→ Sockets TCP em Java

1 – Estude e implemente a classe abaixo.

```
import java.net.*;
import java.io.*;

public class Cliente {
  public Cliente(){
  try {
    Socket sc = new Socket("127.0.0.1", 2222);

    PrintWriter pr =new PrintWriter ( sc.getOutputStream() , true);
    InputStream is = sc.getInputStream();
    BufferedReader br = new BufferedReader (new InputStreamReader(is));

    System.out.println(br.readLine());
    pr.println(" Olá, eu sou o cliente");

    System.out.println(br.readLine());
    pr.println("Cliente: Continuo por aqui");

    pr.close();
    is.close();
    sc.close();
  }
  catch (IOException e){
   System.out.println( e.getMessage());
  }
  }
  public static void main (String args[]){
    Cliente c=new Cliente();
  }
}
```

a) O que acontece se tentar executar este programa?

b) Implemente uma classe, Servidor, que comunique com o Cliente do exercício anterior.

(Nota: Consulte os apontamentos da aula teórica)

c) – Depois de executar o cliente e o servidor anteriores na mesma máquina, teste o seu processo Cliente com o processo Servidor de um dos seus colegas e vice-versa.

d) Modifique o Cliente e o Servidor dos exercícios 1 e 2 de forma a ambos efetuarem primeiro as operações de leitura no socket e depois as operações de escrita. O que acontece?

e) Modifique agora o Cliente e o Servidor dos exercícios 1 e 2 de forma a ambos efetuarem primeiro as operações de escrita no socket e depois as operações de leitura. O que acontece?

2 – Modifique os exercícios 1 e 2 de forma a usar Streams de objectos em vez de Streams de caracteres.

3 – Modifique o Servidor do exercício **2** de maneira a que este permaneça em execução à espera da ligação de novos clientes. Teste o programa com um colega.

4 – Pretende-se construir uma aplicação cliente/servidor cuja comunicação é feita através de Sockets. O processo Servidor recebe um valor do tipo char que identifica o tipo de cliente que está a comunicar com ele.
Clientes do tipo 'A', depois de enviarem ao servidor o seu tipo, enviam um valor inteiro e recebem como resposta o quadrado desse valor.
Clientes do tipo 'B', depois de enviarem ao servidor o seu tipo, enviam um valor do tipo double e recebem como resposta a raiz quadrada desse valor.

- Construa as classes Servidor, Cliente_A e Cliente_B usando ObjectStreams. Os clientes devem poder fazer vários pedidos ao servidor até decidirem terminar. Escolha um critério de paragem.

Notas:
Para gerar os valores a enviar pelos clientes use o método "double Math.random()", para calcular a raiz quadrada use o método "double Math.sqrt(double)".

→ **Sockets UDP em Java**

5 - Estude e implemente as duas classes abaixo, UDPClient e UDPServer.

```
import java.net.*;
import java.io.*;
public class UDPClient{
  public static String readString (){
    BufferedReader canal;
    canal = new BufferedReader ( new InputStreamReader (System.in));
    try {
      return canal.readLine();
   }
    catch (IOException ex) {
      return null;
   }
 }
 public static void main(String args[]){
      String s;
      System.out.print("Qual o servidor? ");
      String host = readString();
      DatagramSocket aSocket = null;
       try {
            aSocket = new DatagramSocket();
             while(true){
               System.out.print("Mensagem a enviar = ");
               s = readString();
               byte [] m = s.getBytes();
               InetAddress aHost = InetAddress.getByName(host);
               int serverPort = 2222;
               DatagramPacket request = new DatagramPacket(m, m.length, aHost,serverPort);
               aSocket.send(request);
               byte[] buffer = new byte[100];
               DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
               aSocket.receive(reply);
               System.out.println("Recebeu: " + new String(reply.getData()));
            } // while
        }catch (SocketException e){System.out.println("Socket: " + e.getMessage());
        }catch (IOException e){System.out.println("IO: " + e.getMessage());
      }finally {if(aSocket != null) aSocket.close();}
  }
}
```

```
import java.net.*;
import java.io.*;
public class UDPServer{
      public static void main(String args[]){
      DatagramSocket aSocket = null;
      String s;
         try{
            aSocket = new DatagramSocket(2222);
            System.out.println("Socket Datagram à escuta no porto 2222");
            while(true){
                byte[] buffer = new byte[100];
                DatagramPacket request = new DatagramPacket(buffer, buffer.length);
                aSocket.receive(request);
                s=new String(request.getData());
                System.out.println("Server Recebeu: " + s);
                DatagramPacket reply = new DatagramPacket(request.getData(),
                        request.getLength(), request.getAddress(), request.getPort());
                aSocket.send(reply);
            }
          }catch (SocketException e){System.out.println("Socket: " + e.getMessage());
         }catch (IOException e) {System.out.println("IO: " + e.getMessage());
      }finally {if(aSocket != null) aSocket.close();}
   }
}
```

a) Experimente eliminar o processo cliente, o que acontece?

b) E se eliminar o processo servidor?

c) Investigue quais as alterações que deve colocar no cliente para este gerar um timeout quando não recebe uma resposta do servidor no espaço de 15 segundos?

d) Altere a classe cliente de forma a calcular o RTT (Round-Trip-Time) da comunicação, isto é, o tempo total que uma mensagem demora a ir de um processo A a um processo B e regressar a A.

Nota: Para medir o tempo pode usar o seguinte método:
long t=System.currentTimeMillis(); // Devolve o tempo actual em mili-segundos.