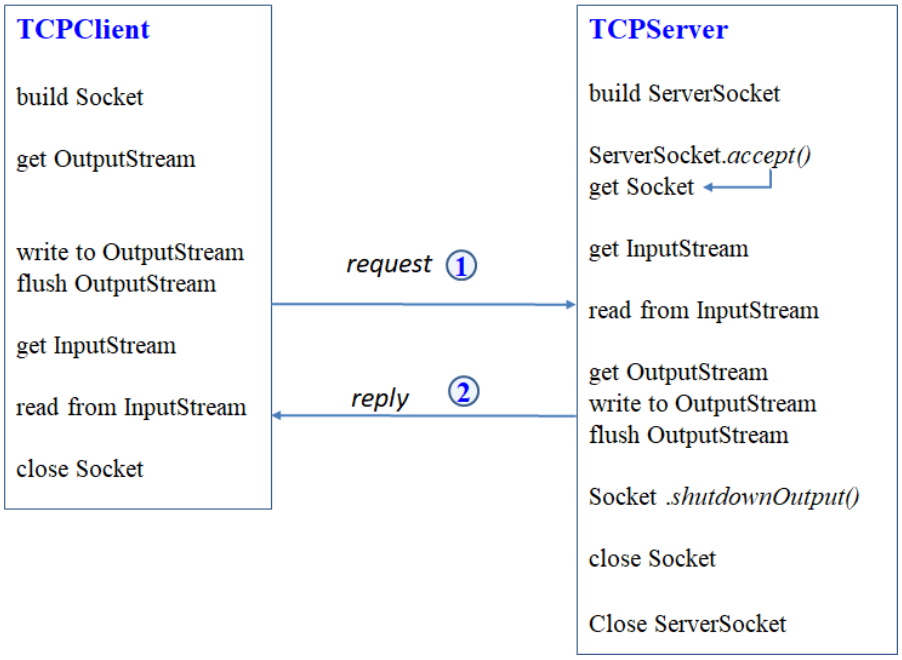
Exercício sobre TCP

1. Arquitectura Geral



2. Objectivos para a Aula 6

O objetivo da 6ª aula TP é permitir a vossa aquisição de conhecimentos no contexto da comunicação TCP (empregando a API Java para esse efeito).

O guião do exercício a ser implementado na aula está disponível em https://web.fe.up.pt/~pfs/aulas/sd2020/labs/l04/tcp_l04.html
(trata-se de mais uma versão do lab1, desta vez empregando TCP na comunicação entre cliente e servido)

No contexto do *assigment 1* a API Java para TCP poderá ser empregue:

- na comunicação entre o *TestClient* e os *Peers*, caso não consigam/queiram desenvolver esta interface em RMI;
- no *enhancement* to "*Chunk Restore Protocol*" (secção 3.3 do guião do *assigment 1*);

3. Notas Sobre a API Java para TCP

As notas abaixo estão de acordo com as transparências apresentadas pelo Professor Pedro Souto na aula teórica, e presentes primeiro "link" no final do guião.

1) A API de Java distingue entre:

- sockets usados para aceitar conexões – deve usar-se uma instância da classe *java.net.ServerSocket* para ficar permanentemente á escuta de ligações TCP numa determinada porta;
- sockets usados para transferir dados – são usadas instâncias da classe *java.net.Socket* para a efectiva leitura e envio de informação para o canal TCP

2) Embora a classe *Socket* seja usada para a troca de informação, esta não oferece métodos para o fazer diretamente. Fornece antes os métodos *getInputStream()* e *getOutputStream()*. Estes retornam referências para objetos (que implementam as classes abstratas *InputStream* e *OutputStream*), os quais permitem a efectiva troca de informação.

3) Os objectos retornados pelos métodos acima mencionados são bastante básicos e permitem apenas transferir *arrays* de bytes.
Pode-se, no entanto, usar estes *streams* básicos para construir *streams* com mais funcionalidades (classes da *package java.io.*). Podem assim criar-se *streams* capazes de efectuar a escrita/leitura de:
- texto formatado (*PrintWriter*/*BufferedReader*),
- tipos primitivos de Java (*DataOutputStream*/*DataInputStream*)
- objetos Java serializáveis (*ObjectOutputStream*/*ObjectInputStream*)

Notem que é importante que os *streams* (instâncias de classes da package *java.io.*) empregues para enviar mensagens numa das extremidades, e recebê-las na outra, sejam os correspondentes (os pares acima indicados). Assim, é pouco útil, por exemplo, usar um *ObjectOutputStream* para enviar dados, se na outra extremidade se usa um *BufferedReader*.

No exercício da aula desta semana fará sentido usarem o par *PrintWriter*/*BufferedReader*. O emprego destas classes permite estruturar as mensagens como linhas e troca-las como tal, usando os métodos *println()* e *readLine()* daquelas classes.

No caso *assignment*, o emprego de TCP será para a transferência de *chunks* (secção 3.3) e assim o uso das classes *PrintWriter* e *BufferedReader* não é adequado. Deverá antes ser suficiente o uso dos *streams* básicos, i.e. *OutputStream* e *InputStream*.

4) Algumas das classes que implementam *streams*, (e.g. *PrintWriter*), fazem *buffer* dos dados antes de os enviarem. Assim podem não enviar os dados que são "escritos" neles imediatamente para o socket, o que poderá levar a comportamentos inesperados. É por isto conveniente que se faça *flush* do *stream* onde se está a escrever (invocando o método *flush()*).
No caso da classe *PrintWriter* existe um construtor que permite configurá-lo de forma a que este faça *flush* automaticamente sempre que nele é escrita uma linha.

5) Para garantir que ao fechar uma conexão (método *close()* das classes *Socket*) não se perdem dados (devido, por exemplo, à questão abordada no ponto anterior), antes desse fecho efectivo deverá chamar-se o método *shutdownOutput()*, e só depois então o metido *close()*.