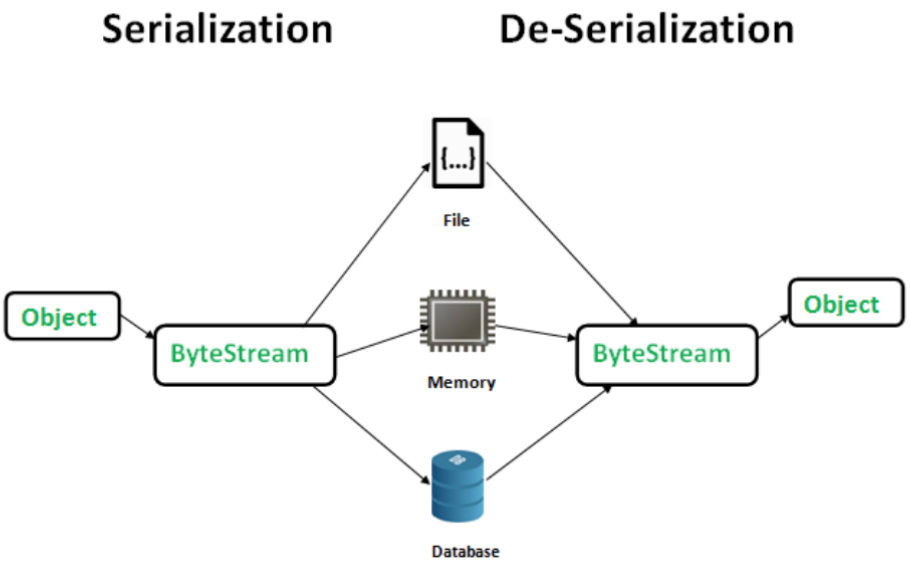
Criterion C: Development

Index:

List of Techniques:

1. Serialization

Serialization is a mechanism of converting the state of an object into a byte stream. Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory. (GeeksforGeeks, 2019)



I used serialization in my program, in order to create independent files for each of the ArrayLists storing the parking space addresses. Thanks to these files, I was able to help my client store, save, organize and update all of the parking space addresses data, and fill up it's according arraylists even when the program was terminated and the computer was turned off.

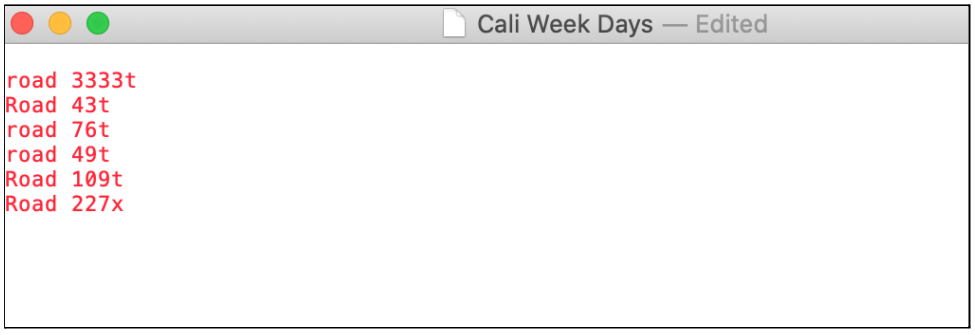
Serialization of ArrayList:

```
321         // Serealization of Arralist: BogotaWeekDay
322         try {
323             FileOutputStream fos = new FileOutputStream( name: "Bogota Week Days");
324             ObjectOutputStream oos = new ObjectOutputStream(fos);
325             oos.writeObject(BogotaWeekDay);
326             oos.close();
327             fos.close();
328         } catch (IOException ioe) {
329             ioe.printStackTrace();
330         }
```

De-Serialization of ArrayList:

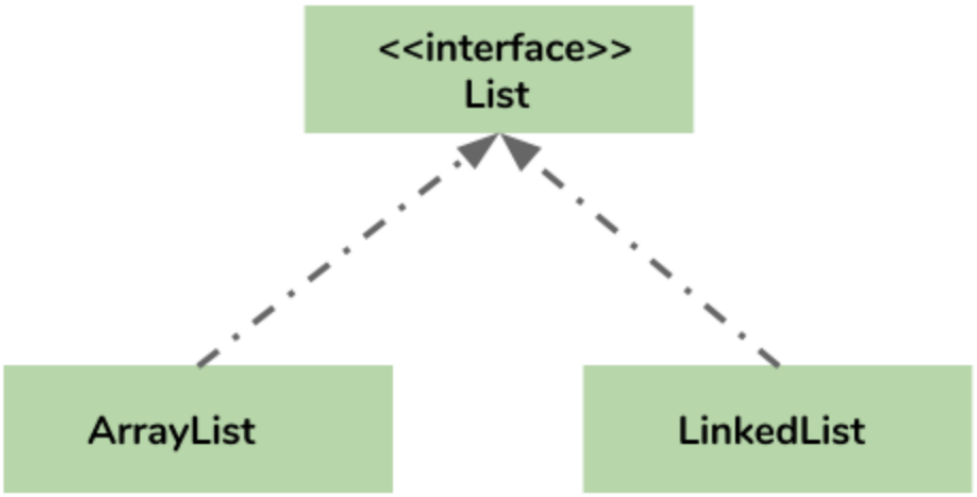
```
34      // De-Serealization for the Arraylist: BogotaWeekDay
35              try
36              {
37                  FileInputStream fis = new FileInputStream( name: "Bogota Week Days");
38                  ObjectInputStream ois = new ObjectInputStream(fis);
39
40                  BogotaWeekDay = (ArrayList) ois.readObject();
41
42
43                  ois.close();
44                  fis.close();
45              }
46              catch (IOException ioe)
47              {
48                  ioe.printStackTrace();
49                  return;
50              }
51              catch (ClassNotFoundException c)
52              {
53                  System.out.println("Class not found");
54                  c.printStackTrace();
55                  return;
56              }
```

Example of file storing the data (Addresses):

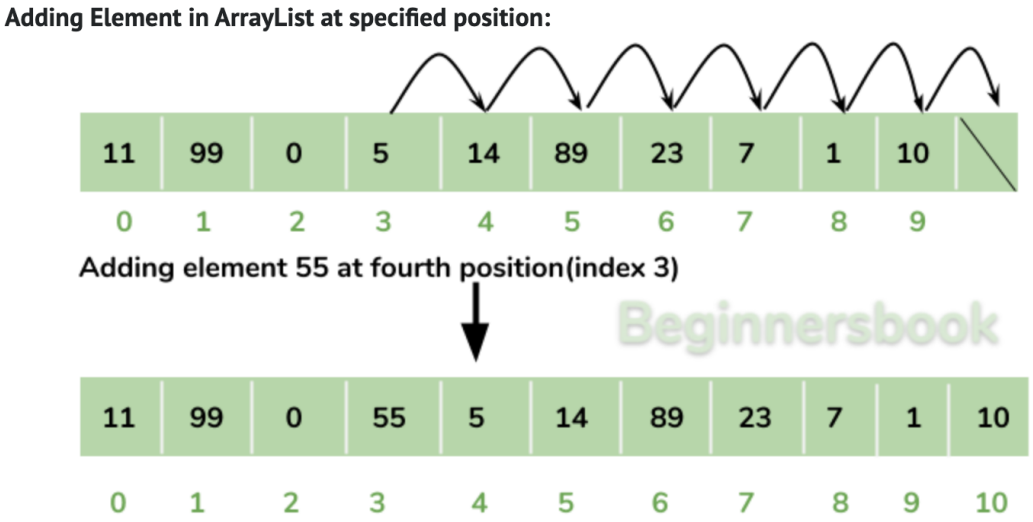


Serialization further information appendix 12 in “Main Class” Section.

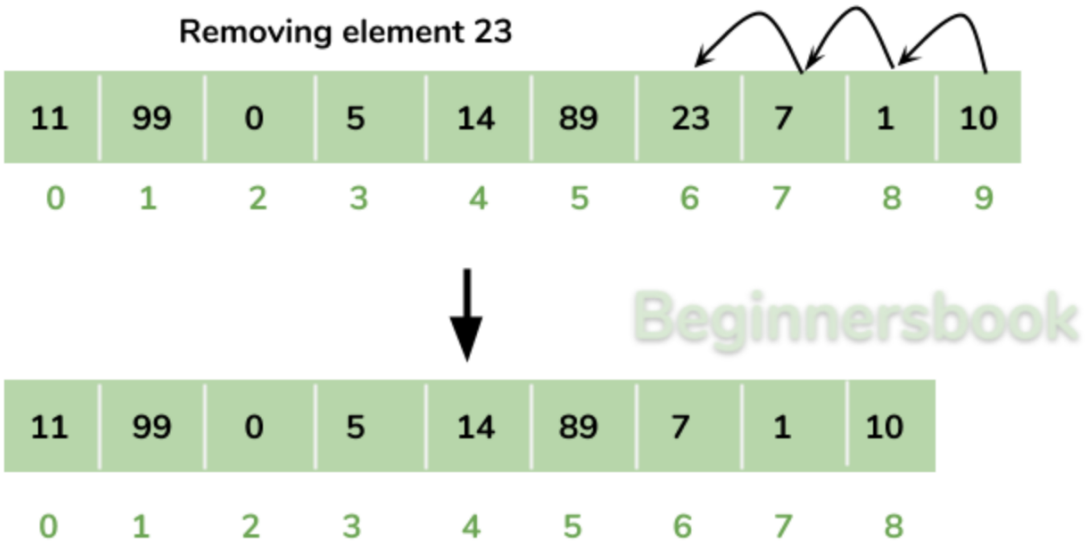
2. ArrayLists



ArrayLists are flexible and allow elements to be added and deleted, enabling me to satisfy my clients needs, as he will be constantly adding and removing parking spaces addresses from the program. (Singh, 2019)
Below, we can see the process which is used (and which I used) in programming to add and remove elements from an Arraylist.



Removing Element from ArrayList:



The following image shows the ArrayLists in my program. Initial values will be added, once my client starts the program and uses the “Add Parking Spot” function.

```
15      // Array lists to store parking spot addresses
16
17      static ArrayList<String> CaliWeekDay = new ArrayList<>();
18
19      static ArrayList<String> CaliWeekEnds = new ArrayList<>();
20
21      static ArrayList<String> BogotaWeekDay = new ArrayList<>();
22
23      static ArrayList<String> BogotaWeekEnds = new ArrayList<>();
```

The code below shows when the ‘add’ and ‘remove’ methods from the ArrayList collection were used.

```
53                  // Adding element to ArrayList
54                  CaliWeekDay.add(adress);
```

```
56                  // Removing element from ArrayList
57                  CaliWeekDay.remove((adressimput - 1));
```

ArrayList further information: appendix 12 , in the ‘Main Class’

3. User-Defined Objects

An object is a basic unit of Object-Oriented-Programming, which is used to represent real life entities from classes, and they allow the programmer to use different classes' functionalities and apply them or use them in multiple parts of the program. (GeeksforGeeks, 2018)

My program had multiple classes, which were used to perform and create functionalities which my client requested during our conversations. In order to access and use data, functionalities, etc, from different classes, User-defined objects were created.

Below is an example of my code, where I instantiated an object from the “Looking_Spot” and “Renting_Spot” class, within the “Main” Class

```
171                 // User-Defined Object
172                 Looking_Spot lookingForSpot = new Looking_Spot();
```

```
248                 //USer-Defined Object
249                 Renting_Spot RentMySpot = new Renting_Spot();
```

4. User-defined methods

A user-defined method is a block of code created by the programmer, which is tailored and suits the needs of the user, in order to create specific functionalities. (Study.com, 2020)

My user needed a function that allowed him to calculate the price he needed to charge a customer when renting a space. As a result, I created different user-defined methods, which perform arithmetic calculations depending on the user input (day of the week and amount of days) and will tell my client how much he needs to charge for that specific space.

The code below shows the user-defined methods:

```
19      // Method to calculate parking space price during week days
20      public static int priceWeekDays() {
21
22          int pwd = priceParkWeekDays * Looking_Spot.days;
23
24          return pwd;
25
26      }
27
28
29      // Method to calculate parking space price in week ends.
30      public static int priceWeekends() {
31
32          int pwe = priceParkWeekends * Looking_Spot.days;
33
34          return pwe;
35      }
36
37      // Method to calculate monthly promotion
38      public static int priceMonth(){
39
40
41          return priceMonth;
42
43      }
```

5. Simple (if-else) and Complex Selection (nested if, if with multiple conditions or switch)

The selection process allows the programmer to plan ahead and create a certain output or functionality for each specific path/decision the client decides to take in the program. (GeeksforGeeks, 2019)

In the first meeting with my client (Appendix 1), we decided on creating the Main Menu, where he will be able to choose different options.

The code below shows how the 'if','else if' and 'else' statements work within the Main Menu, where depending on the choice a certain functionality will follow accordingly.

```
163
164         // Option 1 - "Renting out parking spot" - if statement
165
166         if (option == 1) {
167
168             System.out.println("\n Hello! In which city will your client like to park? Bogota or Cali?");
169             String cityChoice = keyboard.next();
```

```
224             // Option 2 - "Adding Spot" - else if statement
225
226             else if (option == 2) {
227
228                 System.out.println("\n Adding Parking Spot to your data base...");
```

```
308             // Option 3 - View available parking spots - else if statement
309
310             else if (option == 3) {
311
312                 ViewParkingSpots();
313             }
```

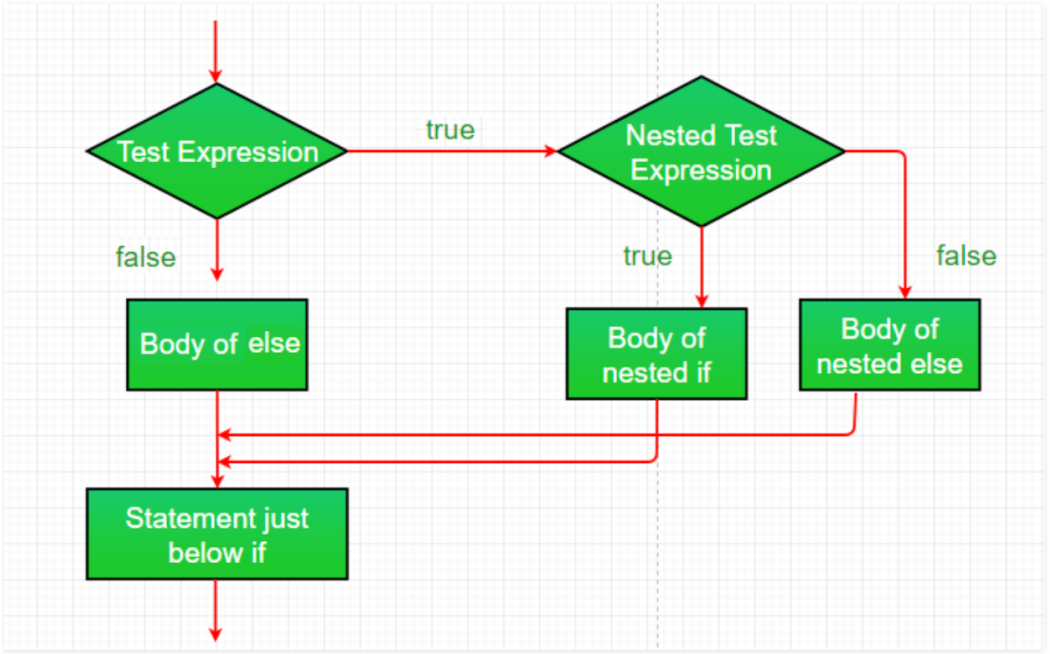
```
314               // Option 4 - "EXIT"
315
316               else if (option == 4) {
317
318                   System.out.println("\n See you soon " + name + "!");
319
320                   break;
```

```
376               // else Statement - Error handling
377           } else {
378               System.out.println("\n ERROR! Please input a correct option. Between 1 and 4");
379           }
```

For further detail, check appendix 12 in the "Main Class" Section:

Further on, nested ‘if’ is an example of complex selection, which works as an ‘if’ statement inside another ‘if’ statement. (GeeksforGeeks, 2019)

Nested ‘if’ representation:



In the “Renting_Out Spot” Class ‘nested if’ was used. This allowed me to give my user a variety of options and taylored my client's path, depending on his different inputs.

The code below, shows the nested ‘if’:

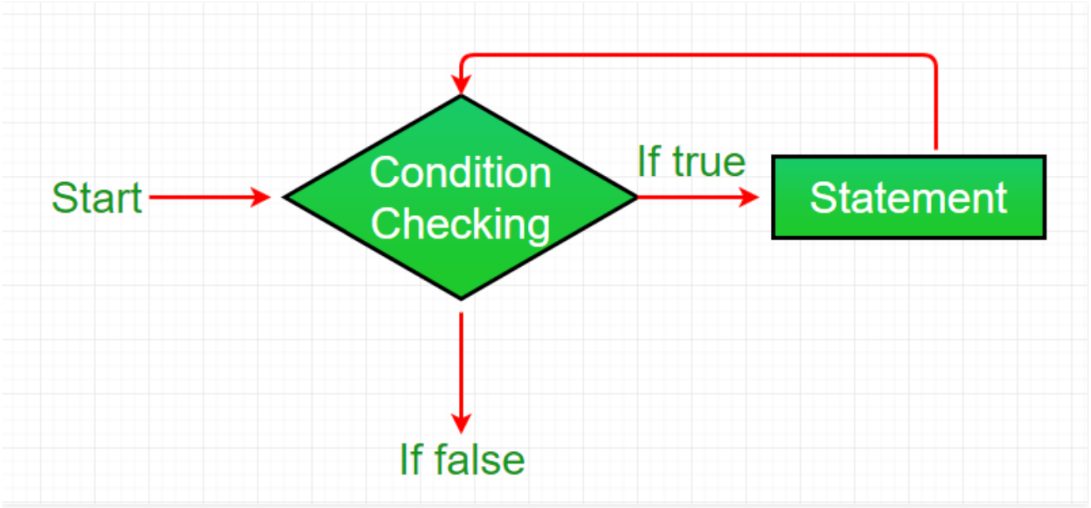
```
34          // if Statement
35          if(CaliWeekDay.size() >= 1 ) {
36              System.out.println();
37              System.out.println("\n These are the available parking spots in Cali During Week Days:");
38              System.out.println();
39              for (int i = 0; i < CaliWeekDay.size(); i++) {
40
41                  System.out.println((i + 1) + ") " + CaliWeekDay.get(i));
42              }
43
44              System.out.println();
45              System.out.println("\n Please choose an option. Type the number that corresponds" +
46                      " to the address which is being rented out");
47              adressimput = keyboard.nextInt();
48              for (int i = 0; i <= CaliWeekDay.size(); i++) {
49
50                  // Nested if
51                  if ((adressimput - 1) <= CaliWeekDay.size()) {
52                      System.out.println("\n You choose address: " + CaliWeekDay.get(adressimput-1)
53                              + " as the parking spot being rented out");
54
55                      // Removing element from ArrayList
56                      CaliWeekDay.remove((adressimput - 1));
57                  }
58                  System.out.println();
59                  System.out.println("\n Weekdays parking is $10");
60                  System.out.println("\n For how many days will your client rent the parking spot?");
61                  days = keyboard.nextInt();
62
63                  // Nested if
64                  if (days < 30) {
65                      System.out.println("\n Your client's total cost will be: $" + extras.priceWeekDays());
66                  } else if (days >= 30) {
67                      System.out.println("\n Your client's monthly promotion will have a total cost of: $" + extras.priceMonth());
68                  }
69
70                  System.out.println("\n Please choose your client's payment method. \n 1)Credit card \n 2) Cash");
71                  int paymentmethod = keyboard.nextInt();
72
73                  exit = exit + 1;
74
75                  if (paymentmethod == 1) {
76                      System.out.println("\n The parking spot has being rented out. Congratulations!");
77
78                  } else if (paymentmethod == 2) {
79                      System.out.println("\n The parking spot has being rented out. Congratulations!");
80
```

6. Loops

Looping is a feature that facilitates the execution of a set of instructions/functions repeatedly while some condition evaluates to ‘true’. (GeesforGeeks, 2020)

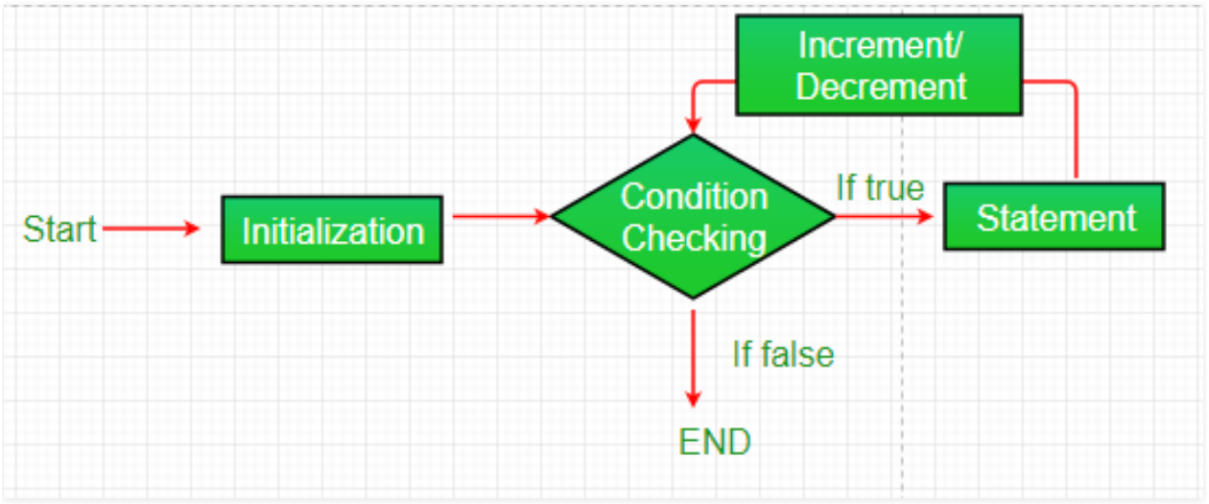
Two of the main requirements of my client were for the Main Menu to be repeated unlimited times, and to iterate/loop over each Arralist element, in order to check and print each address when needed.

The flow chart and code below, show how the ‘while’ loop worked, in order for the main menu to be repeated.




```
148        // while loop
149
150        while (true) {
151
152            // Main menu
153
154            System.out.println("\n =_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=");
155            System.out.println("\n \t MAIN MENU:");
156            System.out.println("\n 1) Renting out Parking Spot \n 2) Adding Parking Spot to Data base " +
157                    "\n 3) View Parking Spots \n 4) EXIT");
158            System.out.println("\n Please choose the number of your option!");
159            int option = keyboard.nextInt();
```

The flow chart and code below shows how I used the ‘for’ loop in order to satisfy the iteration functionality which my user needed.




```
49          // for loop
50           for (int i = 0; i <= CaliWeekDay.size(); i++) {
51
52              // Nested if
53              if ((adressimput - 1) <= CaliWeekDay.size()) {
54                  System.out.println("\n You choose address: " + CaliWeekDay.get(adressimput-1)
55                          + " as the parking spot being rented out");
56
57                  // Removing element from ArrayList
58                  CaliWeekDay.remove((adressimput - 1));
59              }
```

‘for’ loop further information: appendix 12 in “Add_Spot Class” Section.

7. Searching

Searching involves fetching some data stored in data structures like Arraylists, Hash Maps, etc. In order to satisfy my clients needs, I used linear search; which involves sequential searching for an element in the given data structure until either the element is found or the end of the structure is reached. (Singh, 2020)

I used this feature to search for the addresses every time my user needs to know the availability of a parking space. This search function will allow me to compare two inputs (addresses) and provide an output to my client, depending if the address was found or not.

```
133                 // Searching
134                 if ((adressimput - 1) <= CaliWeekEnds.size()) {
135                     System.out.println("\n You choose address: " + CaliWeekEnds.get(adressimput - 1) +
136                             " as the parking spot being rented out");
137                     CaliWeekEnds.remove((adressimput - 1));
138                 }
```

```
164                 // else - search not found
165             } else {
166                 System.out.println("\n ERROR! Your imput is not valid. Please enter the number that corresponds to your adress choice ");
167                 adressimput = keyboard.nextInt();
```

Searching further information: appendix 12 in “RentingOut_Spot Class” section.

8. Use of sentinels or flags

A sentinel or flag is a value that is usually used to terminate a loop or a recursive algorithm. (WeboPedia, 2020)

Firstly, I used loops with numerical conditions, which would terminate once a certain condition was met and the numerical value storing the condition changed. This was done in order to terminate a certain loop after a specific task was successfully performed and hence avoid repetition.

```
82          // While loop with numerical condition
83          int x = 0;
84          while(x == 0) {
85              System.out.println("\n Parking Spot Address: ");
86              adress = keyboard.nextLine();
```

```
93              // Condition MET
94              if(correct.equals("yes")) {
95
96                  x = x + 1;
```

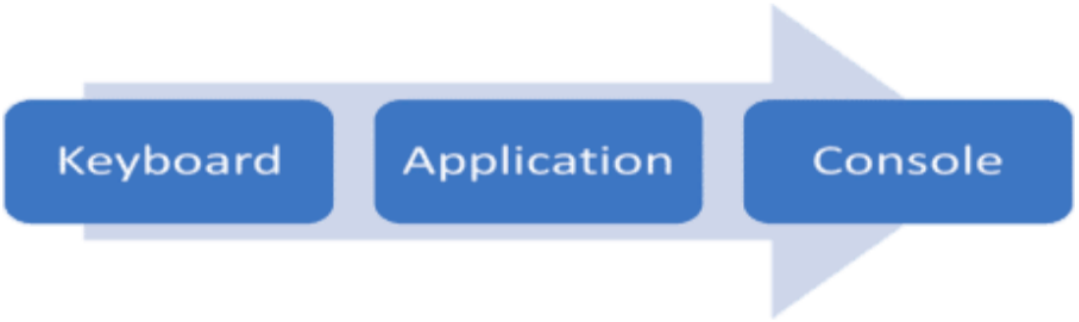
Further on, the “break” statement was used in order to terminate the ‘while’ loop which contained the Main Menu. This was done in order to terminate the actual program, once my user chose the “Exit” option.

```
370                 // Break Statement
371                 System.out.println("\n See you soon " + name + "!");
372
373                 break;
```

Sentinels and flags further information: appendix 12 in “Main and Add_Spot Class” Section.

9. File.io

Java file.io streams the data information given by the user inputs and utilizes file handling to sort out information via packages. As a result, user inputs and information can be used, sent and modeled throughout the program. (tutorialspoint, 2020)



This function allowed my user to input different data (name, address, etc) and then use such information in the different user-defined methods and classes. I did this by using a Scanner, where integers, doubles and Strings where collected, stored and further used in the program. This allowed me to create a functional product, suited and personalized to my user choices.

Some of the user inputs request along the program are shown below:

```
27          Scanner keyboard = new Scanner(System.in);
```

```
89              System.out.println("\n The parking spot address is: " + adress);
90              System.out.println("\n Is this address correct? 'yes' or 'no' ");
91              String correct = keyboard.next();
```

```
140         // Start of the program - Request name and Welcoming message
141
142         System.out.println("\n Hello! What is your name?");
143         String name = keyboard.next();
144         System.out.println("\n Hello! " + name + ", welcome to RentP! We will help" +
145                 " you to managed your parking spot company!");
```

```
159             // Using the Scanner to receive an input from the user
160             System.out.println("\n Please choose the number of your option!");
161             int option = keyboard.nextInt();
```

10. Additional Libraries

Libraries enable programmers to use powerful codes which have been developed in java and implement it in their program. (VertexAcademy, 2016)

The java.util.Scanner library was used to gather my user's inputs. With the Arraylist library, the list of addresses were created and managed, allowing my client to keep a track of such data. Finally, the file output/Input stream libraries were used in order to carry out the java serialization/deserialization function allowing my program to create and organize files. Libraries allowed me to enrich my program and as a result provide a more complex and functional product to my client.

```
3   //Imported libraries
4   import java.util.Scanner;
5   import java.util.ArrayList;
6   import java.io.FileOutputStream;
7   import java.io.IOException;
8   import java.io.ObjectOutputStream;
9   import java.io.FileInputStream;
10  import java.io.ObjectInputStream;
11  import java.io.IOException;
```

Java imported libraries further information: appendix 12 in "Main Class" Section.

List of References:

All references used for this criteria, can be found in appendix 11