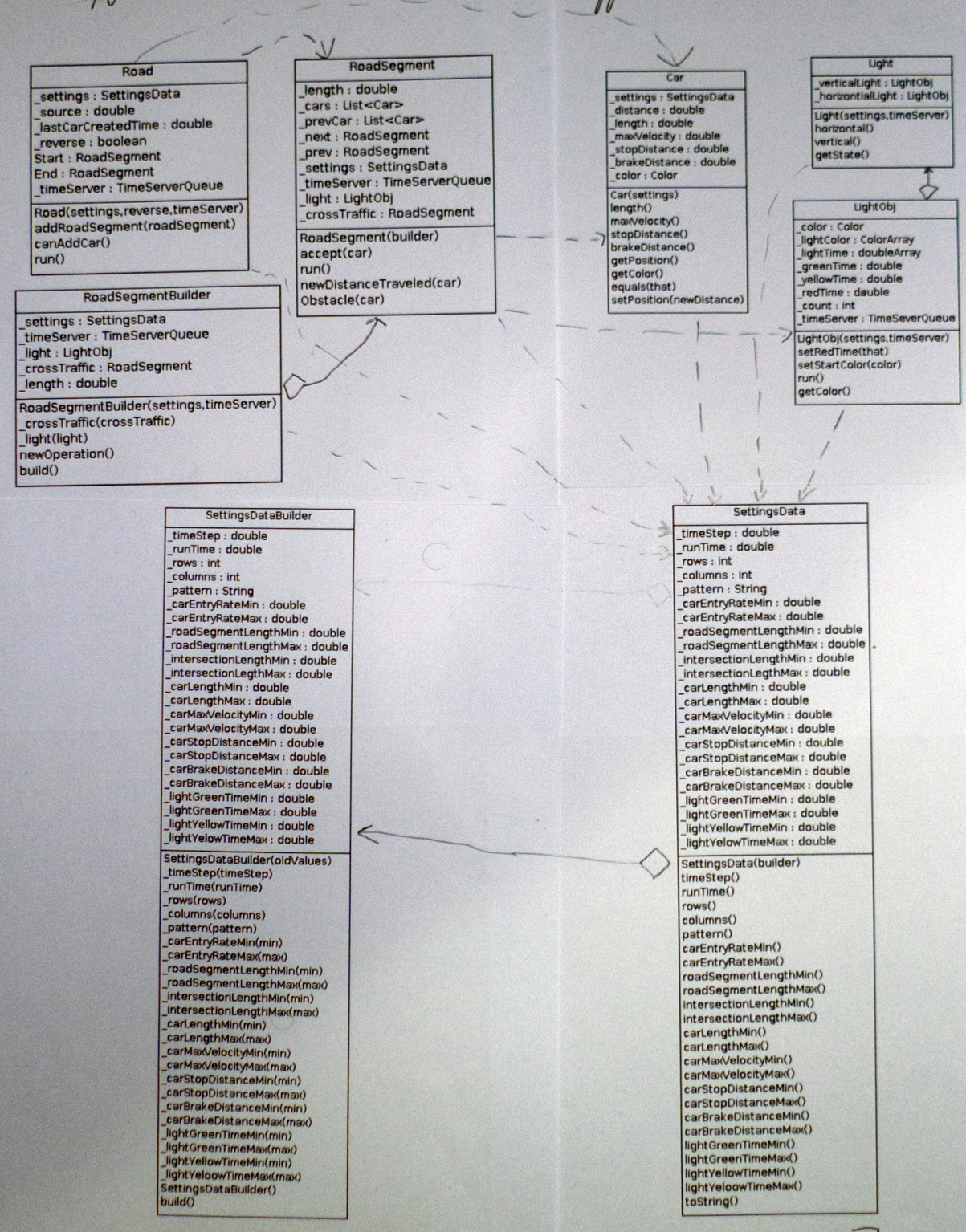
ANDREW JAMES TILLMANN

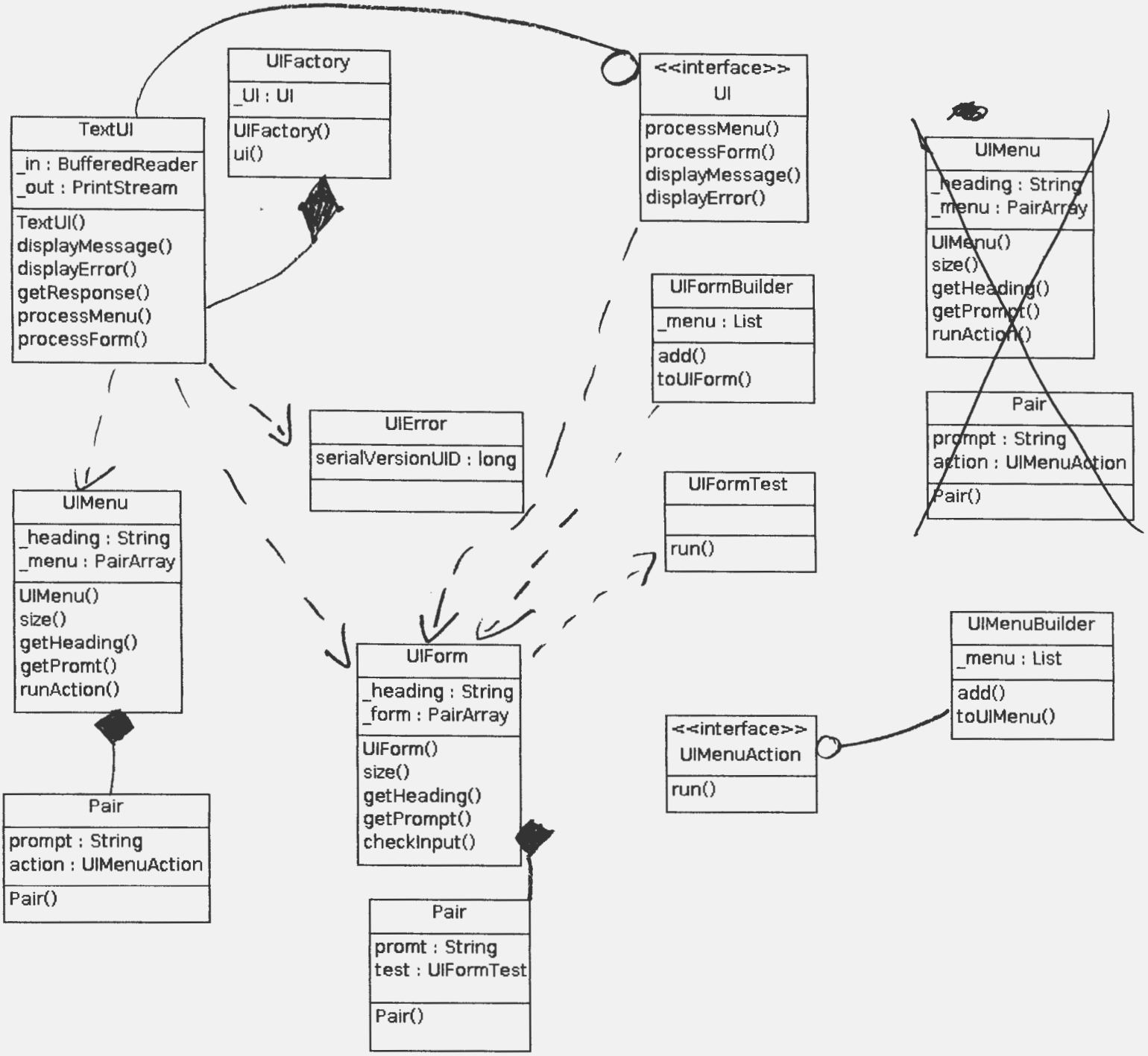
Traffic Simulation Project

SE 450 Winter 2013-2014

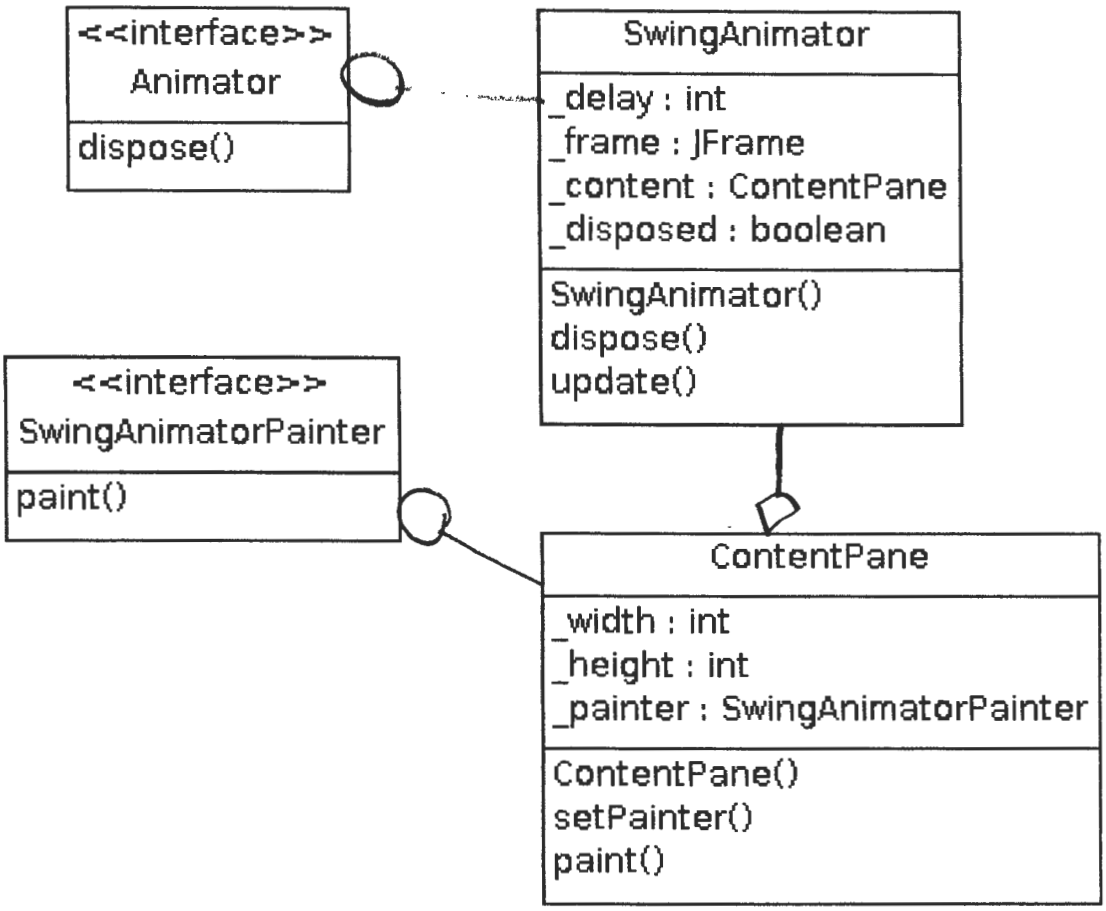
Class diagrams for each package. Include a design class diagram for each package in your project. Be sure to include all significant class relationships: realization, specialization, and association. Show associations as dependencies, aggregations or compositions when appropriate. Show attributes and methods only if they are crucial to understanding the class relations.
Do not use tools that automatically generate diagrams from your code. The diagrams they produce are unreadable.


Road
_settings : SettingsData
_source : double
_lastCarCreatedTime : double
_reverse : boolean
Start : RoadSegment
End : RoadSegment
_timeServer : TimeServerQueue
Road(settings,reverse,timeServer)
addRoadSegment(roadSegment)
canAddCar()
run()
RoadSegment
_length : double
_cars : List<Car>
_prevCar : List<Car>
_next : RoadSegment
_prev : RoadSegment
_settings : SettingsData
_timeServer : TimeServerQueue
_light : LightObj
_crossTraffic : RoadSegment
RoadSegment(builder)
accept(car)
run()
newDistanceTraveled(car)
Obstacle(car)
Car
_settings : SettingsData
_distance : double
_length : double
_maxVelocity : double
_stopDistance : double
_brakeDistance : double
_color : Color
Car(settings)
length()
maxVelocity()
stopDistance()
brakeDistance()
getPosition()
getColor()
equals(that)
setPosition(newDistance)
Light
_verticalLight : LightObj
_horizontialLight : LightObj
Light(settings,timeServer)
horizontal()
vertical()
getState()
LightObj
_color : Color
_lightColor : ColorArray
_lightTime : doubleArray
_greenTime : double
_yellowTime : double
_redTime : double
_count : int
_timeServer : TimeSeverQueue
LightObj(settings,timeServer)
setRedTime(that)
setStartColor(color)
run()
getColor()
RoadSegmentBuilder
_settings : SettingsData
_timeServer : TimeServerQueue
_light : LightObj
_crossTraffic : RoadSegment
_length : double
RoadSegmentBuilder(settings,timeServer)
_crossTraffic(crossTraffic)
_light(light)
newOperation()
build()
SettingsDataBuilder
_timeStep : double
_runTime : double
_rows : int
_columns : int
_pattern : String
_carEntryRateMin : double
_carEntryRateMax : double
_roadSegmentLengthMin : double
_roadSegmentLengthMax : double
_intersectionLengthMin : double
_intersectionLegthMax : double
_carLengthMin : double
_carLengthMax : double
_carMaxVelocityMin : double
_carMaxVelocityMax : double
_carStopDistanceMin : double
_carStopDistanceMax : double
_carBrakeDistanceMin : double
_carBrakeDistanceMax : double
_lightGreenTimeMin : double
_lightGreenTimeMax : double
_lightYellowTimeMin : double
_lightYelowTimeMax : double
SettingsDataBuilder(oldValues)
_timeStep(timeStep)
_runTime(runTime)
_rows(rows)
_columns(columns)
_pattern(pattern)
_carEntryRateMin(min)
_carEntryRateMax(max)
_roadSegmentLengthMin(min)
_roadSegmentLengthMax(max)
_intersectionLengthMin(min)
_intersectionLengthMax(max)
_carLengthMin(min)
_carLengthMax(max)
_carMaxVelocityMin(min)
_carMaxVelocityMax(max)
_carStopDistanceMin(min)
_carStopDistanceMax(max)
_carBrakeDistanceMin(min)
_carBrakeDistanceMax(max)
_lightGreenTimeMin(min)
_lightGreenTimeMax(max)
_lightYellowTimeMin(min)
_lightYeloowTimeMax(max)
SettingsDataBuilder()
build()
SettingsData
_timeStep : double
_runTime : double
_rows : int
_columns : int
_pattern : String
_carEntryRateMin : double
_carEntryRateMax : double
_roadSegmentLengthMin : double
_roadSegmentLengthMax : double
_intersectionLengthMin : double
_intersectionLegthMax : double
_carLengthMin : double
_carLengthMax : double
_carMaxVelocityMin : double
_carMaxVelocityMax : double
_carStopDistanceMin : double
_carStopDistanceMax : double
_carBrakeDistanceMin : double
_carBrakeDistanceMax : double
_lightGreenTimeMin : double
_lightGreenTimeMax : double
_lightYellowTimeMin : double
_lightYelowTimeMax : double
SettingsData(builder)
timeStep()
runTime()
rows()
columns()
pattern()
carEntryRateMin()
carEntryRateMax()
roadSegmentLengthMin()
roadSegmentLengthMax()
intersectionLengthMin()
intersectionLengthMax()
carLengthMin()
carLengthMax()
carMaxVelocityMin()
carMaxVelocityMax()
carStopDistanceMin()
carStopDistanceMax()
carBrakeDistanceMin()
carBrakeDistanceMax()
lightGreenTimeMin()
lightGreenTimeMax()
lightYellowTimeMin()
lightYeloowTimeMax()
toString()

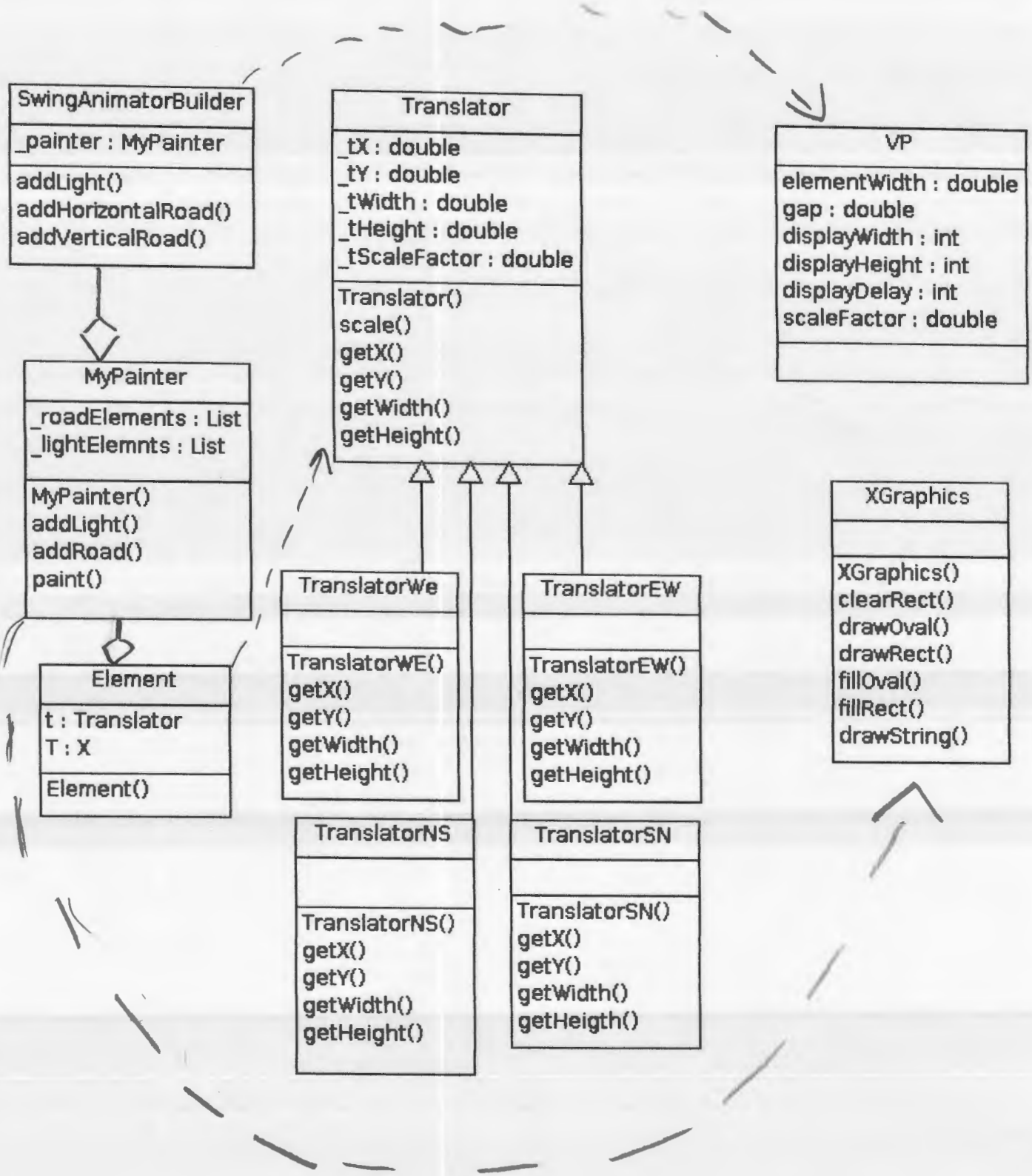
traffic.ui


UIFactory
_UI : UI
UIFactory()
ui()
<<interface>>
UI
processMenu()
processForm()
displayMessage()
displayError()
TextUI
_in : BufferedReader
_out : PrintStream
TextUI()
displayMessage()
displayError()
getResponse()
processMenu()
processForm()
UIFormBuilder
_menu : List
add()
toUIForm()
UIMenu
_heading : String
_menu : PairArray
UIMenu()
size()
getHeading()
getPrompt()
runAction()
Pair
prompt : String
action : UIMenuAction
Pair()
UIError
serialVersionUID : long
UIFormTest
run()
UIMenu
_heading : String
_menu : PairArray
UIMenu()
size()
getHeading()
getPromt()
runAction()
UIForm
_heading : String
_form : PairArray
UIForm()
size()
getHeading()
getPrompt()
checkInput()
UIMenuBuilder
_menu : List
add()
toUIMenu()
<<interface>>
UIMenuAction
run()
Pair
prompt : String
action : UIMenuAction
Pair()
Pair
promt : String
test : UIFormTest
Pair()

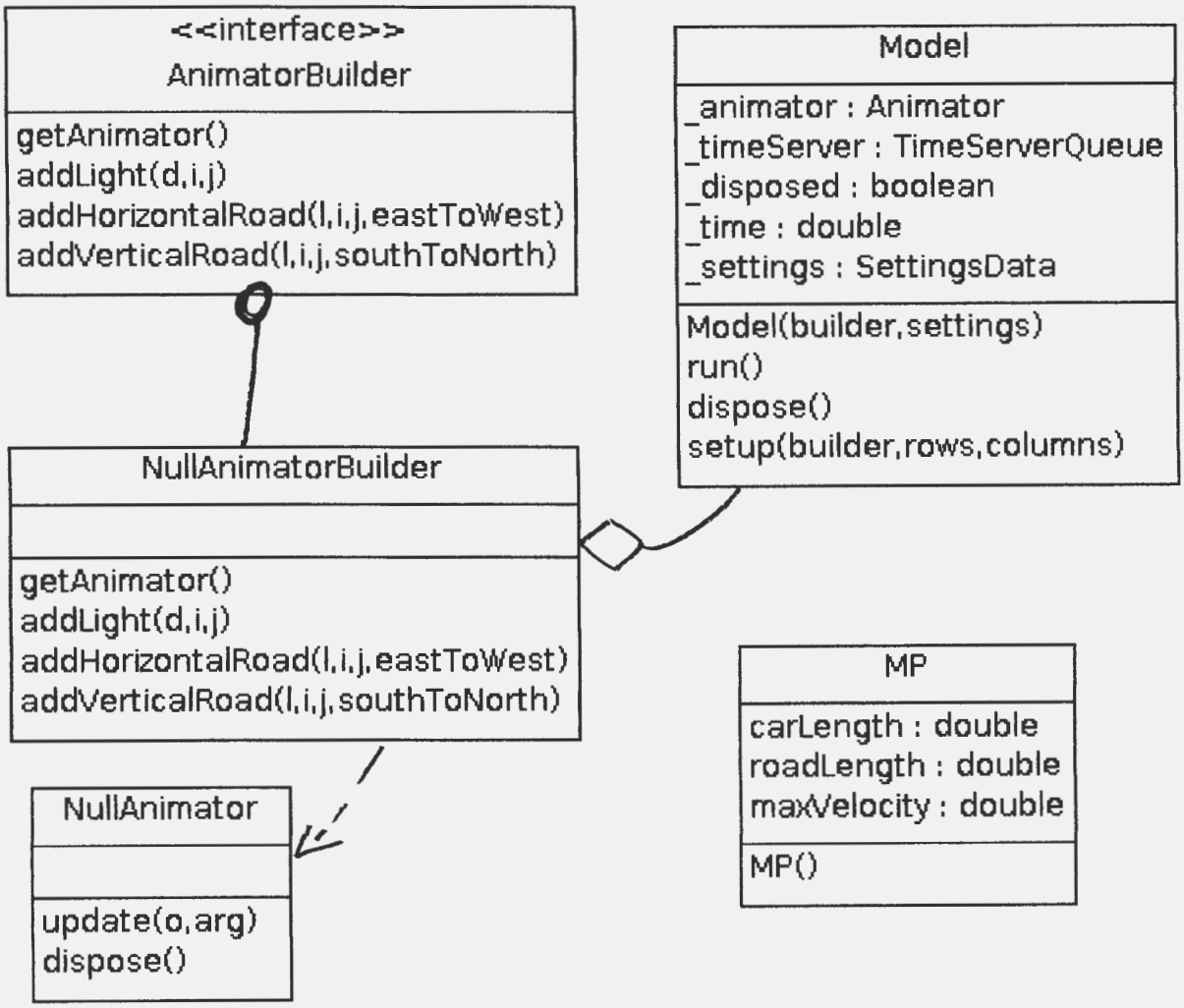
traffic.util



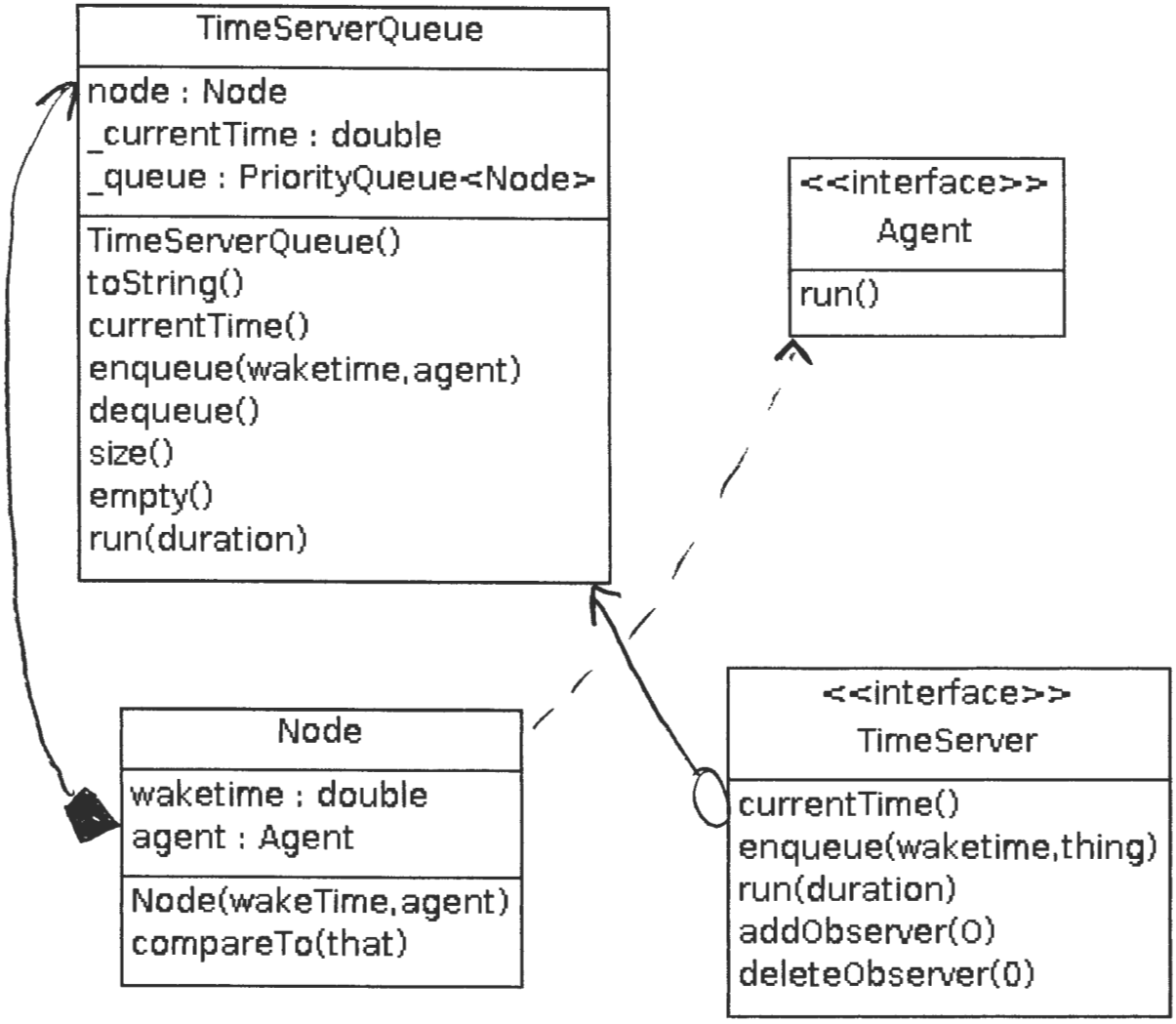
traffic.model.swing



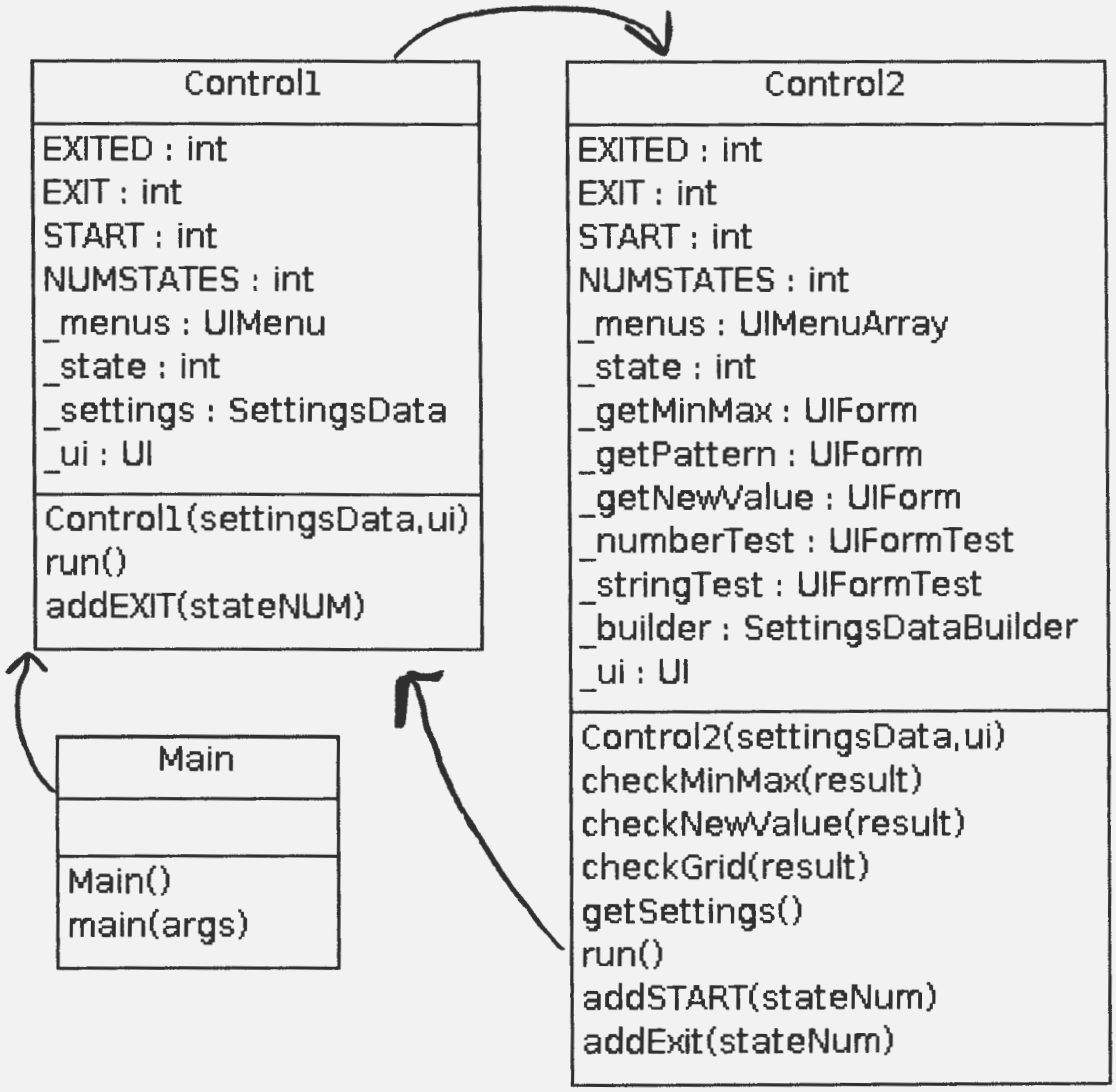
traffic.model.



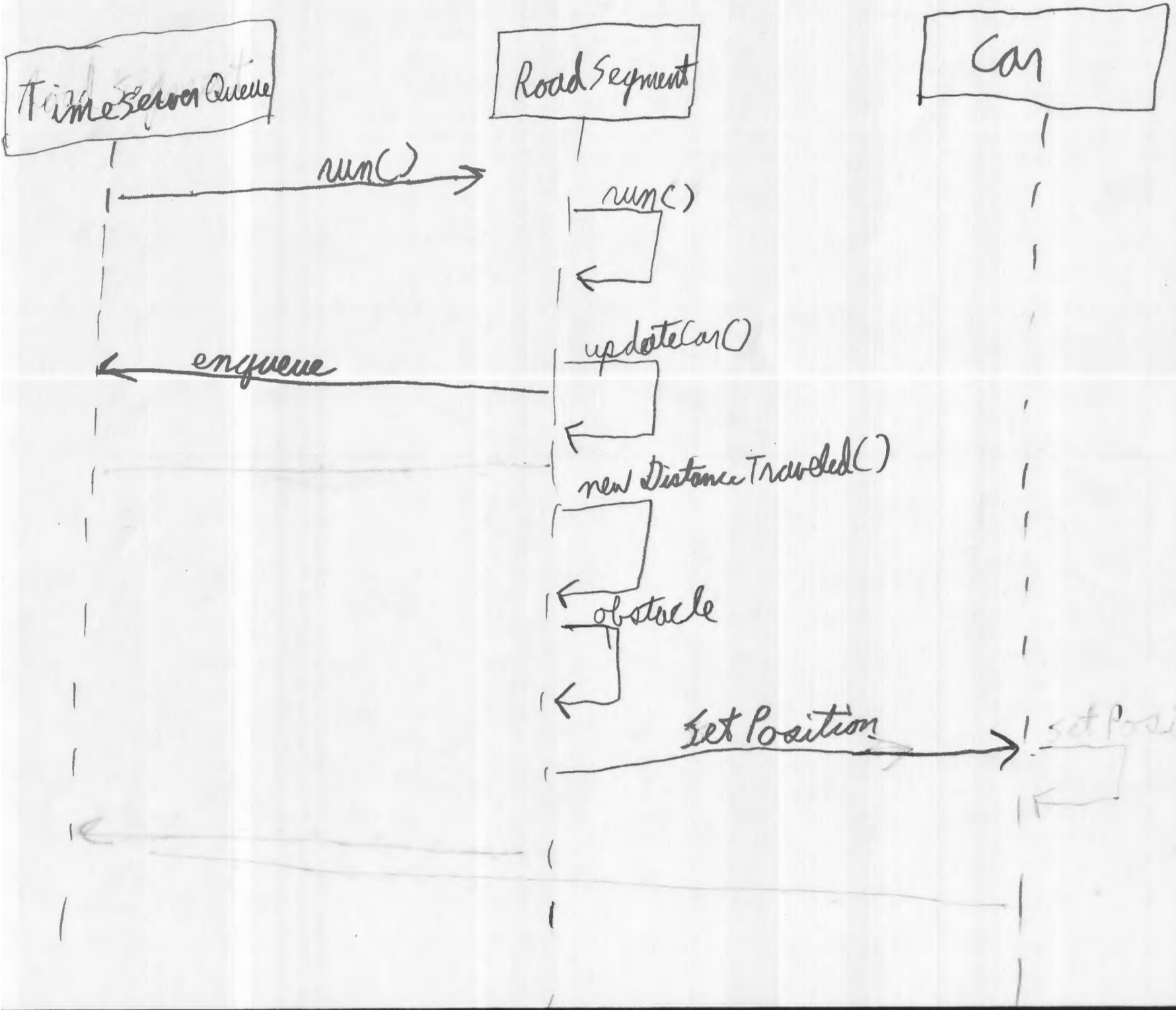
traffic.agent



traffic.main



Sequence Diagram. Draw a sequence diagram indicating the how a car updates its position. Show all the objects involved.


TimeServerQueue
RoadSegment
Car
run()
run()
updateCar()
enqueue
newDistanceTraveled()
obstacle
setPosition
setPosition

Time Summary. Provide a table breaking down the amount of time (in hours) you spent each week in the three areas. The table should look like this:

	Weeks			
	1	2	3	4
Design	7	2	2	0
Coding	5	5	7	1
Bug	0	1	2	2
totals	12	8	11	3

total hours
34

Notes on patterns

I used at least four different patterns to help solve problems that were confronted in this project.

My program relies off of one settingData object. When a user starts the simulation they can pick to run the default values or change them. The problem was that there are many different setting options. Furthermore, I wanted it set up to where if you make a change for one simulation it would stay that way until it was changed by the user again. In the beginning I had a way of getting this to work but after realizing that the builder pattern could be applied the code became much cleaner. The classes were the SettingsData and SettingsDataBuilder .

Another, issue that arise was getting my roadSegments linked with their correct intersection lights and cross-traffic roadSegments at these intersections. My roadSegments needed to know this information since it handled the finding the nearest obstacle for each car. I fixed this by setting up the roadSegments so that when they are created they use a builder pattern. The roadSegments are setup double linked by the road class and actually build together when model iterates through all the road segments to add to the graphics. Anyways, when the model adds the horizontal roads just the normal road or an intersection with a light. However, later when the vertical roads are added the correct cross traffic road segment is linked to and linked back to the newly created road. All this linking was needed to set the staged so the road-segments can now find the nearest obstacle for each car. This setup was a way to solve another problem. I didn't want all the classes to be interdependent off each other. Now RoadSegment uses the mediator pattern to update the car correctly without the car needing access to any information.

Moreover, the TimeSever class also uses a pattern to help solve a problem. The problem is that during each time step in the simulation it is expenses and not always necessary. My TimeSever class fixes this by using the observer pattern. The TimeSever will only update using the last states of the objects if during the next time step an object will change. Thus, if an update is needed TimeSever will let in observer know that a change occurred.

The last class that I will mention that uses patters is the Agent class. It makes use of the strategy pattern. The problem is that to implement the TimeSever some sort of interface needs to be used on the objects. This interface was the Agent class. The classes that implemented this Agent class were RoadSegments, Roads , LightObj.

Thus, now when the TimeSever gives the command all agents have a run function that will update the given object in respect to the simulation rules.

Successes and Failures

The projects problem itself was easy for me to solve. Getting the cars to move correctly and the lights to change color. I originally developed the idea to break the problem down into three changing pieces of data. The lights, cars and sources. First the lights updated which was easy enough to do. It was a little more difficult to adjust the cars correctly. To insure that the cars do not skip each other I had them move similar to a line at a store. The cars that would move first would be the car farthest from the source then work its way in order closest to the source. A little issue was that there were many road segments so you couldn't just have one ordered list. To fix this I have the roads segments linked up in a way that I could start at the road segment farthest away from the source and then work my way back. With all the road segments linked up in my mind the whole thing really would work as a line at a store. After the roads segments were done the sources would run and check if it was time to add a car and if there was space if so then it would add it to the first road.

Doing the above method I was able to get the project to work without really applying any of the material learned in the class. Then I realized my first major mistake. The task at hand was not to get the project to work, that is easy. The task at hand was to get the project to work using good standards and trying to apply the pattern in the class to the project.

So I really started the project after I had everything up and running correct. The first thing that was added was move from a while run loop to a timeserver. This was easy enough since the professor provided the code.

The next change occurred as I was looking over my code I found it easy and very convent to use a builder pattern on my settings data. I should note that I also wanted to look into using a singleton pattern mix with the builder pattern since only one object is need by the project at a time. However, to this day I couldn't seem to get an idea using both at the same time to work. I think you could say this was my second major failure and unlike the first one I couldn't really fix it.

One of the standards in class that the professor wanted us to do was not to use subclasses. My first method did use a subclass on the lights as a road segment. To fix this I removed the subclass and changed the way a road segment was created. It not used the builder pattern and if at an intersection it would link a light and a cross-

traffic road segment if needed. So all road segments had a link to a light and cross-traffic if they are null it is not an intersection. This design is much better since the light is no longer dependent off road segments, road segments are just dependent of the lights.

After that I did that I want to further reduce the number of dependencies so I mixed in the mediator pattern and had the road segments update the cars movements. This removed cars as an Agent extension and no longer had a run function. This process should have been easy for me with how my code was set up but it was not. I would like this to be noted as failure three. It took me about 2 hours to fix it. I should have had it set up much sooner.

Another point I would like to make is that I found coding a test for the roadSegement class was difficult. I failed to compose code that test it all correctly. I believe that test to be a little more difficult that the project itself.