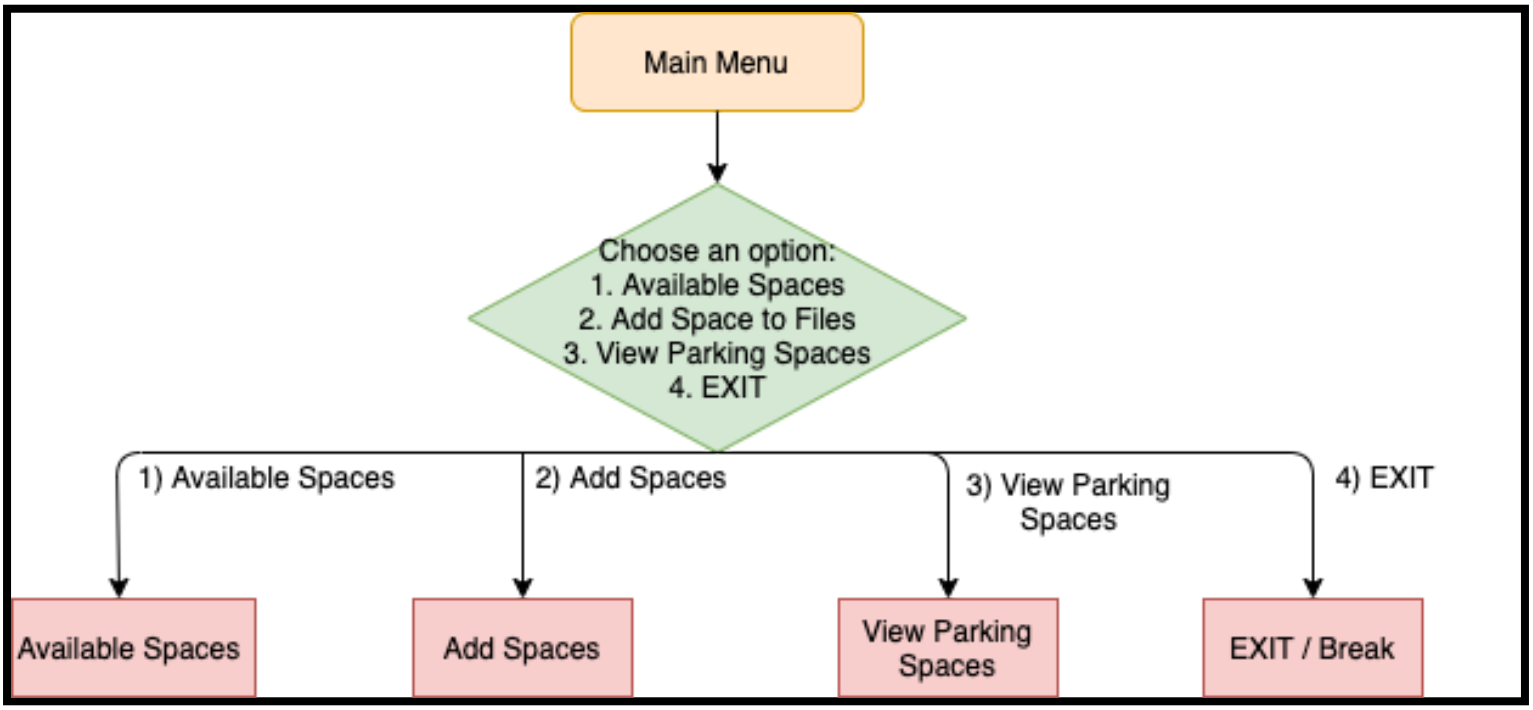
Criterion B: Design Overview

After initially meeting with my client, I began formatting the code structure, and it's key elements. The main aspect of this program is that my client must be able to perform three different tasks but they should all work together in order for the information to be constantly updated. The program should offer three main functions: One which enables my client to add parking spaces to his files, another one for renting out the parking spaces, and a final option which will quickly display all the available parking spaces.
As a result, I did my first flow chart (Figure 1), which shows the main menu that will appear to my client, every time he accesses the program.

Figure 1: Initial Flow Chart - Main Menu



In order to help my client visualize and understand how I was thinking about developing the 'Main Menu' function, I developed a pseudocode in order to show him this feature.
As seen on the pseudocode (Figure 2) , I used a 'while loop' for this Main menu feature. This feature will allow my client to either Add parking spaces, rent out parking spaces or view parking spaces as many times as he wants, without closing or starting the program again. The option 4 ("Exit") will break the loop and hence the program will be stopped. This pseudocode example, allowed me to provide a more visual and accurate representation of the program to my client.

Figure 2: Main Menu Pseudocode

```
EXIT = 0
output "Welcome to your parking spot organizer"

loop while EXIT = 0
  output "Choose an option. 1) Renting out parking spot 2) Adding Spot to database 3) View Parking Spots   4) Exit"

  input OPTION

  if OPTION == 1 then
   output "Renting Out Parking Spaces Class"

  else if OPTION == 2 then
    output "Adding Space Class"

  else if OPTION == 3 then
    output "Viewing Parking Spaces"

  else if OPTION == 4 then
    output "See you later"
    EXIT = 1
  end if
end loop
```

I used the IB Computer Science Pseudocode practice tool (Appendix 10), in order to run this program and show an example of the output to my client. (Figure 3)

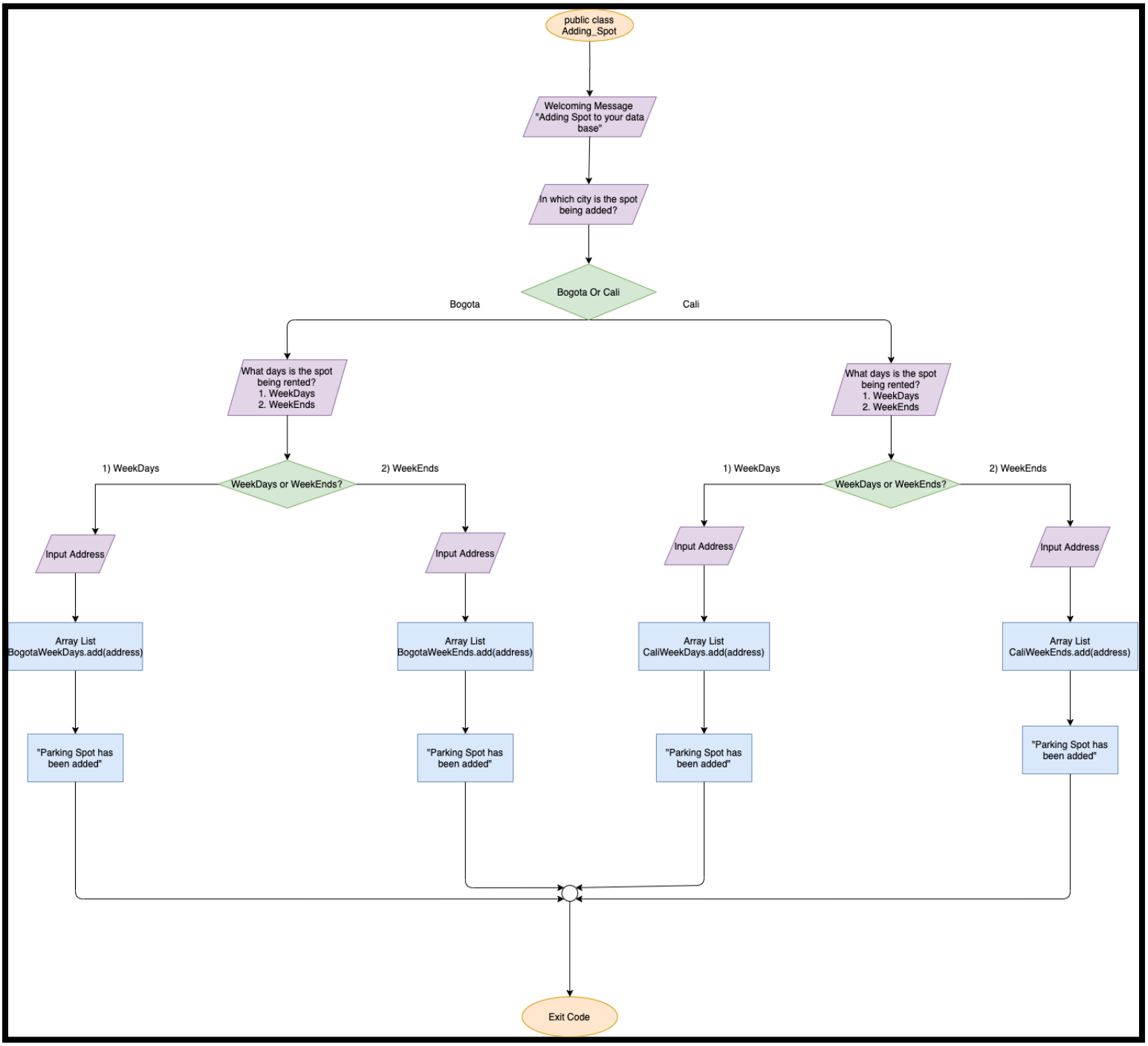
Figure 3: Pseudocode Output

```
EXIT = 0
output "Welcome to your parking spot organizer"

loop while EXIT = 0
   output "Choose an option. 1) Renting out parking spot 2) Adding Spot

   input OPTION

   if OPTION == 1 then
    output "Renting Out Parking Spaces Class"

   else if OPTION == 2 then
     output "Adding Space Class"

   else if OPTION == 3 then
     output "Viewing Parking Spaces"

   else if OPTION == 4 then
     output "See you later"
     EXIT = 1
   end if
end loop
```

```
Welcome to your parking spot organizer
Choose an option. 1) Renting out parking spot 2) Adding Spot to database 3) View Parking Spots 4) Exit
Renting Out Parking Spaces Class
Choose an option. 1) Renting out parking spot 2) Adding Spot to database 3) View Parking Spots 4) Exit
Adding Space Class
Choose an option. 1) Renting out parking spot 2) Adding Spot to database 3) View Parking Spots 4) Exit
Viewing Parking Spaces
Choose an option. 1) Renting out parking spot 2) Adding Spot to database 3) View Parking Spots 4) Exit
See you later
```

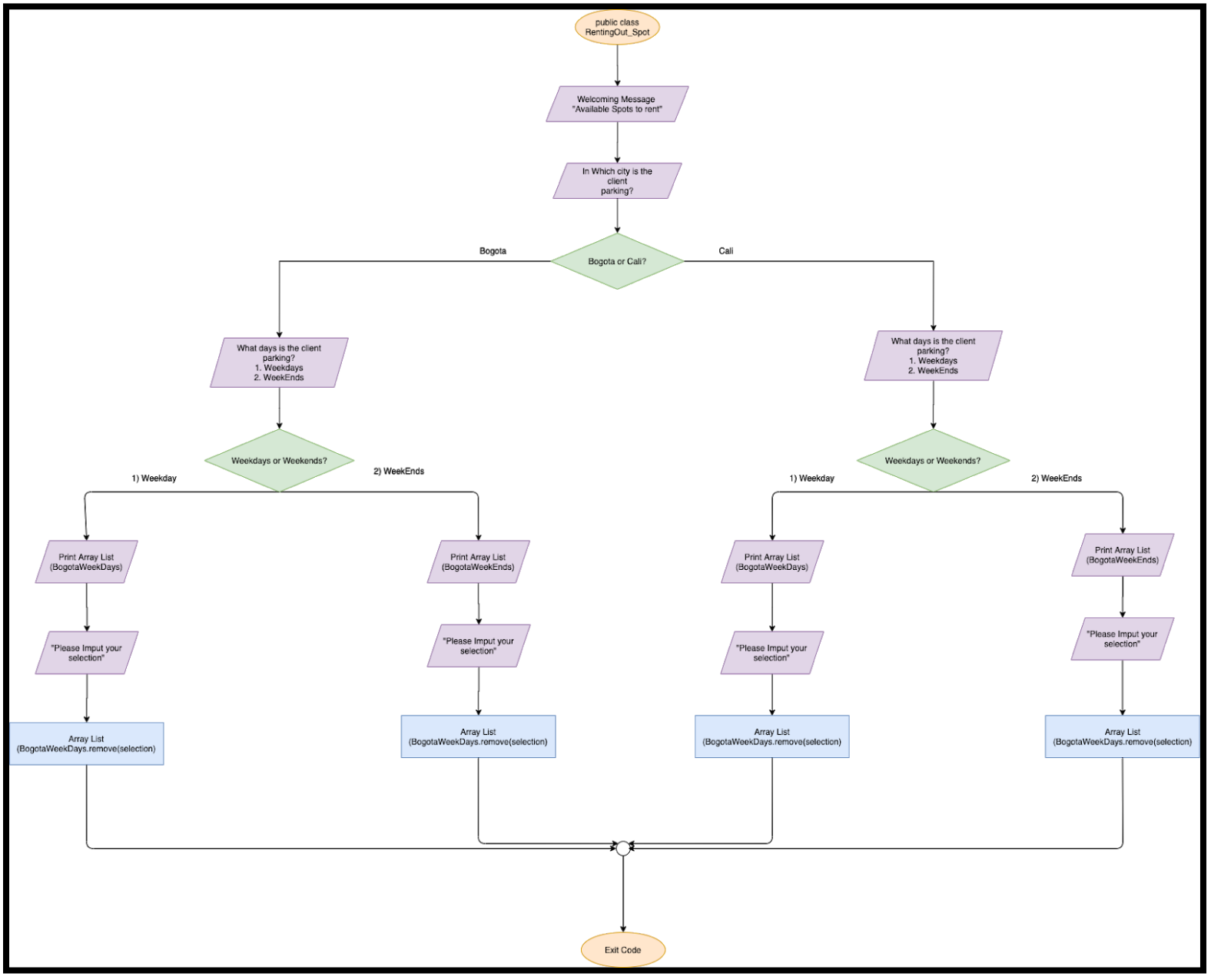
Consequently, I then knew that in order to fully understand and properly explain the code to my client, I had to do a flow chart and a plan for each individual class. This will allow me to explain how each class worked, what elements it had, and what code was required. Therefore I began by developing the “AddSpot to database” class. (Figure 4)

Figure 4: Adding_Spot Class Flow Chart



After developing the flow chart for the first class, I did the same process, for the “Renting Out Spot” Class (Figure 5). This way I will then be able to show each separate class to my client and see what opinions he has about the program and how could I make it better depending on his needs.

Figure 5: RentingOut_Spot Class Flow Chart



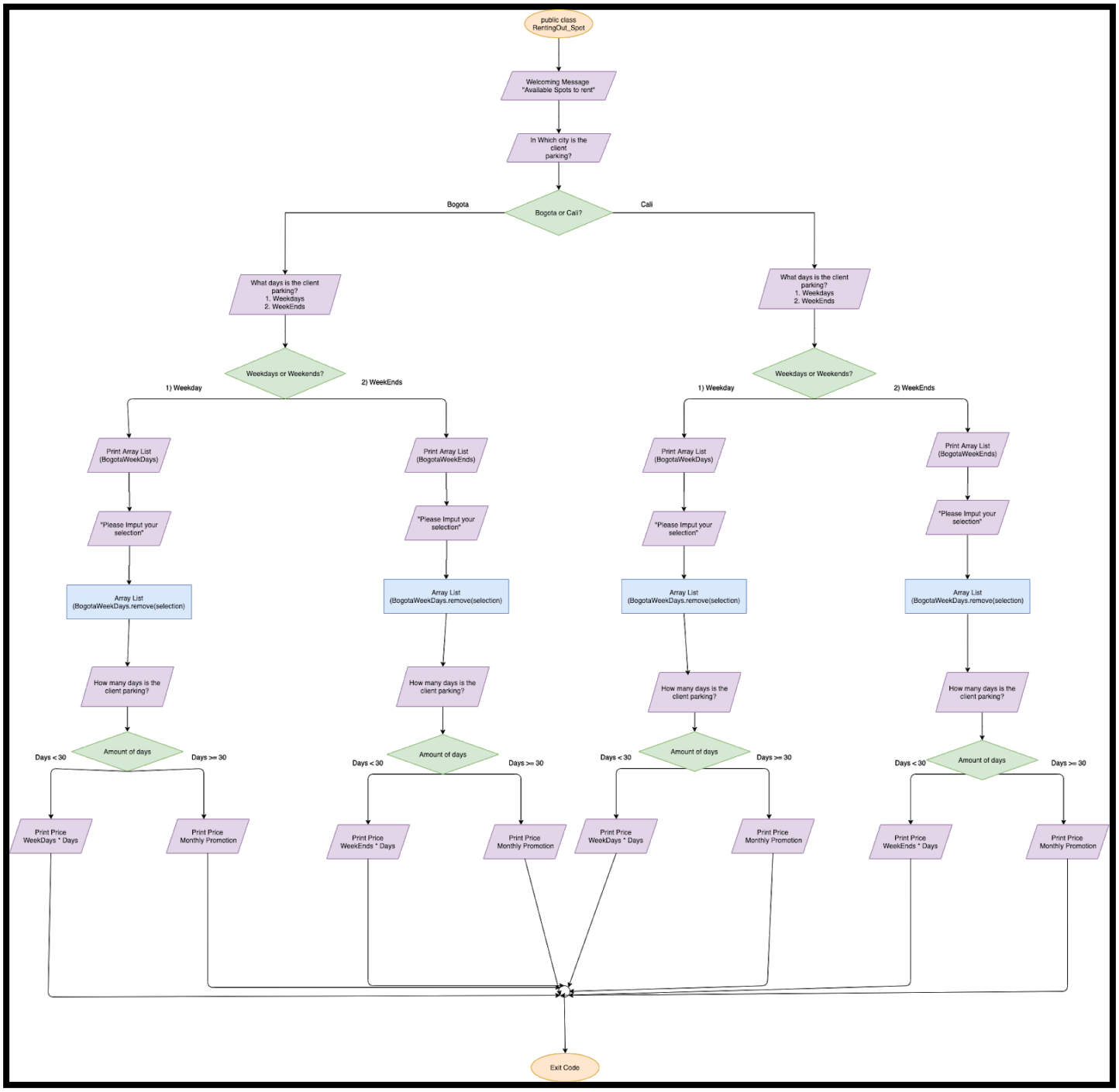
After showing these flow charts to my client, he told me that the general idea was very good and that he liked the way the structure of the program was developing. Nevertheless, he did give me some feedback, especially regarding the “Renting Out Spot” Class. He told me it will be a good idea to have a simple price calculator. (Appendix 3)
Below are some of the conditions:

1. The rent price for 1 day of parking during weekdays is: COP* 15,000
2. The rent price for 1 day of parking during the weekends is: COP 10,000
3. If the client chooses to rent the spot for 1 month, they will get a promotion and the total price for 1 month will be: COP 250,000

*(*COP = Colombian Pesos)*

After getting this information from my client, I developed an updated version of my flow chart. (Figure 6)

Figure 6: RentingOut_Spot Class extended features flow chart



Pseudocode:

Based on my clients feedback, I wrote this pseudocode (Figure 7) in order to show him a more realistic example of how the 'Renting Out' function will operate, depending on the different inputs and commands that he gave to the code.

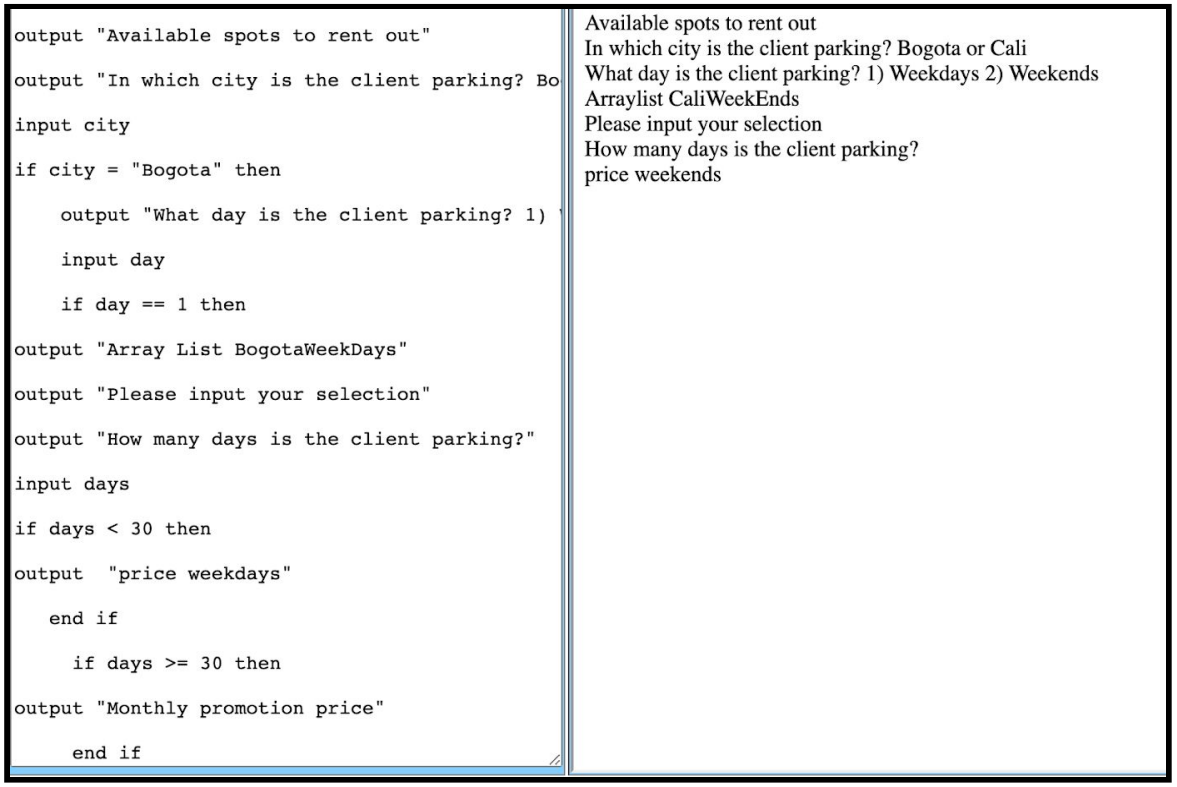
Figure 7: RentingOut_Spot class Pseudocode

```
output "Available spots to rent out"
output "In which city is the client parking? Bogota or Cali"
input CITY

if CITY = "Bogota" then
   output "What day is the client parking? 1) Weekdays 2) Weekends"
   input DAY

   if DAY == 1 then
          output "Array List BogotaWeekDays"
          output "Please input your selection"
          output "How many days is the client parking?"
          input DAYS

          if DAYS < 30 then
          output  "price weekdays"

         else if DAYS >= 30 then
              output "Monthly promotion price"
         end if

   else if DAY == 2 then
          output "Array List BogotaWeekEnds"
          output "Please input your selection"
          output "How many days is the client parking?"
          input DAYS

          if DAYS < 30 then
          output  "price weekends"

        else if DAYS >= 30 then
            output "Monthly promotion price
        end if
    end if

else if CITY = "Cali" then
          output "What day is the client parking? 1) Weekdays 2) Weekends"
          input DAY

    if DAY == 1 then
          output "Arraylist CaliWeekDays"
          output "Please input your address selection"
          output "How many days is the client parking?"
          input DAYS

          if DAYS < 30 then
```

```
            output  "price weekdays"

        else if DAYS >= 30 then
            output "Monthly promotion price"
        end if

   else if DAY == 2 then
        output "Arraylist CaliWeekEnds"
        output "Please input your selection"
        output "How many days is the client parking?"
        input DAYS

       if days < 30 then
          output  "price weekends"

       else if DAYS >= 30 then
           output "Monthly promotion price"
       end if
    end if
end if
```

I used the IB Computer Science Pseudocode practice tool (Appendix 10), in order to run this brief program and show a more visual and user friendly example to my client. (Figure 8)

Figure 8: Pseudocode Output

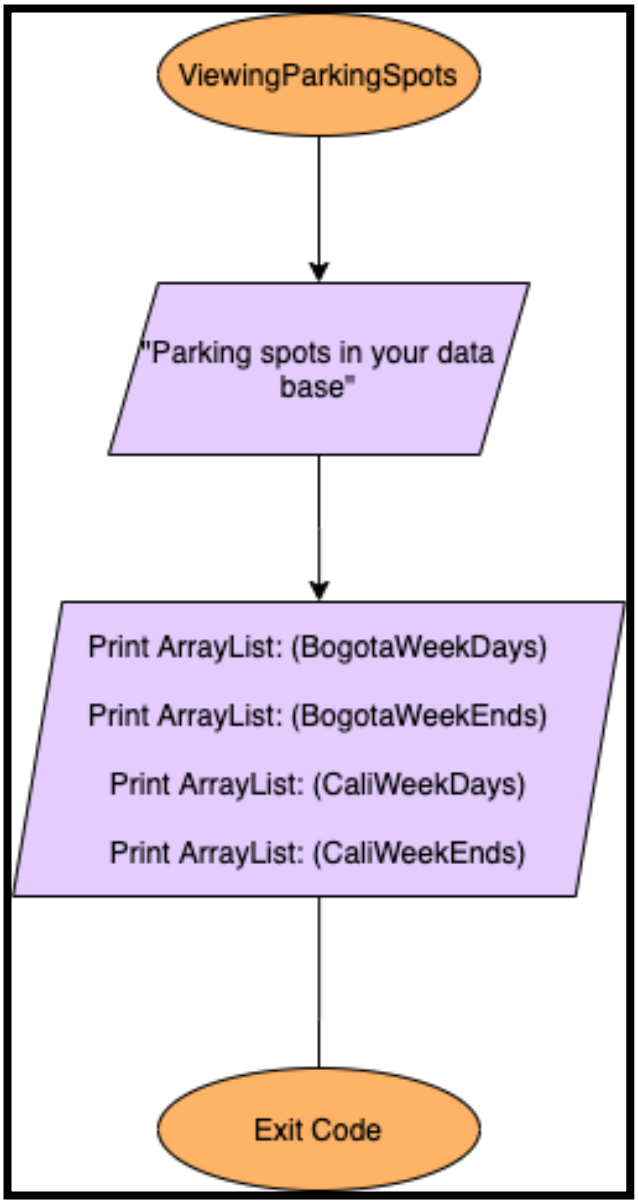


Although this pseudocode program didn't show all the functionalities which the real program will have, it allowed me to show my client a more realistic perspective, where he could actually see and understand how the program would perform and the main functions it had.

Finally, the “View Parking Spots” method will only work as a viewing function, allowing my user to only view the parking spaces in his business, without having to either add or remove any of the parking spaces.

This will be a visual representation as a flow chart for this function. (Figure 9)

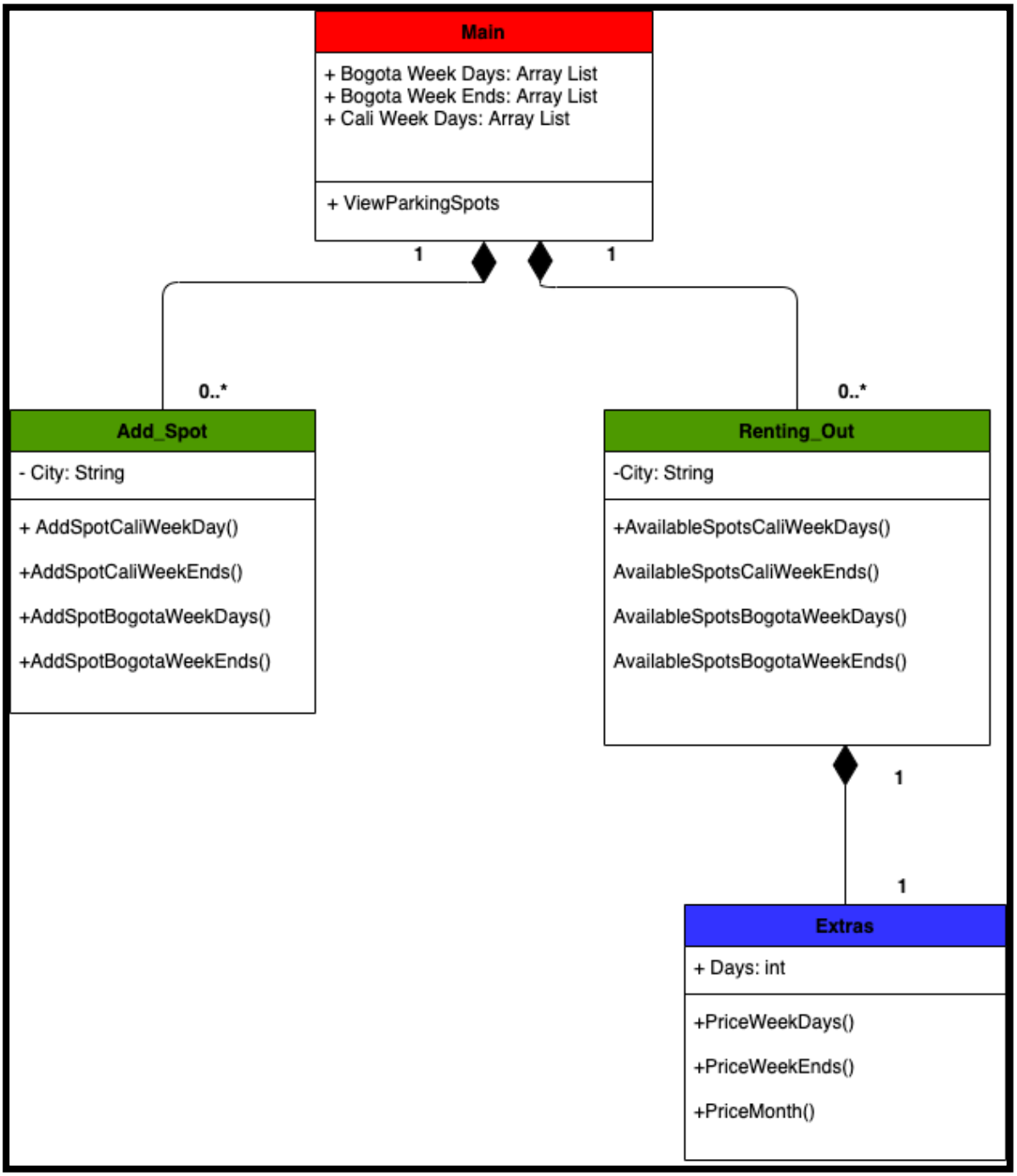
Figure 9: ViewParking_Spots Class Flow Chart



UML Diagram

Finally, in order to represent and show the structure of my complete program, I used a UML diagram (Figure 10). With this diagram, I was able to clearly show my client (Appendix 5) each variable and method which each class will have, as well as the dependency and the composition between classes, which are represented with the numbers and composition arrows. This will allow me to understand and to clearly explain to my client the final version of the planning for the code, where he could clearly identify each class, understand which classes depend on the others and how they are related between them.

Figure 10: UML Diagram



Test Plan:

Success Criteria	Test Type	Nature of Test	Example
The client will be able to access three different main options. One for adding spots, another for renting out spots, and another one for viewing the spots in his files.	Unit Testing: It focuses on the smallest unit of software design. In this we test an individual unit or group of inter related units, in order to check if functions such as loops are working properly.	The program will display three different options and will give the opportunity to the user to choose between them, depending on the task he will like to perform.	Run the code, and choose each option in order to check they all work fine.
The program has a menu, which allows the users to choose the options and exit whenever he wants.	Unit Testing: Will check if the methods/classes called by each option are triggered properly.	A main menu will be displayed where depending on the user choice he will be directed to either the adding spot function, the renting out spot function or the view spots function. There will also be an exit option, which will terminate the program when the user indicates to do so.	Running the code multiple times, choosing all the options to check they work properly. Then, the exit option will be tested, in order to check if the program terminates successfully when the user indicated so.
When adding a spot, the program will ask for the city of the spot and the available days for rental (weekday or weekends) to store each address in its specific place.	Integration Testing: Integration testing is when a group of components are combined to produce a certain output. In this case, the different inputs (city and day) will be combined to calculate and output to the user.	The user will be able to choose between the two cities where the business operates (Cali or Bogota), as well as the day (weekday or weekend) when the spot will be available, and this one should be stored in its specific arraylist.	Run the code multiple times and choose different cities and different days in order to check that each address is properly stored in a specific place depending on its specifications (City and day).
Every time a spot is added, the Arraylist storing the addresses will increase and the new address will be stored. On the other hand, whenever a spot is rented, the address will disappear from the specific Arraylist, in order for my client to keep track and know exactly which spots are currently	White Box Testing: It is used for verification. In this we focus on internal mechanisms i.e. how the output is achieved? In this case we will test the algorithm/mechanisms which adds and removes the addresses to the lists, depending on the user requirements and specifications.	When my client is adding a spot the arraylist which is storing the parking spots will be extended and the spot will be added to the list. On the other hand, when my client rents out a spot, the address will be removed from the arraylist storing it and won't be available any more.	Run the code multiple times, choosing both the 'add spot' and 'rent spot' function in order to check that the spots are added to the list when the add spot class is triggered and on the other hand check that the spots are properly removed

available and dismiss those which have been rented out.			from the list when the ‘renting out spot’ is used.
When renting out a spot, the program will ask for the city where the client needs to park and the day (weekday or weekend) to show the available spots considering those specific conditions (city and day).	Integration Testing: Will test if the combination of the city and days input works properly in order to provide a coherent output to the user.	Once the user inputs the day and city, the program will display the list of the available parking spots which fit the specific city and day specifications required.	Run the code multiple times, choosing different cities and days in order to check that no parking spots are repeated and that each spot is displayed on the specific place depending on the requirements
When renting out a spot, the program will ask the number of days, in order to calculate the prices which should be charged to the customer. (As suggested by the client, (see appendix 1)	Integration Testing: Will test if the price and days unit (input) combines properly and provides a correct output to the user.	Onced my customer has rented a certain parking spot, he will be asked for the amount of days his client will be using the spot, and the program will calculate the price, depending on factors such as the day (weekend or weekdays) and the amount of days. (if it goes over 30 days it should give the monthly promotion)	Run the code multiple times, renting out different spots, at different days and for different amounts of days in order to check if the arithmetic calculations of the program are carried out correctly.
The program will run in an infinite loop (While(true)) and therefore will allow the users to perform multiple tasks without ending or closing the program. Whenever the user wants to exit, an option will be provided and the program will break/terminate.	Unit Testing: Will test if the While(true) loop works properly and terminates when indicated by the user.	The main menu, containing the Add spot, renting out spot and view spot functions will be repeated infinitely, until the user chooses the exit option and the program is terminated.	Run the code multiple times, choosing different options and functions and check if the Main menu actually runs in a loop and options can be chosen as many times as the user wants.
The code meets the coding complexity requirements (see appendix 2), to ensure a	Regression Testing: Every time a new module is added leads to changes in program. This type of testing makes	The code must have at least 10 of the elements listed on the appendix 2, in order to meet the IB	Go over the complexity list and the code, and check if there are minimum

sufficient level of detail, functionality, and complexity, for the client to have a quality end product that meets his requirements and solves his problem.	sure that the whole component works properly even after adding components to the complete program. This will allow us to check that as the complexity requirements are met, they also work properly when combined and the program runs without any errors.	requirements and provide a quality final product to my client.	10 elements which are successfully included in the program.
The code uses imported libraries from java that can enrich the user experience. This might include Array List, File Output Stream, and Scanners.	Unit Testing: Will test that each library (unit) works properly and it is adequate added and used in the program.	In order to have a complex and enrich code, it must use some of the imported libraries from java, in order to create new functions which can complement the code.	Go over the code and check how many java imported libraries are being used and if they are actually enriching and providing value to the code and the program.
The code uses java serialization, in order to store each Arrallist containing the addresses on organized files, keep them updated and use them every time the program starts. This will allow my client to close the program at the end of every workday, and the next day when he starts the program he will have all of the addresses updated and organized thanks to the files which are keeping track of the data. Serialization will also help my user keep its data safe in case the program crashes, the data will still be kept safe and updated within the files created.	Beta Testing: The beta test is conducted at the customer sites by the end-user of the software and in a real time environment. I will give the final product to my client, allow him to use it for a time and then receive feedback from him in order to finalize adjustments and improvements. (Appendix 8)	Java serialization code is used to store and keep the addresses organized in their specific files. This will also allow my client to keep track of the addresses each time the program is finalized and started. My user will test the entire program, including this specific option and will check that all the addresses are properly stored and updated in each of the files.	The program will be completely run and used by my client, where he will be able to use and check all of it's features, including the jAva Serialization, where files are created and even if the program is terminated, data will never be lost and it will always remain updated.