Nota prévia:

- os comandos apresentados executam-se na pasta base do projeto, salvo indicação contrária em algum caso especial
- alguns caminhos/argumentos nesses comandos podem ser diferentes caso importe o projeto para um IDE (mas funcionam com pequeno ajuste na classpath)

Comunicação entre Processos através da Rede

Em Java, como em C e noutras linguagens de programação, é possível recorrer sockets para transmitir dados pela rede. As primitivas genéricas associadas aos sockets são:

```
- connect
- send data
- receive data
- close connection
- bind to a port
- listen for incomming data
- accept
```

Há um conjunto de classes que facilitam o manuseamento de sockets e respetivas primitivas.
Alguns métodos construtores e métodos de instância podem mesmo efetuar mais que uma destas primitivas.

Ver a API de: `java.net.Socket`

- link para java API Doc

---

exercício

Pela análise da documentação, acima, descubra várias sequências de código em Java para estabelecer uma ligação com o protocolo TCP (no sentido de incluir a funcionalidade desde a criação do socket até ao `connect` genérico).

Operações frequentes na programação com sockets:

- obter o endereço IP associado a um nome:

```
InetAddress inet = InetAddress.getByName("alunos.di.uevora.pt");
System.out.println ("IP : " + inet.getHostAddress());
```

- obter o nome (hostname principal) associado a um endereço IP

```
InetAddress inet = InetAddress.getByName("193.137.216.82");
System.out.println ("Host: " + inet.getHostName());
```

- obter o endereço IP do cliente que se ligou via TCP

```
Socket mySock = myServerSocket.accept();
System.out.println ("Connection from : " +
   mySock.getInetAddress().getHostAddress() + ':' + mySock.getPort());
```

e todas as opções de configuração do socket (no C: SO_TIMEOUT, SO_LINGER, TCP_NODELAY, SO_RCVBUF) são possíveis através de métodos que poderá encontrar na API.

Numa aplicação que usa ligação à rede é necessário efetuar um correto **tratamento de exceções**, nomeadamente:

```
ConnectException
NoRouteToHostException
```

---

exercício (Início do CLIENTE TCP)

Obtenha o **código inicial** para esta aula (escolha uma das opções):
- formato projeto: netbeans (alternativas: eclipse, ou só ficheiros e make)

Focando-se apenas no código do cliente, altere um pouco a aplicação cliente para que:

- **receba** (e **verifique** a existência de) um terceiro argumento, que será a mensagem a enviar ao servidor.
- essa mensagem deve ser mostrada em stdout, já no método `go()`.

Tem de editar apenas o ficheiro EchoClient.java.
Os **comandos** devem ser executados **SEMPRE** na pasta base do trabalho (sd02sockets).
Exemplo para executar o cliente (na pasta base do projeto):

# no código com makefile:
$ java -classpath classes sd.EchoClient alunos.di.uevora.pt 8080 "*uma mensagem que vai ao servidor*"

# no projeto **Netbeans**: (a diferença será na classpath, build/classes)
$ java -classpath build/classes sd.EchoClient alunos.di.uevora.pt 8080 "*uma mensagem que vai ao servidor*"

# Eclipse:
$ java -classpath bin sd.EchoClient alunos.di.uevora.pt 8080 "*uma mensagem que vai ao servidor*"

# IntelliJ IDEA
$ java -classpath out/production/PROJECT sd.EchoClient alunos.di.uevora.pt 8080 "*uma mensagem que vai ao servidor*"

Teste a execução da aplicação para as **2 modificações** pedidas (com e sem 3º argumento, e imprimir no método go()).
(ainda não existe a ligação à rede)

---

exercício (cliente acede ao servidor remoto)
Complete a aplicação **cliente** para executar o serviço "echo" com o protocolo TCP.
Estude a API para:

```
java.net.Socket
java.io.InputStream
java.io.OutputStream
```

O cliente envia uma mensagem de texto ao servidor. Este devolve ao cliente a mesma String de volta.
O cliente deve mostrar a resposta, imprimindo a String recebida em stdout.
Para testar a aplicação, direcione o seu cliente para um servidor já em funcionamento na máquina alunos.di.uevora.pt (porto 8080)

```
$ java -classpath build/classes sd.EchoClient alunos.di.uevora.pt 8080 "uma mensagem que vai ao servidor"
```

o resultado visível na consola deve ser:

```
Hi_[uma mensagem que vai ao servidor]_Bye
```

```
(o prefixo e sufixo foram inseridos pela aplicação servidor)
```

---

exercício (servidor TCP)
Complete a aplicação servidor do serviço "echo", com o protocolo TCP.
a) Estude a API para:
`java.net.ServerSocket`

b) O servidor recebe o texto do cliente e devolve-lhe essa mesma String, eventualmente alterada com sufixo e prefixo, como indicado acima...

c) Coloque o servidor em funcionamento, validando a existência de um argumento com o porto em que escuta por ligações.

d) Teste o cliente anterior com esta nova aplicação... ajuste o argumento com o hostname onde o servidor está em execução e o porto que aceita ligações.

e) Teste o seu cliente com servidores implementados por colegas... acedendo ao respetivo endereço (**apenas** na sala ou se existisse conetividade para os portos que usamos!).

---

exercício
Faça um pequeno ajuste no servidor:
- recorra à classe `java.io.BufferedReader` para ler a String recebida, com o respectivo método `readLine`().
Veja aqui um exemplo de inicialização de um objecto deste tipo:

```
DataOutputStream sout= new DataOutputStream(data.getOutputStream());
BufferedReader breader= new BufferedReader(new InputStreamReader(data.getInputStream()));
```

---

exercício (veículos 01)
Considere um serviço de informação sobre veículos.
O servidor armazena dados sobre veículos na sede do IMTT. A aplicação cliente é executada em cada balcão regional daquele instituto, para obter informação sobre os veículos. Há dois tipos de pedido:

- **consulta**: dada uma matrícula espera-se o nome do proprietário
  - cliente executado com **3** argumentos
- **registo**: o cliente fornece ao servidor um par (matrícula,nome) para registo. Neste exercício, o registo pode ser feito numa estrutura de dados em memória.
  - cliente executado com **4** argumentos

Inspire-se no serviço echo para implementar aplicações cliente e servidor para este serviço de informação sobre veículos.
Pretende-se uma solução de simples, semelhante ao que faria na linguagem C.
Assuma que os dados ficam apenas em memória, no servidor, numa HashMap.
Precisa definir um protocolo de comunicação entre as aplicações, para representar o tipo de pedido e respetivos argumentos...
Teste ambas as aplicações.

--

Videos de apoio

- cliente echo
- servidor echo

Outras dicas:

possível importação de um projeto NB para IntelliJ IDEA:

- https://drive.google.com/file/d/1-WcCfvm9d0w3kv-uNz8ZXdmHMn2WO4l8/view
- https://www.jetbrains.com/help/idea/netbeans.html#working-with-projects