Universidade do Minho
Escola de Engenharia

Unidade Curricular de Sistemas Distribuídos

Gestor de deslocações

Daniel Rodrigues a67634

José Carlos Pedrosa Lima de Faria a67638

Gil Gonçalves a67738

Tiago Cunha A67707

03, de Janeiro de 2016

Índice

Introdução 2

Servidor 3

Cliente 3

Handler 4

Central 4

Serviço 5

PacoteDados 5

UtilizadorCondutor 6

UtilizadorPassageiro 6

Conclusão 7

Introdução

Na Unidade Curricular de Sistemas Distribuídos foi proposto a implementação de um gestor de deslocações.

No gestor de deslocações existem dois tipos de clientes:

- Os clientes que estão a procura de um veículo que lhes permita levar ao destino pretendido, sendo no final este serviço pago mediante do preço atribuído pelo condutor.
- Existem também os clientes que queiram registar o seu veículo para poderem oferecer os seus serviços aos clientes que desejarem boleia.

Pretendia-se o uso de um cliente escrito em Java, onde os utilizadores poderiam interagir, intermediados por um servidor *multi-threaded* também escrito em Java, recorrendo a comunicação via TCP.

Neste relatório são explicadas algumas tomadas de decisão e o funcionamento do gestor de deslocações através de algumas imagens exemplificadoras do seu uso.

Classes

Numa primeira análise do enunciado do projeto verificamos com alguma facilidade a necessidade de criar certas classes principais. Depois, à medida que fomos avançando na elaboração do projeto, tornou-se obrigatório criar mais algumas classes que nos ajudariam a completar algumas informações. Assim, terminámos o projeto com as seguintes classes: Servidor, Cliente, Handler, Central, UtilizadorCondutor, UtilizadorPassageiro, Servico, Utilizador e PacoteDados.

Servidor

A classe Servidor é onde se tratam novas ligações de clientes ao servidor e inicia-se os dados da aplicação. Sempre que existir uma nova conexão ao servidor este cria uma *thread*, onde lhe passa como parâmetros o *socket* do cliente associado e a ligação aos dados que estão na memória.

Para termos um fácil acesso ao conjunto de ações que um cliente pode pedir, foram criadas várias variáveis finais com o nome das ações. Facilitando portanto, na criação dos pacotes de dados (PacoteDados).

```
public class Servidor {
    private static final int porta = 55000;
    private static Central k = null;
    //Passageiro

    public static final String REGISTAR_PASSAGEIRO = "RegistarPassageiro";
    public static final String ENTRAR_PASSAGEIRO = "EntrarPassageiro";
    public static final String SAIR_PASSAGEIRO = "SairPassageiro";
```

Cliente

A classe Cliente é onde é oferecida a interface ao utilizador. Esta conta com uma série de variáveis que permitem a interação com o utilizador e as ligações ao servidor através da classe Handler.

```
public class Cliente {
    private final static int porta = 55000;
    private final static String ip = "localhost";
    private static Socket clienteSocket;

    public static Scanner in = new Scanner(System.in);
    public static ObjectOutputStream o = null;
    public static ObjectInputStream i = null;
    public static HashMap<String, String> hash;
    public static String nick = null;
    public static PacoteDados p;
```

As variáveis ip, port e clienteSocket (Socket) são os elos de ligação ao servidor (Servidor).

As variáveis **o, i e p** são as variáveis que permitem a troca de mensagens entre o utilizador e o servidor. A variável **p** guarda o que o cliente deseja fazer e mais tarde a classe handler vai tratar desse informação, que depois enviara para a classe central. A variável **o** permite que o cliente consiga trocar mensagens com o servidor, isto claro com a ajuda da classe handler e a variável **i** permite que o cliente recebe mensagens enviadas do servidor, também com a ajuda da classe handler. É também com ajuda do handler que os utilizadores podem trocar mensagens entre si.

Handler

A classe *Handler* extende a classe *Thread* e tem como principal objetivo tratar individualmente um cliente e os seus pedidos. Quando um cliente faz um pedido ao servidor, o pacote de dados enviado é tratado nesta classe e mediante desse pedido chamará os métodos necessários a realização desse pedido.

```
public class Handler implements Runnable {
    private final Socket s;
    private final Central sk;
    private ObjectInputStream sInput;
    private PrintWriter sOutput;
```

O *Handler* consegue comunicar com a central, onde está a informação relevante para a concretização dos pedidos feitos por cada cliente. Assim, demonstrado, o *Handler* é uma espécie de intermediário.

Central

A classe Central é a classe mais importante porque é nesta classe que será guardada toda a informação relativa aos vários utilizadores. Para guardar esta informação foram criadas 3 HashMap e 1 Array List.

A variável condutor guarda toda a informação dos condutores, a marca do carro, o modelo do carro, como não pode haver dois condutores com o mesmo nickname usamos o nickname como a key da HashMap.

A variável passageiro guarda a informação relativa ao passageiro e como também não pode existir dois passageiros com o mesmo nome a key usada foi o nickname do passageiro.

O ArrayList contém a informação dos condutores que estão disponíveis para partilhar uma viagem mas ainda não lhes foram atribuídos passageiros. A key usada é o nickname do condutor. Quando for atribuído um cliente a um determinado condutor, este condutor é apagado do arrayList.

```
public class Central {
    private HashMap<String, UtilizadorCondutor> condutor;//Todos os condutores registados
    private HashMap<String, UtilizadorPassageiro> passageiro;//Todos os passageiros registados


    //Como um condutor ou passageiro podem estar registados e não solicitarem serviço,
    //então tenho que registar so os que efetuaram pedidos
    //Guardando assim os serviços pelo nick
    private ArrayList<String> servicoCondutor;
    private HashMap<String,Servico> servicosAtivos;


    private Lock lock = new ReentrantLock();
    private Condition naoCondutores = lock.newCondition();
    private Condition naoMensagemChegada = lock.newCondition();
    private Condition naoMensagemFim = lock.newCondition();
```

A variável servicosAtivos contém a informação dos condutores que já lhe foram atribuídos passageiros. A key usada é o nickname do condutor.

É também nesta classe onde se encontram os métodos principais para poder suportar as funcionalidades pretendidas no projeto, como por exemplo: registar os vários utilizador, autenticar os várias utilizadores, solicitar uma viagem, etc.

No sentido de garantir um controlo de concorrência usamos Lock's. Optamos por usar Locks explícitos, para depois podemos usar variáveis de condição de maneira a evitar esperas ativas dos vários processos.

Serviço

Para conseguir ter uma série de informações relativas a um serviço, foi criada a classe Servico.

```
public class Servico {
    private String passageiro;
    private int coordx;
    private int coordy;
    private int coordDestinox;
    private int coordDesdinoy;
    private double preco;
    private String mensagemChegada;
    private String mensagemFim;
```

Assim cada Serviço tem o passageiro associado, as coordenadas do passageiro origem, destino e posteriormente a mensagem que o condutor enviará quando chegar ao local onde se encontra o passageiro. No final da viagem o condutor irá mandar uma mensagem a informar sobre o fim da viagem e o preço que o cliente pagou.

PacoteDados

A classe pacote foi criada para conseguir fazer chegar ao Handler a intenção do Utilizador e os argumentos necessários para que essa intenção possa vir a ser concebida, mediante da intenção do Utilizador o Handler enviará essa informação para a classe Central.

```
public class PacoteDados implements Serializable {
    private final String accao;
    private final HashMap<String,String> argumentos;
```

Esta classe Pacote, contém uma variável accao, onde é registada a intenção do Utilizador, ou seja qual o pedido que ele deseja fazer ao Servidor. E também possuí uma HashMap<String,String> onde vão ser passados os argumentos necessários para completar a ação pretendida. A *key* é como que correspondesse ao nome da variável e o *value* é o valor obtido pelo input do utilizador para essa mesma variável (*key*), por exemplo: quando a key for "REGISTAR_PASSAGEIRO" nós sabemos que a ação é registar um passageiro.

UtilizadorCondutor

A classe UtilizadorCondutor é a classe que contém toda a informação relativa ao condutor, o seu nickname, a sua palavra passe, o modelo do seu carro, a matrícula, as coordenadas onde ele parte e uma variável que diz se ele já esta logado no sistema ou não, para evitar que o mesmo condutor se log duas vezes.

```
public class UtilizadorCondutor {
    private String nick;
    private String pass;
    private String modelo;
    private String matricula;
    private boolean ativo;
    private int coordOX;
    private int coordOY;
```

UtilizadorPassageiro

A classe UtilizadorPassageiro contém toda a informação referente ao passageiro.

Contém a sua palavra passe, o seu nickname, as coordenadas onde se encontra, contém também as coordenadas para onde se dirige e uma variável ativo para evitar que o mesmo passageiro faça login duas vezes.

```
public class UtilizadorPassageiro {
    private String nick;
    private String pass;
    private boolean ativo;
    private int coordOX;
    private int coordOY;
    private int coordDX;
    private int coordDY;
```

Conclusão

Chegado o momento final deste projeto prático de Sistemas Distribuídos torna-se, claro que novas competências foram adquiridas e muitas outras, por certo, desenvolvidas.

Ficamos claramente a perceber melhor o funcionamento de um sistema baseado em comunicação entre Cliente e Servidor através de *Sockets*, isto é, um sistema que aceite a conexão simultânea de múltiplos clientes. A forma como um Servidor trata os dados e pedidos enviados por Cliente, e a maneira como se ligam.

Com a elaboração deste mini servidor *multi-thread,* conseguimos perceber agora as vantagens de usar *threads,* em qualquer sistema que envolva vários utilizadores a usar a mesma aplicação. Como não poderia deixar de faltar, na vertente mais relacionada com a UC, o controlo de concorrência foi um desafio para nós, e que agora depois de realizado o projeto, tornou-se clara a importância dessa implementação.

Para terminar, apesar de, como naturalmente acontece, um grupo ser constituído por diferentes pessoas, com diferentes gostos, opiniões ou vocações, fazemos questão de realçar que cremos na homogeneidade do nosso grupo, isto é, chegado o momento final deste trabalho prático, o contributo de cada um de nós equivaleu-se perfeitamente.