Our project is meant to take wiggle wifi files and use their data to make a few operations. For these operations, we had to keep making constant changes in our code and project planning, every step of the way. In this stage of our project, we have a GUI added to our code, and through the GUI we will be able to make a lot of operations on our data from the files we have.

Before we explain about the GUI, we will ex[lain about our code, the different packages and classes in each, and also the connections between them, and the decision behind writing them.

At the very basic project, **stage #0**, our code had to have two main functionalities:

1. Transfer the data from Wigle wifi files in the original format to a “unified CSV” format, as demonstrated below in this table:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Signal | Frequency | MAC | SSID | #Wifi | Alt | Lon | Lat | ID | Time |

The last four fields repeat themselves – depending on the number

of access points in the specific row.

With this format, we will proceed working in later stages of our project.

For that need, we wrote a new object class called “Row”, matching the fields of the above table, to make it easier and more practical to work with in a continuous project.

In addition to that, we wrote the “**Csv**” class that held all functions that were responsible for passing the information to a unified formatted csv that held all information from the original files with the next requirements: up to 10 wifi’s per row in the unified file, a row is a legal row (has all data in it and a wifi access point), every row is indicated by 6 fields; the first six that are: ID, Time, location data and number of wifi’s in that row – each row that doesn’t appear by exactly the same data in these six fields is a new row in the unified table.

For the need of giving accurate time information and being able to use it as a formatted time field and not only a string, we created the “**Time**” class that had a few functions that made it easier and more suitable for the needs of our project.

Moreover, we wrote a “**Wifi**” class, that defined a new object of a wifi type, that was meant to hold the relevant information for each access point we had, and we used this object to simplify the way we can refer to a wifi in the row object, which now every row object will have an arraylist of up to 10 wifis in it. As mentioned above in the table, the relevant fields for a wifi object are: SSID (name of access point), MAC (unique identifier), Frequency, and Signal. The Wifi class also contains relevant functions to work with a wifi object.

1. Write our data to a KML file, that can be viewed in Google Earth. At this stage we just got familiar with the KML data files and their format and for that we wrote a “**Kml**” class, that had functions that were responsible on passing a “Unified CSV” file to a KML format file using simple printing methods and not any other tools. In this class as you can see in the code and the Javadoc documentation, there were a few other requests such as not showing a mac twice on the map (showing only the strongest appearance of every access point – by signal), therefore we wrote functions that found the strongest appearance for each mac in the unified CSV file, and wrote it in a KML format to match the requirements.

At this stage of our project we had only one package, a general package of our project.

In the next phase of our project, **stage #1,** we started this Github repository which will hold all our versions of our code throughout the commits we have made. We also had to improve our “Kml” class so that it used a Java library that will help us work more efficiently with KML files and data. For that we wrote the new class: “Kml\_by\_lib” using the JAK library, which we found the best since it has more functionalities than the other two options that exist and also since it is the newest one, it has the best chance of being updated so that we can use it for a long time. With the JAK library we had to download another library, which you can find in our “lib” folder in the source files. One of the requirements in the new formatted KML file that our code produces is a timeline that you will be able to see in the downloaded version of Google Earth.

Also, you will be able to find some Wigle files in our Input files folder, a unified CSV file generated by our code in the required format and a KML file generated by our code.

In the next stage of our program, **stage #2**, we already had to start thinking about the structure of our projects and design a better conceptual model for our code. This time we were asked to write two algorithms that will run on the data in a CSV unified file and make certain calculations.  
The files we had to match our code to run were unified CSV files. Here is an explanation about each one of the algorithms, it’s input and output as requested and a file that holds an explanation to making it work. Of course, using the GUI there is no need to run the code through a java IDE and you will not need this next document. (Please note that this document is written in Hebrew: )



In this report you can also find a comparison between out calculations and the calculations given to us by our course professor, so we could compare our results.

The algorithms:

1. The first algorithm has the role of calculating an access’s point locations, with the information given about it in the unified CSV formatted file. Using the different appearances of an access’s point in a unified CSV file, we can calculate its location using the signal of the access point as a weight to the most significant appearances of our access point.

In this algorithm we refer to an access point by its mac, since it is the only information that is unique to every access point. Using that we find the most significant appearances and calculate the location. Please note that our code is flexible, and uses the number of appearances it can find, up to the four most significant if it finds more than four appearances.

This algorithm is calculated for every access point (earlier in this document we referred to an access point as wifi) in the unified CSV formatted file. Later on, in the GUI it will be calculated for a given mac the user inserts to the matching text field in the GUI.

1. The second algorithm is meant to calculate the user location, using a CSV formatted file without the location data as the requests, referring to each row as a different user scan (that meaning that we refer to a row abject as if the user has turned on the wifi option on his phone and is now scanning possible wifis to use). We then look for rows that are similar to the row with the missing location data from the user another CSV formatted file which has location data in its rows, to use them as weights – the significance of the rows is now calculated by the appearance of the quantity of access points appearing in a row that are also in the user scan and their signals – the more similar the signals and access points in a row to the user scan – the higher its significance. Using these rows, after choosing the best (again, flexible like in the first algorithm) and the locations given in them – we can now use the first algorithm to calculate the user’s location.

For this operation, we had to re-consider the way our code is designed and divided to packages.

We now had more functionalities we had to do with our data, and to make the calculations of our algorithms simpler, we decided to create and use some objects which will simplify the process.

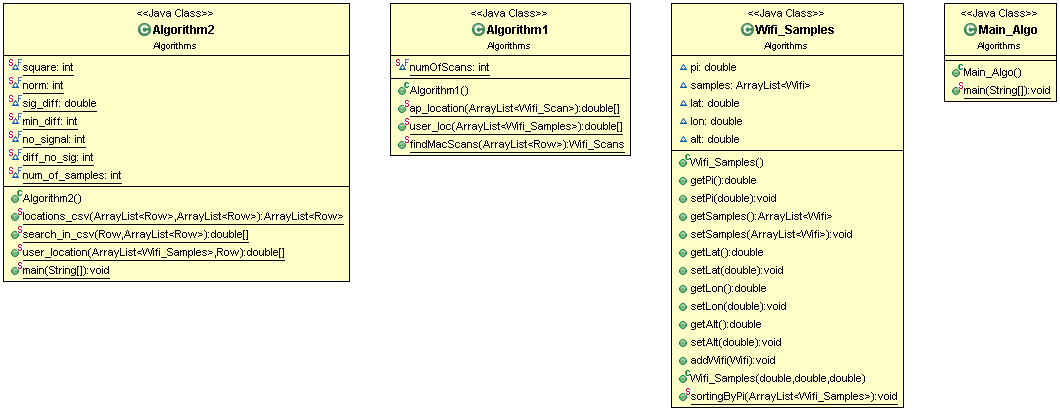
As the calculations mainly used row object information, we had to make an abject to obtain temporarily the data we need from each row to facilitate the calculations and prevent from using the whole rows and copying them to avoid the unnecessary use of memory.

For that purpose, we wrote more classes to define another type of objects we need, and write functions to help us use them, and also we divided our classes into packages to make more sense.

Now in our project we had four different packages, each containing classes that are relevant to it and the project as a whole. The packages in our project were:

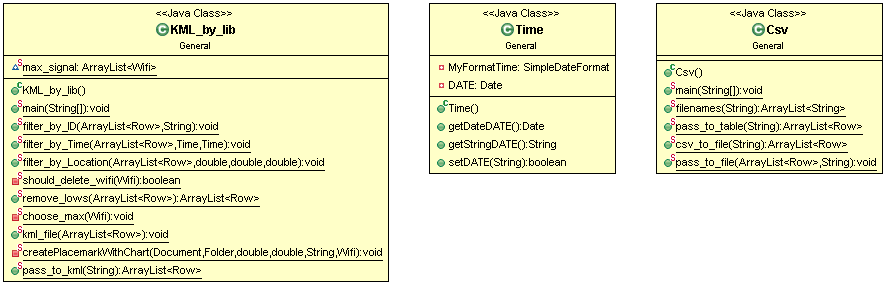
* **Algorithms** – In this package the classes are meant to contain the code for executing the two algorithms explained above, making the whole process: acquiring the data needed, sorting it, calculating the algorithm itself and other necessities to have a good final result. There are four classes in this project: Algorithm1, Algorithm2, Wifi\_Samples and Main\_Algo.
  + - Algorithm1 – Has functions that calculate the first algorithm explained above. Functions: ap\_location for calculating the access point’s location, user\_loc for calculating a user location matching algorithm 2 input and output and findMacScans which searches every mac to calculate the access's point location (matching this stage of the project, not the GUI).
    - Algorithm2 – Has functions that help prepare calculation for the second algorithm explained above (the calculation function is in Algorithm1 class, matching the terminology).
    - Wifi\_Samples – Defined a new object
    - Main\_Algo – Helps to run the code, just to simplify things for the person who uses this code. For each of the algorithms the user can execute, there is a certain format for the request to type in the console, mentioned in the Javadoc documentation in the class. A switch loop differentiates at first if the user wants to execute algorithm 1 or 2, accordingly.

**Algorithm Class Diagram**



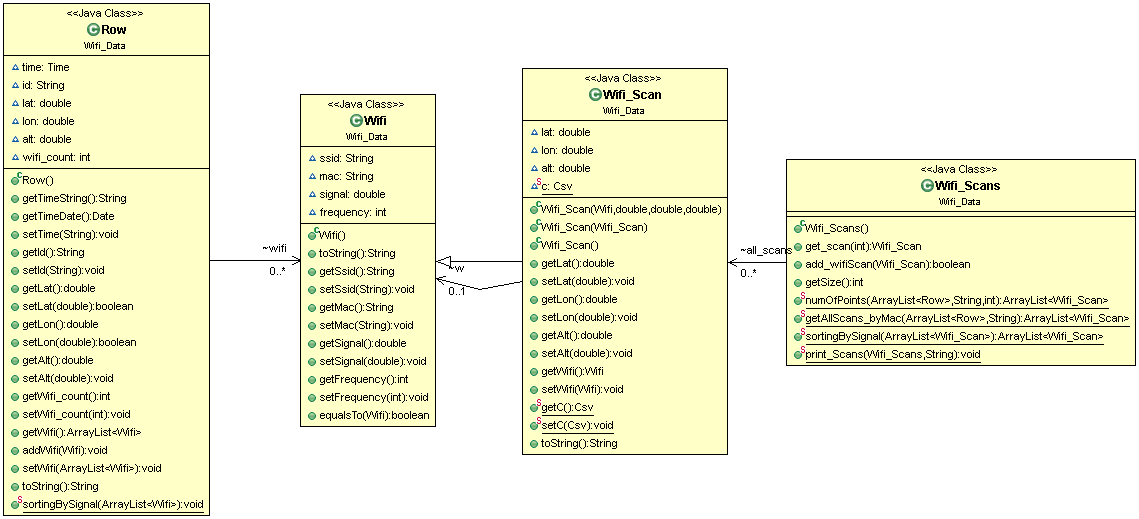
* **General** – This package was meant to have the most general and most used classes in the project up to this stage. That means that the main operations of our project are made in this package, using the classes it obtains. Therefore, the classes in this package are: Csv, KML\_by\_lib and Time. Those classes are explained above in this document, at **stage #0**: Csv class, KML\_by\_lib class and Time class (Row class was moved).

**General Class Diagram**



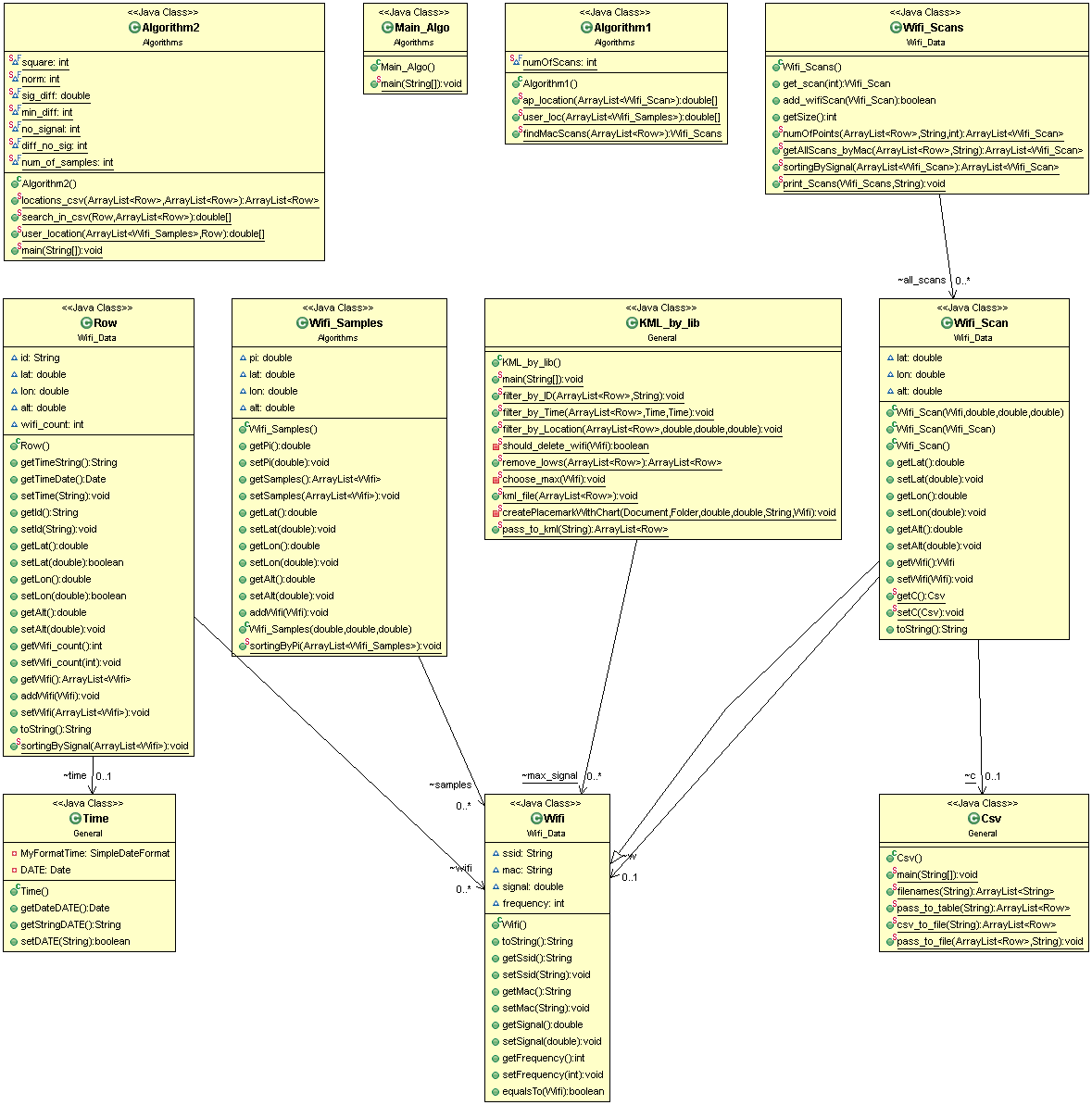
* **Wifi\_Data** – Contains all the classes that worked mainly with wifi data, that means access point information. We noted that a lot of our code was meant to work with this information, make calculations on it and save data temporarily so we wrote 3 classes in this package to help us work more efficiently.
  + - Row – This class, as mentioned above and explained in **stage #0**, was moved to this package to match more accurately the new code terminology.
    - Wifi – kjh
    - Wifi\_Scan – kh
    - Wifi\_Scans –

**Wifi\_Data Class Diagram**



* **Tests** – Contains tests for our code for important functions in classes. Please note that this package is not up to date and further testing has to be coded for our project. For now we have the next tests:
  + - CsvTest
    - Kml\_by\_libTest
    - RowTest
    - TimeTest
    - WifiTest

**Project Diagram (without GUI)**

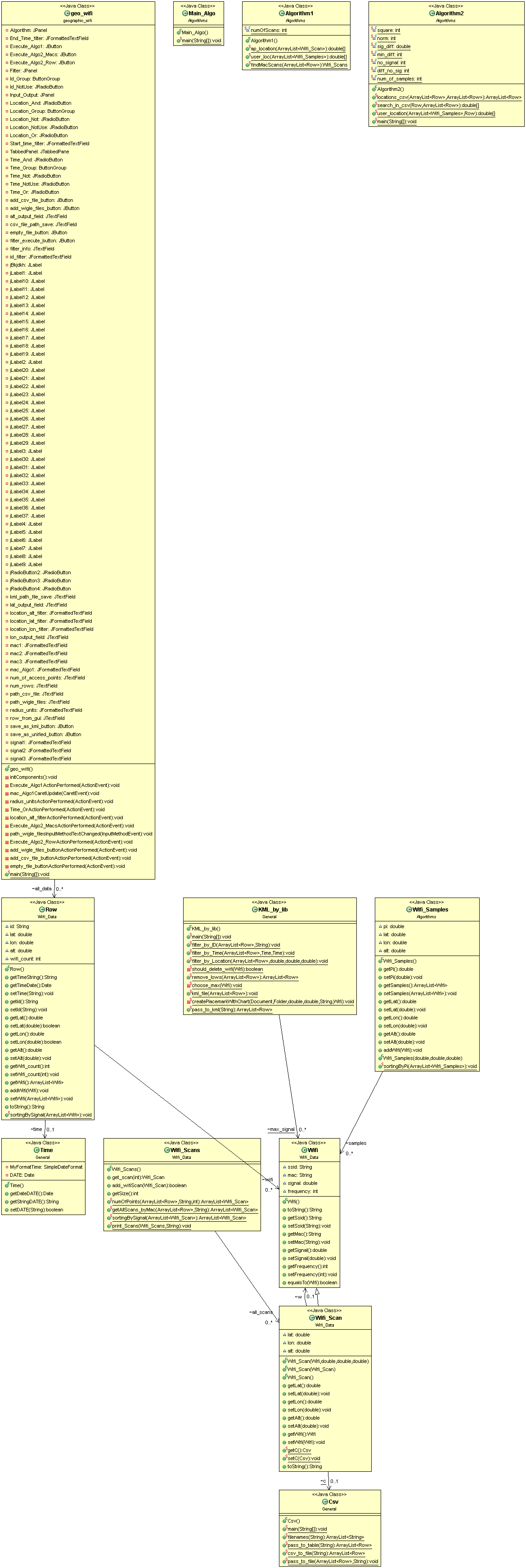


In **stage #3** of our project, we made a further step and turned this IDE project into a GUI, in which the user can run only the class referring to our new GUI and make a whole set of operations using our code. This stage was one of the most time-consuming to plan, and even more time-consuming to code. At this point of our project we were required to write this documentation, that includes a class-diagram to explain our code, the packages and classes in it and the relationships between them. Up until now we have done it whist explaining every stage of our project and every package and class and the decision to write it. To each class we have above (of course, for Test class this isn’t a real necessity, the names indicate the functionality and the relationship with the rest of the code and therefore we haven’t added it’s diagram to this document) we have added a UML diagram explaining the class, and finally a UML diagram of the whole project, explaining the relationships between our packages.

**GUI:**

* Geographic\_wifi package –
  + Geo\_wifi class -

Here is a diagram of the

**Project Diagram with GUI**