Universidade de Aveiro

Departamento de Eletrónica,
Telecomunicações e Informática

Sistemas Distribuídos
Relatório Trabalho Prático nº 1

Rapsódia no Aeroporto
Turma 1 - Grupo 2

Autores:

60456 - Tiago Soares

60089 - David Simões

Supervisor:

Prof. Óscar Pereira

23 de Março de 2014

Conteúdo

Introdução

O objectivo deste trabalho prático consiste em construir uma simulação de atividades ligadas ao desembarque de passageiros no aeroporto, algures nos arredores da cidade de Aveiro, usando um dos modelos estudados de sincronização e de comunicação entre *threads*: monitores. A nossa solução final deve ser uma solução descentralizada com múltiplas regiões de partilha de informação, implementada em JAVA, passível de execução em Linux e facilmente adaptável a sistemas distribuidos.

Uma thread é uma sequência de instruções que compete com outras threads pelo acesso aos recursos do sistema. Um monitor é um dispositivo de sincronização que permite, se implementado corretamente, que as threads disfrutem de exclusão mútua. Os monitores também disponibilizam mecanismos de adormecimento de threads (no caso de Java, o método **wait()**) assim como mecanismos para as acordar (em Java, os métodos **notify()** e **notifyAll()**).

Foram tomadas em conta as seguintes condições:

1. Vão realizar-se K chegadas de aviões, envolvendo cada uma delas o desembarque de N passageiros;
2. Cada passageiro transporta no porão do avião de 0 a M peças de bagagem;
3. O autocarro de transferência entre terminais tem uma lotação de T lugares;

Assumimos por defeito que há cinco chegadas de aviões, cada uma envolvendo o desembarque de seis passageiros, transportando um máximo de duas peças de bagagem no porão do avião, e que a lotação do autocarro de transferência entre terminais é de três passageiros.

Capítulo 1

Entidades

1.1 Threads e Bagagens

Inicialmente começámos por definir que entidades seriam threads e monitores no nosso projecto. De imediato concluímos que faria sentido que os passageiros, bagageiro e motorista seriam as nossas threads, ou seja, classes que fazem extensão à classe *Thread*, dado que estes são os intervenientes do nosso problema (são os que realizam as diversas actividades). Assim, podemos enunciar as nossas threads da seguinte forma:

1. Os **passageiros**, que contêm malas e podem ter terminado a sua viagem ou continuar em trânsito;

2. O **bagageiro**, que descarrega as bagagens do avião (quando este aterra) e as transporta para as zonas de recolha de bagagens ou de armazenamento temporário;

3. O **motorista** do autocarro, que transporta os passageiros em trânsito entre as zonas de transferência de terminal;

Ao longo do desenvolvimento do projecto também notámos que seria necessário criar uma nova classe (não uma thread) que representasse as bagagens, esta contendo duas variáveis: **ownerId** que corresponde à identificação do dono da bagagem e **ownerTravelling** que é um tipo de dados *boolean* e que indica se o dono está em viagem ou não.

1.2 Zonas e Monitores

Quanto aos monitores, decidimos que estes seriam as zonas do aeroport com as quais as threads interagem e onde é necessário haver informação partilhada. Conseguimos isolar 8 zonas: Arrival Zone, Luggage Claim, Storage Zone, Guichet, Arrival Transfer Zone, Departure Transfer Zone, Exit Zone e Entry Zone. No entanto, nem todas estas zonas têm de ser monitores. Conseguimos unir zonas com caracteristicas semelhantes, como as zonas de Entry e de Exit, ou as zonas relacionadas com bagagens (Luggage Claim, Storage e Guichet).

Deste modo, os monitores são os seguintes: *ZoneArrival* - zona de desembarque; *ZoneEntryExit* - entrada no terminal de embarque e saída do terminal de desembarque; *ZoneLuggage* - zona de recolha de bagagens, zona de armazenamento temporário de bagagens e guichet de reclamação de bagagens perdidas; *ZoneTransferArrival* - zona de transferência no terminal de desembarque e *ZoneTransferDeparture* - zona de transferência no terminal de embarque.

Criámos ainda um monitor designado por *General* que permite incorporar um ficheiro de *logging* que descreve de um modo conciso e claro a evolução do estado interno das diferentes entidades envolvidas. Desta forma, no final de todas as operações, é emitido um relatório completo das actividades desenvolvidas num ficheiro de saída ("log.txt"por defeito ou definido por argumento).

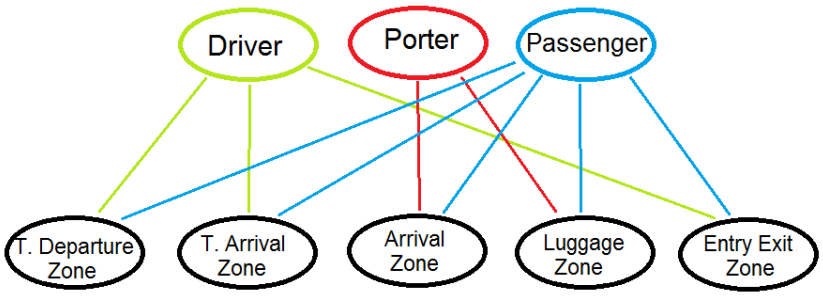

Figura 1.1: Esquema relativo à interação entre Threads e monitores

De acordo com o esquema anterior é possível verificar as relações entre as threads e respectivos monitores. Além destas relações, todas as entidades (monitores e threads) com exceção da ZoneArrival interagem com o monitor General, para tratar do logging correto da simulação.

Foram criadas interfaces para cada Monitor consoante quem interagia com este para garantir que as threads não pudessem alterar informação que não estivesse relacionada com as próprias. As diferentes Zonas também utilizam interfaces para interagir com o monitor General.

Capítulo 2

Main

A thread Main vai simular a chegada dos vários aviões, assim como inicializar os vários Monitores e as threads do aeroporto.

Toma como argumentos os valores de K (número de aviões), N (número de passageiros por avião), M (número máximo de malas que um passageiro pode ter), T (lugares disponiveis no autocarro) e LogName (o nome do ficheiro de log). Estas variáveis têm valores por defeito (5, 6, 2, 3, log.txt, respetivamente) mas, caso os argumentos estejam definidos, vão ser alteradas e criar simulações porventura muito diferentes.

A main está encarregue de criar os 5 monitores de zonas e o monitor de logging, que vai depois enviar nos construtores das threads que deles precisarem. Cria e inicia as threads do Driver e do Porter e, dentro de um ciclo com K iterações, simula as chegadas dos aviões, criando N passageiros por iteração, com 0 a M malas por passageiro. Uma mala tem uma chance de 20% de ser perdida e não ser colocada no Plane's Hold (caso o passageiro não esteja em trânsito; se estiver, a mala não é perdida). Os passageiros têm uma chance de 50% de estar em trânsito.

A cada ciclo, a main espera que todos os passageiros tenham terminado e, depois de todos os ciclos, espera que o Driver e o Porter terminem também. Aí, indica ao monitor de logging que a simulação terminou e que o ficheiro deve ser guardado.

Capítulo 3

Ciclos de Vida

A partir dos diagramas abaixo, podemos retirar que as transições entre estados correspondem a funções que as threads vão executar concorrentemente. Quando as threads são started(), começam no seu estado inicial e percorrem o diagrama até acabarem a sua execução.

Por questões de teste, entre cada mudança de estado, todas as threads fazem um sleep de 0 a 100 milisegundos. Isto foi feito para esforçar o sistema e garantir que não havia cenários de deadlock.

3.1 Porter

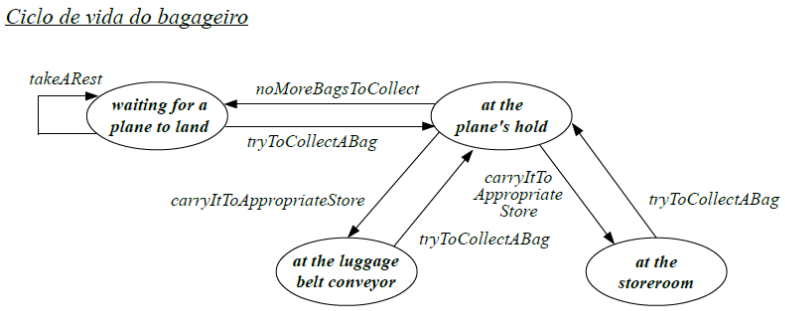

Figura 3.1: Esquema relativo ao Ciclo de vida do Bagageiro

A thread *Porter* executa as seguintes operações:

1. takeARest() - o bagageiro descansa até ao desembarque de todos os passageiros (invoca o método takeARest() que se encontra no monitor ZoneArrival);

2. noMoreBagsToCollect() - quando já não há mais malas para recolher, o bagageiro vai descansar;

3. tryToCollectABag() - quando o bagageiro recolhe as bagagens do porão do avião, uma a uma, e distribui-as pelas zonas de recolha de bagagens e de armazenamento temporário, conforme se trate de bagagens pertencentes a passageiros que terminam a sua viagem no aeroporto ou que estão em trânsito (invoca os métodos getLugNumber() e getLug() que se encontram no monitor ZoneArrival e noMoreBags() que se encontra no monitor ZoneLuggage);

4. carryItToAppropriateStore() - quando o bagageiro distribui as bagagens pertencentes a passageiros que ainda estão em trânsito ou que já terminaram a sua viagem (invoca os métodos putABagSR() ou putABagLBC(), respectivamente, que se encontram no monitor ZoneLuggage);

3.2 Passenger

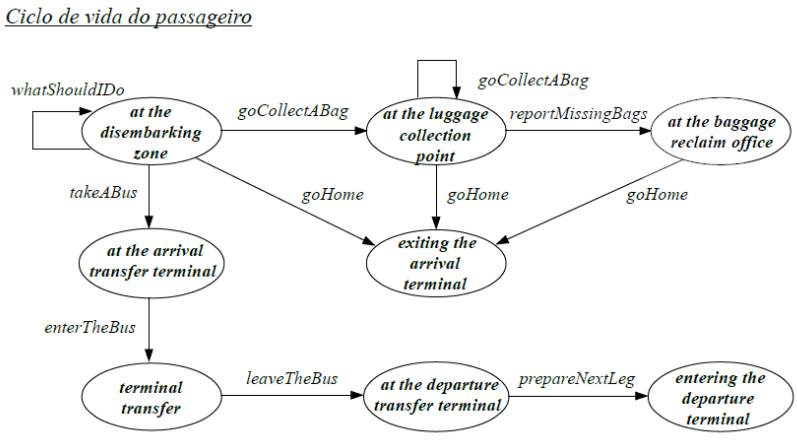

Figura 3.2: Esquema relativo ao Ciclo de vida do Passageiro

A thread *Passenger* executa as seguintes operações:

1. getJourney() - verifica se a viagem chegou efetivamente ao fim ou não;
2. whatShouldIDo() - decide o que o passageiro vai fazer dependendo das situações: se este não está em viagem e tem bagagens então vai recolhe-la; se não está em viagem mas não tem qualquer bagagem então vai-se embora caso contrário vai para a fila de espera para apanhar o autocarro (invoca o método iGotHere() que se encontra no monitor ZoneArrival);
3. goCollectABag() - quando o passageiro já não está em viagem e tem bagagens então vai recolhe-las e caso perca uma delas então terá de reportar o seu desaparecimento (invoca o método getLuggage() que se encontra no monitor ZoneLuggage);
4. reportLuggageMissing() - quando o passageiro reporta o desaparecimento de uma das bagagens perdidas (invoca o método reportLuggageMissing(id, bags) que se encontra no monitor ZoneLuggage)
5. goHome() - quando o passageiro se vai embora porque já não está em viagem e não possui qualquer bagagem (invoca o método goingHome(id) que se encontra no monitor ZoneEntryExit);
6. takeABus() - quando o passageiro deseja apanhar o autocarro (invoca o método takeABus(id) que se encontra no monitor ZoneTransferArrival);
7. enterTheBus() - quando o passageiro entra no autocarro e espera até que este termine (invoca o método finishBusTrip() que se encontra no monitor ZoneTransferDeparture);
8. leaveTheBus() - quando o passageiro abandona o autocarro (invoca o método leaveBus(id) que se encontra no monitor ZoneTransferDeparture);
9. prepareNextLeg() - quando o último passageiro de cada voo acorda todos os passageiros em EXITING_THE_ARRIVAL_TERMINAL ou ENTERING_THE_DEPARTURE_TERMINAL (invoca o método goingHome(id) que se encontra no monitor ZoneEntryExit)

3.3 Driver

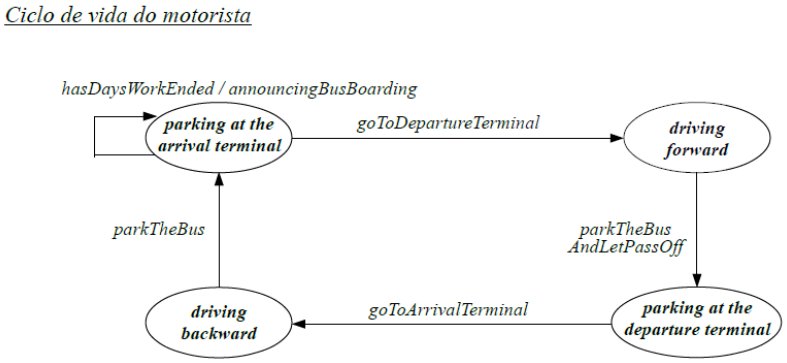

Figura 3.3: Esquema relativo ao Ciclo de vida do Motorista

Por último, a thread *Driver* executa as seguintes operações:

1. void goToDepartureTerminal() - quando o driver parte para a zona de Transferência de embarque (invoca o método depart() que se encontra no monitor ZoneTransferArrival);

2. void parkTheBusAndLetPassOf() - quando o driver estaciona o bus no terminal de embarque (invoca o método park(people) que se encontra no monitor ZoneTransferDeparture);

3. void goToArrivalTerminal() - quando o driver parte para a zona de Transferência de desembarque (invoca o método depart() que se encontra no monitor ZoneTransferDeparture);

4. void parkTheBus() - quando o driver estaciona o bus no terminal de desembarque;

5. boolean hasDaysWorkEnded() - verifica se o dia do motorista chegou ao fim ou não;

6. void announcingBusBoarding() - o driver espera até que hajam passageiros suficientes à espera de entrar no bus e acorda-os para estes entrarem (invoca o método announceBusBoarding() que se encontra no monitor ZoneTransferArrival);

Bibliografia

Referências

[1] *Introduction to Java threads*. URL: http://www.javaworld.com/article/2077138/java-concurrency/introduction-to-java-threads.html.

[2] *Monitor (Java Platform SE 7)*. URL: http://docs.oracle.com/javase/7/docs/api/javax/management/monitor/Monitor.html.

[3] *Slides Teóricos e Exemplos Práticos - SD*. URL: Elearning%20UA.

[4] *Thread (Java Platform SE 7)*. URL: http://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html.