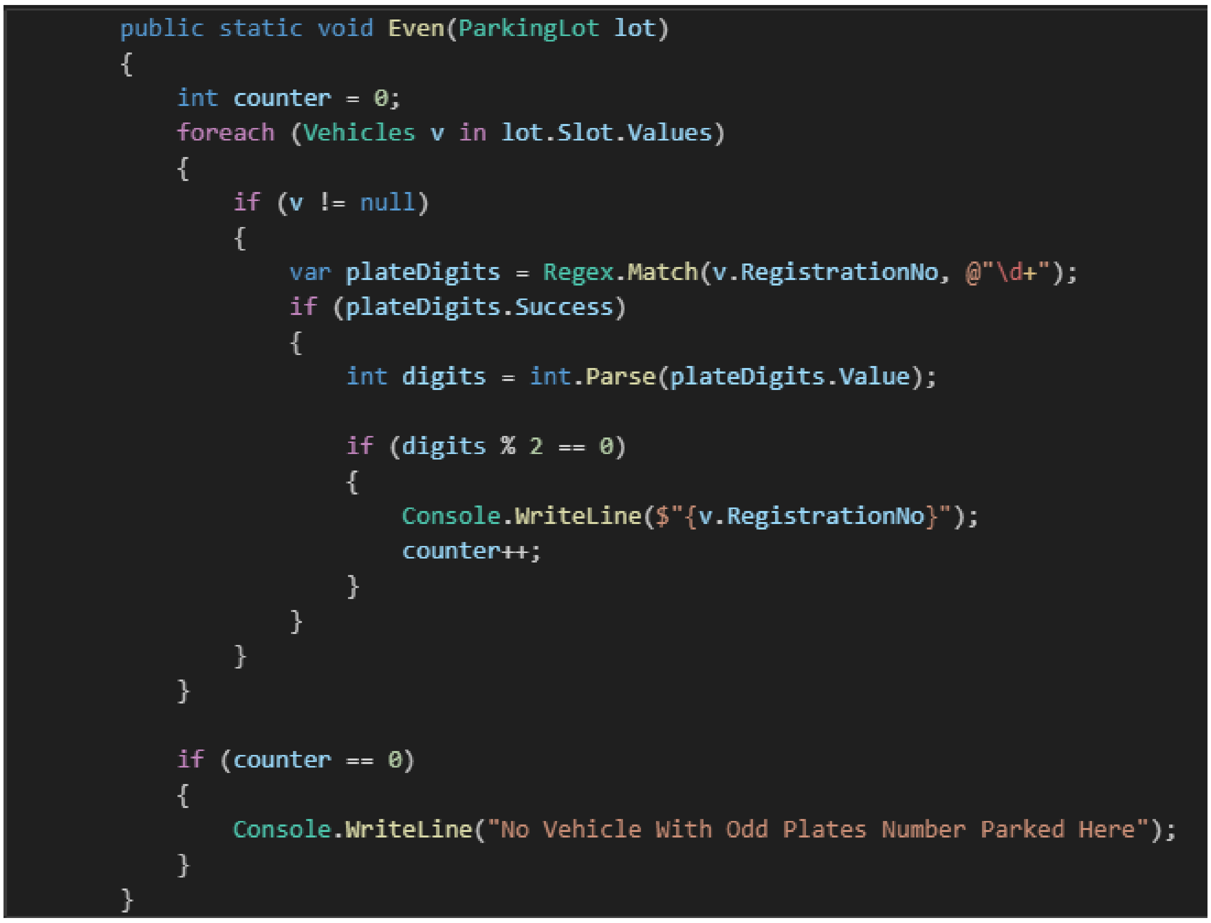


```csharp
namespace ParkingLotApp.Features.Reports
{
    class FindPlates
    {
        public static void Odd(ParkingLot lot)
        {
            int counter = 0;
            foreach (Vehicles v in lot.Slot.Values)
            {

                if (v != null)
                {
                    var plateDigits = Regex.Match(v.RegistrationNo, @"\d+");
                    if (plateDigits.Success)
                    {
                        int digits = int.Parse(plateDigits.Value);

                        if (digits % 2 != 0)
                        {
                            Console.WriteLine($"{v.RegistrationNo}");
                            counter++;
                        }
                    }
                }
            }
            if (counter == 0)
            {
                Console.WriteLine("No Vehicle With Odd Plates Number Parked Here");
            }
        }
```

For the Odd() methods it would mean to Count the vehicles with Odd Numbered Plates, where I declared integer counter to count everytime the program find oddNumbered plate number, the regex Match is a method to match vehicle.RegistrationNo where I only want to take the digits in it, since in the parking process I already specify what the vehicles RegistrationNo should be like, I don't have to make any statement to specifically format the RegistrationNo.

If the program didn't found any Odd Numbered Plate, it will Show Message instead of just literall "0".

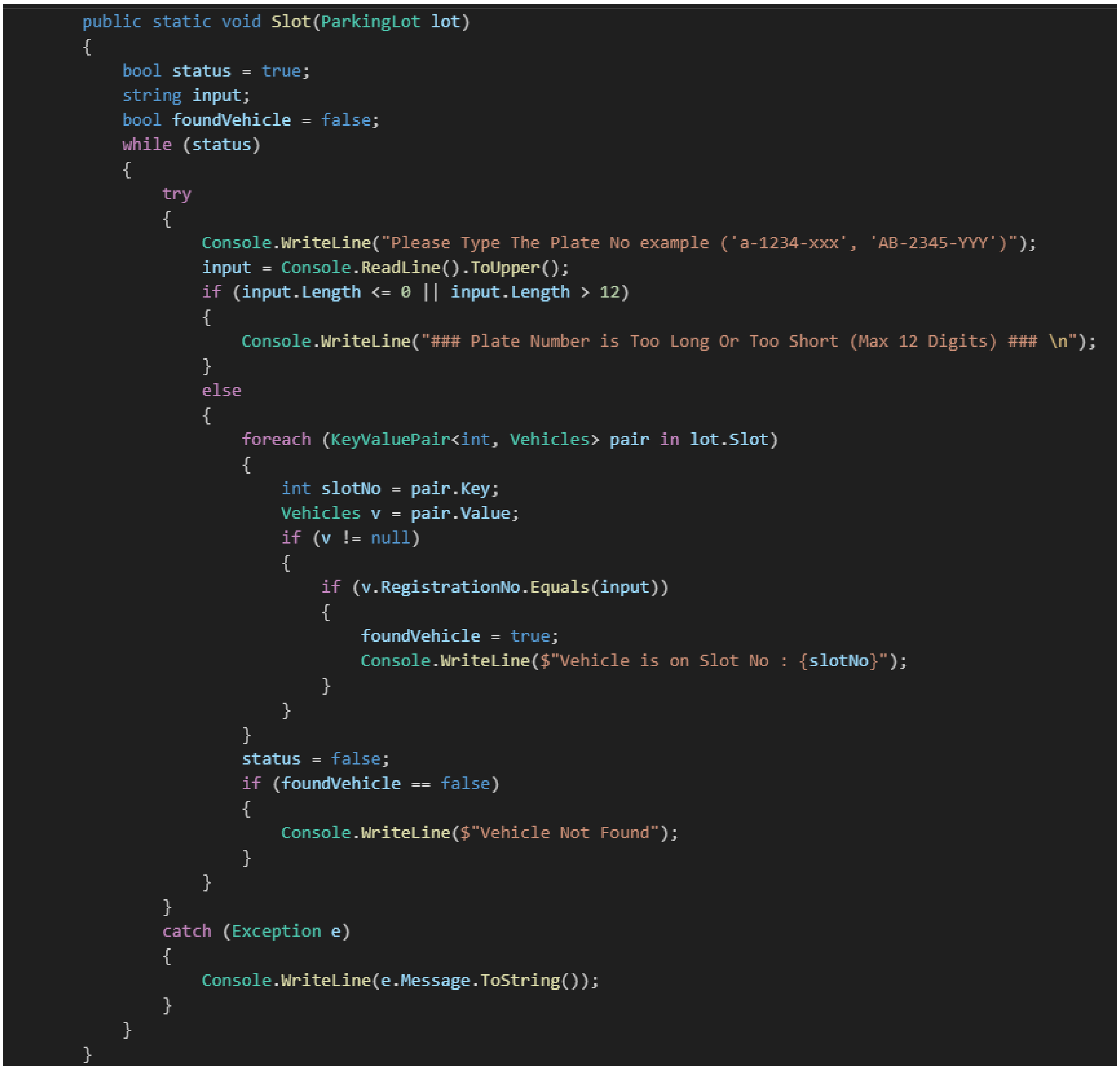
Find Even Number Plates Method



```
public static void Even(ParkingLot lot)
{
    int counter = 0;
    foreach (Vehicles v in lot.Slot.Values)
    {
        if (v != null)
        {
            var plateDigits = Regex.Match(v.RegistrationNo, @"\d+");
            if (plateDigits.Success)
            {
                int digits = int.Parse(plateDigits.Value);

                if (digits % 2 == 0)
                {
                    Console.WriteLine($"{v.RegistrationNo}");
                    counter++;
                }
            }
        }
    }

    if (counter == 0)
    {
        Console.WriteLine("No Vehicle With Odd Plates Number Parked Here");
    }
}
```

The Even Method works the same way as the Odd Method, it only has different in the modulo operations.

Find The Slots of Parked Vehicle by Its Plate Number Method

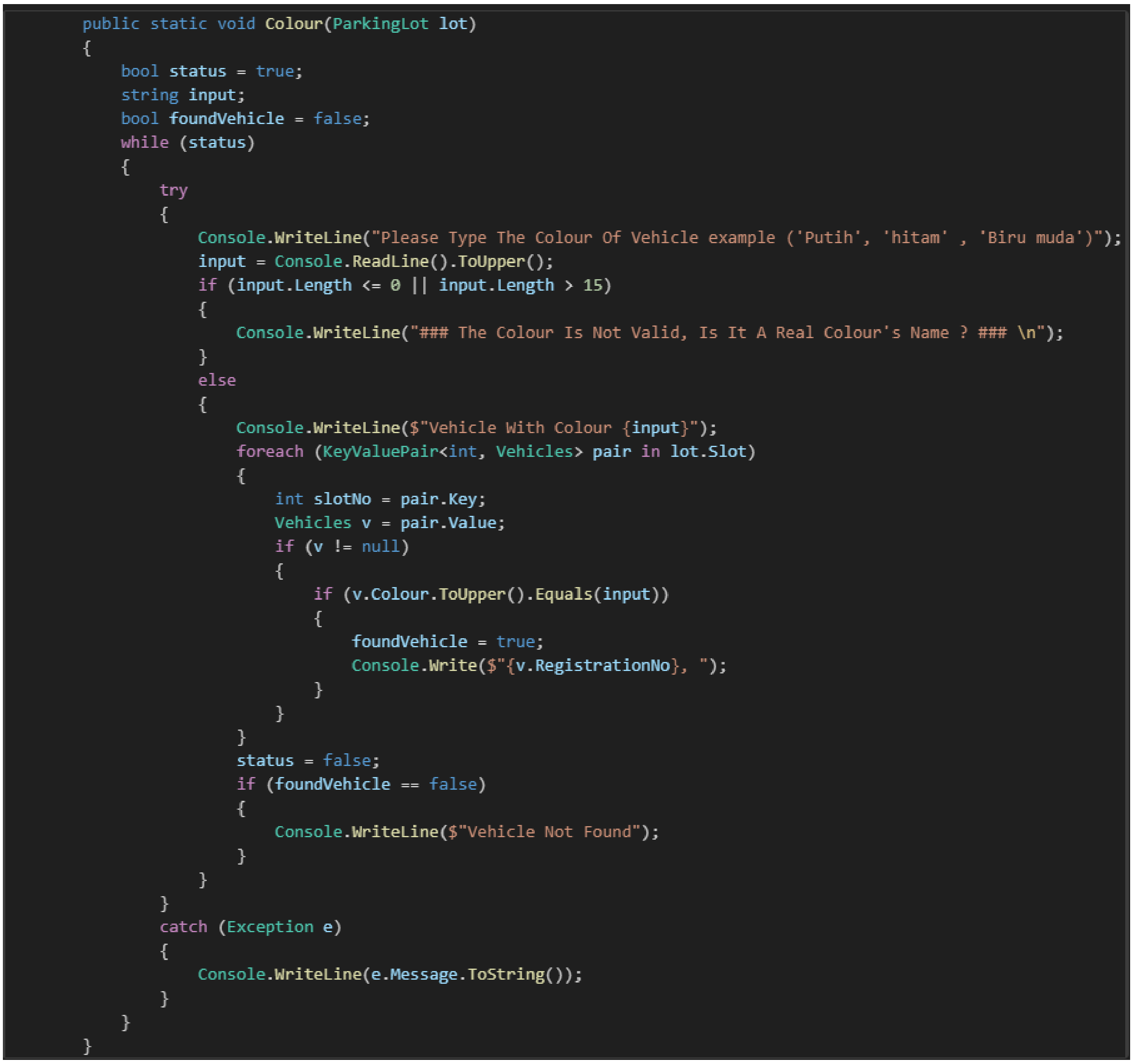


```
public static void Slot(ParkingLot lot)
{
    bool status = true;
    string input;
    bool foundVehicle = false;
    while (status)
    {
        try
        {
            Console.WriteLine("Please Type The Plate No example ('a-1234-xxx', 'AB-2345-YYY')");
            input = Console.ReadLine().ToUpper();
            if (input.Length <= 0 || input.Length > 12)
            {
                Console.WriteLine("### Plate Number is Too Long Or Too Short (Max 12 Digits) ### \n");
            }
            else
            {
                foreach (KeyValuePair<int, Vehicles> pair in lot.Slot)
                {
                    int slotNo = pair.Key;
                    Vehicles v = pair.Value;
                    if (v != null)
                    {
                        if (v.RegistrationNo.Equals(input))
                        {
                            foundVehicle = true;
                            Console.WriteLine($"Vehicle is on Slot No : {slotNo}");
                        }
                    }
                }
                status = false;
                if (foundVehicle == false)
                {
                    Console.WriteLine($"Vehicle Not Found");
                }
            }
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message.ToString());
        }
    }
}
```

Since I want the user to specifically type the exact plate number to find the slots of the parked vehicle slots, I restrict the input to must have literal “-“ although I can just use similar concat like parking methods, I just found this way would be simple and work the same way

The code will try to iterate from the Dictionary of ParkingLots, and accessing it value, where if the value is match from the user’s input, the program will return the SlotNumber of current Vehicle, otherwise it will return message “Vehicle Not Found’.

Find Plate Numbers With Specific Colour in The Parking Lots



```
public static void Colour(ParkingLot lot)
{
    bool status = true;
    string input;
    bool foundVehicle = false;
    while (status)
    {
        try
        {
            Console.WriteLine("Please Type The Colour Of Vehicle example ('Putih', 'hitam' , 'Biru muda')");
            input = Console.ReadLine().ToUpper();
            if (input.Length <= 0 || input.Length > 15)
            {
                Console.WriteLine("### The Colour Is Not Valid, Is It A Real Colour's Name ? ### \n");
            }
            else
            {
                Console.WriteLine($"Vehicle With Colour {input}");
                foreach (KeyValuePair<int, Vehicles> pair in lot.Slot)
                {
                    int slotNo = pair.Key;
                    Vehicles v = pair.Value;
                    if (v != null)
                    {
                        if (v.Colour.ToUpper().Equals(input))
                        {
                            foundVehicle = true;
                            Console.Write($"{v.RegistrationNo}, ");
                        }
                    }
                }
                status = false;
                if (foundVehicle == false)
                {
                    Console.WriteLine($"Vehicle Not Found");
                }
            }
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message.ToString());
        }
    }
}
```

This method will ask the user to input any string that refer to existing Colour even though I don't restrict the input must be a real Number, I just put the conditional if the input is null or longer than 15 letters, it will show message, and then the program will ask the user to input again.

If the user are correct, the program will iterate once again throught the Dictionary, and searching inside the vehicles.Colour property where the value of the vehicle.Colour are match with the user's input, if the results are more than 1, it will show beside of it separated by a comma ','
Otherwise the program will show message "Vehicle Not Found".