|  |
| --- |
|  |
| Individual Assignment: Multi-story Car-Parking Program (MCP) CS12320/CC12320 |
| aar17@aber.ac.uk |

|  |
| --- |
| Aaron Ramsey  5-10-2019 |

Contents

[Introduction 2](#_Toc8386313)

[UML Use Case Diagram 3](#_Toc8386314)

[Design 3](#_Toc8386315)

[Textual description of classes and their relationships 3](#_Toc8386316)

[Main UML Class Diagram 4](#_Toc8386317)

[Functional Requirements 4](#_Toc8386318)

[Pseudo-code of the most complex algorithm 5](#_Toc8386319)

[Testing 6](#_Toc8386320)

[Test Table 6](#_Toc8386321)

[Screenshots 11](#_Toc8386322)

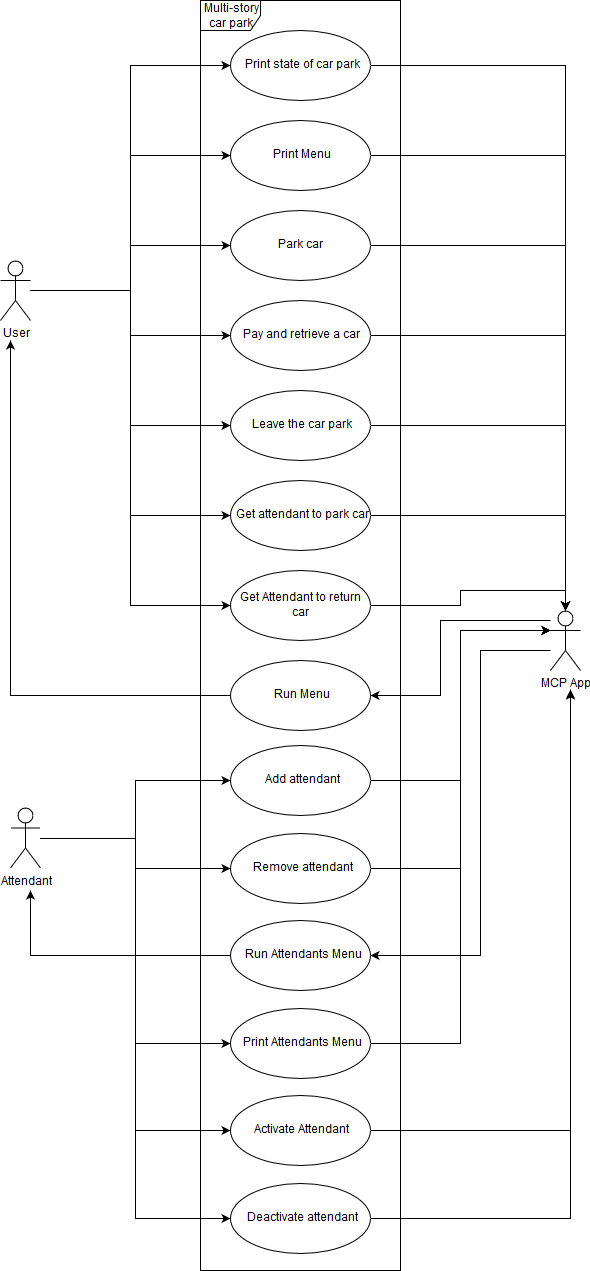
[Evaluation 16](#_Toc8386323)

# Introduction

In this project I had to create a system for a multi-story car park where there were multiple zones with different sizes and costs for each zone. Drivers can arrive and either park the car themselves, where they are allocated a space in a zone, or they can get an attendant to park it for them, where the attendant can roam the car park to pick a free space. Attendants can be added and removed from the system as well as being marked as free or not. When a car is parked the customer is given a receipt with a code on it, they then enter this code on their return which will them bill them the appropriate amount and calculate the appropriate cost adjustment for disabled drivers, if the driver says they are disabled when prompted by the system. It also needs to print the current state of the car park, to show the spaces in the zones. Finally, it must load in the costs for each zone from a text file so it can be easily changed by the final owner of the system.

I finished all the parts of the system that were required in the document specification we were given and also did a section to save the current state of the car park to a text file although didn’t get change to do the load section; but this wasn’t included in the document specification for us to do.

## UML Use Case Diagram

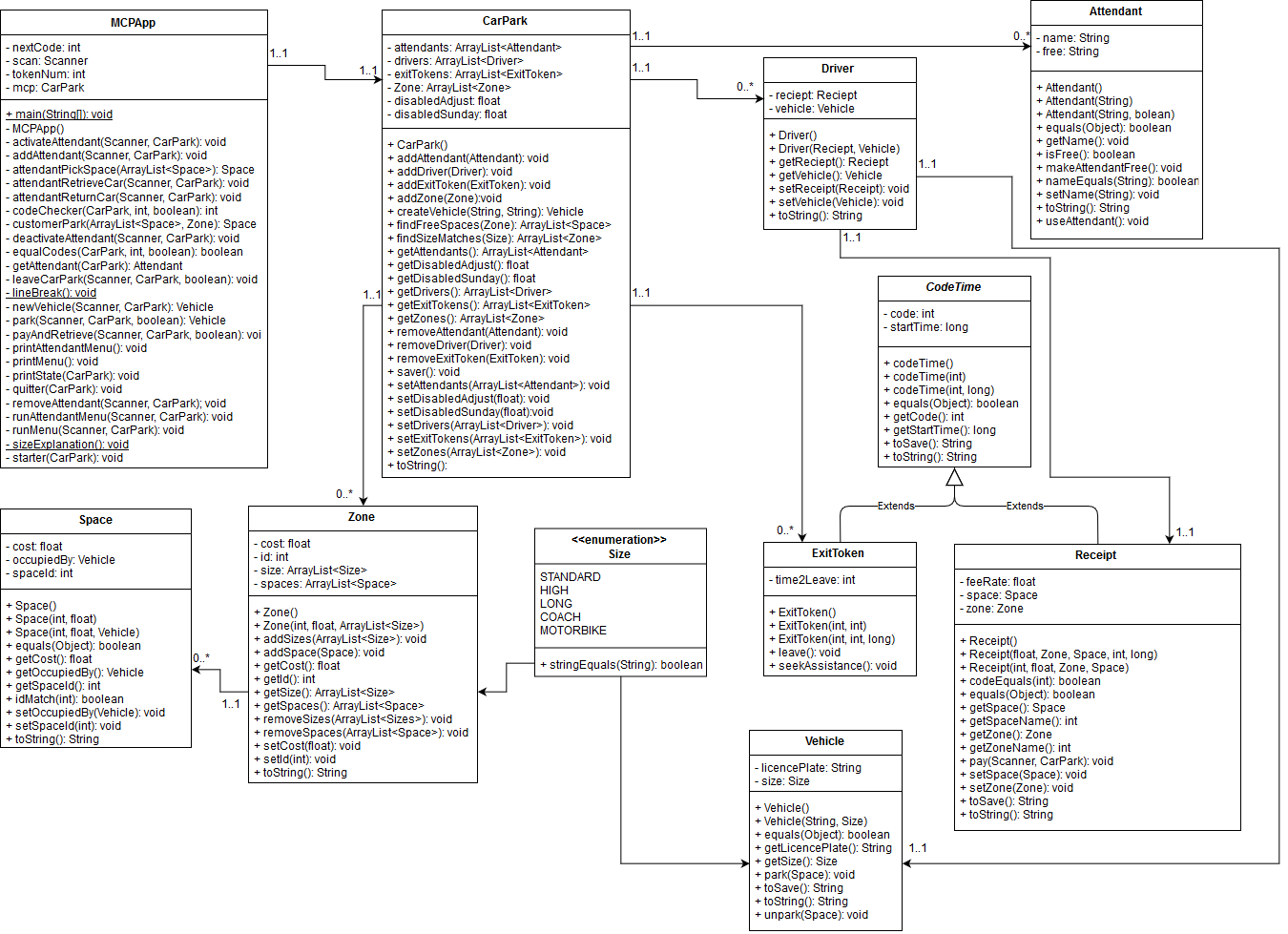


# Design

## Textual description of classes and their relationships

The MCPApp class is the main class where all the code is run from, interacting with the user and all menus, and it links to the CarPark Class. The CarPark class is a class that links to attendants, drivers, exitTokens and Zones; and is used to find all the free spaces and zones that have the same size as the user’s vehicle. The attendants have the name and an attribute to show if they are free so they can then check to see if they are being used by someone else. Drivers are used to link to receipts and vehicles together as one receipt relates directly to one vehicle and can store all that data. exitTokens are stored to know whether to know if a vehicle can be let out by the barrier. The car park stores zones which have spaces attached to them as well as and are used for soring the cost of the zone as well. I have also used an enumerated type to store the size and then link to this in the vehicle and the zone so I can store what sizes are accepted in each of them. Finally, I used inheritance by using the super class code time and the receipt and the exit token inherit the code ad the start time from that abstract super class.

## Main UML Class Diagram



## Functional Requirements

|  |  |
| --- | --- |
| ID | Requirement |
| 1 | Park a car |
| 1.1 | Create a new car |
| 1.1.1 | Add a licence plate |
| 1.1.2 | Give it a size |
| 1.2 | Find a zone and a space to park in |
| 1.2.1 | Find a zone that the car can park in for the size of the car |
| 1.2.2 | Pick a space in that zone |
| 1.3 | Generate a receipt and unique receipt code for the user |
| 2 | Pay parking fee |
| 2.1 | Find receipt |
| 2.2 | Get payment |
| 2.2.1 | Calculate cost for a regular user |
| 2.2.2 | Adjust it for a disabled driver |
| 2.2.3 | Adjust for a disabled driver parking on Sunday |
| 2.3 | Give an exit token |
| 2.3.1 | Generate unique token for the user |
| 3 | Leave car park within 15mins once token has been issued |
| 3.1 | Take a token number of a user and find the matching token |
| 3.2 | Calculate time stayed |
| 3.3 | If time stayed is more than 15mins, tell the user to seek assistance |
| 4 | Get attendant to park car |
| 4.1 | Pass FR1 |
| 4.2 | Allows attendant to pick space |
| 4.3 | Check if there are any attendants |
| 5 | Get attendant to retrieve car |
| 5.1 | Pass FR2 & FR3 |
| 5.2 | Check if there are any attendants |
| 5.3 | Select/use attendant |
| 6 | Add attendant |
| 6.1 | Get name of attendant |
| 6.2 | Create new attendant |
| 7 | Remove Attendant |
| 7.1 | Get attendant name |
| 7.2 | Remove attendant from system |
| 8 | Activate Attendant |
| 8.1 | Get attendant name |
| 8.2 | Make attendant active |
| 9 | Deactivate attendant |
| 9.1 | Get name of attendant |
| 9.2 | Make attendant busy |
| 10 | Print state of car park |

## Pseudo-code of the most complex algorithm

Pay method

timeParkedMillis = currentTime – timeOriginallyParked

timeParkedHours = makeIntoHours(timeParkedMillis)

amountToPay = timeParkedHours \* feeRateOfZone

amountToPay = roundUp(amountToPay)

inp = “Are you disabled?”

If (inp == “yes” or inp == true):

If (today.Day == Sunday):

amountToPay = 0

else:

amountToPay = amountToPay\*0.5

do:

inp = “Please pay “ + amountToPay

if (inp>0):

amountToPay = amountToPay – inp

else:

print(“Please pay a positive amount of money”)

while (amountToPay > 0)

if (amountToPay < 0):

print(“Change given is “ + amountToPay \*-1)

I believe this is the most complex algorithm as it has many parts, dealing with times, getting costs from the object in which its stored, Calculating the money left to pay as well as the change to give. It also must validate that you aren’t paying in negative money and check the day of the week to see if it’s a Sunday for disabled drivers to get free parking.

# Testing

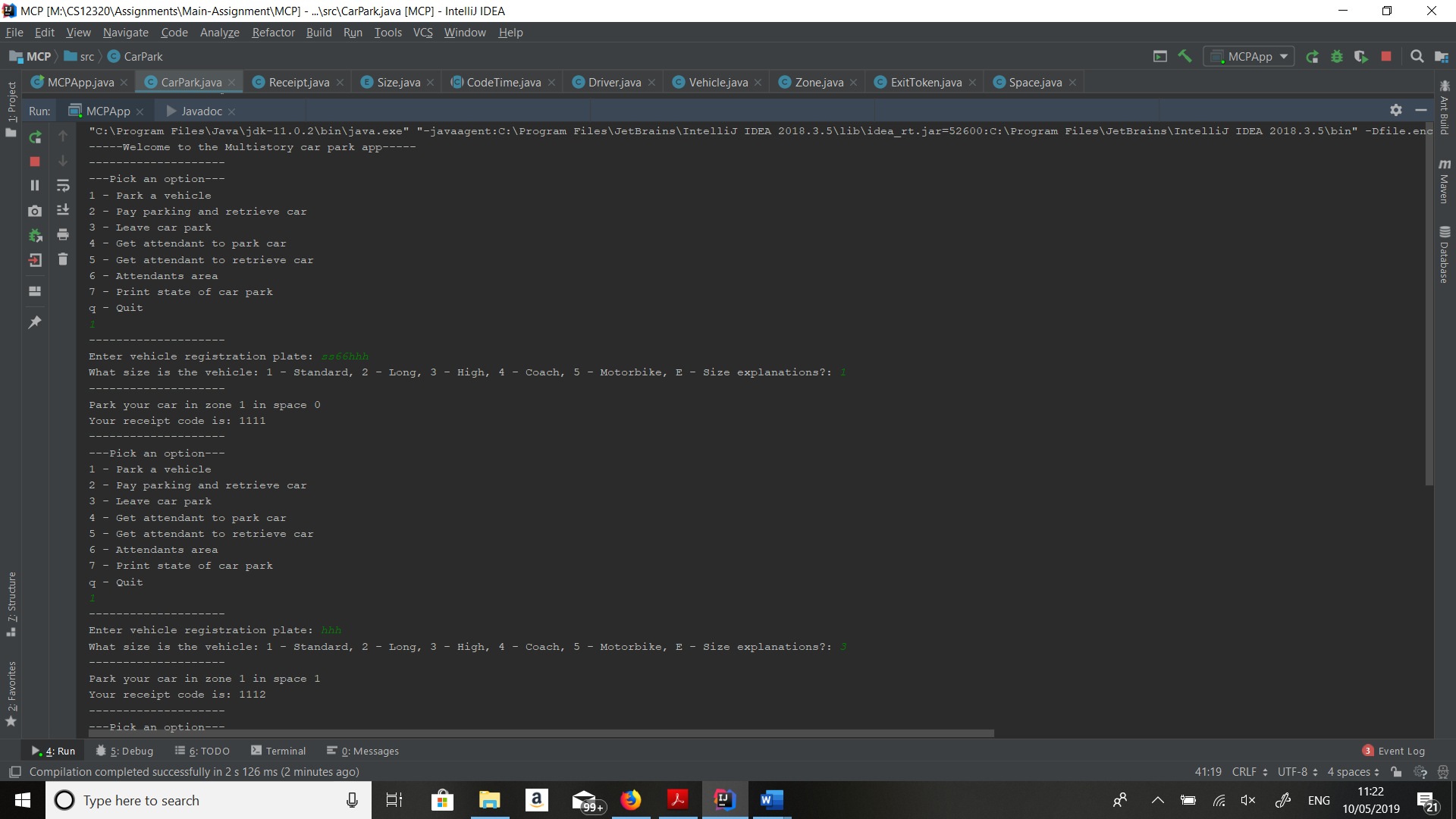
## Test Table

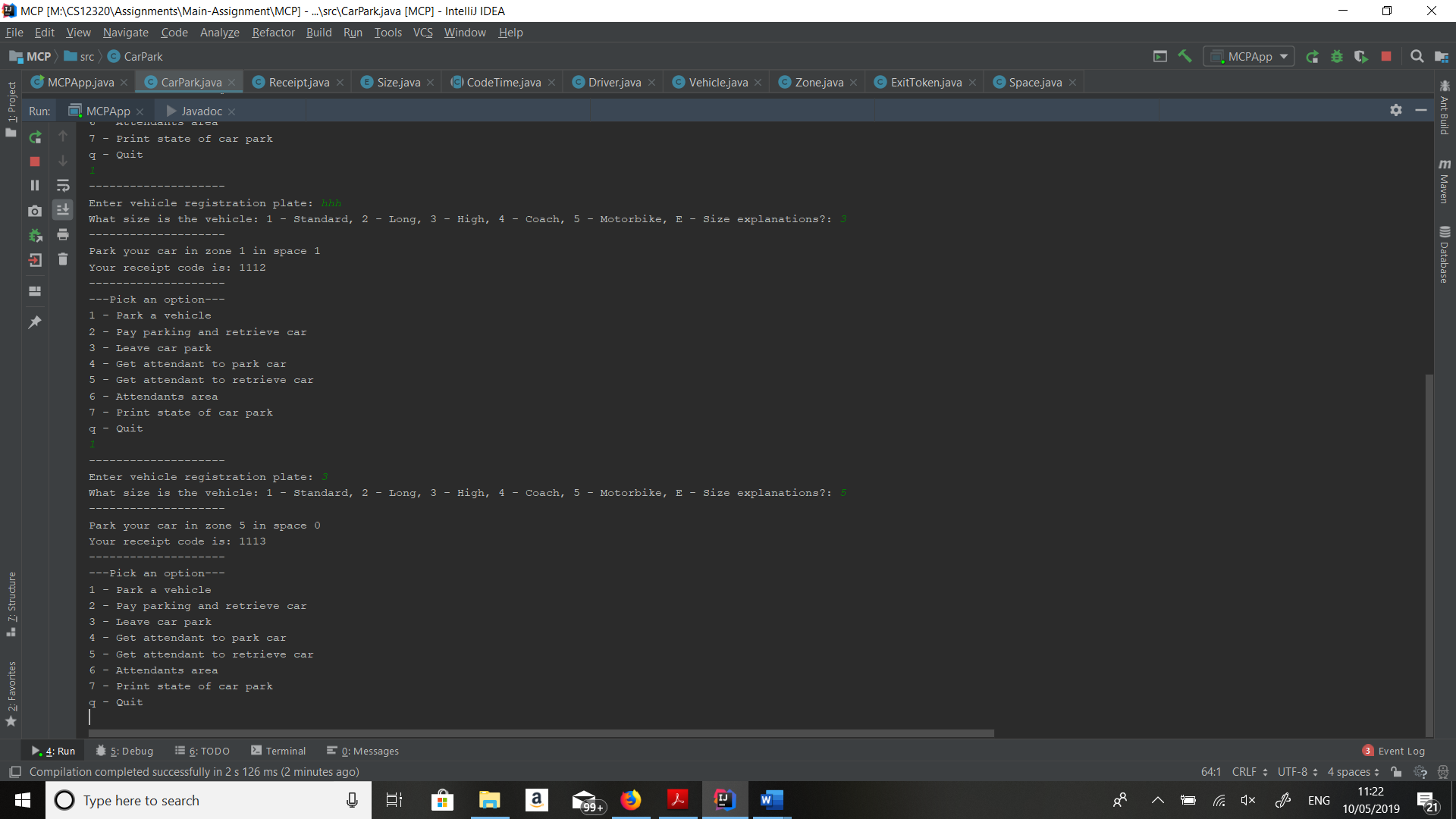
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Requirement | Description | Inputs | Expected Outputs | Pass/Fail | Comments |
| 1 | FR1 | Testing the ability to be able to park a car | N/A | N/A | N/A |  |
| 2 | FR1.1 | Being able to add a car | N/A | N/A | N/A |  |
| 3 | FR1.1.1 | Input a licence plate | SS66 HHH | Moves on to next part of the program | P |  |
| HHH | Moves on to next part of the program | P |  |
| 3 | Moves on to next part of the program | p |  |
| 4 | FR1.1.2 | Pick a size of vehicle | 1 | Sets size as standard | P |  |
| 3 | Sets size as high | P |  |
| 5 | Sets size as motorbike | P |  |
| e | Give size explanation | P |  |
| l | Invalid Input | P & F | It did print out invalid input, but it gets stuck in an infinite loop of printing invalid input |
| 5 | FR1.2 | Allocates a zone and a space for the customer to park in | N/A | N/A |  |  |
| 6 | FR1.2.1 | Picks a zone that accepts the write size vehicles | Using a standard sized vehicle try to park | Parks you in zone 1 or 5 | P |  |
| Using a coach try to park | Parks you in zone 3 | P |  |
| 7 | FR1.2.2 | Allocates the customer an empty space in the zone | Using a standard sized vehicle try to park | Parks you in the first unused space | P |  |
| Using a coach try to park | Parks you in an unused space | P |  |
| 8 | FR1.3 | Get given you a receipt with a unique code | Using a standard sized vehicle try to park | Gives you a receipt which has a unique code | P |  |
| Using a coach try to park | Gives you a receipt which has a unique code | P |  |
| 9 | FR2 | Testing paying the parking fee | N/A | N/A | N/A |  |
| 10 | FR2.1 | Finds the receipt | 1113 | Takes you to the next stage | P |  |
| 1111 | Takes you to the next stage | P |  |
| 0999 | Tells you it is an invalid receipt number | P |  |
| 11 | FR2.2 | Will get the payment off the user | N/A | N/A | N/A |  |
| 12 | FR2.2.1 | Calculate cost for a regular user | Wait for an hour before paying to leave | Charges 1x the cost of the zone | P |  |
| Wait 2 hours before paying to leave | Changes 2x the cost of the zone | P |  |
| Wait 10mins before you pay to leave | Charges 1x the cost of the zone | P |  |
| 13 | FR2.2.2 | Adjust it for a disabled customer in the week | Select disabled customer | Charges half the cost of a regular customer | P |  |
| Select not disabled customer | Charges the expected amount | P |  |
| 14 | FR2.2.3 | Adjust it for a disabled customer parking on a Sunday | Select disabled customer | Charges nothing for disabled customer | P |  |
| Select not disabled customer | Charges the expected amount | P |  |
| 15 | FR2.3 | Giving an exit token | N/A | N/A | N/A |  |
| 16 | FR2.3.1 | Generate a unique token number and give it to the user | Complete the rest of the pay and return form | Gives a unique barrier code | P |  |
| 17 | FR3 | Leaving the car park within the 15mins allowed | N/A | N/A | N/A |  |
| 18 | FR3.1 | Take a token number of a user and find its match in the list of exit tokens | 1111 | Calculates how long you waited | P |  |
| 8029 | Says it’s an invalid token | P |  |
| 19 | FR3.2 | Calculate the time stayed to see if it was under 15mins | Wait 5mins | Let’s you leave | P |  |
| Wait 20mins | Calls the seek assistance method | P&F | It calls seek assistance but also says you’re leaving the car park |
| 20 | FR3.3 | Call seek assistance if the customer has waited more than 15mins after getting the token | Wait 25mins | Prints “Seek Assistance” | P |  |
| 21 | FR4 | Get attendant to park a car | N/A | N/A | N/A |  |
| 22 | FR4.1 | Pass FR1 | N/A | N/A | N/A |  |
| 23 | FR4.2 | Allow attendants to pick space | Attendant picks an empty space | Parks car |  |  |
| Attendant doesn’t pick and empty space | They can’t as it only gives them the option to pick from empty spaces | N/A |  |
| 24 | FR4.3 | Check if there are any free attendants | Run it with no attendants stored | Says no attendants free | P |  |
| Run it with all attendants busy | Says no attendants free | F | Still allocates you an attendant |
| Run it with at least one attendant free | Allocates you an attendant | P |  |
| 25 | FR5 | Get attendant to retrieve a car | N/A | N/A | N/A |  |
| 26 | FR5.1 | Pass FR2 & FR3 | N/A | N/A | F | When adding a vehicle as an attendant it prints off the state of the car park from somewhere and then crashes |
| 27 | FR5.2 | Checking if there are attendants free, enter attendant name | Run it with no attendants stored | Says no attendants free | P |  |
| Run it with all attendants busy | Says no attendants free | F | Same as in test 24 |
| Run it with at least one attendant free | Allocates you an attendant | P |  |
| 28 | FR5.3 | Selecting and using attendant | You can’t test this as you are allocated an attendant | N/A | P |  |
| 29 | FR6 | Adding an attendant | N/A | N/A | N/A |  |
| 30 | FR6.1 | Get name of the new attendant | Select add new attendant | Asks for name of attendant | P |  |
| 31 | FR6.2 | Create attendant | Run it | Attendant is added | P |  |
| 32 | FR7 | Removing an attendant from the system | N/A | N/A | N/A |  |
| 33 | FR7.1 | Type attendants name | Select add new attendant | Asks for name of attendant | P |  |
| 34 | FR7.2 | Removing attendant from the system | Run it | Attendant is removed | P |  |
| 35 | FR8 | Activating an Attendant | N/A | N/A | N/A |  |
| 36 | FR8.1 | Get attendants name from user | Attendants name | Accepts it | P |  |
| 37 | FR8.2 | Makes attendant active | Move to the next step | Attendant is made active | F | Can’t test as I can’t deactivate an attendant |
| 38 | FR9 | Deactivating an attendant | N/A | N/A | N/A |  |
| 39 | FR9.1 | Gets the name of the attendant from the user and validates it is an attendant | Enter attendant name | Accepts it | P |  |
| 40 | FR9.2 | Makes attendant busy | Passes from the name entry | Attendant is made busy | F |  |
| 41 | FR10 | Will output the current state of the car park | Select view state of the car park | Outputs the state of the car park | P |  |

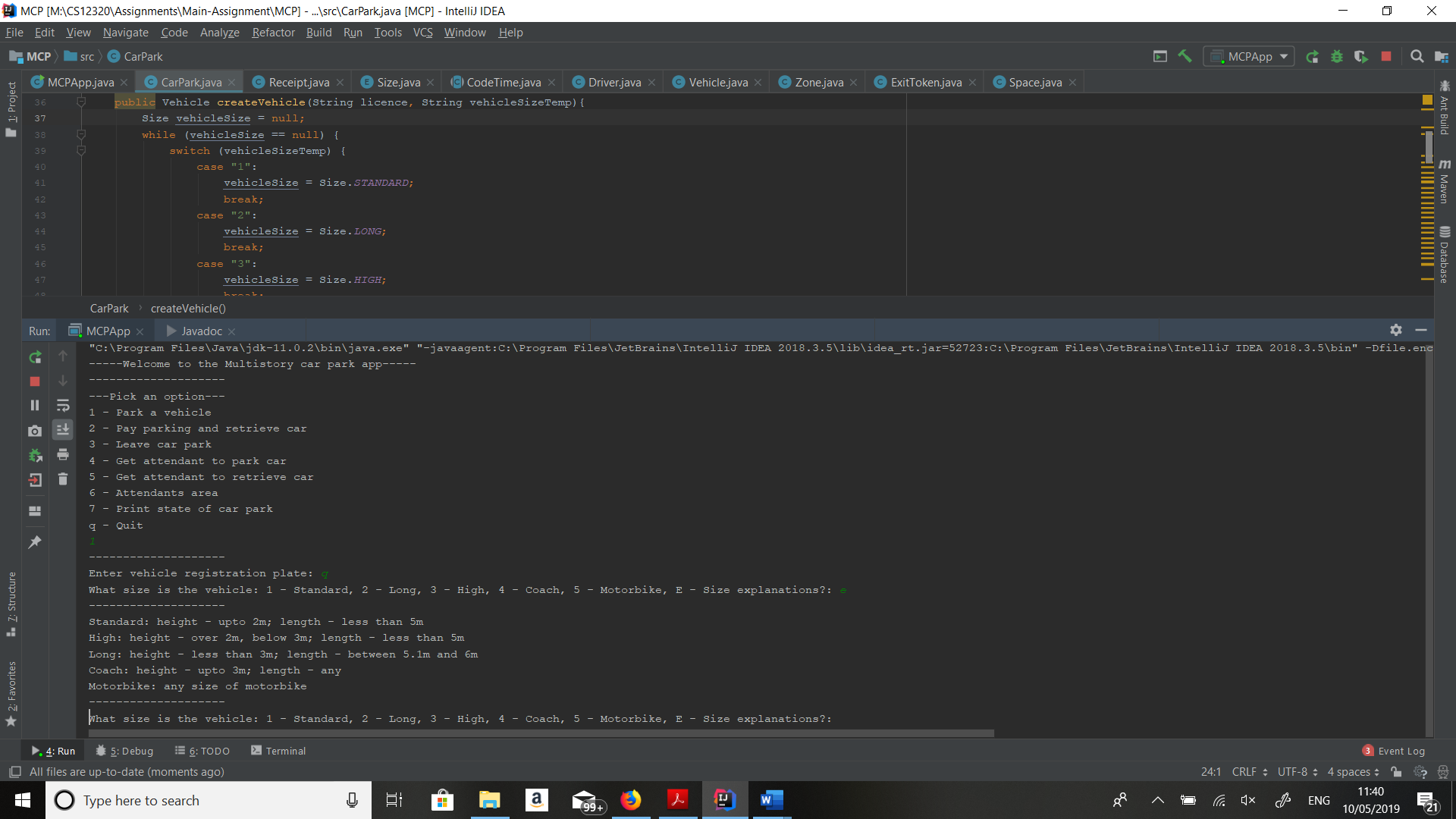
I chose to test with these inputs as it means that there is a basic covering of the whole of the programs function, and it means I can establish if all the fundamental parts of the program are functioning correctly.

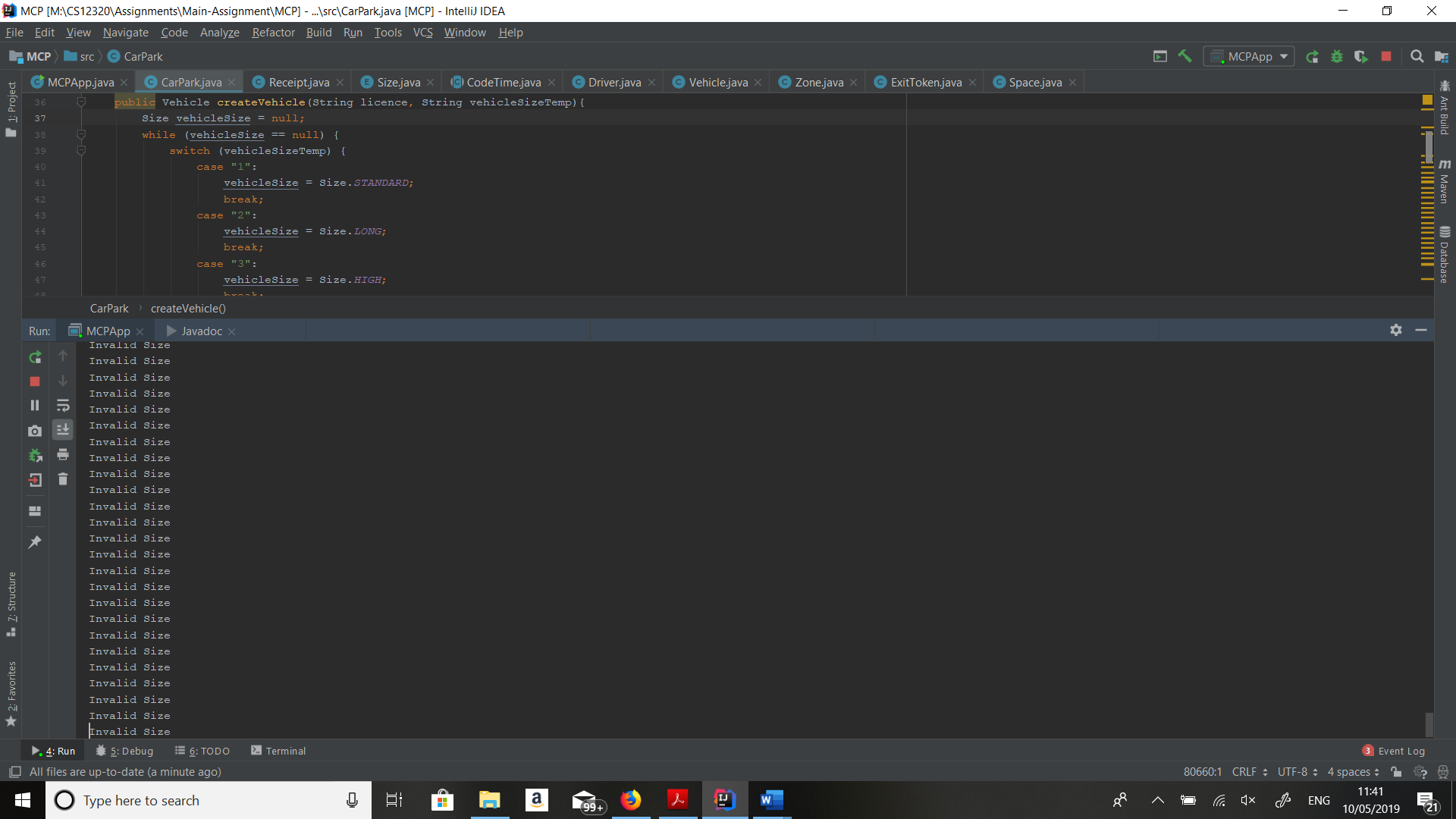
## Screenshots

Parking a vehicle

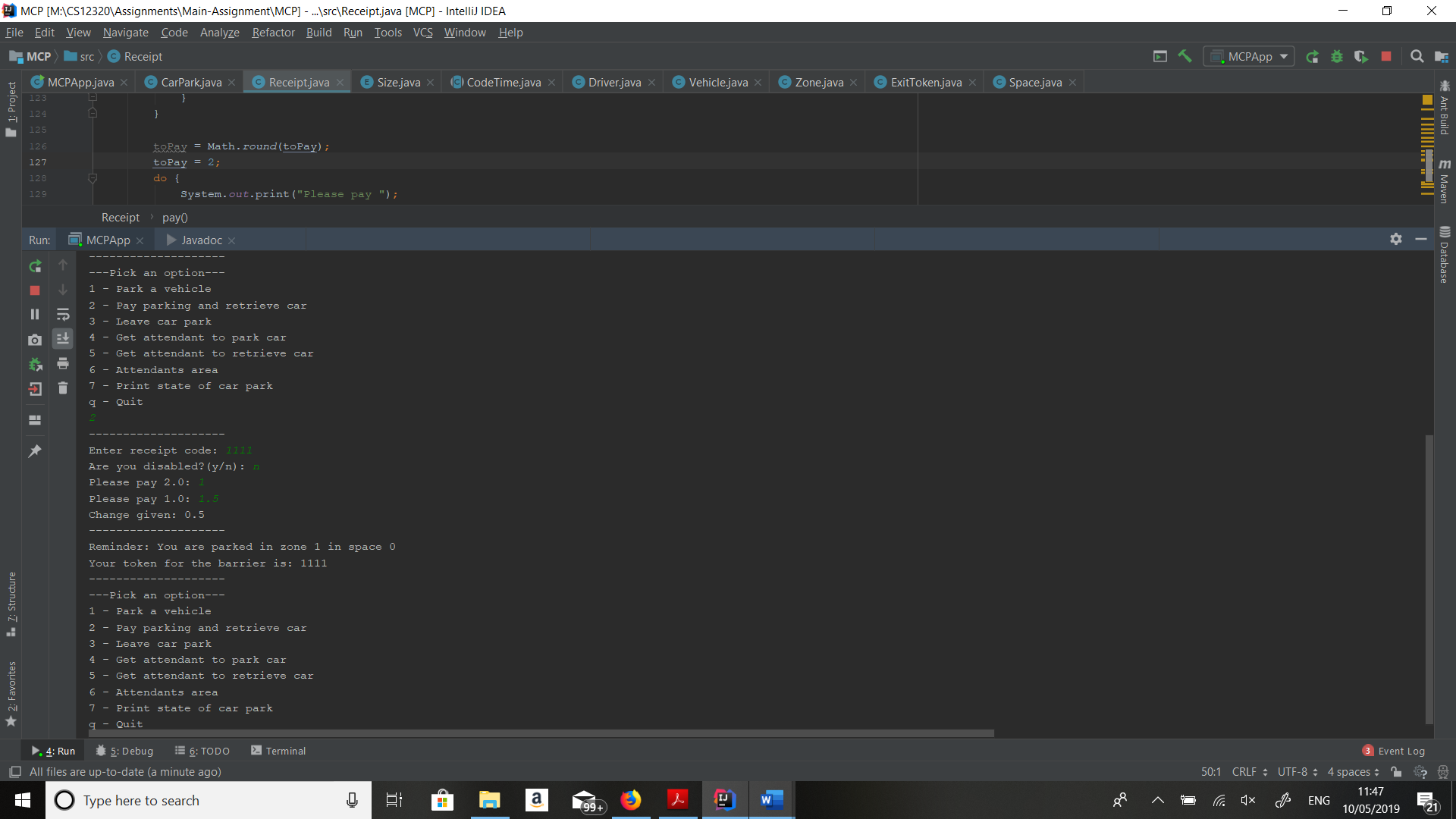


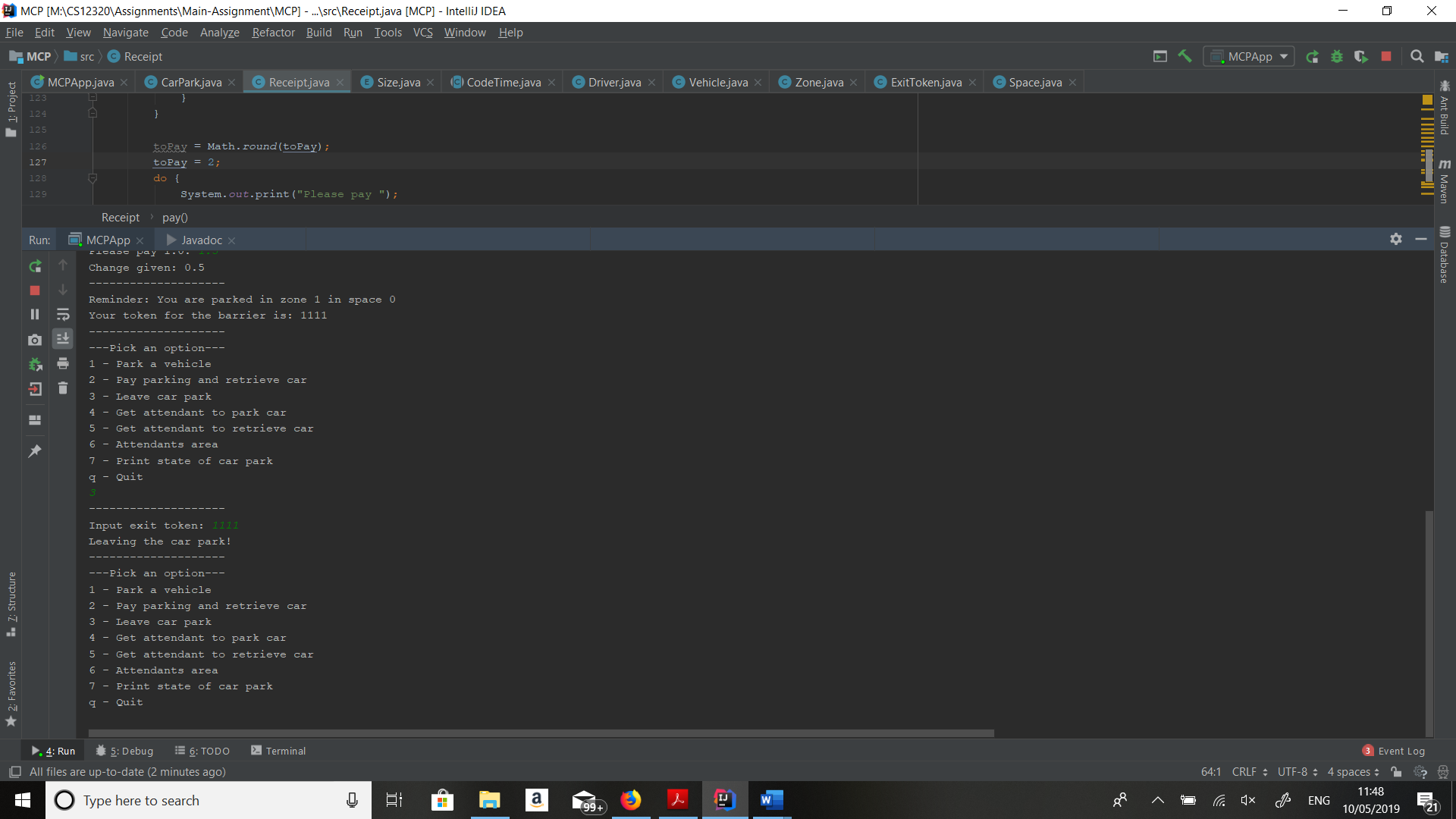


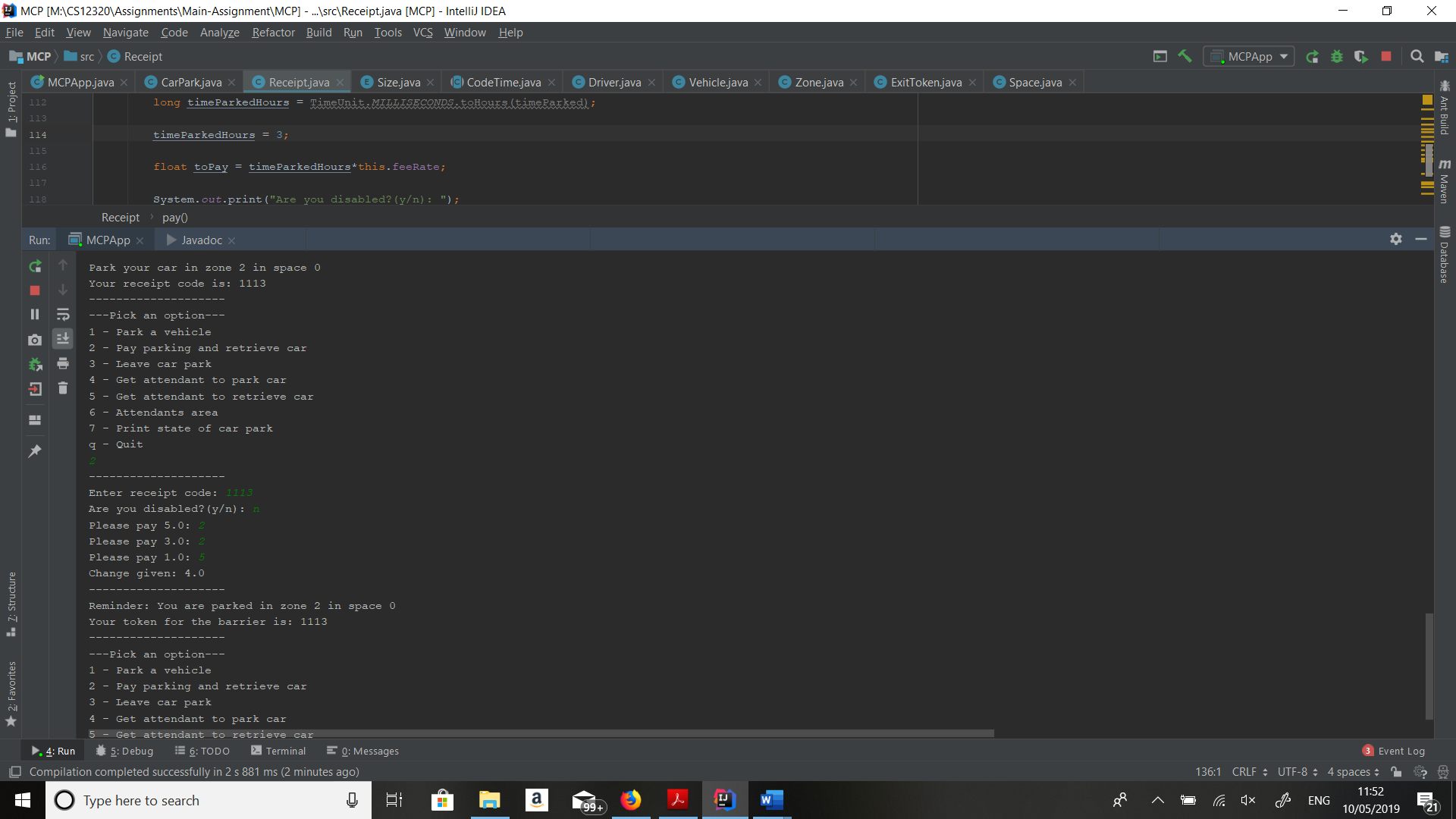


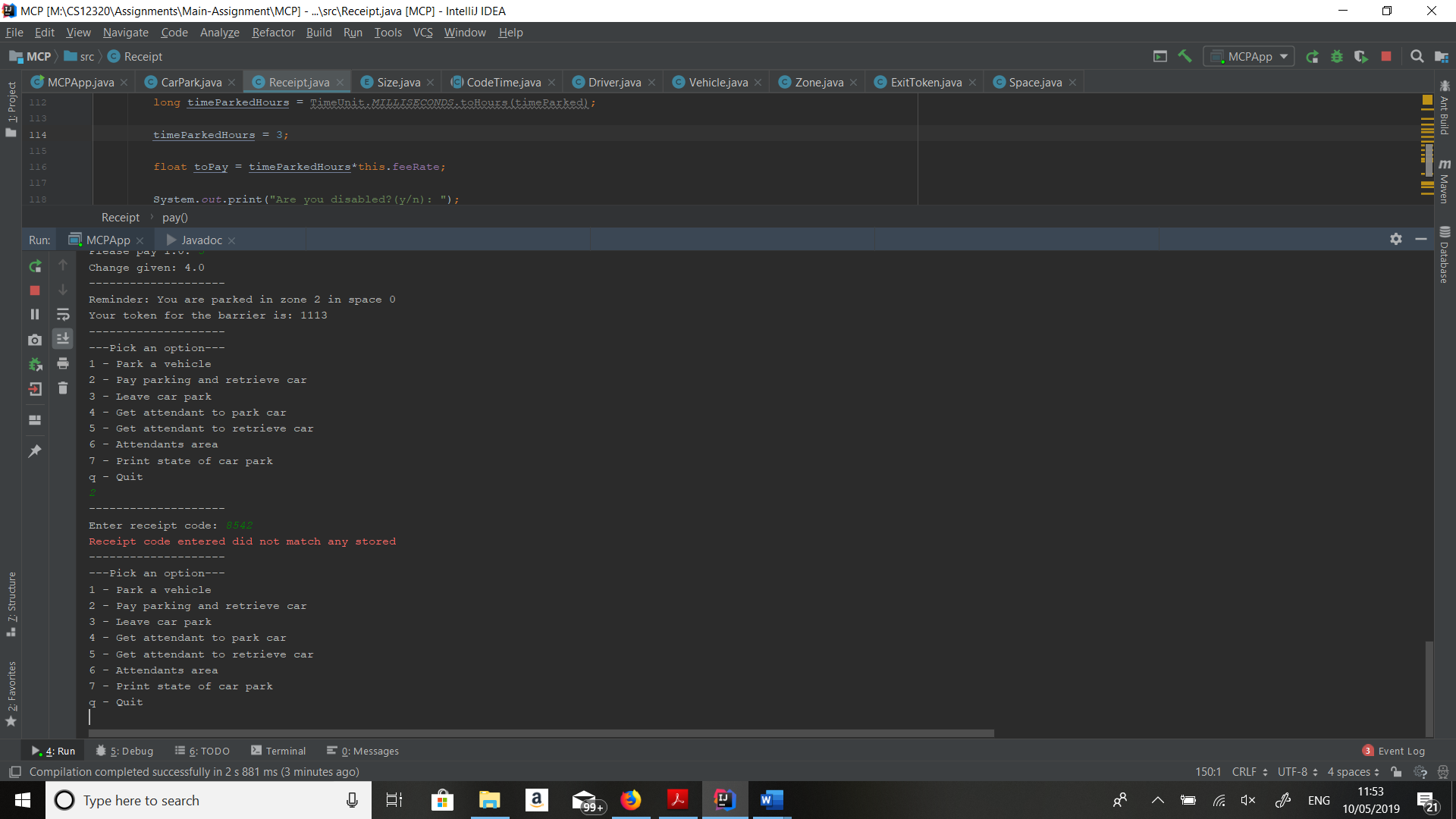


Paying to retrieve vehicle

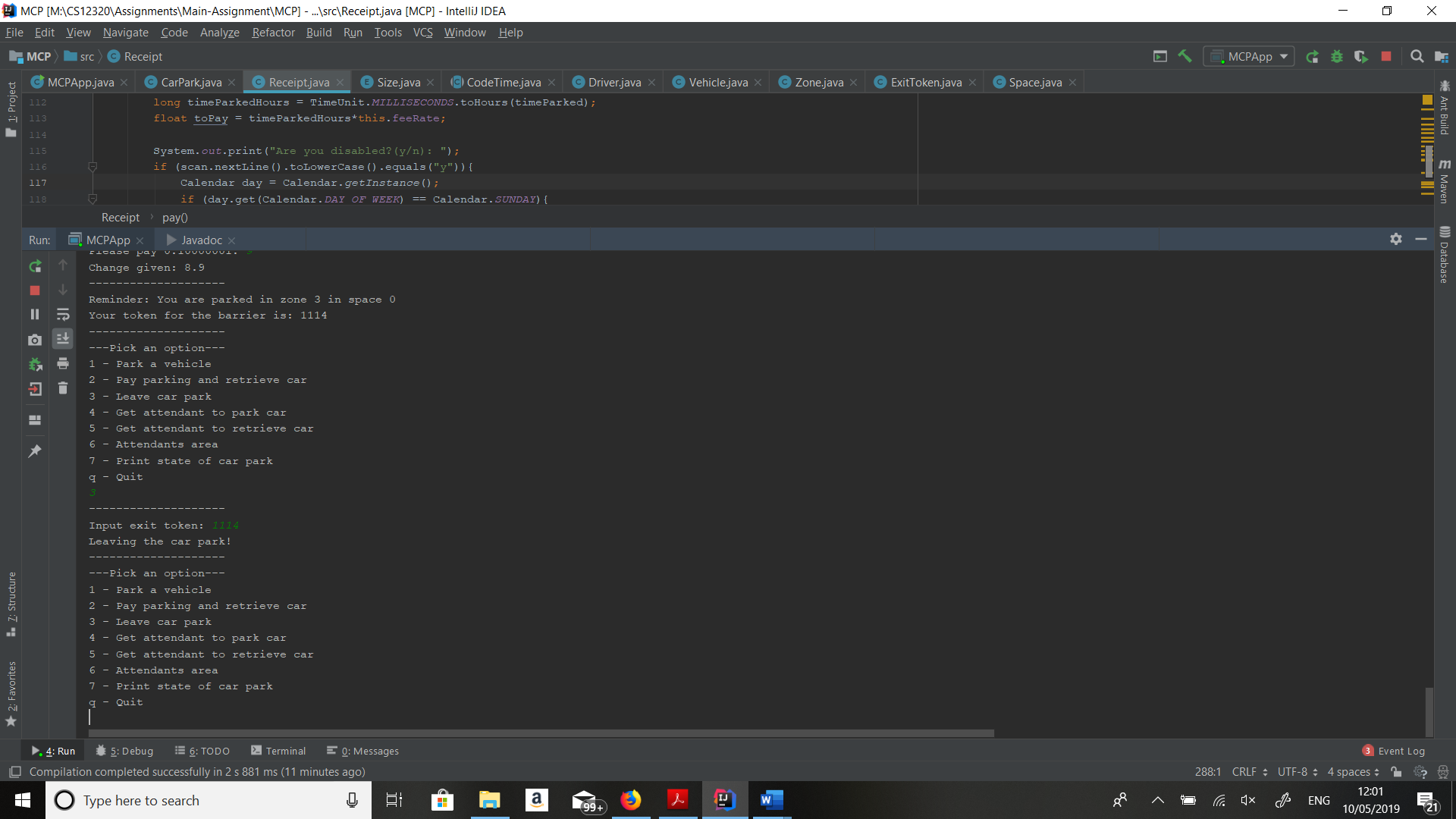


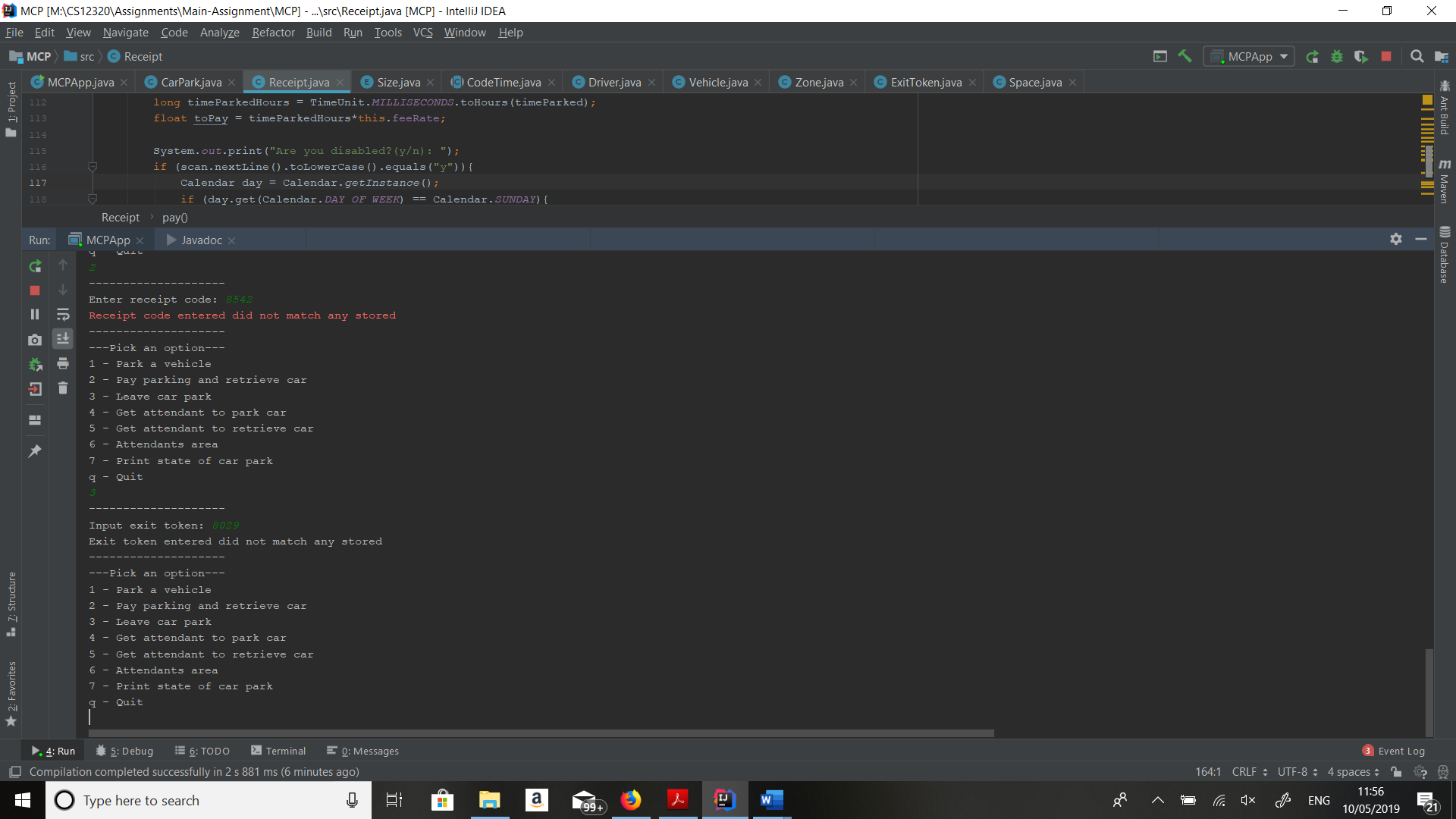


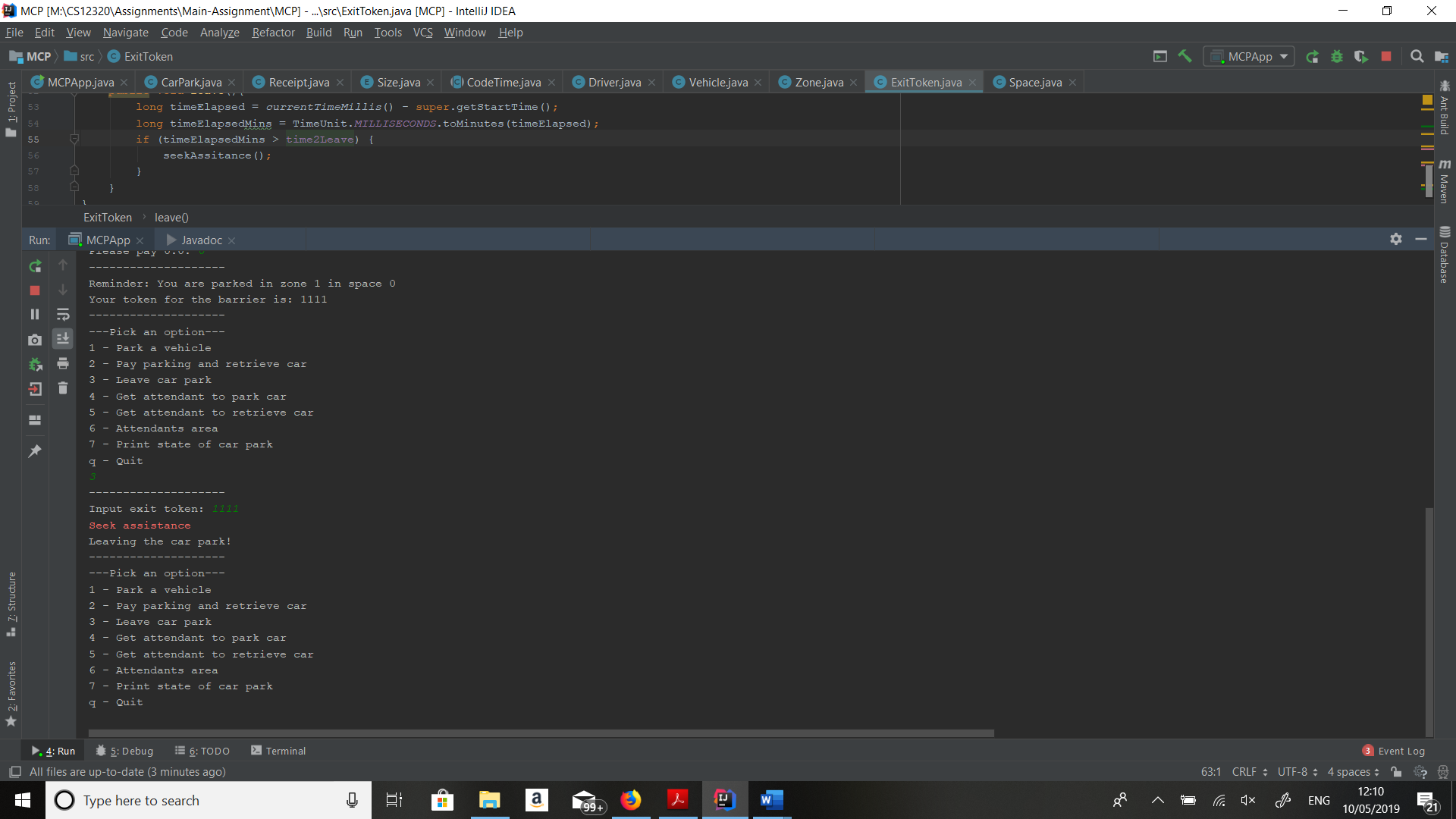




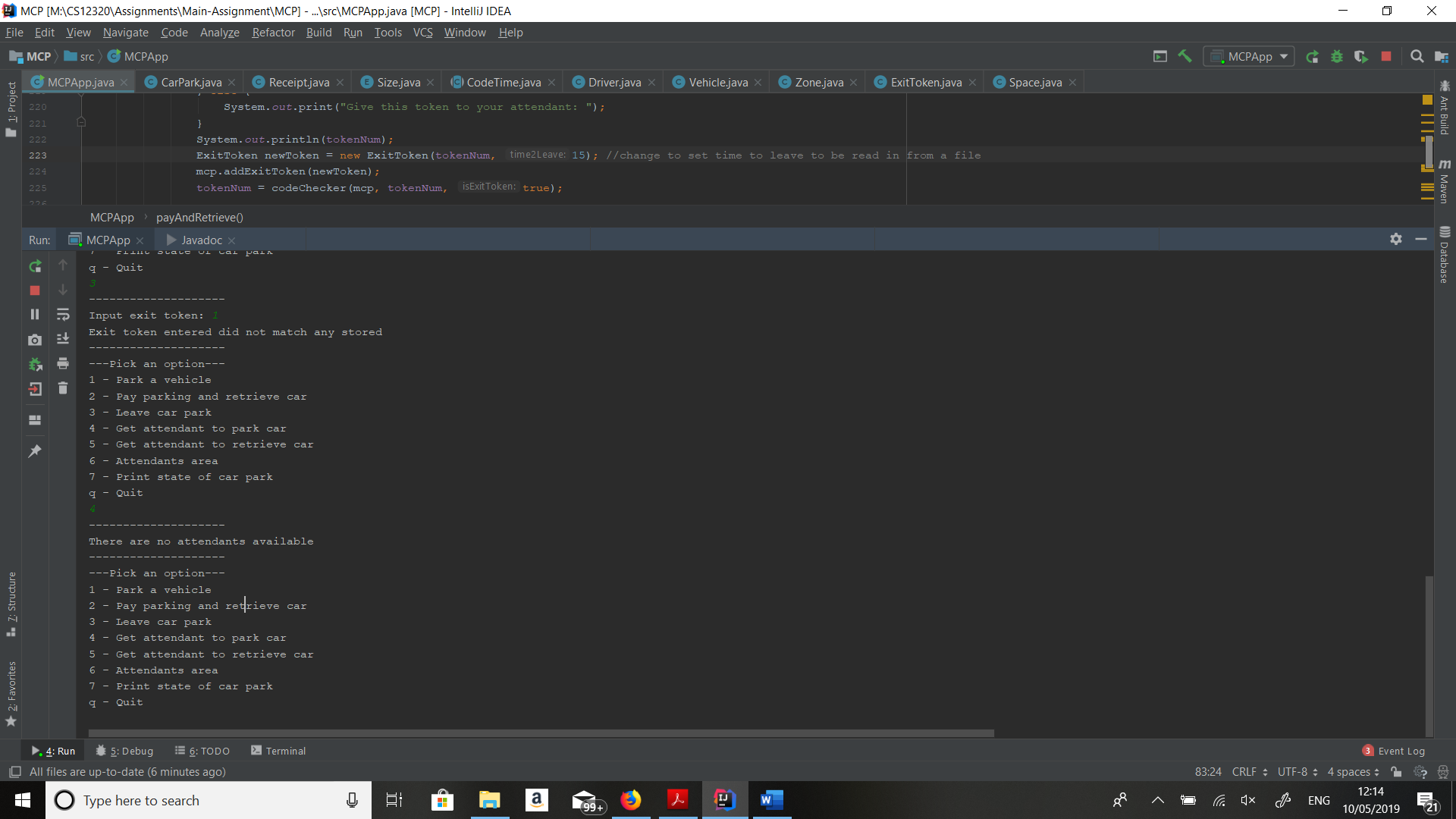
Leaving the car park

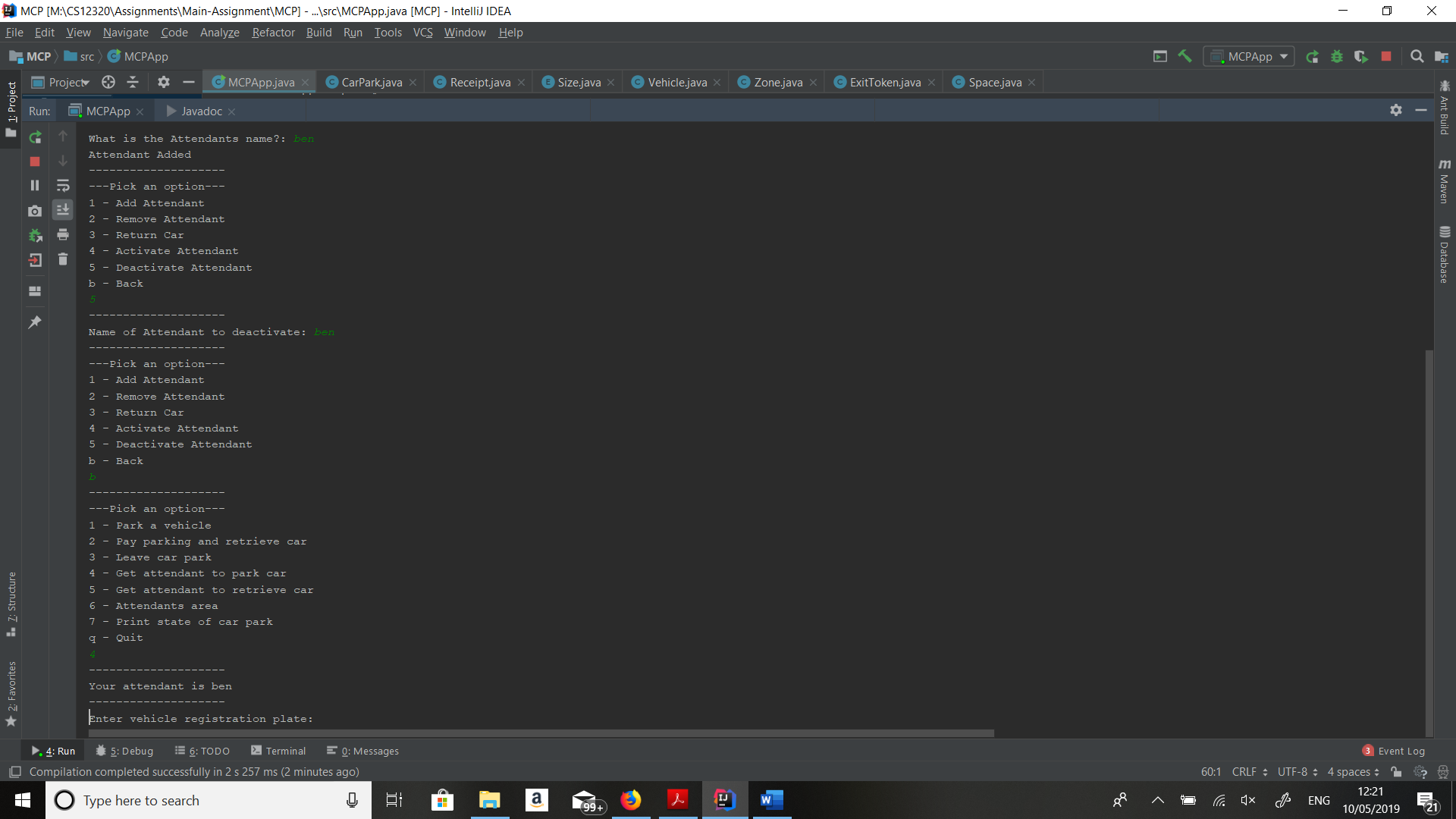




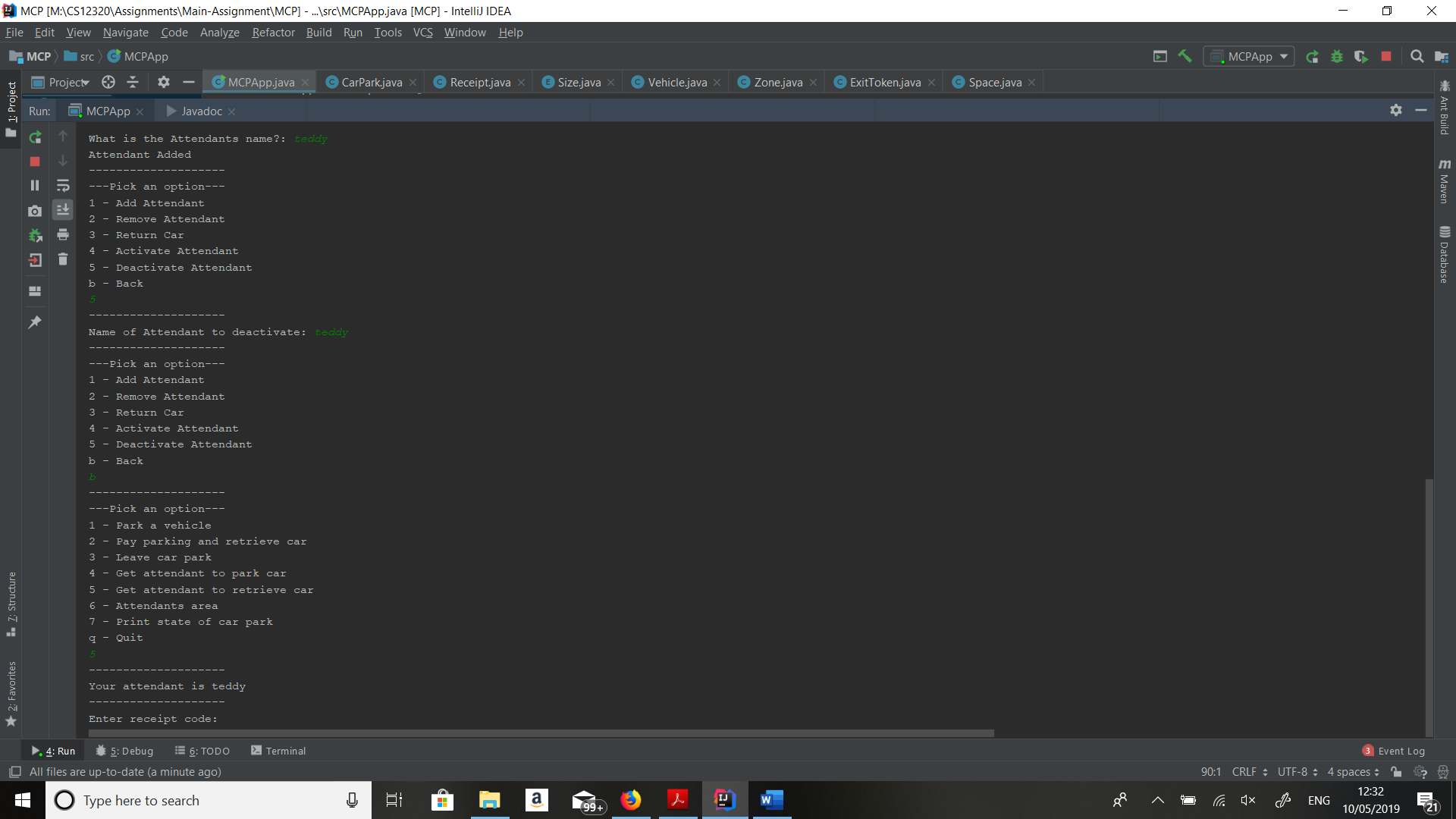


Get attendant to park car

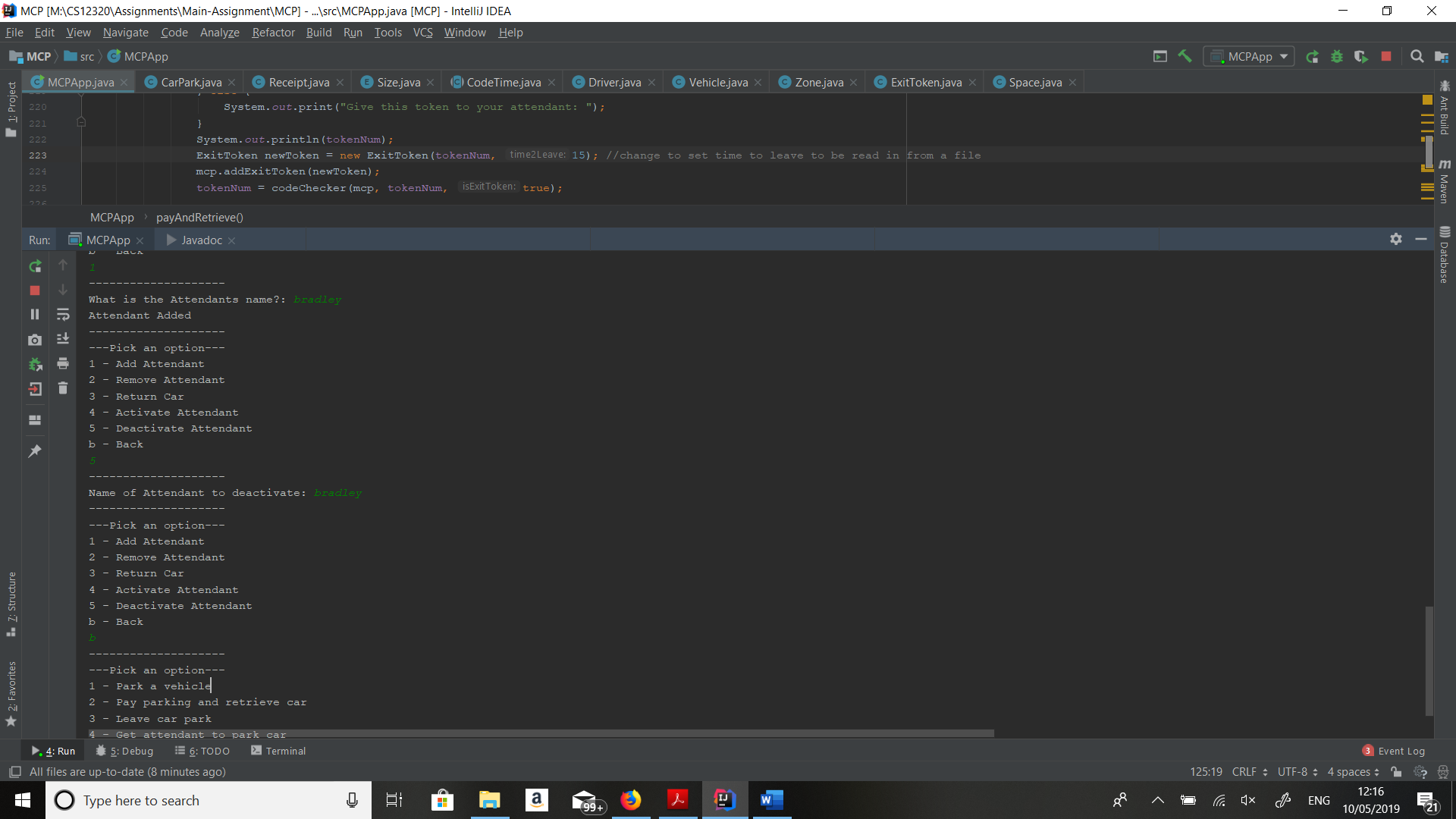




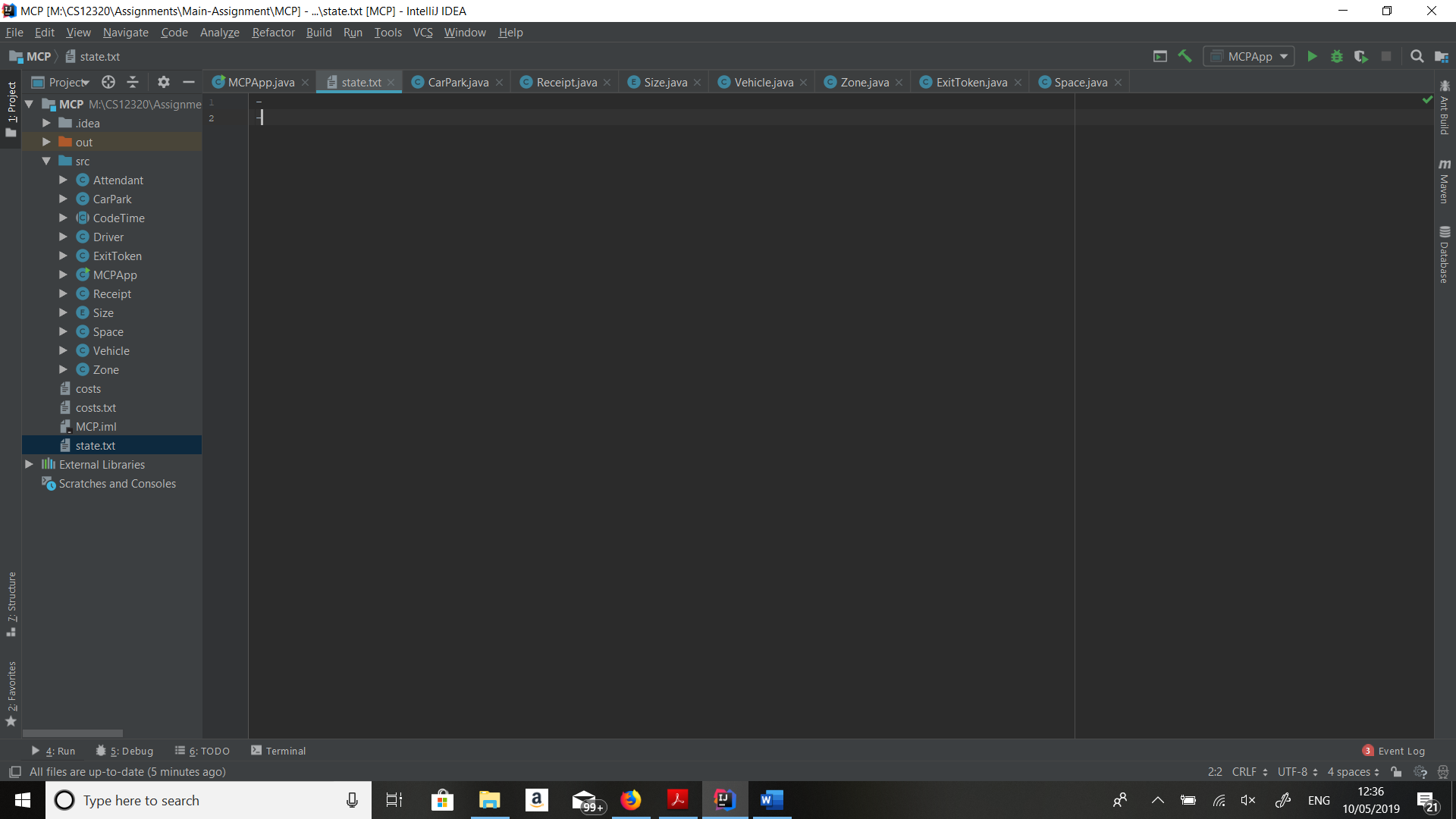
Get attendant to retrieve car

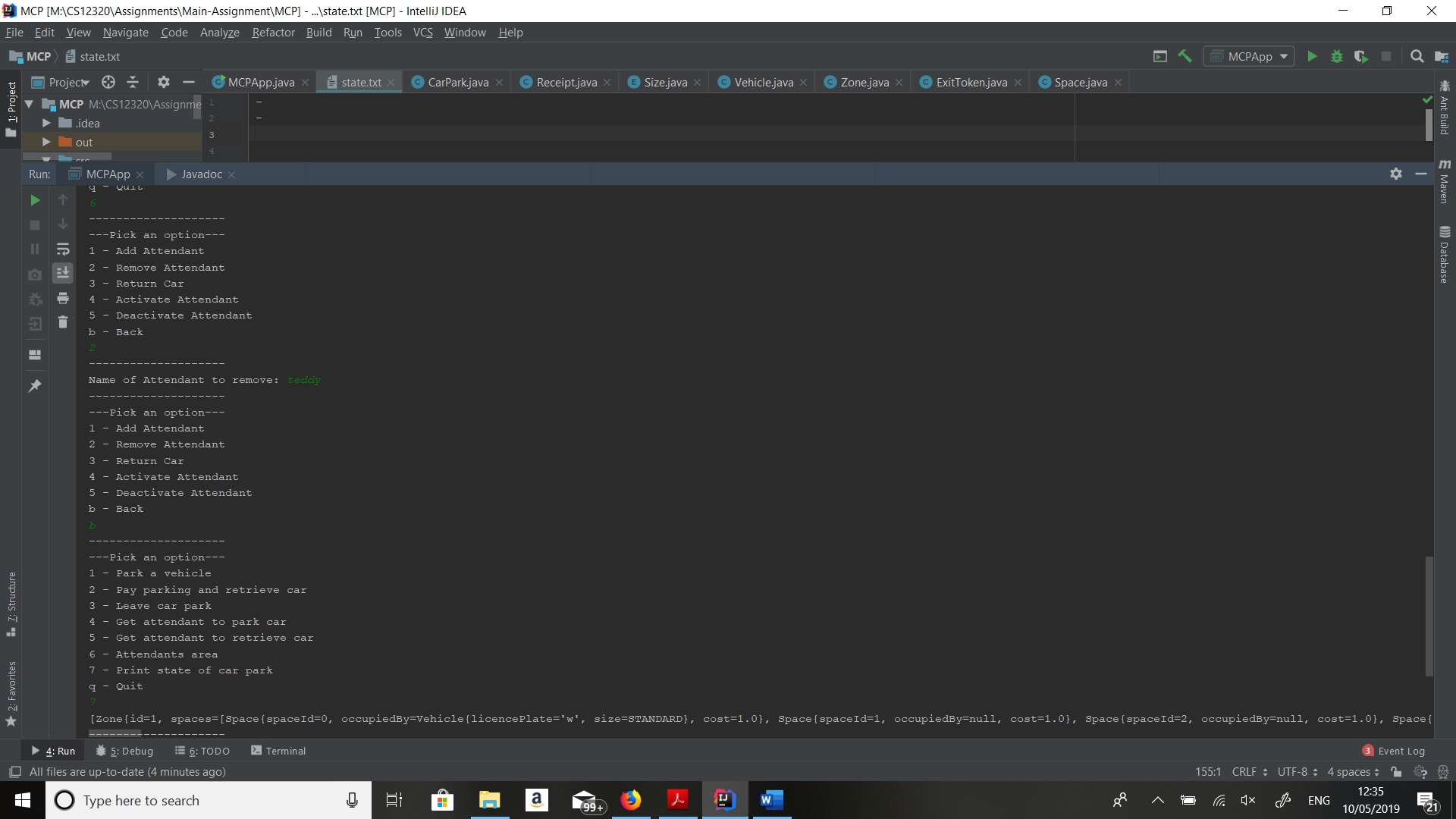


Adding Attendant

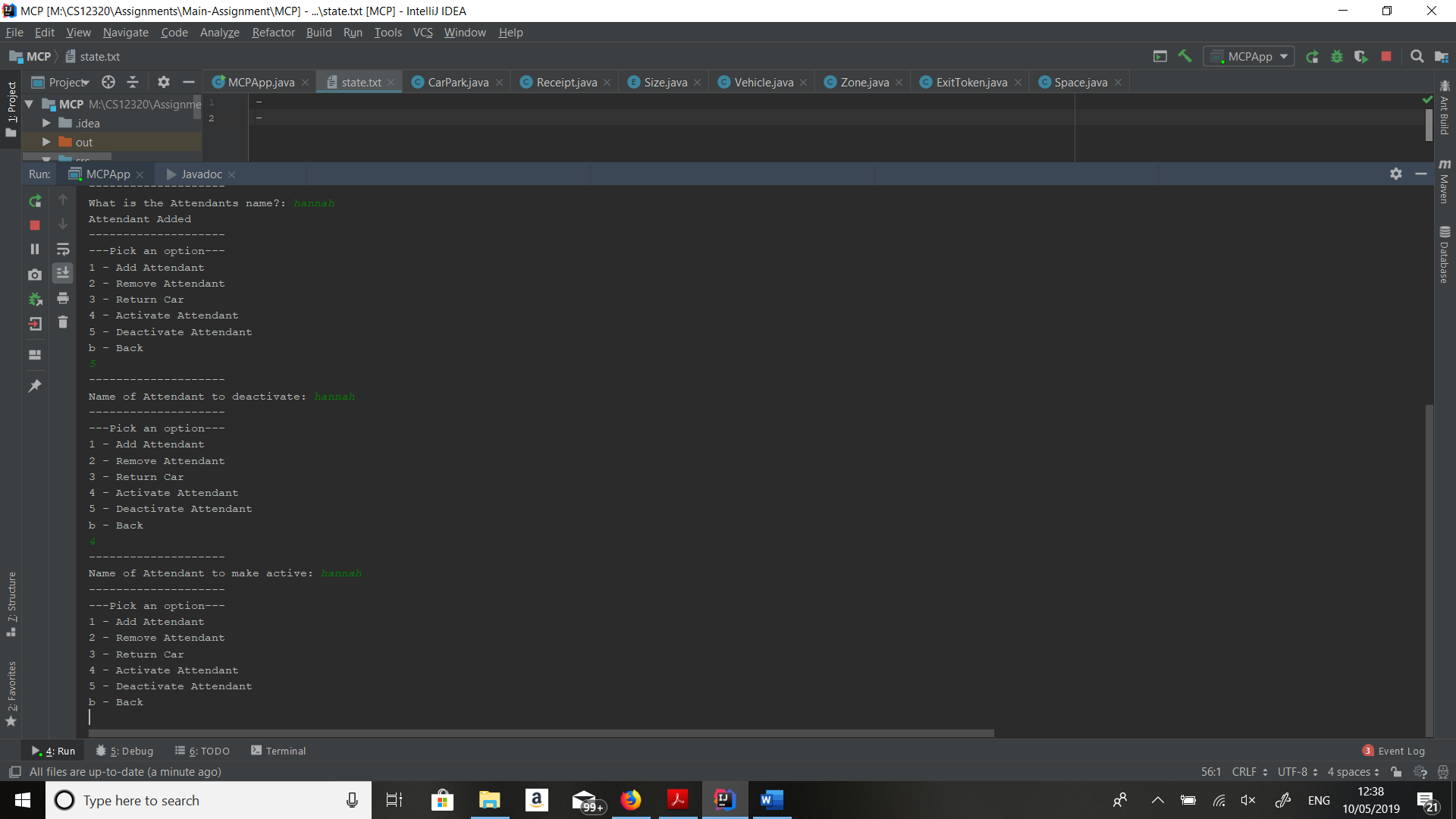


Remove Attendant

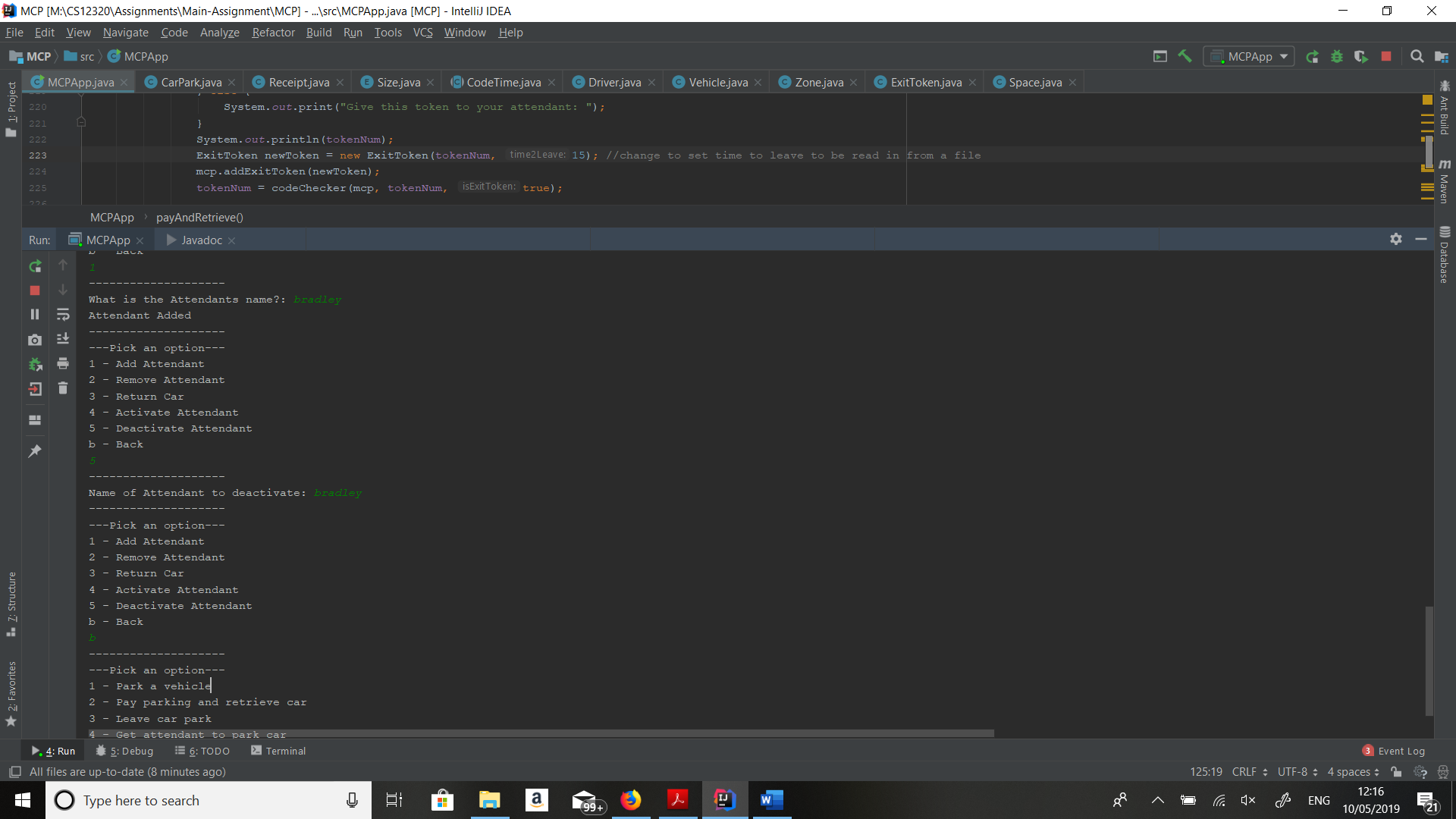
 State file after saving



Activating attendant



Deactivating attendant



# Evaluation

I went about solving the problem by creating the basic outline of each class then worked outwards from there to build the full menu with each menu section being made to work. I started using enumerated types as I believed them to be a nice solution as I could use them in the sizes of vehicle the zone can take and in the sizes of the car. When I was implementing the receipt and the exit tokens, I realised it would make a nice solution for them to inherit from one super class as they store some similar information (code and time generated).

I didn’t get chance to implement the load method, although this wasn’t in the original problem description, also the activation and deactivation of attendants doesn’t work.

Overall, I believe I should get around 60% as my code isn’t very efficient and could be a lot nicer and parts of the code don’t work. Also, I don’t believe my documentation is as good as it could be, as my testing could be in more detail and I could have put more screenshot, showing the system more extensively.