Relatório

SISTEMAS DISTRIBUÍDOS

2016/2017




Bruno Miguel Salgado Machado	A74941
Fábio Luís Baião da Silva	A75662
João Rui de Sousa Miguel	A74237
Luís Manuel Leite Costa	A74819

Utilizador

Para permitir o registo e login dos utilizadores criou-se uma classe *Utilizador*:

```
class Utilizador{
    String username;
    String password;
    [redacted]

    Utilizador (String username, String password){
        this.username = username;
        this.password = password;
        [redacted]
    }

    String getUsername(){
        return this.username;
    }

    String getPassword(){
        return this.password;
    }
```

Atribuímos o nome *ServicoImpl* à classe responsável pela gestão dos utilizadores e dos leilões. Esta classe contem os utilizadores registados e os métodos *registar* e *autenticar*:

```
class ServicoImpl implements Servico{
    Map<String, Utilizador> utilizadores;
    [redacted]

    ServicoImpl (){
        this.utilizadores = new HashMap<>();
        [redacted]
    }

    public Utilizador registar (String username, String password) throws UtilizadorInvalidoException{
        synchronized (utilizadores){
            Utilizador u = this.utilizadores.get(username);
            if (u != null){
                throw new UtilizadorInvalidoException ("E"); // E - > existe
            }
            u = new Utilizador (username, password);
            this.utilizadores.put(username, u);
            return u;
        }
    }

    public Utilizador autenticar (String username, String password) throws UtilizadorInvalidoException{
        synchronized (utilizadores){
            Utilizador u = this.utilizadores.get(username);
            if (u == null){
                throw new UtilizadorInvalidoException ("N"); // N -> nao existe
            }
            if (u.getPassword().equals(password)){
                return u;
            }
            else throw new UtilizadorInvalidoException ("P"); // P -> password errada
        }
    }
```

O método *registar* permite o registo de um novo utilizador no serviço.

É verificado se já existe algum utilizador com o nome de utilizador introduzido. Se existir, é lançada uma exceção com a mensagem "E", que significa que o *username* já existe. Caso contrario é criado um novo utilizador com o *username* e *password* introduzidos e é adicionado aos utilizadores.

O método *autenticar* permite um utilizador autenticar-se no serviço.

Tal como no método *registar*, é verificado se existe algum utilizador com *username* igual ao introduzido. Se este não existir é lançada uma exceção com a mensagem "N" indicando que o *username* não existe. Caso exista é comparada a *password* introduzida com a *password* do utilizador obtido. Neste caso, se as passwords forem iguais o utilizador fica registado. Caso contrário é lançada uma exceção com a mensagem "P", com o significado de password errada.

Nestes dois métodos o controlo de concorrência é assegurado com um bloco *synchronized* que usa o *lock* do objeto *utilizadores*.

Leilão

Para permitir efetuar operações sobre um leilão criou-se a classe *Leilao*.

A criação de um leilão é efetuada indicando uma descrição e devolvendo o número identificativo. Assim, inicialmente, definiu-se a classe da seguinte forma:

```
class Leilao{
    String descricao;
    int id;
    Utilizador vendedor;
    [redacted]

    Leilao (String descricao, int id, Utilizador vendedor){
        this.descricao = descricao;
        this.id = id;
        this.vendedor = vendedor;
        [redacted]
    }
```

Na classe *ServicoImpl*, acrescentou-se um *Map* para guardar os leiloes em curso e um inteiro que representa o id do ultimo leilão criado.

```
class ServicoImpl implements Servico{
    Map<String, Utilizador> utilizadores;
    Map<Integer, Leilao> leiloes;
    int n;

    ServicoImpl (){
        this.utilizadores = new HashMap<>();
        this.leiloes = new HashMap<>();
        n = 0;
    }
```

O método *iniciar*, na classe *ServicoImpl*, que permite iniciar um leilão, foi construído da seguinte forma:

```
public int iniciar (String descricao, Utilizador vendedor){
    synchronized (leiloes){
        this.n++;
        Leilao l = new Leilao (descricao, this.n, vendedor);
        this.leiloes.put(this.n, l);
        return this.n;
    }
}
```

Para permitir efetuar licitações foram adicionadas duas variáveis na classe *Leilao*: uma lista de licitações e a licitação mais alta. Para isso foi necessário criar uma classe *Licitacao*:

```
private class Licitacao{
    Utilizador comprador;
    double valor;

    Licitacao (Utilizador comprador, double valor){
        this.comprador = comprador;
        this.valor = valor;
    }

    double getValor(){
        return valor;
    }

    Utilizador getUtilizador(){
        return comprador;
    }
}
```

```
class Leilao{
    String descricao;
    int id;
    Utilizador vendedor;
    List<Licitacao> licitacoes;
    Licitacao maisAlta;

    Leilao (String descricao, int id, Utilizador vendedor){
        this.descricao = descricao;
        this.id = id;
        this.vendedor = vendedor;
        this.licitacoes = new ArrayList<>();
        this.maisAlta = null;
    }
```

O método *licitar* na classe *Leilao* foi desenvolvido da seguinte forma:

```
void licitar (Utilizador comprador, double valor) throws LicitacaoInvalidaException{
    if (comprador.getUsername().equals(this.vendedor.getUsername()))
        throw new LicitacaoInvalidaException ("P"); // P -> proprio leilao

    if (valor <= 0 || valor <= this.maisAlta.getValor()){
        throw new LicitacaoInvalidaException ("V"); // V -> valor invalido
    }
    Licitacao l = new Licitacao (comprador, valor);
    this.maisAlta = l;
    this.licitacoes.add(l);
}
```

Em primeiro lugar é testado se o utilizador que faz a licitação não é o mesmo que criou. Se for é lançada uma exceção com a mensagem “P”, indicando que é o próprio leilão. De seguida verifica-se se o valor é valido, isto é, se é maior que 0 (importante verificar na primeira licitação) e se é maior que o valor da licitação mais alta efetuada até ao momento. Se o valor for invalido, é lançada uma exceção com a mensagem “V” que informa que o valor é invalido. Por fim, a licitação é criada e é adicionada à lista de licitações.

Na classe *ServicoImpl* o método toma o seguinte aspeto:

```
public void licitar (int id, double valor, Utilizador comprador) throws LicitacaoInvalidaException{
    synchronized (leiloes){
        Leilao l = this.leiloes.get(id);
        if (l == null){
            throw new LicitacaoInvalidaException ("N"); // N -> nao existe
        }
        l.licitar(comprador, valor);
    }
}
```

Aqui apenas se verifica se o id do leilão indicado corresponde a um leilão. Caso contrario é lançada uma exceção com a mensagem “N” que significa que o leilão não existe.

Uma das funcionalidades pedidas é a de listar os leilões em curso. Para isso, foi criado o método *listar* nas classes *Leilao* e *ServicoImpl*.

```
String listar(){
    StringBuilder sb = new StringBuilder();
    String s;
    s = Integer.toString(this.id);
    sb.append(s);
    sb.append(" - ");
    sb.append(descricao);
    sb.append(" - ");
    if (this.maisAlta == null){
        sb.append ("0");
    }
    else {
        s = Double.toString(this.maisAlta.getValor());
        sb.append(s);
    }
    return sb.toString();
}
```

Na classe *Leilao* é criada uma String com os dados que se pretende que sejam listados (id, descrição e valor da licitação mais alta).

```java
public String listar (Utilizador utilizador){
    synchronized (leiloes){
        StringBuilder sb = new StringBuilder();
        for (Leilao l: leiloes.values()){
            sb.append(l.listar());
            Utilizador vendedor = l.getVendedor();
            if (vendedor.getUsername().equals(utilizador.getUsername())){
                sb.append(" *");
            }
            try{
                Utilizador maisAlta = l.getMaisAlta();
                if (maisAlta.getUsername().equals(utilizador.getUsername())){
                    sb.append(" +");
                }
            }
            catch (SemLicitacoesException e){}
            sb.append("\n");
        }
        return sb.toString();
    }
}
```

Na classe *ServicoImpl*, obtém-se a string com os dados para cada leilão em curso e caso o utilizador seja o criador do leilão ou o utilizador com a licitação mais alta é adicionado um "*" ou um "+", respetivamente. É retornado uma string que resulta na concatenação de todas as strings geradas.

A ultima funcionalidade pedida é a de terminar o leilão. Tal como para as outras funcionalidades, foi também criado o método terminar nas classes *Leilao* e *ServicoImpl*.

```
String terminar() throws LeilaoInvalidoException, SemLicitacoesException{
    if (this.maisAlta == null)
        throw new SemLicitacoesException ("S"); // S - > sem licitacoes

    List<Utilizador> list = new ArrayList<>();

    Utilizador vencedor = this.maisAlta.getUtilizador();
    String mensagem = this.mensagem(vencedor);

    for (Licitacao l: licitacoes){
        Utilizador comprador = l.getUtilizador();
        Iterator<Utilizador> it = list.iterator();
        boolean existe = false;
        while(it.hasNext() && !existe){
            Utilizador u = it.next();
            if (u.getUsername().equals(comprador.getUsername()))
                existe = true;

        }
        if (!existe){
            comprador.adicionarMensagem(mensagem);
            list.add(comprador);
        }
    }
    return mensagem;
}

private String mensagem (Utilizador vencedor){
    StringBuilder sb = new StringBuilder();
            String s;
            s = Integer.toString(this.id);
    sb.append(s);
    sb.append(" - ");
    sb.append(this.descricao);
    sb.append(" - ");
    sb.append(vencedor.getUsername());
    sb.append(" - ");
            s = Double.toString(this.maisAlta.getValor());
    sb.append(s);
    return sb.toString();
}
```

Na classe *Leilao* o método terminar caso não tenha havido licitações é lançada uma exceção com a mensagem “S” que indica que não houve licitações.

É criada uma string com o id, a descrição, o nome do vencedor e o respetivo valor. Esta mensagem é enviada a todos os utilizadores que efetuaram licitações no leilão que foi terminado, inclusive o vencedor do mesmo.

Para isso, foi adicionada uma lista de mensagens na classe Utilizador:

```java
class Utilizador{
    String username;
    String password;

    List<String> mensagens;

    Utilizador (String username, String password){
        this.username = username;
        this.password = password;
        this.mensagens = new ArrayList<>();
    }
```

Na classe *ServicoImpl* o método foi implementado da seguinte forma:

```java
public String terminar (int id, Utilizador vendedor) throws LeilaoInvalidoException, SemLicitacoesException{
        Leilao l = null;
        synchronized (leiloes){
            l = this.leiloes.get(id);
            if (l == null){
                throw new LeilaoInvalidoException ("N"); // N -> nao existe
            }
            Utilizador u = l.getVendedor();
            if (!u.getUsername().equals(vendedor.getUsername())){
                throw new LeilaoInvalidoException ("P"); // P -> proprio
            }
            this.leiloes.remove(id);
        }
        return l.terminar();
}
```

Se o id indicado não corresponder a nenhum leilão é lançada uma exceção com a mensagem "N" que significa que o leilão não existe.

Se o utilizador que criou o leilão for diferente do utilizador que está a terminar o leilão é lançada uma exceção com a mensagem "P" indicando que o leilão não é do próprio.

Se nenhuma destas condições lançar uma exceção então o leilão é removido e é executada a função terminar da classe *Leilao*.

Nos métodos relativos à gestão de leilões da classe *ServicoImpl* o controlo de concorrência é realizado com um bloco *synchronized* que usa o *lock* do objeto *leiloes*. Apenas no método terminar desta classe é que existe código fora do bloco. Isto deve-se ao facto de que quando este método for invocado o leilão é removido do *Map*, logo a geração das mensagens pode ser feita de forma concorrente com outros métodos, já que o leilão que foi removido já não pode ser mais utilizado.

Foi ainda acrescentada uma funcionalidade que permite ler as mensagens referentes ao termino de leilões.

Na classe *ServicoImpl* foi adicionado o seguinte método:

```java
String mensagens (Utilizador utilizador){
    return utilizador.getMensagens();
}
```

Na classe *Utilizador*, foram adicionados os métodos *adicionarMensagens* e *getMensagens:*

```
    synchronized void adicionarMensagem(String mensagem){
        mensagens.add(mensagem);
    }

    synchronized String getMensagens(){
        if (mensagens.isEmpty())
            return "";

        StringBuilder sb = new StringBuilder();
        Iterator<String> it = mensagens.iterator();
        while(it.hasNext()){
            String s = it.next();
            sb.append(s);
            sb.append("\n");
            it.remove();
        }
        return sb.toString();
    }
}
```

O método *getMensagens*, para alem de gerar uma string com todas as mensagens, ainda remove as mensagens da lista de mensagens. Assim, quando este método for invocado apenas irão ser devolvidas as mensagens não lidas.

Foi também necessário tornar estes dois métodos *synchronized*, pois podem ocorrer leituras e/ou escritas simultaneamente.

Cliente - Servidor

Depois de termos o serviço de leiloes implementado, desenvolvemos as classes *Cliente* e *Servidor*.

A classe *Cliente* implementa a *User Interface*, invocando métodos da classe *ServicoStub* (uma vez que esta classe implementa a Interface *Servico*, também implementada pela classe *ServicoImpl*, os métodos são os mesmos). A classe *ServicoStub* é então a classe responsável pela comunicação com o Servidor.

A classe Servidor tem como função manter o serviço de leilões funcional e permitir que os utilizadores possam interagir com este. Para isso, sempre que um *Cliente* estabeleça uma conexão com o *Servico* é criada uma nova thread (na classe *ClientHandler*) responsável por comunicar com o *ServicoStub.*

Cada objeto da classe *ClientHandler* (ou seja, cada thread), tal como pedido no enunciado, apenas escreve em um socket. Esta classe tem como variáveis de instancia o um objeto da classe *ServicoImpl* (comum em todas as threads) e, ainda, uma variável da classe Utilizador, que permite saber qual o utilizador com que esta a comunicar.

O *Servidor* pode ser iniciado sem argumentos, sendo assim usada a porta por defeito (12345), ou então pode ser indicada porta que se pretende utilizar no primeiro argumento. Da mesma forma, o *Cliente* pode ser iniciado sem argumentos, sendo assim usado o host e a porta por

defeito (“localhost” e 12345), ou então pode ser indicado o host no primeiro argumento e a porta no segundo.