PROGRAMAÇÃO DISTRIBUÍDA - 2016/17

Exame Teórico – 20 de Janeiro de 2017

Sem consulta / Duração: 60 minutos / Todas as perguntas possuem a mesma cotação / As respostas devem ser objectivas e sintéticas

---

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: É incorreto afirmar que o método ilustrado acima apenas permite copiar o conteúdo de um ficheiro para outro. O que o método faz é copiar (byte a byte) o conteúdo de uma stream de de input para uma stream de output. Como qualquer pessoa com experiência com Java deve saber, streams (tanto de input como de output) são incrívelmente versáteis e existem inúmeras especializações destas duas. Este método como aceita InputStreams e OutputStreams ou qualquer especialização das duas, aceita FileInputStreams e FileOutputStreams, o que significa que método permite copiar o conteúdo de um ficheiro para outro. Mas não é só isso que o método permite, o método também permite (por exemplo) colocar o input que o utilizador colocar no stdin (System.in) através do terminal num ficheiro (FileOutputStream) ou no ecrã (terminal) através do stdout (System.out). O que o método faz é contextual e depende dos seus argumentos (especificamente do tipo de streams que lhe são passados e dos recursos a que estas streams estão associados).

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: Serviços de nomeação são meramente aplicações que permitem "nomear" outros serviços (um exemplo é o rmiregistry que permite definir um nome e URI para serviços do tipo Remote em Java), tornando o acesso a estes serviços na rede fácil e transparente para o utilizador. Para facilidade de explicação vou me focar em RMI na minha resposta mas o processo de nomeação e de acesso de serviços é semelhante nas várias especializações de RPC (RMI/CORBA/outros). O que acontece no arranque de uma aplicação Java que utilize RMI é que o serviço de nomeação do RMI é iniciado (rmiregistry.exe) e assim que um instância de uma classe que implemente a interface Remote for criado, este é nomeado no serviço de nomeação (o que isto significa é que o serviço de nomeação automaticamente escolhe um nome para possibilitar o acesso ao objeto remoto na rede) e na aplicação é aberta uma socket do tipo TCP/IP (em Java, é Socket) com porto definido pelo utilizador (se assim for o caso, através de argumentos do compilador), por

omissão ou automático. O serviço de nomeação passa a "apontar" o nome criado anteriormente para este socket de modo que um cliente que queira aceder ao objeto remoto automaticamente faça pedidos a este socket. A invocação dos métodos no objeto remoto é feita através do envio de mensagens do tipo pedido para o socket designado (através de um socket aberto para o propósito) e a espera por mensagens do tipo resposta (para o socket acabado de abrir e do socket no servidor) que contêm pelo menos os valores devolvidos (se houverem) pelos métodos invocados. Para atender aos pedidos de invocação de métodos no objeto remoto no servidor, é automaticamente criada uma thread (num objeto Skeleton) com este mesmo propósito.

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 objetivo do método invocado no excerto de código acima é procurar e aceder a serviço RMI no Registry da rede representada pelo ip 192.168.1.1 e porto 1099. Rmi é uma palavra-chave utilizada para informar o método de que o tipo de serviço pelo qual estamos à procura é do tipo RMI e como tal encontra-se no Registry RMI e finalmente o RMILightBulb é o nome de um serviço (o serviço ao qual pretendemos aceder no cliente).

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: O método openStream tem a seguinte sequência de ações:

1. Estabelece conexão TCP/IP com socket alojado no porto 80 do servidor Web
2. ?

   m. Envia pedido GET sobre conexão.
   n. Obtém payload (dados – espera-se que seja HTML) alojados no URL.
   o. Devolve InputStream com o conteúdo do payload obtido.

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 t*hread* 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();
            ...
      }
}
```

```
psvm {
 Thread t1 = …;
 t1.join();
 Thread t2 = …;
}
public synchronized metodo1() {
  …
}
```