DEPARTAMENTO DE INFORMÁTICA
MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA
3 DE JANEIRO DE 2015

UBER - SERVIÇO DE TÁXI

RELATÓRIO DO TRABALHO PRÁTICO
SISTEMAS DISTRIBUÍDOS

Trabalho realizado por:

André Ribeiro Ferraz da Cunha Santos a61778
Mariana Imperadeiro Carvalho a67635

Índice

1. Introdução

Neste relatório é apresentado o trabalho desenvolvido para a disciplina de Sistemas Distribuídos. O objetivo do trabalho consiste na implementação em Java de um serviço de gestão de taxis com capacidade para gerir varios passageiros e taxistas em simultâneo, um pouco semelhante aos serviços oferecidos pela empresa Uber. Para conseguir tal objectivo, a aplicação recorre a mecanismos de controlo de concorrência e ao uso de sockets, para a comunicação entre os seus passageiros e condutores.

2. Nota terminológica

A palavra “Cliente” no âmbito deste projeto é ambígua, pois tanto pode significar a aplicação Cliente que comunica com uma aplicação servidor, como pode significar um cliente no sentido de passageiro do serviço de taxis que a aplicação pretende simular. Para esclarecer ambiguidades, sempre que neste relatório se referir Cliente, esta-se a referir o primeiro significado apresentado, sendo a palavra “Passageiro” reservado para o segundo.

3. Arquitetura da Aplicação

3.1 Comunicação Cliente – Servidor

A aplicação baseia-se na gestão de um serviço de táxi, um pouco semelhante aos serviços oferecidos pela empresa Uber.

Os clientes comunicam com o servidor através de sockets TCP. Quando é iniciado, o servidor abre um socket associado a uma porta específico, ficando à espera que os clientes se liguem. Os clientes por sua vez criam um socket com o qual se tentam ligar a essa porta aberta pelo servidor. Quando recebe uma conexão, o servidor inicia uma nova thread para atender os pedidos que vierem desse cliente, continuando à escuta de novas conexões na sua thread principal.

3.2 Funcionalidades Servidor

Como referido anteriormente, o servidor cria uma nova thread para atender cada cliente. O “trabalho” desta nova thread está especificado numa instância da classe RunnerServidor, que implementa a interface Runnable e é passada ao constructor de criação da thread. Sendo responsável pelo atendimento de um cliente esta classe tem acesso aos streams de input e output do socket de comunicação com o cliente,lendo do stream de input do socket comandos correspondentes a pedidos de interação dos utilizadores com o sistema e escrevendo respostas no stream de output desse mesmo socket.

Os comandos que o servidor está preparado para responder disponibilizam as seguintes funcionalidades ao cliente:

- Login de utilizadores
- Registo de utilizadores
- Solicitar Viagens (por passageiros)
- Indicar Localização (por condutores)
- Consulta de dados do utilizador
- Edição de dados do utilizador
- Outras verificações auxiliares (p.e: o passageiro pode ser condutor?)

3.3 Funcionalidades Cliente

O cliente desta aplicação tem como funcionalidade disponibilizar apenas estabelecer a comunicação com o servidor e disponibilizar uma interface textual com a qual passageiros e condutores possam interagir com o sistema.

3.4 Classe Uber

Esta classe serve de suporte ao servidor, onde está guardada toda a informação sobre os utilizadores e viagens em progresso. Quando um servidor recebe um pedido, esse pedido resulta invariavelmente na consulta/alteração de estado de uma instância desta classe. Todas as threads que o servidor cria realizam as operações sobre uma instância desta classe, comum a todas as threads. Visto várias threads terem acesso a uma instância desta classe, para garantir o correcto funcionamento da aplicação esta classe usa técnicas de controlo de concorrência que se referem na secção seguinte.

Esta classe tem ainda a particularidade de ter duas *inner class* privadas: UtilizadorUber e TripUber, que são subclasses de Utilizador e Viagem. Estas subclasses apenas adicionam variáveis de instância correspondentes a variáveis de condição, cujo sentido de existência será claro mais à frente.

4. Controlo concorrência

4.1 Lock Uber

O controlo de concorrência feito pela classe Uber recorre a um ReentrantLock, que protege a grande maioria dos métodos. Regra geral, os métodos tentam adquirir o lock no início e libertam-no quando terminam (falaremos nas excepções mais à frente). Visto que todos os métodos necessitam de aceder às estruturas de dados, essa foi a solução encontrada para proteger essas estruturas do acesso concorrente.

4.2 Variáveis de condição (VC)

As variáveis de condição da aplicação foram todas criadas sobre o lock da uber, destinadas a gerir a atribuição de passageiros a condutores e vice-versa.

Quando um passageiro pede uma viagem, ele tenta verificar se há condutores disponíveis. Caso não haja, é posto a esperar na variável de condição *semCondutor.* Esta VC é também variável de instância da uber e destina-se por isso a colocar sobre ela todos os passageiros à espera porque não há condutor. Quando é acordado, significa que há pelo menos um condutor, então é escolhido o condutor mais próximo e o condutor escolhido (que estava à espera de passageiros) é acordado por um signal à variável de condição *noPassenger*. Esta VC é específica ao condutor. É depois colocada numa estrutura da uber a informação da viagem, que pode ser consultada pelo código do lado do passageiro e condutor e que contém 3 VC: *driverNotAtDeparture*, *passengerDidntConfirm* e *didntArriveAtDestination*. Depois de atribuida a viagem ao passageiro e de esta ter sido colocada na estrutura intermédia, o passageiro espera que o condutor chegue ao local de partida na variavel de condição *driverNotAtDeparture* da viagem. Quando o condutor chega ao local de partida, avisa o passageiro fazendo signal a essa VC e fica à espera que o passageiro confirme a viagem na VC *passengerDidntConfirm* (recorde-se que tanto o código do passageiro como do condutor têm acesso à viagem, e por isso às suas variáveis de condição). O passageiro confirma a viagem com signal e fica à espera que esta termine na VC *didntArriveAtDestination* . Cabe ao condutor acordar o passageiro nessa VC quando tiver chegado ao destino e a viagem terminado.

5. Conclusões e pontos chave

O trabalho entregue os requisitos propostos no enunciado, acrescentando ainda algumas funcionalidades extras como a capacidade de edição de dados dos utilizadores e a mudança de papel entre passageiro e condutor que é permitida a um utilizador sem necessidade de novo login. Embora o âmbito do projecto fosse a comunicação cliente-servidor e o controlo de concorrência, foi tida alguma atenção também ao tratamento de excepções e encapsulamento de dados.