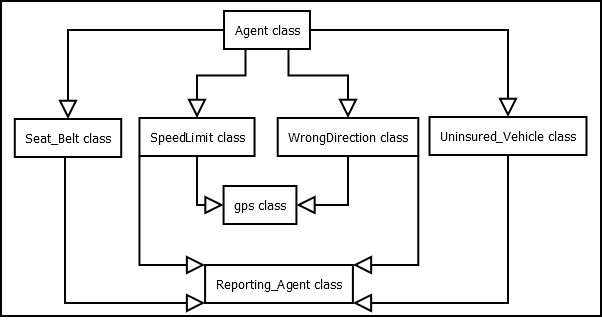
**Main classes:-** Server class and Agent class in Server.java and Agent.java file respectively.

**How to run program:-**

To execute the code first run server.java file and then run Agent.java file.

**Server.java:-** Running server.java file makes the server start and it will keep on running till it is stopped manually. Server class consists of ServerThread class which consists of the backend code where all the databases will get updated as per the violation and agent id when violation is committed by an agent.

**Agent.java:-** Running Agent.java file makes agent class run which contains different classes for violation and other module’s classes which we are using in our model. Below flowchart shows which class is connected to which class.



**Databases used:-**

1. RTO DATABASE:

RTO( S\_no, Agent\_ID, Aadhar\_num, DL\_num, Name, Address, Mobile\_num, Insurance\_expiry\_date, Fine )

**RTO**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S\_no | **Agent\_ID** | Aadhar\_num | DL\_num | Name | Address | Mobile\_num | Insurance\_expiry\_date | Fine |

1. VIOLATION DATABASE:

Violation\_detail( Violation\_ID, Violation\_rule, fine )

**Violation\_detail**

|  |  |  |
| --- | --- | --- |
| **Violation\_ID** | Violation\_rule | fine |

1. USER DATABASE(A table with name as agent id will be created if violation is committed. If table already exists in the table then it will not be created) 🡺

User( Violated\_rule, Date, Time, Fine\_deducted )

User

|  |  |  |  |
| --- | --- | --- | --- |
| Violation\_rule | Date | Time | Fine\_deducted |

1. GPS DATABASE :-

GPS( S\_no,x1\_Coordinate, y1\_Coordinate, x2\_Coordinate, y2\_Coordinate, Area\_name, Speed\_limit, Permitted\_direction)

**GPS**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Sno | x1\_Coordinate | y1\_Coordinate | x2\_Coordinate | y2\_coordinate | Area\_name | Speed\_limit | Permitted\_direction |

**About databases used in the program:-**

As it can be seen we have used database- rto, violation, gps, user(name as agentid).

Rto database stores details of people who have vehicles. We are assuming 10 people who have bought a car and agents are installed in those cars and the necessary data is already registered in rto at the time of purchase.

GPS database stores details of areas. So here we are assuming only rajajinagar Bangalore. So for this particular area we have taken 10 roads with their coordinates as starting and ending points of the road. Also permitted speed and direction on that road is also stored in the gps database.

In the violation database, different types of traffic rules violation with their unique violation id and amount of fine to be charged for the committed violation is stored in the violation database. Here we are assuming only four violations- Seat belt, Overspeeding, Wrong Direction, Uninsured Vehicle.

**Assumptions:-**

* All agents are connected to the same network.
* It is assumed that the user has already bought the car and agents are installed.
* Database of 10 people is assumed.
* Rajajinagar, Bangalore area is considered and all roads, its permitted speed and direction, starting and ending coordinates of roads are assumed for the Rajajinagar area.
* Buffer time of 10min is considered as 10sec and 15min is considered as 15sec for simulation purpose.For seatbelt, assuming that the engine has already started in the beginning and later based on a random number it can turn off.
* For fetching speed of the car’s speed we are assuming that it can be fetched using speedometer for now. In future if there is any new technology to detect speed in a efficient way then we can use that tech.
* For insurance we are checking insurance from year 2018 to 2021.
* After 3 days buffer, if insurance is not yet updated then it will be reported to the server automatically by sending his details.

**Section 1(Server class):-**

In server class, a socket server is defined with port number 1342 where it can accept multiple clients at the same time. So here clients are our agent so multiple agents can access the server at the same time and update the database. In the main function of the server class, the default constructor is called and then the server is started. Everytime agent tries to establish a connection with the server, a new object is created for each agent. Server will keep on running until it is stopped manually. After a connection is established, ServerThread class is called and executed using executor services which automatically runs the program by creating an override method void run(). Agent will connect to the server whenever there is a violation. Reporting agent will send the violating agent id and violation type to the server and then the server will update its database using that information. First database will be connected and a query will be executed to fetch violation id using violation type. After getting a violation id using another query, fine for the type of violation is stored in the database. Then using agent id, the current fine of that agent is fetched from the rto database and added to the fine for currently committed violation and the new value of fine is again stored back to the rto database by updating the rto table. After rto is updated and fine is charged to that agent then a seperate table for the user is created named as agent id. If the table already exists in the database for that agent then it will just update that table and if the table doesn't exist then a table is created and updated. This table is used as a user database which will be used by the user application to store the number of violations committed on which date and time and how much the user is charged for the committed violation. As of now, there is no user app so this table is used to print messages to notify the user. Once the database is updated and a committed violation is reported successfully, connection with that reporting agent will be closed.

**About program’s implementation:-**

There are two main classes- Agent and Server class. To run the program, the server should start and then the  agent. After the server is started, it waits for  the agent to create a connection. Here multithreading socket programming is used to create connection between agent and server.

In agent class, a while loop is created which runs for 1 hour excluding other buffer times. Inside while loop we are calling each violation class one by one to check whether that violation is committed or not. First seat belt violation is checked. Object of the seat belt class is created and then the object list function is defined to get multiple values in return from the function. After getting those values from the class, if else condition is used which checks whether violation was committed or not and based on that, a message will be displayed. So in the if statement, a condition is given as: if the engine is on and the driver is not wearing a seat belt then violation is committed. If the condition holds true then the message will be displayed accordingly. From the array list of seat belt class, agent id, receiver agent id, seat and engine value is taken in return. Seat=0 means driver is not wearing the seat belt and 1 means driver is wearing a seat belt. Similarly if engine = 0 means that engine has stopped and 1 means engine is still running. How seat belt class is implemented is given below:-

Seat\_Belt class:-

In this class we have implemented how the agent will be checked for seat belt violation. We have assumed here that engine is initially in start state. So first agent id is fetched from the database by using random serial number. Other way to get agent id was to generate agent id only randomly but for simulation purpose both cases works. To get agents details, reporting agent class’s object is created and then obj.violated\_agent and obj.reciever\_agent function is called to get violated agent id and receiver agent id respectively. In those function, database is connected and then agent id is fetched from rto database and sent back to seat belt class. Now after engine is started, buffer time of 10 min is given by calling buffer1 function. For simulation purpose we running this buffer1 only for 10sec but in actual implementation it should run for 10min. A random value is generated between 0 and 1 for the seat.

Then a while loop is started which will run till engine is turned off. Now randomly generated value for the seat is checked if its value is 0 or 1 if it is 1 then it means that driver is wearing a seatbelt so after 15min again it will check if the engine is still on. For 15min, buffer2 function is called. For simulation purpose this buffer is given only for 15sec but in actual implementation it will run for 15min. So after buffer of 15min again random value of engine and seat is generated. If engine’s value is 0 then it means that engine if off so it will stop checking and will come out of the loop. If the engine is still on then again it will check the value of the seat.

If it is 1 then again the above process will repeat and if it is 0 then it means that driver is not wearing a seat belt then inside else block code will be executed. When it is found that driver is not wearing seat belt, a warning will be given to the driver as five beeps. After five beeps a buffer time of 10 min is given by calling buffer1 function. Here it is only 10sec but in actual implementation 10min buffer time will be given. A random value of seat and engine is again generated. If the engine's value is not 0 then it will again check and if it is 0 then it means that engine is stopped and the loop will be terminated.

So if the engine is on then the value of the seat is checked. If it is 1 then it means that the driver is wearing a seat belt after warning and again the same process for seat=1 will happen. If seat =0 even after giving warning then it means that driver didn’t wear seat belt after warning so seat belt violation is committed which has to be reported by the reporting agent. So nearby agents will be searched and then agent id and violation type will be sent and that reporting agent will send these details to the server and the server will update the database. To carry out these tasks, reporting agent class’s object is created to which violated agent id andviolation type is sent to the function sent by calling and passing values to the function as obj.send(). In obj.send, a connection with the server will be established and data will be transferred to the server and database will be updated by executing queries.

How the server will update the database is given in section 1(server class) above. After sending details to the server a buffer time of 10sec will be given to the server to update the database and then by calling obj.report(), the user will be notified about the violation. Obj.report function acts as a user application where the user will receive notification about all the violations he has committed on which date and at what time and also how much fine is deducted for the committed violation. A buffer time of 10sec is given after sending data to the server because just after sending, the server will be busy in updating the database of that agent at that moment. If in case while checking for another violation of that agent again a violation is committed, it will again try sending details to the server which will give a socket error because the server will be still updating the database and it is not expecting any input from the agent. After reporting and sending notification to the user, it will go back to agent class and check for the next violation.

After the seat belt violation is checked, there is a transfer of control to the wrong direction class. In the agent class, an object for the wrong direction is created which returns multiple values from the wrong direction class. Therefore an object array list is used to return the agent ID, reporting agent ID, address and a flag value.

WrongDirection class:-

The basic idea of the wrong direction is to report the agent if it is moving in the direction other than the permitted direction.

The wrong direction class is implemented as follows:

* Two random numbers are generated between 1 and 10 as a number which will determine the location of the two agents. In the gps table, locations under consideration are numbered from 1 to 10, therefore the random number generated here will match to a location/s in the gps table.
* Two random numbers for violating agent ID and reporting agent ID are generated between 1 and 10 which will determine the agent ID from the rto gps table. An object of reporting agent class is created with the two randomly generated numbers as parameters and the IDs corresponding to the random numbers are returned from the rto table.
* For an agent to report another agent they have to be in the same vicinity. Therefore, the reporting agent checks if there are any agents in the vicinity by checking whether the two random numbers are equal. If they are not equal then the flag is assigned with the value 0 and returned to the agent class.
* If the two numbers are equal, then it means that an agent is found in the vicinity of the reporting agent.
* A random direction is generated from an array of all possible directions. The permitted direction for the location where the agent is fetched from the gps table using the gps class.
* Now the reporting agent has to check whether the violating agent is moving in the permitted direction. This is implemented by checking whether the fetched permitted direction and the randomly generated direction are equal.
* If the two directions are equal, then it means that no violation has been committed. The value of flag is assigned to 2 and is returned to the agent class.
* If the two directions are not equal, it means that the agent is moving in the opposite to the permitted direction. Therefore, this has to be reported and the value of flag is assigned to 1 and returned to the agent class.
* The agent ID, the reporting agent ID and the violation type is sent to the reporting agent class by sending these details to obj.send() function where connection to the server is established. A buffer of 10sec is given so that there is time for the server to receive information and update the database accordingly. How the server will update the database is given in section 1(server class) above.

If the violation is committed then the name, violation type, area in which the violation is committed and the fine deducted is displayed.

SpeedLimit class:-

After a connection is established between the server and the agent, an object for the speed\_limit class is created in the agent class. An array list is created to get multiple return values from the function check\_speed in the speed\_limit class.In the speed\_limit class, the variable violation\_type is set to “ Speed limit violation”. In the check\_speed function, it is checked if there are any nearby agents. This is done in the following way, since this is a violation, random numbers are used to find coordinates,current speed, agent id and the reporting agent id. First, random numbers from 1-10 are generated and stored as the  coordinates of the reporting agent and the violating agent. Then, again random numbers from 1-10 are generated to fetch the reporting agent id and the violating agent id, a while loop is used to make sure that the two random numbers generated are different. The violating agent id and the reporting agent id are fetched by creating an object of the class Reporting\_agent and passing the two random numbers as the parameters. The fetched violating agent id and the reporting agent id are stored in the variables agent\_id and rec\_agent\_id respectively. Now, the coordinates of the violating agent and the reporting agent are checked for equality, if they are equal it is assumed that an agent is found in the nearby area and the agent id of the violating agent is displayed. The current speed of the agent is found by generating a random number from 1-100. The permitted speed and the area is fetched by creating an object of the class gps by passing the violating agent coordinate as parameter. The current speed is checked with the permitted speed, if the current speed exceeds the permitted speed, then a buffer of 10 seconds is given to the agent to reduce its speed by calling the function buffer1(), again a new current speed is found by generating a random number from 1-100, if the new current speeds exceeds the permitted speed then the variable flag is set to 1 and a message is displayed telling that a violation has occurred, the agent id, reporting agent id and the violation type is sent to the class Reporting agent and the function report is called else the flag is set to 2. The flag is set to 2 even if the current speed is within the permitted speed. The variables - flag, area, agent id and reporting agent id are sent as an array list to the agent class.

The agent ID, the reporting agent ID and the violation type is sent to the reporting agent class by sending these details to obj.send() function where connection to the server is established. A buffer of 10sec is given so that there is time for the server to receive information and update the database accordingly. How the server will update the database is given in section 1(server class) above.

After checking for overspeeding it will check for uninsured vehicles. Here also, an object of uninsured vehicle class is created and an object list function is called which will return agent id, receiver agent id and check. If check is 1 then it means that agent has a valid insurance or was expired and got his insurance updated within given buffer time. If the check's value is 0 then it means that agent has committed this violation and the committed violation by the agent is reported by another agent i.e. receiver agent. How uninsured vehicle class is implemented is given below:

Uninsure\_Veicle class:-

First, I have globally declared violation\_type, expiry\_date, year, month, date, buffer in the public class  “Uninsured\_Vehicle”. Where violation type is Insurance violation, expiry\_date is the expiry date of the insurance of the vehicle. Year, month, date are of the current date. Buffer as buffer1 is the time given to the driver to update the required information. Here buffer time of 3 days should be given, for simulation purposes I have given buffer time of 10 seconds only.

Second, I have created another public static class, Update\_insurance, of type string. Here, the updated year will be current year +1.The date is randomly generated between 10th and 13th of the month. The updated month will be the same as the month of the previous expiry date. The query is returned where the query has updated year, month and date of the expiry date of the particular agent id.

There will be a delay because the server is waiting for the agent to send input to it. Whereas, the agent is waiting for the classes to find the violation and send to it.

Further I have created a public static class, getInsuranceDetail, of type List<Object>. Here I have declared a variable, check of type Int. If the check is equal to 1 then the insurance is already valid. If the check is equal to 0 then the insurance is invalid. Then agent id and reporting agent id is generated randomly using random sno and rsno numbers. Reporting agent class’s object is created and agent id and reporting agent id function are called to get agent id and reporting agent id.

A connection with rto database is established. Insurance expiry date of the vehicle is fetched from rto database of the agent id generated. For the current date, we are assuming the date to be 10 whereas, year and month are randomly generated.

Current date and Insurance expiry date are compared.

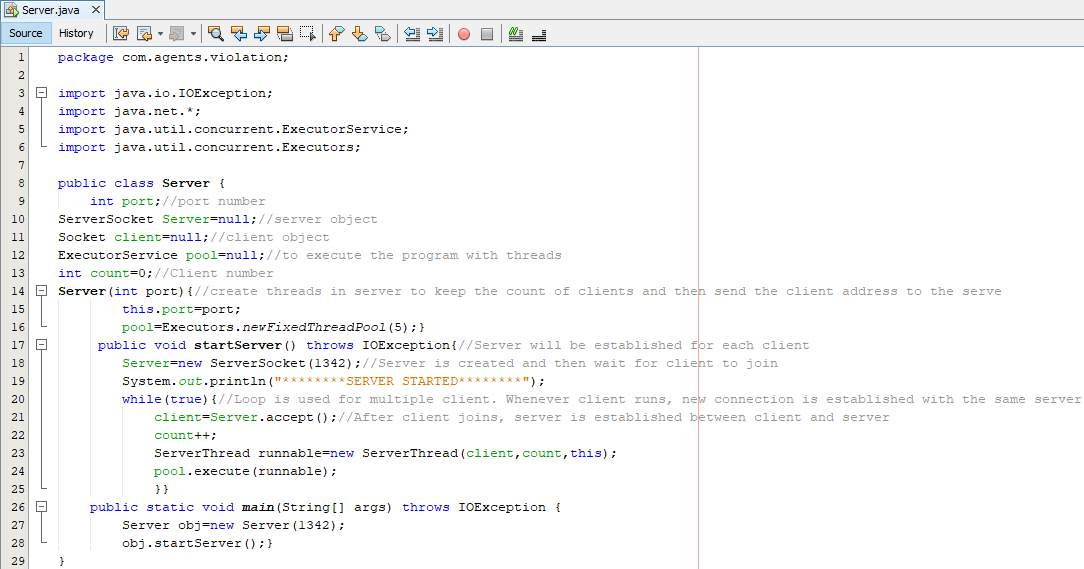
If the current date is greater than 0, then the Insurance of the vehicle has expired already. A buffer is given. The Insurance expiry date will be checked again after the warning for three days is over. A random number(r) is generated. If the random number(r) generated is equal to 0 then the Insurance has not been updated yet. The agent id and violation type will be sent to a nearby agent through WIFI module. A buffer of 10 seconds is given and check is set to 0. If the random number(r) generated is 1 then the insurance is updated.

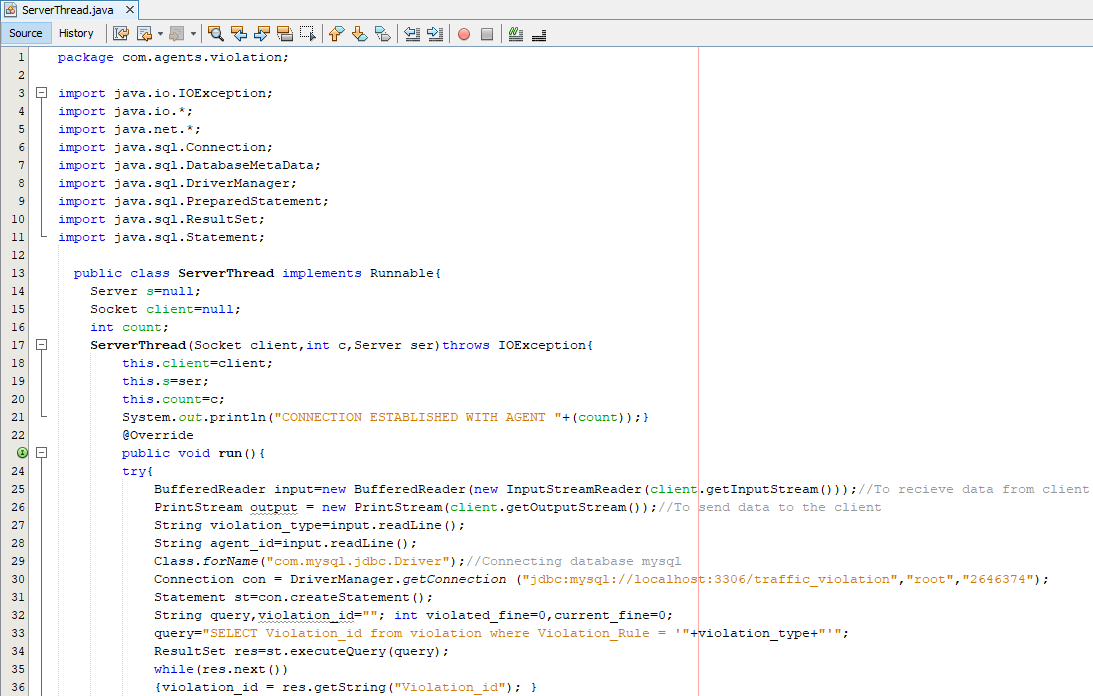
If the current date is equal to 0 then that means the insurance is expiring on the same date. A buffer is given. The Insurance expiry date will be checked again after the warning for three days is over. A random number(r) is generated again. If the random number(r) generated is equal to 0 then the Insurance has not been updated yet. The agent id and violation type will be sent to a nearby agent through WIFI module. A buffer of 10 seconds is given and check is set to 0. If the random number(r) generated is 1 then the insurance is updated.

If the violation is committed, that is the insurance is not updated even after warning then this violation will be reported by reporting agent id by calling send() function which will send agent id and violation type to the server and server will update the database. The updating process of database by server is explained in section 1(server class). After the violation is reported, buffer time of 10sec is given for the server to update the database and after that notification to the user is sent. Since we are simulating so just a message will be printed and after doing that agent id, receiver agent id and check is sent back to agent class and based on check’s value, message is printed.

After all the four violations are checked then again it will check all the violations till the loop has completed running for 1 hrs. Including buffer time that is given in the violation, it can take more time than 1hour but while loop will be terminated after his 1 hrs is completed.

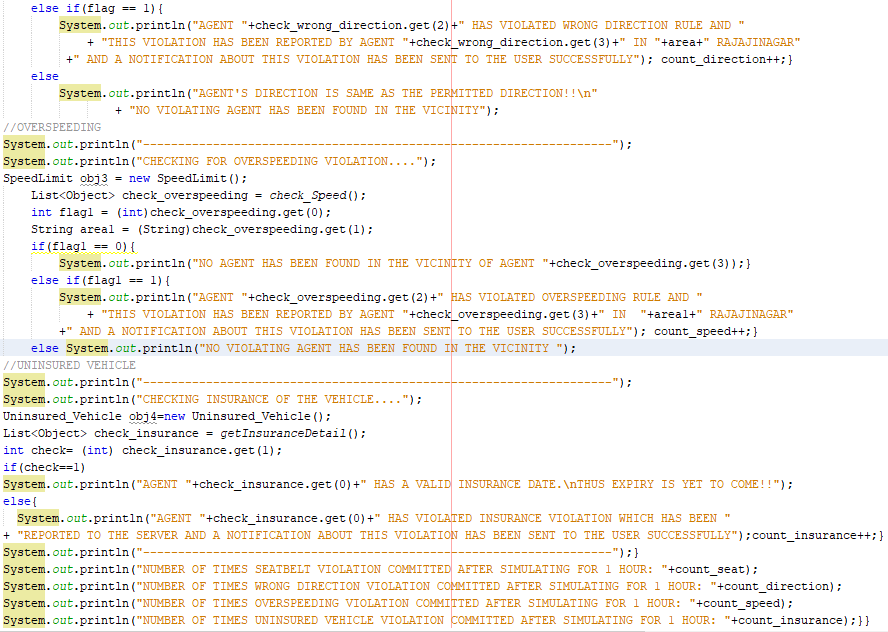
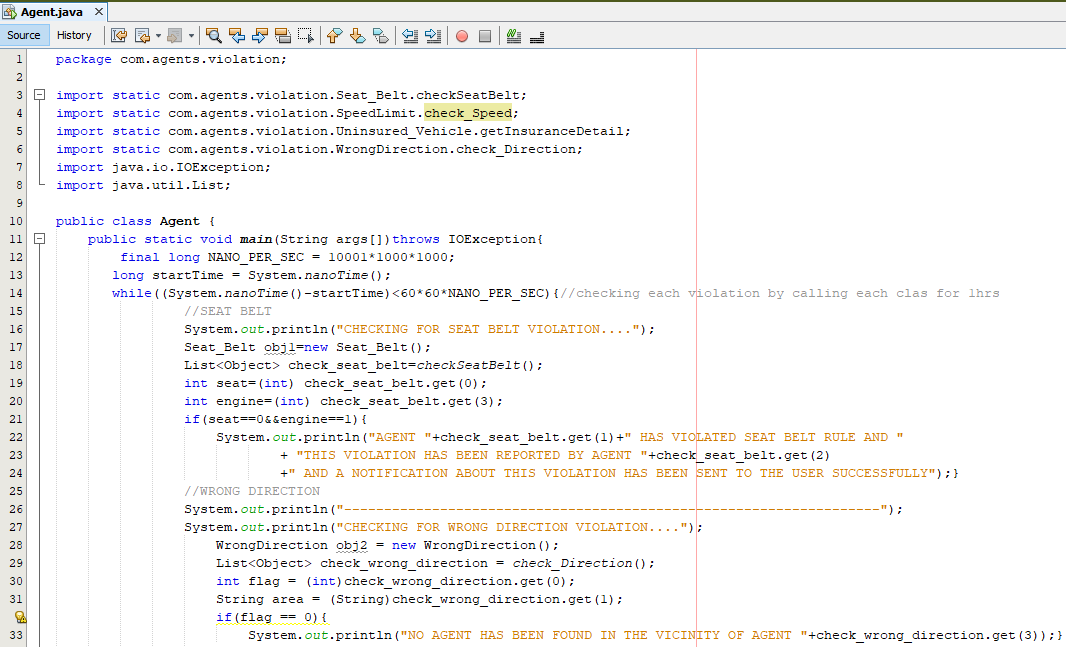
**Main Server class and connected ServerThread class:-**



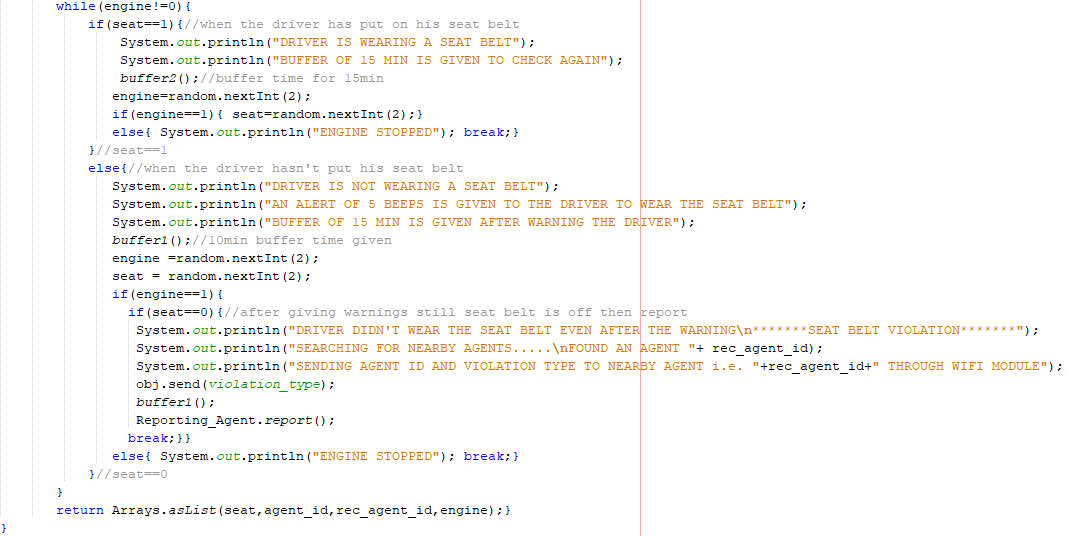
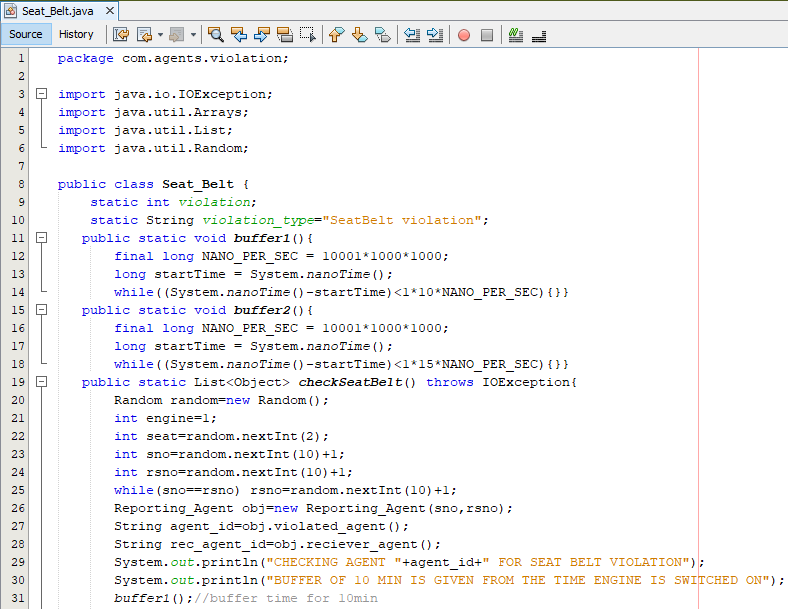


**Main Agent class and its connected classes:-**

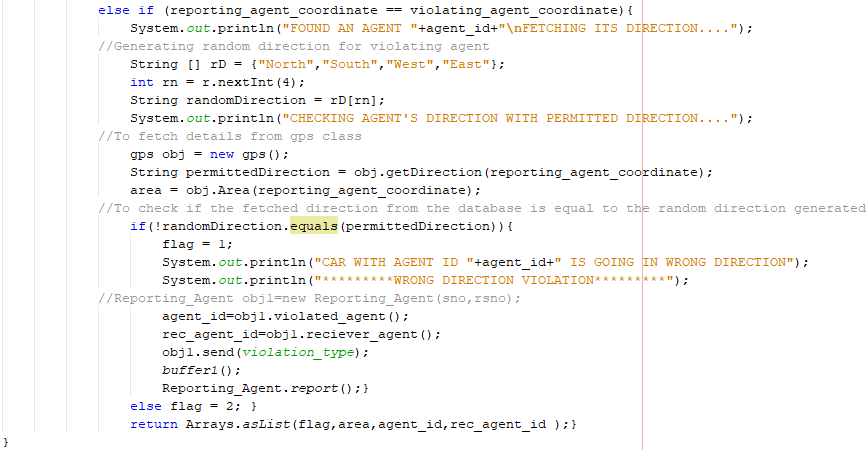
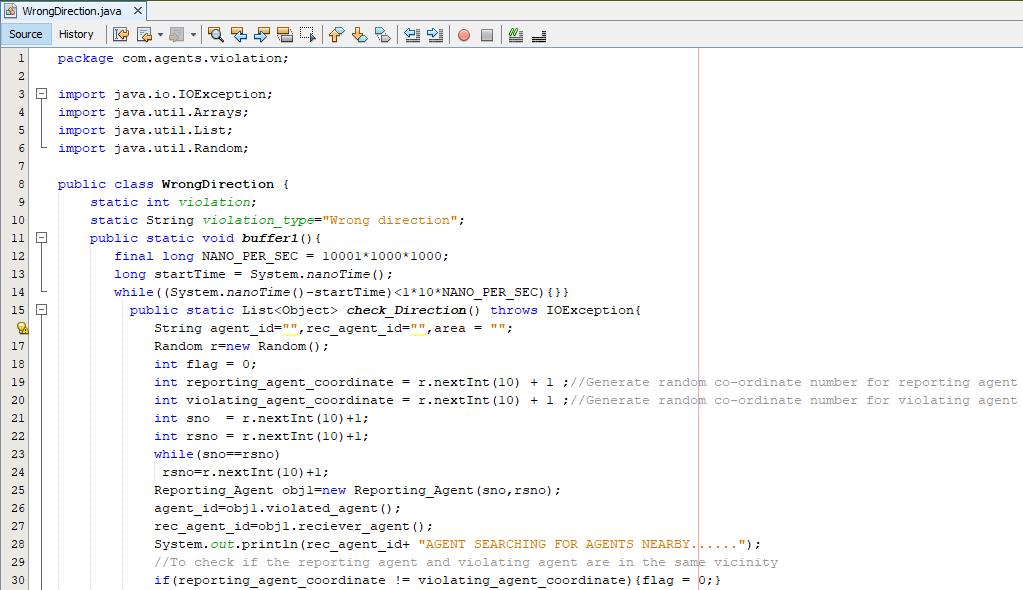
1. Agent class:-



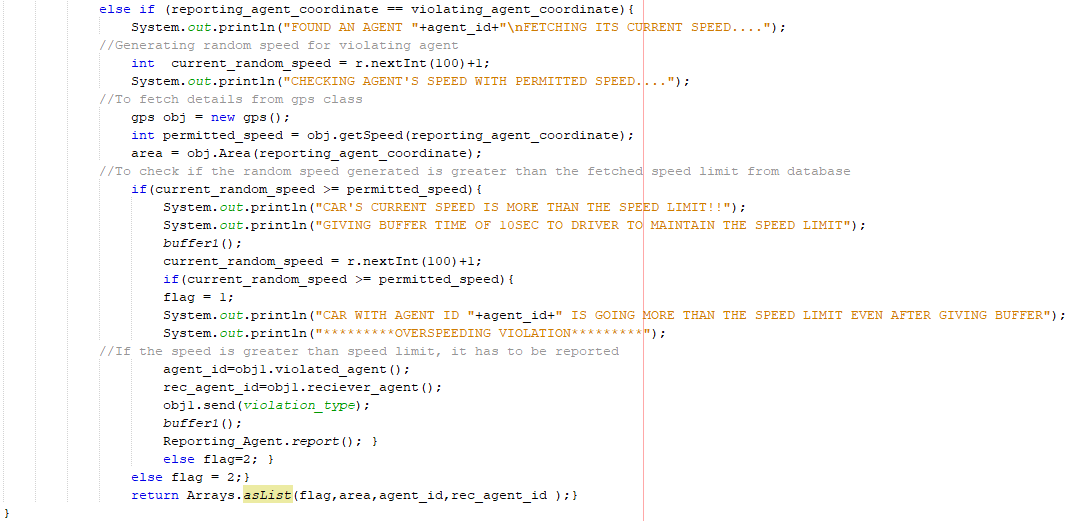
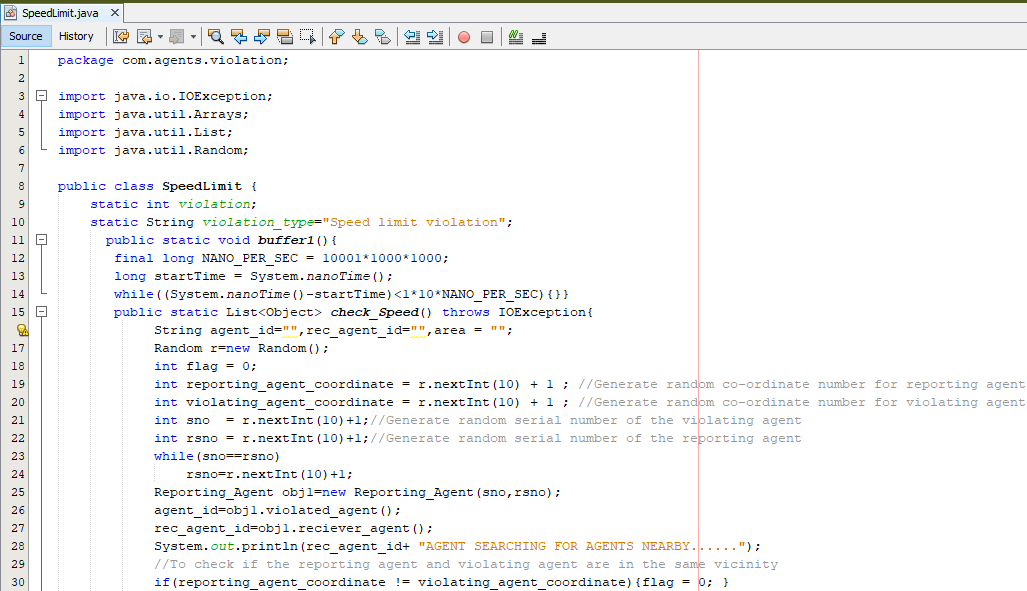
1. Seat\_Belt class:-



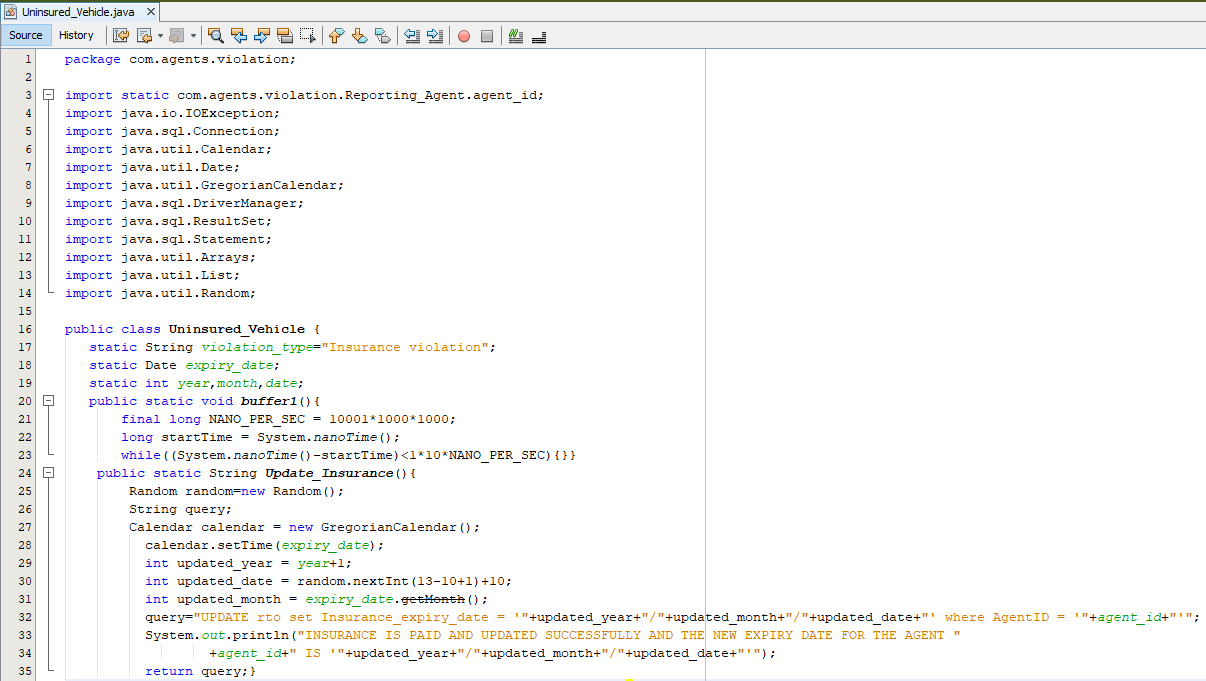
1. WrongDirection class:-

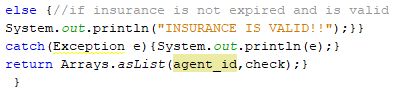
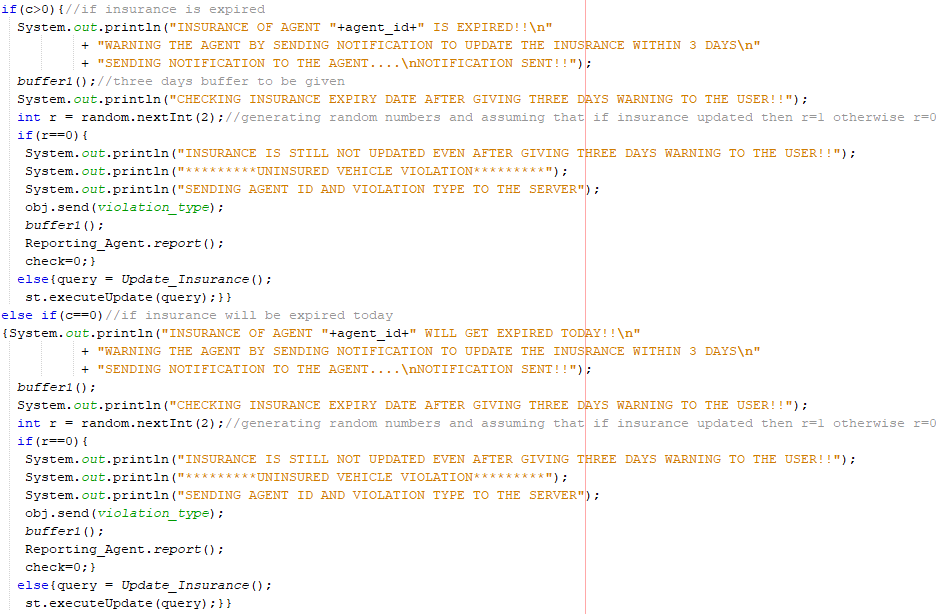


1. SpeedLimit class:-

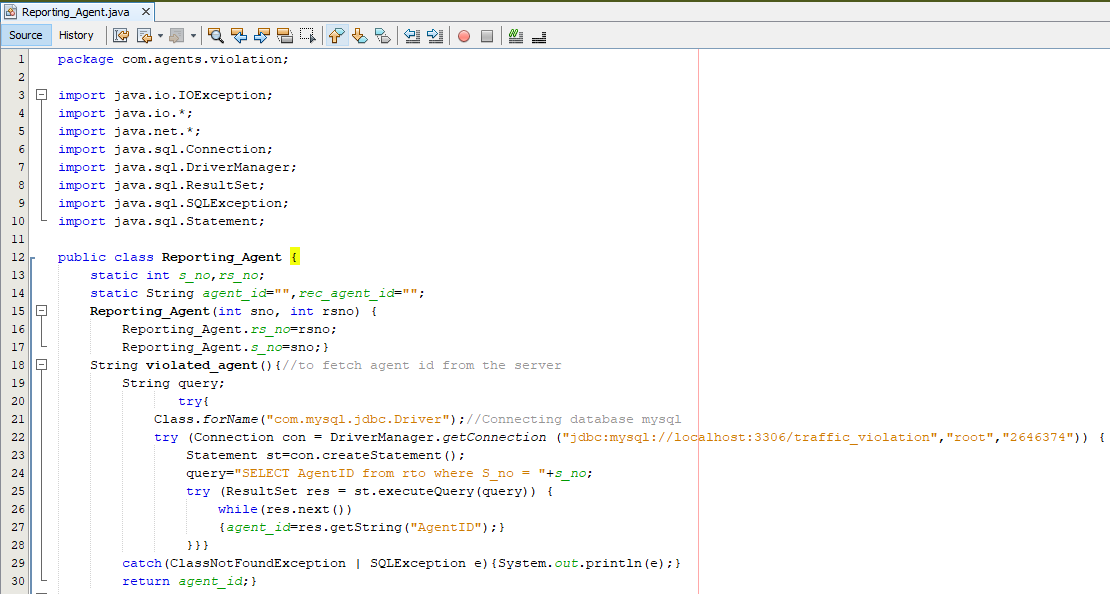


1. Unisured\_Vehicle class:-





1. Reporting\_Agent class:-





1. Gps class:-

