Xarxes de Computadors Curs 2015-2016

Pràctica 2: Introducció a Java.

1 Objectiu

L'arquitectura Java fa possible crear aplicacions totalment portables a diferents plataformes. Les seves característiques la fan idònia per al desenvolupament d'aplicacions en entorns de xarxa. Des de 1995, any d'aparició pública de Java, aquest llenguatge ha anat guanyant acceptació entre la comunitat de programadors i fabricants de programari.

Amb aquesta pràctica l'alumne adquirirà els coneixements bàsics de programació en Java. Aprendrà a utilitzar la seva API i serà capaç d'utilitzar tècniques avançades de programació en Java com la programació concurrent (*Multi-thread*) i les comunicacions en Internet.

2 Enunciat

Cada grup programarà **un servidor HTTP (*Hyper Text Transfer Protocol*)**.
El servidor HTTP `Apache`, és el servidor HTTP més utilitzat en el món. Nosaltres en farem una versió **molt** més reduïda.
El nostre servidor serà un programa que servirà fitxers ubicats en el seu sistema de fitxers.
Un altre programa, que anomenarem client, es connectarà al nostre servidor i li demanarà descarregar-se un fitxer.
Els missatges que s'enviaran el programa client i el programa servidor són missatges del protocol HTTP.
No caldrà que programem el programa client. Utilitzarem qualsevol navegador, `Firefox`, `Chrome`, `Safari`, `Opera`, etc..., que són clients `HTTP`.
Mitjançant el navegador, doncs, ens connectarem al programa servidor, li demanarem un fitxer i el navegador, automàticament, l'interpretarà, si cal, i el visualitzarà, o bé se'l descarregarà, que això és el que fan els navegadors.

3 Comunicació entre el client i el servidor

La comunicació entre el client i el servidor es farà mitjançant *sockets Internet*.
Un socket és una estructura de dades que s'utilitza per enviar i rebre dades cap a,

o desde la xarxa.
Així a cada aplicació que s'estigui executant en una màquina, que li calgui enviar/rebre dades a la xarxa haurà d'utilitzar aquesta estructura.
Associarem cada una d'aquestes estructures, sockets, a una aplicació concreta dins de la màquina.
A més per distingir entre les diferents aplicacions que estan usant sockets, cada socket estarà associat a un número, un port.

El **servidor** crearà un *socket*, associat a un port, dedicat a escoltar peticions HTTP que li arribin del client.
Per a cada petició que rebi es crearà un nou *socket* per continuar la comunicació amb el client.

El codi **Java** per a implementar aquest esquema de comunicació és el següent:

```
....

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

public class WServer
{
 .....

 public static void main (String[] args) {
   ....
   ServerSocket serverSocket;
   Socket socket2;
   InputStream is;
   OutputStream os;

   try
   {
     serverSocket = new ServerSocket( 8811 );
     while(true)
     {
       //obtenim socket per comunicar-nos amb el client
       socket2 = serverSocket.accept();

       // Ara podem obtenir els fluxes d'entrada i
       // sortida associats al client:
       is = socket2.getInputStream();
```

```
        os = socket2.getOutputStream();
      }
    }catch (IOException e) {
       e.printStackTrace();
     }
   }
}
```

El *socket* `serverSocket` només el dedicarem a esperar rebre connexions de clients.
Un cop acceptem aquestes connexions, el servidor crearà unaltre `socket`, la variable `socket2` que utilitzarà per continuar la comunicació, enviar i rebre dades, amb el client que li ha fet la petició `HTTP`.

Des de el **servidor**, mitjançant aquest *socket*, variable `socket2`, podeu:

- Obtenir un fluxe de dades d'**entrada**, per **rebre/llegir** les dades que us envia el client.

  Aquest fluxe de dades el teniu guardat en la variable `is`:
  `is = socket.getInputStream();`

- Obtenir un fluxe de dades de **sortida**, per **enviar** dades cap al client.

  Aquest fluxe de dades el teniu guardat en la variable `os`:
  `os = socket.getOutputStream();`

  És molt possible que diferents grups de pràctiques estiguin executant el seu servidor en el mateix host. Caldrà, doncs, que **cada grup de pràctiques tingui un port diferent** associat al socket que utilitzarà el seu servidor per la comunicació.

  Per saber quin és el port que utilizareu farem el següent càlcul:

  ```
  Port = 8000 + grup + sub_grup
  ```

  A on grup s'ha de canviar per: A→100, B→200, C→300, etc. Així, el grup `x-a11` tindrà com a port el 8111.
  D'aquesta manera els vostres servidors estaran escoltant per ports diferents tot i que s'estiguin executant en la mateixa màquina.

4 Protocol HTTP per la comunicació entre client i servidor

4.1 Client

Quan desde el client us vulgueu connectar al servidor per demanar-li un fitxer, l'enllaç que heu de posar en el navegador tindrà el següent format:

```
http://host_on_esta_el_servidor_corrent:port/
  fixer_que_volem.extensio[?parametre=valor[&parametre=valor]*]
```

Noteu que en `[?parametre=valor[&parametre=valor]*]` els corxets `[]` indiquen que és opcional.

Pel que fa al format de l'enllaç:

http: Indiquem que el client *parlarà* HTTP amb el servidor.

host_on_esta_el_servidor_corrent És el host, la màquina, on tenim executant-se el servidor.

port Port associat al servidor.

fixer_que_volem.extensio És el fitxer que ens volem descarregar o visualitzar. Aquest fitxer està en el sistema de fitxers del servidor.

&parametre=valor Ho veurem més endavant.

Suposem que el servidor el tenim executant-se en la màquina `deic-dc3.uab.cat` i teniu assignat el port `8811`.
Si desde el client voleu connectar-vos al servidor i demar-li el fitxer `index.html` caldrà que especifiqueu l'enllaç:

```
http://deic-dc3.uab.es:8811/index.html
```

A partir d'aquest enllaç el client, automàticament, envia al servidor la següent petició HTTP:

```
GET /index.html HTTP/1.1\n\n
```

4.2 Servidor

En el servidor per a a llegir/rebre aquesta cadena, utilitzareu el fluxe d'entrada, variable **is** que vam obtenir de la variable `socket2`, associat a la connexió establerta amb el client.
Acte seguit el servidor ha d'enviar la resposta, el fitxer `index.html` al client mitjançant el protocol `HTTP`.
Per a tal efecte, el primer que ha de fer és enviar capçaleres `HTTP` amb la informació del document que vol transmetre. Després de les capçaleres ha d'enviar el fitxer:

```
[Capçalarea HTTP de resposta\n]*\n
Contingut del fitxer
```

Associada a la petició HTTP: `GET index.html HTTP1.1` que hem rebut, hauríem de tornar el següent missatge:

```
HTTP/1.1 200 OK\n
Content-Type: text/html\n\n
<!DOCTYPE html><html><head></head><body>....</body></html>
```

Concretament la cadena:

```
HTTP/1.1 200 OK\nContent-Type: text/html\n\n<!DOCTYPE html>....
```

Fixeu-vos que amb aquesta capçalera estem indicant la versió del protocol HTTP, el tipus del fitxer que enviem i el nom amb el que el client pot desar aquest fitxer en el seu sistema de fitxers.

Per a enviar aquestes dades cap al client utilitzarem el fluxe de sortida, variable **os** que vam obtenir de la variable `socket2`, associat a la connexió establerta amb el client.

4.2.1 Ruta dels fitxers en el servidor

Els fitxers que el servidor serveix es troben en el vostre directori **`java/files`**. Trobareu que ja teniu uns quants fitxers però en podeu afegir més.

Per a que el vostre servidor trobi aquests fitxers en temps d'execució, no heu d'especificar cap ruta en el servidor. Només cal que executeu el servidor des d'aquest directori:

```
cd java/files
java -cp ../src WServer
```

4.2.2 Tipus de fitxers que podem servir

Els diferents tipus de fitxers que podrà servir el servidor, `tipus mime`, seran:

`text/plain`	per a fitxers de texte
`text/html`	per a fitxers HTML
`image/gif`	per a imatges en format gif
`image/jpeg`	per a imatges en format jpeg
`image/png`	per a imatges en format png
`application/zip`	per a fitxers comprimits en format zip
`application/x-gzip`	per a fitxers comprimits en format gzip
`application/xml`	per a fitxers XML
`application/octet-stream`	per a la resta de fitxers

El *tipus mime* l'especificarem en el camp `Content-Type` de les capçaleres `HTTP` que enviem com a resposta, indicades en la secció anterior.

```
HTTP/1.1 200 OK\n
Content-Type: text/html\n\n
```

4.2.3 Fitxers que no son de text ni HTML

En el cas de fitxers que no siguin ni text ni `HTML`, el navegador no els pot interpretar ni visualitzar.
En aquest cas, si els voleu desar a disc, en el sistema de fitxers del client, a part d'especificar la capçalera `Content-Type`, caldrà que especifiqueu la capçalera `Content-Disposition`, on indicareu el nom amb el que voleu desar el fitxer a disc en el client.

```
Content-Disposition: filename="fitxer.html.zip"\n
```

4.2.4 Page not found

Si el fitxer no existeix, el servidor, com a resposta, retornarà el codi `404` indicant que la pàgina no existeix. Per a tal efecte només caldrà que el sevidor enviï la capçalera:

```
HTTP/1.1 404 Not Found\n\n
```

5 Fluxes d'entrada i sortida: `is` i `os`

És molt important, per al bon funcionament de la pràctica, que en acabar d'enviar el fitxer, tanqueu, en aquest ordre els fluxes de dades:

1. el fluxe de sortida de comunicació amb el client
2. el *socket* que heu utilitzat per comunicar-vos amb el client. No tanqueu el *ServerSocket* del vostre servidor ja que sempre ha d'estar en funcionament escoltant peticions que li puguin arribar per aquest socket.

Tingueu en compte el següent:

- Per a treballar amb els fluxes d'entrada i sortida, podeu utilitzar directament les classes de la Api de Java `java.io.InputStream` i `java.io.OutputStream`.
- Per llegir la petició que us ha fet el client podeu utilitzar la classe: `java.io.BufferedReader`
- Per llegir el fitxer que heu d'enviar al client podeu utilitzar la classe `java.io.FileInputStream`
- A partir d'un mètode de la classe *java.lang.String* podeu obtenir un array de bytes. Consulteu la `Api`.

6 Paràmetres de la petició HTTP

Com hem vist, una petició HTTP que genera el client i que rep el servidor té el següent format:

```
http://host:port/fixer[?parametre=valor[&parametre=valor]*]
```

Les peticions que rep el nostre servidor, poden venir acompanyades de paràmetres. Aquests paràmetres indiquen el format amb el que hem de tornar el fitxer.
A continuació us detallem quins són aquests paràmetres i la funcionalitat que ha d'implementar el servidor en funció d'aquests paràmetres.

6.1 Paràmetre asc

Si s'especificica aquest paràmetre cal fer una conversió lleugera de `HTML` cap a `plain-ASCII-text`.

L'enllaç que posaríeu en en navegador seria:

```
http://deic-dc3.uab.es:8811/index.html?asc=true
```

En navegador generaria la següent petició HTTP:

```
GET /index.html?asc=true HTTP/1.1\n\n
```

Si s'ha indicat el paràmetre `asc` el servidor cal que tregui els *tags* del fitxer que s'ha demanat en la petició `HTTP`, abans d'enviar-li al client. Aquesta manipulació només és aplicable als fitxers que siguin de tipus *text/html*.

Noteu que en aquest cas la capçalera de resposta que enviarà el servidor al client: **`Content-Type`** ha de pendre el valor:

```
Content-Type: text/html\n
```

Aquesta transformació consisteix en ignorarar les parts del document amb la sintaxis: `< byte* >` (fragments del document entre els símbols: '<' i '>'). Per exemple, el següent document HTML:

```
<html>
<head><title>Document de xarxes</title>
<body>
Aquest és el contingut del document
</body>
</html>
```

En aplicar el filtre, us quedaria:

```
\n
Document de xarxes\n\n
Aquest és el contingut del document.\n
\n\n
```

Cal que l'algorisme per enviar el contingut del fitxer sigui independent d'aquesta conversió. És molt important que no dupliqueu codi. Per a aconseguir això cal que implementeu la classe:

```
class AsciiInputStream extends FilterInputStream
```

Caldrà que feu ús del **polimorfisme** de la classe `java.io.InputStream`. Recordeu que una classe derivada com: `AsciiInputStream` o `FilterInputStream`, es pot tractar com si fos qualsevol de les seves superclasses (polimorfisme). Vegeu l'arbre de la jerarquia de la classe `java.io.FilterInputStream`.

Així la resposta `HTTP` que enviarà el servidor al client serà:

```
HTTP/1.1 200 OK\n
Content-Type: text/html\n\n
\n
Document de xarxes\n\n
Aquest és el contingut del document.\n
\n\n
```

6.2 Paràmetre zip

Si s'especifica el paràmetre `zip`, s'enviarà el fitxer comprimit utilitzant el format de compressió `ZIP`.

L'enllaç que posaríeu en en navegador seria:

```
http://deic-dc3.uab.es:8811/index.html?zip=true
```

El navegador generaria la següent petició HTTP:

```
GET /index.html?zip=true HTTP/1.1\n\n
```

En fer aquesta conversió, en les capçaleres `HTTP`, caldrà afegir l'extensió '`.zip`' al nom del fitxer, i el tipus mime del fitxer serà: `application/zip`

Com que aquest tipus de fitxer no pot ser interpretat pel navegador afegirem la capçalera `Content-Disposition` amb el nom amb el que volem que es desi el fitxer `zip` en el sistema de fitxers del client.
En rebre aquesta capçalera el navegador obrirà un diàleg preguntant si volem obrir o desar a disc el fitxer.

```
HTTP/1.1 200 OK\n
Content-Type: application/zip\n
Content-Disposition: filename="index.html.zip"\n\n
```

Per a la implementació de la compressió podeu utilitzar la classe de la Api de Java:
`java.util.zip.ZipOutputStream`.

Un fitxer zip és un `archive` que conté fitxers comprimits. Cada un d'aquests fitxers és una entrada d'aquest `archive`. Vegeu el mètode de la classe `java.util.zip.ZipOutputStream`:

```
public void putNextEntry(ZipEntry e) throws IOException
```

Des del client, amb la comanda `unzip <nom_fitxer>` podreu comprovar si el fitxer s'ha comprimit correctament.

Noteu que la idea és que **imbriqueu els fluxes** de sortida d'aquesta classe i l'`outputSteam` que heu obtingut del socket que utilitzeu per a la comunicació amb el client, variable `os`. **No** pas que **creeu**, en el sistema de fitxers del servidor, **un nou fitxer comprimit** i que després l'envieu al client.
De nou caldrà que feu ús del **polimorfisme** de la classe **`java.io.OutputStream`**.

6.3 Paràmetre gzip

L'opció `gzip` farà una compressió de manera similar a l'anterior, però utilitzant l'algorisme `GZIP`.

L'enllaç que posaríeu en en navegador seria:

```
http://deic-dc3.uab.es:8811/index.html?gzip=true
```

En navegador generaria la següent petició `HTTP`:

```
GET /index.html?gzip=true HTTP/1.1\n\n
```

En aquest cas, caldrà afegir, en les capçaleres `HTTP`, l'extensió '`.gz`' al nom del fitxer i el tipus mime serà: `application/x-gzip`.

```
HTTP/1.1 200 OK\n
Content-Type: application/x-gzip\n
Content-Disposition: filename="index.html.gz"\n\n
```

Per a la implementació de la compressió podeu utilitzar la classe de la Api de Java: **`java.util.GZIPOutputStream`**.

Des del client amb la comanda `gzip -d <nom_fitxer>` podreu comprovar si el fitxer s'ha comprimit correctament.

6.4 Paràmetres zip i gzip

En cas d'indicar-se ambdues opcions de compresió simultàniament, el fitxer resultant ha de ser un fitxer `GZIP` que contingui un fitxer `ZIP`, dins del qual hi haurà el fitxer original.
El fitxer resultant tindrà el nom: `index.html.zip.gz`.

L'enllaç que posaríeu en en navegador seria:

```
http://deic-dc3.uab.es:8811/index.html?zip=true&gzip=true
```

En navegador generaria la següent petició `HTTP`:

```
GET /index.html?zip=true&gzip=true HTTP/1.1\n\n
```

Fixeu-vos que el tipus de fixer resultant és un `gzip`. Noteu que l'extensió més externa és la de `gz`. Per tant, en aquest cas, caldrà afegir, en les capçaleres `HTTP`, l'extensió '`.gz`' al nom del fitxer i el tipus mime serà: `application/x-gzip`.

```
HTTP/1.1 200 OK\n
Content-Type: application/x-gzip\n
Content-Disposition: filename="index.html.zip.gz"\n\n
```

La implementació de la compressió l'haureu de fer imbricant els fluxes de sortida de les classes de la Api `java.util.GZIPOutputStream` i `java.util.zip.ZipOutputStream`. Jugant, com sempre, amb el polimorfisme respecte la classe pare de totes dues: `java.io.OutputStream`. **No heu de crear fitxers intermitjos**.

6.5 Paràmetres asc, zip i gzip

Finalment, una petició a on s'especifiquen els paràmetres: `asc, zip` i `gzip` ha de retornar un fitxer amb el nom: `nom_fitxer.html.asc.zip.gz`.

L'enllaç que posaríeu en en navegador seria:

```
http://deic-dc3.uab.es:8811/index.html?asc=true&zip=true&gzip=true
```

El navegador generaria la següent petició `HTTP`:

```
GET /index.html?asc=true&zip=true&gzip=true HTTP/1.1\n\n
```

Fixeu-vos que el tipus de fixer resultant és un `gzip`. Noteu que l'extensió més externa és la de `gz`. Per tant en aquest cas, caldrà afegir, en les capçaleres `HTTP`, l'extensió '`.gz`' al nom del fitxer i el tipus mime serà: `application/x-gzip`.

```
HTTP/1.1 200 OK\n
Content-Type: application/x-gzip\n
Content-Disposition: filename="index.html.asc.zip.gz"\n\n
```

En aquest fitxer resultant el primer que li hem aplicat ha sigut treure-li els tags HTML, al resultat d'això l'hem comprimit en format zip i al resultat d'aquesta compressió l'hem aplicat la compressió amb format GZIP.
Tot això sense haver creat cap fitxer intermig!!!

Noteu el com hem anat construïnt el nom del fitxer resultant:

- La primera transformació ha implicat afegir-li al fitxer `index.html` l'extensió `.asc`, `index.html.`**`asc`**.
- El fitxer `index.html.asc` l'hem comprimit en `zip` i per tant li hem afegit l'extensió `.zip`, `index.html.asc.`**`zip`**.
- En aplicar la compressió en gzip li hem hagut d'afegir al fitxer l'extensió `gz`, `index.html.asc.zip.`**`gz`**

Noteu també que l'ordre en el que li passem els paràmetres és indiferent.
Quan el client rep la petició `HTTP`, a través de `is` la rep com un `String`, com una cadena. Tampoc cal que comproveu el valor dels paràmetres, `asc`, `zip` i `gzip`, directament comproveu si hi han aquests paràmetres.

Podeu combinar aquests paràmetres com vulgueu:

```
http://deic-dc3.uab.es:8811/index.html?asc=true&zip=true&gzip=true
http://deic-dc3.uab.es:8811/img.jpg?asc=true&gzip=true
http://deic-dc3.uab.es:8811/text.txt?zip=true&gzip=true
http://deic-dc3.uab.es:8811/a.html?asc=true
http://deic-dc3.uab.es:8811/b.html
etc...
```

Noteu que us podeu descarregar qualsevol dels fitxers que teniu en el directori `files` del servidor.

7 Concurrència

L'aplicació haurà de **servir en paral·lel totes les peticions** de fitxers que li facin els clients. Penseu que podeu tenir molts clients fent-vos peticions simultàniament. Per a tal efecte es llençarà un `Thread` per servir cada una de les peticions que es rebin.

Per a implementar la concurrència en Java, vegeu la classe `java.lang.Thread` a la API.

Per a cada petició d'un fitxer, doncs, la vostra aplicació llençarà un *Thread* que durà a terme les tasques de:

- Obtenció del nom del fitxer de la petición HTTP rebuda.
- Transformació del fitxer, si s'escau.
- L'enviament del contingut del fitxer.

8 Com executar el Servidor

Si considerem que teniu les classes `java` en el directori `java/bin`, el servidor l'executariem d'aquesta manera:

```
cd java/files
java -cp ../bin WServer [-p <port>]
```

on l'argument opcional `<port>` és el port on escoltarà el servidor les peticions que li arribin dels clients. Per defecte serà el que teniu assignat al vostre grup.

Per executar-ho des de l'`Eclipse` ho explicarem a classe.

9 Opcional

9.1 Opció ascii avançada

Modificar el comportament de l'opció `asc` per, a més a més de la funcionalitat d'eliminar els tags HTML, eliminar també els comentaris HTML:
<!-- Un comentari HTML -->.

Recordeu que poden haver símbols '<' o '>' dins dels comentaris, per exemple codi Javascript.

Aquesta part opcional està valorada amb **1 punt**.

9.2 Generar la documentació de les vostres classes amb `javadoc`

Java ofereix un format molt específic per comentar el vostre codi font. Cal que utilitzeu aquest format per comentar les classes que definiu i tots els mètodes d'aquestes classes.

Amb l'aplicació `javadoc` de Java, a partir dels comentaris que heu fet al vostre codi, es genera, en format HTML, la documentació de la vostra aplicació seguint

el look and feel de la API de Java.

No es valorarà que només executeu aquesta aplicació havent afegit molt pocs o cap comentari al vostre codi.

Aquesta part opcional està valorada amb 1 punt.

10 Observacions

- L'equivalent a la funció en C `printf(char*)`, en Java és:
  - `System.out.print(String)` o bé
  - `System.out.println(String)` per afegir un retorn de carro.

  De totes maneres, per a més informació, mireu a la API de Java l'atribut `out` en la classe `System`.
- Aquesta pràctica es corregirà desde el vostre compte de pràctiques.
- La versió de Java que hi ha instal·lada és la `1.8.74 de Oracle`.

11 Proposta de planificació de la pràctica

Febrer	22	Servir un fitxer sense tenir en compte la petició HTTP feta pel client.
	29	Implementar la concurrència; Tractar l'opció: asc
Març	7	**Control 1: Funcionalitat de la setmana del 22 i del 29** Tractar opcions: zip; gzip i zip+gzip;
	14	**Control 2: Opcions zip i Gzip i zip+Gzip** Tractar la petició `HTTP`
	21	*Setmana Santa:*: Opcionals.
	28	**Entrega final.**

12 Lliurament de la pràctica

Heu d'exportar el vostre projecte `WServer` en un fitxer comprimit en format `jar`, *java archive*: `WServer.jar`.
Aquest fitxer l'heu de desar en el vostre directori `entregues/java/final` **una hora abans** de la sessió d'entrega de la **setmana del 28 de març**.
Una hora abans de la vostra sessió copiarem de forma automàtica els vostres comptes de pràctiques.