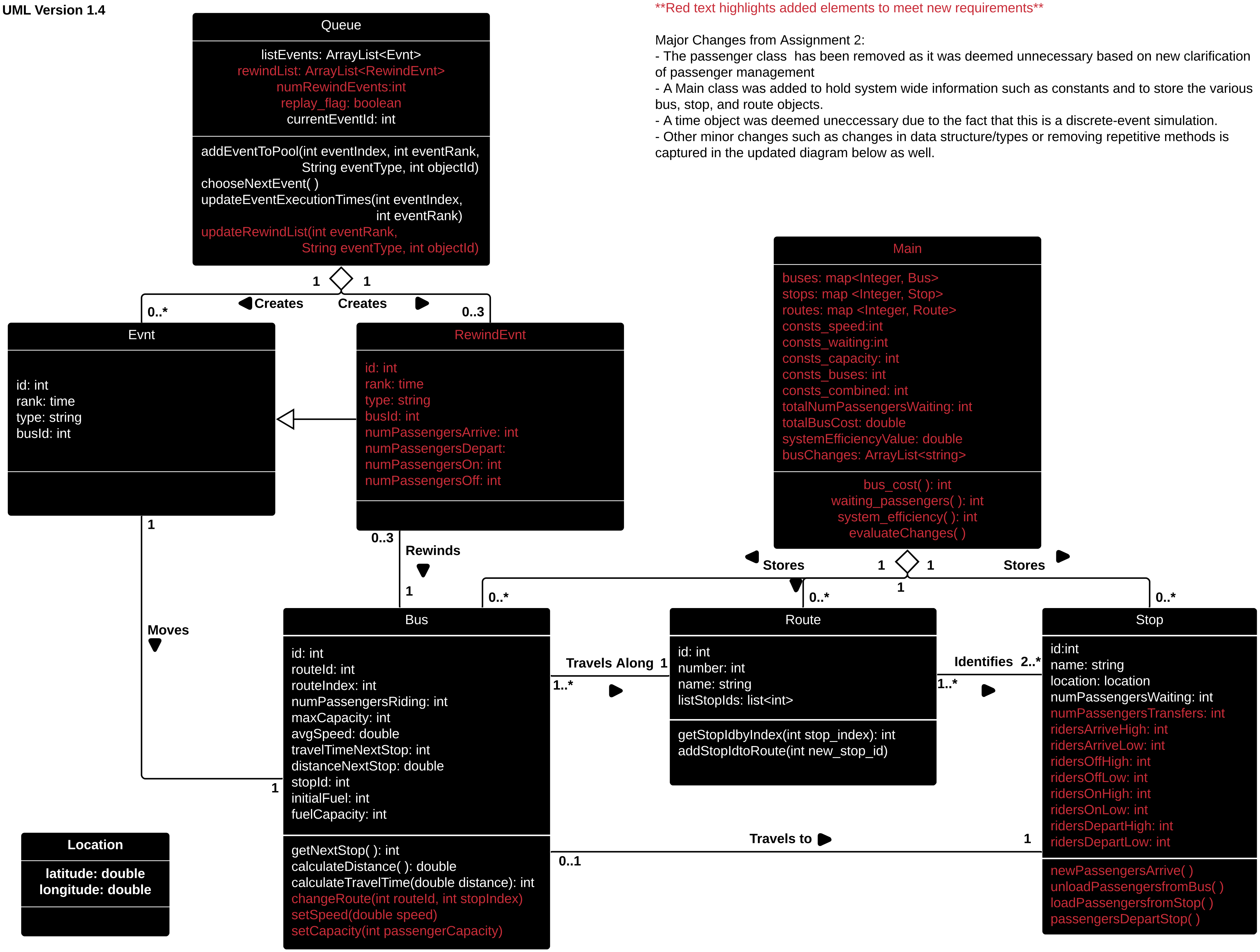
CS 6310 - Assignment #6 - Mass Transit Simulation System UML Class Diagram
Fall 2018 - Kristen Beaufait
UML Version 1.4

Red text highlights added elements to meet new requirements

Major Changes from Assignment 2:
- The passenger class has been removed as it was deemed unnecessary based on new clarification of passenger management
- A Main class was added to hold system wide information such as constants and to store the various bus, stop, and route objects.
- A time object was deemed uneccessary due to the fact that this is a discrete-event simulation.
- Other minor changes such as changes in data structure/types or removing repetitive methods is captured in the updated diagram below as well.



Bus Changes:
Bus changes will be stored in a string (ex "speed, busId, speedValue)") in the busChanges list. Each time a bus reaches a new stop, updateEventExecutionTimes() will call evaluateChanges( ) which will evaluate any strings in the busChanges list and update the bus information before the system updates the move_next_bus event.
(Route changes will include a routeid and stopid.)

Passenger Exchanges:
For all passenger exchange calculations, the random numbers used for distribution calculations will be calculated using a uniform distribution in the following manner:

Random random = new Random( )
random.nextInt((upperlimit - lowerlimit) + 1) - lowerlimit

This will generate a random number between lowerlimit and upperlimit (inclusively) using a uniform distribution.
After updateEventExecutionTimes( ) is called and all bus changes are applied, the passenger exchanges will take place before the move_next_bus event is updated.
1) newPassengersArrive( ) will calculate the number of new passenger arrivals at the stop and add them to numPassengersWaiting.
2) unloadPassengersfromBus ( ) will calculate the number of riders getting off the bus. The minimum of this result and numPassengersRiding will then be subtracted from numPassengersRiding and added to numPassengersTransfers. If a bus change occured with respect to bus capacity such that numPassengersRiding > maxCapacity for this bus after the unloading passengers have been moved to the transfers group, numPassengersRiding will continue to be decreased by one and numPassengersTransfers increased by one until numPassengersRiding <= maxCapacity.
3) loadPassengersfromStop( ) will calculate the number of waiting passengers loading the bus. The minimum of this result and (maxCapacity-numPassengersRiding) for this bus will be subtracted from numPassengersWaiting and added to numPassengersRiding for this bus.
4) passengersDepartStop( ) will calculate the number of passengers who depart the stop. The minimum of this result and the numPassengersTranfers will be subtracted from numPassengersTransfers. Any remaining depatures are subtracted from the numPassengersWaiting or any remaining transfers are added to the numPassengersWaiting.

System Efficiency:
system_efficiency( ) will call both waiting_passengers( ) and bus_cost( ) and then use the updated values in main to calculate system efficiency.
waiting_passengers( ) will loop through each stop in the system and add the numPassengersWaiting for this stop to totalNumPassengersWaiting.
bus_cost( ) will loop through each bus in the system and calculate the cost as a function of speed and capacity using the provided formula. This cost will be added to totalBusCost.

The system_efficiency function will be called in updateEventExecutionTimes( ) after the bus changes have been applied and the Passenger Exchanges have taken place. The efficiency value will then be updated and displayed as part of the normal graphical output. The GUI will include capability to update the constants used in system efficiency, and the system will use setter functions to change the stored values upon new input from the user.

Replays:
updateEventExecutionTimes( ) will create a RewindEvnt that contains the just-processed event and passenger information, and add this RewindEvnt to the rewindList. If size of rewindList > 3, the oldest rewind event will be removed from the list, and events 1 and two will shift down in the rewindList order.
(rewindList[2] = rewindList[1], set this EvntId = 2
rewindList[1] = rewindList[0], set this EvntId = 1
rewindList[0] = newRewindEvnt(eventid, busid, rank, type, numPassengersArrive, numPassengersDepart,
numPassengersOn, numPassengersOff)
when the replay button is selected by the user, a replay_flag is turned on. chooseNextEvent( ) will first check this flag, and if turned on, the most recent event (rewindList[0]) will become the next event to be processed. The bus from the event will then begin "rewinding" and traveling backwards to its previous stop (routeIndex of stop in saved event - 1). At its previous stop, the replay_flag will indicate for the numbers used in the distribution for passenger exchanges to be the saved numbers in rewindList[0] instead of being recalculated with the stop passenger exchange methods. The GUI will also have a dropdown menu to allow the user to select how many events (1-3) they wish to rewind. choose_next_event( ) will decrement this numRewindEvents by one each time a rewind event is scheduled until numRewindEvents ==0, and then the replay_flag will be turned off and normal operations resume.