→ Sockets em Java (continuação)

1 – Construa uma aplicação cliente – servidor, com comunicação por Sockets TCP, que permita dois utilizadores manterem uma conversa em que as mensagens devem ser linhas de texto (Strings) introduzidas através dos respetivos teclados. Na comunicação use ObjectStreams.
A conversa deverá seguir o seguinte protocolo:
. O utilizador do processo cliente é o primeiro a "falar";
. O utilizador do processo servidor decide quando termina a conversa enviando uma linha de texto com a palavra **fim**.
Quando o utilizador do processo servidor termina a conversa, o cliente termina, mas o processo servidor deverá continuar a execução esperando que um novo cliente estabeleça ligação.
- Teste o programa com duas máquinas.

➢ Implementação da invocação de métodos de um objeto remoto através de sockets

2 - Construa uma classe Aluno em que cada aluno tem um número (correspondente ao seu número de aluno) um nome, o curso que frequenta e um contacto (telefone ou e-mail). - Defina as operações básicas de consulta e modificação (getters e setters) para cada um dos atributos e os métodos toString e equals (*use a opção "insert code" do NetBeans*).

- Pretende-se uma aplicação, cliente - servidor, em que o processo servidor armazena de forma persistente (num ficheiro) os dados dos alunos que se quiserem registar, e o processo cliente permitirá o seguinte conjunto de operações:

i) Registar aluno (operação em que envia os dados de um aluno ao servidor e recebe como resposta o número total de alunos registados até ao momento). Deverá ser verificado se o aluno ainda não está registado.

ii) Consultar quais os alunos registados (Obter uma lista com os dados de todos os alunos já registados.

iii) Consultar o número de acessos ao servidor ocorridos até ao momento.

iv) Dado um nome de aluno devolver o seu número e o seu contacto, ou caso haja mais de um aluno com o mesmo nome devolver os vários números e contactos.

A classe que implementa o processo Servidor possui os atributos,

- alunosRegistados, que será um objecto do tipo java.util.Vector<Aluno>, e irá conter os dados de todos os alunos registados;
- numeroAcessos – valor inteiro que representa o número de acessos ao servidor;

e os métodos que implementam cada uma das operações a que o processo cliente pode aceder.

a) Implemente a aplicação usando Sockets TCP, e as Streams (ObjectInputStream e ObjectOutputStream). Considere dois modelos de comunicação:

1 - Uma conexão entre os processos cliente e servidor para a invocação de cada método.

2 - Uma conexão para cada cliente, isto é, um cliente deve poder executar várias operações no servidor através da mesma conexão entre os dois processos.

- Para qualquer das opões anteriores o servidor ao terminar a ligação com um cliente, deverá poder aceitar a ligação a outro cliente.

b)Teste com várias máquinas e vários clientes a aceder em simultâneo ao servidor. O que acontece?

[Para treinar]

3 - Pretende-se construir um servidor que a cada ligação de um cliente através de um Socket responda com uma saudação aleatória. O servidor deverá possuir um array de Strings com diferentes saudações, por exemplo {"Bom dia", "Bem disposto?", "Oi", “Salvé”, ... }.

Sempre que um processo cliente envia uma mensagem ao servidor (independentemente do conteúdo dessa mensagem) o servidor responderá com uma das saudações do array escolhida aleatoriamente. O servidor deve possuir um ciclo infinito onde aceita a ligação com o cliente, recebe a mensagem daquele e envia a sua saudação aleatória.

- Construa as classes cliente e servidor. Na comunicação use ObjectStreams.

→ Threads em Java

Num programa em JAVA é possível definir diferentes sequências de execução independente: **Threads**.

Uma Thread é similar a um processo no sentido em que corresponde a um conjunto de instruções que pode ser escalonado para execução num dado processador. No entanto, as Threads definidas num dado programa partilham o mesmo espaço de endereçamento que o processo principal que lhes deu origem.

⇨ Criar uma Thread

Existem duas formas de criar uma Thread em JAVA. A primeira consiste em criar uma **subclasse da classe Thread** e usar o método start() definido em Thread para iniciar a execução. O método start() invoca por sua vez um método run() a definir pelo utilizador, e que deverá conter a sequência de execução da Thread instanciada.

4 - Implemente e teste o seguinte exemplo:

```
public class MyThread extends Thread {
  public void run() {
   System.out.println("Hello there, from  " + getName() );
  }
}
public class Teste {
public static void main (String[] str){
  MyThread Ta, Tb;
  Ta = new MyThread();
  Tb = new MyThread();
  Ta.start();
  Tb.start();
}}
```

- Este processo de criar uma Thread não pode ser usado se pretendermos que a classe a criar seja subclasse de alguma outra, por exemplo no caso de querermos construir uma Applet.
A segunda forma de criar uma Thread é usar a interface **Runnable**. [Recordando …] Em termos simples uma "interface" é a definição de um conjunto de métodos que a classe ou classes que implementam essa interface terão que implementar:

A interface Runnable:

```
package java.lang
public interface Runnable{
  public abstract void run();
}
```

No exemplo abaixo, a classe MyThread2 implementa a interface Runnable, implementado o método run() que é o único método definido na interface. A diferença entre esta classe e a classe MyThread definida anteriormente é que a classe MyThread2 não herda os métodos da classe Thread. Para agora criarmos uma nova Thread temos de passar ao construtor dessa Thread a referência de uma classe que implemente a interface Runnable.

5 - Implemente e estude o exemplo que se segue.

```
public class MyThread2 implements Runnable{
 public void run(){
  System.out.println("Hi there, from " + Thread.currentThread().getName() );
 } }
public class Teste {
 public static void main  (String[] str){
  MyThread2 T = new MyThread2();
  Thread Ta, Tb;
  Ta = new Thread( T );
  Tb = new Thread( T );
  Ta.start();
  Tb.start();
 } }
```

6 – Pretende-se construir um programa que permita adicionar arrays de grande dimensão. Para isso, vamos dividir os arrays a somar em várias porções menores e somar cada porção numa thread independente.
- Construir o código de uma thread, ThreadSoma, que receba como parâmetros as "referências" de 3 arrays, A, B e C e 2 valores inteiros, p e u. A thread deverá fazer a soma dos arrays, A e B, desde a posição p até à posição u-1, colocando o resultado em C.

$$C[i] = A[i] + B[i] \quad \text{com } i = p, p+1, p+2, \ldots u-1$$

a) Construa um programa onde deverá criar dois arrays de inteiros com valores aleatórios. Os arrays, A e B, devem ter a mesma dimensão. Para somar os dois arrays, divida-os em dois e cada metade deverá ser somada por uma instância da classe ThreadSoma que criou na alínea anterior.

b) Queremos agora que no programa anterior, o programa principal calcule a soma de todos os valores do array resultado, C. Essa soma só deve ser feita após ambas as instâncias de ThreadSoma terminarem a execução. Explore os métodos da classe Thread para ver como pode fazer com que uma Thread espere que outra termine a execução.