Unidade Curricular de Sistemas Distribuídos

Aplicação: *"Gestor de Leilões"*

Ano Letivo de 2016/2017

Bruno Nascimento

A67647

Tiago Araújo

A71346

Ângelo Teixeira

A73312

Vitor Rodrigues

A67703

Janeiro 2017

Introdução

As aplicações distribuídas lidam normalmente com vários utilizadores, e como tal devem estar preparadas para responder aos vários pedidos por eles feitos simultaneamente. Assim, neste tipo de sistemas, o controlo de concorrência torna-se essencial para que o sistema tenha um bom desempenho, mantendo ao mesmo tempo a consistência dos dados.

Neste trabalho procuramos explorar as funcionalidades do modelo cliente servidor com comunicação orientada à conexão via TCP, através do desenvolvimento de um servidor multithreaded que funciona como intermediário na comunicação entre clientes, utilizando a linguagem de programação Java para implementar tanto o servidor como o cliente.

O tema desenvolvido consiste na implementação de uma aplicação distribuída que permite a gestão de um serviço de leilões, com a qual os utilizadores interagem, podendo adicionar leilões e fazer licitações a leilões já existentes.

Estrutura da aplicação

A aplicação é constituída pelas seguintes classes principais:

Cliente: Esta classe implementa uma interface de utilizador, comunicando com o servidor através de sockets TCP, enviando-lhe pedidos (input) do utilizador e recebendo as respetivas respostas. Esta classe faz uso das classes "BufferedReader" e "PrintWriter" para trocar (enviar e receber) mensagens com o servidor, bem como ler input do utilizador.

Servidor: Esta classe implementa um servidor, que recebe conexões de clientes, mantendo em memória toda a informação relativa aos utilizadores e respetivos leilões, sendo esta informação partilhada por várias threads (duas por cada cliente).

Utilizador: Esta classe representa um utilizador do sistema, cuja informação abrange o seu nome, a sua password, e também, a indicação de que está online/offline.

Utilizadores: Esta classe possui uma coleção de utilizadores organizados por nome de forma alfabética e fornece operações para atuar sobre esse conjunto, mais especificamente métodos para adicionar utilizadores bem como um método para efetuar login e outro para efetuar logout de um dado utilizador.

Leilao: Esta classe representa um leilão, cuja informação abrange o item a ser leiloado, a maior oferta, o maior licitador bem como todos os licitadores, qual o vendedor, e se encontra-se ou não activo . Contém também métodos como "addLicitacao" que serve para adicionar uma nova licitação e "fecharLeilao" que termina o respectivo leilão.

Leiloes: Esta classe possui uma coleção de leilões e fornece operações para actuar sobre esse conjunto tais como para adicionar ou terminar um leilão bem como para adicionar uma licitação.

LeiloesLog: Esta classe contém todos os leilões e é utlizada para notificar todos os licitadores de um determinado leilão quando este é terminado.

Leitor: A classe leitor representa um objecto runnable que corre numa thread do cliente para ler dados do servidor.

NotificacaoHandler: Esta classe corre numa thread e passa a maior parte do tempo adormecida, acordando quando um leilão é terminado e avisa os seus licitadores.

Handler: Esta classe corresponde a uma thread do servidor que atende um cliente, tratando do registo e autenticação de utilizadores e gestão de tarefas de abertura, licitão, listagem e término de leilões.

Para além das classes referidas, há ainda sete classes de exceção, cujos nomes permitem facilmente deduzir o seu propósito: UtilizadorNaoExisteException, UtilizadorJaExisteException, UtilizadorOnlineException, UtilizadorOfflineException, PasswordIncorretaException, VendedorErradoException e LeilaoInexistenteException.

Implementação das funcionalidades pretendidas

Registo de utilizador

O servidor solicita ao cliente o nome e palavra passe e chama o método "adicionarUtilizador" na classe Utilizadores que verifica se o nome já existe na coleção de utilizadores, caso não exista, cria um novo Utilizador inserindo-o na coleção.

O método "adicionarUtilizador" da classe Utilizadores é synchronized, já que invocações simultâneas deste método poderiam ser conflituosas, visto que duas threads poderiam ambas confirmar a inexistência, na coleção de utilizadores, de um mesmo nome que elas pretendem inserir e ambas criarem um utilizador, inserindo-os na coleção, mas ocorrendo um "overwrite" de tal forma que apenas um novo utilizador fará parte da coleção, nomeadamente o último a ser inserido. Desta forma, torna-se necessário obter o lock (neste caso, optou-se pelo lock implícito) da instância da classe Utilizadores antes de efetuar a operação de registo.

Login

O cliente comunica ao servidor o nome e palavra passe, e este por sua vez chama o método "logIn" da classe Utilizadores, que por sua vez, caso verifique a existência de um utilizador com o nome referido chama o método "logIn" da classe Utilizador sobre esse utilizador que verifica o estado do utilizador e, se a password está correta, procedendo à atualização do estado da ligação.

O método "logIn" da classe Utilizador é synchronized, já que duas threads podem tentar autenticar-se com o mesmo nome, sendo que ambas confirmam a existência do utilizador na coleção através do método "logIn" da classe Utilizadores e, nesse caso, é necessário impedir atualizações simultâneas do estado desse utilizador, que é conseguido obtendo o lock implícito do objeto que representa o utilizador em questão.

Logout

Se o utilizador se encontrar offline é lançada a excepção "UtilizadorOfflineException" caso contrário é utilizado o método "logOut" da classe Utlizadores para terminar a sessão, que faz uso do lock existente nessa classe garantindo assim que não há vários clientes a alterarem a coleção armazenadora dos utilizadores ao mesmo tempo.

Adicionar leilão

Primeiro é verificado se o utilizador se encontra online, caso se verifique, o servidor solicita ao utilizador o item e o seu valor base e utiliza essa informação para criar e inserir o leilão numa coleção através do método "adicionarLeilao" da classe Leiloes, que faz uso do lock existente nessa classe, de modo a que vários utilizadores não tentem inserir ao mesmo tempo na coleção. Após a criação e armazenamento do leilão acabado de criar, o servidor fornece ao utilizador o seu id.

Listar todos os leilões

Lista todos os leilões activos de todos os utilizadores através do método "printLeilao".

Adicionar Licitação

Caso o utilizador se encontre online é-lhe solicitado pelo servidor o id do leilão que contém o item ao qual quer fazer uma oferta e o valor desta. Sendo o id de um leilão existente e activo é utilizado o método "adicionarLicitacao" da classe Leiloes para adicionar a licitação ao leilão corespondente, de notar que o método "adicionarLicitacao" faz uso do lock existente nessa classe, de modo a que não sejam inseridas várias licitações na coleção ao mesmo tempo.

Fechar Leilão

Encontrando-se o utilizador online, fornece ao servidor o id do leilão que quer terminar e caso este exista, se encontre activo e pertença ao primeiro, será terminado através do método "fecharLeilao" da classe Leiloes que faz uso do lock existente nessa classe de modo a não ser possível a vários utilizadores alterarem a estrutura responsável por armazenar os leilões simultaneamente, sendo todos os licitadores desse leilão notificados do seu encerramento.

Conclusão

Durante o desenvolvimento deste trabalho pudemos pôr em prática muito do que estudámos durante este semestre. Usando os mecanismos de controlo de concorrência do java conseguimos assim criar uma aplicação distribuída capaz de garantir o bom funcionamento de um serviço de leilões tal como esperado. Surgiram algumas dúvidas durante o desenvolvimento, apesar disso terminámos com uma aplicação funcional que cumpre os requisitos propostos e por isso achamos que este trabalho foi bem sucedido e nos foi muito útil para melhor interiorizar a matéria lecionada