Unidade Curricular

de

Sistemas Distribuídos

Gestor de Leilões

Trabalho Realizado por:
- Ricardo Certo A75315
- Guilherme Guerreiro A73860
- Marco Barbosa A75278
- Daniel Rodrigues A67634

Índice

Introdução

Nesta Unidade Curricular foi-nos proposto a implementação de um gestor de leilões.
Neste Gestor existem **dois tipos de clientes**:

- Os clientes que pretendem fazer uma licitação sobre um dado produto que está a ser leiloado, tendo que as apostas serem recorrentemente mais elevadas, ganhando o Cliente com a licitação mais alta;
- Os clientes que queiram leiloar um produto, permitindo que outros clientes possam fazer licitações sobre o produto em questão.

Pretendia-se o uso de um cliente desenvolvido em **Java**, onde os utilizadores poderiam interagir itermediados por um servidor *multi-threaded* escrito na mesma linguagem. Essa comunicação é feita através de *sockets* TCP.
Ao longo deste relatório são explicadas decisões que foram tomadas, bem como o funcionamento do gestor de leilões, recorrendo ao uso de imagens exemplficadoras.

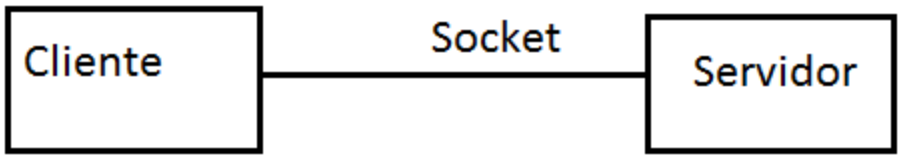
Classes

A partir do enunciado do projeto e tendo em conta as funcionalidades que era suposto o serviço oferecer, facilmente idealizamos a maioria das classes que seriam necessárias para tal. Assim sendo, para irmos de encontro ao objetivo, desenvolvemos as seguintes Classes:

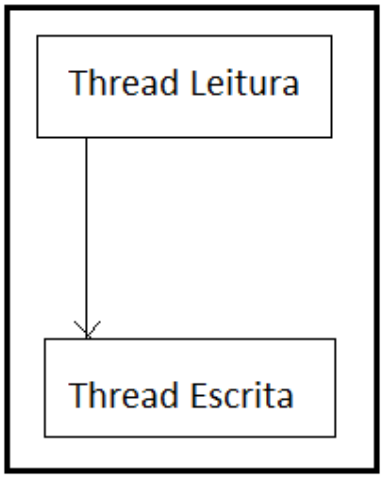
- Servidor
- Cliente
- Utilizador
- Comprador
- Vendedor
- CasaLeiloes
- Leilao
- Menu
- ThreadInputCliente/ThreadOutputCliente
- ThreadOutputServidor / ThreadInputServidor
- Mensagem

Servidor e Cliente

Servidor e Cliente vão ser as classes que vão comunicar entre si através de um Socket. Assim, para cada classe são criadas 2 Threads. No seguinte esquema está demonstrado graficamente como funciona a comunicação entre as duas classes.



Dentro do Cliente e do Servidor vamos ter duas threads dentro de cada um, sendo que uma thread fica responsável pela leitura de dados enquanto a outra fica responsável pela escrita de dados, como podemos observar no seguinte esquema:



Menu

A utilidade desta classe é oferecer uma simples interface ao utilizador. Por questões de logísticas e de maneira a permitir a interação entre o utilizador e o Servidor.

CasaLeiloes

A classe CasaLeiloes é a a peça fundamental neste trabalho pois é aqui que será guardada toda a informação relativa aos seus utilizadores. Para esse efeito foram criadas 3 HashMaps,1 ArrayList, um Lock e um id.

A variável **compradores** guarda toda a informação acerca dos mesmos, sendo a atribuição da chave da HashMap feita utilizando o **username** de cada comprador. A variável **vendedores** funciona de forma semelhante, mas nesta é guardada informação relativamente aos vendedores deste sistema.

A variável leiloesAtivos, contém informação dos leilões que se encontram disponíveis para licitação, utilizando como chave da HashMap o username do leiloeiro, uma vez que um vendedor pode apenas iniciar e manter um leilão ativo de cada vez. Além disso, desta forma conseguimos associar um dado vendedor ao seu leilão.

Relativamente ao ArrayList que estamos a utilizar, **compradoresAtivos**, guarda a lista dos compradores que estão a participar no leilão, isto é, a partir do momento em que um dado comprador faz uma licitação, é guardado o seu username no ArrayList, mantendo assim em registo para possível uso posterior no anúncio do vencedor.

É também nesta classe que se encontram os principais métodos que permitem aos utilizadores tirar partido das funcionalidades deste projeto desde os **registos, logins e logout** dos clientes, bem como as **operações possíveis** dos mesmos. Algumas das nossas funções implementadas nesta classe sofrem um lock, ou seja, quando estávamos a realizar uma determinada ação mais nenhum utilizador podia ler ou escrever sobre aquela ação, tem de esperar que termine essa ação.

```
public class CasaLeiloes {

    private HashMap<String, Comprador> compradores; //
    private HashMap<String, Vendedor> vendedores; // Ma
    // Map que contem os leiloes ativos no momento
    private HashMap<String, Leilao> leiloesAtivos;
    // Lista de compradores que se encontram ativos em
    private ArrayList<String> compradoresAtivos;
    // por o lock no inicio
    private Lock lock = new ReentrantLock();
    private int id=0;
```

Leilão

Como seria de esperar, foi criada uma Classe Leilão, a qual irá conter toda a informação útil relativamente a um leilão.

```
public class Leilao {

        private String vendedor;
        private HashMap<Comprador, Float> licitadores;
        private float preçoInicial;
        private float maxLicitacao;
        private int idleilao;
        private String maxLicitador;
```

Por questões de logística e apresentação, mantemos guardada informação sobre a licitação mais alta e o Comprador respetivo. Assim, aquando a finalização do leilão, o servidor irá enviar uma mensagem a informar sobre o encerramento do leilão e do vencedor do mesmo.

Utilizador

Esta classe serve de base ao **comprador** e ao **vendedor.** Esta contém informações sobre o username, password e indica se um dado utilizador se encontra ou não loggado ao serviço.
Assim, tanto o comprador como o vendedor, são classes que extendem da superclasse utilizador.

```
public abstract class Utilizador implements Serializable
{
    private String username, password;
    private boolean logged;
    private Socket s;
    private PrintWriter output;
```

ThreadInputCliente/ThreadOutputCliente

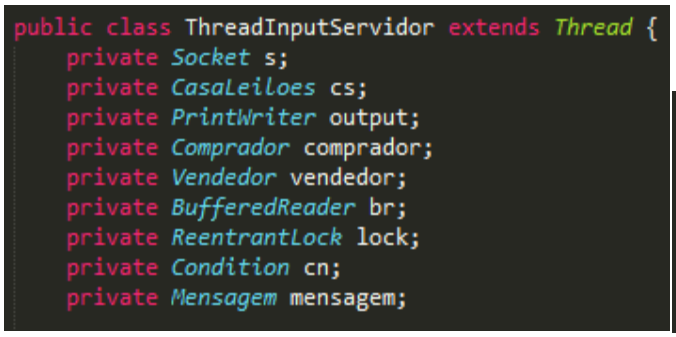
Estas duas threads dizem respeito ao Cliente, pois vai ser através delas que o Cliente vai ler a informação do Socket com uma thread de leitura e depois envia a sua resposta por uma thread de escrita que de seguida essa informação vai chegar ao Servidor através do Socket.

```
public class ThreadInputCliente extends Thread {
    private Socket s;
    private PrintWriter output;
    private Menu menu;
    private BufferedReader br;
    private ReentrantLock lock;
    private Condition cn;
```

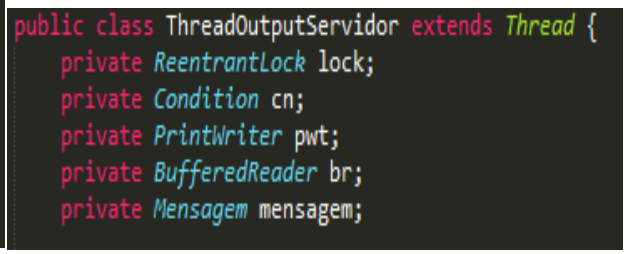
```
public class ThreadOutputCliente extends Thread{
    private BufferedReader br;
    private Menu menu;
    private ReentrantLock lock;
    private Condition cn;
```

ThreadInputServidor / ThreadOutputServidor

Estas duas threads dizem respeito ao Servidor, pois vai ser através delas que o Servidor vai ler a informação do Socket com uma thread de leitura e depois envia a sua resposta por uma thread de escrita que de seguida essa informação vai chegar ao Cliente através do Socket.



```
public class ThreadInputServidor extends Thread {
    private Socket s;
    private CasaLeiloes cs;
    private PrintWriter output;
    private Comprador comprador;
    private Vendedor vendedor;
    private BufferedReader br;
    private ReentrantLock lock;
    private Condition cn;
    private Mensagem mensagem;
```



```
public class ThreadOutputServidor extends Thread {
    private ReentrantLock lock;
    private Condition cn;
    private PrintWriter pwt;
    private BufferedReader br;
    private Mensagem mensagem;
```

Mensagem

A classe mensagem é uma classe simples, que é responsável por fazer chegar ás Threads a intenção do utilizador e os argumentos necessários para que esta informação possa vir a ser concebida, mediante da intenção do Utilizador.

Conclusão

Este trabalho prático permitiu aprofundar conhecimentos sobre comunicação entre um servidor *multithreaded* e vários clientes através de programação de sockets em Java, no contexto de uma Casa de Leilões, onde os vários utilizadores interagem com o servidor e trocam informações com ele. Os conhecimentos adquiridos com a realização deste trabalho prático vão ser de tremenda importância pois vão ser a base para qualquer projeto no futuro em que queiramos garantir vários acessos á mesma informação, sem nunca a informação lá contida ser corrompida.