#exame 2019-2020 e #exame 2016-2017
Exame da Época Especial - Avaliação prática - P1

==teórica)
1. Diga, justificando, se é correcto afirmar-se que o seguinte método permite apenas copiar o conteúdo de um ficheiro para outro. Se a resposta for negativa, inclua na justificação pelo menos um contra-exemplo.

```
public static void copy(java.io.InputStream in, java.io.OutputStream out) throws
java.io.IOException
{
int c;
while ((c = in.read()) != -1){
out.write(c);
}
}
```

R:
pode ser usado também para leitura e escrita em sockets
as classes InputStream e OutputStream são classes abstractas, e uma classe abstracta são aquelas em que existe pelo menos um método não implementado
neste exemplo temos um método que recebe dois métodos: InputStream e OutputStream
o InputStream é uma classe abstracta que tem métodos do tipo: read, read só com um caracter, receb arrays de bytes e devolve o numero de bytes copiados, e subarrays, ..
o OutputStream é uma classe abstracta que tem métodos do tipo: tem métodos para escrever, e que permite ir buscar bytes ou um fluxo de bytes, por exemplo no ecrã, num ficheiro, numa ligação TCP, pode ser um array de bytes,..

mas a resposta é não porque são classes abstractas
posso copiar de ficheiro para ficheiro
posso copiar de ficheiro para socket
posso copiar de socket para ficheiro

para teste:
o OutputStream só permite serializar para ficheiro? ou só permite serializar para socket?
r:
não porque ao criar um objecto OutputStream passamos-lhe qualquer coisa que é OutputStream
o que o wirte and share do OutputStream precisa de receber um objecto que implemente os métodos do OutputStream, e não interessa como

para teste:
o que faz o método write object no OutputStream
r:
o write object implementa o algortimo de serialização, pega num objecto, que na realidade é uma arvore de objecto (porque depois temos referências), e o que produz num formato especifico que é um binário java, produz um fluxo de bytes contiguos que permite desctever no objecto as várias classes envolvidas,as versões, os valores, os membros, .., e que permite depois ser transmitido ou guardar

para teste:
o que um driver, o que faz um driver jbdc?
R:
o driver é um conjunto de classes que vamos buscar num jar e que tem classes/métodos que implementam a api do jbdc
o driver é a implementação dessa API, dessas classes, para acesso a servidores de BD especificos

para teste:
R:
porque não posso usar um conector mysql para aceder a uma base de dados de oracle?
têm drivers diferentes, porque muitas das coisas que as classes/métodos fazem é estabelecer uma ligação TCP ao servidor remotamente
e enviar pedidos num determinado formato que tem a ver com o protocolo e os SGBD, e quem desenvolveu o protocolo não desenvolveu nenhum protocolo de comunicação ao nivel a aplicação, entre o cliente o servidor BD, não implemntou de forma standart e por esse motivo as mensagens e as trocas de mensagens são diferentes

==teórica)

2. Geralmente, associado a plataformas de middleware que oferecem a abstracção de objecto remoto/distribuído, existem aplicações que, em termos genéricos, podem ser designadas de serviços de nomeação (por exemplo, rmiregistry.exe, tnamrserv.exe e orbd.exe). Explique quais são os seus dois objectivos principais e de que forma estes são atingidos (ou seja, enumere as principais operações/passos executados desde a fase de arranque). Na resposta, que não deve incluir código, utilize, entre outros, os termos socket, mensagem, porto e thread.
r:
conceito de RMi (ou corba)
estas têm uma aplicação autónoma, mas que têm o mesmo objectivo e que são os serviços de nomeação
rmiregistry (no RMi)
orbd (no Corba)
o rmiregistry
os dois principais objectivos são:

ao inves de saber o ip e o porto automático do servidor, mas eu coloco numa maquina qualquer com o ip conhecido e com porto fixo de 1099,
assim
o servidor quando arranca contacta (ligação TCP) o rmiregistry
e envia uma informação a dizer regista este proxy/este serviço sob este nome
uma aplicação que quer aceder ao serviço contacta o rmiregistry (ligação TCP, no ip e no porto 1099) a pedir o serviço remoto
o rmiregistry faz o lookup e devolve serializado ou envia uma excepção.
a aplicação recebe, tem uma classe, recebe um objecto do tipo, onde tem o ip e o porto da maquina(servidor) onde está o serviço

r2:
dois objectivos:
registar serviços com um determinado nome, referências para objectos remotas sob determinados nomes, e permite obter essas referencias através de queries
e permite obter a referência, isto é, permite fazer um submit e um get
e enumere as principais fazes de arranque:
é um servidor TCP concorrente, que é posto a correr,
e quando quero utilizar o serviço faço uma ligação TCP com a indicação do porto, e é enviado um pedido sobre a forma de uma mensagem

o servidor está à espera de pedidos de ligação, e como é um servidor concorrente, sempre que alguém se liga cria uma thread, aguarda o pedido e responde
(lembrar que só existe uma forma de comunicar na internet, e é através de sockets)

para teste:
o que faz, qual o objectivo este método estático e o que é desencadeado quando o chamam:
Naming.bind("rmi://192.168.1.1/nome1", o);
objetivo: solicitar a uma aplicação rmi, que está a correr na maquina com o ip 192.168.1.1, com o porto 1099, e enviar-lhe o registo do serviço/referencia remoto o, com o nome nome1
estabelece um ligação TCP, e enviar-lhe uma mensagem, com o regista-me aí um serviço RMi com o nome nome1, e também toma a referência serializada o
se correr bem ele acrescenta
se correr mal ou se já existir ele lança uma excepção
e ele envia uma resposta (uma excepção ou um true ou um false)

para teste:
Naming.rebind("rmi://192.168.1.1/nome1", o);

para teste:
o que faz, qual o objectivo este método estático e o que é desencadeado quando o chamam:
Naming.lookup("rmi://192.168.1.1/nome2");
contactar uma aplicação Rmi registry que está no ip 192.168.1.1, na porta 1099, de qualquer coisa que ele tem com o nome nome2
estabelece um ligação TCP, no ip e no porto
constroi uma mensagem que obtem a referencia para o nome2
serializa a mensagem, através da ligação TCP
do outro lado o registry vai receber a mensagem, e verifca se corre bem, se encontra, e devolver o resultado
e quando se recebe a mensagem desserializa-a e vê a resposta, e com base na resposta devolve a referencia remota ou lança uma excepção

==teórica)
3.Diga qual é o objectivo do seguinte método e descreva o significado de cada um dos campos que compõem a URL passada como argumento (rmi, 192.168.1.1, 1099 e RMILightBulb):

```
java.rmi.Remote r = java.rmi.Naming.lookup("rmi://192.168.1.1:1099/RMILightBulb").
```

r:
o objecto é obter uma referencia remota para o objecto que está registado no rmiregistry com o nome RMILightBulb
rmi, é o protocolo usado na comunicação, é o que define as mensagens trocadas entre a aplicação que chama isto e a aplicação do outro lado
192.168.1.1, é o ip onde está a correr o rmiregistry
1099, é o porto onde está a correr o rmiregistry
RMILightBulb, é o nome do serviço

para teste:
o que é que acontece por baixo com java.rmi.Remote r = java.rmi.Naming.lookup("rmi://192.168.1.1:1099/RMILightBulb")?
r:
estabelece uma ligação TCP naqueles ip e porto, envia uma mensagem com um pedido para uma referência remota de um objecto que se chama RMILightBulb e aguarda resposta
se a resposta for um objecto do tipo excepção faz o throw da excepção
caso contrário faz o return do que recebeu e depois temos que fazer o cast

==teórica)
4. O pedaço de código seguinte permite obter um recurso alojado um servidor Web através do protocolo HTTP, recorrendo a uma classe específica que encapsula esse tipo de interacção. Sabendo que o HTTP é um protocolo do nível de aplicação que recorre ao protocolo de transporte TCP e, por imissão, ao porto 80, deduza a sequência de acções/passos principais desencadeados pelo método openStream (1- Estabelece… 2- … m- Envia… n- Obtém… o- Devolve…). A resposta não deve incluir código.

```
java.net.URL myURL = new java.net.URL ("https://moodle.isec.pt/moodle/mod/folder/view.php?id=470");
java.net.InputStream in = myURL.openStream();
int b;

while(true){
b = in.read();
...
}
```

r:
existe um objecto do tipo:
java.net.URL myURL
depois temos
java.net.InputStream in = myURL.openStream();
e de seguida:
b = in.read();
e com tudo isto estou a receber o conteudo da resposta get que foi enviado para este servidor e para este recursoview.php?id=470

o URL pode dar-me uma excepção
quando se faz o myURL.openStream(), e
estabelece uma ligação TCP nesta maquina (https://moodle.isec.pt/)
envia um pedido get oara identificar como recurso o URI:
moodle/mod/folder/view.php?id=470
e do outro lado ele vai enviar uma resposta com várias linhas de texto com vários \n \n e fecha a ligação TCP
no buffer de entrada eu vou ter os bytes enviados, que é a resposta
fica associado ao objecto java.net.InputStream in = myURL.openStream();
e ao fazer o read b = in.read();
vão ser lidos os bytes

==teórica)
5. Acrescente uma única linha de código na classe UseMyThreads ou MyThread de modo a que a thread t2 apenas inicie depois de t1 deixar de estar activa. Altere igualmente a declaração do método metodo1 de modo a que, se várias threads possuírem uma referência para a mesma instância da classe MyThread, este apenas possa ser executado por uma única thread em cada instante.

```
public class MyThread extends Thread {
X x;
public MyThread(X x){
this.x = x;
}
```

```
...
public metodo1(){
...
}
...
public void run() {
...
}
...
}


public class UseMyThreads {
...
public static void main(String args[]){
...
Thread t1 = new MyThread(new X()).start();
Thread t2 = new MyThread(new X()).start();
...
}
}
r:
a Thread t1 = new MyThread(new X()).start();
vai correr o run

assim e no UseMyThreads podemos usar um JOIN
Thread t1 = new MyThread(new X()).start();
t1.join;

r2:
ou até podia olhar para:
assim e no MyThread e no run() podemos fazer
synhcronized(x){...};
mas quando fazermos  new MyThread(new X()).start();
estamos a criar objectos novos e não vão bloquear logo esta solução não é válida só seria se
ao invés de:
Thread t1 = new MyThread(new X()).start();
Thread t2 = new MyThread(new X()).start();
e se existir:
declaração de um X = new X
e passar o X para dentro de cada thread este mesmo X
e agora sim vem o synhcronized do X
mas isto tb não é veridico face ao não sabes como vai ser o escalonamento e isso depende do SO

atenção: não é porque no código existe uma sequencia que u output é sempre igual.. as coisas variam face ao escalonamento

r3:
segunda parte da resposta é meter o método 1 como synhcronized
na classe MyThread
public synhcronized metodo1(){ ... }
eu posso fazer synhcronized ao nivel do método no run nao posso só internamente


r4 final:
primeira parte:
assim e no UseMyThreads podemos usar um JOIN
Thread t1 = new MyThread(new X()).start();
t1.join;

segunda parte da resposta é meter o método 1 como synhcronized
na classe MyThread
public synhcronized metodo1(){ ... }
eu posso fazer synhcronized ao nivel do método no run nao posso só internamente

==teórica) #exame 19-20 especial
1.
considere uma api web (web service) que incluir entre outras s seguintes URI:
1) /users?address="coimbra"
2) /users/delete?address="coimbra"
3) /users/coimbra
```

4) /users/coimbra10
extras
5) /users/10/update?address="coimbra"
6) /users/insert?id=10&name="ana"&address="coimbra"

a) indique para cada uma um possivel significado
r:
1) lista de utilizadores com morada em coimbra
2) manda apagar da lista de utilizadores com a morada em coimbra
3) lista de utilizadores que vivem em coimbra
4) identifica o utilizador com o codigo 10, dos utilizadores que vivem em coimbra OU a 10ª página de utilizadores de coimbra OU o registo 10 em coimbra
extras
5) um pedido de update da morada do utilizador com o codigo 10
6) inserir nos utilizadores com os dados...

b) identifique, justificando a URI que não serve os principios REST e apresente, igualmente uma alternativa que esteja correcta
olhar para as primeiras 4
2) tem um verbo, a URI só deve identificar o recurso a URI correta seria
as URI identificam o recurso, as acções é através de um verbo, que vai na mensagem http

==teórica) #exame 19-20 especial
2)
explique o conceito de callback/notificação assincrona em sistemas distribuidos e de que forma pode conseguir-se esta funcionalidade em aplicações baseadas no paradigma de objecto remoto (e.g JavaRmi). não recorra a código na resposta
explique o conceito de callback/notificação assincrona em sistemas distribuidos e de que forma pode conseguir-se esta funcionalidade em aplicações baseadas em sockets TCP (e.g., instâncias da classe java.net.Socket na linguagem java). não recorra a código na resposta
explique o conceito de callback/notificação assincrona em sistemas distribuidos e de que forma pode conseguir-se esta funcionalidade em aplicações baseadas em sockets UDP (e.g. instâncias da classe java.net.DatagramSocket na lingiagem java). não recorra código na resposta)

==teórica) #exame 19-20 especial
3.
consider a seguinte instrução:
java.rmi.Naming.unbind("rmi://reg.isec.pt/5001/time");
java.rmi.Naming.bind("rmi://reg.isec.pt/5001/time", timeService);
java.rmi.Remote r = java.rmi.Naming.lookup("rmi://reg.isec.pt:5001/time");

a) explique o que identificam os elementos "reg.isec.pt", "5001", e "time"
b) deduza a sequência de acções/passos principais desencadedos quando esta insgtrução é invocada (ligações estabelecidas, mensagens trocadas, etc.) a resposta não deve incluir código

==teórica) #exame 19-20 especial
4. para uma aplicação desenvolvida em Java aceda a um determinado sistema de gestão de base de dados (SGBD), pode recorrer-se à API JBDC. para o efeito é necessário usar um driver adequado ao tipo de SGBD acedido (por exemplo, mysql-connector-java-5.1.45-bin.jar para MySQL e obdc8.jar para oracle)
explique:
a) o que é comum a todos os drivers JDBC, independentemente do SGBD alvo;
r:
todos oferecem a API jbdc, é o api do jbdc
b) o que difere entre drivers JDBC destinados a SGBD distintos
r:
são implementados de forma diferente
o protocolo de comunicação é diferente do procotolo de comunicação MySQL

==prática)
1. (pergunta de api rest) considere a seguinte mensagem HTTP destinada a pedir, a uma determinada API/serviço web, a lista de undiades curriculares concluidas por um determinado aluno, identificado através do seu número de estudante num determinado ano lectivo e assmindo um esquema de autenticação baseado em tokes:

POST /alunos/list/ucs/concluidas HTTP/1.1
Host: api.pd.com

```
User-Agent: curl/7.55.1
Student-number: 201012345
Year: 202-21
Accept: */*

{"Authorization":"TK asdasdasdsadsadsad"}
```

Este pedido apresenta várias incorreções relativamente à aplicação dos principios REST e à estruturação habitual de pedidos HTTP (i.e, colocação da informação no formato adequado e na parte certa da mensagem).
a) complete o pedido HTTP seguinte de modo a que este seja uma reformulação correcta da mensagem anterior
R:
as boas práticas dizem que:
na URI deveia ser um GET
GET /alunos/list/ucs/concluidas HTTP/1.1
o token deve estar no cabeçalho e num campo chamado Authorization
e o numero do aluno devia estar no corpo do pedido ou colocar mesmo na URI, mas neste caso até podiamos fazer uso do token
e também existe um erro em /alunos/list/ucs/concluidas pois o list devia surgir no final. partir de uma coisa mais genérica e vamos afunilando

ou resposta possivel (com parametros):

```
GET /alunos/22232323/ucs/concluidas?year=2020-21 HTTP/1.1
Host: api.pd.com
User-Agent: curl/7.55.1
Student-number: 201012345
Accept: */*
Authorization:TK asdasdasdsadsadsad
```

ou

```
GET /alunos/22232323/ucs/concluidas?year=2020-21&nota-minima=15 HTTP/1.1
```

b) complete a mensaem HTTP seguinte, sendo este um exemplo de resposta ao pedido anterior e considerando que é devolvida uma lista com duas unidades curriculares, sendo cada uma caracterizada por apenas dois atributos: cod_uc e nota. Pode atribuir qualquer valor aos atributos desde que façam algum sentido no contexto da pergunta

```
HTTP/1.1 ??????
Content-type: application/json
Date: Tue, 2 Fev 2021 10:13:05 GMT
?????
```

r:

```
HTTP/1.1 200 OK
Content-type: application/json
Date: Tue, 2 Fev 2021 10:13:05 GMT
[{"cod_uc":"1111","nota":"11"},{"cod_uc":"2222","nota":"12"}]
```

cuidados:
ao invés de: Content-type: application/json
pode ser application/txt
ou pode ser application/html

==prática)
2. explique, sem recorrer a código, quais são as ações principais que o método stop realiza ao ser invocado no pedaço de código abaixo (nota: não é o objectivo do méotod, ou seja, parar um senor remoto com código igual a 2, que deve explicar)

```
import Java.rmi.*
Remote ref = Naming.Lookup("rmi://193.138.11.34/sensors_controller");
ControllerInterface remoteSenosrs = (ControllerInterface) ref;
remoteSenosrs.stop(2);
```

a resposta deve obrigatoriamente mencionar os termos "serviço RMI", "TCP", "porto de escuta automático", "endereço de IP", "serialização binária" ou algo relacionado, "pedido" e "resposta", entre outros possiveis. Quando referir os termos "porto de escuta automático" e "endereço IP" e "endereço IP" deve igualmente indicar onde os respetivos valores estão guardados. seja analitico e apresente a resposta soba forma de uma sequência ordenada de acções/passos (10 no máximo)

r:

==prática)
1. Pretende-se que desenvolva, em linguagem Java, um método atendendo aos

```
seguintes requisitos:
Protótipo do método pretendido:
void processRequest(Socket s) throws Exception;

O socket TCP s já se encontra criado e conectado a um par remoto, estando pronto para o envio e recepção de dados;
O médoto processRequest vai recebendo objectos serializados do tipo Request através do socket s até que ocorra uma excepção qualquer;
A classe Request possui, entre outros, os métodos int getUdpPort(), String getIpAddress() e String getMsg();
Para cada objecto Request recebido, a string devolvida pelo método getMsg() é enviada, através do protocolo UDP (DatagramSocket) e em formato texto (i.e., sequência de caracteres) para o destino com o endereço IP e o porto estipulados;

Quando ocorre uma excepção de qualquer tipo, o método processRequest:
Encerra o socket s;
Apresenta a mensagem associada à excepção;
Volta a lançar a excepção, o que faz com que termine.

import java.net.*;
void processRequest(Socket s) throws Exception

{
. . .
while(true)
{
/* Receive a serialized Request object */
. . .
/* Transmit the specified message to the specified IP
address and UDP port.
The message is transmitted as a sequence of characters (i.e., it is not transmitted as a serialized String object).

*/
. . .
}
}

r1:
para:
O médoto processRequest vai recebendo objectos serializados do tipo Request através do socket s até que ocorra uma excepção qualquer;
temos:
criar um objecto inpuToStream
e vamos fazendo read object com cast para Request

para:
Para cada objecto Request recebido, a string devolvida pelo método getMsg() é enviada, através do protocolo UDP (DatagramSocket) e em formato texto (i.e., sequência de caracteres) para o destino com o endereço IP e o porto estipulados;
temos:
pega no Request e no getMsg() e getBytes()
e de seguida construir um dataGramPacket com o inetadress passado o byname o getIpAddress() e getUdpPort() criar o DatagramSocket e enviar
e apanhar as excepções


r2 final:
import java.net.*;
class Exame{
void processRequest(Socket s) throws Exception
{

if(s==null){
//return;
throw new NullPointerException("socket s passado como null");
}

DatagramSocket ds =null;//crio e vi sempre usado o mesmo socket
//e foi passado para fora do try para poder criar a excepção
try{
ObjectInputStream oin = new ObjectInputStream(s.getInutStream()); //aqui porque só se abre um ficheiro uma vez e já ta
```

```
403   ds = new DatagramSocket();
404
405   //para usar o DatagramPacket
406   while(true)
407   {
408   /* Receive a serialized Request object */
409   //
410   Request req = (Request)oin.readObject();
411
412   . . .
413   /* Transmit the specified message to the specified IP
414   address and UDP port.
415   The message is transmitted as a sequence of characters (i.e., it is not transmitted
      as a serialized String object).
416
417   */
418   //se fosse necessário sair do ciclo
419   if(req.getMsg().equals("EXIT"){
420   return;
421   }
422
423   DatagramPacket pkt =  new DatagramPacket(req.getMsg().getBytes(),
      req.getMsg().getBytes().lenght, InetAddress.getByName(req.getIpAddress()),
      req.GetUdpPort() );
424   //ObjectInputStream oin = new ObjectInputStream(s.getInutStream());//sai daqui
425
426   //agora o datagramSocket para enviar a mensagem
427   //DatagramSocket ds = new DatagramSocket(); //sai daqui
428   //enviar a mensagem
429   ds.send(pkt);
430
431   }
432   } catch(Exception e){
433   System.out.println("Excepcao :"+ e );
434   //e para lançar a excepção é preciso ainda fazer:
435   throw e;
436   }
437   finally{
438   try{
439   s.close(); //este try abafa a IOException que for gerada, apenas aqui
440   }catch(IOException e){}
441   if(ds!=null){ds.close();}
442   }
443   }
444   }
445
446   //tudo que é feito no finally é feito ao inves de usar
447   catch(Exception e){
448   s.close();
449
450
451
452   }
453
454
455   r3:
456   e se no socket podermos receber objectos do tipo Request ou do tipo XPTO
457   import java.net.*;
458   class Exame{
459   void processRequest(Socket s) throws Exception
460   {
461   . . .
462   while(true)
463   {
464   /* Receive a serialized Request object */
465   ObjectInputStream oin = new ObjectInputStream(s.getInutStream());
466   Object obj = oin.readObject();
467
468   if(obj instanceof Request){
469       ((Request).oin).getUdpPort();
470   }else if(obj instanceof XPTO){
471       ((XPTO).oin).getUdpPort();
472   }
```

```
. . .
/* Transmit the specified message to the specified IP
address and UDP port.
The message is transmitted as a sequence of characters (i.e., it is not transmitted
as a serialized String object).

*/
. . .
}
}
}

==prática)
3. (RMi) Pretende-se que desenvolva um serviço remoto Java RMI, com um mecanismo de
callback, atendendo aos seguintes requisitos:
Nome da classe que representa o serviço: MessageReflector;
Interface remota implementada pelo serviço MessageReflector:
Nome: MRInterface;
Métodos:
boolean registerClient (MRClientInterface cliRef);

Se ainda não existir, acrescenta a referência RMI passada como argumento a uma lista
interna e devolve true. Se já existir, ignora o pedido e devolve false.
Notas:
1. Em vez de uma lista, também pode recorrer a um conjunto;
2. Cada cliente implementa um serviço RMI baseado na interface remota
MRClientInterface.

boolean unregisterClient (MRClientInterface cliRef);
Se existir, remove a referência RMI passada como argumento da lista interna e
devolve true. Caso contrário, devolve false.

void broadcastMessage(String msg);
Comunica a string passada como argumento a todos os clientes registados. Para o
efeito, invoca o método void postMessage(String msg) pertencente à interface remota
MRClientInterface. Qualquer problema que surja na invocação do método
postMessage leva à eliminação da respectiva referência remota (i.e., do cliente) da
lista interna.

int getNumRegisteredClients();
Devolve o número de clientes registados (i.e., o número de referências remotas
existentes na lista interna).

r:
Class MessageReflector extends unitCastRemoteObject implments MRInterface {
//fazer o construtor vazio
MessageReflector{}
//e os métodos
boolean registerClient(){}

//enivar a todos
com broadcastMessage e ir ao arrayList e chamar

}

r2 final:
notas:
os clientes devem registar.se, isto é , invocar um método do serviço
não é preciso o main do servidor

//a base sem RMi
interface MRClientInterface extends Remote{
    void postMessage(String msg) throws RemoteException;
}


public interface MTinterface{

//métodos
boolean registerClient (MRClientInterface cliRef);
boolean unregisterClient (MRClientInterface cliRef);
void broadcastMessage(String msg);
```

```
536   int getNumRegisteredClients();
537
538   }
539
540   //classe que implementa a interface
541   class MessageReflector implements MRInterface
542   {
543   //lista de referencias remotas para os clientes
544   List<MRClientInterface> clientList = new ArrayList<>();
545
546   @Override
547   public boolean registerClient (MRClientInterface cliRef){
548   return false;
549   }
550
551   @Override
552   boolean unregisterClient (MRClientInterface cliRef){
553   return false;
554   }
555
556   @Override
557   void broadcastMessage(String msg){
558   }
559
560   @Override
561   int getNumRegisteredClients(){
562   return -1;
563   }
564
565
566   }
567
568   //agora aplicar o RMi
569   interface MRClientInterface extends Remote{
570       void postMessage(String msg) throws RemoteException;
571   }
572
573
574   public interface MTinterface extends Remote{
575   //métodos
576   boolean registerClient (MRClientInterface cliRef) throws java.rmi.RemoteException;
577   boolean unregisterClient (MRClientInterface cliRef)throws RemoteException;
578   void broadcastMessage(String msg) throws RemoteException;;
579   int getNumRegisteredClients() throws RemoteException;;
580   }
581
582   //classe que implementa a interface, e agora já é um serviço e surge extends
      UnicastRemoteObject
583   //servidor concorrente UnicastRemoteObject
584   //e cada cliente que se ligue vai ser criada uma thread
585   class MessageReflector extends UnicastRemoteObject implements MRInterface
586   {
587   //lista de referencias remotas para os clientes
588   List<MRClientInterface> clientList = new ArrayList<>(); //este atributo é mexido por
      várias threads, vou então serializar
589
590   //e falta o construtor
591   public MessageReflector () throws RemoteException{}
592   //ou public MessageReflector (List<Qualquercoisa>l) throws RemoteException{}
593
594   //e em todos os métodos da interface remota surge throws RemoteException
595   @Override
596   public synhcronized boolean registerClient (MRClientInterface cliRef) throws
      RemoteException{
597   //se já existe, do tipo equals
598   if(clientList.contains(cliRef)){
599   return false;
600   }
601   return clientList.add(cliRef);
602   }
603
604   @Override
605   boolean synhcronized unregisterClient (MRClientInterface cliRef) throws
```

```
    RemoteException{
606 return clientList.remove(cliRef);
607 }
608
609 @Override
610 void synhcronized broadcastMessage(String msg) throws RemoteException{
611 //versão1
612 for(MRClientInterface cli: clientList){
613     try{
614     cli.postMessage(msg);
615     }catch(RemoteException e){
616         clientList.remove(cli);
617     }
618     }
619 //versão2 com indices
620 for(int i = 0 ; i<clientList.size();i++){
621     try{
622         clientList.get(i).postMessage(msg);
623     }catch(RemoteException e){
624         clientList.remove(i--);
625         //i--;
626     }
627     }
628 }
629
630 @Override
631 int synhcronized getNumRegisteredClients()throws RemoteException{
632 return clientList.size();
633 }
634
635
636 }
637
638 //ou synhcronized por bloco
639 @Override
640 public  boolean registerClient (MRClientInterface cliRef) throws RemoteException{
641 synhcronized{
642
643 }
644 return false;
645 }
646
647
```