

# Introducció als llenguatges de servidor

Àlex Salinas Tejedor

Desenvolupament web en entorn servidor



# Índex

<b>Introducció</b>	<b>5</b>
<b>Resultats d'aprenentatge</b>	<b>7</b>
<b>1 Entorns a punt per desenvolupar aplicacions web</b>	<b>9</b>
1.1 El vostre primer projecte web: 'Hola, Món' . . . . .	10
1.1.1 Creació d'un projecte amb Maven . . . . .	10
1.1.2 Exemple 'Hola, Món' . . . . .	13
1.1.3 Pàgines dinàmiques i servidors d'aplicacions . . . . .	21
1.1.4 Què s'ha après? . . . . .	24
1.2 El vostre primer projecte PHP: 'Hola, Món' . . . . .	25
1.2.1 Posar a punt el servidor web . . . . .	25
1.2.2 Creació d'un projecte PHP en un servidor remot . . . . .	29
1.2.3 Exemple 'Hola, Món' . . . . .	33
1.2.4 Què s'ha après? . . . . .	36
<b>2 PHP i VDL a Java EE: dades, estructures de control i 'arrays'</b>	<b>37</b>
2.1 Descriuint una persona amb PHP . . . . .	38
2.1.1 Conversions de tipus . . . . .	40
2.1.2 Àmbit de les variables . . . . .	43
2.2 Gestionant un hotel amb PHP . . . . .	45
2.2.1 Gestió de les taules del restaurant del hotel . . . . .	46
2.2.2 Gestió de les habitacions de l'hotel . . . . .	50
2.3 Descriuint una persona a JSP, JSTL . . . . .	57
2.3.1 Sintaxi estàndard JSP . . . . .	57
2.3.2 Sintaxi JSP Standard Tag Library (JSTL) . . . . .	63
2.3.3 Calcular l'edat d'una persona . . . . .	67
2.4 Gestionant un hotel amb JSP . . . . .	69
2.4.1 Gestió de les taules d'un restaurant . . . . .	69
2.4.2 Gestió de les habitacions d'un hotel . . . . .	75
2.5 Què s'ha après? . . . . .	81



## Introducció

Al començament, l'èxit espectacular de la web va tenir lloc gràcies al protocol HTTP i al llenguatge HTML. El protocol HTTP és una implementació fàcil i senzilla d'un sistema de comunicacions que permet enviar qualsevol tipus de fitxers per la xarxa. Aquest protocol permet, a més, atendre milers de peticions reduint la potència dels servidors i, com a conseqüència, reduint els costos de desplegament dels serveis.

La web era un conjunt de pàgines estàtiques, documents simples d'informació que podien consultar-se o descarregar-se. A poc a poc va anar evolucionant fins a permetre desenvolupar pàgines dinàmiques que podien ser creades en funció de la petició enviada. Aquest mètode es va conèixer com a CGI (Common Gateway Interface). Els CGI defineixen una comunicació entre el client i el servidor mitjançant HTTP on els clients podien demanar informació als programes executats en el servidor. CGI especifica un estàndard per transferir dades entre el client i el programa. És un mecanisme de comunicació entre el servidor web i una aplicació externa. El resultat final de l'execució són objectes MIME. Aquestes aplicacions, que s'executen en el servidor, reben el nom de CGI.

Però els CGI tenen un punt feble: cada vegada que reben una petició, el servidor web comença un procés on executa el programa CGI. Com, d'altra banda, la majoria de CGI estaven escrits en algun llenguatge interpretat (Perl, Python, etc.), això implica una gran càrrega per a la màquina del servidor. A més, si la web té molts accessos al CGI, això suposa problemes greus de rendiment.

Per això es comencen a desenvolupar alternatives als CGI. Les solucions consisteixen a dotar el servidor d'un intèrpret d'algun llenguatge de programació (PHP, JSP, etc.) que permeti incloure el codi en les pàgines de manera que el servidor sigui qui ho executi, reduint així el temps de resposta.

A partir d'aquest moment es viu una explosió del nombre d'arquitectures i llenguatges de programació que ens permeten desenvolupar aplicacions web. Sorgeixen els llenguatges de programació integrats en el servidor que permeten interpretar instruccions incrustades a les pàgines HTML i un sistema d'execució de programes més enllaçat amb el servidor que no presenta els problemes de rendiment dels CGI.

En aquesta unitat abordarem amb detall uns llenguatges que permeten incrustar codi interpretable a les pàgines HTML i que el servidor tradueix a programes executables, JSP (Java Server Pages) i PHP (Hypertext Preprocessor). Però primer començarem la unitat explicant l'entorn de desenvolupament NetBeans, eina que ens ajudarà molt a l'hora d'escriure programes. S'explicarà com crear projectes fàcilment exportables utilitzant Maven i s'explicarà la diferència entre una pàgina web estàtica i una pàgina dinàmica.

En l'apartat "Entorns a punt per desenvolupar aplicacions web" estudiarem també el funcionament d'un servidor web i d'un servidor d'aplicacions, i veurem una pinzellada del protocol HTTP. S'acabarà l'apartat instal·lant i configurant un servidor PHP on puguem executar els exercicis realitzats.

En l'apartat "PHP i VDL a Java EE: dades, estructures de control i *arrays*" ens endinsarem en els llenguatges JSP i PHP. S'explicaran les peculiaritats d'ambdós llenguatges i s'explicaran les estructures de control i les funcions més utilitzades de les seves llibreries.

En acabar la unitat serem capaços d'instal·lar i configurar un servidor d'aplicacions, tant en JSP com en PHP, i crear aplicacions web senzilles que ens permetin gestionar petits negocis. Tots els apartats d'aquesta unitat s'han elaborat proposant un exemple pràctic per introduir tots els conceptes abans esmentats. Es recomana que l'estudiant faci els exemples mentre els va llegint. Així, anirà aprenent mentre va practicant els conceptes exposats. Finalment, per treballar completament els continguts d'aquesta unitat és convenient anar fent les activitats i els exercicis d'autoavaluació.

## Resultats d'aprenentatge

En finalitzar aquesta unitat, l'alumne/a:

**1.** Selecciona les arquitectures i tecnologies de programació web en entorn servidor, analitzant les seves capacitats i característiques pròpies.

- Caracteritza i diferencia els models d'execució de codi al servidor i al client web.
- Reconeix els avantatges que proporciona la generació dinàmica de pàgines web i les seves diferències amb la inclusió de sentències de guions a l'interior de les pàgines web.
- Identifica els mecanismes d'execució de codi en els servidors web.
- Reconeix les funcionalitats que aporten els servidors d'aplicacions i la seva integració amb els servidors web.
- Identifica i caracteritza els principals llenguatges i tecnologies relacionats amb la programació web en entorn servidor.
- Verifica els mecanismes d'integració dels llenguatges de marques amb els llenguatges de programació en entorn servidor.
- Reconeix i avalua les eines de programació en entorn servidor.

**2.** Escriu sentències executables per un servidor web reconeixent i aplicant procediments d'integració del codi en llenguatges de marques.

- Identifica els mecanismes de generació de pàgines web a partir de llenguatges de marques amb codi encastrat.
- Identifica les principals tecnologies associades.
- Utilitza etiquetes per a la inclusió de codi en el llenguatge de marques.
- Reconeix la sintaxi del llenguatge de programació que s'ha d'utilitzar.
- Escriu sentències simples i comprova els seus efectes en el document resultant.
- Utilitza directives per modificar el comportament predeterminat.
- Empra els diferents tipus de variables i operadors disponibles en el llenguatge.
- Identifica els àmbits d'utilització de les variables.





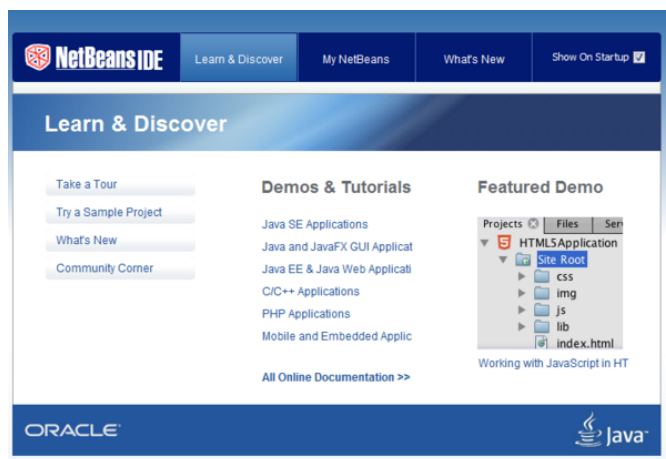
## 1. Entorns a punt per desenvolupar aplicacions web

Java és important. Els desenvolupadors web necessiten un coneixement exhaustiu del llenguatge que fa que les aplicacions web cobrin vida. Per poder començar a codificar en aquest llenguatge és important disposar d'un entorn de desenvolupament on puguem realitzar totes les tasques associades a la programació:

- editar codi
- compilar-lo
- executar-lo
- depurar-lo

L'IDE (*Integrated Development Environment*) Netbeans us permet, d'una manera senzilla, elaborar programes complexos portant a terme totes les tasques abans esmentades. Per això, us animem a fer la primera activitat que tracta la instal·lació del IDE NetBeans (figura 1.1).

FIGURA 1.1. IDE Netbeans



En aquesta unitat posareu les bases de l'aprenentatge d'aplicacions web amb Java, tot abordant els següents continguts:

- Com crear un projecte web amb NetBeans i Maven.
- Quina diferència hi ha entre un servidor web i un servidor d'aplicacions.
- Com interacciona un navegador amb el servidor web.
- El protocol HTTP.
- Els diferents tipus de tecnologies que poden aparèixer en una aplicació web.

A més a més, es donarà una pinzellada al llenguatge de programació PHP, tot estudiant els següents aspectes:

- Com instal·lar i configurar un servidor PHP.
- Com configurar NetBeans per treballar amb un servidor PHP remot.
- Com interacciona un navegador web amb el servidor PHP.
- Com crear una pàgina senzilla amb aquest llenguatge.

Tots els conceptes tractats s'explicaran sempre partint de l'exemple. Quan acabeu aquesta unitat estareu preparats per endinsar-vos en la programació amb Java Servlets.

## 1.1 El vostre primer projecte web: 'Hola, Món'

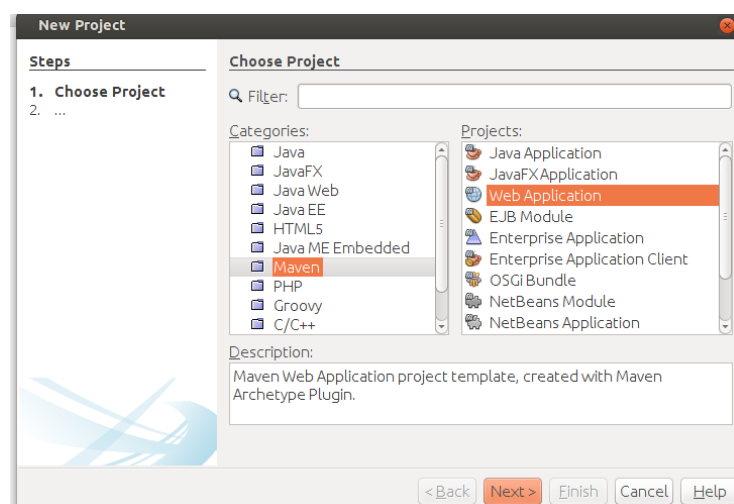
En aquest apartat veureu com crear un projecte web amb Netbeans utilitzant Maven, què és un servidor web, com interaccionen els navegadors amb els servidors web, quins tipus de servidors existeixen i quines tecnologies s'executen al servidor i quines al navegador.

### 1.1.1 Creació d'un projecte amb Maven

Utilitzareu Maven per administrar els projectes que creareu al llarg d'aquest mòdul. Maven és una eina d'administració de projectes que engloba tot el cicle de vida d'una aplicació, des de la seva creació fins als binaris amb els quals distribuir el projecte.

Per crear un nou projecte amb Maven i Netbeans és molt senzill, només heu d'anar a *File / New Project / Maven / Web Application* (vegeu la figura figura 1.2).

**FIGURA 1.2.** Creació d'una aplicació web utilitzant Maven



Podeu posar com a nom de projecte *Hola, Món*, i com a identificador de grup *cat.xtec.ioc*, tal com podeu veure a la figura figura 1.3

**FIGURA 1.3.** Propietats de Maven

Finalment, a l'última pantalla escollireu **GlassFish** com a servidor d'aplicacions.

Un projecte Maven es defineix mitjançant un fitxer anomenat **POM** (*Project Object Model*). Aquest fitxer s'utilitza per definir les instruccions per compilar el projecte, les dependències del projecte (llibreria), etc. En Maven, l'execució d'un fitxer POM sempre genera un **artefacte**. Aquest artefacte pot ser qualsevol cosa: un fitxer *jar*, un fitxer *swf* de Flash, un fitxer *zip* o el mateix fitxer *pom*.

Un exemple de fitxer POM podeu trobar-lo a la carpeta *ProjectFiles*:

```

1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org
2 /2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM
3 /4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4 <modelVersion>4.0.0</modelVersion>
5
6 <groupId>cat.xtec.ioc</groupId>
7 <artifactId>holamon</artifactId>
8 <version>1.0</version>
9 <packaging>war</packaging>
10
11 <name>holamon</name>
12
13 <properties>
14 <endorsed.dir>${project.build.directory}/endorsed</endorsed.dir>
15 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
16 </properties>
17
18 <dependencies>
19 <dependency>
20 <groupId>javax</groupId>
21 <artifactId>javaee-web-api</artifactId>
22 <version>7.0</version>
23 <scope>provided</scope>
24 </dependency>
25 </dependencies>
26
27 <build>
28 <plugins>
29 <plugin>
30 <groupId>org.apache.maven.plugins</groupId>
31 <artifactId>maven-compiler-plugin</artifactId>
32 <version>3.1</version>

```

```
31         <configuration>
32             <source>1.7</source>
33             <target>1.7</target>
34             <compilerArguments>
35                 <endorseddirs>${endorsed.dir}</endorseddirs>
36             </compilerArguments>
37         </configuration>
38     </plugin>
39     <plugin>
40         <groupId>org.apache.maven.plugins</groupId>
41         <artifactId>maven-war-plugin</artifactId>
42         <version>2.3</version>
43         <configuration>
44             <failOnMissingWebXml>>false</failOnMissingWebXml>
45         </configuration>
46     </plugin>
47     <plugin>
48         <groupId>org.apache.maven.plugins</groupId>
49         <artifactId>maven-dependency-plugin</artifactId>
50         <version>2.6</version>
51         <executions>
52             <execution>
53                 <phase>validate</phase>
54                 <goals>
55                     <goal>copy</goal>
56                 </goals>
57                 <configuration>
58                     <outputDirectory>${endorsed.dir}</outputDirectory>
59                     <silent>>true</silent>
60                     <artifactItems>
61                         <artifactItem>
62                             <groupId>javax</groupId>
63                             <artifactId>javaee-endorsed-api</artifactId>
64                             <version>7.0</version>
65                             <type>jar</type>
66                         </artifactItem>
67                     </artifactItems>
68                 </configuration>
69             </execution>
70         </executions>
71     </plugin>
72 </plugins>
73 </build>
74 </project>
```

Les etiquetes més importants del fitxer POM són:

- **groupId**: és com el paquet del projecte. Normalment es posa el nom de l'empresa, ja que tots els projectes amb un *groupId* pertanyen a una sola empresa.
- **artifactId**: és el nom del projecte.
- **version**: nombre de versió del projecte.
- **packaging**: tipus de fitxer generat en compilar el projecte Maven.

### 1.1.2 Exemple 'Hola, Món'

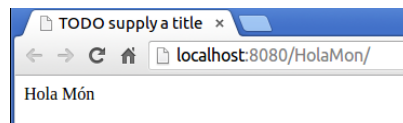
Una vegada s'ha creat el projecte, veieu que Maven genera un fitxer anomenat `index.html` dintre de la carpeta *Web Pages*, semblant a aquest:

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Start Page</title>
5     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
6   </head>
7   <body>
8     <h1>Hello World!</h1>
9   </body>
10 </html>
```

Noteu que heu canviat el text “Hello World!” per “Hola, Món”. Aquest últim apareixerà com a contingut de la pàgina HTML.

Quan executeu el projecte anterior apareix en el navegador l'HTML anterior (vegeu la figura figura 1.4).

**FIGURA 1.4.** Execució del projecte 'Hola, Món'



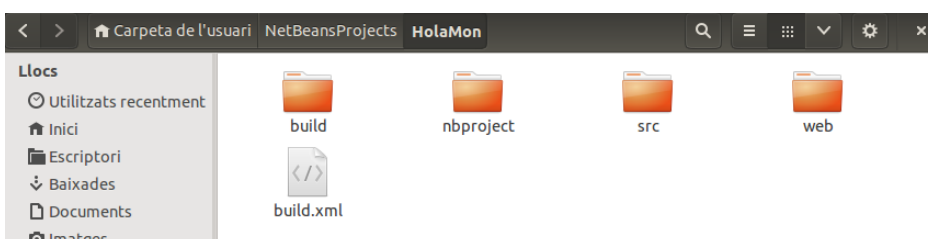
Fixeu-vos en la URL de la pàgina que ha executat el navegador: <http://localhost:8080/HolaMon/>.

Intenteu contestar a les següents preguntes:

- Què és `http://localhost:8080`?
- I *Hola, Món* què significa, a què fa referència?

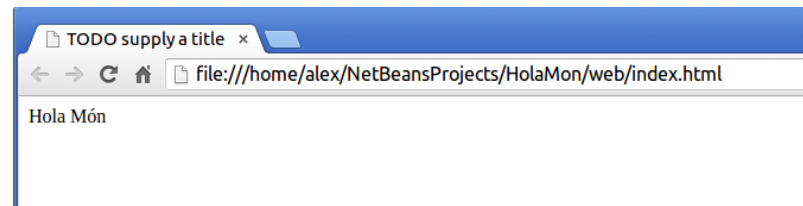
Abans de saber la resposta de les preguntes anteriors, cerqueu al vostre PC el fitxer `index.html` que heu creat anteriorment. Feu la cerca amb l'explorador de fitxers. Hauríeu de trobar-lo dintre de la carpeta *NetBeansProject/HolaMon/web* (vegeu la figura figura 1.5).

**FIGURA 1.5.** Fitxers del projecte 'Hola, Món'



Si obriu amb un navegador web el fitxer `index.html` es veurà correctament? Proveu-ho (vegeu la figura figura 1.6).

**FIGURA 1.6.** Execució del fitxer `index.html` directament al navegador



Observeu que el fitxer s'ha visualitzat exactament igual. No hi ha cap dubte que és la mateixa pàgina web. L'únic que ha canviat respecte a l'execució anterior és la URL del navegador. En aquest cas, es mostra la ruta des de l'arrel del sistema fins al fitxer que es veu en el navegador:

```
1 file:///home/alex/NetBeansProjects/HolaMon/web/index.html
```

Una pàgina **web estàtica** és aquella pàgina enfocada principalment a mostrar una informació permanent, on el navegant es limita a obtenir aquesta informació.

Llavors, quina és la diferència entre aquesta execució i l'execució de la figura figura 1.4?

Aquesta vegada el navegador ha llegit directament el fitxer, per això a la URL apareix la seva localització exacta. No hi ha cap altra intervenció. Ha obert el fitxer, ho ha interpretat (HTML) i l'ha tancat.

Per explicar que succeeix a la figura figura 1.4 ha arribat el moment de contestar a les preguntes abans formulades:

- Què és `http://localhost:8080`?
- I *Hola, Món* què significa, a què fa referència?

Com podeu suposar, el navegador no està obrint cap fitxer. El navegador, mitjançant el protocol HTTP, ha demanat un recurs a un servidor web. Aquest recurs és un fitxer HTML que ha llegit el servidor web i l'ha enviat al navegador a través de la xarxa fent servir el protocol HTTP.

#### Tecnologies al servidor

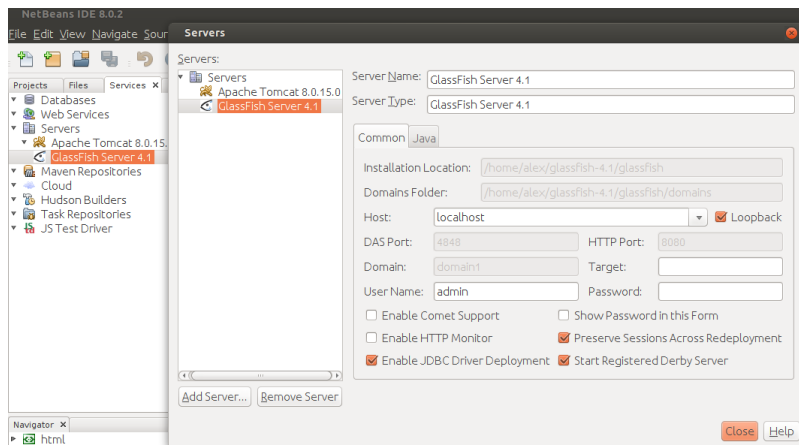
Es poden utilitzar diverses tecnologies per augmentar la potència del servidor més enllà de la seva capacitat de lliurar pàgines HTML; aquestes inclouen *scripts*, CGI, PHP, ASP, Java Servlets...

Un **servidor web** serveix contingut estàtic a un navegador, carrega un arxiu i el serveix a través de la xarxa al navegador d'un usuari. Aquest intercanvi d'informació entre el navegador i el servidor es produeix perquè parlen l'un amb l'altre mitjançant HTTP.

I el paràmetre **:8080**? Aquest paràmetre fa referència al port d'escolta del servidor web. Normalment, el paràmetre no el veieu quan accediu a una pàgina web

que està allotjada en un servidor públic, perquè s'accedeix a un port d'escolta estàndard (el 80 per a pàgines no segures o el 443 per a pàgines segures). En aquest cas, l'entorn de desenvolupament NetBeans us proporciona un servidor integrat anomenat **GlassFish** que escolta les peticions pel port 8080. A més a més, com que està funcionant en el mateix PC des d'on s'està executant NetBeans, el nom del servidor és el mateix PC, o en altres paraules, *localhost*. Podeu veure les propietats si aneu a la finestra *Services* i després a les propietats del servidor (vegeu la figura figura 1.7).

**FIGURA 1.7.** Propietats del servidor GlassFish

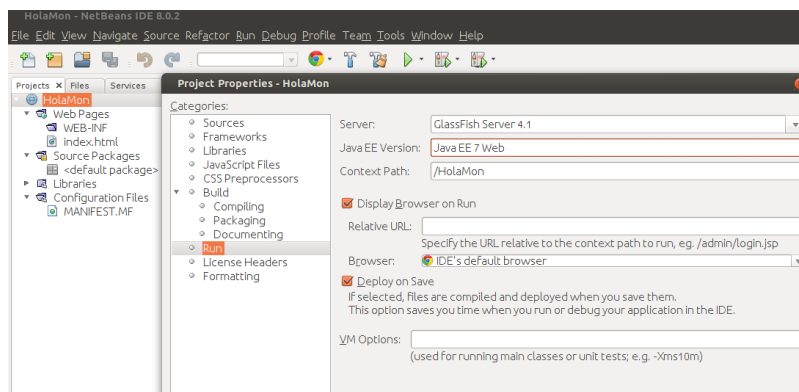


Si us hi fixeu, hi ha un paràmetre del servidor anomenat *HTTP Port* on s'especifica el port utilitzat per escoltar peticions HTTP, i un altre anomenat *HOST* on s'especifica que el servidor és el mateix PC (*localhost*). Recordeu que quan vàreu crear el projecte *Hola, Món* vàreu configurar-lo perquè utilitzés el servidor web GlassFish, que ve integrat amb NetBeans.

I *Hola, Món* què significa, a què fa referència?

*Hola, Món* identifica, dintre del servidor GlassFish, el **recurs web** que es vol visualitzar. Penseu que GlassFish pot publicar molts projectes alhora i que ha de tenir una manera per identificar cadascun. Una vegada ha estat identificat el projecte, GlassFish mostra la pàgina principal (*index.html*). Si veieu la figura figura 1.8 comprovareu on s'especifica el nom d'aquest recurs (*context path*).

**FIGURA 1.8.** Propietats del projecte 'Hola, Món'



Però perquè mostra la pàgina *index.html*? Per defecte, els servidors web cerquen aquest fitxer, però podeu canviar aquest comportament creant un arxiu **web.xml** dintre de la carpeta *WEB-INF*. Per crear-lo podeu anar a *New / Other / Standard Deployment Descriptor*, i aquí li podeu dir un altre arxiu d'inici de l'aplicació amb, per exemple:

```
1 <welcome-file-list>
2   <welcome-file>pagina_principal.html</welcome-file>
3 </welcome-file-list>
```

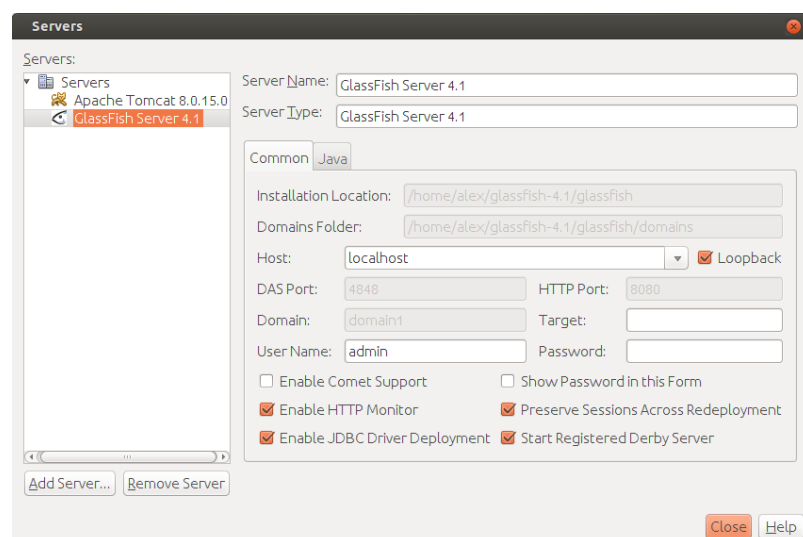
HTTP disposa d'una variant xifrada mitjançant SSL anomenada HTTPS.

Falta veure com es comunica el navegador amb el servidor web. Com hem dit abans, aquesta comunicació es fa mitjançant el protocol HTTP.

El protocol de transferència d'hipertext o **HTTP** (*HyperText Transfer Protocol*) estableix el protocol per a l'intercanvi de documents d'hipertext i multimèdia al web.

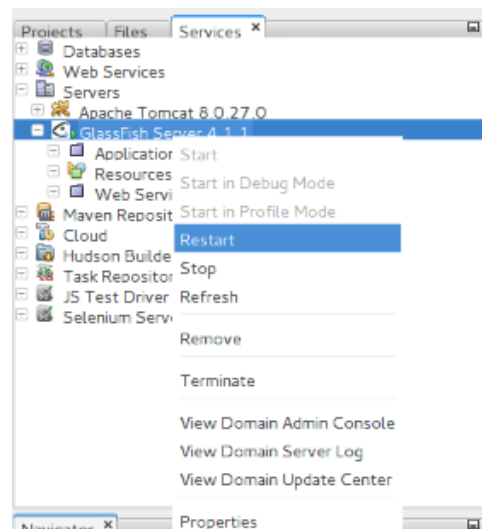
Molt bé, ara activareu la monitorització d'aquest protocol amb el servidor Glassfish. Accediu a les propietats del servidor i activeu la monitorització (vegeu la figura figura 1.9).

**FIGURA 1.9.** Activació del monitoratge HTTP

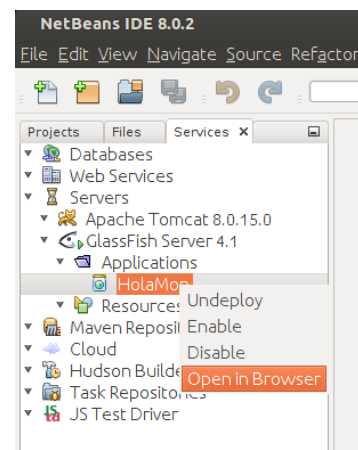
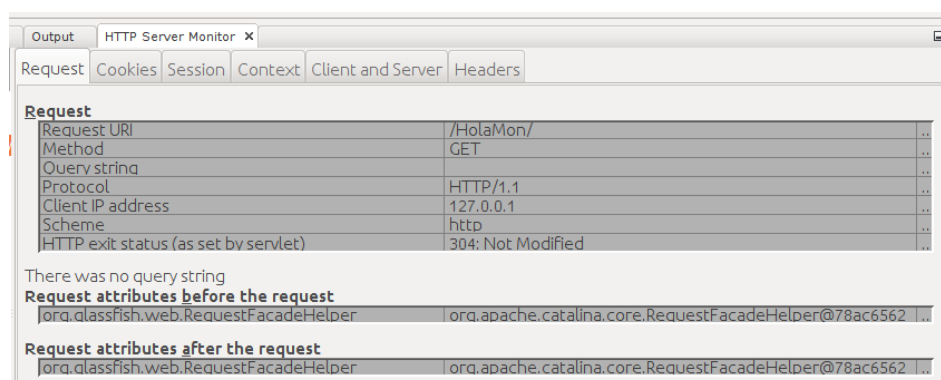


En activar el monitoratge HTTP s'ha de reiniciar el servidor, i finalment s'ha de compilar el projecte un altre cop (vegeu la figura figura 1.10).



**FIGURA 1.10.** Reiniciar el servidor web

Una vegada heu compilat el projecte podeu obrir-lo al navegador i us apareixerà una finestra al Netbeans amb els missatges HTTP intercanviats (vegeu les figures figura 1.11 i figura 1.12).

**FIGURA 1.11.** Obrir el recurs web en el navegador**FIGURA 1.12.** Missatge HTTP Get

El propòsit del protocol HTTP és permetre la transferència d'arxius (principalment, en format HTML) entre un navegador (el client) i un servidor web. La comunicació entre el navegador i el servidor es duu a terme en dues etapes:

- El navegador realitza una sol·licitud HTTP.
- El servidor processa la sol·licitud i després envia una resposta HTTP.

Una **sol·licitud HTTP** és un conjunt de línies que el navegador envia al servidor. Inclou:

- Una línia de sol·licitud: és una línia que especifica el tipus de document sol·licitat, el mètode que s'aplicarà i la versió del protocol utilitzada. La línia està formada per tres elements que han d'estar separats per un espai:
  - el mètode
  - l'adreça URL
  - la versió del protocol utilitzada pel client (en general, HTTP/1.1)
- Els camps de l'encapçalat de sol·licitud: és un conjunt de línies opcionals que permeten aportar informació addicional sobre la sol·licitud i/o el client (navegador, sistema operatiu, etc.). Cadascuna d'aquestes línies està formada per un nom que descriu el tipus d'encapçalat, seguit de dos punts (:) i el valor de l'encapçalat.
- El cos de la sol·licitud: és un conjunt de línies opcionals que han d'estar separades de les línies precedents per una línia en blanc i, per exemple, permeten que s'enviïn dades per una comanda POST durant la transmissió de dades al servidor utilitzant un formulari.

Per tant, una sol·licitud HTTP posseeix la següent sintaxi:

```
1 GET http://ioc.xtec.cat HTTP/1.1
2 Accept : Text/html
3 If-Modified-Since : Friday, 11-December-2015 14:37:11 GMT
4 User-Agent : Mozilla/4.0 (compatible; MSIE 5.0; Windows 95)
```

Les comandes HTTP més importants per fer una sol·licitud són:

- **GET**: s'utilitza per recollir qualsevol tipus d'informació del servidor. S'empra sempre que es prem sobre un enllaç o es tecleja directament a una URL. Com a resultat, el servidor HTTP envia el document corresponent a la URL seleccionada.
- **HEAD**: sol·licita informació sobre un objecte (fitxer): grandària, tipus, data de modificació... És utilitzat pels gestors de *cache* (memòria cau) de pàgines o els servidors *proxy* per conèixer quan és necessari actualitzar la còpia que es manté d'un fitxer.
- **POST**: serveix per enviar informació al servidor, per exemple les dades contingudes en un formulari. El servidor passarà aquesta informació a un procés encarregat del seu tractament (generalment una aplicació CGI/PHP/ASP...). L'operació que es realitza amb la informació proporcionada depèn de la URL utilitzada. S'utilitza sobretot en els formularis.

Una **resposta HTTP** és un conjunt de línies que el servidor envia al navegador. Està constituïda per:

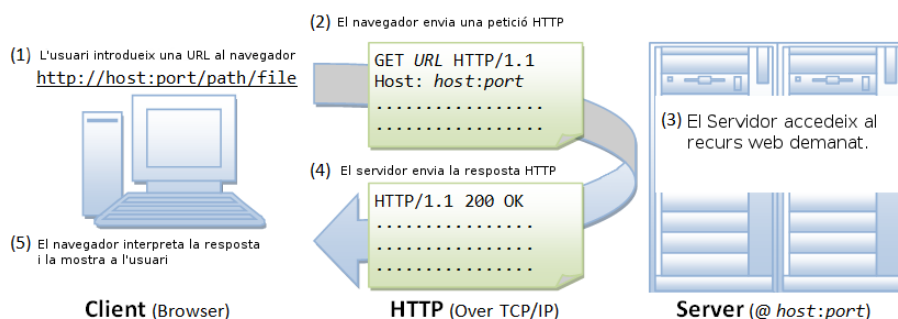
- Una línia d'estat: és una línia que especifica la versió del protocol utilitzada i l'estat de la sol·licitud en procés mitjançant un text explicatiu i un codi. La línia està composta de tres elements que han d'estar separats per un espai:
  - la versió del protocol utilitzada
  - el codi d'estat
  - el significat del codi
- Els camps de l'encapçalat de resposta: és un conjunt de línies opcionals que permeten aportar informació addicional sobre la resposta i/o el servidor. Cadascuna d'aquestes línies està composta d'un nom que qualifica el tipus d'encapçalat, seguit de dos punts (:) i del valor de l'encapçalat.
- El cos de la resposta: conté el document sol·licitat.

Exemple de resposta d'una petició GET a un servidor web:

```
1 HTTP/1.1 200 OK
2 Server: Microsoft-IIS/5.0\r\n
3 Content-Location: http://www.microsoft.com/default.htm\r\n
4 Date: Fri, 10 Dec 2015 19:33:18 GMT\r\n
5 Content-Type: text/html\r\n
6 Accept-Ranges: bytes\r\n
7 Last-Modified: Thu, 9 Dec 2015 20:27:23 GMT\r\n
8 Content-Length: 26812\r\n
9 <html>
10 ....//pàgina html que es veurà en el navegador
11 </html>
```

La manera de funcionar que des d'un navegador us permet accedir a la informació que existeix en un servidor es diu **arquitectura client-servidor**.

**FIGURA 1.13.** Arquitectura client-servidor



Penseu diverses aplicacions que utilitzeu diàriament i fan servir l'arquitectura client-servidor.

A les aplicacions client-servidor, al client se'l coneix amb el terme *front-end* o interfície d'usuari. La part client té les següents responsabilitats:

- Interaccionar amb l'usuari.
- Manipular les dades que introdueix l'usuari.
- Enviar les peticions del usuari al servidor.
- Rebre el resultat del processament de les dades enviades al servidor.
- Interpretar la resposta del servidor i mostrar-la a l'usuari.

En canvi, el servidor, també anomenat *back-end*, porta a terme les següents funcions:

- Processar la lògica del programa necessària per donar resposta a la petició del client.
- Realitzar les validacions necessàries a la base de dades.
- Accedir a la base de dades, si fos necessari, per donar una solució als requeriments de la petició enviada pel client.
- Formatar les dades per enviar-les al client.

Per donar a l'usuari una experiència molt més agradable i dinàmica a l'hora d'utilitzar aplicacions client-servidor s'han desenvolupat diferents tecnologies que s'executen o bé al *front-end* o bé al *back-end*.

Proveu el següent codi i digueu si s'executa al *front-end* o al *back-end*:

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>TODO supply a title</title>
5     <meta charset="UTF-8">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7     <script src="exemple.js" type="text/javascript"></script>
8     <link href="estils.css" rel="stylesheet" type="text/css"/>
9   </head>
10  <body>
11    <div>HolaMón</div>
12    <button onclick="hola();" value="salutació">salutació</button>
13  </body>
14 </html>
```

Com veieu, en l'HTML anterior utilitzeu dos fitxers addicionals: *estils.css* i *exemple.js*. A continuació teniu el contingut del fitxer *exemple.js*:

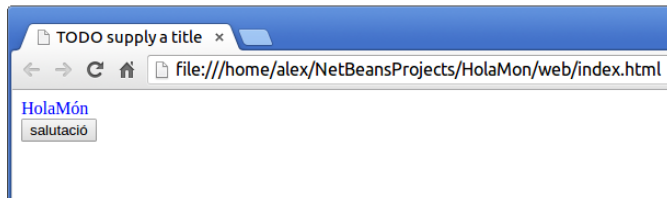
```
1 function hola(){
2   alert("hola");
3 }
```

Aquest és el contingut del fitxer “*estils.css*”:

```
1 div{
2   color:blue;
3 }
```

Una manera que teniu per determinar si el codi Javascript i el codi CSS s'executen al servidor o al navegador és executar el codi sense la intervenció del servidor. Igual que heu fet anteriorment, executeu el codi directament fent doble clic al fitxer. La URL que veureu en el navegador és la localització exacta del fitxer en el disc dur (vegeu la figura figura 1.14).

**FIGURA 1.14.** Execució del projecte directament al navegador



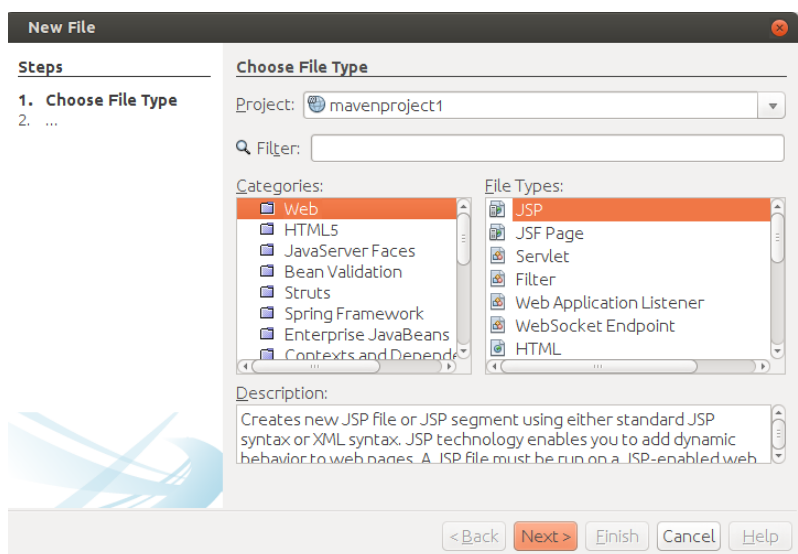
En aquest cas, com podeu comprovar, funciona tot exactament igual que abans. Significa que tant el CSS com el programa en Javascript són tecnologies que s'executen exclusivament en el client. No cal un servidor web perquè funcionin.

Intenteu descobrir quines de les següents tecnologies s'executen al client o al servidor: Java, PHP, CGI, FLASH, JSF, *applets*, CSS, ASP i Javascript.

### 1.1.3 Pàgines dinàmiques i servidors d'aplicacions

Creareu una pàgina nova, però en comptes de ser de tipus HTML la creareu de tipus *JSP* (Java Server Pages) i l'anomenareu `index.jsp`. La creareu utilitzant el mateix projecte de l'apartat anterior mitjançant l'opció *File / New File / Web / JSP* del menú principal (vegeu la figura figura 1.15).

**FIGURA 1.15.** Nova pàgina JSP



Modifiqueu-la amb el següent codi i digueu si s'executa al *front-end* o al *back-end*:

```
1 <%@page contentType="text/html" pageEncoding="UTF-8"%>
2 <!DOCTYPE html>
```

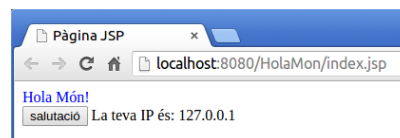
```

3 <html>
4   <head>
5     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
6     <title>Pàgina JSP</title>
7     <script src="exemple.js" type="text/javascript"></script>
8     <link href="estils.css" rel="stylesheet" type="text/css"/>
9   </head>
10  <body>
11    <div>Hola, Món!</div>
12    <button onclick="hola();" value="salutació">salutació</button>
13    <%
14      out.println("La teva IP és: " + request.getRemoteAddr());
15    %>
16  </body>
17 </html>

```

Si l'executeu des del programa NetBeans fent servir el servidor web veieu que s'executa tot el codi correctament, fins i tot el codi que hi ha dintre de les etiquetes ”<%” (vegeu la figura 1.16).

**FIGURA 1.16.** Execució de la pàgina index.jsp utilitzant el servidor GlassFish



Però s'ha executat tot en el *front-end* o hi ha algun tros de codi que s'ha executat en el *back-end*?

En comptes d'anar directament al fitxer i fer doble clic, us proposo que mireu el codi font HTML que ha interpretat el navegador. Podeu veure-ho fent clic amb el botó esquerre i després seleccionant del menú l'opció *Visualitza l'origen de la pàgina*, sempre que el ratolí estigui posicionat dintre de la finestra del navegador.

```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
5     <title>Pàgina JSP</title>
6     <script src="exemple.js" type="text/javascript"></script>
7     <link href="estils.css" rel="stylesheet" type="text/css"/>
8   </head>
9   <body>
10    <div>Hola, Món!</div>
11    <button onclick="hola();" value="salutació">salutació</button>
12    La teva IP és: 127.0.0.1
13
14  </body>
15 </html>

```

Noteu que tot el codi que hi havia entre les etiquetes ”<%” i ”%>” no apareix. En el seu lloc apareix el resultat de la seva execució. Quan el servidor ha volgut respondre a la petició *HTTP GET* del client ha hagut d'executar el codi que hi ha dintre de les etiquetes ”<%” i ha enviat un missatge *HTTP Response* amb el codi HTML substituint les etiquetes ”<%” pel resultat de l'execució.

És a dir, el navegador no ha tingut l'oportunitat de veure que s'havia d'executar codi perquè ni tan sols l'ha rebut. De fet, el servidor web ha fet alguna cosa més del

que feia fins ara. El servidor web s'ha transformat en un **servidor d'aplicacions** on ha hagut de transformar la pàgina web original.

Un **servidor d'aplicacions** és un servidor web i un *framework* de programari on es poden executar aplicacions. És a dir, és un servidor web que permet l'execució d'un programa en el mateix servidor amb les dades proporcionades per una aplicació client.

Com que el servidor ha executat codi JSP, la pàgina HTML ha canviat (l'encerclat per "<% %>"). Qualsevol usuari que es connecti amb el navegador veurà una pàgina diferent. No totalment diferent però sí parcialment, perquè cadascú visualitzarà la seva IP quan accedeixi a aquesta pàgina.

### Servidor d'aplicacions Tomcat

NetBeans disposa, a més del servidor GlassFish, del servidor d'aplicacions Tomcat. **Tomcat** és un servidor d'aplicacions d'Apache Software Foundation que executa *servlets* Java i mostra pàgines web que inclouen la codificació Java Server Page. Tomcat és *open source* i està disponible en el lloc web d'Apache.

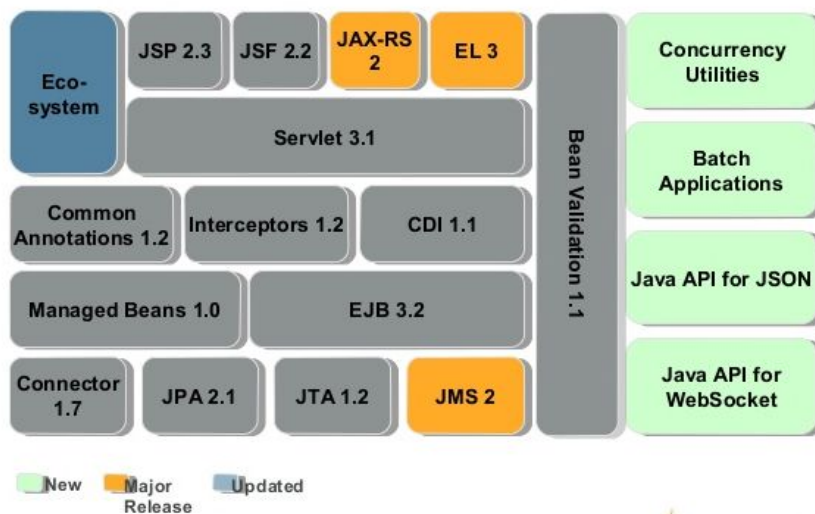
Tomcat implementa un contenidor de *servlets* i JSP, que és el que la majoria de les aplicacions web necessiten, però no és un motor de Java EE (Java Enterprise Edition). La seva instal·lació i configuració és molt fàcil, i sovint es pot fer en 10-20 minuts. En ser Tomcat simple i fàcil d'usar, no té diverses característiques importants del Java Enterprise.

Tot i que NetBeans inclou el servidor Tomcat, podeu veure com instal·lar-lo en una màquina Ubuntu en aquest enllaç: <https://help.ubuntu.com/its/serverguide/tomcat.html>.

Tomcat és només un contenidor de *servlets*, és a dir, que només implementa l'especificació de *servlets* i JSP. En canvi, **Glassfish** és un servidor de Java EE complet (interpretant tecnologies com EJB, JMS...). Amb Glassfish teniu la implementació de tota la pila Java EE (Java EE stack, vegeu la figura 1.17).

FIGURA 1.17. Pila de components Java 7 EE

## Java EE 7



Proveu el següent exemple:

```
1 <%@page contentType="text/html" pageEncoding="UTF-8"%>
2 <!DOCTYPE html>
3 <html>
4   <head>
5     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
6     <title>Pàgina JSP</title>
7     <script src="exemple.js" type="text/javascript"></script>
8     <link href="estils.css" rel="stylesheet" type="text/css"/>
9   </head>
10  <body>
11    <div>Hola, Món!</div>
12    <button onClick="hola();" value="salutació">salutació</button>
13    <p>Avui és <%= new java.util.Date() %> </p>
14  </body>
15 </html>
```

Cada vegada que actualitzeu la pàgina anterior, us retorna la data i l'hora actualitzada. El servidor Glassfish està canviant la pàgina cada vegada que la demanem.

Una **web dinàmica** és aquella que conté aplicacions dins de la mateixa web, atorgant major interactivitat amb el navegant.

Exemples d'aplicacions dinàmiques són enquestes i votacions, fòrums de suport, llibres de visita, enviament de correus electrònics intel·ligents, reserva de productes, comandes *on line*, atenció al client personalitzada...

És important no confondre multimèdia i interactivitat amb pàgines dinàmiques. Una pàgina web estàtica pot ser multimèdia (contenir diversos tipus de vídeos, sons, imatges...) i interactiva a través d'enllaços i hipervincles, sense ser dinàmica per aquestes característiques.

A les pàgines dinàmiques, el contingut sol generar-se al moment de visualitzar-se, poden variar, mentre que en les estàtiques el contingut sol estar predeterminat.

L'important d'aquesta classificació entre dinàmiques i estàtiques és que una pàgina web estàtica la podeu emmagatzemar fàcilment, mentre que en una dinàmica no serà així.

#### 1.1.4 Què s'ha après?

Fins a aquest moment heu après a instal·lar l'entorn de desenvolupament NetBeans. Heu creat el vostre primer projecte amb Maven i heu après el funcionament de la tecnologia de comunicacions client-servidor.

També heu de ser capaços de diferenciar entre un servidor web i un servidor d'aplicacions. S'ha fet una primera aproximació al llenguatge JSP i s'ha explicat la diferència entre una pàgina estàtica i una altra de dinàmica.



Per tal de ser més àgils i autònoms a l'hora de programar, és recomanable fer les activitats d'ampliació on s'explica com depurar el codi des del mateix IDE i com veure la comunicació client-servidor amb eines instal·lades en el navegador web.

Resumint, en aquest apartat s'ha après a:

- Instal·lar i configurar l'entorn de desenvolupament NetBeans.
- Crear un projecte amb Maven.
- Comprendre la comunicació client-servidor, és a dir, el protocol HTTP.
- Configurar el servidor Glassfish.
- Diferenciar una pàgina estàtica d'una dinàmica.

Arribats a aquest punt, hauríeu de ser autònoms a l'hora d'utilitzar l'entorn de desenvolupament NetBeans, així com saber preparar un projecte creat amb Maven. Hauríeu de ser capaços de reconèixer els avantatges de la programació de pàgines web dinàmiques en contraposició a les estàtiques. Finalment, hauríeu de ser capaços d'entendre la comunicació client-servidor amb el protocol HTTP.

## 1.2 El vostre primer projecte PHP: 'Hola, Món'

Tot i que en aquest mòdul utilitzarà el llenguatge Java com a llenguatge vehicular al llarg de les diferents unitats, és interessant que vegeu algunes pinzellades del llenguatge PHP (*PHP Hypertext Preprocessor*), ja que és àmpliament utilitzat en el món empresarial.

En aquest apartat veureu com crear un projecte web PHP amb NetBeans, com instal·lar el programari necessari per posar a punt un servidor PHP i com crear una pàgina molt senzilla.

**PHP** és un llenguatge de codi obert molt popular, especialment adient per al desenvolupament web, que pot ser afegit directament al codi HTML.

Utilitzareu una màquina virtual amb Ubuntu Server per emprar-la com a servidor web. En aquesta màquina instal·lareu el servidor PHP i utilitzareu el programa Netbeans per crear pàgines web directament al servidor.

### 1.2.1 Posar a punt el servidor web

L'IDE NetBeans no té un servidor web PHP instal·lat, heu de crear-vos nosaltres un servidor a part. Teniu dues opcions: la primera és instal·lar en la mateixa

màquina on teniu l'IDE el mateix servidor web (amb un XAMPP, per exemple), i la segona és utilitzar una màquina virtual on instal·lar el servidor i fer que l'IDE NetBeans treballi directament amb aquesta.

És molt més real fer-ho de la segona manera. Instal·lareu pas a pas un servidor PHP a una màquina virtual i hi connectareu l'IDE.

### Instal·lació del sistema operatiu

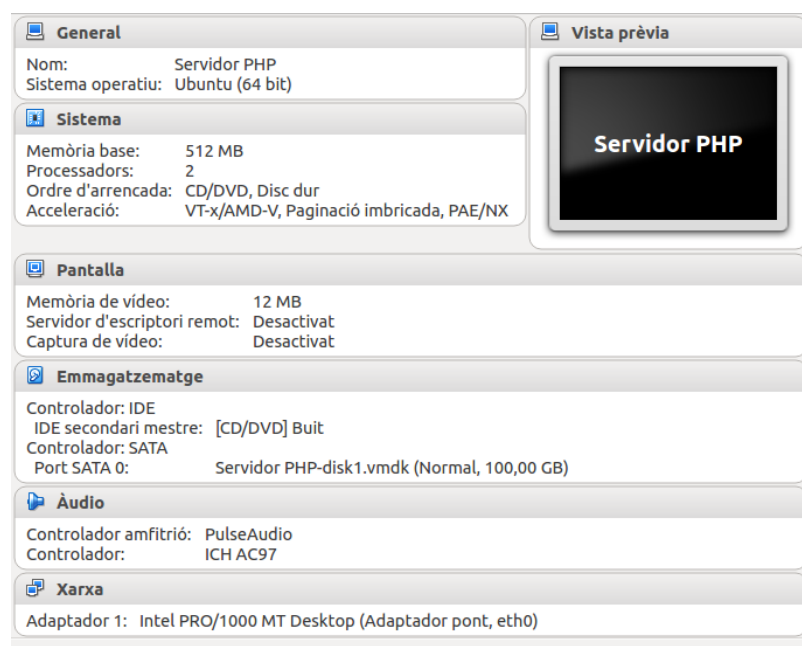
En aquest apartat suposeu que teniu instal·lat el programa de màquines virtuals **VirtualBox** i sabeu com crear una màquina nova i com instal·lar un sistema operatiu.

Utilitzareu el servidor Ubuntu Server LTS 14.04. Podeu descarregar-lo de l'adreça: [www.ubuntu.com/download/server](http://www.ubuntu.com/download/server).

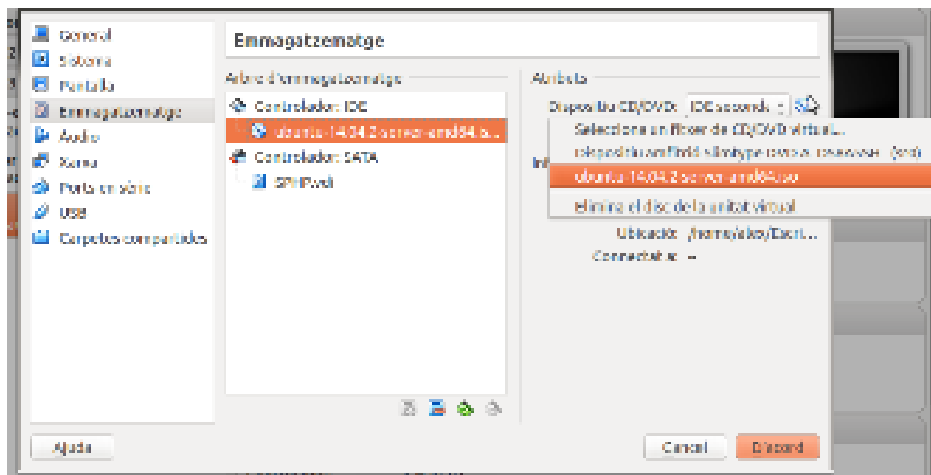
Creareu una màquina virtual nova amb Virtualbox amb les següents característiques (vegeu la figura figura 1.18):

- nom: servidor PHP
- 512 MB de RAM
- 100 GB de disc dur
- xarxa amb adaptador pont

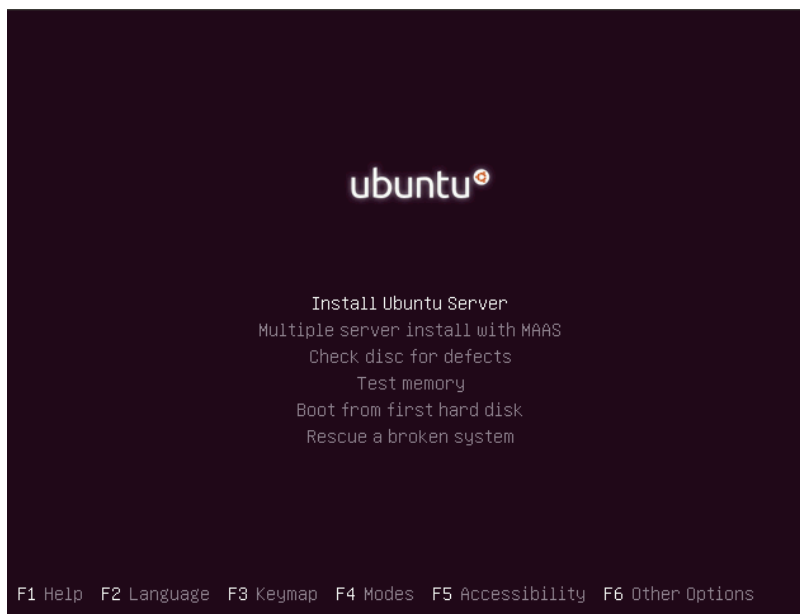
**FIGURA 1.18.** Configuració servidor PHP amb VBox



Utilitzeu la ISO que heu descarregat dels servidors d'Ubuntu per instal·lar el sistema operatiu. Poseu-la al CD virtual de la màquina “Servidor PHP” (vegeu la figura figura 1.19) i comenceu la instal·lació.

**FIGURA 1.19.** Selecció de la ISO d'Ubuntu Server al programa Virtualbox

Les característiques per instal·lar el sistema operatiu (vegeu la figura figura 1.20) són les característiques per defecte. No cal instal·lar, en aquest punt, res addicional.

**FIGURA 1.20.** Pantalla d'instal·lació d'Ubuntu Server

En finalitzar aquest punt s'ha de tenir una màquina virtual amb el sistema operatiu Ubuntu Server instal·lat.

### Instal·lació del servidor web

Una vegada teniu preparat el sistema operatiu Ubuntu Server heu de convertir-lo en un servidor web. Per fer-ho heu d'instal·lar una sèrie de paquets, i és molt més còmode si us connecteu via SSH a la màquina virtual; per tant, això és el primer que fareu: instal·lareu el servei SSH.

#### L'interpret d'ordres segur SSH

SSH (acrònim de *Secure Shell*, 'interpret d'ordres segur') és un protocol que permet accedir de manera segura a un ordinador des d'un altre. Permet administrar totalment l'ordinador remot sempre i quan l'usuari tingui els permisos per fer-ho.

```
1 sudo apt-get update
2 sudo apt-get install openssh-server
```

A continuació us connectareu via SSH a la màquina virtual. Per poder-vos connectar heu de saber quina adreça IP té el servidor. La podeu esbrinar executant la següent comanda directament al servidor:

```
1 ifconfig
```

Una vegada sabeu l'adreça IP del servidor, podeu utilitzar un terminal de la màquina real i continuar instal·lant els paquets. Per fer la connexió SSH des de la màquina real a la màquina virtual executem:

```
1 ssh usuari_servidor_linux@ip_servidor_linux
2 exemple:
3 ssh alex@192.168.56.101
```

A continuació instal·lareu el servidor Apache2:

```
1 sudo apt-get install apache2
```

I finalment, instal·lareu el mòdul PHP:

```
1 sudo apt-get install php5
2 sudo apt-get install libapache2-mod-php5
```

Només queda configurar els permisos perquè el programa NetBeans pugui escriure a una carpeta propietària de *root*. No configurareu ara els permisos i els usuaris o els grups que tenen permisos per accedir-hi, ja que no és l'objectiu de l'assignatura. Per això, canviareu els permisos perquè tothom pugui entrar a escriure, llegir o executar. En concret, executareu la següent comanda en el terminal d'Ubuntu:

```
1 sudo chmod 777 /var/www/html
```

Una vegada s'ha complert aquest punt ja esteu preparats per crear un nou projecte PHP directament al servidor PHP. Comproveu que ha sortit tot correctament accedint, mitjançant un navegador web des de la màquina real, al servidor PHP. En el vostre cas, per exemple, la IP del servidor és 192.168.56.101 i, com podeu veure a la figura 1.21, hi podeu accedir via web.

**FIGURA 1.21.** Servidor Apache funcionant



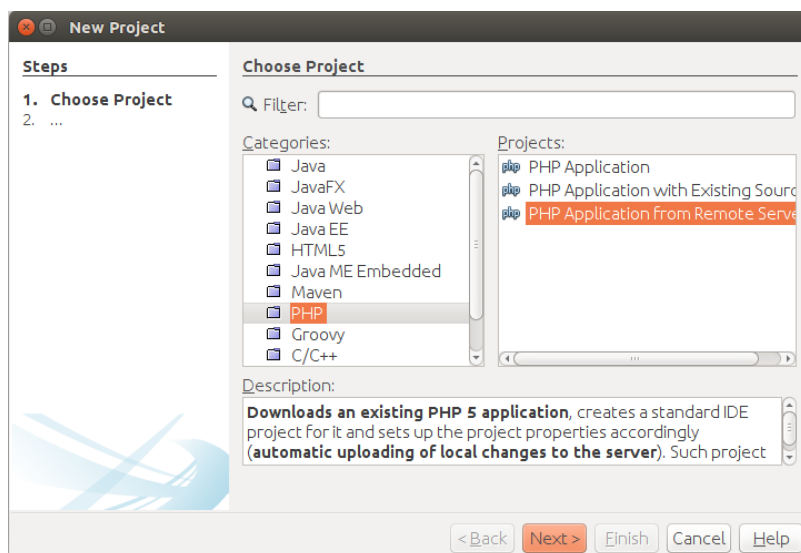
### 1.2.2 Creació d'un projecte PHP en un servidor remot

Fareu la connexió de l'IDE NetBeans amb el servidor PHP. Com podeu imaginar, funcionaria igual si el servidor tingués una adreça IP pública i vosaltres volguéssiu modificar o crear alguna pàgina PHP.

En crear un nou projecte de tipus PHP, NetBeans dóna tres opcions (vegeu la figura figura 1.22):

- aplicació PHP
- aplicació PHP amb fitxers ja creats
- aplicació PHP des d'un servidor remot

FIGURA 1.22. Nou projecte remot amb PHP

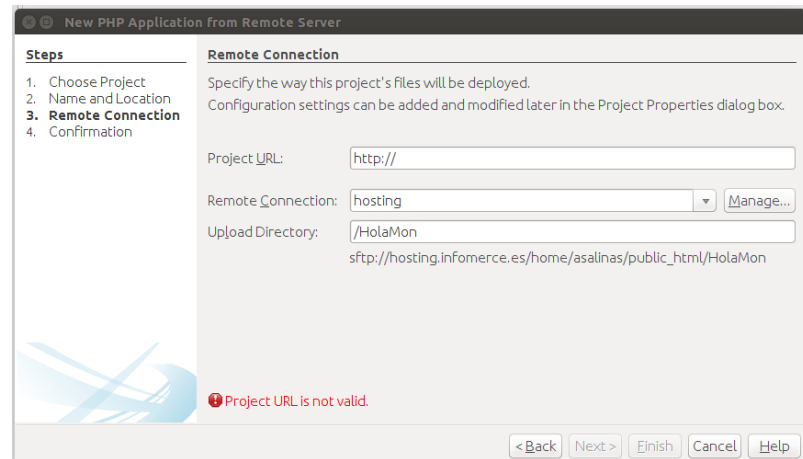


Les dues primeres opcions són vàlides quan el projecte PHP es troba en la mateixa màquina que l'IDE NetBeans. L'única diferència és si es vol crear de zero (primera opció) o ja està creat i es vol afegir o modificar alguna cosa de l'aplicació (segona opció).

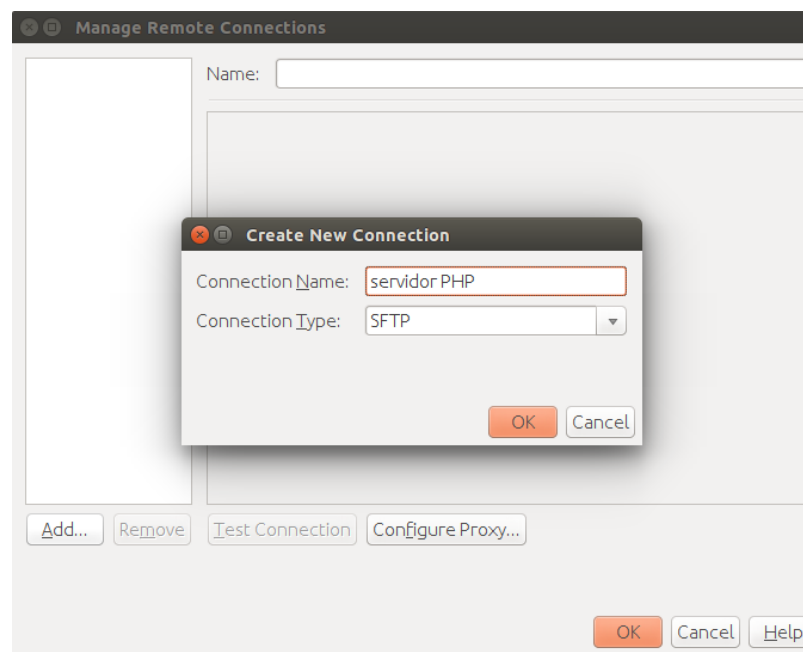
L'última opció és la que fareu servir, aplicació PHP des d'un servidor remot. En el vostre cas, el servidor remot és la màquina virtual (servidor PHP). Una vegada heu escollit aquesta opció, us demana la informació típica d'un projecte:

- nom del projecte (per exemple: *Hola, Món*)
- carpeta amb els fitxers originals
- versió PHP, etc.

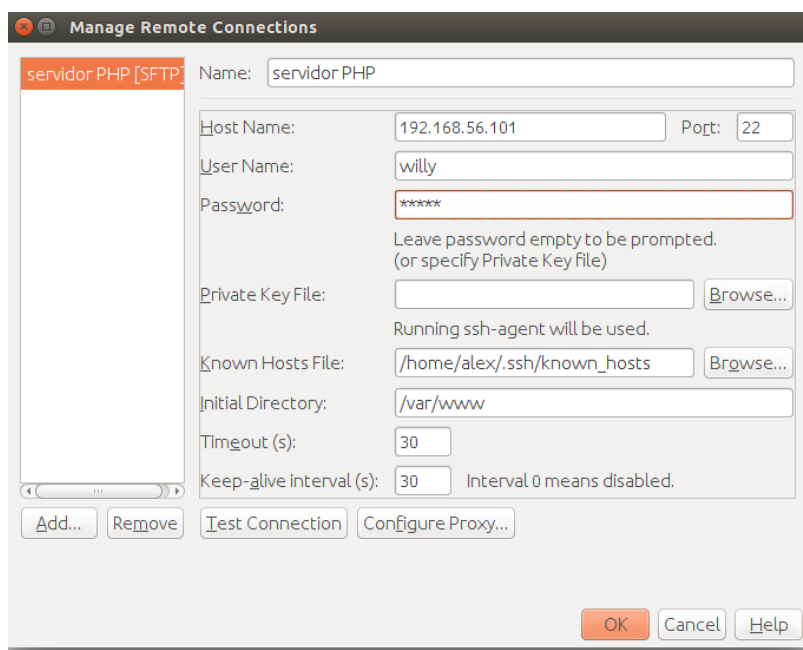
A continuació us demana la configuració per a la connexió amb el servidor remot (vegeu la figura figura 1.23). Haureu de crear primer la connexió per després configurar la carpeta del servidor que voleu utilitzar per copiar els fitxers creats.

**FIGURA 1.23.** Nova aplicació PHP des d'un servidor remot

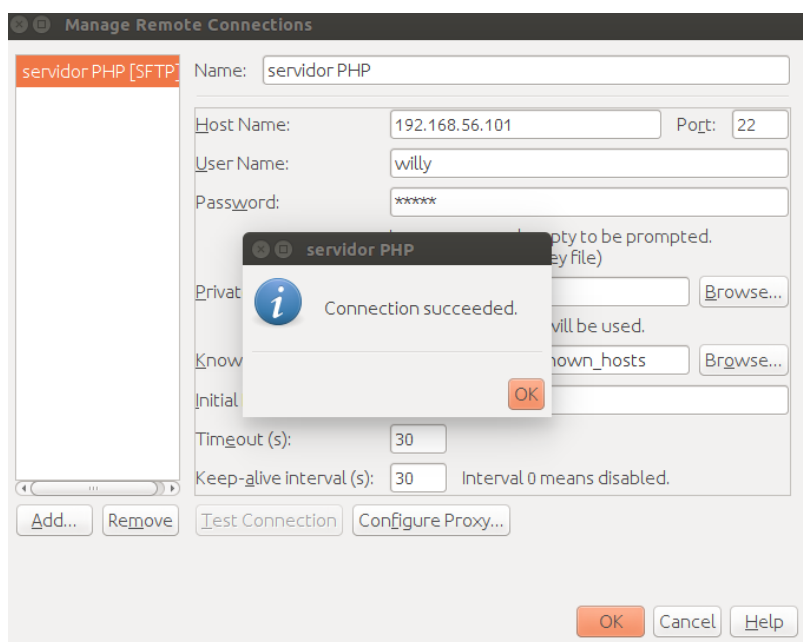
Per crear la connexió anireu a *Administrar (Manage)*. Afegireu una connexió nova de tipus SFTP i li posareu un nom per identificar la connexió, per exemple *Servidor PHP* (vegeu la figura figura 1.24).

**FIGURA 1.24.** Administrar les connexions als servidors PHP

Una vegada creada la connexió s'ha de configurar afegint la IP del servidor, així com l'usuari del servidor PHP que té permisos per escriure al disc dur. En aquest cas pot ser l'usuari que vau crear en instal·lar el sistema operatiu (vegeu la figura figura 1.25).

**FIGURA 1.25.** Configurar els paràmetres de la connexió al servidor PHP

Podeu provar la connexió clicant al botó *Test Connection*. Hauria d'aparèixer una finestra semblant a la de la figura figura 1.26.

**FIGURA 1.26.** Provar la connexió creada

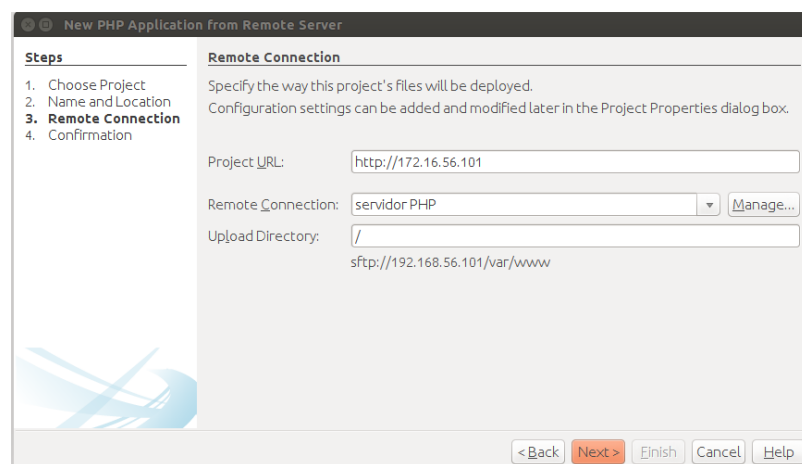
Penseu que si no heu instal·lat el servei SSH al servidor PHP no funcionarà. El servei SSH també instal·la el servei de transferència segura de fitxers SFTP, necessari per establir la connexió entre NetBeans i el servidor PHP.

Finalment, omplireu la finestra que va sortir abans de configurar la connexió amb les dades adients (vegeu la figura figura 1.27):

- URL del projecte: la IP del servidor PHP. Per exemple: <http://192.168.56.101>.

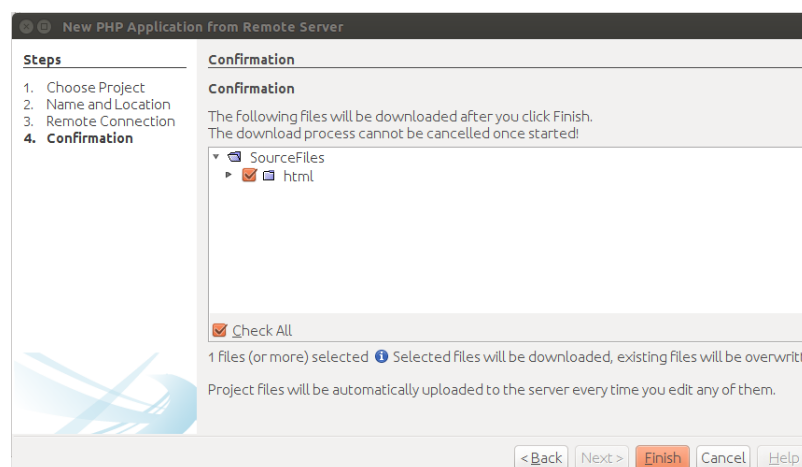
- Connexió remota: seleccioneu la connexió que acabeu de crear (servidor PHP).
- Directori de pujada: directori del servidor PHP on es pujaran els fitxers creats amb el programa NetBeans. Normalment és el mateix directori que utilitza Apache per servir els fitxers web, per exemple `/var/www`. Si poseu `"/` ja utilitza aquest directori.

**FIGURA 1.27.** Configuració dels paràmetres de la connexió al servidor PHP



Una vegada heu realitzat tots els passos anteriors, NetBeans detecta que existeix la carpeta HTML i la baixa perquè pugueu crear les pàgines PHP dintre d'aquesta (vegeu la figura figura 1.28).

**FIGURA 1.28.** Detecció de Netbeans de la carpeta HTML



Arribats a aquest punt, ja teniu el projecte preparat per començar a desenvolupar remotament una aplicació PHP.

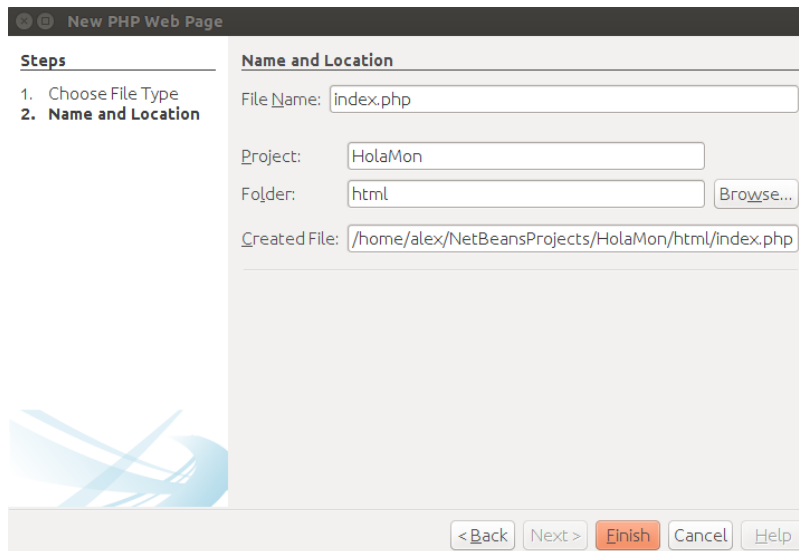


### 1.2.3 Exemple 'Hola, Món'

Creareu un nou arxiu dintre de la carpeta HTML del servidor remot. Aquest arxiu ha de ser de tipus PHP, i utilitzareu el menú de NetBeans per crear-lo. Aneu a *File / New File / PHP / PHP Web Page*.

El nom del fitxer serà `index.php` (vegeu la figura figura 1.29).

**FIGURA 1.29.** Nou fitxer PHP

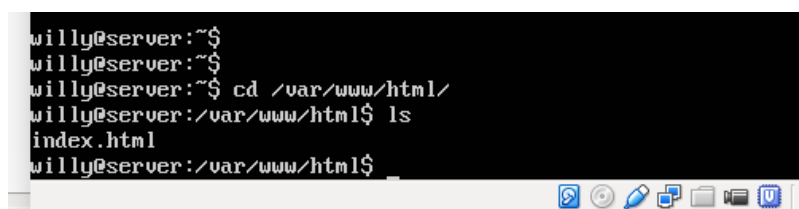


Com veieu, a l'hora de crear un fitxer primer es crea a l'ordinador local, és a dir, on està instal·lat l'entorn de desenvolupament. Si us hi fixeu, s'ha creat una còpia del projecte del servidor dintre de la carpeta de projectes locals amb el mateix nom que al servidor. Podeu saber on es troba veient la configuració del paràmetre *Created File* de la figura figura 1.29.

En tenir una còpia al vostre ordinador heu de saber gestionar correctament les versions del projecte, ja que periòdicament haureu de pujar la nova versió al servidor per veure els resultats de la modificació. Com veureu, Netbeans simplifica l'administració de les versions, permetent pujar o baixar del servidor tot el projecte o una part del mateix.

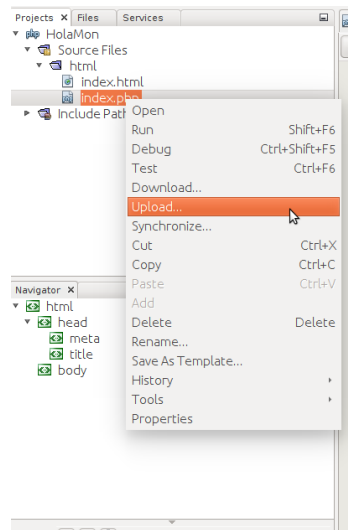
De fet, el servidor no té encara el fitxer creat. Si aneu a la màquina virtual i llisteu els fitxers que hi ha a la carpeta `/var/www/html` veureu que només existeix el fitxer `index.html` que té per defecte el servidor Apache. Podeu veure la figura figura 1.30 per comprovar-ho.

**FIGURA 1.30.** Llistat del fitxers del servidor PHP



Llavors heu de pujar aquest fitxer al servidor. Si feu clic amb el botó dret del ratolí damunt del fitxer index.php veureu un menú semblant al de la figura 1.31. Cliqueu a *Upload*.

**FIGURA 1.31.** Menú per pujar o baixar un fitxer del servidor PHP



Aquest menú permet executar:

- **Upload:** per pujar un fitxer o una carpeta al servidor.
- **Download:** per baixar una carpeta o fitxer del servidor al PC local.
- **Synchronize:** obre una finestra que us permet escollir si voleu pujar o baixar fitxers del servidor.

Intenteu crear un fitxer nou directament a la màquina virtual i baixeu-lo amb NetBeans utilitzant l'opció *Synchronize* o *Download* del menú anterior.

En aquest punt ja heu de tenir el fitxer index.php creat des de NetBeans pujat al servidor PHP. El fitxer index.php ha de ser semblant a aquest:

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="UTF-8">
5     <title></title>
6   </head>
7   <body>
8     <?php
9       // put your code here
10    ?>
11   </body>
12 </html>
```

De moment, aquest fitxer no té codi PHP. Només s'han creat dues etiquetes específiques de PHP que permeten escriure codi PHP dintre d'aquestes.

PHP és un llenguatge de programació que coexisteix dintre del llenguatge de marques HTML. L'etiqueta **<?php** és l'etiqueta d'inici d'aquest llenguatge, i

l'etiqueta `?>` és l'etiqueta de tancament. Tot el codi dintre d'aquestes etiquetes no l'interpreta el servidor web Apache2, sinó que el compila i executa el mòdul PHP que s'ha instal·lat a la màquina.

El resultat de l'execució d'aquestes instruccions és el que es mostrarà en comptes del codi que hi ha entre les etiquetes d'inici i fi del llenguatge PHP.

Per exemple, voleu escriure la frase “Hola, Món” des de PHP i que es vegi a la pàgina HTML. El codi que hauríeu d'afegir seria aquest:

```
1 echo "Hola, Món";
```

També podríeu afegir etiquetes de títol (*Heading*), per exemple:

```
1 echo "<h1>Hola, Món</h1>";
```

Com podeu observar, per enviar informació des de PHP a la pàgina HTML s'utilitza la instrucció ***echo***, que envia una cadena de text directament a la pàgina HTML en la mateixa posició on es troben les etiquetes PHP dintre de la pàgina.

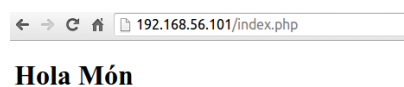
La pàgina “index.php” resultant, després d'afegir la instrucció anterior, seria:

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="UTF-8">
5     <title></title>
6   </head>
7   <body>
8     <?php
9       echo "<h1>Hola, Món</h1>";
10    ?>
11   </body>
12 </html>
```

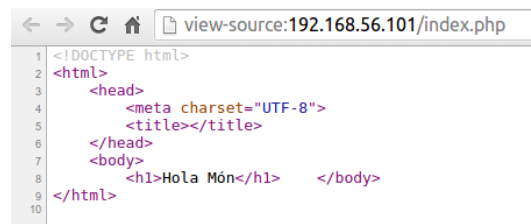
Guardeu el fitxer anterior i pugeu-lo al servidor.

Com veieu en el navegador web el fitxer index.php? Heu d'accedir a l'adreça del servidor PHP i afegir el recurs que voleu obtenir; en aquest cas, el fitxer index.php. Per exemple, en el vostre cas, en el navegador web haureu d'escriure: [192.168.56.101/index.php](http://192.168.56.101/index.php) (vegeu la figura figura 1.32).

**FIGURA 1.32.** Execució de la pàgina index.php



Ara fixeu-vos en el codi font que us mostra el navegador web (vegeu la figura figura 1.33). Veieu que no hi ha cap rastre que indiqui que aquest és un fitxer PHP, no apareixen les etiquetes pròpies d'aquest llenguatge. En el seu lloc, apareix el text que heu posat a la instrucció ***echo***.

**FIGURA 1.33.** Codi font de la pàgina index.php

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="UTF-8">
5     <title></title>
6   </head>
7   <body>
8     <h1>Hola Món</h1>
9   </body>
10 </html>
```

Observeu que en comunicar-se el navegador amb el servidor PHP aquest no solament ha enviat el codi HTML de la pàgina index.php, sinó que ha hagut de compilar-la i executar-la tal com vam veure amb les pàgines JSP. En aquest cas, el compilador ha estat el mòdul *php5* instal·lat a la màquina virtual.

### 1.2.4 Què s'ha après?

Heu après que existeixen diferents llenguatges de programació que coexisteixen dintre d'una pàgina web. En aquest cas, heu vist com s'interpreta el llenguatge PHP.

Resumint, heu après a:

- Instal·lar i configurar un servidor PHP.
- Configurar NetBeans per treballar amb un servidor PHP remot.
- Com interacciona un navegador web amb el servidor PHP.
- Crear una pàgina senzilla amb aquest llenguatge.

Ja esteu preparats per començar les activitats proposades en aquest apartat per tal de poder endinsar-vos en el món de la programació amb JSP i PHP.

## 2. PHP i VDL a Java EE: dades, estructures de control i 'arrays'

En aquesta unitat parlarem dels llenguatges de marques executables per un servidor web identificant-los i escrivint sentències simples per comprovar els seus efectes en la pàgina web resultant. Els llenguatges de marques que s'explicaran seran PHP, JSP i JSTL.

El llenguatge PHP s'explicarà de manera molt breu, ja que el llenguatge vehicular del mòdul serà Java (JEE7 i Spring MVC) i, per tant, el seu llenguatge de marques serà JSP.

Veurem les estructures de control més habituals, el tipus de dades que es poden utilitzar i l'estructura de dades més utilitzada per realitzar iteracions, els *arrays*.

Començarem amb PHP descrivint una persona. Definirem les variables necessàries per guardar la informació i les mostrarem pel navegador. En l'exemple següent utilitzarem els *arrays* per emmagatzemar les dades referents a la gestió d'un hotel i aprendrem a recórrer-los utilitzant les estructures de control més habituals.

Utilitzarem els mateixos exemples per descriure una persona amb JSP i amb JSTL. Així, apreciarem les diferències i les igualtats d'implementar en aquests llenguatges.

En tots els casos s'explicaran els tipus de variable que existeixen, com convertir els valors d'un tipus a un altre i l'àmbit de les variables creades.

En aquest apartat posem les bases de l'aprenentatge d'aplicacions web, i explicarem:

- el llenguatge PHP
- el llenguatge JSP
- el llenguatge JSTL

S'explicarà de cada llenguatge:

- les variables
- els àmbits de les variables
- les estructures de control més habituals
- els tipus de dades i *arrays*

Tots els conceptes que es veuran s'explicaran sempre partint de l'exemple. Quan acabeu aquest apartat estareu preparats per començar a fer les primeres aplicacions web amb PHP i JSP.

No triguem més temps i comencem descrivint una persona amb el llenguatge PHP. Però abans que comenceu aquest apartat comproveu que heu portat a terme la instal·lació de l'entorn PHP proposada a l'apartat anterior. Penseu que per provar els exemples heu de tenir un servidor PHP per executar-los.

## 2.1 Descrivint una persona amb PHP

En aquest apartat veurem quins tipus de dades existeixen a PHP, com crear variables i quin és el seu àmbit. Aprendrem aquests conceptes en el procés de descriure una persona i intentar emmagatzemar les seves dades.

Començarem creant un nou projecte amb l'IDE Netbeans del tipus PHP/PHP Application. A continuació creem la variable *NomPersona* i li assignem un nom de tipus *string*:

Als annexos del material didàctic trobareu el codi PHP que utilitzarem en aquest apartat, per poder-lo descarregar. Recordeu que podeu utilitzar la funció d'importar del Netbeans per accedir a aquest contingut més fàcilment.

```
1 $nomPersona = "Àlex";
```

Observem que a PHP les variables comencen amb el símbol del dòlar. Sempre, en utilitzar una variable, aquesta ha d'estar precedida per aquest símbol: \$. Les variables a PHP han de complir els següents requisits:

- Han de començar amb el símbol \$.
- El nom de la variable ha de començar per una **lletra** o amb el símbol guió baix \_.
- El nom de la variable no pot començar amb un nombre.
- El nom de la variable només pot contenir nombres, lletres i guions baixos.
- PHP distingeix entre majúscules i minúscules. Les variables *\$persona* i *\$Persona* són diferents.

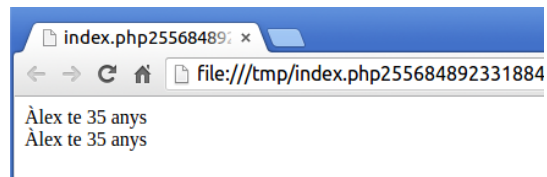
També veiem que en crear la variable *\$nomPersona* no definim prèviament el seu tipus, ni tan sols el declarem. PHP no suporta la definició explícita de tipus en la declaració de variables, i el tipus d'una variable es determina pel seu context. Aquesta peculiaritat ens permet que una mateixa variable a vegades sigui d'un tipus i unes altres vegades sigui d'un altre.

Executeu el següent codi:

```
1 $nomPersona = "Àlex";  
2 $edat = "35";  
3 // la variable edat és de tipus string  
4 echo "$nomPersona té $edat anys<br>";  
5 //Ara la variable edat és de tipus integer:  
6 $edat = 34;  
7 $edat++;  
8 echo "$nomPersona té $edat anys";
```

Observem a la figura 2.1 que els dos missatges són idèntics i la variable *\$edat* primer era de tipus *string* i després ha canviat a ser de tipus *integer*.

**FIGURA 2.1.** Resultat de l'execució del codi anterior



Ara que ja sabeu crear variables a PHP, intenteu definir el patró dels enginyers informàtics, Ramon Llull. En concret, es vol:

- nom
- cognoms
- edat de tipus numèric
- data de naixement
- telèfon
- adreça postal
- adreça electrònica

Quan hàgiu acabat podeu veure la solució:

```

1 $nomPersona = "Ramon";
2 $cognoms = "Llull";
3 $edat = 83; //edat aprox. de la seva mort
4 $data_naixement = strtotime("1232-01-01"); //no se sap el dia exacte
5 $telefon = "935555555";
6 $adrecaPostal = "Ciutat de Mallorca, Convent de Sant Francesc de Palma";
7 $email = "ramon.llull@ioc.cat";

```

A continuació es mostra una altra configuració per a una altra persona:

```

1 $nomPersona = "Clara"; //tipus string
2 $cognoms = "Oswin"; // tipus string
3 $edat = 30; // tipus integer
4 $sou = 30000; //tipus integer
5 $data_naixement = strtotime("1986-03-11"); //timestamp (int)
6 $telefon = "935555555"; // tipus string
7 $adrecaPostal = "Blackpool, England"; //tipus string
8 $email = "oswin@dr.who"; //tipus string
9 $treballa = true; // tipus boolean
10 $alcada = 167.23; // tipus float
11 $convStringData = date("jS F, Y", $data_naixement); //Converteix a string una
    data segons un format donat.

```

Com podeu veure, no cal definir cap tipus. Automàticament, la variable agafa el valor que li hem donat i és quan es defineix el seu tipus. Si executem el codi anterior veiem que no podem veure res per pantalla.

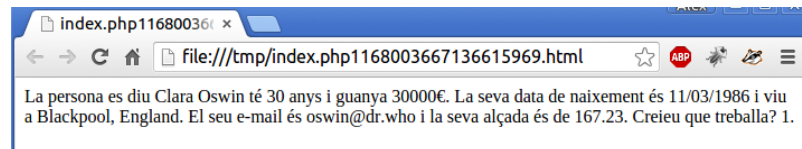
A PHP, en ser un llenguatge que s'executa en el servidor, necessitem una manera de dir que volem enviar una informació a l'usuari. Es fa amb la comanda **echo**.

Llavors, per poder veure per pantalla la informació anterior podem afegir al final la següent instrucció:

```
1 echo "La persona es diu $nomPersona $cognoms té $edat anys i guanya $sou €." .  
2 " La seva data de naixement és $convStringData i viu a $adrecaPostal." .  
3 " El seu e-mail és $email i la seva alçada és de $alcada. Creieu que treballa?  
   $treballa.";
```

Podeu veure el resultat de l'execució del codi anterior en la figura [figura 2.2](#).

**FIGURA 2.2.** Resultat de l'execució de la descripció d'una persona



En utilitzar la comanda **echo** veieu que sempre intenta convertir les variables a *string*. De fet, concatenar les variables amb els texts que volem donar a l'usuari és molt fàcil. Només s'ha de posar la variable dintre de l'*string* sempre que aquest estigui construït amb dobles cometes ("). També es pot fer amb la forma tradicional concatenant *string* purs amb variables. Per concatenar *strings* s'utilitza el símbol del punt (.).

Si ens fixem, en convertir el booleà *true* a *string* el converteix en el nombre "1". Si la conversió hagués estat a *integer* el resultat seria 1, però si el valor del booleà hagués estat *false* la seva conversió a *integer* seria un 0 i a *string* seria la cadena buida.

PHP té aquest comportament perquè és un derivat del llenguatge C. Per a tots aquells que sabeu C us serà molt fàcil programar amb PHP, s'assemblen molt.

### 2.1.1 Conversions de tipus

Tot i que les variables a PHP no tenen tipus, sí que es pot convertir la dada d'un tipus a un altre. Existeixen diferents maneres de fer-ho. Observeu el següent codi:

```
1 $quantitat = "10";  
2 $quantitat = (int) $quantitat;  
3 $quantitat++;  
4 echo $quantitat;
```

El codi anterior funciona correctament. Al final, la variable *\$quantitat* té un valor de 11. Així, amb la conversió a *integer* s'ha pogut convertir el tipus de dada de *string* a *integer* i realitzar una operació matemàtica.

Aquest tipus de conversió es diu que és una **conversió forçada (casting)**. A la taula [taula 2.1](#) podeu veure les diferents conversions forçades que es poden utilitzar.



TAULA 2.1. Conversions de tipus forçades

Instrucció de conversió	Significat
<i>(int)</i> , <i>(integer)</i>	conversió a <i>integer</i>
<i>(bool)</i> , <i>(boolean)</i>	conversió a <i>boolean</i>
<i>(float)</i> , <i>(double)</i> , <i>(real)</i>	conversió a <i>float</i>
<i>(string)</i>	conversió a <i>string</i>
<i>(array)</i>	conversió a <i>array</i>
<i>(object)</i>	conversió a <i>object</i>
<i>(unset)</i>	conversió a <i>null</i>

## Conversió a booleà

Ara que ja sabem com convertir un tipus de dades ens falta per saber què hem d'esperar. És a dir, quina dada hem d'esperar obtenir després de la conversió. En aquest cas, volem convertir a booleà. Vegeu les conversions a booleà següents:

```

1 var_dump((bool) "");           // bool(false)
2 var_dump((bool) 1);           // bool(true)
3 var_dump((bool) -2);          // bool(true)
4 var_dump((bool) "foo");       // bool(true)
5 var_dump((bool) 2.3e5);       // bool(true)
6 var_dump((bool) array(12));   // bool(true)
7 var_dump((bool) array());     // bool(false)
8 var_dump((bool) "false");     // bool(true)

```

En aquest cas, en comptes d'utilitzar la instrucció *echo* s'ha escollit utilitzar la instrucció *var\_dump*. Aquesta instrucció dona informació estructurada l'expressió donada incloent-hi el seu tipus i el seu valor.

Sembla, pel codi anterior, que sempre o quasi sempre es converteix a *true* qualsevol cosa. Existeixen molt pocs casos en què un tipus de dades es converteix a *false*, només quan:

- És l'*integer* 0.
- És el *float* 0.0.
- És un *string* buit o un *string* amb el valor "0".
- Un *array* sense elements.
- El tipus especial *null*.
- Objectes *simpleXML* creats des d'etiquetes buides.

## Conversió a 'string'

Hem vist que es pot convertir a *string* utilitzant la conversió forçosa, però també podem utilitzar la funció *strval()*.

```

1 var_dump(strval(1));           // "1"
2 var_dump(strval(array()));    // "Array"
3 var_dump(strval(-2));         // "-2"
4 var_dump(strval(false));      // ""
5 var_dump(strval(true));       // "1"

```

```

6 var_dump(strval(2.3e5)); // "230000"
7 var_dump(strval(NULL)); // ""

```

Els únics casos remarcables és quan s'intenta convertir un *false*, un *null* o un *array*. En els dos primers casos, la conversió esdevé cadena buida. A l'últim cas, curiosament, ens retorna una cadena amb la paraula *array*.

Tot i així, tots els valors es poden convertir de manera correcta en *string* per després convertir-los novament en el seu tipus. Amb la funció *serialize()* podem emmagatzemar en un *string* un objecte sencer per després tornar-lo a restaurar. La forma de restaurar-lo és utilitzant la funció *unserialize()*.

```

1 $arr = array("a","e","i","o","u");
2 $serial = serialize($arr);
3 echo($serial);

```

El resultat de l'execució del codi anterior és el següent:

```

1 a:5:{i:0;s:1:"a";i:1;s:1:"e";i:2;s:1:"i";i:3;s:1:"o";i:4;s:1:"u";}

```

Com veieu, aquesta execució ha convertit l'*array* a una cadena, i utilitzant la funció *unserialize()* podem reconstruir l'*array* original. Fem-ho:

```

1 $arrayText = 'a:5:{i:0;s:1:"a";i:1;s:1:"e";i:2;s:1:"i";i:3;s:1:"o";i:4;s:1:"u";}';
2 $arrayOriginal = unserialize($arrayText);
3 echo $arrayOriginal[1]; //e

```

Gràcies a la funció *unserialize()* podem recuperar l'objecte original i tractar les dades normalment.

## Conversió a 'integer'

Igual que hem vist amb la conversió de *string*, la conversió a un valor de tipus *integer* es pot realitzar de dues maneres: la primera amb la conversió forçosa utilitzant la paraula reservada (*int*) o (*integer*). La segona, utilitzant la funció *intval()*.

```

1 var_dump(intval("1")); // 1
2 var_dump(intval(array(3,9,8,6,"w"))); // 1
3 var_dump(intval(array())); // 0
4 var_dump(intval("-2")); // -2
5 var_dump(intval(false)); // 0
6 var_dump(intval(true)); // 1
7 var_dump(intval(2.3e5)); // 230000
8 var_dump(intval(NULL)); // 0
9 var_dump(intval("casa")); // 0

```

Observant el resultat de l'execució del codi anterior notem que si no pot traduir un objecte a un *integer* el resultat de la conversió és el nombre 0. Aquests casos són quan, per exemple, volem convertir l'*string* "casa" o un *array* buit. En canvi, si l'*array* conté algun element es tradueix pel nombre 1.

## Conversió a 'array'

Es pot convertir qualsevol dada en un *array*. El que es fa és crear un *array*, i com a únic i primer element la dada que volies convertir en *array*. Exemple:

```
1 $dada = "hola";
2 $arr = (array) $dada;
3 echo $arr[0]; // hola
```

En executar el codi anterior ens retorna la paraula “hola”. Per això, convertir una dada en *array* és el mateix que crear un *array* amb aquesta dada dintre:

```
1 $arr = array("hola");
2 echo $arr[0]; // hola
```

### 2.1.2 Àmbit de les variables

En els exemples anteriors sempre teníem accés a les variables i podíem veure i manipular el seu valor. No sempre és així. Si tornem a l'exemple del principi, “descriuint una persona”, el podem modificar de la següent manera:

```
1 function calculaEdat($data){
2     $avui = date_create('today');
3     $diferencia = date_diff($avui, $data)->y;
4     return $diferencia;
5 }
6 $nomPersona = "Clara"; //tipus string
7 $cognoms = "Oswin"; // tipus string
8 $data_naixement = date_create("1986-03-11"); //tipus date
9 $edat = calculaEdat($data_naixement);
10 echo $edat;
```

Si executeu el codi anterior veureu que no hi ha cap sorpresa. S'executa perfectament sense error i ens proporciona les dades esperades. Resultat de l'execució:

```
1 29
```

Però podríem fer que la funció *calculaEdat* accedeixi directament a la data de naixement sense haver d'enviar-la per paràmetre? Podríem fer:

```
1 $nomPersona = "Clara"; //tipus string
2 $cognoms = "Oswin"; // tipus string
3 $data_naixement = date_create("1986-03-11"); //tipus date
4 function calculaEdat(){
5     $avui = date_create('today');
6     $diferencia = date_diff($avui, $data_naixement)->y;
7     return $diferencia;
8 }
9 $edat = calculaEdat();
10 echo $edat;
```

En altres paraules, la variable *\$data\_naixement* és una variable d'àmbit global? Proveu-ho.

En executar el codi us adoneu que no funciona. No podeu accedir a la variable *\$data\_naixement* dintre de la funció *calculaEdat()*. Les variables que s'utilitzen dintre d'una funció es troben en un àmbit diferent de les variables que existeixen fora de la funció. Sembla lògic pensar que la variable *\$diferencia* no es pot utilitzar fora de la funció:

```
1 $nomPersona = "Clara"; //tipus string
2 $cognoms = "Oswin"; // tipus string
3 $data_naixement = date_create("1986-03-11"); //tipus date
4 function calculaEdat($data){
5     $avui = date_create('today');
6     $diferencia = date_diff($avui, $data )->y;
7     return $diferencia;
8 }
9 calculaEdat($data_naixement);
10 echo $diferencia;
```

Efectivament, si ho proveu tampoc funciona. Definitivament, es troben en àmbits diferents.

**L'àmbit d'una variable** és el context dins de la qual la variable està definida. La major part de les variables PHP només tenen un àmbit simple.

### Àmbit global i àmbit local

No existeixen variables globals? No podem accedir dintre de la funció *calculaEdat* a la data de naixement sense que ens l'enviïn per paràmetre?

Sí que es pot. De fet, la variable *\$data\_naixement* està declarada en l'**àmbit global**. Totes les variables que es declaren fora de les funcions són variables globals. Podem tenir totes les variables globals que vulguem. En canvi, les variables que es declaren dintre d'una funció són variables d'**àmbit local** a les quals mai podrem accedir fora de la funció (un exemple és la variable *\$diferencia*).

Com és que en la variable *\$data\_naixement*, tot i ser d'àmbit global, no hem pogut accedir al seu valor dintre de la funció *calculaEdat()*?

PHP no sap si vols una variable local amb el mateix nom o vols utilitzar la variable global. Si no informes a PHP que la variable *\$data\_naixement* és global pensarà que vols una variable local amb aquest nom. Observa el següent codi per veure com utilitzar *\$data\_naixement* com a variable global:

```
1 function calculaEdat(){
2     global $data_naixement;
3     $avui = date_create('today');
4     $diferencia = date_diff($avui, $data_naixement )->y;
5     return $diferencia;
6 }
7 $nomPersona = "Clara"; //tipus string
8 $cognoms = "Oswin"; // tipus string
9 $data_naixement = date_create("1986-03-11"); //tipus date
10 $edat = calculaEdat();
11 echo $edat;
```

S'executa perfectament sense error i ens proporciona les dades esperades. Resultat de l'execució:

```
1 29
```

Dintre de la funció `calculaEdat()` hem hagut de declarar la variable `$data_naixement` com a global. Hem utilitzat la paraula reservada **global**.

També podem utilitzar l'*array* associatiu **GLOBALS** per accedir a les variables globals i, d'aquesta manera, ens estalviem haver de declarar-la *global*.

```
1 function calculaEdat(){
2     $avui = date_create('today');
3     $diferencia = date_diff($avui, $GLOBALS["data_naixement"] )->y;
4     return $diferencia;
5 }
6 $nomPersona = "Clara"; //tipus string
7 $cognoms = "Oswin"; // tipus string
8 $data_naixement = date_create("1986-03-11"); //tipus date
9 $edat = calculaEdat();
10 echo $edat;
```

Funciona exactament igual que abans, el resultat de l'execució és:

```
1 29
```

Ara veiem una peculiaritat dintre de l'àmbit local. Normalment, les variables perden el seu valor una vegada la funció s'ha executat, o no?

```
1 function suma(){
2     static $num;
3     $num++;
4     return $num;
5 }
6 echo suma();
7 echo suma();
8 echo suma();
```

El resultat de l'execució del codi anterior és:

```
1 1
2 2
3 3
```

La paraula reservada **static** permet guardar el valor d'una variable d'àmbit local dintre de la funció. Cada vegada que s'executi la funció, la variable tindrà l'últim valor que havia tingut a l'execució anterior.

## 2.2 Gestionant un hotel amb PHP

En aquest apartat veurem el tipus de dades *array* i les estructures de control necessàries per tractar amb aquest tipus de dades. Aprendre aquests conceptes gestionant, d'una manera molt senzilla, alguns aspectes de la gestió d'un hotel.

### 2.2.1 Gestió de les taules del restaurant del hotel

Començarem amb la gestió de les taules del restaurant. Volem saber quants comensals hi ha en cadascuna de les taules del restaurant. Suposeu que tenim 10 taules i que en cadascuna de les taules hi pot haver fins a 5 comensals. Aneu pensant quins passos haurem de realitzar per aconseguir-ho.

Si només tinguéssim una taula en el restaurant escriuríem un codi com aquest:

```
1 $comensals = rand(0,5);  
2 echo "A la taula hi ha $comensals comensals";
```

La funció *rand(min, max)* ens dona un nombre pseudoaleatori entre el nombre mínim i el nombre màxim definits en la crida de la funció. Un possible resultat de l'execució del codi anterior seria:

```
1 A la taula hi ha 3 comensals.
```

Imagineu que la funció aleatòria ens retorna el valor 0. No seria millor que el programa ens digués que “la taula està buida”, en comptes de “a la taula hi ha 0 comensals”? Modifiquem el codi anterior per afegir aquesta millora:

```
1 $comensals = rand(0,5);  
2 if ($comensals == 0){  
3     echo "La taula està buida";  
4 }  
5 else{  
6     echo "A la taula hi ha $comensals comensals";  
7 }
```

L'estructura de control *if* és una de les estructures més importants. Permet l'execució condicional d'un tros de codi. Funciona de la següent manera:

- L'expressió que hi ha entre parèntesis, després de la paraula clau *if*, s'avalua i el resultat ha de ser *true* o *false*.
- Si s'avalua com a *true* s'executarà el tros de codi immediatament posterior.
- En canvi, si l'expressió s'avalua com a *false* s'executarà el tros de codi que hi ha després de la paraula reservada *else*.
- En ser una estructura condicional, en cap cas s'executaran els dos trossos de codi a la vegada.

Per exemple, si la variable *\$comensals* tingués el valor 1, 2, 3, 4 o 5, el resultat de l'execució seria:

```
1 A la taula hi ha 1 comensals // quan $comensals = 1
```

En canvi, si la variable *\$comensals* tingués el valor 0, el resultat de l'execució seria:

## 1 La taula està buida

Molt bé, doncs ara volem afegir el condicional de quan la taula està plena. No volem que ens digui que hi ha 5 comensals, si no que volem que ens digui la frase: “La taula està plena”.

```
1 $comensals = rand(0,5);
2 if ($comensals == 0){
3     echo "La taula està buida";
4 }
5 elseif ($comensals == 5){
6     echo "La taula està plena";
7 }
8 else {
9     echo "A la taula hi ha $comensals comensals";
10 }
```

En aquest cas afegim l'estructura *elseif(expressió)*. Com afecta, en l'execució del codi anterior, haver afegit aquesta estructura?

- L'expressió que hi ha entre parèntesis, després de la paraula clau *if*, s'avalua i el resultat ha de ser *true* o *false*.
- Si s'avalua com a *true* s'executarà el tros de codi immediatament posterior.
- En canvi, si l'expressió s'avalua com a *false* s'avaluarà l'expressió de l'estructura *elseif*. Si aquesta s'avalua com a *true* s'executarà el tros de codi immediatament posterior.
- Si no, si s'avalua com a *false* s'executarà el tros de codi que hi ha després de la paraula reservada *else*.
- En ser una estructura condicional, en cap cas s'executaran els tres trossos de codi a la vegada.

Us recomanem que feu diverses proves forçant la variable *\$comensals* perquè executi cadascun dels trossos anteriors. Ho podeu fer actualitzant la pàgina (F5) per canviar el valor donat de la funció *random (rand)*.

Ha arribat el moment de començar a ampliar el restaurant. Amb una única taula no fareu negoci. Tornem al problema que es va plantejar al principi: voleu saber quants comensals hi ha en cadascuna de les taules del restaurant. Supposeu que tenim 10 taules i en cadascuna de les taules hi pot haver fins a 5 comensals.

Si heu estat pensant quina és la millor manera de guardar les dades segur que heu arribat a la conclusió que un *array* ens facilitaria la gestió de les dades. Podem tenir ordenades les taules i podem guardar el nombre de comensals en cadascuna de les seves posicions.

```
1 $taules = array();
2 $taules[0] = rand(0,5);
3 $taules[1] = rand(0,5);
4 $taules[2] = rand(0,5);
5 $taules[3] = rand(0,5);
6 $taules[4] = rand(0,5);
```

```
7 $taules[5] = rand(0,5);
8 $taules[6] = rand(0,5);
9 $taules[7] = rand(0,5);
10 $taules[8] = rand(0,5);
11 $taules[9] = rand(0,5);
```

En el codi anterior, veiem com guardar el nombre de comensals en cadascuna de les taules. Els *arrays* en PHP no cal inicialitzar-los amb el nombre de posicions que ha de tenir (la seva mida). Cada vegada que accedim a una posició de l'*array*, aquesta es crea i s'afegeix la dada. També ho podríem haver codificat de la següent manera:

```
1 $taules = array();
2 array_push($taules, rand(0,5));
3 array_push($taules, rand(0,5));
4 array_push($taules, rand(0,5));
5 array_push($taules, rand(0,5));
6 array_push($taules, rand(0,5));
7 array_push($taules, rand(0,5));
8 array_push($taules, rand(0,5));
9 array_push($taules, rand(0,5));
10 array_push($taules, rand(0,5));
11 array_push($taules, rand(0,5));
```

En aquest cas utilitzem la funció *array\_push*, que afegeix una dada al final de l'*array*. Va creant noves posicions a mesura que es necessiti afegir dades noves.

Tot i que els codis anteriors són correctes i inicialitzen l'*array*, sembla que estem repetint molt de codi, no? I si en comptes de 10 taules tinguéssim 100? Sembla estrany haver de repetir una línia de codi 100 vegades. Observeu la següent estructura de control:

```
1 $taules = array();
2 for($numTaula=0;$numTaula < 10 ; $numTaula++){
3     $taules[$numTaula] = rand(0,5);
4 }
5 echo $taules[4];
```

En aquest cas estem repetint una instrucció deu vegades, però cada vegada que es repeteix canvia la variable *\$numTaula*. A cada volta la variable canvia el seu valor, segons el tercer paràmetre de l'estructura de control, *\$numTaula++*.

Utilitzem aquesta variable per canviar de posició en l'*array* *\$taules* i d'aquesta manera podem afegir els comensals a la nova taula.

L'estructura de control *for* té la següent estructura:

```
1 for (inicialitzar comptador; comparació per saber quan parar; incrementar
2     comptador) {
3     codi a executar;
```

Aquesta estructura és útil quan saps quantes vegades s'han d'executar les instruccions. En aquest cas, com que tenim 10 taules s'ha d'executar 10 vegades. De fet, és l'estructura típica del bucle *for*. Però a PHP hi ha un tipus de bucle *for* que és exclusiu per a *arrays*.



Utilitzarem un *array* associatiu en què en comptes de tenir un nombre per accedir a una posició de l'*array* utilitzarem un *string*. Observeu:

```
1 //creació de l'array
2 $taules = array();
3
4 //Omplir l'array associatiu:
5 for($numTaula=0;$numTaula < 10 ; $numTaula++ ){
6     $taules["taula ".$numTaula] = rand(0,5);
7 }
8
9 //visualització del contingut de l'array
10 foreach($taules as $posicio => $comensals){
11     echo "La $posicio té $comensals comensals\n<br>";
12 }
```

Una possible sortida de l'execució del codi anterior:

```
1 La taula 0 té 4 comensals
2 La taula 1 té 1 comensals
3 La taula 2 té 4 comensals
4 La taula 3 té 1 comensals
5 La taula 4 té 5 comensals
6 La taula 5 té 3 comensals
7 La taula 6 té 3 comensals
8 La taula 7 té 5 comensals
9 La taula 8 té 1 comensals
10 La taula 9 té 5 comensals
```

Amb l'estructura de control *foreach* podem accedir a la clau (*\$posicio*) i al valor (*\$comensals*). Aquests valors s'omplen de manera automàtica i a cada volta ens dóna la següent posició i el seu valor corresponent: *\$comensals = \$taules[\$posicio];*.

L'estructura de control *foreach* té la següent estructura:

```
1 foreach (expresio_array as $clau => $valor){
2     sentencies
3 }
```

Doncs ara, per acabar la gestió del restaurant haurem d'unir les estructures condicionals per saber quan una taula està plena o buida amb les estructures de control iteratives.

```
1 //creació de l'array
2 $taules = array();
3
4 //Omplir l'array associatiu:
5 for($numTaula=0;$numTaula < 10 ; $numTaula++ ){
6     $taules["taula ".$numTaula] = rand(0,5);
7 }
8
9 //visualització del contingut de l'array
10 foreach($taules as $posicio => $comensals){
11     if ($comensals == 0){
12         echo "La $posicio està buida\n<br>";
13     }
14     elseif ($comensals == 5){
15         echo "La $posicio està plena\n<br>";
16     }
17     else {
18         echo "A la $posicio hi ha $comensals comensals\n<br>";
19     }
```

20 }

Un possible resultat del codi anterior és el següent:

```

1 A la taula 0 hi ha 2 comensals
2 A la taula 1 hi ha 3 comensals
3 A la taula 2 hi ha 1 comensals
4 A la taula 3 hi ha 3 comensals
5 La taula 4 està buida
6 A la taula 5 hi ha 2 comensals
7 La taula 6 està plena
8 La taula 7 està plena
9 La taula 8 està plena
10 La taula 9 està buida

```

## 2.2.2 Gestió de les habitacions de l'hotel

En aquest apartat es veurà el tractament d'informació amb *arrays* de més d'una dimensió. Apreneu com crear *arrays* multidimensionals, com modificar els valors i com cercar-los.

Es tractarà de programar la gestió d'habitacions d'un hotel. Imagineu un hotel amb 5 plantes i 10 habitacions en cadascuna de les plantes. Es vol guardar el nombre de clients que hi ha en cada habitació. Com a màxim hi pot haver 4 clients per habitació.

Començarem creant un *array* de dues dimensions:

```

1 define("NUM_PLANTES", 5);
2 define("NUM_HAB", 10);
3 $habitacions = array();
4 for($pis=0; $pis < NUM_PLANTES; $pis++){
5     $habitacions[$pis] = array();
6     for($porta=0; $porta < NUM_HAB; $porta++){
7         $habitacions[$pis][$porta] = "";
8     }
9 }

```

Veieu que s'utilitza la funció *define*, que defineix una constant en el programa.

Una **constant** és un identificador que s'utilitza per expressar un valor simple, que no variarà en el transcurs del programa. Per convenció, els identificadors de les constants sempre se solen declarar en majúscules.

Per utilitzar la constant en el codi només s'ha d'utilitzar el nom definit prèviament a la funció *define*.

El codi anterior genera una estructura com aquesta:

```

1 Array
2 (
3     [0] => Array
4     (

```

```
5      [0] =>
6      [1] =>
7      [2] =>
8      [3] =>
9      [4] =>
10     [5] =>
11     [6] =>
12     [7] =>
13     [8] =>
14     [9] =>
15 )
16
17 [1] => Array
18 (
19     [0] =>
20     [1] =>
21     [2] =>
22     [3] =>
23     [4] =>
24     [5] =>
25     [6] =>
26     [7] =>
27     [8] =>
28     [9] =>
29 )
30
31 [2] => Array
32 (
33     [0] =>
34     [1] =>
35     [2] =>
36     [3] =>
37     [4] =>
38     [5] =>
39     [6] =>
40     [7] =>
41     [8] =>
42     [9] =>
43 )
44
45 [3] => Array
46 (
47     [0] =>
48     [1] =>
49     [2] =>
50     [3] =>
51     [4] =>
52     [5] =>
53     [6] =>
54     [7] =>
55     [8] =>
56     [9] =>
57 )
58
59 [4] => Array
60 (
61     [0] =>
62     [1] =>
63     [2] =>
64     [3] =>
65     [4] =>
66     [5] =>
67     [6] =>
68     [7] =>
69     [8] =>
70     [9] =>
71 )
72
73 )
```

Per crear un *array* de dues dimensions (matriu bidimensional) a PHP hem de crear un *array* d'*arrays*, és a dir, un *array* en què cadascuna de les seves posicions és un altre *array*. En aquest cas, amb el primer índex de l'*array* *\$habitacions* accedim al pis, i amb el segon índex accedim a l'habitació d'aquell pis.

Per exemple, si accedim a la posició de l'*array* *\$habitacions[0][4]* estem accedint al pis 0 (planta baixa) habitació 4.

Doncs bé, igual que abans amb el restaurant, ara ens falta inicialitzar l'*array* amb el nombre de clients per habitació.

```
1 define("NUM_PLANTES", 5);
2 define("NUM_HAB", 10);
3 define("MAX_CLIENTS", 4);
4
5 $habitacions = array();
6 for($pis=0; $pis < NUM_PLANTES; $pis++){
7     $habitacions[$pis] = array();
8     for($porta=0; $porta < NUM_HAB; $porta++){
9         $habitacions[$pis][$porta] = rand(0,MAX_CLIENTS);
10    }
11 }
```

Només calia afegir la constant *MAX\_CLIENTS*, que indica el nombre màxim de persones que hi pot haver en una habitació i la crida a la funció *random* (*rand()*) perquè de manera aleatòria afegeixi persones a les habitacions.

Ara volem un llistat amb l'ocupació de les habitacions de l'hotel. Aquest llistat s'assemblarà molt al que hem fet abans amb les taules del restaurant.

Aquells que ja teniu una idea de com implementar-ho aneu pensant com faríeu per saber si hi ha alguna habitació lliure.

Codi per obtenir el llistat amb el nombre de persones per habitació:

```
1 //Llistat del nombre de persones per habitació
2
3 //Crear l'estructura inicial
4 define("NUM_PLANTES", 5);
5 define("NUM_HAB", 10);
6 define("MAX_CLIENTS", 4);
7 $habitacions = array();
8 for($pis=0; $pis < NUM_PLANTES; $pis++){
9     $habitacions[$pis] = array();
10    for($porta=0; $porta < NUM_HAB; $porta++){
11        $habitacions[$pis][$porta] = rand(0,MAX_CLIENTS);
12    }
13 }
14
15 //recórrer l'estructura per obtenir la informació:
16 for($pis=0; $pis < NUM_PLANTES; $pis++){
17     for($porta=0; $porta < NUM_HAB; $porta++){
18         switch ($habitacions[$pis][$porta]){
19             case 0:
20                 echo "La habitació $porta de la planta $pis està buida.\n";
21                 break;
22             case 4:
23                 echo "La habitació $porta de la planta $pis està plena.\n";
24                 break;
25             default :
26                 echo "A la habitació $porta de la planta $pis hi ha ".
27                     $habitacions[$pis][$porta]. " persones.\n";
28         }
29     }
30 }
```

```
28     }  
29 }
```

El resultat de l'execució del codi anterior és el següent:

```
1  A l'habitació 0 de la planta 0 hi ha 1 persones.  
2  L'habitació 1 de la planta 0 està buida.  
3  L'habitació 2 de la planta 0 està plena.  
4  L'habitació 3 de la planta 0 està buida.  
5  A l'habitació 4 de la planta 0 hi ha 3 persones.  
6  L'habitació 5 de la planta 0 està buida.  
7  L'habitació 6 de la planta 0 està plena.  
8  A l'habitació 7 de la planta 0 hi ha 1 persones.  
9  A l'habitació 8 de la planta 0 hi ha 3 persones.  
10 L'habitació 9 de la planta 0 està buida.  
11 L'habitació 0 de la planta 1 està plena.  
12 A l'habitació 1 de la planta 1 hi ha 3 persones.  
13 A l'habitació 2 de la planta 1 hi ha 2 persones.  
14 L'habitació 3 de la planta 1 està buida.  
15 A l'habitació 4 de la planta 1 hi ha 1 persones.  
16 A l'habitació 5 de la planta 1 hi ha 2 persones.  
17 L'habitació 6 de la planta 1 està plena.  
18 A l'habitació 7 de la planta 1 hi ha 3 persones.  
19 A l'habitació 8 de la planta 1 hi ha 3 persones.  
20 L'habitació 9 de la planta 1 està plena.  
21 L'habitació 0 de la planta 2 està plena.  
22 L'habitació 1 de la planta 2 està buida.  
23 A l'habitació 2 de la planta 2 hi ha 1 persones.  
24 A l'habitació 3 de la planta 2 hi ha 2 persones.  
25 L'habitació 4 de la planta 2 està plena.  
26 L'habitació 5 de la planta 2 està plena.  
27 A l'habitació 6 de la planta 2 hi ha 2 persones.  
28 L'habitació 7 de la planta 2 està plena.  
29 A l'habitació 8 de la planta 2 hi ha 2 persones.  
30 A l'habitació 9 de la planta 2 hi ha 2 persones.  
31 L'habitació 0 de la planta 3 està buida.  
32 L'habitació 1 de la planta 3 està plena.  
33 A l'habitació 2 de la planta 3 hi ha 2 persones.  
34 L'habitació 3 de la planta 3 està buida.  
35 L'habitació 4 de la planta 3 està buida.  
36 L'habitació 5 de la planta 3 està buida.  
37 L'habitació 6 de la planta 3 està buida.  
38 L'habitació 7 de la planta 3 està plena.  
39 A l'habitació 8 de la planta 3 hi ha 2 persones.  
40 L'habitació 9 de la planta 3 està plena.  
41 L'habitació 0 de la planta 4 està plena.  
42 A l'habitació 1 de la planta 4 hi ha 1 persones.  
43 A l'habitació 2 de la planta 4 hi ha 2 persones.  
44 A l'habitació 3 de la planta 4 hi ha 2 persones.  
45 A l'habitació 4 de la planta 4 hi ha 2 persones.  
46 A l'habitació 5 de la planta 4 hi ha 3 persones.  
47 L'habitació 6 de la planta 4 està plena.  
48 A l'habitació 7 de la planta 4 hi ha 2 persones.  
49 A l'habitació 8 de la planta 4 hi ha 2 persones.  
50 A l'habitació 9 de la planta 4 hi ha 3 persones.
```

Com podeu observar, l'estructura per recórrer la matriu bidimensional i l'estructura per crear-la és la mateixa. S'ha de recórrer cada *array* que s'ha creat a cada posició de l'*array* principal. Una vegada hem accedit a una habitació d'una planta s'ha de comprovar el nombre de persones que hi ha.

Aquesta vegada, per fer-ho una mica diferent, s'ha utilitzat l'estructura ***switch***. Aquesta estructura de control és similar a una sèrie de sentències *if*. En moltes ocasions, és possible que es vulgui comparar la mateixa variable (o expressió) amb molts valors diferents i executar una part del codi depenent de quin valor és igual.

Exemple de dues maneres d'escriure el mateix: la primera amb les estructures de control *if* i la segona amb l'estructura de control *switch*:

```
1  if ($i == 0) {
2      echo "i és igual a 0";
3  } elseif ($i == 1) {
4      echo "i és igual a 1";
5  } elseif ($i == 2) {
6      echo "i és igual a 2";
7  }
8  switch ($i) {
9      case 0:
10         echo "i és igual a 0";
11         break;
12     case 1:
13         echo "i és igual a 1";
14         break;
15     case 2:
16         echo "i és igual a 2";
17         break;
18 }
```

Ha arribat l'hora de solucionar el repte plantejat anteriorment. Volem saber si existeix una habitació lliure. Volem recórrer la matriu bidimensional, però quan hagi trobat la primera habitació lliure ha de terminar el programa. L'estructura de control que ens permet parar la cerca una vegada hem trobat la dada és la següent:

```
1  while (expr){
2      sentencies
3  }
```

Per realitzar la cerca demanada primer inicialitzarem la matriu i després, amb aquesta estructura de control, buscarem la primera habitació on hi hagi 0 persones:

```
1  //Cercar si hi ha habitacions lliures
2  //Crear l'estructura inicial
3  define("NUM_PLANTES", 5);
4  define("NUM_HAB", 10);
5  define("MAX_CLIENTS", 4);
6  $habitacions = array();
7  for($pis=0; $pis < NUM_PLANTES; $pis++){
8      $habitacions[$pis] = array();
9      for($porta=0; $porta < NUM_HAB; $porta++){
10         $habitacions[$pis][$porta] = rand(0,MAX_CLIENTS);
11     }
12 }
13 //recórrer l'estructura per obtenir la informació:
14 $pis=0;
15 $porta=0;
16 $trobat = false;
17 while (!$trobat && $pis < NUM_PLANTES){
18     if($habitacions[$pis][$porta] === 0){
19         $trobat = true;
20     }
21     if($porta == NUM_HAB -1){
22         $porta = 0;
23         $pis++;
24     }
25     else{
26         $porta++;
27     }
28 }
29 echo ($trobat)? "Almenys hi ha una habitació lliure." : "No existeixen
    habitacions lliures.";
```

Per recórrer la matriu tenim dos índexs: *\$pis* i *\$porta*. Amb el primer índex accedim a la planta de l'hotel, i amb el segon a l'habitació d'aquella planta. Primer hem d'inicialitzar els índexs:

```
1 $pis = 0;
2 $porta = 0;
```

Una vegada inicialitzats ja podem recórrer la matriu. Com? Doncs pis a pis. Hem de recórrer totes les habitacions del primer pis (*\$porta++*), i quan les haguem recorregut hem d'anar al següent pis. Anar al següent pis significa que hem arribat a l'última porta, llavors inicialitzem el número de porta (*\$porta = 0;*) i a l'índex *\$pis* sumar-hi un més.

```
1 if($porta == NUM_HAB -1){
2     $porta = 0;
3     $pis++;
4 }
5 else{
6     $porta++;
7 }
```

Però hem dit que hem de cercar la primera habitació on no hi hagi cap persona. Quan trobem una que compleixi la condició hem d'avisar, posant una variable a *true*, que ja hem trobat el que estàvem cercant:

```
1 if($habitacions[$pis][$porta] === 0){
2     $trobat = true;
3 }
```

En aquest cas, utilitzem una comparació amb tres símbols *=*. Els tres iguals significa que el valor ha de ser el mateix, com sempre, però a més a més ha de ser del mateix tipus. En aquest cas, ha de ser un 0 i de tipus enter, ja que al costat dret de la comparació utilitzem directament el símbol de tipus enter amb valor 0.

Si trobem una habitació amb 0 persones la variable *\$trobat* canvia el seu valor a *true*. Fent aquest canvi, la condició per continuar dintre del bucle esdevé *false* i no tornarà a executar cap sentència de dintre del bucle.

```
1 $trobat = false;
2 while (!$trobat && $pis < NUM_PLANTES){
3     if($habitacions[$pis][$porta] === 0){
4         $trobat = true;
5     }
6     ....
7 }
```

El bucle acaba i ja podem donar una resposta a l'usuari. En canvi, què passa si no hi ha cap habitació lliure? És a dir, què passaria si no es posa a *true* la variable *\$trobat*? Doncs hem de tenir en compte aquesta situació a l'hora d'elaborar la condició de sortida del bucle. Si no ho tenim en compte el bucle mai acabaria i continuaria indefinidament.

Quina altra condició podem utilitzar per sortir del bucle si no hi ha cap habitació buida? És clar, haver recorregut tota la matriu. En aquest cas, quan haguem arribat

al cinquè pis. Si arribem a aquest pis, aquesta condició  $\$pis < NUM\_PLANTES$  esdevé falsa i el bucle s'atura.

Només ens faltaria comunicar a l'usuari si existeixen o no habitacions lliures.

```
1 echo ($trobat)? "Almenys hi ha una habitació lliure." : "No existeixen habitacions lliures.";
```

Amb el codi anterior estem realitzant una estructura condicional alternativa al típic *if()...else....* S'anomena **operador ternari** i ens estalvia molt temps en l'escriptura del nostre codi, alhora que el simplifica.

```
1 expressio1 ? expressio2 : expressio3
```

S'avalua l'*expressio1*, i si el seu resultat és veritat, llavors s'avalua i retorna com a resultat l'*expressio2*. Si *expressio1* és fals, s'avalua i retorna *expressio3*. En aquest cas, és equivalent a escriure:

```
1 if (expressio1) {  
2     expressio2;  
3 } else {  
4     expressio3;  
5 }
```

Un exemple d'assignació a una variable:

```
1 //sense operador ternari  
2 $faFred=false;  
3 if ($faFred) {  
4     $temps = "Fa molt de fred";  
5 } else {  
6     $temps = "No fa fred";  
7 }  
8 echo $temps;  
9 //amb operador ternari  
10 $faFred=false;  
11 $temps = ($faFred)? "Fa molt de fred" : "No fa fred";  
12 echo $temps;
```

Així, el codi equivalent a l'operador ternari utilitzat a la solució seria:

```
1 //operador ternari utilitzat a la solució  
2 echo ($trobat)? "Almenys hi ha una habitació lliure." : "No existeixen habitacions lliures.";  
3  
4 //estructura equivalent amb if...else  
5 if($trobat){  
6     echo "Almenys hi ha una habitació lliure.";  
7 }  
8 else{  
9     echo "No existeixen habitacions lliures.";  
10 }
```



## 2.3 Descriuint una persona a JSP, JSTL

En aquest apartat començarem a utilitzar el llenguatge de marques de Java: JSP/JSTL, que pertany a JEE7. Veurem quins tipus de dades existeixen a JSP/JSTL, com crear variables i quin és el seu àmbit. Aprendre aquests conceptes en el procés de descriure una persona i intentar emmagatzemar les seves dades.

Als annexos del material didàctic trobareu el codi JSP/JSTL que utilitzarem en aquest apartat, per poder-lo descarregar. Recordeu que podeu utilitzar la funció d'importar del Netbeans per accedir a aquest contingut més fàcilment.

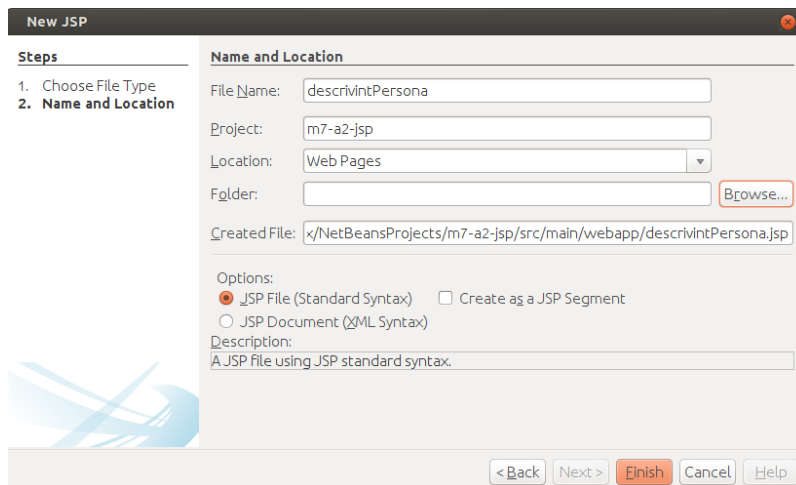
### 2.3.1 Sintaxi estàndard JSP

Comencem creant un projecte amb Netbeans, escollint l'opció *Maven / Web Application*. Crear un projecte amb Maven, com sabeu, és molt útil. És una eina capaç de generar totes les estructures de directoris per al projecte, descarregar automàticament les llibreries que s'utilitzin i, a més a més, està integrat a la majoria dels IDE.

Una vegada s'han descarregat les dependències del projecte, començarem afegint una pàgina JSP. Amb aquesta pàgina descriurem una persona.

A l'hora de crear la pàgina JSP ens demana la informació que podeu veure a la figura 2.3.

FIGURA 2.3. Creació d'una Java Servlet Page



De moment, seleccionem l'opció JSP File (Standard Syntax). Ens apareixerà un codi com aquest:

```
1 <%@page contentType="text/html" pageEncoding="UTF-8"%>
2 <!DOCTYPE html>
3 <html>
4   <head>
5     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
6     <title>JSP Page</title>
7   </head>
```

```

8      <body>
9          <h1>Hello World!</h1>
10     </body>
11 </html>

```

Sembla una pàgina web normal, però hi ha etiquetes addicionals. Per exemple, fixeu-vos-hi:

```

1 <%@page contentType="text/html" pageEncoding="UTF-8"%>

```

El codi JSP s'introdueix utilitzant les etiquetes: `<% — codi JSP — %>`. Dintre d'aquestes etiquetes podem utilitzar codi Java. Recordeu que aquest codi s'executa en el servidor abans d'enviar la pàgina al navegador. En aquest cas, `<%@page` és una **directiva** on es defineixen uns paràmetres que s'han de complir a tota la pàgina. En concret, es defineix el tipus de contingut de la pàgina JSP i la seva codificació.

També podem fer altres coses, com importar classes o bé indicar si volem activar la sessió de l'usuari. Exemple:

```

1 <%@page session="true" import="java.io.*, miPackage.miClase"%>

```

Per descriure les dades d'una persona necessitem declarar les variables necessàries per a la seva descripció. Fixeu-vos com es declaren les variables:

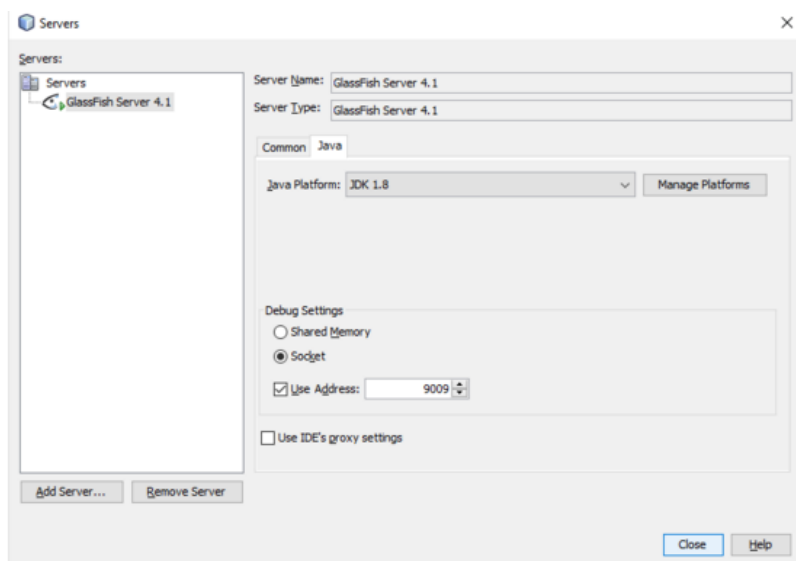
```

1 <%@page import="java.time.LocalDate"%>
2 <%@page contentType="text/html" pageEncoding="UTF-8"%>
3 <%!
4 String nomPersona = "Clara" ;
5 String cognoms = "Oswin";
6 int edat = 30;
7 LocalDate dataNaixement = LocalDate.of(1986,3,11);
8 String telf = "935555555";
9 String adrecaPostal = "Blackpool, England";
10 String email = "oswin@dr.who";
11 boolean treballa = false;
12 float alcada = 167.23f;
13 %>
14 <!DOCTYPE html>
15 <html>
16     <head>
17         <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
18         <title>JSP Descriuint a una Persona</title>
19     </head>
20     <body>
21         <h1>Descriuint una persona</h1>
22         <h2> Les dades de la persona són:</h2>
23         <ul>
24             <li>Es diu: <%=nomPersona + " " + cognoms%> </li>
25             <li>Té <%=edat%> anys</li>
26             <li>Va néixer l'any: <%=dataNaixement.toString()%> </li>
27             <li>El seu telèfon és: <%=telf %> </li>
28             <li>Viu a <%=adrecaPostal %> </li>
29             <li>El seu e-mail és el <%=email %> </li>
30             <li>i actualment <%= (treballa) ? "si" : "no" %> treballa.</li>
31         </ul>
32     </body>
33 </html>

```

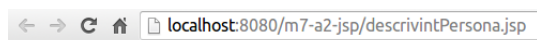
Per executar el codi anterior es fa servir la llibreria *java.time.LocalDate*, amb la qual és molt més senzill fer càlcul amb dates. Per poder utilitzar-la s'ha de canviar la versió del JDK del servidor GlassFish del 1.7 al 1.8. Vegeu la figura 2.4.

**FIGURA 2.4.** Canvi de la versió del JDK al servidor Glassfish



L'execució del codi anterior la podeu veure a la figura 2.5.

**FIGURA 2.5.** Descriuint una persona amb JSP



## Descriuint a una persona

### Les dades de la persona són:

- Es diu: Clara Oswin
- Té 30 anys
- Va néixer l'any: 1986-03-11
- El seu telèfon és el: 935555555
- Viu a Blackpool, England
- El seu e-mail és el oswin@dr.who
- i actualment no treballa.

Podeu executar el codi directament executant aquest fitxer (botó dret del ratolí damunt del fitxer i seleccionar *Run File*). El codi HTML resultant és:

```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
5     <title>JSP Descriuint a una Persona</title>
6   </head>
7   <body>
8     <h1>Descriuint una persona</h1>
9     <h2>Les dades de la persona són:</h2>
10    <ul>
11      <li>Es diu: Clara Oswin </li>
12      <li>Té 30 anys</li>
13      <li>Va néixer l'any: 1986-03-11 </li>
14      <li>El seu telèfon és el: 935555555 </li>
15      <li>Viu a Blackpool, England </li>
16      <li>El seu e-mail és oswin@dr.who </li>
17      <li>i actualment no treballa.</li>

```

```

18     </ul>
19 </body>
20 </html>

```

Per declarar les variables hem utilitzat el símbol **<%! –declaracions– %>**:

```

1 <%!
2 String nomPersona = "Clara" ;
3 String cognoms = "Oswin";
4 int edat = 30;
5 LocalDate dataNaixement = LocalDate.of(1986,3,11);
6 String telf = "935555555";
7 String adrecaPostal = "Blackpool, England";
8 String email = "oswin@dr.who";
9 boolean treballa = false;
10 float alcada = 167.23f;
11 %>

```

Com que es necessita la llibreria *LocalDate* per declarar la data de naixement s'ha hagut d'importar, declarant-la al principi del document per poder-la utilitzar:

```

1 <%@page import="java.time.LocalDate"%>

```

Una vegada tenim les variables inicialitzades les utilitzem amb el símbol **<%= –expressió que retorna un valor – %>**.

```

1 ...
2 <li>Es diu: <%=nomPersona + " " + cognoms%> </li>
3 ...
4 <li>i actualment <%=(treballa)?"si":"no" %> treballa.</li>

```

Aquest símbol **<%=** necessita d'una dada que mostrar. No permet executar més instruccions, només una, i aquesta ha de retornar un valor que es pugui mostrar en aquella posició en la pàgina JSP.

També podem haver codificat la descripció de la persona de la següent manera:

```

1 <%@page import="java.time.LocalDate"%>
2 <%@page contentType="text/html" pageEncoding="UTF-8"%>
3 <%!
4     String nomPersona = "Clara";
5     String cognoms = "Oswin";
6     int edat = 30;
7     LocalDate dataNaixement = LocalDate.of(1986, 3, 11);
8     String telf = "935555555";
9     String adrecaPostal = "Blackpool, England";
10    String email = "oswin@dr.who";
11    boolean treballa = false;
12    float alcada = 167.23f;
13 %>
14 <!DOCTYPE html>
15 <html>
16     <head>
17         <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
18         <title>JSP Descriuint a una Persona</title>
19     </head>
20     <body>
21         <h1>Descriuint una persona</h1>
22         <h2> Les dades de la persona són:</h2>
23         <ul>
24             <%
25                 String html = "<li>Es diu: " + nomPersona + " " + cognoms + " </li>";

```

```

26     html += "<li>Té" + edat + " anys</li>";
27     html += "<li>Va néixer l'any:" + dataNaixement.toString() + "</li>";
28     html += "<li>El seu telèfon és el: " + telf + "</li>";
29     html += "<li>Viu a " + adrecaPostal + " </li>";
30     html += "<li>El seu e-mail és " + email + " </li>";
31     html += "<li> actualment " + ((treballa) ? "si" : "no") + "
        treballa.</li>";
32     html += "<li>i té una alçada de " + alcada + " cm.</li>";
33     out.println(html);
34     %>
35 </ul>
36 </body>
37 </html>

```

Veiem que amb el símbol `<% –codi java – %>` podem crear variables, podem executar instruccions i podem enviar el seu valor amb la instrucció `out.println(expr)` cap al navegador.

La pregunta que ens podem fer és: quina diferència existeix entre declarar les variables dintre dels símbols `<%! ... %>` o dintre de `<% ... %>`? Proveu el següent codi:

```

1 <%@page contentType="text/html" pageEncoding="UTF-8"%>
2 <%! int comptador=0; %>
3 <!DOCTYPE html>
4 <html>
5     <head>
6         <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
7         <title>Comptador</title>
8     </head>
9     <body>
10        <h1>Hola has executat aquesta pàgina <%= comptador++ %> vegades.</h1>
11    </body>
12 </html>

```

Si executeu la pàgina anterior veureu al navegador la següent informació:

```

1 Hola has executat aquesta pàgina 0 vegades.

```

Si, sense tancar la pàgina del navegador, premem F5 (la tornem a carregar), veurem:

```

1 Hola has executat aquesta pàgina 1 vegades.

```

La variable `comptador` no perd el seu valor, es manté. Aquest tipus de variables conserven el seu valor entre successives execucions. En canvi, si les declarem dintre de `<%!..%>` no es manté el seu valor. Exemple:

```

1 <%@page contentType="text/html" pageEncoding="UTF-8"%>
2 <!DOCTYPE html>
3 <html>
4     <head>
5         <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
6         <title>Comptador</title>
7     </head>
8     <body>
9         <%
10            int comptador = 0;
11        %>
12        <h1>Hola, has executat aquesta pàgina <%= comptador++ %> vegades.</h1>
13    </body>

```

```
14 </html>
```

Si executem el codi anterior i premem F5 moltes vegades, no canviarà el seu valor. Sempre mostrarà:

```
1 Hola, has executat aquesta pàgina 0 vegades.
```

## Resum etiquetes JSP (Standard Syntax)

Una vegada hem arribat a aquest punt, recapitem el tipus d'informació i sintaxi que podem utilitzar amb JSP (Standard Syntax):

- **Directives** (`<%@..%>`): indiquen informació general de la pàgina, com pot ser importació de classes, pàgina a invocar davant errors, si la pàgina forma part d'una sessió, incloure una altre pàgina o fitxer, etc.

Exemple:

```
1 <%@page session="true" import="java.util.ArrayList"%>
2 <%@include file="titulo.txt"%>
```

- **Declaracions** (`<%! .. %>`): serveixen per a funcions mètodes o variables. Aquestes variables conservaran el seu valor entre successives execucions. A més a més, les variables declarades dintre d'aquestes etiquetes esdevenen variables globals i poden ser utilitzades des de qualsevol lloc de la pàgina JSP.

```
1 <%-- Exemple --%>
2 <%!
3 private String ahora()
4 {
5     return ""+new java.util.Date();
6 }
7 %>
8 //Exemple2
9 <%! int comptador=0; %>
```

- **Scriptlets** (`<%...%>`): codi Java embegut.

```
1 <%-- Exemple --%>
2 <table>
3     <% for (int i=0;i<10;i++)
4     {
5         %>
6         <tr><td> <%=i%> </td></tr>
7         <% }
8         %>
9 </table>
```

- **Expressions** (`<%=...%>`): expressions Java que s'avaluen i s'envien a la sortida.

```
1 <%= new java.util.Date() %>
2 <%= Math.PI*2 %>
```

### 2.3.2 Sintaxi JSP Standard Tag Library (JSTL)

Veurem una altra manera d'afegir codi Java a una pàgina web i utilitzarem les llibreries d'etiquetes JSTL. Aquesta manera de definir una pàgina web està basada en etiquetes XML.

La llibreria estàndard d'etiquetes **JSTL** és una col·lecció d'etiquetes JSP que encapsula les funcionalitats principals de moltes aplicacions JSP.

Començarem creant una nova pàgina web JSP i afegirem el conjunt d'etiquetes principals JSTL. Per accedir al nou conjunt d'etiquetes posarem el prefix *c*.

```
1 <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

A més, com que utilitzarem dates necessitem el conjunt d'etiquetes *fmt* (internacionalització), que ens permetrà utilitzar-les d'una manera més còmoda.

```
1 <%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
```

Una vegada hem afegit les llibreries de les etiquetes XML que utilitzarem comencem a definir les variables i a mostrar el seu valor per pantalla. El codi resultant és el següent:

```
1 <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
2 <%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
3 <%@page contentType="text/html" pageEncoding="UTF-8"%>
4 <!DOCTYPE html>
5 <html>
6   <head>
7     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
8     <title>JSTL Descriuint a una Persona</title>
9   </head>
10  <body>
11    <h1>Descriuint una persona</h1>
12    <h2> Les dades de la persona són:</h2>
13    <c:set var="nom" value="Clara" scope="page" />
14    <c:set var="cognoms" value="Oswin" scope="page" />
15    <c:set var="edat" value="30" scope="page" />
16    <fmt:parseDate var="datanaixement" pattern="dd-MM-yyyy" value="
17      11-03-1986"/>
18    <c:set var="telefon" value="935555555" scope="page" />
19    <c:set var="adreca" value="Blackpool, England" scope="page" />
20    <c:set var="email" value="oswin@dr.who" scope="page" />
21    <c:set var="treballa" value="false" scope="page" />
22    <c:set var="alcada" value="167.23f" scope="page" />
23    <ul>
24      <li>Es diu: <c:out value="${nom} ${cognoms}" /></li>
25      <li>Té <c:out value="${edat}" /> anys</li>
26      <li>Va néixer el <fmt:formatDate value="${datanaixement}"
27        dateStyle="full"/>.</li>
28      <li>El seu telèfon és el: <c:out value="${telefon}" /> </li>
29      <li>Viu a <c:out value="${adreca}" /> </li>
30      <li>El seu e-mail és <c:out value="${email}" /> </li>
31      <li>actualment <c:out value="${treballa? 'si' : 'no'}" />
32        treballa.</li>
33      <li>i té una alçada de <c:out value="${alcada + 1}" /> cm.</li>
34    </ul>
35  </body>
```

33 `</html>`

Com veieu, per declarar una variable utilitzem l'etiqueta `<c:set`. Recordeu que hem utilitzat la lletra **c** com a prefix per poder utilitzar les etiquetes del nucli *jstl*, i per això les etiquetes comencen amb `<c:`. Una vegada escollida l'etiqueta per declarar variables, a continuació hem de donar-li un nom, un valor i un àmbit. Per donar-li el nom utilitzem l'atribut de l'etiqueta *var*, per donar-li un valor utilitzem l'atribut *value*, i per donar-li un àmbit a la variable utilitzem l'atribut *scope*. Exemple:

1 `<c:set var="edat" value="30" scope="page" />`

Com podeu veure en l'exemple anterior, no cal definir el tipus de variable. Automàticament es converteix en el tipus que necessitem per fer l'operació que necessitem. Vegeu com s'utilitza la variable *alcada*.

Per mostrar la variable per pantalla utilitzem l'etiqueta `<c:out`. Aquesta etiqueta només té l'atribut *value*, amb el qual podem mostrar per pantalla el valor d'una variable o d'una expressió. Per poder utilitzar una variable de les definides prèviament s'ha de fer servir la sintaxi `${expr}`, on *expr* pot ser el nom de la variable a mostrar. Aquest nom correspon a l'atribut *var* de l'etiqueta `<c:set`. Exemple:

1 `<li>i té una alçada de <c:out value="${alcada + 1}" /> cm.</li>`

A l'exemple anterior agafem el valor d'una variable i li sumem 1. Aquest valor es veurà per pantalla en aquesta posició. Com veieu, no cal definir els tipus; automàticament s'ha convertit en un *float*, ha sumat un 1 i després l'ha convertit en un *string*.

Per utilitzar les variables creades s'utilitza la notació `${expr}`, que és estàndard i molt potent. És un llenguatge en si mateix que s'anomena *EL* (Expression Language).

**EL (Expression Language)** permet la resolució dinàmica dels objectes i els mètodes de Java en pàgines JSP i Facelets. Les expressions *EL* tenen la forma de `${foo}` i `#{bar}`.

### JavaServer Faces

JavaServer Faces (**JSF**) és una tecnologia per a aplicacions Java basades en web que simplifica el desenvolupament d'aplicacions Java EE. El principal avantatge sobre JSP és que permet una clara separació entre el comportament de l'aplicació i la seva presentació.

A més a més, per poder programar amb aquest llenguatge necessitem saber quines són les etiquetes que ens proporciona la llibreria principal JSTL (vegeu la taula 2.2).

**TAULA 2.2.** Etiquetes de la llibreria JSTL

Etiqueta	Significat
<i>set</i>	S'utilitza per donar un valor a una propietat o a una variable en qualsevol dels seus àmbits
<i>remove</i>	S'utilitza per eliminar una variable
<i>choose</i>	És una etiqueta condicional utilitzada per a la creació d'un condicional del tipus <i>if...else</i>
. . . . .	



TAULA 2.2 (continuació)

Etiqueta	Significat
<i>when</i>	Aquesta etiqueta s'utilitza dintre de l'etiqueta <i>choose</i> . Serveix per avaluar una expressió a <i>true</i> o <i>false</i>
<i>otherwise</i>	Aquesta etiqueta s'utilitza dintre de l'etiqueta <i>choose</i> . El contingut d'aquesta etiqueta s'executa si l'expressió avaluada per l'etiqueta <i>when</i> és <i>false</i>
<i>forEach</i>	Aquesta etiqueta serveix per fer un bucle
<i>forEachTokens</i>	Aquesta etiqueta serveix per iterar sobre un <i>string</i> . A cada volta dona la porció de l' <i>string</i> que hi ha entre uns delimitadors
<i>import</i>	Aquesta etiqueta serveix per importar un fitxer. Per exemple, podem importar el contingut d'una pàgina o un fitxer dintre d'una variable per després tractar la informació
<i>param</i>	S'utilitza per afegir un paràmetre a la URL
<i>redirect</i>	S'utilitza per redirigir a l'usuari a una altra URL
<i>url</i>	S'utilitza per crear una URL a una pàgina
<i>catch</i>	Serveix per agafar els errors que es produeixen a la pàgina
<i>out</i>	Aquesta etiqueta és l'equivalent a les expressions <code>&lt;%=..%&gt;</code> de la llibreria estàndard de JSP

Anirem utilitzant les etiquetes anteriors segons les requerim per solucionar els diversos problemes que se'ns presentin a l'hora de codificar els exemples amb JSTL.

Només queda per explicar l'atribut *scope* de l'etiqueta *set*. Aquest atribut no deixa de ser l'àmbit de la variable declarada. A JSP existeixen diferents àmbits (taula 2.3).

TAULA 2.3. Tipus d'àmbits a JSP i JSTL

Tipus d'àmbit	Significat
<i>page</i>	Els objectes declarats en aquest àmbit només són accessibles en la pàgina declarada. La dada d'aquest objecte només és vàlida durant el processament de l'actual resposta.
<i>request</i>	Els objectes d'àmbit de <i>request</i> són accessibles en les pàgines processades en la mateixa petició en la qual es va crear.
<i>session</i>	Els objectes declarats en aquest àmbit són accessibles en les pàgines processades de sol·licituds que es troben en la mateixa sessió que en la que van crear.
<i>application</i>	Els objectes declarats en aquest àmbit són accessibles per qualsevol pàgina JSP que formi part de la mateixa aplicació.

Millor veiem amb un exemple com s'utilitzen:

```

1 <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
2
3 <html>
4   <head>
5     <title>Àmbits de les variables a JSLT</title>

```

```

6      </head>
7      <body>
8          <c:set var="test" value="Valor d'una variable d'àmbit de pàgina" scope=
              "page" />
9
10         <c:set var="test" value="Valor d'una variable d'àmbit de Request"
              scope="request" />
11
12         <c:set var="test" value="Valor d'una variable d'àmbit de sessió"
              scope="session" />
13
14         <c:set var="test" value="Valor d'una variable d'àmbit d'aplicació"
              scope="application" />
15
16         <h1>Àmbits de les variables a JSLT:</h1>
17         <table border="1">
18             <tr>
19                 <td>
20                     <b>Àmbit per defecte: </b>
21                 </td>
22                 <td>
23                     <c:out value="${test}" />
24                 </td>
25             </tr>
26             <tr>
27                 <td>
28                     <b>Àmbit de pàgina</b>
29                 </td>
30                 <td>
31                     <c:out value="${pageScope.test}" />
32                 </td>
33             </tr>
34             <tr>
35                 <td>
36                     <b>Àmbit de Request: </b>
37                 </td>
38                 <td>
39                     <c:out value="${requestScope.test}" />
40                 </td>
41             </tr>
42             <tr>
43                 <td>
44                     <b>Àmbit de sessió</b>
45                 </td>
46                 <td>
47                     <c:out value="${sessionScope.test}" />
48                 </td>
49             </tr>
50             <tr>
51                 <td>
52                     <b>Àmbit d'aplicació</b>
53                 </td>
54                 <td>
55                     <c:out value="${applicationScope.test}" />
56                 </td>
57             </tr>
58         </table>
59     </body>
60 </html>

```

Observeu que el nom de la variable utilitzada és el mateix (*test*). Aquesta variable té diferents valors depenent de l'àmbit on està. En altres paraules, existeixen diferents variables anomenades *test* en diferents àmbits del programa.

En executar el codi anterior obtenim la pàgina que podeu veure a la figura [figura 2.6](#)

**FIGURA 2.6.** Àmbits de les variables en JSP


Àmbit per defecte:	Valor d'una variable d'àmbit de pàgina
Àmbit de pàgina	Valor d'una variable d'àmbit de pàgina
Àmbit de Request:	Valor d'una variable d'àmbit de Request
Àmbit de sessió	Valor d'una variable d'àmbit de sessió
Àmbit d'aplicació	Valor d'una variable d'àmbit d'aplicació

### 2.3.3 Calcular l'edat d'una persona

Continuant amb l'exemple de descripció d'una persona, volem afegir la funcionalitat de calcular l'edat si sabem l'any que va néixer. Volem fer el càlcul de dues maneres: amb la sintaxi estàndard i amb la sintaxi JSTL.

Començarem amb la sintaxi estàndard. Afegirem una funció que rebi per paràmetre la data de naixement i retorni l'edat de la persona fins al dia actual.

```

1  <%@page import="java.time.temporal.ChronoUnit"%>
2  <%@page import="java.time.LocalDate"%>
3  <%@page contentType="text/html" pageEncoding="UTF-8"%>
4  <%!
5  private long calculaEdat(LocalDate dataNaix){
6      LocalDate now = LocalDate.now();
7      return ChronoUnit.YEARS.between(dataNaix, now);
8  }
9  %>
10 <!DOCTYPE html>
11 <html>
12     <head>
13         <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
14         <title>JSP Descrivint a una Persona</title>
15     </head>
16     <body>
17         <h1>Descrivint una persona</h1>
18         <h2> Les dades de la persona són:</h2>
19         <ul>
20             <%
21                 String nomPersona = "Clara";
22                 String cognoms = "Oswin";
23                 LocalDate dataNaixement = LocalDate.of(1986, 3, 11);
24             %>
25             <li>Es diu: <%=nomPersona + " " + cognoms%> </li>
26             <li>Té <%=calculaEdat(dataNaixement)%> anys</li>
27             <li>Va néixer l'any: <%=dataNaixement.toString()%> </li>
28         </ul>
29     </body>
30 </html>

```

La declaració de la funció es fa dintre de la zona de declaracions. Una vegada s'ha declarat la funció es pot utilitzar en qualsevol lloc del codi JSP. En aquest cas, la utilitzem directament a una expressió per tal de retornar l'edat de la persona i mostrar-la en el lloc adient de la pàgina JSP.

La variable *dataNaixement* és una variable local que només es pot fer servir en el mateix *scriptlet* i en les expressions d'assignació (*<%=*). Per poder utilitzar aquest valor dintre d'una funció s'ha d'enviar mitjançant un paràmetre, tal com s'ha fet a l'exemple.

També podríem haver utilitzat variables globals. No queda tan bé, però es pot implementar. Exemple:

```

1 <%@page import="java.time.temporal.ChronoUnit"%>
2 <%@page import="java.time.LocalDate"%>
3 <%@page contentType="text/html" pageEncoding="UTF-8"%>
4 <%!
5 public long calculaEdat(){
6     LocalDate now = LocalDate.now();
7     return ChronoUnit.YEARS.between(dataNaixement, now);
8 }
9 LocalDate dataNaixement = LocalDate.of(1986, 3, 11);
10 %>
11 <!DOCTYPE html>
12 <html>
13     <head>
14         <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
15         <title>JSP Descriuint a una Persona</title>
16     </head>
17     <body>
18         <h1>Descriuint una persona</h1>
19         <h2> Les dades de la persona són:</h2>
20         <ul>
21             <%
22                 String nomPersona = "Clara";
23                 String cognoms = "Oswin";
24             %>
25             <li>Es diu: <%=nomPersona + " " + cognoms%> </li>
26             <li>Té <%=calculaEdat()%> anys</li>
27             <li>Va néixer l'any: <%=dataNaixement.toString()%> </li>
28         </ul>
29     </body>
30 </html>

```

La variable global *dataNaixement* s'ha declarat dintre de la zona dedicada a la declaracions de variables i funcions. Aquesta variable, en ser global, es pot utilitzar perfectament dintre de la funció (sense haver d'enviar-la per paràmetre) i en les expressions d'assignacions (<%=).

La segona manera de calcular l'edat d'una persona és utilitzant etiquetes JSTL.

```

1 <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
2 <%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
3 <%@page contentType="text/html" pageEncoding="UTF-8"%>
4 <!DOCTYPE html>
5 <html>
6     <head>
7         <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
8         <title>JSTL Descriuint a una Persona</title>
9     </head>
10    <body>
11        <h1>Descriuint una persona</h1>
12        <h2> Les dades de la persona són:</h2>
13        <c:set var="nom" value="Clara" scope="page" />
14        <c:set var="cognoms" value="Oswin" scope="page" />
15        <fmt:parseDate var="datanaixement" pattern="dd-MM-yyyy" value="
16            11-03-1986"/>
17        <c:set var="avui" value="<%=new java.util.Date()%>" />
18        <ul>
19            <li>Es diu: <c:out value="{nom} {cognoms}" /></li>
20            <li>Té <c:out value="{avui.year - datanaixement.year}" /> anys</li>
21        </ul>
22    </body>
23 </html>

```

Observem que en aquesta solució només es té en compte, en el càlcul de l'edat de la persona, els anys de diferència que hi ha entre les dues dates. Primer creem la data de la persona amb l'etiqueta *fmt*.

```
1 <fmt:parseDate var="datanaixement" pattern="dd-MM-yyyy" value="11-03-1986"/>
```

Aquesta etiqueta crea una data a la variable *datanaixement*. Aquesta data té l'any de naixement de la persona en el format *dia-mes-any*. També podem crear dates utilitzant etiquetes JSTL amb etiquetes JSP estàndard. Fixeu-vos com hem creat la data actual:

```
1 <c:set var="avui" value="<%=new java.util.Date()%>" />
```

Amb el codi anterior hem creat una variable anomenada *avui* que conté la data actual. Ara només s'ha de calcular la diferència entre aquestes dues dates per tal d'obtenir l'edat de la persona.

```
1 <c:out value="${avui.year - datanaixement.year}" />
```

El codi anterior no fa el càlcul exacte de l'edat, però per començar ja ens va prou bé. Fa la resta exacta entre l'any de naixement i l'any actual sense tenir en compte el mes i el dia. L'interessant d'aquesta notació és que podem accedir als *getters* de la classe *java.util.Date* directament, sense haver de posar *datanaixement.getYear()*. Aquest notació ens permet accedir a la propietat d'una classe si aquesta té la funció *get* associada.

## 2.4 Gestionant un hotel amb JSP

En aquest apartat veurem el tipus de dades *array* i les estructures de control necessàries per tractar amb aquest tipus de dades. Aprendre aquests conceptes gestionant, d'una manera molt senzilla, alguns aspectes de la gestió d'un hotel.

### 2.4.1 Gestió de les taules d'un restaurant

Igual que vam fer amb PHP, volem utilitzar el llenguatge JSP per guardar el nombre de comensals que hi ha en cada taula i mostrar la informació.

Penseu com determinar quines taules estan buides. Volem crear una funció que donat un *array* amb el nombre de comensals de cada taula ens retorni en un altre *array* el nombre de cada taula que estigui buida.

```
1 <%@page contentType="text/html" pageEncoding="UTF-8"%>
2 <%!
3     private int rand(int min, int max) {
4         return (int) (Math.random() * (max+1 - min) + min);
5     }
```

```

6  %>
7  <!DOCTYPE html>
8  <html>
9    <head>
10     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
11     <title>Gestió de taules d'un restaurant</title>
12   </head>
13   <body>
14     <%
15       //inicialització de les taules amb els comensals
16       int[] taules = new int[10];
17       for (int i = 0; i < 10; i++) {
18         taules[i] = rand(0, 5);
19       }
20     %>
21     <h1>Gestió de taules d'un restaurant</h1>
22     <ul>
23       <%
24         //llistat de les taules
25         for(int i = 0; i < 10; i++)
26         {
27           switch(taules[i]){
28             case 0:
29               %>
30               <li>La taula <%=i%> està buida</li>
31               <%
32                 break;
33             case 5:
34               %>
35               <li>La taula <%=i%> està plena.</li>
36               <%
37                 break;
38             default:
39               %>
40               <li>La taula <%=i%> té <%=taules[i]%> comensals</li>
41               >
42             <%
43           }
44         }
45       %>
46     </ul>
47   </body>
</html>

```

Una possible resposta a l'execució del codi anterior és el següent:

```

1  Gestió de taules d'un restaurant
2
3  La taula 0 està plena.
4  La taula 1 té 1 comensals
5  La taula 2 està plena.
6  La taula 3 té 4 comensals
7  La taula 4 té 1 comensals
8  La taula 5 està buida
9  La taula 6 té 1 comensals
10 La taula 7 està buida
11 La taula 8 està buida
12 La taula 9 té 3 comensals

```

Observeu el codi anterior. Veieu que s'ha utilitzat l'estructura *for* per recórrer l'*array*. Aquesta estructura funciona igual que a Java i a PHP. El que pot sobtar és la manera d'enviar la informació al navegador. Observeu amb deteniment l'estructura del bucle *for* i com utilitzem les etiquetes `<% ... %>`.

```

1  <%
2  for (int i = 0; i < 10; i++) {

```

```

3      switch (taules[i]) {
4          case 0:
5              %>
6                  <li>La taula <%=i%> està buida</li>
7              <%
8                  break;
9          case 5:
10             %>
11                 <li>La taula <%=i%> està plena.</li>
12             <%
13                 break;
14             default:
15             %>
16                 <li>La taula <%=i%> té <%=taules[i]%> comensals</li>
17             <%
18             }
19         }
20     %>

```

En comptes d'utilitzar la funció *out.println(expr)* per enviar la informació al navegador hem tancat l'etiqueta JSP i hem posat el codi HTML directament a la pàgina. Perquè el codi es pugui enviar a la pàgina HTML ha de complir les condicions posades anteriorment dintre de les etiquetes JSP. Així, podem anar alternant entre codi HTML i codi JSP.

Una alternativa del codi anterior podria ser:

```

1  <%
2  for (int i = 0; i < 10; i++) {
3      switch (taules[i]) {
4          case 0:
5              out.println("<li>La taula " + i + " està buida</li>");
6
7              break;
8          case 5:
9              out.println("<li>La taula " + i + " està plena.</li>");
10
11             break;
12             default:
13                 out.println("<li>La taula " + i + " té " + taules[i] + "
14                     comensals.</li>");
15             }
16     }
17 }
18 %>

```

Solucionem el repte anterior. Volem crear una funció que donat un *array* amb el nombre de comensals de cada taula ens retorni en un altre *array* el nombre de cada taula que estigui buida. Mostrarem per pantalla totes les taules buides.

```

1  <%@page import="java.util.ArrayList"%>
2  <%@page contentType="text/html" pageEncoding="UTF-8"%>
3  <%!
4      private int rand(int min, int max) {
5          return (int) (Math.random() * (max + 1 - min) + min);
6      }
7      private ArrayList cercarBuides(int[] taules){
8          ArrayList buides = new ArrayList();
9          for (int i = 0; i < taules.length; i++) {
10              if(taules[i] == 0){
11                  buides.add(i);
12              }
13          }
14          return buides;
15      }

```

```

16 %>
17 <!DOCTYPE html>
18 <html>
19   <head>
20     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
21     <title>Gestió de taules d'un restaurant</title>
22   </head>
23   <body>
24     <%
25       //inicialització de les taules amb els comensals
26       int[] taules = new int[10];
27       for (int i = 0; i < 10; i++) {
28         taules[i] = rand(0, 5);
29       }
30       //cerquem les taules buides
31       ArrayList taulesBuides = cercarBuides(taules);
32     %>
33     <h1>Les següents taules estan buides:</h1>
34     <ul>
35       <%
36         //llistat de les taules buides
37         if(taulesBuides.isEmpty()){
38           out.println("No hi han taules buides.");
39         }
40         for (Object taula: taulesBuides){
41           out.println("<li>La taula " + taula + " està buida.</li>");
42         }
43       %>
44     </ul>
45   </body>
46 </html>

```

Una possible resposta a l'execució del codi anterior seria:

```

1 Les següents taules estan buides:
2
3 La taula 1 està buida.
4 La taula 6 està buida.
5 La taula 9 està buida.

```

Hem emprat la classe `ArrayList` per emmagatzemar les taules buides. Com que, a priori, no sabem quantes taules buides tindrem, no podem utilitzar un *array* de mida fixa. Java ens proporciona la classe `ArrayList` per a aquests casos. La funció `cercarBuides` recorre l'*array* taules per cercar el número de taula que té 0 comensals. Quan trobem una taula que no té cap comensal l'afegim a l'*array* de taules buides.

```

1 private ArrayList cercarBuides(int[] taules){
2   ArrayList buides = new ArrayList();
3   for (int i = 0; i < taules.length; i++) {
4     if(taules[i] == 0){
5       buides.add(i);
6     }
7   }
8   return buides;
9 }

```

Quan hem obtingut l'*ArrayList* amb el llistat de taules buides hem de mostrar-lo per pantalla. Aquesta vegada iterem l'*ArrayList* amb un bucle del tipus *foreach*. A Java, aquests bucles es codifiquen de la següent manera:

```

1 for (Object taula: taulesBuides){
2   out.println("<li>La taula " + taula + " està buida.</li>");

```



```
3 }
```

Si ens hi fixem, és un bucle *for* però amb una altra sintaxi. Aquesta vegada, dintre dels parèntesis hem d'indicar el tipus d'element que tindrem a cada iteració, el nom de la variable que contindrà l'element i la col·lecció d'elements (podria ser un *array*, però en el nostre cas és un *ArrayList*). En general:

```
1 for(Tipus element: col·lecció){
2     //fer alguna cosa amb l'element
3 }
```

Ja hem vist com utilitzar *arrays* d'una dimensió amb JSP, i ara veurem els mateixos exemples però amb la sintaxi JSTL. Veureu que no tot es pot fer amb JSTL, però farem tot el possible per utilitzar-ho al màxim.

Començarem amb la implementació del programa que llista les taules d'un restaurant.

```
1 <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
2 <%@page contentType="text/html" pageEncoding="UTF-8"%>
3 <%!
4     private int rand(int min, int max) {
5         return (int) (Math.random() * (max + 1 - min) + min);
6     }
7 %>
8 <!DOCTYPE html>
9 <html>
10     <head>
11         <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
12         <title>Gestió de taules d'un restaurant</title>
13     </head>
14     <body>
15         <%
16             //inicialització de les taules amb els comensals
17             int[] taules = new int[10];
18             for (int i = 0; i < 10; i++) {
19                 taules[i] = rand(0, 5);
20             }
21             request.setAttribute("taules", taules);
22         %>
23         <h1>Gestió de taules d'un restaurant</h1>
24         <ul>
25             <c:forEach var="num" varStatus="loop" items="${taules}">
26                 <c:choose>
27                     <c:when test="${num eq 0}">
28                         <c:out value="${<li>La taula'} ${loop.index} {'està
29                             buida</li>'}" escapeXml="false"></c:out>
30                     </c:when>
31                     <c:when test="${num eq 5}">
32                         <c:out value="${<li>La taula'} ${loop.index} {'està
33                             plena</li>'}" escapeXml="false"></c:out>
34                     </c:when>
35                     <c:otherwise>
36                         <c:out value="${<li>La taula'} ${loop.index} {'té '
37                             ${num} {'comensals</li>'}" escapeXml="false"></c:
38                             out>
39                     </c:otherwise>
40                 </c:choose>
41             </c:forEach>
42         </ul>
43     </body>
44 </html>
```

Si observem detingudament el codi anterior ens adonarem que just després de crear l'*array* *taules* i omplir-lo amb els comensals s'ha enregistrat en l'objecte *request*.

```
1 //inicialització de les taules amb els comensals
2 int[] taules = new int[10];
3 for (int i = 0; i < 10; i++) {
4     taules[i] = rand(0, 5);
5 }
6 request.setAttribute("taules", taules);
```

Els objectes que es creen amb la sintaxi estàndard JSP no es poden utilitzar, per defecte, amb les etiquetes JSTL. Aquests objectes se'ls ha de fer accessibles des de JSTL. Una manera de fer-lo accessible és afegint-lo com a atribut del objecte *request* de la pàgina. Una vegada hem fet accessible l'objecte el podem assignar a una variable de JSTL o utilitzar-lo com a col·lecció amb la qual poder iterar.

```
1 <c:forEach var="num" varStatus="loop" items="${taules}">
2     ...
3 </c:forEach>
```

És exactament el que hem fet amb l'*array* *taules*. L'hem assignat a l'atribut *items* de l'etiqueta *foreach*. Aquest atribut ha de ser una col·lecció amb la qual poder extreure tota la informació element a element. Cada element de l'*array* *taules* s'emmagatzema dintre de la variable *num*. A més a més, podem accedir a l'índex de l'*array* de la iteració actual. El nom de l'índex de l'*array* s'ha d'especificar en l'atribut *varStatus* de l'etiqueta *forEach*.

Una vegada ja tenim l'estructura del bucle hem d'utilitzar una estructura de *switch*. A JSTL aquesta estructura la podem crear utilitzant la combinació de les etiquetes *choose*, *when* i *otherwise*.

```
1 //estructura equivalent a switch() però amb etiquetes JSTL
2 <c:choose>
3     <c:when test="${num eq 0}">
4         ...
5     </c:when>
6     <c:when test="${num eq 5}">
7         ...
8     </c:when>
9     <c:otherwise>
10        ...
11    </c:otherwise>
12 </c:choose>
```

Podem utilitzar tantes etiquetes *when* com ens faci falta per especificar els *case* de l'estructura *switch*. L'únic atribut que admet aquesta etiqueta és *test*. Aquest atribut admet una EL (*Expression Language*) que en ser avaluada retorna *true* o *false*. Si s'avalua com a *true* s'executaran les instruccions que hi ha dintre de la mateixa etiqueta *when*. En el cas que s'avalui a *false* anirà a avaluar l'atribut *test* de la següent etiqueta *when* fins que, finalment, s'executa l'etiqueta *otherwise*.

Per acabar l'exemple només ens queda imprimir per pantalla la informació que ens demana l'enunciat de l'exercici. Volem fer un llistat amb les taules i el nombre de comensals.

Per fer aquesta part només hem d'utilitzar les etiquetes *<c:out>* per enviar la informació al navegador.

```

1 <c:choose>
2   <c:when test="{num eq 0}">
3     <c:out value="{<li>La taula'} ${loop.index} {'està buida</li>'}"
4       escapeXml="false"></c:out>
5   </c:when>
6   <c:when test="{num eq 5}">
7     <c:out value="{<li>La taula'} ${loop.index} {' està plena</li>'}"
8       escapeXml="false"></c:out>
9   </c:when>
10  <c:otherwise>
11    <c:out value="{<li>La taula'} ${loop.index} {' té ' } ${num} {'
      comensals</li>'}" escapeXml="false"></c:out>
12  </c:otherwise>
13 </c:choose>

```

Per imprimir correctament la informació veieu que s'ha utilitzat una concatenació d'expressions *EL*. Per imprimir un *string* s'han d'utilitzar les cometes simples. Dintre de l'*string* podem afegir etiquetes HTML perquè les interpreti el navegador sempre que l'atribut **escapeXML** estigui desactivat. Si no el desactivem, el navegador no interpretarà les etiquetes enviades amb l'etiqueta *<c:out*.

Dintre de l'expressió *EL* podem utilitzar la variable de l'etiqueta *foreach* i l'índex de la col·lecció per mostrar la informació a l'usuari. L'execució del codi anterior ens dóna aquesta sortida:

```

1 Gestió de taules d'un restaurant
2
3 La taula 0 està plena
4 La taula 1 té 4 comensals
5 La taula 2 té 4 comensals
6 La taula 3 està buida
7 La taula 4 està buida
8 La taula 5 té 3 comensals
9 La taula 6 està plena
10 La taula 7 té 2 comensals
11 La taula 8 té 2 comensals
12 La taula 9 té 3 comensals

```

Intenteu implementar amb etiquetes JSTL, dintre del possible, el següent exercici: creareu una funció que donat un *array* amb el nombre de comensals de cada taula ens retorni un altre *array* amb el nombre de cada taula que estigui buida. Mostrareu per pantalla totes les taules buides.

## 2.4.2 Gestió de les habitacions d'un hotel

En aquest apartat es veurà el tractament d'informació amb *arrays* de més d'una dimensió. Aprendreu com crear *arrays* multidimensionals, com modificar els valors i com cercar-los.

Es tractarà de programar la gestió d'habitacions d'un hotel. Imagineu un hotel amb 5 plantes i 10 habitacions en cadascuna de les plantes. Es vol guardar el nombre de clients que hi ha en cada habitació, i com màxim n'hi pot haver 4.

Començarem creant un *array* de dues dimensions (matriu bidimensional) on a cada posició guardarem el nombre de clients que hi ha. Amb el primer índex de

l'*array* accedirem al pis, i amb el segon índex accedim a la habitació d'aquell pis. Per exemple, si accedim a la posició de l'*array* habitacions[0][4] estem accedint al pis 0 (planta baixa) habitació 4.

Doncs bé, igual que abans amb el restaurant, inicialitzarem l'*array* de manera aleatòria amb el nombre de clients per habitació. Una vegada tinguem la matriu preparada mostrarem per pantalla la informació.

```

1  <%@page contentType="text/html" pageEncoding="UTF-8"%>
2  <%!
3      private static final int NUM_PLANTES = 5;
4      private static final int NUM_HAB = 10;
5      private static final int MAX_CLIENTS = 5;
6      private int rand(int min, int max) {
7          return (int) (Math.random() * (max + 1 - min) + min);
8      }
9
10 %>
11 <!DOCTYPE html>
12 <html>
13     <head>
14         <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
15         <title>Gestió de les habitacions d'un hotel</title>
16     </head>
17     <body>
18         <H1>Gestió de les habitacions d'un hotel</H1>
19         <%
20             //inicialització de les taules amb els comensals
21             int[][] hotel = new int[NUM_PLANTES][NUM_HAB];
22             for (int pis = 0; pis < NUM_PLANTES; pis++) {
23                 for (int hab = 0; hab < NUM_HAB; hab++) {
24                     hotel[pis][hab] = rand(0, MAX_CLIENTS);
25                 }
26             }
27         %>
28         <ul>
29             <%
30                 //Llistat amb el nombre de clients per habitacions
31                 for (int pis = 0; pis < NUM_PLANTES; pis++) {
32                     for (int hab = 0; hab < NUM_HAB; hab++) {
33                         out.println("<li>A l'habitació " + hab + " de la planta " +
34                             pis + " hi ha " + hotel[pis][hab] + " clients</li>");
35                     }
36                 }
37             %>
38         </ul>
39     </body>
40 </html>

```

Una possible execució del codi anterior és la següent:

```

1  Gestió de les habitacions d'un hotel
2
3  A l'habitació 0 de la planta 0 hi ha 0 clients
4  A l'habitació 1 de la planta 0 hi ha 3 clients
5  A l'habitació 2 de la planta 0 hi ha 5 clients
6  A l'habitació 3 de la planta 0 hi ha 5 clients
7  A l'habitació 4 de la planta 0 hi ha 2 clients
8  A l'habitació 5 de la planta 0 hi ha 5 clients
9  A l'habitació 6 de la planta 0 hi ha 3 clients
10 A l'habitació 7 de la planta 0 hi ha 4 clients
11 A l'habitació 8 de la planta 0 hi ha 1 clients
12 A l'habitació 9 de la planta 0 hi ha 4 clients
13
14 ...
15
16 A l'habitació 0 de la planta 4 hi ha 2 clients

```

```

17 A l'habitació 1 de la planta 4 hi ha 3 clients
18 A l'habitació 2 de la planta 4 hi ha 4 clients
19 A l'habitació 3 de la planta 4 hi ha 0 clients
20 A l'habitació 4 de la planta 4 hi ha 2 clients
21 A l'habitació 5 de la planta 4 hi ha 1 clients
22 A l'habitació 6 de la planta 4 hi ha 5 clients
23 A l'habitació 7 de la planta 4 hi ha 2 clients
24 A l'habitació 8 de la planta 4 hi ha 5 clients
25 A l'habitació 9 de la planta 4 hi ha 2 clients

```

Fixem-nos en la declaració de les constants. A JSP s'han de declarar a l'espai de declaracions, és a dir, dintre de les etiquetes `<%! .... %>`. Les constants es declaren utilitzant les paraules reservades **static** i **final**, tot i que només amb **final**, en el nostre cas, ja ens serviria.

```

1 <%!
2     private static final int NUM_PLANTES = 5;
3     private static final int NUM_HAB = 10;
4     private static final int MAX_CLIENTS = 5;
5     private int rand(int min, int max) {
6         return (int) (Math.random() * (max + 1 - min) + min);
7     }
8 %>

```

La diferència entre posar **static** i no posar-ho és que les variables estàtiques (*static*) pertanyen a la classe, i en les no estàtiques hi ha una per a cada objecte de la classe. La paraula reservada **final** significa que el valor de la variable no canviarà, és a dir, serà constant.

A l'hora de recórrer i de crear la matriu veiem que no hi ha sorpreses. Per crear la matriu utilitzem la instrucció:

```

1 int[][] hotel = new int[NUM_PLANTES][NUM_HAB];

```

Per crear la matriu hem de dir la mida que tindran els dos índexs. I per recórrer la matriu utilitzem dos bucles de tipus **for**.

```

1 //Llistat amb el nombre de clients per habitacions
2 for (int pis = 0; pis < NUM_PLANTES; pis++) {
3     for (int hab = 0; hab < NUM_HAB; hab++) {
4         out.println("<li>A l'habitació " + hab + " de la planta " + pis + " hi
5             ha " + hotel[pis][hab] + " clients</li>");
6     }
7 }

```

Per acabar, volem un llistat amb l'ocupació de les habitacions de l'hotel i una funció que ens indiqui si hi ha habitacions lliures. Penseu com fer la funció mentre traïem el llistat amb l'ocupació de l'hotel. Per determinar-la es vol un llistat amb el nombre d'habitacions ocupades per planta.

```

1 private int[] ocupacio(int[][] hotel){
2     int [] ocupades = new int[NUM_PLANTES];
3     int num = 0;
4     for (int pis = 0; pis < NUM_PLANTES; pis++) {
5         for (int hab = 0; hab < NUM_HAB; hab++) {
6             if(hotel[pis][hab] != 0){
7                 num++;
8             }
9         }
10    }

```

```

10     ocupades[pis] = num;
11     num=0;
12 }
13 return ocupades;
14 }

```

Amb la funció anterior es determina quantes habitacions tenen com a mínim un client a cada planta. Utilitzem la variable *num* com a comptador del nombre d'habitacions plenes. Cada vegada que recorrem una planta tornem a posar a 0 el comptador. Finalment, la funció retorna un *array* on cada posició correspon a la planta, i el valor de la posició al nombre d'habitacions que estan ocupades.

Una manera de llistar l'*array* resultant de l'execució de la funció *ocupació* podria ser:

```

1 //Llistat amb l'ocupació de l'hotel
2 int planta = 0;
3 for (int num : ocupacio(hotel)) {
4     switch(num){
5         case 0:
6             out.println("<li>A la planta " + planta + " estan totes les
7                 habitacions buides</li>");
8             break;
9         case NUM_HAB:
10            out.println("<li>A la planta " + planta + " estan totes les
11                habitacions plenes.</li>");
12            break;
13        default:
14            out.println("<li>A la planta " + planta + " hi ha " + num + "
15                habitacions ocupades.</li>");
16    }
17    planta++;
18 }

```

Hem utilitzat l'estructura del bucle *foreach* perquè és molt més neta. Queda molt clar el recorregut que es fa amb l'*array*. Però com que amb aquesta estructura no podem accedir a l'índex de l'element actual a cada volta del bucle, hem de crear-nos el nostre propi índex. Aquest índex només el voldrem per donar la informació de la planta actual a l'usuari.

Un possible resultat de l'execució del codi anterior, quan afegim les funcions anteriors al codi del hotel, seria:

```

1 Gestió de les habitacions d'un hotel
2
3 A la planta 0 estan totes les habitacions buides.
4 A la planta 1 hi ha 9 habitacions ocupades.
5 A la planta 2 hi ha 9 habitacions ocupades.
6 A la planta 3 estan totes les habitacions plenes.
7 A la planta 4 hi ha 8 habitacions ocupades

```

Per acabar, codificarem la funció que determina si existeix alguna habitació buida a l'hotel.

```

1 private boolean habBuides(int[][] hotel) {
2     boolean trobat = false;
3     int pis = 0, porta = 0;
4     while (!trobat && pis < NUM_PLANTES) {
5         if (hotel[pis][porta] == 0) {
6             trobat = true;

```

```

7      }
8      if (porta == NUM_HAB - 1) {
9          porta = 0;
10         pis++;
11     } else {
12         porta++;
13     }
14 }
15 return trobat;
16 }

```

Per mostrar el resultat de la funció anterior a l'usuari s'ha optat per utilitzar un operador ternari dintre de les etiquetes `<%= ... %>`.

```

1 <h2><%= habBuides(hotel)? "Almenys hi ha una habitació lliure." : "No
   existeixen habitacions lliures." %></h2>

```

La funció per recórrer la matriu és la mateixa que la que hem vist a l'apartat de PHP, únicament s'ha adaptat al llenguatge Java.

Un exemple de l'execució del codi anterior seria el següent:

```

1 Gestió de les habitacions d'un hotel
2
3 A la planta 0 estan totes les habitacions plenes.
4 A la planta 1 hi han 9 habitacions ocupades.
5 A la planta 2 estan totes les habitacions plenes.
6 A la planta 3 hi han 9 habitacions ocupades.
7 A la planta 4 hi han 9 habitacions ocupades.
8 Almenys hi ha una habitació lliure.

```

Ja hem vist com utilitzar una matriu amb la sintaxi estàndard de JSP. Ara veurem la traducció d'alguns exemples a la sintaxi JSTL, i farem el primer exemple. Es vol llistar quants clients hi ha en cada habitació de l'hotel.

```

1 <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
2 <%@page contentType="text/html" pageEncoding="UTF-8"%>
3 <%!
4     private static final int NUM_PLANTES = 5;
5     private static final int NUM_HAB = 10;
6     private static final int MAX_CLIENTS = 5;
7     private int rand(int min, int max) {
8         return (int) (Math.random() * (max + 1 - min) + min);
9     }
10 }
11 %>
12 <!DOCTYPE html>
13 <html>
14     <head>
15         <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
16         <title>Gestió de les habitacions d'un hotel</title>
17     </head>
18     <body>
19         <H1>Gestió de les habitacions d'un hotel</H1>
20         <%
21             //inicialització de les taules amb els comensals
22             int[][] hotel = new int[NUM_PLANTES][NUM_HAB];
23             for (int pis = 0; pis < NUM_PLANTES; pis++) {
24                 for (int hab = 0; hab < NUM_HAB; hab++) {
25                     hotel[pis][hab] = rand(0, MAX_CLIENTS);
26                 }
27             }
28             request.setAttribute("hotel", hotel);
29             //Llistat amb el nombre de clients per habitacions

```

```

30      %>
31      <ul>
32          <c:forEach var="planta" varStatus="numplanta" items="${hotel}">
33              <c:forEach var="clients" varStatus="numporta" items="${planta}"
34                  ">
35                  <c:out value="${<li>La habitació' } ${numporta.index} ${'de
36                      la planta'} ${numplanta.index} ${'té'} ${clients} ${'
37                          clients</li>'}" escapeXml="false"></c:out>
38                  </c:forEach>
39              </c:forEach>
40      </ul>
41      </body>
42  </html>

```

En aquesta ocasió s'ha enregistrat a l'objecte *request* la matriu bidimensional *hotel*, que està inicialitzada amb tots els clients que hi ha a l'hotel. Es deixa a les etiquetes JSTL el llistat de la informació que conté.

Per poder iterar l'*array* calen dos bucles *foreach*. El primer bucle ens dóna l'*array* corresponent a les habitacions que s'ha de recórrer al bucle següent.

```

1  <c:forEach var="planta" varStatus="numplanta" items="${hotel}">
2  ...
3  </c:forEach>

```

La matriu *hotel* s'assigna a l'atribut *items* de l'etiqueta *foreach*. A cada iteració omplirà la variable *planta* amb l'*array* de clients per habitació que hi ha en aquella planta de l'hotel. A la variable *numplanta* tenim el número d'iteració actual, és a dir, el número de planta actual.

```

1  <c:forEach var="planta" varStatus="numplanta" items="${hotel}">
2      <c:forEach var="clients" varStatus="numporta" items="${planta}">
3          ....
4      </c:forEach>
5  </c:forEach>

```

Així, al bucle intern iterem sobre l'*array planta* i accedim al nombre de clients de cada habitació de la planta amb la variable *clients*. La variable *numporta* ens dóna la informació sobre el número d'habitació actual.

Amb totes aquestes dades ja podem mostrar per pantalla la informació que ens demana l'enunciat:

```

1  <c:out value="${<li>La habitació' } ${numporta.index} ${'de la planta'} ${
2      numplanta.index} ${'té'} ${clients} ${'clients</li>'}" escapeXml="false"
3  ></c:out>

```

Utilitzant l'etiqueta *<c:out* i amb expressions *EL* podem enviar al navegador la informació demanada. El resultat de l'execució del codi anterior seria el següent:

```

1  Gestió de les habitacions d'un hotel
2
3  L'habitació 0 de la planta 0 té 4 clients
4  L'habitació 1 de la planta 0 té 0 clients
5  L'habitació 2 de la planta 0 té 1 clients
6  L'habitació 3 de la planta 0 té 3 clients
7  L'habitació 4 de la planta 0 té 0 clients
8  L'habitació 5 de la planta 0 té 3 clients
9  L'habitació 6 de la planta 0 té 2 clients

```



```
10 L'habitació 7 de la planta 0 té 3 clients
11 L'habitació 8 de la planta 0 té 4 clients
12 L'habitació 9 de la planta 0 té 3 clients
13 ...
14 L'habitació 0 de la planta 4 té 0 clients
15 L'habitació 1 de la planta 4 té 3 clients
16 L'habitació 2 de la planta 4 té 2 clients
17 L'habitació 3 de la planta 4 té 5 clients
18 L'habitació 4 de la planta 4 té 4 clients
19 L'habitació 5 de la planta 4 té 2 clients
20 L'habitació 6 de la planta 4 té 4 clients
21 L'habitació 7 de la planta 4 té 1 clients
22 L'habitació 8 de la planta 4 té 4 clients
23 L'habitació 9 de la planta 4 té 5 clients
```

Intenteu traduir a JSTL, dintre de les possibilitats del llenguatge, l'ocupació de l'hotel tal com hem fet anteriorment. En concret, volem un llistat amb l'ocupació de les habitacions de l'hotel i una funció que ens indiqui si hi ha habitacions lliures.

## 2.5 Què s'ha après?

En aquest apartat l'alumne ha après:

- Les nocions bàsiques dels llenguatges: PHP, JSP i JSTL.
- Crear petites aplicacions amb els llenguatges anteriors.
- Utilització dels *arrays* per emmagatzemar un conjunt de dades.
- Crear i utilitzar matrius bidimensionals.

Per aprofundir en aquests llenguatges es recomana la realització de les activitats associades a aquest apartat. Una vegada fetes, l'alumne estarà preparat per endinsar-se en la programació amb Java Servlets.