**Breve descrizione del progetto:**

Il file Bus.java all'interno del package main contiene la logica del bus e viene istanziato dalla classe SimConfigurationFrame.java in base alle scelte effettuate dall'utente in fase di inizializzazione. Come prima cosa il bus ottiene una lista di coordinate; questa contiene l'insieme di punti ordinati che l'autobus deve attraversare e costituisce il percorso dell'autobus.

Inoltre l'autobus deve essere in grado di riconoscere se delle coordinate corrispondono alle coordinate di una fermata di sua competenza. Di conseguenza in fase di inizializzazione viene creata una HashMap contenente proprio i punti relativi alle sole fermate che l'autobus deve effettuare.

Così facendo ogniqualvolta l'autobus avanza di un punto, deve interrogarsi per verificare se è arrivato o no ad una fermata, e a seconda dei casi eseguire le opportune operazioni. Notare che entrambe le liste (lista delle coordinate da attraversare e lista delle coordinate delle fermate) sono ottenute facendo parsing (1) dei relativi file .gpx presenti nella cartella gpx. Una volta inizializzate le strutture dati necessarie, l'autobus è pronto per partire. In particolare vengono inserite tutte le informazioni non mutevoli all'interno della SIB (per maggiori dettagli consultare il grafo dell'ontologia):

* **locationData** corrispondente all'autobus
* **personData** corrispondente all'autobus
* busId
* linea e massimo numero di posti disponibili

Queste informazioni sono statiche, e dunque non variano mai all'interno della stessa simulazione. Tuttavia assumeranno valori diversi gli attributi degli oggetti elencati precedentemente: ad esempio ogniqualvolta l'autobus avanza di una posizione, aggiornerà i literal “latitude” e “longitude” dell'oggetto **locationData**.

In modo analogo vengono aggiornati gli attributi di **personData** ad ogni fermata.

Le operazioni fondamentali dell'autobus si sviluppano attraverso tre cicli for: il ciclo più esterno corrisponde al numero di giorni da simulare, quello intermedio al numero di corse da simulare, mentre quello più interno corrisponde alla lista di punti sulla mappa che l'autobus deve attraversare.

Un esempio di scenario classico di esecuzione è il seguente:

* si ottengono le coordinate del prossimo punto da attraversare
* si controlla se le coordinate relative al punto corrispondono a quelle di una fermata:
  + in caso negativo si aggiorna semplicemente l'oggetto locationData, in modo da simulare il movimento dell’autobus;
  + in caso affermativo è necessario inserire nella SIB tutte le informazioni necessarie in accordo con l'ontologia, ad esempio si aggiornano gli attributi dell'oggetto personData, oppure viene indicato l'eventuale presenza del controllore.

Infine le ultime due operazioni fondamentali dell'autobus sono necessarie per avere una corretta simulazione.

A questo scopo infatti, ciascun autobus al termine della simulazione di una giornata, invoca il metodo SimulationConfig.getInstance().waitForBarrier().

Questa barriera serve per sincronizzare l’inizio di un nuovo giorno di simulazione frat tutti gli autobus (in modo che un autobus non inizi il giorno i+1 prima che gli altri autobus abbiano terminato il giorno i).

In modo del tutto analogo è presente una barriera al termine dell'intera simulazione; in questo modo solo quando tutti gli autobus hanno percorso l'ultima corsa dell'ultimo giorno di simulazione, viene sbloccato il main, che successivamente invoca il metodo opportuno sulla classe StatisticsFrame per calcolare e visualizzare a schermo le statistiche della simulazione.

La generazione delle persone che salgono e scendono ad ogni fermata avviene internamente alla classe Bus, così come l'emissione delle multe da parte del controllore quando quest'ultimo sale a bordo dell’autobus.

La logica di generazione delle persone fa uso di sei variabili (descendedRealPerson, ascendedRealPerson, descendedPayingPerson, ascendedRealPerson, realPerson e payingPerson) che tengono il conto delle persone, paganti e non, salite e scese ad ogni fermata, e di quelle paganti e non presenti in un determinato istante nel bus.

Ad ogni fermata (eccetto l'ultima) vengono generate in modo casuale le persone reali che salgono (dal metodo generateAscendingRealPerson()) tenendo in considerazione le persone reali attualmente a bordo e la massima capienza dell'autobus (2).

Successivamente si determina il numero di persone paganti fra quelle reali salite (metodo generateAscendingPayingPerson()) utilizzando la probabilità di evasione del biglietto (2).

Quindi si generano casualmente le persone reali che scendono (metodo generateDescendingRealPerson()) fra le persone reali presenti al momento sul pullman.

Infine viene determinato il numero di persone paganti scese (metodo generateDescendingPayingPerson()) fra le reali che scendono, tenendo in considerazione sia la probabilità di evasione del biglietto che il numero di persone reali e paganti presenti a bordo.

All'ultima fermata invece si fanno semplicemente scendere tutte le persone a bordo.

Dopo aver generato i flussi di salita/discesa, i valori delle variabili vengono scritti dall'autobus nella SIB inserendo nuove istanze per le entità **Affluence** (relativa a realPerson e payingPerson) e **GetOn** (per ascendedRealPerson e ascendedPayingPerson).

L'emissione delle multe avviene alla salita del controllore sul bus, cioè quando la variabile inspectorPresent assume il valore true.

Il controllore multerà tutte le persone sprovviste di biglietto presenti sul bus nella tratta fra la fermata in cui sale e la fermata successiva (è stato supposto che il controllore scenda alla fermata successiva a quella in cui è salito).

Il numero di multe effettuate è ottenuto quindi semplicemente dalla sottrazione fra le persone reali a bordo e quelle paganti, ed è salvato nella variabile fines.

Per questa operazione non è necessaria una query del controllore alla SIB in quanto salendo sul pullman è in grado di vedere "con i suoi occhi" chi ha pagato e chi no.

Dopo aver determinato il numero di multe effettuate, tale valore viene salvato nella SIB come nuova istanza dell'entità **Report**.

La visualizzazione in tempo reale si divide in due tipologie:

* posizione degli autobus sulla mappa
* dati di affluenza sull’autobus.

Per la prima tipologia è presente un handler che ottiene in subscribe dalla SIB le variazioni dell’istanza di **locationData** collegata al singolo autobus. Ottenendo quindi le nuove coordinate può spostare sulla mappa il marker che rappresenta l’autobus.

Anche per la seconda tipologia di visualizzazione a livello logico sarebbe necessaria una subscribe, ma per non sovraccaricare la SIB si è optato per l’esecuzione a polling della query. Tramite l’interfaccia grafica è possibile determinare il refresh rate da utilizzare.

Durante la fase di inizializzazione del sistema vengono inseriti nella SIB anche tutti i dati relativi alle fermate. Anche in questo caso i dati sono resi disponibili al programma tramite file gpx (1).

La gestione delle fermate avviene tramite la classe singleton BusStopManager, che ottiene e inserisce nella SIB le varie fermate (astratte tramite la classe BusStop).

Per la generazione dei controllori alle fermate si utilizzano due dati: il numero massimo di controllori presenti e la probabilità di comparsa ad una determinata fermata. La lista di tutte le fermate presenti nel sistema viene "mescolata" e scansionata. Per ogni fermata viene quindi applicata la probabilità di comparsa del controllore. Tutto ciò avviene fino al raggiungimento del numero di controllori massimi nel sistema. La generazione dei controllori avverrà quindi solamente all'avvio.

La classe BusStop si occupa di mantenere consistente il dato della presenza dell’ispettore fra la classe e la SIB.

Dal punto di vista delle utilità prodotte è presente un file con la configurazione del server (SIBConfiguration) dove è presente la SIB, una classe che contiene tutte le informazioni dell'ontologia e la classe Triple. Quest'ultima permette di aggiungere un livello di astrazione maggiore per l'inserimento delle triple all'interno della SIB. Fornisce quindi all'utente una vista sotto forma di soggetto, predicato e oggetto interfacciandosi con le API pre-esistenti che utilizzano solamente dei Vector<String>.

1. I parser SAX permettono di analizzare dei file xml per ricavare l'elenco dei punti di cui è composto il percorso di una linea e l'elenco delle fermate per ogni linea. Nel caso dei punti appartenenti al percorso il parser si limiterà ad estrapolare le coordinate di ogni punto, mentre per le fermate verrà considerato anche il tag contenente il nome della fermata.
2. Tutti i dati di simulazione sono disponibili all'interno del programma tramite il singleton SimulationConfig.  
   Una serie di dati iniziali sono richiesti all’utente all’avvio tramite l’interfaccia SimConfigurationFrame, che provvede invece a gestire il frame. Ulteriori dati di configurazione (come i posti massimi per autobus) sono invece presenti ma configurabili tramite codice.