

# 36019475. CSIFC03. MP0485. Programación

**XUNTA DE GALICIA**CONSELLERÍA DE CULTURA, EDUCACIÓN  
E ORDENACIÓN UNIVERSITARIA

Páxina principal ► Os meus cursos ► Formación Profesional a Distancia ► Curso 2016-2017 ►  
36019475 IES de Rodeira ► CSIFC03 Desenvolvemento de aplicacións web ►  
125\_36019475\_ZSIFC03\_MP0485\_A ► Unidade didáctica 6 ► Introducción á comunicación con Sockets

## NAVEGACIÓN

Páxina principal

● Amiña área persoal

Páxinas do sitio

O meu perfil

Curso actual

125\_36019475\_ZSIFC03\_MP0485\_A

Participantes

Distincións

Xeral

Unidade didáctica 1

Unidade didáctica 2

Unidade didáctica 3

Unidade didáctica 4

Unidade didáctica 5

Unidade didáctica 6

Orientaciones para el alumnado. PROG06.

Solución a la tarea para PROG06.

PROG06 Guiada.- Almacenando datos.

Actividades presenciales de la UD6 en la tutoría c...

Tarefa 6 - Soluciona da Titoria Presencial

Foro para PROG06.

Mapa conceptual para PROG06.

PROG06 Completa.- Almacenando datos.

Recursos complementarios UD06.

1.- Introducción a Entrada/Saída en Java

Tarefa a Entregar 1

Solución Tarefa 1

2.- Fluxos Binarios e Fluxos de Caracteres

3.- Acceso a Ficheiros

Tarefa a Entregar 2

Solución

4.- Traballando con Streams Binarios

5 - Traballando con Fluxos de Caracteres

Tarefa a Entregar 3

Solución

6 - Traballando con Ficheiros de Acceso Aleatorio ...




**Introducción á comunicación con Sockets**

Leeweb.java

Chat

Tarea para PROG06.

Solución Alternativa

 Tarefa a Entregar 4 Solución (Versión sen clase para acceso a ficheiros) Solución (Con clase independente para acceder aos ... Tarefa a entregar 5 Solución Tarefa a Entregar 6 Modificacións Propostas Examen de la UD06. Aplicación Hotel Completa

Unidade didáctica 7

Unidade didáctica 8

Unidade didáctica 9

Unidade didáctica 10

Unidade didáctica 11

Os meus cursos

## ADMINISTRACIÓN

Administración do curso

Configuración do meu perfil

## Introducción á comunicación con Sockets

Unha das características da serialización é que nos pode permitir o envío de obxectos a través da rede, recollendoos e reconstruíndo os mesmos dende o outro lado. Para realizar este tipo de actividades, Java dispón de dous métodos principais:

- **Mediante a clase URL** - Nos proporciona a posibilidade de leer información dun servidor
- **Mediante as clases Socket/ServerSocket** - Nos proporciona a posibilidade de establecer unha comunicación bidireccional entre dous procesos.

*Socket* e *ServerSocket* proporcionan acceso ás técnicas de programación sobre TCP e UDP. A clase *Socket* proporciona unha interfaz de socket para cliente, similar aos sockets estándar de UNIX. Para abrir unha conexión é necesario crear unha nova instancia de *Socket(NomeHost, Porto)* e posteriormente utilizar os fluxos de entrada e de saída para ler ou escribir no mesmo :

```
Socket connexion=new Socket(NomeHost, Porto);
```

```
BufferedInputStream datain=new BufferedInputStream(connection.getInputStream());
```

```
BufferedOutputStream dataout=new BufferedOutputStream(connection.getOutputStream());
```

```
DataInputStream in=new DataInputStream(datain);
```

```
DataOutputStream out=new DataOutputStream(dataout);
```

Unha vez que se rematou de traballar co socket é necesario pechalo con *connection.close()*;

Os sockets do lado servidor traballan dun modo similar. Un socket servidor 'escoita' nun porto TCP esperando por unha petición de conexión por parte dun cliente, e mediante o método *accept()* pode aceptar a mesma. Para crear un socket de servidor e asocialo a un porto é necesario crear un novo obxecto *ServerSocket*:

```
ServerSocket escoita=new ServerSocket(38910);
```

e para escoitar no porto e aceptar novas conexións utilízase o método *accept*:

```
escoita.accept();
```

Unha vez feito isto pódense utilizar fluxos de entrada e saída para comunicarse co cliente.

A hora de traballar con conexións entre distintas máquinas é importante, tanto a posibilidade de poder atender a varias conexións ao mesmo tempo (caso dun servidor), como que os tempos de espera para a recepción de datos non conxelen o resto do procesamento (especialmente importante nas aplicacións con interface gráfica de usuario). Tradicionalmente, para que unha aplicación poda realizar máis de unha acción ao mesmo tempo se empregan dúas aproximacións distintas:

- **Procesos** - Se lanza un proceso (unha copia exacta do programa en funcionamento) para atender cada procesamento que queremos que se realice de xeito simultáneo.

Os sistemas Windows tradicionalmente utilizan Threads para a multitarefa, mentras que os sistemas Linux/UNIX tradicionalmente utilizan o sistema de Procesos. JAVA utiliza Threads.

O uso de threads é máis eficiente que a utilización de Procesos, xa que evita a necesidade de crear un novo proceso para cada acción concurrente, aforrando memoria e o tempo necesario para a creación de procesos, pero plantexa outra serie de problemas que non existen na aproximación mediante procesos, sendo o máis importante a sincronización entre os distintos fios (Os atributos son os mesmos para todos os fios de execución: O que implica que un fio de execución pode afectar ao resto, sendo necesario o bloqueo de recursos).

Para crear programas multitarefa en Java existen dúas aproximacións:

- Mediante a implementación da interface *Runnable*
- Extendendo (heredando) da clase *Thread*

O interface Runnable dispón dun método *run* que implementará as accións a realizar de xeito simultáneo a execución da aplicación. A clase *Thread* é unha clase que xa implementa o interface Runnable, pero cun método run que por defecto non fai nada (sendo necesario sobreescrilo).

Para máis información:

<https://docs.oracle.com/javase/tutorial/essential/concurrency/>  
<https://docs.oracle.com/javase/7/docs/api/java/lang/Runnable.html>  
<https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html>  
<https://docs.oracle.com/javase/tutorial/essential/concurrency/syncmeth.html>

Última modificación: Martes, 13 de Decembro do 2016, 19:16





