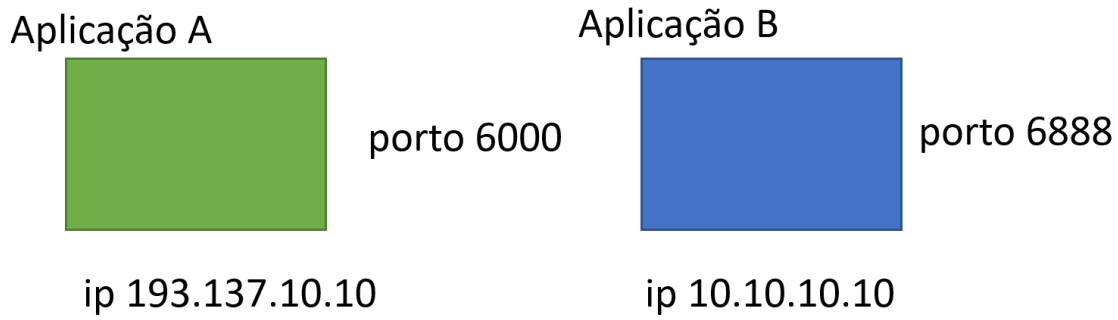
#sockets Java
na comunicação na internet entre aplicações existem dois protocolos: o UDP e o TCP
o ip identifica a máquina, e o porto ajuda a app a saber que a comunicação é para essa app



UDP - enviamos mensagens (datagramas) e que inclui:
dados da mensagem (conteúdo) e um envelope com ip, porto de destino e IP porto de origem
UDP não faz verificação, são enviadas cartas (como se fosse um envelope)

TCP - temos um fluxo de BYTES, diferente de UDP

existem classes em java importantes para proceder à comunicação:
inetadress, manipula relações com endereços e resolução de nomes
datagram packt, tem atributos e métodos e que vai ser encapsulada num datagramPackt
datagram socket, só o UDP tem isto porque se trabalha com datagrams

o Inetadress:
encapsula endereços IP
não tem construtor
tem métodos estáticos

getByName(IP ou nome)
métodos a aplicar depois:
getHostAddress
getHostName
getAllByName(array)

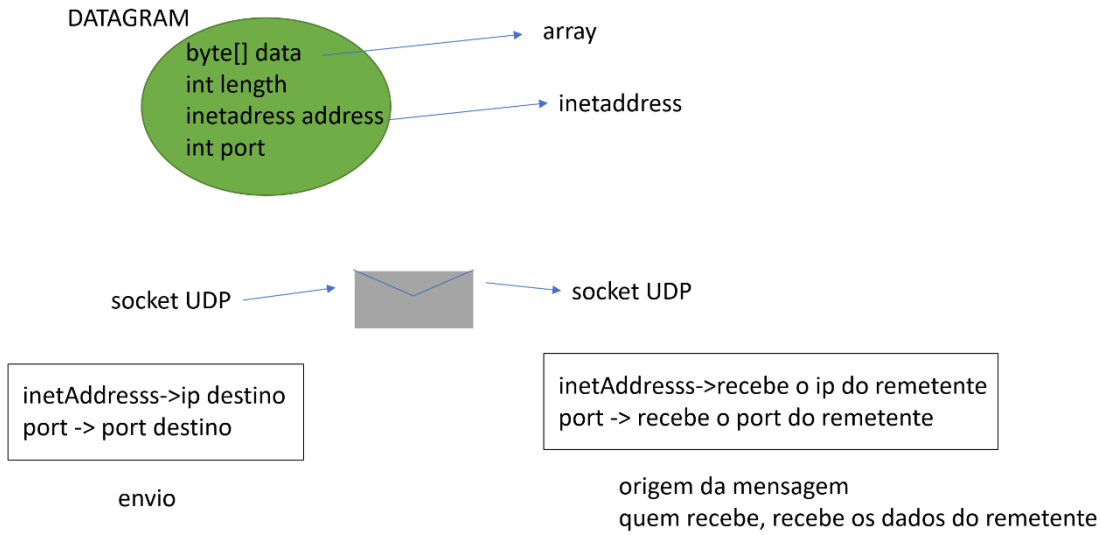
para saber qual o meu ip:
inetAdress localAdress = InetAddress.getLocalHost();
e uma exceção:
catch(UnknowHostException e){
...
}

DataGramPacket
encapsula as mensagens individuais
construção de um datagram UDP para envio

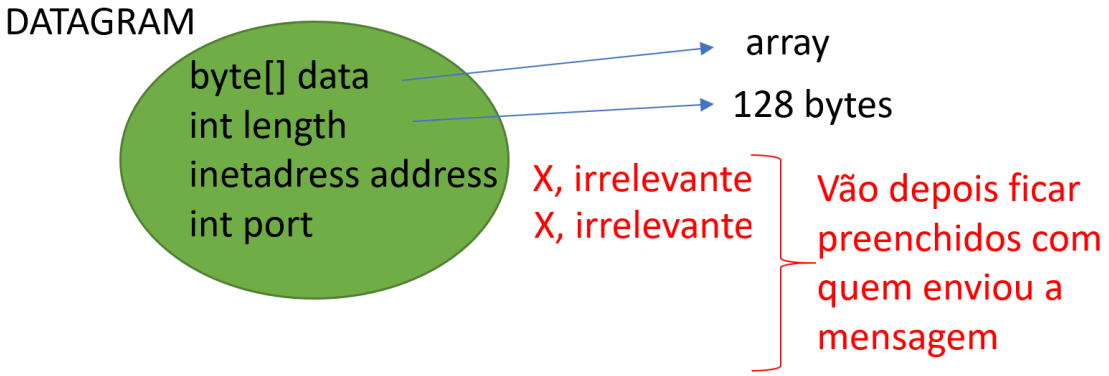
Uma mensagem tem um endereço ip, um porto e uma mensagem (conteúdo)

DataGRamPAcket packet = new DataGramPAcket(data, data.length, addr, 2000);
data vem byte [] data = new byte[128]
data.length vem 128

addr vem InetAdress addr = InetAdress.getByName("192.168.0.1");
2000 vem o porto



Operação de receção de um DATGRAM PACKET



assim no recetor uso outro construtor

try{
DATAGRAMPACKET packet = new DTAGRAMPACKETS (new byte[256], 206);
DATAGRAMSOKCET socekt = new DATAGRAMSOCKET(2000);
boolean finished = false;
while(!finished){
socket.receive(packet);
...

}
socket.close();
}catch

#protocolo UDP
a class DATAGRAMPACKET tem vários métodos:
send
receive (bloqueante)

close
setTimeout(0), desbloqueio o TIMEOUT é um no TIMEOUT

o lenght, consoante o contexto de utilização um determinado packet pode significar:
quantidade para enviar
quantidade para receber/aceitação
quantidade que foi recebida

#exemplo como enviar uma mensagem:
DATAGRAmSOCKET socket = new DATAGRAMSOCKET();
DATAGRAMPACKET packet = new DATAGRAMPACKET(new byte[256], 256);
packet.setAddress(INETADDRESS,getByName(someHost));
packet.setPort(2000);
boolean finished = false;
while(!finished){
//-> escrever no buffer do packet
seocket.send(packet);
//-> verificar se existe mais coisas para enviar
}
socket.close();

#protocolo TCP
relembrar que no UDP:
DATGRAM PACKT, que encapsula uma mensagem com endereços (endereço, porto, dados recebidos)
DATAGRAM SOCKET, end point de uma comunicação, onde enviamos e recebemos bytes
ÎNETADRESS, serve para encapsular o endereço ip e os nomes do dns

Protocolo TCP:
não é orientado a mensagens;
serve para comunicação ponto a ponto (entre dois extremos)
abstração do socket (ligação virtual)
o socket está ligado a um endereço remoto (end point), surge o conceito de fluxo de dados
o que vai ser colocado no buffer são bytes e não mensagens (é o mesmo que escrever num ficheiro)
enviar bytes para uma aplicação remote, como se estivesse a escrever num ficheiro

UDP não faz controlo de fluxo, não verifica o numero de sequência, não deteta se existem mensagens perdidas, não pede retransmissão (envio uma carta e esqueço a carta)

TCP, as instâncias de TCP, trocam informações, mensagens de controlo, confirmações, timeouts, para garantir que os bytes são colocados no socket de destino pela mesma ordem e sem haver perda de bytes no meio

#entrada/saída
input stream e output stream, são classes abstratas cujos métodos não estão implementados

vão-se é obter subclasses que vão implementar métodos.

INPUTSTREAM (vou ter um read) e OUTPUTSTREAM (vou ter um write) geram exceções IOException

usar SCOKET ou ArrayBytes ou Ficheiro, vai ser indiferente

pseudocódigo:
a conexão (servidor, recebe a ligação)
xxServerSocket // Socket socket = new Scoket ("www...", 80);
a transferência
xx Socket

outros exemplos:
socket (host, port);
socket (host, port, bindAddress, localhost)

onde bindAddress, especifica-se a placa (por exemplo um das duas placas do portátil, rede sem fios)

onde localhost, porto do cliente

por exemplo: socket(..., ..., InetAddress.getByNumber("10.10.10.10"), 5001);
a entrada e saída de dados tem que ser feita através desta placa de rede, mas pode acontecer que não queremos restringir placas mas fixar porta 5001, então surge
socket(..., ..., InetAddress.getByNumber("0.0.0.0"), 5001);

exemplo no cliente:
```
try{
Socket mySocket= new Socket("www.sol.pt", 80);
....
}catch(Exception e){
System.err.println("Err - " + e);
}
```

O socket TCP, tem métodos:
getOutPutStream()
getInputStream()
getLocalPort
getPort, acesso ao porto remoto
GetTimeout()

por exemplo no cliente:
```
try{
Socket socket = new Socket(someHost, somePort);
BufferReader reader = new BufferReader(new InputStreamReader(socket.getInputStream()));
PrintStream pstream = new PrintStream(socket,getOutPutStream());
}catch(IOException e){
System.err.println("erro - " + e);
}
```

em que: socket.getInputStream(), faço o read
em que: socket,getOutPutStream(), faço write bytes

por exemplo, operação de leitura e que não fique bloqueado para sempre:
```
try{
Socket socket = new Socket(someHost, somePort);
socket.setTIMEOUT(2000); // onde setTIMEOUT(0);//desliga-se o TIMEOUT
BufferReader reader = new BufferReader(new InputStreamReader(socket.getInputStream()));
PrintStream pstream = new PrintStream(socket,getOutPutStream());
}catch(Interrupt IOException e){
```

```
timeoutflag = true;
}catch(IOException e){
System.err.println("erro - " + e);
System.exit(0);
}
```

#protocolo TCP
serverSocket(porta) e tem alguns métodos como é o caso do close()

o serverSocket não serve para enviar dados e receber, este só serve para aceitar ligações

#aplicações multicast
endereços de classe D (11110)
representam grupos (ou endereços)
como enviar para um grupo e como receber?
escrever via broadcast (255.255.255.255 6001)
assim todas as aplicações que estão a correr na mesma máquina no domínio de difusão e que tenham um socket associado ao porto 6001 vão receber a mensagem

exemplo:
InetAddress group = InetAddress getByName("224.1.2.3");
MulticastSocket socket = new MulticastSocket(port); //port é o porto de escuta
socket.joinGroup(group); //joinGroup(group) para apanhar o que aparece na rede destinada ao 22.4.1.2.3 port
socket.setTimeToLeave(1); // setTimeToLeave, tempo de vida da mensagem, neste caso (1) elimina, e fica a zero e não reencaminha

e para enviar e receber:
DATAGRAM packet = new DATAGRAM (buffer, buffer.lenght, group, port);
socket.send(packect);
//como pertenço ao mesmo grupo, também recebo
byte[] response = new byte[1024];
DATAGRAM packet = new DATAGRAMPACKET(response, response.length);
socket.receive(packet);

Exemplo:
String NICID = "127.0.0.1","en0",...
NETWORKINTERFACE nif;
try{
nif = NETWORKInterface.getByInetAddress.getByName(NICID));
}catch(Exception ex){
nif = NETWORKInterface.getByName(NICID);
}
socket = new MultiCast(port);
socket.joingroup(new InetSocketAddress(group, port), nif);

em que:
joingroup, associo a um grupo
InetSocketAddress, completo IP+porto
nif, qual a interface de rede a associar ao grupo

#serialização de objectos
Esta é a forma mais simples de trocar informação

A informação está encapsulada em objectos
Serialização vs DesSerialização

a Serialização é binária, mas exibe em formato de texto como o JSON

#serialização de objectos, ligações TCP
exemplo:
s = new Socket(serverAddress, serverPort); // estabelecer a ligação
in = new ObjectInputStream(s.getInputStream());
out = new ObjectOutPutStream(s.getOutPutStream()); //enviar
out.writeObject(object to transmit); //escrever no object
//ou out.writeUnShared(object to transmit); //deve ser usado, por exemplo para enviar um arraylist então o usado é o unshared para evitar caching
out.flush();

ou usar writeobject mas fazer antes o out.reset() e ficaria:
out.reset();
out.writeObject(object to trnsmit);

//e do outro lado faz-se a leitura
returnObject = (MyClass)in.readObject();

#serialização de objectos ligações UDP

o UDP não tem fluxo de bytes, datagrams)

exemplo para o envio:
s = new DatagramSocket();
bout = new ByteArrayOutputStream();
out = new ObjectOutputStream(bout);
out.writeObject(Object to transmite);
//ou
out.writeUnshared(object to transmite);
out.flush();
packet = new DatagramPacket(bout.toBytetoarray(), bout.size(), serverAdress, serverPort);
s.send(packet);

exemplo para receber:
packet = new DATAGRAM(new Byte[MAX_SIZE], MAX:SIZE);
s.receive(packet);
n = new ObjectInputStream(new ByteArrayInutStream, packet.getData), 0, packet.getLenght());
returnObject = (MyCalss) in.readObject();

#threads
processo é diferente de thread
todas as threads pertencem ao mesmo processo, as threads são linhas de execução

as threads podem ser iniciadas em dois modos distintos:
modo por omissão, modo normal (modo utilizador) // bloqueante
modo através do daemon(serviço para correr em background) // não bloqueante

Uma aplicação em java termina quando já não existe qualquer, em que já não existem threads em modo utilizador activas.

As thraeds activas em modo de utilizador, mesmo chegando ao fim do main elas continuam activas, se forem as threads em modo daemon, chegada ao fim do main elas terminam

```
public class RunnableThreadDemo implements java.long.Runnable
{
public void run(){
SystemOut.println("....");
}
public static void main (String args[]){
System.Out.println("...");
Runnable run = new RunnableThreadDemo();
System.Out.println("...");
thread t1 = new Thread(run, "thread 1"); //run, é o nome da thread
System.Out.println("...");
thread t2 = new Thread (run, "trhead 2"); //run é o nome da thread
System.Out.println("...");
t1.start();
t2.start();
}
}
```

a diferença entre os dois modo de criar threads pode ser importante
assim o método II é melhor que o método I, já que posso criar várias threads baseadas no mesmo objecto runabble e elas vão partilhar os mesmos membros. posso assim ter várias threads concorrentes mas a trabalhar sobre os mesmos dados.
E outra vantagem é contornar uma das limitações do já de não haver herança múltipla

class ExtendThreadDemo extends java.long.thread // esta classe não pode estender outro tipo de objecto

assim consigo dar a volta a isso fazendo/usando:
class ExtendThreadDemo implements java.long.thread

eu posso implementar vários tipos de classes mas só posso derivar de uma classe

threads, faz-se o new thread mas depois tem que se fazer o start, e só depois do start o new cria um objecto que representa uma thread, mas o start lança de forma efectiva

#interromper threads

é desaconselhável

thread sleep- > interreupt -> resume
alternativa é a thread que a lançar, fechar a socket, ao fechar é gerada uma exceção na thread e antes de fechar o socket eu coloco um atributo na thread para ela saber que é para terminar

#parar threads usar o stop

fazer uso de uma boolean e "avisar" que vai ter que encerrar

#operações sobre threads

verificar se ainda está viva:
if(t1.isAlive(){ //booleand
...
}

e nunca usar um while..

#método join

bloqueia, e só regressa depois da thread estar inactiva
t1.join(1000,10000);
em que 1000, é o numero máximo de segundos milésimo de segundos
100000, são nanosegundos

exemplo:
thread die = new WaitForDeath(); //a thread
die.start(); //inica a thread
die.join(); // bloqueia ate a thread terminar

#sincronização de threads

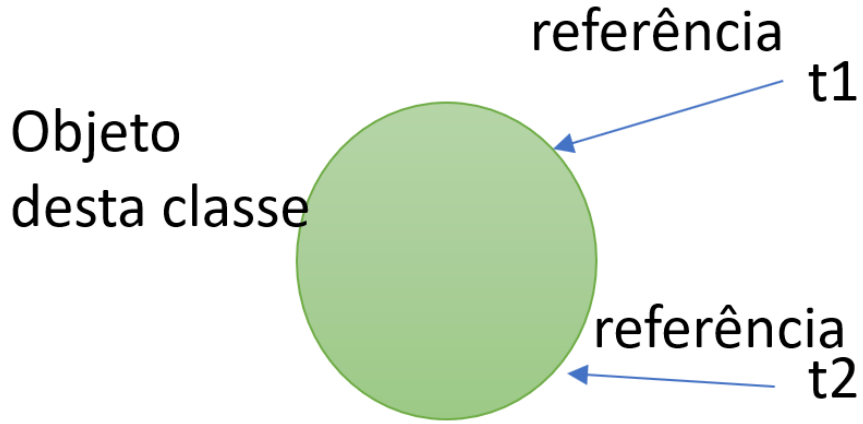
concorrência no acesso a recursos partilhados
solução: serializar o acesso ao recurso partilhado, isto é, enquanto uma thread está a mexer nos recursos, as outras threads que pretendam aceder aos recursos ficam bloqueadas, ou seja, surge então o objecto sincronização

acesso mutuamente exclusivo
assim se surgir outra thread que quer aceder a métodos não podem executar esse ou outro métodos do objecto

a sincronização pode se feita:
método // bloqueia apenas alguns dos métodos
bloco de código

#sincronização ao nível do método// thread safe

public synchronized void mudaData() {...}



public synchronized void changeData() {...}

exemplo:
public class someClass{
public synchonized void ChangeData(){...}
}

O = new someClass();
t1 e t2 têm a referêcnia para O
t1 chama o método changeData
e se t1 chamar o méotodo changeData, fica bloqueada

#sincronização ao nivel do bloco de código

sincronização referente a um bloco
não é necessário os métodos serem synchronized
existe o thread-safe (serializado) sem necessidade de ter acesso ou alterar o código fonte
synchronized (Object O){
...
}

esta forma dá mais flexibilidade

se for só um classe minha será mais fácil colocar o método sincronized mas se for uma que já existe será o modo bloco

#grupos de threads
é limitado, um conjunto de operações limitado

#comunicação entre threads
no mesmo processo, no mesmo espaço de endereçamento é feito através de pipes de comunicação:
_troca directa de dados entre threads
_comunicação bidirecional
(temos usado um BufferedReader bin = ...

#notificação entre threads
surgem os métodos:
Object.wait();
Object.notify();
Object.notifyAll();

#prioridades das threads
abordagem round-robin
prioridades variam entre:
1 mais baixa
10 mais alta

com uso de:
MAX_PRIORITY
NORM_PRIORITY
MIN_PRIORITY

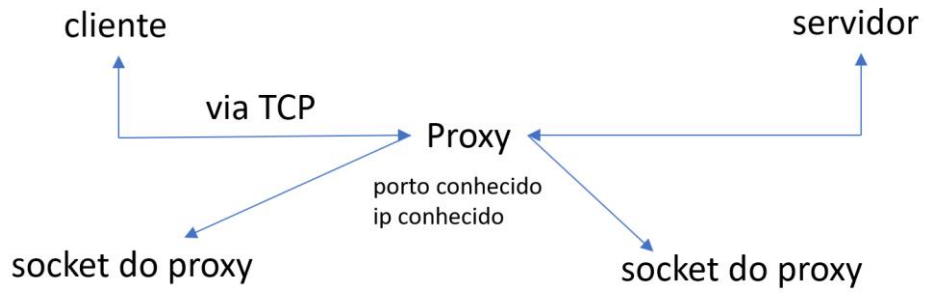
métodos:
t.setPriority(thread.MIN_...)

#servidores TCP concorrentes

while(true){
cliente = socket.accepts();
t = new thread();
t.start();
...
}

#computação paralela baseada em múltiplas threads
processamento real

#exercicio
resolver um proxy: é uma aplicação que serve de intermediário entre duas aplicações em rede usando TCP: (guarda log dos acessos)



o proxy para fazer logs do que é trocado

um proxy para servidor é uma ponte. o servidor do mail do isec pop-isec.pt:110

um server socket no porto 8080, e crio uma thread que vai tratar o que se pasa no socket, mas essa thread tem outra socket para comunicar com o servidor de mail do isec (cliente-servidor)
para cada cliente exist uma thread cirada
essa thread tem um socket para comunicar com o cliente
essa thread tem um socket para comunicar com o pop.isec (servidor)

mais ainda não se consegue receber um char e reencaminhar porque existe um sistema de bloqueio de espera de mensagem no socket
usar um timeout não se pode fazer porque é espera activa, assim para cada cliente vai ser necessário duas threads, e as duas threads têm acesso aos mesmos sockects
uma thread tem acesso a ler bloqueada num socket e aquela que recebe envia para outro socket e outra faz o inverso

#arquitectura n-tier
separação lógica e física
a arquitetura 3-tier é mais habitual:
apresentação
lógica (do negócio)
dados (MySQL)

o 3-tier, é parecido com o MVC:
camada de apresentação -> cliente
camada lógica -> SGRDS e servidores
camada de dados -> usar o MySQL

#JDBC
java database connectivity

```
Connection con = DriverManager.getConnection(BD, username, password);
statment stmt = con.createStatment();
resultset rs = stmt.executeQuery("select * from table;");
while(rs.next){
int x = rs.getInt("a");
String s = rs.getString("b");
float x = rs.getFloat("c");
}
```

porque é que temos que usar drivers diferentes para bases de dados diferentes? se tiver que usar SQL, porque é que tenho que usar um driver diferente se tiver que usar ORACLE?
porque o protocolo de comunicação é diferente

#inovação remota de métodos
o middleware, sistema que vai abstrair o programador dos aspectos de baixo nivel, nomeadamente de:
troca de mensagens/comunicação
protocolo de comunicação
sistemas operativos
hardware

exemplos de middleware:
RMi
RPC
CORBA
NetRemoting

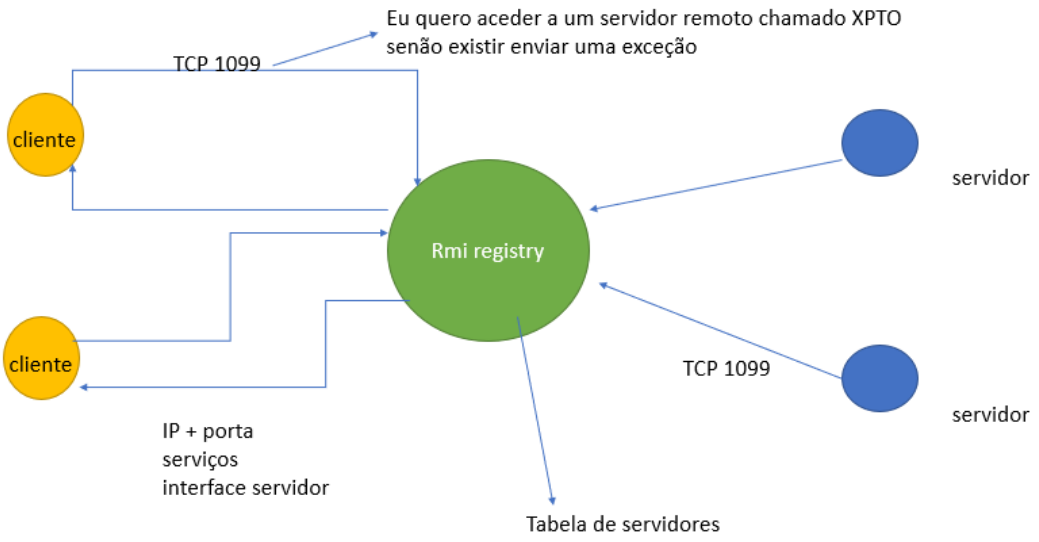
um proxy que tem os métodos, e eu localmente tenho um objecto que tem os mesmos métodos
RPC é o inicio da história para o paradigma dos sistemas distribuídos (miiddleware)

os passo do RPC
passo1: o servidor fica a correr e vai-se registar na porta 111
passo2: o cliente solicita o porto do servidor
passo3: recebe a porta
passo4: faz o pedido
passo5: recebe a resposta

#introdução ao java RMi
agora ao invés de se chamar funções, vou chamar métodos usando objetos remotos

o input ao sistema é uma interface (conjunto de protótipos de métodos)

#arquitectura java RMi
existem servidores, clientes, RMI registry



os passos do RMi:
1_ o servidor regista no Rmi registry
2_ o RMiregistry providencia referências ao cliente

o que é que um serviço e um proxy têm em comum?
a interface, os métodos que implementam

o Sturb (ou proxy) é do lado do cliente e actua como um proxy. Implementa neste a interface remota, é invocado pelo cliente, envia mensagem ao serviço.
o Skeleton é do lado do servdor, aguarda pedidos na aplicação servidora, invoca o método pretendido, devolve o resultado ao Sturb.

A comunicação é feita via serialização e por isso todos os argumentos que se passam em funções e resultados têm que ser serializados

porque razão é que o java RMi nas interfaces remotas tem que ter os argumentos e os resultados serializados?
porque o sistema, invocamos o método remoto e o que acontece é que é enviado um objecto que representa um pedido para o servidor, e o RMi faz a serialização dos objetos logo o pedido vai ter a identificação do método, etc, e assim tem que ser tudo serializado

#Arquitectura RMi
permite chamar os métodos de um serviço que não está na minha máquina, sendo que existem três componentes principais e que são: servidor RMi, cliente RMi, RMi registry

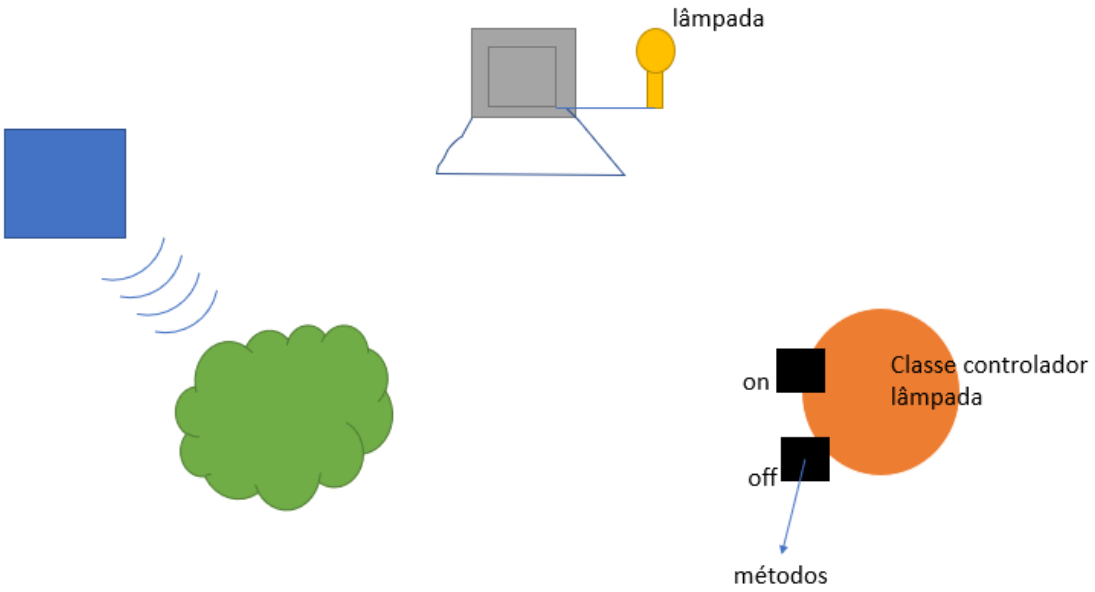
o servidor RMi: usa TCP, com um server socket à escuta num porto, é concorrente (sempre que existe um pedido de ligação cria uma thread, essa thread recebe pedidos, vê qual o objecto e qual o método para executar, e executa e devolve o resultado na origem.) No servidor só temos que especificar o que o método faz

o cliente RMi: aplicação que tem referências para objectos remotos. O cliente quando contacta o RMi registry recebe o Sturb. E no Sturb tem dois membros obrigatórios: o ip da maquina e o porto de escuta

o RMi registry: é o serviço que está à escuta num porto automático TCP, sendo que os clientes contactam o RMi Registry TCP

#Passos principais
1) definir a interface remota: métodos, com argumentos e valor de retorno (Sendo que tudo o que é valor de retorno tem que ser serializavel)
2) implementar o serviço: ter classes que implementam a interface
3) é preciso um servidor (uma aplicação que cria objecto, vai registar no RMi registry)
4) os clientes (é uma aplicação que tem que invocar o método no serviço XPTO, assim em vez desta aplicação fazer um NEW, vai obter uma referência remota, contact o servidor, e se o serviço existir eu obtenho o sturb/proxy (referencia remota))
5) lançar o RMi registry (1099, TCP)
6) lançar o servidor
7) correr os clientes

#interfaces remotas
por exemplo controlar uma lâmpada remotamente



surge então uma aplicação em java, em que no main faz-se um NEW controlador (que poderia ter alguns parametros) e nesta app apenas surge a opção do menu de ligar e desligar.
ainda no main do servidor, teria que criar um server socket que vai fazer accepts, se quiser qe seja concorrente lanço threads
então e na comunicação posso usar mensagens curtas, booleans, ou objectos serializados com mais campos

do lado do cliente existe um socket que constroi os pedidos
usando o RMi vamos abstrairnos desta situação, com o RMi vai ser mais simples pois não se pensa em mensagens, mas sim nos servidores, e quais os métodos que temos nos servidores e as apps clientes vão invocar métodos do utilizador (gets, sets, ..) (por baixo existe o TCP, serialização, mas não somos nós a tratar disso)

No RMi temos que obrigatóriamente usar em todos os métodos o throws
java.rmi.remoteException
public void on() throws java.rmi.remote.exception

e as interfaces remotas têm que ser:
public interface Classe extends java.rmi.remote {
public void ont() throws java.rmi.remote.exception

muito importante:
throws java.rmi.remote.exception
e
public interface Classe extends java.rmi.remote

assim:
1) no servidor existem métodos
2) os clientes vão invocar os métodos no utilizador (gets, sets...)

no servidor java RMi, vai pensar no que quero que seja invocado remotamente

no servidor, exemplo:
public interface RMiLight extends java.rmi.remote{
public void on() throws java.rmi.remoteException;
public void off() throws java.rmi.remoteException;
public void isOn() throws java.rmi.remoteException;
}

os métodos podem lançar para fora excepções que eu entender, em que, e é muito importante:
todos os métodos têm que lançar a expeção java.rmi.exception
as interfaces remotas extends java.rmi.remote

implementar o serviço, exemplo:

public class RMiLight extends java.rmi.unicastRemoteObject implements RMiLight{
public RMiLight()throws java.rmi.remote.exception{
...
}
public void on()throws java.rmi.remoteException {
...
}
}

em que:
public class RMiLight extends java.rmi.unicastRemoteObject implements RMiLight é um servidor base que não executa nada TCP
public RMiLight()throws java.rmi.remote.exception é um construtor obrigatorio. os construtores não aparecem na interface remota, mas os serviços remoto precisam de construtor que lançam excepções remotas (nem que seja vazio)

nos serviços tenho:
os métodos que pretendo que sejam invocados remotamente

só os métodos que fazem parte da interface remota é que precisam do RemoteException?
assim os outros métodos do serviço que não fazem parte da interface remota já não vão dar

assim ao criar uma instância da classe RMiLightBullImp no servidor vem que, exemplo do servidor:
public class void main(){
try{

```
//lançar uma thread em ciclo que cria +1 thread para cada cliente em paralelo que espera pedido de ligação
RMi bs = new RMiLightImp();
RemoteRef location = bs.getRef(); //não é necessário
string registry = "localhost"; //ip do registry e nó do serviço
if(args.lenght >1) { registry = args[0]; }
string registration = "rmi//" + registry + "/RMiLightBulb"; //registo do serviço
naming.rebind(registration, bs); //RMi e serviço
}catch(Remote Exception e){
}
}
```

apesar de chegar ao fim do main, ainda existe uma thread no modo de utilizador, e as aplicações em java só terminam se não existir mais threads de modo utilizador a decorrer

as aplicações agora só precisam de saber qual o IP onde está e qual o porto do servidor

naming.rebind(registration, bs);
em que:
se estabelece uma ligação TCP
no porto 1099 (porto de escuta do RMi registry)
manda-lhe uma margem para registar como o nome RMi Light bulb a referencia ao objecto Bs

rmi://192.10.10.10/xpb
o que faz quando se faz naming rebind

isto não estabelece uma ligação com um serviço
naming.lookup(rmi://192.10.10.10/xpb);

o naming.lookup no cliente para ele obter a referência para um serviço com este nome

quando se faz o lookup o que é feito é uma ligação TCP com qualquer coisa no IP no porto 1099 que é do registry e quero obter uma referência para um objecto com o nome xpb, esta referência tem interface e tem IP e porto

qual a diferença entre o método Bind e Rebind? (Bind e Rebind, tem a ver com tabelas no registry)
Com o rebind se existir é substituído senão existir cria
com o bind se não existir cria e se já existir cria uma excepção

no cliente o que é que faz o ON em bullbService.on(); sabendo que existiu
SErviceRmi bullBservice = (serviçoRmi) remoteSErvice

o cliente não faz um NEW local do objecto o que ele faz é um naming.lookup
no cliente depois vou ter um objecto que implementa a interface do serviço (on, off, ...)
into é em sturb, e vai ser enviado ao fazer on, envia serializado a operação que tem que ser executada e que é o on, e o serviço recebe o pedido, executa e devolve o resutlado

qual é o aspecto comum entre proxy/sturb e o serviço?
é a interface, tem um conjunto de métodos coincidentes isto é, no serviço existe um on() e no sturb também existe um on() que na realidade envia uma mensagem para o outro lado para ligar

#cliente RMi

naming.lookup(registry);
isto faz uma ligação TCP com o RMi registry. faz um pedido do RMi registry e recebe um array de strings com o nome dos serviços registados
registry é o ip da maquina

relembrar que no RMi registry podemos solicitar:
lookup, pedir uma referÊncia para o objecto
bind/rebind, regista um serviço

#funcionalidades adicionais
para alunar uma referência que está na tabela
naming.unbind("rmi://192.0.0.1/RMiLightBuld")

para terminar no servidor e no main:
unexportObject(bulbSErvice, true);
em que bulbSErvice é o serviço

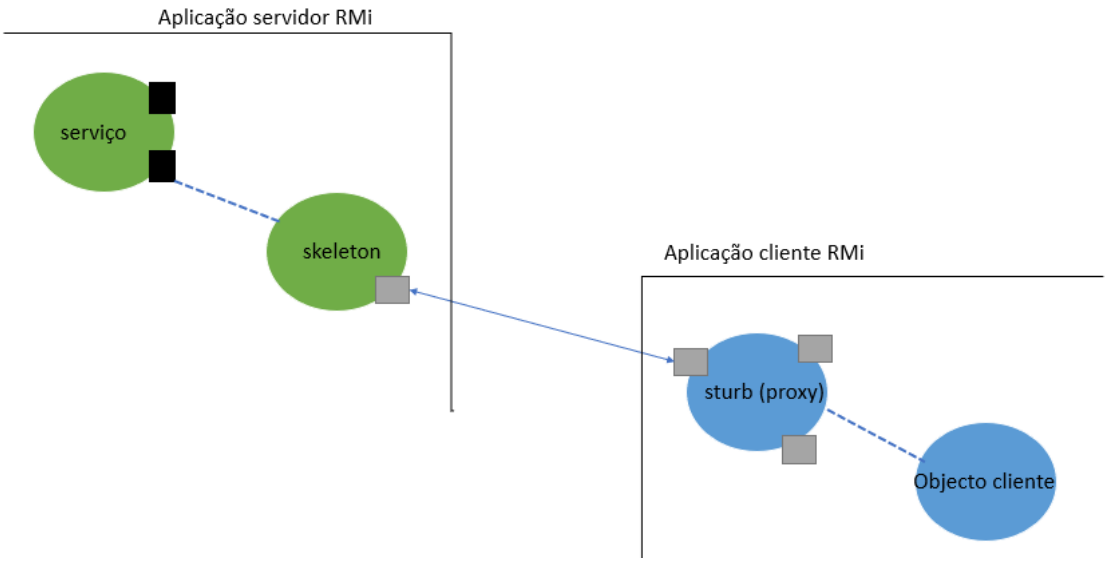
No servidor pode ser criado algo do tipo, prima qualquer tecla para terminar o serviço e com a inclusão do endpoint Object termina a thread em modo de utilizador
e só termina depois dos clientes serem atendidos, não é de forma abrupa, nas se for false termina logo tudo

o que significa?
endPoint(10.20,4,128.77:53561);
10.20,4,128.77 é o IP do serviço
53561 é o porto automático, criado pelos server socket
quando o cliente faz o lookup do registry obtêm uma referencia remota, o que se indica no lookup é o ip do registry e o porto (do registry) mas aqui é isto que eu obtenho

#funcionalidades adicionais
no servidor vou lançar o registry da maquina:
createRegistry
getRegistry

#invocação remota dos métodos
desenvolver aplicações remotas sem ter que se preocupar com sockets, sem usar o paradigma de troca de mensagens, sem usar TCP/UDP directamente
no cliente temos um Sturb (proxy), uma referência para o objecto remoto
no servidor existe o Skeleton, um servidor TPC, concorrente que recebe os pedisos, serializados, serialização binária do java.rmi, identifica qual o método que é para invocar, e argumentos, invoca e envia o resultado, e o método no cliente regressa e não é proxy

o que é que é comum entre o serviço (servidor) e a referência remota (Sturb, no cliente)?
é a interface implementada
o proxy não faz aquilo que é suposto o método fazerem, o que ele faz é pegar nos argumentos e envia para o servidor (Skeletton) que no servidor vai ser executado, que depois serializa e envia a resposta e o Sturb devolve a resposta ao cliente



quando o serviço é lançado (no servidor) ele regista-se no RMi registry (que é tipo um DNS) onde fica registado o IP e o porto onde pode ser encontrado o serviço

o cliente contacta o RMi registry à procura do serviço

o que acontece inernamente (por baixo) no cliente RMi quando ele faz so dos métodos, por exemplo em:
bulbSErvice.isOn()
é feito um pedido de envio
aguarda resposta
o pedido é serialziado
e é devolvida a resposta

situação de dois clientes acederem ao mesmo objecto ao mesmo tempo?
o cliente através do lookup contacta o registry (TCP, 1099) e envia o pedido para o que quer
o cliente depois recebe a referência remota do objecto, que tem a interface pretendida (métodos...) e também tem o ip e porto do serviço (serviço APi)
no servidor depois de contactado cria uma thread que vai cuidar desse cliente
e se existir outro cliente vai também ser o mesmo procedimento, e no serviço (servidor) é criada outra thread
assim eu tenho que verificar se é critico os acessos concorrentes, dois clientes têm acesso a um determinado serviço, estão a aceder ao mesmos erviço e mesmos recursos, então eu terei que colocar nas classes concretas/serviços o sincronized
por exemplo:
private boolean lightOn() sincronized;
pois pode ser alterada pelos vários clientes logo estes métodos têm que ser sincronized

#callBacks Rmi
callback são notoificações assincronas (o observador observável), por exemplo:

as vistas são actualizadas quando existem alterações no modelo, esta operação é o callback

quando existe um alteração no modle é notificada (avisada) a vista, por exemplo:

o cliente tem a referêcnai remota do serviço
o cliente quer saber sempre quando existe uma alteração em um membro do serviço
o cliente é um observador e o serviço um observavel
ambos estão na mesma maquina virtual
o servidor vai ter uma lista, coleção de referecnias para aplicações (observador/clientes) que esão interessados em ser notificados, é uma lista de sockets
do lado do cliente existe uma thread que está sempre à escuta, que recebe a mensagem e atualiza a vista

mas agora usamos objectos remotos, a lista no servidor vai ter coisas que implementam interfaces remotas nos clientes
os clientes passam a ter uma interface remota que vai ser usada pelo servidor
é muito importante perceber: os clientes implementam uma interface remota

o cliente vai logo terminar?
no main do cliente encontra-se:
tenho uma referencia
obtive a temperatura
crio o serviço (NEW...)
registo o sensor (temperature change...)
mas a aplicação do cliente não termina porque foi criado um objecto do tipo unicast object, que tem uma thread que está à escuta no server socket e que é um serviço

#protocolo HTPP em java
no HTTP temos um pedido (get) e uma resposta, através de uma ligação TCP
na resposta surge sempre um código de 3 digitos, e um cabeçalho e um corpo (opcional)

#URi
URi ou URL,
o URi, é um identificador
o URL, é um locator
ambos querem dizer a mesma coisa, servem para identificar recursos

o que é que isto identifica?
RMi://services.isec.pt/servicosacademicos

RMi registry, que é uma base de dados com uma tabela que tem nome e caminho remoto (dos sturbs)
em que servicosacademicos é nome do serviço

na tabela surgem referências remotas, do tipo javaRMi remote

com o método lookup, obtenho o valor correspondente aos servicosacademicos
com o método nbing, serve para apagar
com o método bing ou rebind, serve para acrescentar (bind) ou para substituir (rebind)

#campos cabeçalho
expansivel - campos prédifinidos
pedido - usando o método get, recurso, protocolo, mudança de linha
get / widget / xpto / 3 / http/1.1 \n\n
repsosta - http/1.1 200 ok
em que 200 é o código

se for feito um post serve para actualziar recursos nos ervidor, ou submeter autenticação

#java.net.url
o que faz o openStramÎnputStream in = myURL.openStream();
faz uma ligação TCP no porto adequado no URL, envia para lá um get com o ficheiro não identificado por omissão

#URL connection
prmite aceder aos códigos de estado das mensagens de resposta

#alguns métodos de autenticação HTTP
fazer um pedido HTTP, reebo uma resposta por parte do servidor
básico: em cada pedido tenho o login e password que vai no cabeçalho, e no cabeçalho vem Authorization: basic usernamepassword
codificado em base 64, e é apenas uma forma de codificar

#token
o token depois de ser feita a autenticação é dado ao utilziador um token que só lhe pertence e esse token é unico, só do individuo
este é o token que permite ao utilziador fazer as suas operações depois do login

//----- perguntas de exame:
==1)
aparece este bocado de código:
PrintStream pout = new PrintStream(out);
pout.printl(new Java.util.date());
pout.flush();
1) o que faz este código?
o new java.util.date() é escrito no pout
2) concorda com a seguinte afirmação:
"este pedaço de código escreve no ecrã a hora atual seguida de uma mudança de linha"
Não, não é totalmente coreto, pode ser mas também pode não ser, porque o printStream está a receber um objeto do tipo outputStream e o outputstream não é só System.out, não é só fileOutPutStream, ele pode estar a escrever isto para um ficheiro, ou para o ecrã pode ser para um socket TCP, como ode ser um byteArray outpuStream (temos aqui uma questão de polimorfismo)

==2)
multicast(subclasse de DATAGRAMSOCKET)

em que:
multicast: quando queremos também receber as mensagens que são enviadas para o grupo
DATAGRAMSOCKET: se eu só pretendo enviar para um grupo de multicast é o envio de um DATAGRAMSOCKET UDP clássico, enviar para um determinado IP/porto

não existe nenhum servidor de gestão de grupos

==3)
as threads podem ser iniciadas em dois modos distintos:
modo por omissão, modo normal (modo utilizador) // bloqueante
modo através do daemon(serviço para correr em background) // não bloqueante

Uma aplicação em java termina quando já não existe qualquer, em que já não existem threads em modo utilizador activas.
As thraeds activas em modo de utilizador, mesmo chegando ao fim do main elas continuam activas, se forem as threads em modo daemon, chegada ao fim do main elas terminam

```
public class RunnableThreadDemo implements java.long.Runnable
{
public void run(){
SystemOut.println("....");
}
public static void main (String args[]){
System.Out.println("...");
Runnable run = new RunnableThreadDemo();
System.Out.println("...");
thread t1 = new Thread(run, "thread 1"); //run, é o nome da thread
System.Out.println("...");
thread t2 = new Thread (run, "trhead 2"); //run é o nome da thread
System.Out.println("...");
t1.start();
t2.start();
}
}
```

a diferença entre os dois modo de criar threads pode ser importante
assim o método II é melhor que o método I, já que posso criar várias threads baseadas no mesmo objecto runabble e elas vão partilhar os mesmos membros. posso assim ter várias threads concorrentes mas a trabalhar sobre os mesmos dados.
E outra vantagem é contornar uma das limitações do já de não haver herança múltipla

class ExtendThreadDemo extends java.long.thread // esta classe não pode estender outro tipo de objecto

==4)
threads, faz-se o new thread mas depois tem que se fazer o start,e só depois do start o new cria um objecto que respresenta uma thread, mas o start lança de forma efectiva

==5)
complete o código, em que temos:
for(i = 0; i< nThreads; i++){
threads[i] = new ParallelPi(i+1, nThreads, nIntervals);
threads[i].start();
}

se eu estou a criar a minha thread com new ParallelPi, quer dizer que ParallelPi extends thread, se fosse implements runnable eu teria que ter um new Thread e em argumento um new ParallelPi
se existe um new ParallelPi e depois faço um start então é porque é extends thread e não implements Runnable

==6)
porque é que temos que usar drivers diferentes para bases de dados diferentes? se tiver que usar SQL, porque é que tenho que usar um driver diferente se tiver que usar ORACLE?
porque o protocolo de comunicação é diferente

==7)
um proxy que tem os métodos, e eu localmente tenho um objecto que tem os mesmos métodos
RPC é o inicio da história para o paradigma dos sistemas distribuídos (miiddleware)

os passo do RPC
passo1: o servidor fica a correr e vai-se registar na porta 111
passo2: o cliente solicita o porto do servidor
passo3: recebe a porta
passo4: faz o pedido
passo5: recebe a resposta

==8)
o que é que um serviço e um proxy têm em comum?
a interface, os métodos que implementam

==9)
porque razão é que o java RMi nas interfaces remotas tem que ter os argumentos e os resultados serializados?
porque o sistema, invocamos o método remoto e o que acontece é que é enviado um objecto que representa um pedido para o servidor, e o RMi faz a serialização dos objetos logo o pedido vai ter a identificação do método, etc, e assim tem que ser tudo serializado

==10)
só os métodos que fazem parte da interface remota é que precisam do RemoteException?
assim os outros métodos do serviço que não fazem parte da interface remota já não vão dar

==11)
naming.rebind(registration, bs);
em que:
se estabelece uma ligação TCP
no porto 1099 (porto de escuta do RMi registry)
manda-lhe uma margem para registar como o nome RMi Light bulb a referencia ao objecto Bs

==12)
rmi://192.10.10.10/xpb
o que faz quando se faz naming rebind

isto não estabelece uma ligação com um serviço
naming.lookup(rmi://192.10.10.10/xpb);

o naming.lookup no cliente para ele obter a referência para um serviço com este nome

quando se faz o lookup o que é feito é uma ligação TCP com qualquer coisa no IP no porto 1099 que é do registry e quero obter uma referência para um objecto com o nome xpb, esta referência tem interface e tem IP e porto

==13)
qual a diferença entre o método Bind e Rebind? (Bind e Rebind, tem a ver com tabelas no registry)
Com o rebind se existir é substituído senão existir ria
com o bind se não existir cria e se já existir cria uma excepção

==14)
no cliente o que é que faz o ON em bullbService.on(); sabendo que existiu
SErviceRmi bullBservice = (serviçoRmi) remoteSErvice

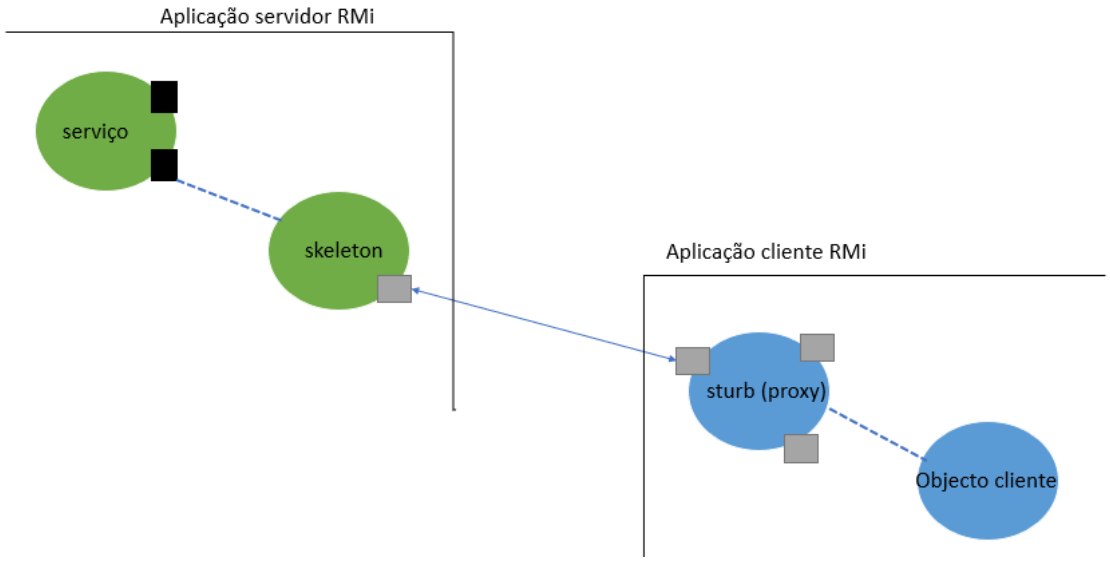
o cliente não faz um NEW local do objecto o que ele faz é um naming.lookup
no cliente depois vou ter um objecto que implementa a interface do serviço (on, off, ...)
into é em sturb, e vai ser enviado ao fazer on, envia serializado a operação que tem que ser executada e que é o on, e o serviço recebe o pedido, executa e devolve o resutlado

==15)
qual é o aspecto comum entre proxy/sturb e o serviço?
é a interface, tem um conjunto de métodos coincidentes isto é, no serviço existe um on() e no sturb também existe um on() que na realidade envia uma mensagem para o outro lado para ligar

==16)
o que significa?
endPoint(10.20,4,128.77:53561);
10.20,4,128.77 é o IP do serviço
53561 é o porto automático, criado pelos server socket
quando o cliente faz o lookup do registry obtêm uma eferencia remota, o que se indica no lookup é o ip do registry e o porto (do registry) mas aqui é isto que eu obtenho

==17)
o que é que é comum entre o serviço (servidor) e a referência remota (Sturb, no cliente)?

é a interface implementada
o proxy não faz aquilo que é suposto o método fazerem, o que ele faz é pegar nos argumentos e envia para o servidor (Skeletton) que no servidor vai ser executado, que depois serializa e envia a resposta e o Sturb devolve a resposta ao cliente



quando o serviço é lançado (no servidor) ele regista-se no RMi registry (que é tipo um DNS) onde fica registado o IP e o porto onde pode ser encontrado o serviço

==18)
o que acontece inernamente (por baixo) no cliente RMi quando ele faz so dos métodos, por exemplo em:
bulbSErvice.isOn()
é feito um pedido de envio
aguarda resposta
o pedido é serialziado
e é devolvida a resposta

==19)
situação de dois clientes acederem ao mesmo objecto ao mesmo tempo?
o cliente através do lookup contacta o registry (TCP, 1099) e envia o pedido para o que quer
o cliente depois recebe a referência remota do objecto, que tem a interface pretendida (métodos...) e também tem o ip e porto do serviço (serviço APi)
no servidor depois de contactado cria uma thread que vai cuidar desse cliente
e se existir outro cliente vai também ser o mesmo procedimento, e no serviço (servidor) é criada outra thread
assim eu tenho que verificar se é critico os acessos concorrentes, dois clientes têm acesso a um determinado serviço, estão a aceder ao mesmos erviço e mesmos recursos, então eu terei que colocar nas classes concretas/serviços o sincronized
por exemplo:
private boolean lightOn() sincronized;
pois pode ser alterada pelos vários clientes logo estes métodos têm que ser sincronized

==20)
callbacks RMi
o cliente vai logo terminar?
no main do cliente encontra-se:

tenho uma referencia
obtive a temperatura
crio o serviço (NEW...)
registo o sensor (temperature change...)
mas a aplicação do cliente não termina porque foi criado um objecto do tipo unicast object, que tem uma thread que está à escuta no server socket e que é um serviço

==21)
o que é que isto identifica?
RMi://services.isec.pt/servicosacademicos

RMi registry, que é uma base de dados com uma tabela que tem nome e caminho remoto (dos sturbs)
em que servicosacademicos é nome do serviço

na tabela surgem referências remotas, do tipo javaRMi remote

com o método lookup, obtenho o valor correspondente aos servicosacademicos
com o método nbing, serve para apagar
com o método bing ou rebind, serve para acrescentar (bind) ou para substituir (rebind)

==22)
o que faz o openStramÎnputStream in = myURL.openStream();
faz uma ligação TCP no porto adequado no URL, envia para lá um get com o ficheiro não identificado por omissão