The Class diagram has been updated to describe the changes based on the new requirements introduced in this assignment. At a high level, the changes are:
1. Bus Changes: new location, new_route_id, new_speed, and new_passenger_capacity are introduced as Bus attributes. These attributes help determine whether the client updated the Bus attributes since the last event.
2. Passenger Exchange: The upper and lower bounds of each database operation (e.g., riders_arrive_high and riders_arrive_low) have been added as attributes to the Database class.
3. System Efficiency: All constants related to the system efficiency calculation are added to the Simulation class. Moreover, a system_efficiency() operation was added to support this new calculation.
4. Replay: New data structures (stacks) have been added to track the previous bus stop, number of riders on the bus, and the number of riders waiting at the station. The FIFO Stack data structure was chosen because it helps to track previous state. A rewind operation was also added to support the new Replay event (not yet implemented in this assignment).

UML Version 1.4
Yifan Guo
October 28, 2018
Assignment 2: UML Diagram, CS 6310, Fall 2018

The stacks with names ending in "history" are provided to enable the Rewind functionality requested by the client. When the client wants to rewind the simulation n steps back, each stack will be popped n times and the popped items will become the new values of the attributes of the corresponding Bus object.

rewind() will undo the most recent move_bus event. Essentially, it will pop an item off of every aforementioned history stack. The popped item then becomes the Bus's new attribute value. OCL will ensure that the client cannot rewind more than three events at a time (OCL is not required as part of assignment 6).

executeNextEvent(Event e) invokes executeMoveBusEvent(e) if the type of Event e is "move_bus". This method becomes useful later when there are event types besides "move_bus".

In order to order events in the events priority queue in Simulation, we need to implement the compareTo method. This method returns -1 if the event's rank is less than, 0 if equal to, and 1 if greater than the other event's rank.

Simulation

events: PriorityQueue<Event> {ordered}
buses: Map<Integer, Bus>
routes: Map<Integer, Route>
stops: Map<Integer, Stop>
k_speed: double = 1.0
k_capacity: double = 1.0
k_waiting: double = 1.0
k_buses: double = 1.0
k_combined: double = 1.0

num_passengers_history: Stack<int> {ordered}
num_waiters_history: Stack<int> {ordered}
bus_location_history: Stack<int> {ordered}
bus_route_id_history: Stack<int> {ordered}

+ add_stop(id: int, name:string, riders: int, latitude: double, longitude: double)
+ add_depot(id: int, name: string, latitude: double, longitude: double)
+ add_route(id: int, number: int, name: string)
+ extend_route(route_id: int, stop_id: int)
+ add_bus(id: int, route: Route, location: int, initial_passengers: int, passenger_capacity: int, initial_fuel: int, fuel_capacity: int, speed: double)
+ add_event(time: int, type: string, id: int)
+ contains_events(): bool
+ executeNextEvent(e: Event): void
+ executeMoveBusEvent(e: Event): void
- waiting_passengers(): int
- bus_cost(): double
+ system_efficiency(): double
+ rewind(): void

Talks to -->

Database

riders_arrive_low: int
riders_arrive_high: int
riders_off_low: int
riders_off_high: int
riders_on_low: int
riders_on_high: int
riders_depart_low: int
riders_depart_high: int

+ridersArrive(): int
+ridersOff(): int
+ridersOn(): int
+ridersDepart(): int

Event

rank: int
type:string
object_id: int

+compareTo(other: Event): int

Moves -->

Bus

bus_id:int
route_id: int
location: int
num_riders:int = 0
passenger_capacity:int
speed:int
new_location: int
new_route_id: int
new_speed: int
new_passenger_capacity: int

+addRiders(howMany: int): int
+subtractRiders(howMany: int): int
+getNumAvailableSeats(): int

<-- Follows

Route

route_id:int
route_number:int
route_name:string
stop_ids: List<Integer> {ordered}

+ extend(stop_id: int): void
+ getRouteLength(): int
+ getStopIdAt(index: int): int

Stops At -->

Stop

num_waiters: int

+addWaiters(howMany: int): int
+subtractWaiters(howMany: int): int

Depot

stop_id: int
stop_name: string
latitude: double
longitude: double

Key assumptions:
1. All attributes are private
2. All attributes have getters and setters (not listed to save space).

Since changes to the bus attributes must not take effect until the bus reaches its next stop, those changes must be saved somewhere and used to update the bus attributes when the bus reaches its next stop. **new_location, new_route_id**, **new_speed**, and **new_passenger_capacity** serve this purpose.

When processing a move_bus event, the Bus's current attributes are updated with these new values via the methods **setLocation(), setRouteID(), setSpeed(), and setPassengerCapacity()** to ensure that the changes go into effect.

addRiders, subtractRiders, addWaiters, subtractWaiters return the actual number of passengers added or subtracted from the bus or stop. The return values are not used but helpful to have.