Mecanismos de Comunicação

Sockets em java

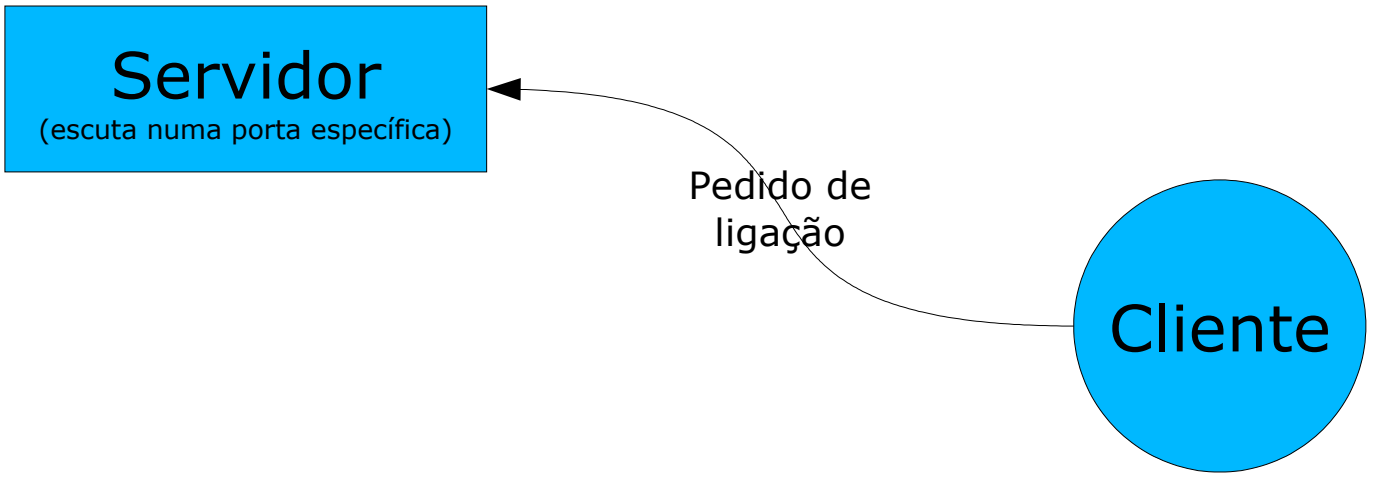
(http://java.sun.com/docs/books/tutorial/networking/sockets/)

Sockets – o que é?

- Um socket é um mecanismo de comunicação (dois sentidos) entre dois programas a funcionar (normalmente) numa rede
- O pacote java.net contém duas classes – *Socket* e *ServerSocket* – que implementam, respectivamente, o cliente e o servidor numa ligação

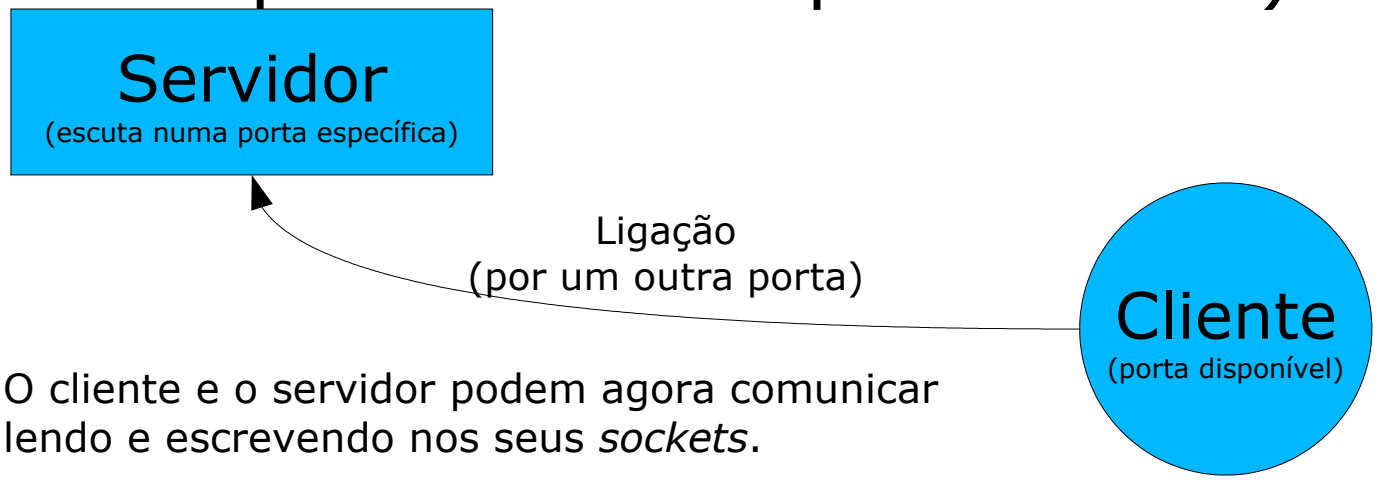
Sockets – o que é?

- Uma aplicação servidor é executada numa determinada máquina e tem um *socket* ligado a uma porta específica dessa máquina
- O servidor espera que um cliente faça um pedido de ligação através desse *socket*



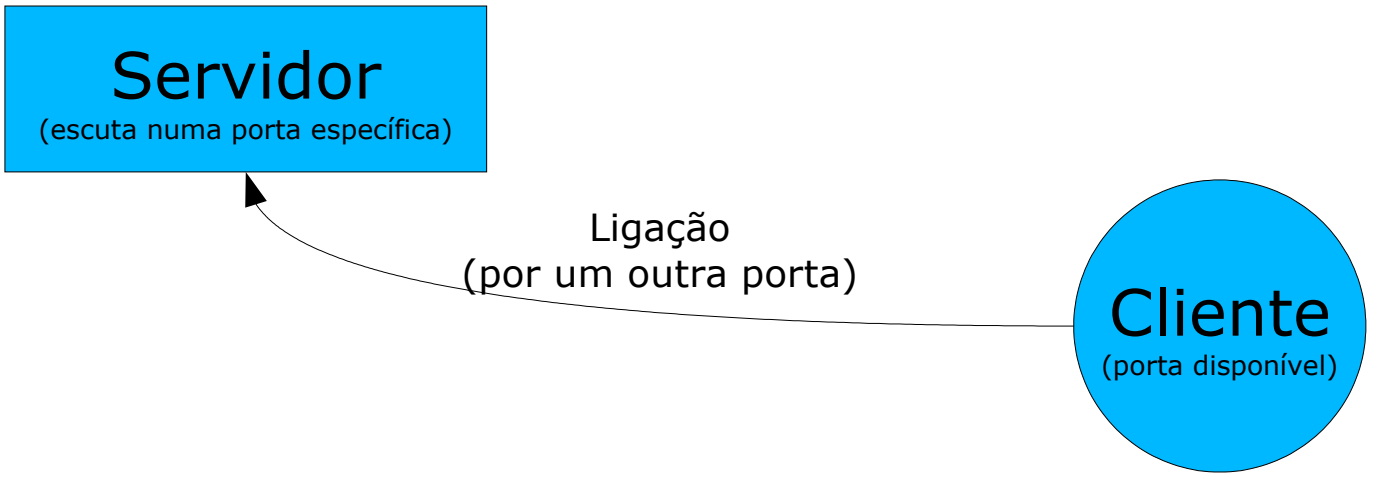
Sockets – o que é?

- O servidor ao aceitar a ligação cria um novo *socket* para uma porta diferente (e assim permite novas ligações)
- No lado do cliente um socket é criado e é usado para comunicar com o servidor (numa porta disponível na máquina cliente).



Sockets – o que é?

- O pacote java.net contém a classe Socket que esconde todos os detalhes particulares a cada sistema.
- Ainda tem uma segunda classe, SocketServer, que implementa a parte de servidor.



Sockets – Exemplo cliente

```
import java.io.*;
import java.net.*;
public class EchoClient {
    public static void main(String[] args) throws IOException {
        Socket echoSocket = null;
        PrintWriter out = null;
        BufferedReader in = null;
        try {
            echoSocket = new Socket("max.uma.pt", 7);
            out = new PrintWriter(echoSocket.getOutputStream(), true);
            in = new BufferedReader(new InputStreamReader(
                                    echoSocket.getInputStream()));
        } catch (UnknownHostException e) {
            System.err.println("Não encontro o host: Max.");
            System.exit(1);
        } catch (IOException e) {
            System.err.println("Não foi possível a ligação a: Max.");
            System.exit(1);
        }
        BufferedReader stdIn = new BufferedReader(
                                new InputStreamReader(System.in));
        String userInput;
        while ((userInput = stdIn.readLine()) != null) {
            out.println(userInput);
            System.out.println("echo: " + in.readLine());
        }
        out.close();
        in.close();
        stdIn.close();
        echoSocket.close();
    }
}
```

- Ligação à porta 7 (Echo Server)
- A negrito:
  - a definição do *socket* e como pode ser lido e escrito
  - Ciclo *while* para enviar e receber mensagens

Passos para o criar um cliente:

1. Abrir um socket;
2. Criar os *streams* de leitura e escrita
3. Ler e escrever mediante os objectivos do sistema
4. Fechar os *streams*.
5. Fechar o *socket*.

Sockets – Exemplo Servidor

```
import java.net.*;
import java.io.*;
public class KnockKnockServer {
    public static void main(String[] args) throws IOException {
        ServerSocket serverSocket = null;
        try {
            serverSocket = new ServerSocket(4444);
        } catch (IOException e) {
            System.err.println("Could not listen on port: 4444.");
            System.exit(1);
        }
        Socket clientSocket = null;
        try {
            clientSocket = serverSocket.accept();
        } catch (IOException e) {
            System.err.println("Accept failed.");
            System.exit(1);
        }
        PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
        BufferedReader in = new BufferedReader(
                                new InputStreamReader(
                                clientSocket.getInputStream()));
        String inputLine, outputLine;
        KnockKnockProtocol kkp = new KnockKnockProtocol();
        outputLine = kkp.processInput(null);
        out.println(outputLine);
        while ((inputLine = in.readLine()) != null) {
             outputLine = kkp.processInput(inputLine);
             out.println(outputLine);
             if (outputLine.equals("Bye."))
                break;
        }
        out.close();
        in.close();
        clientSocket.close();
        serverSocket.close();
    }
}
```

- Classe servidor
- Espera o pedido de ligação do cliente
- O resto é semelhante ao cliente

Sockets – Exercício

- Copie os ficheiros do exemplo *Knock Knock Server* e adapte de forma que o cliente execute na máquina do laboratório e o servidor no max.uma.pt

  Nota: este servidor aceita múltiplas ligações

Página:

http://java.sun.com/docs/books/tutorial/networking/sockets/clientServer.html

Sockets – Stream *Vs* Datagrama

- *Stream* (TCP): existe uma ligação entre cliente e servidor (iniciada, mantida e terminada)
  - Vantagem: fiável
  - Desvantagem: velocidade
- Datagrama (UDP): mensagem independente cuja entrega, tempo e conteúdo não são garantidos
  - Vantagem: rapidez
  - Desvantagem: não é fiável

Sockets – Datagrama

- O Java tem as classes *DatagramPacket* e *DatagramSocket* no pacote java.net que implementa a comunicação de Datagramas sobre o UDP.

Para mais informação ver:

http://java.sun.com/docs/books/tutorial/networking/datagrams/