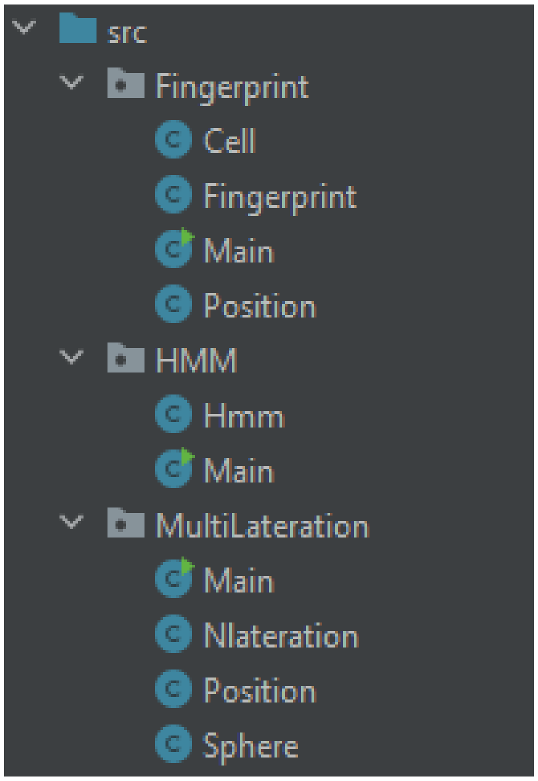
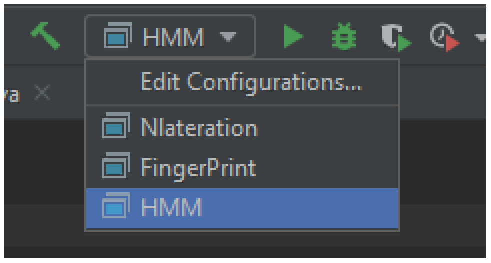
Positioning Systems TECHNIQUES

Project Structure :

For each problem we created a package that contains the classes we used in order to solve it, inside every class that has the name of the problem (exp : Nlateration), there's a void run() method that we will call inside the the **public static void** main() function of every Main class. and this run() method will execute the algorithms we used to solve the corresponding problem and return the result to the Main CLass if needed.

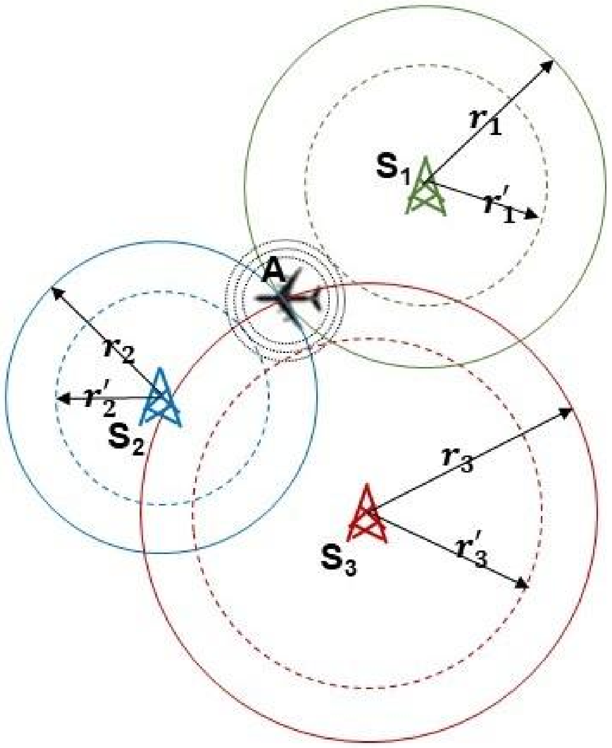


"If you import the project using **IntiliJ Idea** , you will be able to see 3 configurations for the 3 problems so we can switch between them and execute them easily."

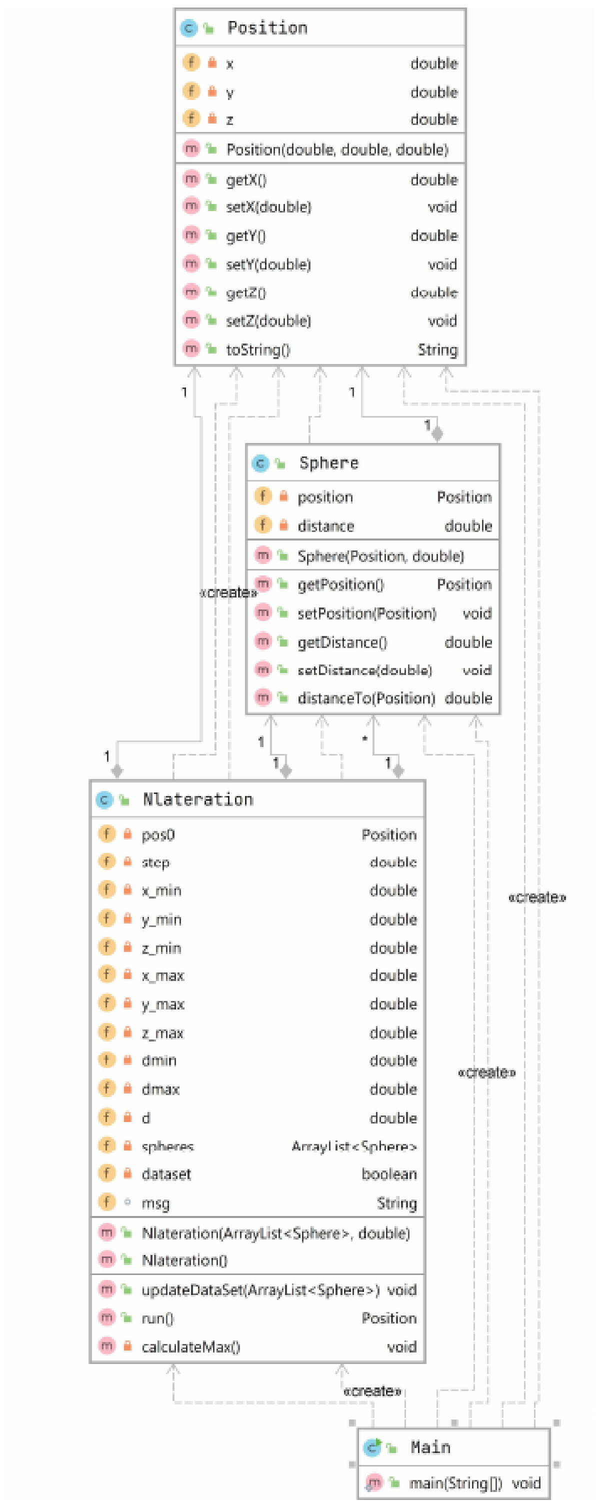


MultiLateration

- - - - - - - - - - - - - - - X

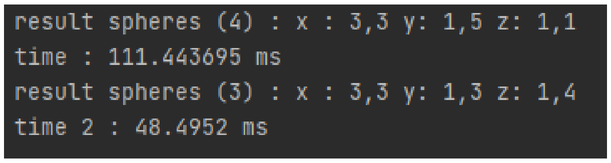


For this problem, we used 3 classes(the Main Class is in every package thats why we don't count it). the first one called position which we will use to store the x,y,z of the spheres (stations / satellites) and also for the result(predicted position). the second class is "Sphere" Class, this one will store the details of the the stations. the 3rd Class is "Nlateration", it will recieve an array list of spheres and the value of the step, using these data it will calculate the minx,y,z and maxx,y,z using calculateMax() method(we had to initialize the min values with Integer.MAX_VALUE first). with these min and max value the run() method will execute the Nlateration algorithm which will try to find the x,y,z values that gives the minimum distance to the MT. at the end it will return the predicted/calculated position(x,y,z) of the mobile terminal in form of an instance of Position(x,y,z). and finally , this latter will get displayed in the Main class's main() method.



Multi Lateration
CLASS DIAGRAM

RESULT :

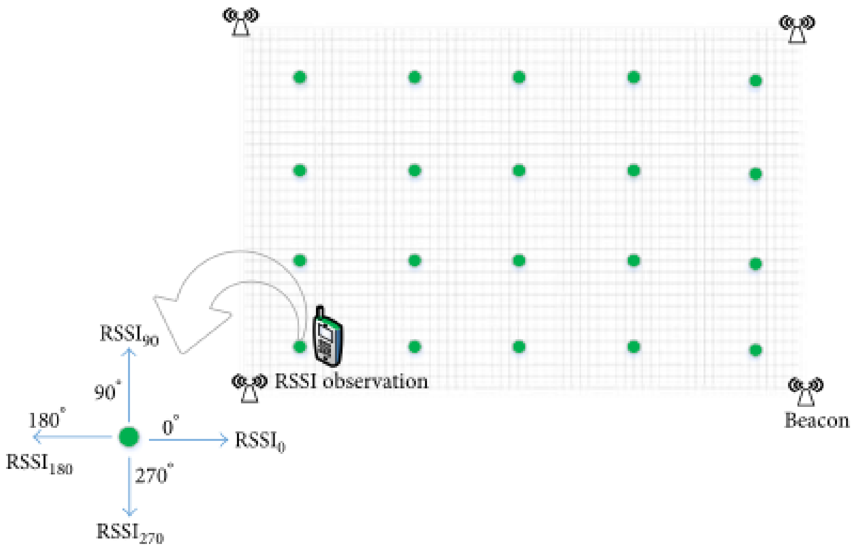


```
result spheres (4) : x : 3,3 y: 1,5 z: 1,1
time : 111.443695 ms
result spheres (3) : x : 3,3 y: 1,3 z: 1,4
time 2 : 48.4952 ms
```

FingerPrint

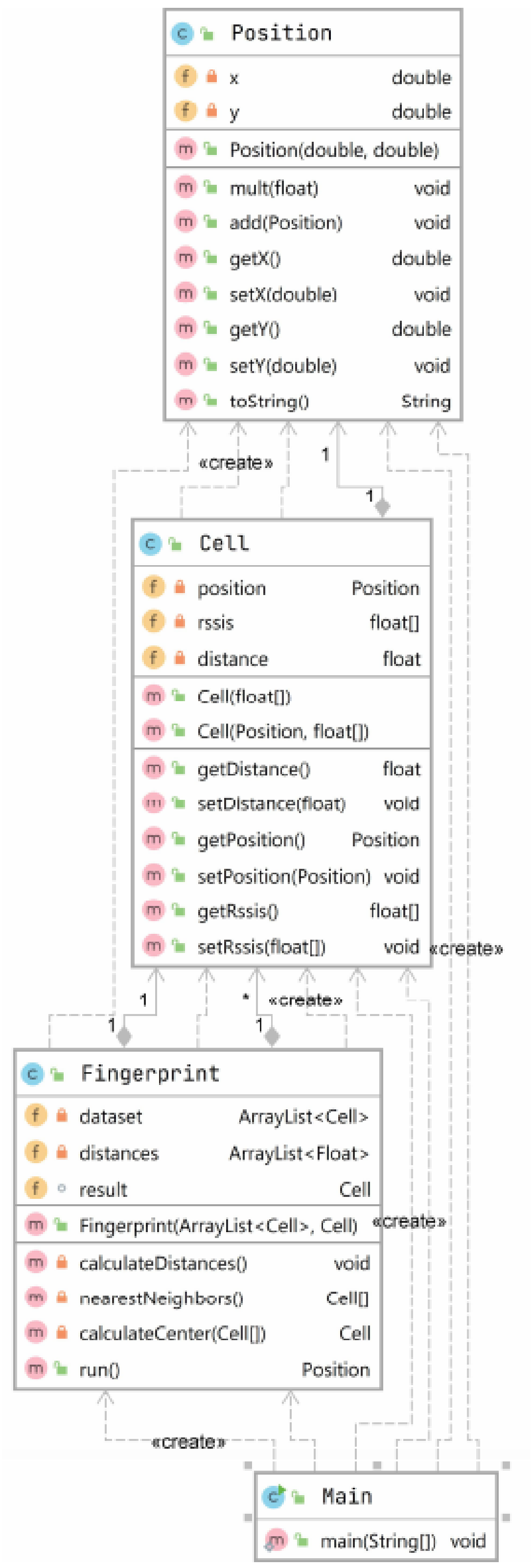
- - - - - - - - - - X

For this problem, we used 3 classes. the first one called Position, it is the same as in the N lateration but it has only x,y because we are working on 2d.
the second class is Cell, it will store the position of the calibration points and the RSSI information that are related to it along with the distance attribute which will store the calculated distance basing on the the difference between the rssi of the calibration points and the mobile terminal. the last class is “FingerPrint”, it will receive the dataset of the calibration points and the mobile terminal information(result attribute) with these information it will calculate the distances between the calibration points and the mobile terminal using calcultaDistances() function which will be used to get the kNearestNeighbors of the MT (k = 4) using nearestNeighbors() method, with the position of the 4 nearest neighbors the calculateCenter(Cell[]) method will compute their barry center which is the position of our Mobile Terminal. finally, we the run() method that will organize the calls of the previous method and return the calculated Position. **For the BaryCenter** we calculated it by: for each cell we multiply the position with 1/(1+ the distance of the current cell divided by every distance of the other cells in the array of neighbors) and then we add them together to get the final result.

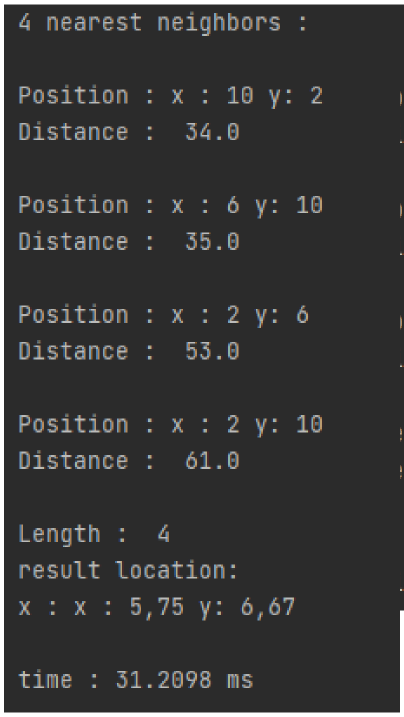


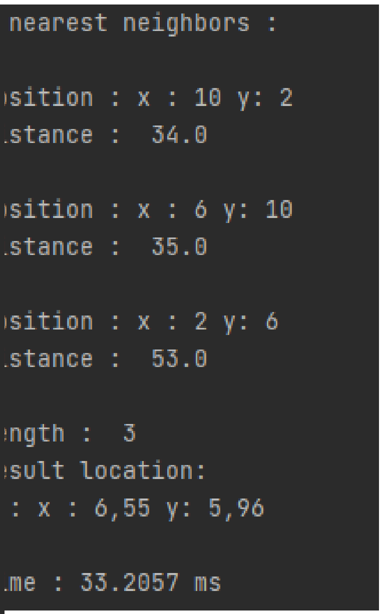
FingePrint

CLASS DIAGRAM



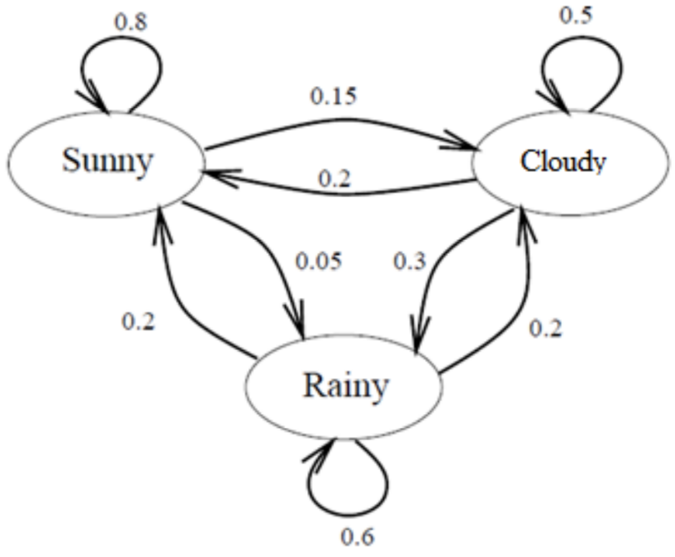
RESULT :


```
4 nearest neighbors :

Position : x : 10 y: 2
Distance :  34.0

Position : x : 6 y: 10
Distance :  35.0

Position : x : 2 y: 6
Distance :  53.0

Position : x : 2 y: 10
Distance :  61.0

Length :  4
result location:
x : x : 5,75 y: 6,67

time : 31.2098 ms
```

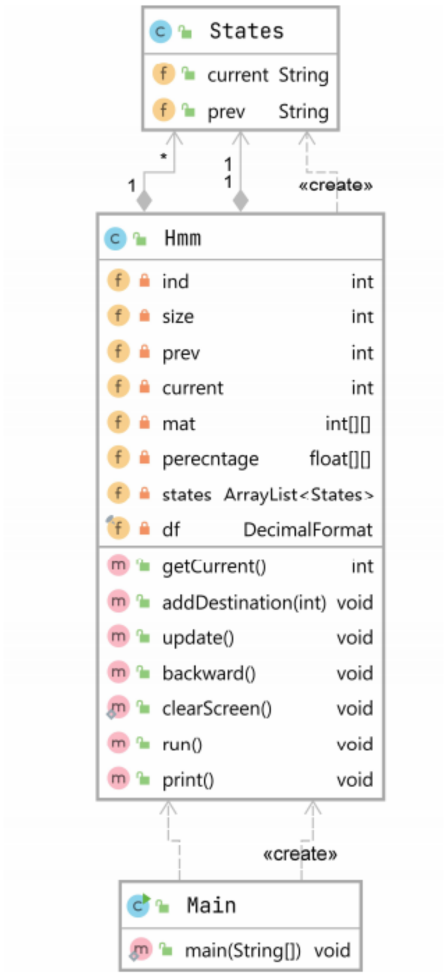

```
 nearest neighbors :

sition : x : 10 y: 2
stance :  34.0

sition : x : 6 y: 10
stance :  35.0

sition : x : 2 y: 6
stance :  53.0

ngth :  3
sult location:
: x : 6,55 y: 5,96

me : 33.2057 ms
```

Hidden Markov Model

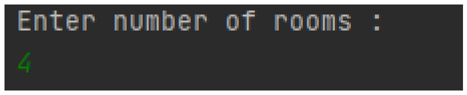
For this problem, we used two classe, the first one is called State, we used it to store the history of the user so he can go backward .the second one is called Hmm; it has : an attribute called size that will define how many fields we have. two attributes “prev” and “current” that will store the last visited field and the current one and an arraylist of States that we used to implement the backward functionality. and finally two matrixes “mat” and “percentage” where the first one will store the number of times the field has been visited (for each other field/room) and the other one will store the percentages. the main methods in this class are the addDestination(int dest) backward() and the run() ones, the first one will increment the number of times the field has been visited basing on the “dest” parameter and calculate the “line count” which will be used for the percentage of the fields in the line(how much the room has been visited and from which source);the second one is backward(), this function will decrement the appropriate zone when the user wants to move backward finally the “run()” method will just manage the communication between the user and the addDestination(int dest) method(asking for input and checking if it is valid; showing warnings...etc). the last methode is the print() one, from it’s name you can know what it does, it just prints the result in an organized way.



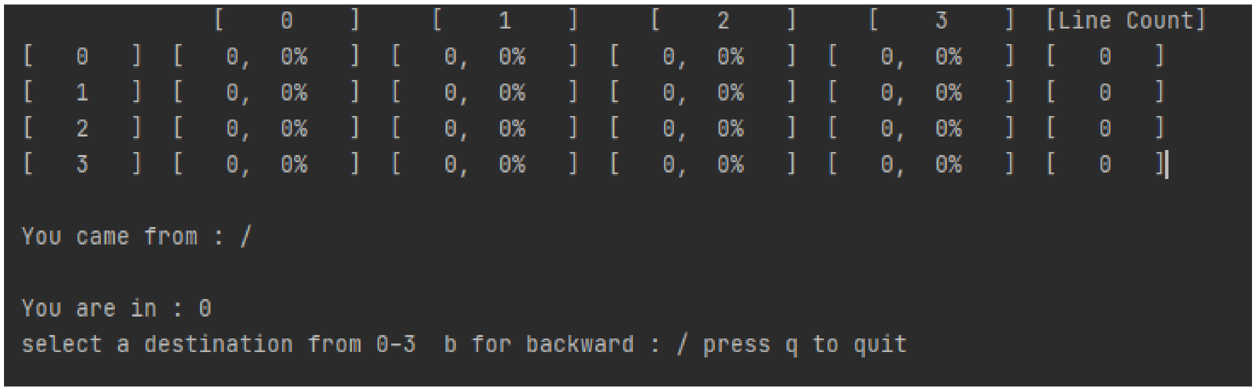
HMM
CLASS DIAGRAM



RESULT :



```
Enter number of rooms :
4
```



```
              [    0    ]    [    1    ]    [    2    ]    [    3    ]  [Line Count]
[   0   ]  [   0,  0%    ]  [   0,  0%    ]  [   0,  0%    ]  [   0,  0%    ]  [   0   ]
[   1   ]  [   0,  0%    ]  [   0,  0%    ]  [   0,  0%    ]  [   0,  0%    ]  [   0   ]
[   2   ]  [   0,  0%    ]  [   0,  0%    ]  [   0,  0%    ]  [   0,  0%    ]  [   0   ]
[   3   ]  [   0,  0%    ]  [   0,  0%    ]  [   0,  0%    ]  [   0,  0%    ]  [   0   ]

You came from : /

You are in : 0
select a destination from 0-3  b for backward : / press q to quit
```