Meilenstein A Roman Tabachnikov 2565209, Antoine Scheffold 2556024

Wir führen zwei neue Klassen ein. Train und TrainService.

Wir haben zur Connection ein Feld mit einem Lock hinzugefügt, und eine Methode getLock() die das Lock zurück gibt.

Wir haben zu Location noch eine Methode getParking() hinzugefügt, die True zurückgibt, wenn ein Parkplatz reserviert wurde, ansonsten False. Dazu noch eine Methode freeParking() die den Parkplatz wieder freigibt.

Die Klasse TrainService wird davor erstellt. TrainService ist unsere Fahrdienstleitung. Sie bekommt keine eigenen Informationen, sondern stellt nur Methoden bereit, die alle synchronized sind. Diese Methoden werden von den Trains aufgerufen.
ReserveConnnections(List Connections) soll dabei alle übergebenen Connections locken. Wenn dies nicht möglich ist, werden natürlich alle bereits gelockten Connections wieder freigegeben.
FreeConnection(Connection) gibt eine Connection frei (ruft unlock drauf auf) und führt notifyAll() aus, um wartende Züge zu wecken. Diese Methode ist mit synchronized gekennzeichnet, da sonst das notifyAll() eine Exception wirft.
waitingForReservedConnections(List Connections) soll Punkt (i) vom Lokführer 1x1 implementieren, also versucht diese Methode solange alle Locks zu bekommen, bis das geht, und falls nicht, ruft sie wait() auf und gibt alle bereits gehaltenen Locks frei. Muss synchronized sein.
Das wars auch schon mit unserem TrainService.

Die Klasse Train stellt dabei einen Zug dar, und erweitert Thread. Bei der Inititialisierung bekommt jeder Train seine TrainSchedule, den Recorder, die Map, und einen TrainService (den sich alle Trains teilen). D.h. es werden soviele Trains vom Simulator gestartet, wie es TrainSchedules gibt.
Train hat nur drei Methoden, und einen Konstruktor. Er merkt sich seine currentLocation.
Die Methode drive(List Connections) fährt von der Aktuellen Location die Liste der Connections ab, also fährt durch, und ruft nach dem ankommen an einer neuen Location immer trainService.freeConnection(connection) auf, und die Strecke wieder freizugeben.
findAndReserveParking(List Connections) soll dabei auf diesen connections beginnend vom Ziel erst die Location finden, die noch einen freien Parkplatz hat und diesen Reservieren(Methode getParking() auf Location), und die Connections von der Eingabe Liste entfernen, die mehr gefahren werden (also alle zwischen Parkplatz und Ziel).
Run() läuft in einer while(true) loop und terminiert, sobald die currentLocation == destination ist. Dabei wird das Zugführer 1x1 implementiert.

Die Schwierigkeit hier ist zum einen, die Connections Korrekt zu locken. Das Locken und unlocken macht beides nur die Klasse TrainService. Das Lock ist in der Klasse Connection gespeichert, und wird mit getLock() hergegeben. Hierbei brauchen wir kein synchronized, weil nix geschrieben oder verändert wird. Dann wird im TrainService immer nur lock.tryLock() aufgerufen, und im Falle dass nicht die Gesamte strecke gelockt werden konnte, werden alle bereits gehaltenen locks auch wieder freigegeben. TrainService bräuchte keine synchronized Methode, da die komplette Methode nur mit Lokalen Variablen arbeiten wird, bis auf die Connections, welche nur gelesen werden →keine Dataraces. Allerdings verlangt wait() und notifyAll() ein synchronized. Wir schauen, ob wir das irgendwie umgehen können, um vor allem die Methode waitingForReservedConnections nicht sequentiell laufen zulassen.
Somit funktioniert das Locken der Connections.
Die andere Schwierigkeit ist, Parkplätze entsprechend zu verwalten. Wir wollen hier ausnutzen, dass Züge nur an Bahnhöfen starten können (laut Forum) und rufen einfach vor jedem

drive(connections) (soll durchfahren) noch auf der currentLocation einfach leave() auf, das schaut ob dies ein Bahnhof war, dann wird nichts gemacht, und ansonsten wird einfach wieder zum Parkplatz eine stelle hinzugefügt (die Methode kann ja nur aufgerufen werden, nachdem ein Train auf dieser Location parkt). Wie geparkt wird, wurde schon erklärt. Parken und leave() müssen dabei synchonized sein, da sie auf einer gemeinsamen Variable operieren.

Zum Thema Deathlocks:
Immer wenn ein Lock für eine Connection genommen wird, aber nicht die ganze Strecke reserviert werden konnte, wird das lock wieder freigegeben.
Wenn eine Connection verlassen wird, wird sie auch freigegeben (freeConnection auf dem TrainService).
Es können somit nicht zwei Züge gleichzeitig fahren auf einer Connection.

Die Methoden zum Parken und Parkplatz verlassen sind beide synchronized, so dass nur ein Zug parken kann und die Variable Parkplätze somit verändern kann. Somit sind die Parkplätze in einem Monitor geschützt. Diese Methoden blockieren nicht, sie terminieren recht schnell.

Nun noch zu den beiden synchonized Methoden vom TrainService:
freeConnection(connection) ruft nur auf der Connection getLock().unlock() auf, und macht dann ein NotifyAll. Da sollte nichts blockieren können.
Die Methode waitingForReservedConnections(connections) versucht alle connections zu reservieren, wenn das nicht geht, wird wait() aufgerufen, und das Lock wieder hergegeben. Da Blockiert auch nichts, und aufgeweckt werden die immer, wenn ein Lock wieder freigemacht wird.