CS 4326
Assignment #3
Due: Wednesday, March 21
100 points

You may work with one other person on this program. I reserve the right to assign different grades to each person if I feel that an equal contribution has not been made.

You are to write a program that will assist in tracking of vehicles. The overall system consists of 4 parts:

a) GPS device, vehicle laptop. I will give you a program which will simulate this.
b) Server program which communicates with vehicles and the mapping portion. It maintains a list of all vehicles and locations and sends vehicle information to the mapping portion.
c) Mapping program which gets vehicle information from the server program and creates a file so that the vehicles can be mapped.
d) Web page map.

You will only be responsible for the second part: the program to maintain a list of vehicles. This will require reading position messages from 'vehicles' and responding to vehicle information from the mapping program.

Vehicle Messages

The 'vehicle' will send in positions using a modified NMEA 0183 format and specifically the RMC command (Recommended Minimum command). There are several sources available to describe NMEA message format. In brief, an RMC message looks like:

```
$GPRMC,123519,A,4807.038,N,01131.000,E,022.4,084.4,230394,003.1,W*6A

Where:
       RMC          Recommended Minimum sentence
       123519       Fix taken at 12:35:19 UTC
       A            Status A=active or V=Void.
       4807.038,N   Latitude 48 deg 07.038' N
       01131.000,E  Longitude 11 deg 31.000' E
       022.4        Speed over the ground in knots
       084.4        Track angle in degrees True
       230394       Date - 23rd of March 1994
       003.1,W      Magnetic Variation
       *6A          The checksum data, always begins with *
```

An NMEA sentence will always end with endline (<CR><LF>). The checksum is a XOR of all the bytes between the $ and *.

The modification will be that there is an extra field indicating vehicle identifier added just before the checksum. For example, if the vehicle identifier is V101, the message would now be:

```
$GPRMC,123519,A,4807.038,N,01131.000,E,022.4,084.4,230394,003.1,W,V101*30
```

Your program should listen on UDP port 9099 for these modified RMC messages. You should ensure that the checksum is correct and discard any messages with invalid checksum (with error messages as appropriate).

Vehicles tend to go to places where regular updates may not happen. In the absence of updates, you should keep the last known position/status of a vehicle for up to 30 seconds. If you have not received an update from a vehicle for 30 seconds, you should drop the vehicle from the list of vehicles. You do not need to use the time in the RMC message. You can use system times as in LocalTime.now().

Map Requests and Response

Map requests will be made with TCP port 9099. In the best case, your program should be able to handle multiple map program connections on that port. After establishing a connection, the mapping program will request vehicle positions using XML. As long as requests are being received, your program should reply to each and maintain the connection. However, if a mapping program fails to request an update for over 30 seconds, the server program should close the connection.

TCP tends to work best if an entire message ends with end-of-line… otherwise, you will have to receive each line individually and group them together. In order to simplify your TCP receive, you can assume that there will be no end-of-lines until one at the end of the entire message.

A valid update request will look like:
For all vehicles (with valid checksum):

```
<AVL><vehicles>all</vehicles></AVL>
```

For vehicles with 'Active' status only:

```
<AVL><vehicles>active</vehicles></AVL>
```

Check to make sure it is valid XML. If the XML is not correct format, your program should discard the message (with error messages as appropriate).

NOTE #1
My test program will send XML that may have random errors anywhere in the message. You will need to ensure that it is both correctly formatted XML and that it has either 'all' or 'active'.

The response to the mapping program will be a list of all vehicle information in JSON format. For each vehicle, you should provide vehicle identifier, status, latitude, longitude, speed, heading (track angle) and time
Notes on this:

1) Lat/Long in NMEA is in Degrees and Minutes and uses E/W and N/S. The Lat/Long in this message should be in decimal degrees and use negatives for South latitudes and West longitudes.
2) Speed in NMEA is in knots. You should convert to MPH for this message.
3) Use 24 hr format with colons added for 'prettiness'.

A valid json response for one vehicle in an 'all' request would be:

```
{
  "status":"all",
  "vehicles":[
    {
            "ident":"222",
            "status":"A",
            "latitude":"30.693883895874023",
            "longitude":"-95.5493392944336",
            "speed":"18.100000381469727",
            "heading":"160.1999969482422",
            "time":"13:09:16"
    }
     ]
}
```

Without newlines in message, this would actually be:

```
{ "status": "active", "vehicles": [{"ident": "V102","status":
"A","latitude": "-49.2273","longitude": "-23.47","speed":
"70","heading": "45",time": "13:35:19"]}
```

A valid json response for two vehicles in an 'all' request would be:

```
{
      "status":"active",
      "vehicles":[
      {
            "ident":"111",
            "status":"A",
            "latitude":"30.74230194091797",
            "longitude":"-95.5747299194336",
            "speed":"31.100000381469727",
            "heading":"50.70000076293945",
            "time":"13:09:10"
      },
      {
            "ident":"222",
            "status":"A",
            "latitude":"30.693883895874023",
            "longitude":"-95.5493392944336",
            "speed":"18.100000381469727",
            "heading":"160.1999969482422",
            "time":"13:09:16"
      }
      ]
}
```

{"status":"active","vehicles":[{"ident":"111","status":"A","latitude": "30.74230194091797","longitude":"-95.5747299194336","speed":"31.100000381469727", "heading":"50.70000076293945","time":"13:09:10"},{"ident":"222","status":"A","latitude":"30.693883895874","longitude":"-95.5493392944","speed":"18.1","heading":"160.199", "time":"13:09:16"}]}

If mapping program sent an invalid XML request, the reply should be:

```
   {
     "status": "invalid",
     "vehicles": [
      ]
   }
```

In actual message, no newline until one at very end:
{"status":"Invalid","vehicles":[]}

You may use any language you wish, however I would prefer you don't use .Net languages because I do not have a Winders computer in my CS office. Turn in source code on Blackboard as well as hand in a paper copy of the source code. I will test it against my own programs to simulate vehicles and mapping programs. You should probably develop your own in order to test it prior to handing in your server program code however, only the server program should be turned in… as long as you adhere to the defined interfaces, it should work with mine.

NOTE #2

Again, you are only responsible for the Server part. That server part will need to multithreaded to handle asynchronous UDP messages and TCP request with a Thread to handle each TCP connection. You will need to have some type of control over shared data. For example, UDP messages will update list of vehicles, while TCP requests will need to send that list of vehicles. In Java, you can do this with either synchronized classes to handle the vehicle data or with synchronized sets, maps or lists to let Java do the synchronization for you. Things like ArrayLists are NOT thread-safe. They can be embedded in other thread-save collections, however.