

Universitatea *Transilvania* din Braşov  
Facultatea de Matematică–Informatică

ERNEST SCHEIBER

**PROGRAMARE  
DISTRIBUITĂ ÎN JAVA**

Braşov



# Prefață

Scopul acestui curs este prezentarea tehnologiilor de programare Java care permit realizarea aplicațiilor client - server:

- socluri Java;
- apelarea metodelor de la distanță
  - *Remote Method Invocation - RMI*;
  - *Common Object Request Broker Architecture - CORBA*;
- mesageria Java;
- servlet;
- *Java Server Pages - JSP*;
- *Java Web Start*;
- *WebSocket*;
- *Enterprise Java Bean*.

Accentul cade pe detaliile tehnice de realizare a comunicațiilor și pe arhitectura programelor / aplicațiilor care le utilizează. Trebuie semnalat faptul că exemplele date nu încorporează aspecte indispensabile unei aplicații informatice la standardele zilei:

- securitate, autentificare și autorizare;
- interfață grafică pentru componenta client;
- utilizarea bazelor de date relaționale / orientate obiect / NoSQL.

Metodele și instrumentele de programare vor fi exemplificate, de cele mai multe ori, pe problema foarte simplă de calcul a celui mai mare divizor comun a două numere naturale. Codul acestei metode de calcul poate fi

#### **Varianta imperativă ca metodă**

```
1 public long cmmdc(long m, long n){
2     long r, c;
3     do{
4         c=n;
5         r=m % n;
6         m=n;
7         n=r;
8     }
9     while (r!=0);
10    return c;
11 }
```

care se poate transforma într-o lambda expresie

```
1 interface CmmdcService {
2     long cmmdc(long m, long n);
3 }
4
5 static CmmdcService cmmdcService=(long m, long n) -> {
6     long r, c;
7     do{
8         c=n;
9         r=m % n;
10        m=n;
11        n=r;
12    }
13    while (r!=0);
14    return c;
15 };
```

#### **Varianta declarativă / recursivă**

```
1 public long cmmdc(long m, long n){
2     if(m==n)
3         return m;
4     else
5         if(m>n)
6             return cmmdc(m-n, n);
7         else
8             return cmmdc(m, n-m);
9 }
```

*Guava: Google Core Libraries for Java* posedă o metoda de calcul a celui mai mare divizor a două numere naturale `com.google.common.math.LongMath.gcd(...)`.

Pentru aplicațiile care utilizează o bază de date, sistemul de gestiune a bazei de date (SGBD) va fi una dintre sistemele *Derby/Javadb* sau *mysql*.

**Tiparul de învățare** propus este

1. Se instalează toate resursele necesare (Se exemplifică la laborator).

2. Se execută aplicația / aplicațiile din curs (Se exemplifică la laborator).
3. Pentru fiecare tehnologie, pe suportul oferit de curs, se programează o altă aplicație.

Propunem următoarele teme:

- Conversia dintre grade Celsius și grade Fahrenheit ( $F = 1.8C + 32$ ).
  - Crearea, întreținerea și utilizarea unei agende de adrese de e-mail. Agenda este o bază de date.
4. În final, se rezolvă tema pentru examen.

Nu de puține ori metodele / tehnologiile utilizate presupun utilizarea unui șablon de programare specific. Din acest punct de vedere, acest curs se dorește a fi un suport metodic.

Utilizând sistemul de operare MS Windows propunem utilizarea următoarelor produse informatice auxiliare:

- *Google Chrome* – Navigator
- *Notepad++* – Editor de fișiere
- *MultiCommander* – Gestionar de fișiere

Probleme:

- http2
- Docker, Vagrant, Kubernetes
- JSR 371 mvc
- Dukescript - knockout.js



# Produsele informatice utilizate

Pe durata existenței, produsele informatice evoluează prin versiunile pe care producătorii ni le pun la dispoziție. Nu de puține ori o versiune nouă nu este compatibilă cu versiunea anterioară, fapt care necesită adaptarea programelor la modificările operate.

Lista următoare precizează versiunile produselor utilizate în curs, indicate în majoritatea cazurilor prin resursa de instalare.

Versiunile produselor informatice utilizate în lucrare		
No.	Produsul informatic	Resursa/versiunea
1	apache-ant	apache.ant-1.9.6-bin.tar.gz
2	apache-commons-fileupload	commons-fileupload-1.3.1-bin.tar.gz
3	apache-ftpserver	ftpserver-1.0.6.tar.gz
4	apache-maven	apache-maven-3.3.9-bin.tar.gz
5	apache-shiro	apache-shiro-1.2.4
6	apache-tomcat	apache-tomcat-8.0.30.tar.gz
7	apacheds	apacheds-2.0.0-M20.tar.gz
8	glassfish	java_ee_sdk_7u2.zip, glassfish-4.1.1.zip
9	google appengine	appengine-java-sdk-1.9.28.zip
10	db-derby	db-derby-10.11.1.1-bin.tar.gz
11	httpcomponents-client	httpcomponents-client-4.5.1-bin.tar.gz
12	httpcomponents-asyncclient	httpcomponents-asyncclient-4.1-bin.tar.gz
13	Java (jdk)	jdk-8u66-windows-{i586;x64}.exe
14	junit	junit4.12.zip
15	mysql	mysql-5.7.10-win{32;x64}.zip
16	mysql-connector	mysql-connector-java-5.1.37.tar.gz
17	openmq	openmq5_1_1-binary-windows.zip





# Cuprins

<b>1</b>	<b>Introducere</b>	<b>15</b>
<b>I</b>	<b>TEHNOLOGII PENTRU REȚELE LOCALE</b>	<b>21</b>
<b>2</b>	<b>Programare cu socluri Java</b>	<b>23</b>
2.1	TCP vs. UDP . . . . .	23
2.2	Soclu TCP . . . . .	24
2.2.1	Aplicație client – server cu socluri . . . . .	25
2.2.2	Streaming . . . . .	30
2.3	Datagrame . . . . .	33
2.3.1	Aplicații client – server cu datagrame. . . . .	38
2.3.2	Multicast vs. Broadcast . . . . .	43
2.4	Canale de comunicație . . . . .	44
2.4.1	SocketChannel & DatagramChannel . . . . .	44
2.4.2	Recepție cu Selector . . . . .	58
2.4.3	Comunicații asincrone prin canale . . . . .	61
<b>3</b>	<b>Regăsirea obiectelor prin servicii de nume</b>	<b>69</b>
3.1	<i>Java Naming and Directory Interface</i> . . . . .	69
3.1.1	LDAP . . . . .	73
<b>4</b>	<b>Invocarea procedurilor la distanță</b>	<b>77</b>
4.1	Remote Method Invocation . . . . .	77
4.1.1	Crearea unei aplicații RMI . . . . .	81
4.1.2	Tipare de programare . . . . .	87
4.1.3	Obiect activabil la distanță . . . . .	93
4.2	CORBA . . . . .	98
4.2.1	Conexiunea RMI - CORBA . . . . .	98
4.2.2	Aplicație Java prin CORBA . . . . .	104

<b>5</b>	<b>Mesaje în Java</b>	<b>113</b>
5.1	Java Message Service (JMS)	113
5.2	<i>Open Message Queue 5</i>	115
5.3	Elemente de programare - JMS-2	116
5.3.1	Modul programat: Trimiterea unui mesaj	116
5.3.2	Recepția sincronă a unui mesaj	118
5.3.3	Recepția asincronă a unui mesaj	120
5.3.4	Publicarea mesajelor	122
5.3.5	Abonare și recepția mesajelor	123
5.3.6	Cazul abonatului partajat	124
5.3.7	Obiecte administrator prin JNDI	127
5.3.8	Comunicația prin coadă - queue	128
5.3.9	Comunicația pe bază de subiect - topic	131
5.3.10	Utilizarea mesajelor de tip <i>StreamMessage</i>	133
5.3.11	Aplicație JMS slab cuplată	136
5.3.12	Programare JMS prin <i>glassfish</i>	139
<b>II</b>	<b>COMUNICAȚII PRIN INTERNET</b>	<b>143</b>
<b>6</b>	<b>HyperText Transfer Protocol</b>	<b>145</b>
6.1	Transacție http	145
6.2	Server Web - container de servlet	152
6.3	Serverul Web <i>apache-tomcat</i>	153
6.4	Glassfish	155
<b>7</b>	<b>Conexiune simplă prin clase din java.net</b>	<b>157</b>
7.1	Clasa <i>java.net.URL</i>	157
<b>8</b>	<b>Servlet</b>	<b>159</b>
8.1	Marcajul <i>&lt;form&gt;</i>	160
8.2	Realizarea unui servlet	161
8.2.1	Codul unui servlet	164
8.3	Procesare asincronă în servlet	171
8.4	Dezvoltări în <i>servlet-api 3.1</i>	176
8.4.1	Procesare asincronă neblokantă	176
8.4.2	Modificarea protocolului <i>http: upgrade</i>	179
8.5	Facilități de programare cu servlet	185
8.5.1	Program client al unui servlet	185
8.5.2	Servlete înlănțuite	193

8.5.3	Sesiune de lucru . . . . .	195
8.5.4	Cookie . . . . .	197
8.5.5	Gestiunea butoanelor - <i>TimerServlet</i> . . . . .	199
8.5.6	Autentificare . . . . .	203
8.5.7	Servlet cu conexiune la o bază de date . . . . .	204
8.5.8	Imagini furnizate de servlet . . . . .	207
8.5.9	Servlet cu RMI . . . . .	209
8.5.10	Servlet cu JMS . . . . .	210
8.5.11	Servlet cu jurnalizare . . . . .	213
8.6	FileUpload . . . . .	214
8.7	Descărcarea unui fișier . . . . .	219
8.8	Filtru . . . . .	220
8.9	Eveniment și auditor . . . . .	223
8.10	Server <i>apache-tomcat</i> încorporat . . . . .	224
<b>9</b>	<b>Java Server Page – JSP</b>	<b>227</b>
9.1	Tehnologia JSP . . . . .	227
9.1.1	Declarații JSP . . . . .	232
9.1.2	Directive JSP . . . . .	234
9.1.3	Marcaje JSP predefinite . . . . .	235
9.1.4	Pagini JSP cu componente Java . . . . .	236
9.2	JSP Standard Tag Library JSTL . . . . .	239
9.2.1	Biblioteca de bază . . . . .	240
9.2.2	Biblioteca de lucru cu baze de date . . . . .	245
9.3	Marcaje JSP personale . . . . .	246
9.3.1	Marcaje fără attribute și fără corp. . . . .	247
9.3.2	Marcaje cu attribute și fără corp. . . . .	250
9.3.3	Marcaje cu corp. . . . .	252
9.4	Autentificare și autorizare cu <i>apache-shiro</i> . . . . .	254
<b>10</b>	<b>Desfășurarea în <i>nor</i></b>	<b>263</b>
10.1	Servlet și JSP în <i>Google App Engine</i> . . . . .	264
10.2	<i>Heroku</i> . . . . .	266
10.2.1	JSP în <i>Heroku</i> . . . . .	267
10.2.2	Servlet în <i>Heroku</i> . . . . .	270
10.3	<i>OpenShift</i> . . . . .	273
<b>11</b>	<b>Java Web Start</b>	<b>277</b>
11.1	Java Web Start . . . . .	277

<b>12 WebSocket</b>	<b>283</b>
12.1 Protocolul WebSocket . . . . .	283
12.2 Interfața de programare HTML5 de client WebSocket . . . . .	284
12.3 WebSocket în Java . . . . .	285
12.3.1 Programare prin adnotări . . . . .	286
12.3.2 Programare fără adnotări . . . . .	291
12.3.3 Client Java pentru WebSocket . . . . .	293
12.3.4 Transmiterea datelor prin adnotarea <b>PathParam</b> . . . . .	294
12.3.5 Conversie și deconversie prin JSON / XML . . . . .	297
12.3.6 Streaming . . . . .	304
<b>13 Enterprise Java Beans</b>	<b>311</b>
13.1 Session EJB . . . . .	312
13.1.1 Componentă EJB sesiune stateless . . . . .	313
13.1.2 Componentă cu metode asincrone . . . . .	316
13.1.3 Aplicație JEE cu module EJB, Web și client RMI-IIOP . . . . .	317
13.1.4 Componentă EJB sesiune singleton . . . . .	319
13.1.5 Componentă EJB sesiune stateful . . . . .	321
13.2 Componentă EJB <i>MessageDriven</i> . . . . .	324
13.3 Componentă EJB <i>Entity</i> . . . . .	327
<b>III ANEXE</b>	<b>333</b>
<b>A Unelte de dezvoltare</b>	<b>335</b>
A.1 XML . . . . .	335
A.2 <i>apache-ant</i> . . . . .	338
A.3 <i>apache-maven</i> . . . . .	340
<b>B Fir de execuție prin <math>\lambda</math>-expresie</b>	<b>355</b>
<b>C Testare cu <i>junit</i></b>	<b>359</b>
<b>D Jurnalizare</b>	<b>363</b>
<b>E Componentă Java</b>	<b>367</b>
<b>F Serializare fără XML</b>	<b>369</b>
F.1 YAML Ain't Markup Language - YAML . . . . .	370
F.2 JavaScript Object Notation - JSON . . . . .	372

---

<b>G</b>	<b>Adnotări</b>	<b>379</b>
G.1	Definirea unei adnotări . . . . .	379
G.2	Declararea unei adnotări . . . . .	380
G.3	Procesarea unei adnotări . . . . .	381
<b>H</b>	<b>Utilizarea SGBD în Java</b>	<b>385</b>
H.1	<i>Derby / Javadb</i> . . . . .	385
H.2	<i>mysql</i> . . . . .	386
H.3	Șablonul de utilizare a unei baze de date . . . . .	388
<b>I</b>	<b>Injectarea dependențelor</b>	<b>395</b>
I.1	Weld . . . . .	395
<b>IV</b>	<b>TEME DE LABORATOR</b>	<b>399</b>
<b>J</b>	<b>Teme de aplicații</b>	<b>401</b>
J.1	Probleme propuse . . . . .	401
	<b>Bibliografie</b>	<b>407</b>



# Capitolul 1

## Introducere

Rețelele locale, internetul, răspândirea pe o arie geografică a resurselor și a locațiilor în care se petrec acțiuni ce țin de o activitate bine definită sau sunt urmărite, gestionate din alte locuri au drept consecință existența aplicațiilor distribuite. Termenul distribuit se referă tocmai la faptul că componente ale aplicației se află pe calculatoare diferite dar între care au loc schimburi de date. Dacă părțile unei aplicații sau resursele utilizate se găsesc pe calculatoare distincte atunci aplicația se numește distribuită.

Între părțile sau resursele unei aplicații distribuite au loc schimburi de date, ceea ce se face utilizând diferite mecanisme la realizarea cărora concură sistemul de calcul, sistemul de operare și limbajul de programare.

Astfel se vorbește de programare distribuită ca mijloc de realizare a aplicațiilor distribuite. Pe lângă *algoritm*, *structuri de date*, *limbaj de programare*, la realizarea unei aplicații distribuite intervin *comunicațiile*: schimbul de date dintre două componente aflate pe calculatoare diferite.

Transmisia datelor poate fi:

- *discretă* - numărul datelor transmise este fixat;
- în flux (*streaming*) - se transmit un volum de date în mod continuu.

În cadrul comunicațiilor, transmisia de date presupune transformarea acestora din formatul tipizat în șir de octeți și invers - serializarea datelor. În acest scop tehnologiile de programare utilizează modalități diverse, de la atribuirea acestei sarcini către programator până la asigurarea unui cadru fix de realizare a serializării.

Punem în evidență două modele de aplicații distribuite:

- **client-server**: Programul server execută cererile clienților.

O aplicație client – server se compune din:

- componenta *server* - alcătuită din programe / clase ce asigură una sau mai multe funcțiuni (servicii), care pot fi apelate de către clienți.
- componenta *client* - alcătuită din programe / clase care permit accesul la server și apelarea serviciilor acestuia.

Serverul și clientul (clienții) rulează, de obicei, pe calculatoare distincte. Un server trebuie să satisfacă cererile mai multor clienți.

Durata de viață a unei aplicații client – server este dată de durata funcționării serverului. Această durată poate fi alcătuită din intervale disjuncte de timp, între acele intervale, din diverse motive, serverul este inactiv.

Intervalul de timp determinat de conectarea unui client la server și până la deconectare poartă numele de sesiune. În această perioadă, clientul poate invoca mai multe servicii ale serverului. Un client poate iniția mai multe sesiuni.

Un client trebuie să-și regăsească datele în cadrul unei sesiuni, între sesiuni, pe toată durata de viață a aplicației. Astfel se pune problema reținerii / persistenței datelor pentru fiecare client în parte.

Amintim următoarele tehnologii Java pentru realizarea aplicațiilor client-server:

- Tehnologii destinate rețelelor locale, nebazate pe protocolul **http**.
  - \* *Socket*
  - \* *Remote Method Invocation* (RMI)
  - \* *Common Object Request Broker Architecture* (CORBA)
  - \* *Java Message Service* (JMS)

Comunicațiile utilizează porturi care nu trebuie să fie închise de eventuale aplicații de tip *firewall*.

- Tehnologii cu comunicații prin Internet, bazate pe protocolul **http**.
  - \* *Servlet* și *Java Server Pages* (JSP)
  - \* *Java Web Start*
  - \* *WebSocket*

- **dispecer-lucrător**: Programul dispecer distribuie sarcinile de executat lucrătorilor și le coordonează activitatea. Există multe abordări de programare a aplicațiilor dispecer-lucrător.



Există diferențe mari între o aplicație care rulează pe un calculator și o aplicație distribuită<sup>1</sup>:

- latența (*latency*) - există mai multe definiții:
  - diferența în timp între o operație executată pe un calculator la distanță de execuția ei pe calculatorul local.
  - diferența în timp între momentul recepționării răspunsului la o cerere și momentul lansării ei.
  - diferența în timp între momentul recepționării unei cereri și momentul transmiterii răspunsului.

Recepția rezultatului unei operații executate pe un calculator la distanță se poate programa

- sincron – de obicei recepția blochează firul de execuție al apelului până la sosirea rezultatului;
- asincron – concept care are mai multe realizări:
  - \* recepția se obține într-un obiect dedicat care se execută în afara firului de execuție al apelului.  
(JMS, servlet)
  - \* rezultatul solicitării este un obiect de tip **Future**< *T* > și a cărui procesare se poate face după instanțierea obiectului *T*.  
(socluri cu canale de comunicații, client servlet)

O preocupare continuă este dezvoltarea de tehnologii hard și soft pentru micșorarea latenței.

- accesul la memorie (*memory access*). Instrumente de programare - cadre de lucru (*framework*) care mijlocesc realizarea aplicațiilor asigură accesul la resursele aflate în memoria calculatoarelor (*read, write, send, receive*).
- interoperabilitate. Aplicațiile distribuite pot rula pe platforme diferite (de exemplu Linux și Windows sau Java și .NET) iar componentele ei pot fi realizate în limbaje de programare diferite.

Există o preocupare pentru produse care asigură interoperabilitatea dintre platformele de calcul.

---

<sup>1</sup>Waldo J., Wyant G., Wollrath A., Kendall S., 1994, *A Note on Distributed Computing*. Sun Microsystems Corporation, Technical report, SMLI TR-94-29.

- prăbușirea parțială (*partial failure*) constă în încetarea funcționării unei părți a aplicației distribuite, dar care continuă să fie accesibilă. Tratarea acestei probleme pare a fi cea mai dificilă temă a programării distribuite. Teorema CAP (Eric Brewer, 2000) tratează legătura dintre prăbușirea parțială, disponibilitatea aplicației și consistența datelor.

Precizăm sensul unor noțiuni utilizate în lucrare:

- *protocol* - pachet de reguli, șablon utilizat în comunicații, în accesarea unor resurse. Exemple de protocoale utilizate sunt:
  - *http* - *HyperText Transfer Protocol* este principalul protocol utilizat în comunicațiile prin Internet;
  - *https* protocol *http* securizat;
  - *file* - protocol pentru specificarea fișierelor aflate pe calculatorul local;
  - *ftp* - *File Transfer Protocol* - protocol pentru transferul fișierelor între două calculatoare;
  - *smtp* - *Simple Message Transfer Protocol* utilizat de poșta electronică.
- *host* - calculatorul gazdă, cel pe care se lucrează. Acest calculator se specifică printr-o adresă IP (*Internet Protocol*) sau printr-un nume.
- *port* - un număr care identifică aplicația responsabilă de prelucrarea solicitării. Implicit, anumitor protocoale le sunt asociate porturi. Dintre acestea amintim:

Port	Utilizat de
80	http
443	https
25	smtp

Referințele resurselor se indică folosind *Uniform Resource Identifiers* - (URI) și mai precis *Uniform Resource Locator* - (URL). URI identifică o resursă în timp ce URL desemnează locația resursei. URL se consideră ca un caz particular de URI. Sintaxa folosită pentru URI este

`protocol://host[:port][cale][?cerere]`

Un rol important în dezvoltarea tehnologiilor legate de limbajul Java revine organizațiilor de standardizare:

- *Java Community Process* JCP;
- *Organization for the Advancement of Structured Information Standards* OASIS;
- *The Internet Engineering Task Force* IETF.

## Întrebări recapitulative

1. Explicați noțiunea de latență.
2. Explicați termenul de protocol și dați exemple de protocoale de comunicație.
3. Explicați noțiunea de prăbușire parțială.
4. Ce desemnează denumirea *Java Community Process*?
5. Precizați sintaxa unui URI.



**Partea I**

**TEHNOLOGII PENTRU**  
**REȚELE LOCALE**



## Capitolul 2

# Programare cu socluri Java

Comunicația bazată pe socluri Java constituie modalitatea de nivel inferior pentru realizarea aplicațiilor distribuite. Din punct de vedere fizic un soclu este o interfață de conectare la procesor. Soclurile se află pe placa de bază. Noțiunea de soclu dintr-un mediu de programare este diferită de soclul fizic. În cele ce urmează ne interesează soclul în mediul Java.

### 2.1 TCP vs. UDP

Calculatoarele ce rulează în rețea comunică între ele folosind protocolul TCP (Transmission Control Protocol) sau UDP (User Datagram Protocol).

Într-un program Java se utilizează clasele pachetului `java.net` prin intermediul cărora se accesează nivelele deservite de protocoalele TCP sau UDP. În felul acesta se pot realiza comunicații independente de platforma de calcul. Pentru a alege clasa Java care să fie utilizată trebuie cunoscută diferența dintre TCP și UDP.

**TCP** Când două aplicații comunică între ele se stabilește o conexiune prin intermediul căreia se schimbă date. Folosind protocolul TCP, comunicația garantează că datele trimise dintr-un capăt ajung în celălalt capăt cu păstrarea ordinii în care au fost trimise. Acest tip de comunicație seamănă cu o convorbire telefonică. TCP furnizează un canal sigur de comunicație între aplicații.

**UDP** Utilizarea protocolului UDP presupune trimiterea unor pachete de date – numite datagrame – de la o aplicație la alta fără să se asigure faptul că datagramele ajung la destinație și nici ordinea lor de sosire. Acest tip de comunicație seamănă cu trimiterea scrisorilor prin poștă.

## 2.2 Soclu TCP

Pentru a comunica utilizând TCP programul client și programul server stabilesc o conexiune sigură. Fiecare program se leagă la conexiune printr-un soclu (socket). Un soclu este capătul unei căi de comunicație bidirecțional între două programe ce rulează în rețea. Un soclu este legat de un port – prin care nivelul TCP poate identifica aplicația căreia îi sunt transmise datele.

Pentru a comunica atât clientul cât și serverul citesc date de la și scriu date la soclul legat la conexiunea dintre ele.

În pachetul `java.net` clasele `Socket` și `ServerSocket` implementează un soclu din partea clientului și respectiv din partea serverului.

Clientul cunoaște numele calculatorului pe care rulează serverul cât și portul la care acesta este conectat. Pentru stabilirea conexiunii, clientul încearcă un rendez-vous cu serverul de pe mașina serverului și la portul serverului. Dacă totul decurge bine, serverul acceptă conexiunea. După acceptare, serverul crează pentru client un nou soclu legat la un alt port în așa fel încât ascultarea cererilor la soclul inițial să poată continua în timp ce sunt satisfăcute cererile clientului conectat. Din partea clientului, după acceptarea conexiunii soclul este creat și este utilizat pentru comunicația cu serverul.

### Clasa `java.net.Socket`

Resursele clasei `Socket` sunt destinate clientului.

#### Constructori

- `public Socket(String host, int port) throws UnknownHostException, IOException`

Crează un soclu conectat la calculatorul cu portul specificat.

- `public Socket(InetAddress host, int port) throws IOException`

Crează un soclu conectat la calculatorul cu portul specificat.

#### Metode

- `public InputStream getInputStream() throws IOException`

Returnează un flux de intrare atașat soclului, pentru citirea (preluarea) informațiilor de la soclu.



- `public OutputStream getOutputStream() throws IOException`  
Returnează un flux de ieșire atașat soclului, pentru scrierea (transmiterea) informațiilor la soclu.
- `public synchronized void close() throws IOException`  
Închide soclul de referință.

## Clasa `java.net.ServerSocket`

Resursele clasei `ServerSocket` sunt destinate serverului.

### Constructori

- `public ServerSocket(int port) throws IOException`  
Crează un soclu la portul specificat. Dacă *port*=0, atunci va fi utilizat orice port disponibil. Capacitatea șirului (tamponului) de așteptare pentru cererile de conectare se fixează la valoarea implicită 50. Cererile în exces vor fi refuzate.
- `public ServerSocket(int port, int lung) throws IOException`  
În plus fixează lungimea șirului (tamponului) de așteptare.

### Metode

- `public Socket accept() throws IOException`  
Metoda blochează procesul (firul de execuție) apelant până la sosirea unei cereri de conectare și crează un soclu client prin care se va desfășura comunicarea cu solicitantul acceptat.
- `public synchronized void close() throws IOException`  
Închide soclul de referință.

### 2.2.1 Aplicație client – server cu socluri

Serverul trebuie să satisfacă simultan solicitările mai multor clienți. Fiecare client apelează programul server la același port și în consecință cererile de conectare sunt recepționate de același `ServerSocket`. Serverul recepționează apelurile secvențial. La un apel, se crează de partea serverului un soclu prin care se va face schimbul de date cu clientul. Cererile clienților pot fi satisfăcute concurrent/paralel, utilizând fire de execuție ce implementează serviciul oferit sau secvențial - în cazul unor servicii de durată scurtă.

**Exemplul 2.2.1** *Sistem client - server pentru calculul celui mai mare divizor comun a două numere naturale. Portul obiectului de tip `ServerSocket` este 7999.*

**Programul client** *CmmdcClient* se conectează la server, transmite serverului cele două numere naturale și recepționează rezultatul pe care apoi îl afișează.

În esență orice program client trebuie să execute:

1. Deschide/crează un soclu.
2. Deschide/crează fluxuri de date pentru comunicația cu serverul.
3. Transmite și recepționează date potrivit specificului aplicației (protocolului serverului). Acest pas variază de la un program client la altul.
4. Închiderea fluxurilor de date.
5. Închiderea soclului.

```

1 import java.io.DataInputStream;
2 import java.io.DataOutputStream;
3 import java.net.Socket;
4 import java.util.Scanner;

6 public class CmmdcClient {
7     public static void main(String[] args) {
8         String host="localhost";
9         int port=7999;
10        if (args.length>0)
11            host=args[0];
12        if (args.length>1)
13            port=Integer.parseInt(args[1]);
14        Scanner scanner=new Scanner(System.in);
15        long m,n,r;
16        System.out.println("m=");
17        m=scanner.nextLong();
18        System.out.println("n=");
19        n=scanner.nextLong();
20        try(Socket cmmdcSocket = new Socket(host, port);
21            DataInputStream in=new DataInputStream(cmmdcSocket.getInputStream());
22            DataOutputStream out=
23                new DataOutputStream(cmmdcSocket.getOutputStream())){
24            out.writeLong(m);
25            out.writeLong(n);
26            r=in.readLong();
27            System.out.println("Required result : "+r);
28        }
29        catch(Exception e){
30            System.err.println("Client communication error : "+e.getMessage());
31        }
32    }
33 }

```

Se presupune că programul server rulează pe calculatorul local și utilizează portul 7999. Dacă acești parametri se modifică - de exemplu serverul rulează în rețea pe calculatorul atlantis la portul 8200 - atunci la apelare transmitem acești parametri prin `java CmmdcClient atlantis 8200`

**Partea server** este alcătuită din mai multe clase:

- Clasa *MyMServer*, independentă de un serviciu anume, preia apelurile clienților și lansează satisfacerea cererii.

```

1 import java.net.ServerSocket;
2 import java.util.concurrent.ExecutorService;
3 import java.util.concurrent.Executors;
4 import java.io.IOException;

6 public class MyMServer {
7     static final int NTHREADS=100;
8     static ExecutorService exec=Executors.newFixedThreadPool(NTHREADS);

10    public static void main(String[] args){
11        int port=7999;
12        boolean listening=true;
13        AppThread appThread=new AppThread();
14        try(ServerSocket serverSocket=new ServerSocket(port)) {
15            System.out.println("The server is listening on port 7999");
16            while(listening){
17                exec.execute(appThread.action.service(serverSocket.accept()));
18            }
19        }catch (IOException e){
20            e.printStackTrace();
21        }
22    }
23 }

```

În ciclul `while`, la recepția unei solicitări de conexiune se crează și se lansează un fir de execuție a cărei metodă *run* conține acțiunile ce răspund solicitării.

- Clasa *AppThread* - fir de execuție responsabil de preluarea datelor și de transmitere a rezultatului.

```

1 import java.net.Socket;
2 import java.io.DataInputStream;
3 import java.io.DataOutputStream;
4 import java.io.IOException;

6 public class AppThread{
7     // Firul de executie lansat de server
8     interface ServerSocketAction{
9         Thread service(Socket socket);
10    }

12    static ServerSocketAction action=(Socket socket)->{
13        return new Thread()->{
14            try(DataOutputStream out =

```

```

15         new DataOutputStream(socket.getOutputStream());
16         DataInputStream in =
17         new DataInputStream(socket.getInputStream());{
18         long m=0,n=0,r;
19         m=in.readLong();
20         n=in.readLong();
21         App app=new App();
22         r=app.cmmdcService.cmmdc(m,n);
23         out.writeLong(r);
24         socket.close();
25     }
26     catch(IOException e){
27         e.printStackTrace();
28     }
29 });
30 };
31 }

```

- Clasa *App* corespunzătoare calcului celui mai mare divizor comun a două numere naturale.

```

1 public class App{
2     // Functia indeplinita de server
3     interface CmmdcService {
4         long cmmdc(long m, long n);
5     }
6
7     static CmmdcService cmmdcService=(long m, long n) -> {...}
8
9 }

```

**Rularea programelor.** Se pornește la început programul server **MyMServer** iar apoi clientul **CmmdcClient**. Clientul se poate rula de pe orice calculator al rețelei. Dacă programul client se execută pe alt calculator decât cel pe care rulează programul server, atunci la apelarea clientului trebuie precizat numele calculatorului server și eventual portul utilizat.

O altă arhitectură a aplicației este dezvoltată în continuare. Această arhitectură este mai bună în sensul că oferă o flexibilitate mai mare.

Aplicația distribuită se va compune din:

- Aplicația server alcătuită din:

– Interfață

```

1 package iserver;
2 import java.net.ServerSocket;
3 public interface IMyMServer{
4     public ServerSocket getServerSocket(int port);
5     public void myAction(ServerSocket serverSocket);
6 }

```

## – Implementarea interfeței

```

1 package server.impl;
2 import iserver.IMyMServer;
3 import java.net.ServerSocket;
4 import java.io.IOException;
5 import java.util.concurrent.ExecutorService;
6 import java.util.concurrent.Executors;

8 public class MyMServer implements IMyMServer{

10     public ServerSocket getServerSocket(int port){
11         ServerSocket serverSocket = null;
12         try{
13             serverSocket = new ServerSocket(port);
14         }
15         catch (IOException e) {
16             System.err.println("Could not listen on port: "+port);
17             System.err.println(e.getMessage());
18             System.exit(1);
19         }
20         System.out.println("ServerSocket is ready ...");
21         return serverSocket;
22     }

24     public void myAction(ServerSocket serverSocket){
25         int NTHREADS=8192;
26         ExecutorService exec=Executors.newFixedThreadPool(NTHREADS);
27         AppThread appThread=new AppThread();
28         while(true){
29             try{
30                 exec.execute(appThread.
31                     action.service(serverSocket.accept()));
32             }
33             catch(IOException e){
34                 e.printStackTrace();
35             }
36         }
37     }
38 }

```

Clasele *AppThread*, *App* sunt cele utilizate anterior.

## – Clasă de lansare a serverului

```

1 package server;
2 import java.net.ServerSocket;
3 import server.impl.MyMServer;
4 import iserver.IMyMServer;

6 public class AppServer{
7     public static void main(String[] args){
8         int port=7999;
9         if(args.length>0)
10             port=Integer.parseInt(args[0]);
11         IMyMServer myMServer=new MyMServer();
12         ServerSocket serverSocket=myMServer.getServerSocket(port);
13         myMServer.myAction(serverSocket);
14     }

```

```
15 | }
```

- Aplicația client este nemodificată.

### 2.2.2 Streaming

Prin socluri se pot transmite fluxuri de date (*streaming*). Modul de programare depinde de natura datelor ce trebuie vehiculate (text, sunet, grafică).

**Exemplul 2.2.2** *Clientul solicită serverului transmisia unui fișier text, cerere care va fi satisfăcută de server. Clientul cunoaște lista fișierelor pe care le poate trimite serverul.*

Structura aplicației este:

Client    *StreamClient.java*  
 Server   *MyMServer.java*  
           *AppThread.java*

Clasa *StreamClient*

```
1 package streamingtext;
2 import java.io.DataInputStream;
3 import java.io.DataOutputStream;
4 import java.net.Socket;
5 import java.io.IOException;
6 import java.io.EOFException;
7 import java.io.UTFDataFormatException;
8 import java.util.Scanner;
9
10 public class StreamClient{
11     public static void main(String [] args){
12         String host="localhost";
13         int port=7997;
14         System.out.println(" Alegeti fisierul :");
15         System.out.println("1 : capitol.txt");
16         System.out.println("2 : junit.tex");
17         Scanner scanner=new Scanner(System.in);
18         int noFile=scanner.nextInt();
19         try(Socket clientSocket = new Socket(host, port);
20             DataInputStream in =
21                 new DataInputStream(clientSocket.getInputStream());
22             DataOutputStream out=
23                 new DataOutputStream(clientSocket.getOutputStream())){
24             out.writeInt(noFile);
25             System.out.println(" Received : ");
26             String s;
27             while(!(s=in.readUTF()).equals("endOfFile"))System.out.println(s);
28         }
29         catch(IOException e){
30             System.err.println(" Client communication error : "+e.getMessage());
31         }
```

```

32 }
33 }

```

### Clasa *AppThread*

```

1 package streamingtext;
2 import java.net.Socket;
3 import java.io.DataInputStream;
4 import java.io.DataOutputStream;
5 import java.io.File;
6 import java.io.FileReader;
7 import java.io.BufferedReader;
8 import java.io.IOException;
9
10 public class AppThread extends Thread{
11     Socket socket=null;
12
13     public AppThread(Socket socket){
14         this.socket=socket;
15     }
16
17     public void run(){
18         String path=" ../resources/text/";
19         try(DataInputStream in =
20             new DataInputStream(socket.getInputStream());
21             DataOutputStream out=
22             new DataOutputStream(socket.getOutputStream())){
23             int noFile=in.readInt();
24             String fileName="";
25             if(noFile==1)
26                 fileName="capitol.txt";
27             else
28                 fileName="junit.tex";
29             File inputFile=new File(path+fileName);
30             FileReader fr=new FileReader(inputFile);
31             BufferedReader br= new BufferedReader(fr);
32
33             String s;
34             while((s=br.readLine())!=null) out.writeUTF(s);
35             out.writeUTF("endOfFile");
36             out.flush();
37             br.close();
38             fr.close();
39             System.out.println("Activity Finished !");
40             socket.close();
41         }
42         catch(IOException e){
43             e.printStackTrace();
44         }
45     }
46 }

```

**Exemplul 2.2.3** *Clientul solicită serverului transmisia unui fișier grafic, cerere care va fi satisfăcută de server. Clientul cunoaște lista fișierelor pe care le poate trimite serverul.*

Pe aceeași structură codurile claselor sunt

Clasa *StreamClient*

```

1 package streamingimage;
2 import java.io.DataOutputStream;
3 import java.io.InputStream;
4 import java.awt.image.BufferedImage;
5 import java.net.Socket;
6 import java.io.IOException;
7 import javax.imageio.ImageIO;
8 import java.awt.Image;
9 import java.util.Scanner;

11 public class StreamClient{
12     public static void main(String [] args){
13         String host="localhost";
14         int port=7998;
15         System.out.println(" Alegeti fisierul :");
16         System.out.println("1 : xml-pic.jpg");
17         System.out.println("2 : brasov.jpg");
18         Scanner scanner=new Scanner(System.in);
19         int noFile=scanner.nextInt();

21         try(Socket clientSocket = new Socket(host, port);
22             InputStream in=clientSocket.getInputStream();
23             DataOutputStream out=
24                 new DataOutputStream(clientSocket.getOutputStream())){
25             out.writeInt(noFile);
26             BufferedImage bi=ImageIO.read(in);
27             Image image=(Image)bi;
28             ShowImage s=new ShowImage(image);
29             s.show();
30         }
31         catch(Exception e){
32             System.err.println("Client communication error : "+e.getMessage());
33         }
34     }
35 }

```

Clasa *ShowImage* afișează imaginea descărcată.

Clasa *AppThread.java*

```

1 package streamingimage;
2 import java.net.Socket;
3 import java.io.InputStream;
4 import java.io.OutputStream;
5 import javax.imageio.stream.FileImageOutputStream;
6 import java.io.DataInputStream;
7 import javax.imageio.ImageIO;
8 import java.io.IOException;
9 import java.io.File;
10 import java.awt.image.BufferedImage;
11 import java.awt.Image;

13 public class AppThread extends Thread{
14     Socket socket=null;

16     public AppThread(Socket socket){
17         this.socket=socket;
18     }

```



```
20 public void run(){
21     String path=" ../resources/images/";
22     try(OutputStream out=socket.getOutputStream();
23         DataInputStream in=
24             new DataInputStream(socket.getInputStream())){
25         int noFile=in.readInt();

27         String fileName="";
28         if(noFile==1)
29             fileName="xml-pic.jpg";
30         else
31             fileName="brasov.jpg";
32         File file=new File(path+fileName);
33         BufferedImage bi=ImageIO.read(file);
34         ImageIO.write(bi,"jpg",out);
35         out.flush();
36         socket.close();
37     }
38     catch(Exception e){
39         e.printStackTrace();
40     }
41 }
42 }
```

## 2.3 Datagrame

Pentru utilizarea datagramelor pachetul `java.net` pune la dispoziție clasele

- `java.net.DatagramSocket`
- `java.net.DatagramPacket`
- `java.net.MulticastSocket`

O aplicație trimite și recepționează pachete `DatagramPacket` prin intermediul unui `DatagramSocket`. Un pachet `DatagramPacket` poate fi trimis la mai mulți destinatari prin intermediul unui `MulticastSocket`.

Reamintim că o datagramă este un mesaj trimis prin rețea a cărei sosire nu este garantată iar momentul de sosire este neprecizat.

### Clasa `java.net.DatagramPacket`.

Trimiterea unui pachet UDP necesită crearea unui obiect `DatagramPacket` care conține corpul mesajului și adresa destinației. Apoi acest obiect `DatagramPacket` poate fi pus în rețea în vederea trimiterii sale. Primirea unui pachet UDP necesită crearea unui obiect `DatagramPacket` și

apoi acceptarea unui pachet UDP din rețea. După primire, se poate extrage din obiectul `DatagramPacket` adresa sursă și conținutul mesajului.

### Constructori

Există doi constructori pentru datagrame UDP. Primul constructor este folosit pentru primirea de pachete și necesită doar furnizarea unei memorii tampon, iar celălalt este folosit pentru trimiterea de pachete și necesită specificarea adresei destinatarului.

- `DatagramPacket(byte[ ] buffer, int lung)`

Acest constructor este folosit pentru primirea pachetelor. Un pachet se memorează în tamponul *buffer* având *lung* octeți. Dacă lungimea pachetului depășește această lungime, atunci pachetul este trunchiat iar octeții în plus se pierd.

- `DatagramPacket(byte[ ] buffer, int lung, InetAddress adresa, int port)`

Acest constructor este folosit pentru crearea unui pachet în vederea expedierii. Corpul pachetului este conținut în tamponul *buffer* având *lung* octeți. Pachetul va fi trimis către adresa și portul specificat. Trebuie să existe un server UDP care ascultă la portul specificat pentru trimiterea pachetelor. Un server UDP poate coexista cu un server TCP care ascultă același port.

### Metode

- `InetAddress getAddress()`  
returnează adresa IP a expeditorului.
- `int getPort()`  
returnează portul expeditorului.
- `byte[] getData()`  
returnează conținutul pachetului.
- `int getLength()`  
returnează lungimea pachetului.
- `void setAddress(InetAddress adresa)`  
fixează adresa IP a pachetului.

- `void setPort(int port)`  
fixează portul.
- `void setData(byte[] buffer)`  
fixează conținutul pachetului.
- `void setLength(int lung)`  
fixează lungimea pachetului.

## Clasa `java.net.DatagramSocket`

Această clasă se folosește atât pentru trimiterea, cât și pentru primirea obiectelor `DatagramPacket`. Un obiect `DatagramSocket` ascultă la un port cuprins între 1 și 65535 (porturile cuprinse între 1 și 1023 sunt rezervate pentru aplicațiile sistem). Deoarece UDP nu este orientat pe conexiune, se va crea un singur obiect `DatagramSocket` pentru trimiterea pachetelor către diferite destinații și primirea pachetelor de la diferite surse.

### Constructori

- `DatagramSocket()` throws `SocketException`  
Crează un obiect `DatagramSocket` cu un număr de port aleator;
- `DatagramSocket(int port)` throws `SocketException`  
Crează un obiect `DatagramSocket` cu numărul de port specificat;

### Metode

Clasa `DatagramSocket` conține metode pentru trimiterea și primirea de obiecte `DatagramPacket`, închiderea soclului, determinarea informațiilor adresei locale și setarea timpului de primire.

- `void send(DatagramPacket packet) throws IOException`  
Trimite *packetul* prin rețea. Dacă se trimite pachete la o destinație necunoscută sau care nu ascultă, în cele din urmă se generează o excepție `IOException`.
- `void receive(DatagramPacket packet) throws IOException`  
Metoda primește un singur pachet UDP în obiectul *packet* specificat. Apoi, pachetul poate fi inspectat pentru determinarea adresei IP sursă, portul sursă și lungimea mesajului. Execuția metodei este blocată până când se primește cu succes un pachet sau se scurge timpul de așteptare.

- `InetAddress getLocalAddress()`  
Returnează adresa locală către care este legat acest `DatagramSocket`;
- `int getLocalPort()`  
Returnează numărul de port unde ascultă `DatagramSocket`.
- `void close()`  
Închide `DatagramSocket`.
- `void setSoTimeout(int timpDeAșteptare) throws SocketException`  
Metoda fixează timpul de așteptare (în milisecunde) a soclului. Metoda `receive()` se va bloca pentru timpul de așteptare specificat pentru primirea unui pachet UDP, după care va arunca o excepție `InterruptedException`. Dacă valoarea parametrului este 0, atunci soclul este blocat.
- `int getSoTimeout() throws SocketException`  
Returnează timpul de așteptare.
- `void setSendBufferSize(int lungime) throws SocketException`  
Fixează lungimea tamponului de trimitere a soclului la valoarea specificată. Nu poate fi trimis mesaj UDP de lungime mai mare de această valoare.
- `int getSendBufferSize() throws SocketException`  
Returnează lungimea tamponului de trimitere a soclului.
- `void setReceiveBufferSize(int lungime) throws SocketException`  
Fixează lungimea tamponului de primire a soclului la valoarea specificată. Nu poate fi primit un mesaj UDP de lungime mai mare de această valoare.
- `int getReceiveBufferSize() throws SocketException`  
Returnează lungimea tamponului de primire a soclului.
- `void disconnect()`  
Deconectează soclul conectat.

- `InetAddress getAddress()`

Returnează obiectul `InetAddress` către care este conectat soclul sau `null` dacă acesta nu este conectat.

- `int getPort()`

Returnează portul la care este conectat soclul sau `-1` dacă acesta nu este conectat.

## Clasa `java.net.InetAddress`

Datele pot fi trimise prin rețea indicând adresa IP corespunzătoare mașinii destinație. Clasa `InetAddress` furnizează acces la adresele IP.

Nu există constructori pentru această clasă. Instanțele trebuie create folosind metodele statice:

- `InetAddress getLocalHost() throws UnknownHostException`  
Returnează un obiect `InetAddress` corespunzător mașinii locale.
- `InetAddress getByName(String host) throws UnknownHostException`  
Returnează un obiect `InetAddress` corespunzător mașinii *host*, parametru care poate fi specificat prin nume (de exemplu "atlantis") sau prin adresa IP ("168.192.0.1").
- `InetAddress [ ]getAllByName(String host) throws UnknownHostException`  
Returnează un șir de obiecte `InetAddress` corespunzător fiecărei adrese IP a mașinii *host*.

## Metodele clasei `InetAddress`

- `byte [ ] getAddress()`  
Returnează șirul de octeți corespunzător obiectului *InetAddress* de referință.
- `String getHostName()`  
Returnează numele mașinii gazdă.
- `String.getHostAddress()`  
Returnează adresa IP a mașinii gazdă.

- boolean `isMulticastAddress()`

Returnează `true` dacă obiectul `InetAddress` reprezintă o adresă IP multicast (cuprins între 224.0.0.0 și 239.255.255.255).

Exemplul următor afișează numele și adresa calculatorului gazdă cât și acela al calculatoarelor ale căror nume este transmis programului ca parametru.

### Exemplul 2.3.1

```

1 import java.net.InetAddress;
2 import java.net.UnknownHostException;

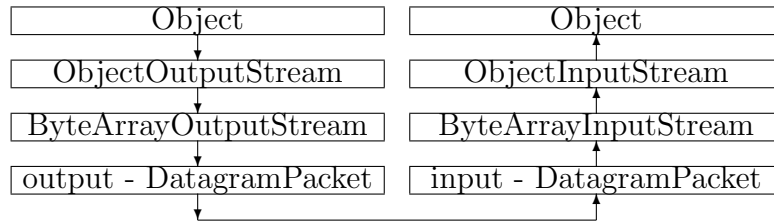
4 public class AdreseIP{
5     public static void main(String arg[]){
6         InetAddress adresa=null;
7         try{
8             adresa=InetAddress.getLocalHost();
9             System.out.println(" Calculatorul gazda are:");
10            System.out.println(" numele : "+adresa.getHostName());
11            System.out.println(" adresa IP : "+adresa.getHostAddress());
12        }
13        catch(UnknownHostException e){
14            System.out.println("UnknownHostException : "+e.getMessage());
15        }
16        if(arg.length>0){
17            for(int i=0;i<arg.length;i++){
18                try{
19                    adresa=InetAddress.getByName(arg[i]);
20                    System.out.println(" Calculatorul "+arg[i]+" are:");
21                    System.out.println(" adresa IP \"getByName\" : "+adresa);
22                    byte [] b=adresa.getAddress();
23                    for(int j=0;j<b.length;j++){
24                        if(b[j]<0)
25                            System.out.print(256+b[j]+".");
26                        else
27                            System.out.print(b[j]+".");
28                    System.out.println();
29                }
30                catch(UnknownHostException e){
31                    System.out.println("UnknownHostException : "+
32                        e.getMessage());
33                }
34            }
35        }
36    }
37 }

```

### 2.3.1 Aplicații client – server cu datagrame.

Un pachet de tip `DatagramPacket` este alcătuit dintr-un șir de octeți. Este datoria programatorului să transforme datele (mesajul) în șiruri de octeți la expediere și la recepție.

Transformarea unui obiect serializabil într-un șir de octeți și invers se poate realiza cu schema



Fie *socket* un obiect de tip **DatagramSocket** prin intermediul căruia se execută expedierea / recepționarea pachetelor de tip **DatagramPacket**.

În principiu expedierea și transformarea obiectului *obj* într-un șir de octeți se poate realiza cu secvența de cod

```

ByteArrayOutputStream baos=new ByteArrayOutputStream(256);
ObjectOutputStream out=new ObjectOutputStream(baos);
out.writeObject(obj);
byte[] bout=baos.toByteArray();

```

```

DatagramPacket packet=new DatagramPacket(bout,bout.length,
        address,port);
socket.send(packet);

```

unde *address* și *port* sunt adresa și portul destinatarului. Dacă *packet* este un obiect **DatagramPacket** recepționat atunci metodele **getAddress()** și **getPort()** furnizează adresa și portul expeditorului

Recepționarea și transformarea inversă este

```

byte[] bin=new byte[256];
packet=new DatagramPacket(bin,bin.length);
socket.receive(packet);

```

```

ByteArrayInputStream bais=new ByteArrayInputStream(bin);
ObjectInputStream in=new ObjectInputStream(bais);
obj=in.readObject();

```

Dacă mesajul este un obiect **String** atunci el se transformă într-un șir de octeți cu metoda `byte[] String.getBytes()` și invers, el se obține prin `new String(bin)` sau `new String(packet.getData())`.

**Exemplul 2.3.2** *Programăm aplicația de calculul a celui mai mare divizor comun a două numere naturale.*

În vederea transportului definim clasa

```

1 package server;
2 import java.io.Serializable;

4 public class Protocol implements Serializable{
5     long x,y;
6     Protocol(long x,long y){
7         this.x=x;
8         this.y=y;
9     }
10 }
```

Clientul va trimite un obiect *Protocol*, care va conține datele problemei și va recepționa un obiect de același tip cu rezultatul (cel mai mare divizor comun) în primul câmp al obiectului. Aceasta clasa este disponibilă atât serverului cât și clientului.

Se va utiliza arhitectura de aplicație dezvoltată în finalul secțiunii anterioare. Aplicația se va compune din:

- Aplicația server alcătuită din:

– Interfață

```

1 package iserver;
2 import java.net.DatagramSocket;
3 public interface IMyMServer{
4     public DatagramSocket getDatagramSocket(int port);
5     public void myAction(DatagramSocket datagramSocket);
6 }
```

– Implementarea interfeței

```

1 package server.impl;
2 import server.Protocol;
3 import iserver.IMyMServer;
4 import java.net.DatagramSocket;
5 import java.net.DatagramPacket;
6 import java.net.InetAddress;
7 import java.io.IOException;
8 import java.io.ByteArrayInputStream;
9 import java.io.ByteArrayOutputStream;
10 import java.io.ObjectInputStream;
11 import java.io.ObjectOutputStream;

13 public class MyMServer implements IMyMServer{
14     interface ServerDatagramAction{
15         void service(DatagramSocket socket);
16     }
```



```

18  static ServerDatagramAction action =
19      (DatagramSocket datagramSocket)->{
20      DatagramPacket packet=null;
21      Protocol p=null;
22      try{
23          byte[] bin=new byte[4048];
24          packet=new DatagramPacket(bin,bin.length);
25          datagramSocket.receive(packet);
26          ByteArrayInputStream bais=new ByteArrayInputStream(bin);
27          ObjectInputStream in=new ObjectInputStream(bais);
28          p=(Protocol)in.readObject();
29          App app=new App();
30          p.x=app.cmmdcService.cmmdc(p.x,p.y);
31          p.y=0;
32          ByteArrayOutputStream baos=new ByteArrayOutputStream(256);
33          ObjectOutputStream out=new ObjectOutputStream(baos);
34          out.writeObject(p);
35          byte[] bout=baos.toByteArray();
36          InetAddress address=packet.getAddress();
37          int port=packet.getPort();
38          packet=new DatagramPacket(bout,bout.length,address,port);
39          datagramSocket.send(packet);
40      }
41      catch(Exception e){
42          e.printStackTrace();
43      }
44  };

46  public DatagramSocket getDatagramSocket(int port){
47      DatagramSocket datagramSocket = null;
48      try{
49          datagramSocket = new DatagramSocket(port);
50      }
51      catch (IOException e) {
52          e.printStackTrace();
53      }
54      System.out.println("DatagramSocket is ready ...");
55      return datagramSocket;
56  }

58  public void myAction(DatagramSocket datagramSocket){
59      while(true){
60          action.service(datagramSocket);
61      }
62  }
63  }

```

— Clasă de lansare a serverului

```

1  package server;
2  import java.net.DatagramSocket;
3  import server.impl.MyMServer;
4  import iserver.IMyMServer;

6  public class AppServer{
7      public static void main(String[] args){
8          int port=7999;
9          if(args.length>0)
10             port=Integer.parseInt(args[0]);

```

```

11      IMyMServer myMServer=new MyMServer();
12      DatagramSocket datagramSocket =
13          myMServer.getDatagramSocket(port);
14      myMServer.myAction(datagramSocket);
15  }
16 }

```

### ● Aplicația client

```

1 package client;
2 import java.net.DatagramSocket;
3 import java.net.DatagramPacket;
4 import java.net.InetAddress;
5 import java.io.ByteArrayInputStream;
6 import java.io.ByteArrayOutputStream;
7 import java.io.ObjectInputStream;
8 import java.io.ObjectOutputStream;
9 import server.Protocol;
10 import java.util.Scanner;

12 public class CmmdcClient{
13     public static void main(String [] args){
14         String hostServer="localhost";
15         int portServer=7999;
16         if(args.length>0)
17             hostServer=args[0];
18         if(args.length>1)
19             portServer=Integer.parseInt(args[1]);
20         try{
21             DatagramSocket socket=new DatagramSocket();
22             Protocol p=new Protocol(0,0);
23             Scanner scanner=new Scanner(System.in);
24             System.out.println("Introduceti primul numar: ");
25             p.x=scanner.nextLong();
26             System.out.println("Introduceti al doilea numar: ");
27             p.y=scanner.nextLong();
28             ByteArrayOutputStream baos=new ByteArrayOutputStream(256);
29             ObjectOutputStream out=new ObjectOutputStream(baos);
30             out.writeObject(p);
31             byte[] bout=baos.toByteArray();
32             InetAddress address=InetAddress.getByName(hostServer);
33             DatagramPacket packet =
34                 new DatagramPacket(bout,bout.length,address,portServer);
35             socket.send(packet);
36             byte[] bin=new byte[4048];
37             packet=new DatagramPacket(bin,bin.length);
38             socket.receive(packet);
39             ByteArrayInputStream bais=new ByteArrayInputStream(bin);
40             ObjectInputStream in=new ObjectInputStream(bais);
41             p=(Protocol)in.readObject();
42             System.out.println("Cmmdc = "+p.x);
43             socket.close();
44         }
45         catch(Exception e){
46             System.out.println(e.getMessage());
47         }
48     }
49 }

```

### 2.3.2 Multicast vs. Broadcast

#### Clasa `java.net.MulticastSocket`

Prin intermediul unui soclu de tip `MulticastSocket` se pot recepționa datagrame expediate de un server către toți clienții cu un asemenea soclu.

##### Constructori

- `MulticastSocket(int port)` throws `SocketException`

##### Metode

- `void joinGroup(InetAddress adresă)` throws `SocketException`  
Soclu se "conectează" la grupul definit de adresa IP (de tip D, adică cuprins între 224.0.0.0 și 239.255.255.255).
- `void leaveGroup(InetAddress adresă)` throws `SocketException`  
Soclu se "deconectează" la grupul definit de adresa IP.
- `void close()`

Pregătirea clientului în vederea recepționării datagramelor printr-un soclu de tip `MulticastSocket` constă din

```
MulticastSocket socket= new MulticastSocket(port);  
InetAddress adresa=InetAddress.getByName("230.0.0.1");  
socket.joinGroup(adresa);
```

În final, clientul se deconectează și închide soclul.

```
socket.leaveGroup(adresa);  
socket.close();
```

Pachetele trimise de programul server trebuie să se adreseze grupului, identificat prin adresa IP de tip D.

Astfel prin multicast serverul trimite pachete la o adresă de grup și la un port fixat. Pachetele emise de server sunt recepționate de orice client ce crează un soclu de tip `MulticastSocket` pentru portul la care emite serverul și care se alătură grupului.

Prin broadcast serverul emite datagrame către orice calculator al rețelei locale la un anumit port. Faptul că emiterea datagramelor este de tip broadcast se indică prin

```
DatagramSocket.setBroadcast(true)
```

Orice client care își crează un soclu la portul la care emite serverul recepționează datagramele trimise de server. Adresa utilizată de server la crearea datagramelor trebuie să identifice rețeaua.

**Observație.** În cazul unui calculator "izolat" este nevoie de instalarea driverului *Microsoft Loopback Adapter*, care simulează existența unei plăci de rețea active.

**Exemplul 2.3.3 *Multicast și Broadcast:*** programele server emit din cinci în cinci secunde ora exactă. Un client va recepționa câte cinci datagrame.

Codurile sunt date în Fig. 2.1 și Fig. 2.2, respectiv pentru partea de server și cea de client.

## 2.4 Canale de comunicație

Odată cu versiunea j2sdk1.4 apar clase noi pentru operații de intrare - ieșire în pachetele `java.nio` și `java.nio.channels`. Pachetul `java.nio.channels` conține clase pentru comunicația în rețea, și anume canalele de comunicație.

### 2.4.1 SocketChannel & DatagramChannel

Utilizăm clasele `java.nio.channels.SocketChannel`, `java.nio.channels.DatagramChannel`. Informația transportată în aceste canale de comunicație trebuie înglobată în obiecte container de tip `Buffer`.

### Clasa `java.nio.Buffer`

Ierarhia claselor `Buffer`

```
abstract Buffer
  ByteBuffer
  ShortBuffer
  IntBuffer
  LongBuffer
  FloatBuffer
  DoubleBuffer
  CharBuffer
```

MulticastServer	BroadcastServer
<pre> import java.io.*; import java.net.*; import java.util.*; import java.text.DateFormat;  public class MulticastServer{ public static void main(String[] args){ long FIVE_SECONDS = 5000; boolean sfarsit=false; int serverPort=7000; int clientPort=7001; byte[] buf = new byte[256]; Date data=null; DatagramPacket packet = null; try(DatagramSocket socket= =new DatagramSocket(serverPort)){ while (! sfarsit){ data=new Date(); String df=DateFormat. getTimeInstance().format(data); buf = df.getBytes();  // send it  InetAddress group = InetAddress.getByName("230.0.0.1"); packet=new DatagramPacket(buf, buf.length,group,clientPort);  socket.send(packet);  // sleep for a while Thread.sleep(FIVE_SECONDS); } } catch (Exception e) { System.out.println(e.getMessage()); } } } </pre>	<pre> import java.io.*; import java.net.*; import java.util.*; import java.text.DateFormat;  public class BroadcastServer{ public static void main(String[] args){ long FIVE_SECONDS = 5000; boolean sfarsit=false; int serverPort=7000; int clientPort=7001; byte[] buf = new byte[256]; Date data = null; DatagramPacket packet = null; try(DatagramSocket socket= new DatagramSocket(serverPort)){ while (! sfarsit) { data=new Date(); String df=DateFormat. getTimeInstance().format(data); buf = df.getBytes();  InetAddress group = InetAddress.getByName("192.168.0.255"); packet=new DatagramPacket(buf, buf.length,group,clientPort); socket.setBroadcast(true); socket.send(packet);  Thread.sleep(FIVE_SECONDS); } } catch (Exception e) { System.out.println(e.getMessage()); } } } </pre>

Table 2.1: Clasele server.

MulticastClient	BroadcastClient
<pre> import java.io.*; import java.net.*;  public class MulticastClient {      public static void main(String[] args)         throws IOException {         DatagramPacket packet;         byte[] buf = new byte[256];         int clientPort=7001;         MulticastSocket socket=             new MulticastSocket(clientPort);         InetAddress address=             InetAddress.getByName("230.0.0.1");         socket.joinGroup(address);         int i=-1;         do{             i++;             packet=new DatagramPacket(buf, buf.length);             socket.receive(packet);             String received=new String(packet.getData());             System.out.println("Am primit: "+received);         }         while(i&lt;5);         socket.leaveGroup(address);         socket.close();     } } </pre>	<pre> import java.io.*; import java.net.*;  public class BroadcastClient {      public static void main(String[] args)         throws IOException {         DatagramPacket packet;         byte[] buf = new byte[256];         int clientPort=7001;         DatagramSocket socket=             new DatagramSocket(clientPort);          int i=-1;         do{             i++;             packet=new DatagramPacket(buf, buf.length);             socket.receive(packet);             String received=new String(packet.getData());             System.out.println("Am primit: "+received);         }         while(i&lt;5);          socket.close();     } } </pre>

Table 2.2: Clasele client.

Un obiect de tip **typeBuffer** este un tampon (container) care conține date de tipul specificat de denumirea clasei.

Un obiect de tip **Buffer** este caracterizat de

- *capacitate (capacity)* numărul elementelor care pot fi înmagazinate în tampon;
- *limită (limit)* marginea superioară a indicelui;
- *indice (position)* valoarea curentă a indicelui, ce corespunde unui cursor ce indică începutul zonei unde se introduc sau de unde se extrag date din tampon.

Metode generale

- *clear()* permite unui obiect de tip **Buffer** să fie reîncărcat. Fixează *limita = capacitate* și *indice = 0*.
- *flip()* pregătește obiectul de tip **Buffer** pentru consultare (citire). Fixează *limita = numărul elementelor din tampon* și *indice = 0*.
- *rewind()* pregătește obiectul de tip **Buffer** pentru re-citire.
- *remaining()* returnează numărul octeților cuprinși între valoarea indicelui și limită.

## Clasa `java.nio.ByteBuffer`

Instanțierea unui obiect se obține prin metoda statică

```
static ByteBuffer allocate(capacitate)
```

Introducerea și extragerea datelor din tampon se poate face în mod

- relativ - implică modificarea indicelui tamponului;
- absolut - fără modificarea indicelui.

## Metode

Introducerea datelor în mod relativ:

- `ByteBuffer put(byte b)`

- `ByteBuffer putTip(Tip x)`

Extragerea datelor în mod relativ:

- `byte get()`
- `Tip getTip()`

Introducerea datelor în mod absolut:

- `ByteBuffer put(int index, byte b)`
- `ByteBuffer putTip(int index, Tip x)`

Extragerea datelor în mod absolut:

- `byte get(int index)`
- `Tip getTip(int index)`

unde *tip* poate fi `char`, `short`, `int`, `long`, `float`, `double`.

Alte metode

- `public final boolean hasRemaining()`  
Dacă indicele nu este egal cu limita atunci returnează `true`, semnalând existența în tampon a unor octeți.
- `static ByteBuffer wrap(byte[] array)`
- `public static ByteBuffer wrap(byte[] array, int offset, int length)`  
Convertește șirul de octeți într-un obiect `ByteBuffer`.
- `public final byte[] array()`  
Transformarea inversă, obiectul `ByteBuffer` este convertit într-un șir de octeți.

Un obiect de tip `ByteBuffer` se poate percepe ca un obiect de tip `LongBuffer`, `DoubleBuffer`, etc prin

```
ByteBuffer bb=ByteBuffer.allocate(10);
LongBuffer lb=bb.asLongBuffer();
DoubleBuffer db=bb.asDoubleBuffer();
```



## Clasa `java.net.InetSocketAddress`

Clasa `InetSocketAddress` extinde clasa `SocketAddress` și încapsulează adresa unui calculator din Internet împreună cu un port în vederea *legării* la un `ServerSocket`.

### Constructorii:

- `InetSocketAddress(InetAddress addr, int port)`
- `InetSocketAddress(String numeCalculator, int port)`
- `InetSocketAddress(int port)`

## Clasa `java.nio.channels.ServerSocketChannel`

Crearea unui obiect de tip `ServerSocketChannel` se realizează prin

```
static ServerSocketChannel open() throws IOException
```

Unui asemenea obiect i se asociază un `ServerSocket` prin metoda

```
ServerSocket socket() throws IOException.
```

Obiectul de tip `ServerSocket` trebuie leagat la un port de comunicație prin metoda

```
void bind(InetSocketAddress endpoint) throws IOException.
```

Șablonul de utilizare este

```
try{
    ServerSocketChannel ssc=ServerSocketChannel.open();
    ServerSocket ss=ssc.socket();
    InetSocketAddress isa=new InetSocketAddress(addr,port);
    ss.bind(isa);
}
catch(Exception e){. . .}
```

La apelul unui client, serverul trebuie să genereze un obiect de tip `SocketChannel` prin care se vor derula comunicațiile cu clientul. Acest canal de comunicație se obține cu metoda `accept()`.

## Clasa `java.nio.channels.SocketChannel`

Crearea unui obiect de tip `SocketChannel` se realizează prin

```
static SocketChannel open() throws IOException
```

Acest obiect trebuie conectat la obiectul `ServerSocketChannel` al serverului cu metoda `connect(InetSocketAddress addr)`.

Datele vehiculate printr-un `SocketChannel` sunt de tip `ByteBuffer`. Datele se transmit prin metoda

```
int write(ByteBuffer sursă)
```

și se recepționează prin metoda

```
int read(ByteBuffer destinație)
```

Valoarea returnată de cele două metode reprezintă numărul octeților trimiși / recepționați.

Doar octeții unui obiect de tip `ByteBuffer` cuprinși între indice și limită sunt transmiși prin canal. Astfel, după încărcarea unui obiectului `ByteBuffer` cu metode relative trebuie apelată metoda `flip()`.

Canalul se închide cu metoda `close()`.

**Exemplul 2.4.1** *Calculul celui mai mare divizor comun a două numere naturale.*

Aplicație client-server bazat pe canale de comunicație prin socluri se compune din:

- Aplicația server alcătuită din:

– Interfață

```
1 package iserver;  
2 import java.nio.channels.ServerSocketChannel;  
3 public interface IMyMServer{  
4     public ServerSocketChannel getServerSocketChannel(int port);  
5     public void myAction(ServerSocketChannel serverSocketChannel);  
6 }
```

– Implementarea interfeței  
Clasa *MyMServer*

```

1 package server.impl;
2 import iserver.IMyMServer;
3 import java.net.InetSocketAddress;
4 import java.net.ServerSocket;
5 import java.nio.channels.ServerSocketChannel;
6 import java.util.concurrent.ExecutorService;
7 import java.util.concurrent.Executors;
8 import java.io.IOException;

10 public class MyMServer implements IMyMServer{
11     public ServerSocketChannel getServerSocketChannel(int port){
12         ServerSocketChannel serverSocketChannel=null;
13         try{
14             serverSocketChannel = ServerSocketChannel.open();
15             InetSocketAddress isa=new InetSocketAddress(port);
16             ServerSocket ss=serverSocketChannel.socket();
17             ss.bind(isa);
18         }
19         catch(IOException e){
20             e.printStackTrace();
21         }
22         System.out.println("Server ready ... ");
23         return serverSocketChannel;
24     }

26     public void myAction(ServerSocketChannel serverSocketChannel){
27         int NTHREADS=100;
28         AppThread appThread=new AppThread();
29         ExecutorService exec=Executors.newFixedThreadPool(NTHREADS);
30         while(true){
31             try{
32                 exec.execute(appThread.
33                     action.service(serverSocketChannel.accept()));
34             }
35             catch(IOException e){
36                 e.printStackTrace();
37             }
38         }
39     }
40 }

```

— Clasa *AppThread*

```

1 package server.impl;
2 import java.io.IOException;
3 import java.nio.channels.SocketChannel;
4 import java.nio.ByteBuffer;

6 public class AppThread{
7     interface ServerSocketChannelAction{
8         Thread service(SocketChannel socket);
9     }

11     static ServerSocketChannelAction action =
12         (SocketChannel socketChannel)->{
13         return new Thread(()->{
14             try{
15                 ByteBuffer bb = ByteBuffer.allocate(16);

```

```

16      //LongBuffer lb = bb.asLongBuffer();
17      socketChannel.read(bb);
18      // Varianta 1
19      long m=bb.getLong(0);
20      long n=bb.getLong(8);
21      // Varianta 2
22      // long m=lb.get(0);
23      // long n=lb.get(1);
24      App app=new App();
25      long r=app.cmmdcService.cmmdc(m,n);
26      bb.clear();
27      // Varianta 1
28      bb.putLong(0,r);
29      // Varianta 2
30      // lb.put(r);
31      socketChannel.write(bb);
32      socketChannel.close();
33  }
34  catch(IOException e){
35      e.printStackTrace();
36  }
37  });
38  };
39  }

```

– Clasă de lansare a serverului

```

1  package server;
2  import java.nio.channels.ServerSocketChannel;
3  import server.impl.MyMServer;
4  import iserver.IMyMServer;

6  public class AppServer{
7      public static void main(String[] args){
8          int port=7999;
9          if(args.length>0)
10             port=Integer.parseInt(args[0]);
11             IMyMServer myMServer=new MyMServer();
12             ServerSocketChannel serverSocketChannel =
13                 myMServer.getServerSocketChannel(port);
14             myMServer.myAction(serverSocketChannel);
15         }
16     }

```

- Aplicația client

```

1  package client;
2  import java.net.UnknownHostException;
3  import java.net.InetSocketAddress;
4  import java.nio.channels.SocketChannel;
5  import java.nio.ByteBuffer;
6  import java.util.Scanner;
7  import java.io.IOException;

9  public class CmmdcClient {
10     public static void main(String[] args) {
11         String host="localhost";
12         int port=7999;

```

```

13     if (args.length>0)
14         host=args[0];
15     if (args.length>1)
16         port=Integer.parseInt(args[1]);
17     SocketChannel sc=null;
18     try{
19         InetAddress isa=new InetAddress(host,port);
20         sc=SocketChannel.open();
21         sc.connect(isa);
22     }
23     catch (UnknownHostException e) {
24         System.err.println("Server necunoscut: "+host+" "+e.getMessage());
25         System.exit(1);
26     }
27     catch (IOException e) {
28         System.err.println("Conectare imposibila la: "+
29             host+" pe portul "+port+" "+e.getMessage());
30         System.exit(1);
31     }
32     Scanner scanner=new Scanner(System.in);
33     long m,n,r;
34     System.out.println("m=");
35     m=scanner.nextLong();
36     System.out.println("n=");
37     n=scanner.nextLong();

39     ByteBuffer bb=ByteBuffer.allocate(16);
40     // Varianta 1
41     bb.putLong(0,m).putLong(8,n);
42     // Varianta 2
43     // LongBuffer lb=bb.asLongBuffer();
44     // lb.put(0,m).put(1,n);
45     try{
46         sc.write(bb);
47         bb.clear();
48         sc.read(bb);
49         // Varianta 1
50         r=bb.getLong(0);
51         // Varianta 2
52         // r=lb.get(0);
53         System.out.println("Cmmdc : "+r);
54         sc.close();
55     }
56     catch(Exception e){
57         System.err.println("Eroare de comunicatie"+e.getMessage());
58     }
59 }
60 }

```

Varianta comentată corespunde cazului în care se utilizează clasa acoperitoare **LongBuffer**.

Alternativ, se poate lucra cu obiecte. Pentru exemplul dat, utilizând clasa *Protocol*, introdusă în secțiunea dedicată datagramelor, codul privind trimiterea și recepția datelor din clasele *AppThread* și *CmmdcClient* va fi

```

1  . . .
2  ByteBuffer bb = ByteBuffer.allocate(1024);

```

```

3      socketChannel.read(bb);
4      byte[] bin=bb.array();
5      ByteArrayInputStream bais=new ByteArrayInputStream(bin);
6      ObjectInputStream in=new ObjectInputStream(bais);
7      Protocol p=(Protocol)in.readObject();
8      App app=new App();
9      p.x=app.cmmdc(p.x,p.y);
10     p.y=0;
11     ByteArrayOutputStream baos=new ByteArrayOutputStream(256);
12     ObjectOutputStream out=new ObjectOutputStream(baos);
13     out.writeObject(p);
14     byte[] bout=baos.toByteArray();
15     bb=ByteBuffer.wrap(bout);
16     socketChannel.write(bb);
17     socketChannel.close();
18     . . .

```

și, respectiv,

```

1     . . .
2     ByteBuffer bb=ByteBuffer.allocate(2048);
3     Protocol p=new Protocol(m,n);
4     try{
5         ByteArrayOutputStream baos=new ByteArrayOutputStream(256);
6         ObjectOutputStream out=new ObjectOutputStream(baos);
7         out.writeObject(p);
8         byte[] bout=baos.toByteArray();
9         bb=ByteBuffer.wrap(bout);
10        sc.write(bb);
11        bb.clear();
12        sc.read(bb);
13        byte[] bin=bb.array();
14        ByteArrayInputStream bais=new ByteArrayInputStream(bin);
15        ObjectInputStream in=new ObjectInputStream(bais);
16        p=(Protocol)in.readObject();
17        System.out.println("Cmmdc : "+p.x);
18        sc.close();
19        . . .

```

## Clasa java.nio.channels.DatagramChannel

Șablonul de programare pentru instanțierea unui obiect de tip `DatagramChannel` este

```

DatagramChannel dc=null;
InetSocketAddress isa=new InetSocketAddress(port);
try{
    dc=DatagramChannel.open();
    DatagramSocket datagramSocket=dc.socket();
    datagramSocket.bind(isa);
}
catch(Exception e){. . .}

```

unde *port* este portul folosit de `DatagramChannel`. Dacă *port=0* atunci se alege aleator un port disponibil.

Transmiterea unui obiect `ByteBuffer` se face cu metoda

```
public int send(ByteBuffer src, SocketAddress target) throws
    IOException
```

Metoda returnează numărul de octeți expediați. Recepția unui `ByteBuffer` se obține cu metoda

```
public SocketAddress receive(ByteBuffer dst) throws IOException
```

Obiectul returnat reprezintă adresa expeditorului.

**Exemplul 2.4.2** *Calculul celui mai mare divizor comun a două numere naturale.*

- Aplicația server este alcătuită din:

– Interfață

```
1 package iserver;
2 import java.nio.channels.DatagramChannel;
3 public interface IMyMServer{
4     public DatagramChannel getDatagramChannel(int port);
5     public void myAction(DatagramChannel datagramChannel);
6 }
```

– Implementarea interfeței

```
1 package server.impl;
2 import iserver.IMyMServer;
3 import java.net.DatagramSocket;
4 import java.net.InetSocketAddress;
5 import java.net.SocketAddress;
6 import java.nio.channels.DatagramChannel;
7 import java.nio.ByteBuffer;
8 import java.io.IOException;

10 public class MyMServer implements IMyMServer{

12     interface ServerDatagramChannelAction{
13         void service(DatagramChannel datagramChannel);
14     }

16     static ServerDatagramChannelAction action =
17         (DatagramChannel datagramChannel)->{
18         try{
19             ByteBuffer bb = ByteBuffer.allocate(16);
20             //LongBuffer lb = bb.asLongBuffer();
21             SocketAddress sa=datagramChannel.receive(bb);
22             // Varianta 1
23             long m=bb.getLong(0);
```

```

24         long n=bb.getLong(8);
25         // Varianta 2
26         // long m=lb.get(0);
27         // long n=lb.get(1);
28         App app=new App();
29         long r=app.cmmdcService.cmmdc(m,n);
30         bb.clear();
31         // Varianta 1
32         bb.putLong(0,r);
33         // Varianta 2
34         // lb.put(r);
35         datagramChannel.send(bb,sa);
36     }
37     catch(Exception e){
38         e.printStackTrace();
39     }
40 };

42 public DatagramChannel getDatagramChannel(int port){
43     DatagramChannel datagramChannel=null;
44     InetSocketAddress isa=new InetSocketAddress(port);
45     try{
46         datagramChannel = DatagramChannel.open();
47         DatagramSocket datagramSocket=datagramChannel.socket();
48         datagramSocket.bind(isa);
49     }
50     catch(IOException e){
51         System.out.println("DatagramChannelError : "+e.getMessage());
52         System.exit(0);
53     }
54     System.out.println("Server ready... ");
55     return datagramChannel;
56 }

58 public void myAction(DatagramChannel datagramChannel){
59     while(true){
60         action.service(datagramChannel);
61     }
62 }
63 }

```

– Clasă de lansare a serverului

```

1 package server;
2 import java.nio.channels.DatagramChannel;
3 import server.impl.MyMServer;
4 import iserver.IMyMServer;

6 public class AppServer{
7     public static void main(String[] args){
8         int port=7999;
9         if(args.length>0)
10             port=Integer.parseInt(args[0]);
11         IMyMServer myMServer=new MyMServer();
12         DatagramChannel datagramChannel =
13             myMServer.getDatagramChannel(port);
14         myMServer.myAction(datagramChannel);
15     }
16 }

```



### ● Aplicația client

```

1 package client;
2 import java.net.UnknownHostException;
3 import java.net.InetSocketAddress;
4 import java.net.InetAddress;
5 import java.net.DatagramSocket;
6 import java.nio.channels.DatagramChannel;
7 import java.nio.ByteBuffer;
8 import java.util.Scanner;
9 import java.io.IOException;

11 public class CmmdcClient {
12     public static void main(String[] args) throws IOException {
13         String serverHost="localhost";
14         int port=7999;
15         if (args.length>0)
16             serverHost=args[0];
17         if (args.length>1)
18             port=Integer.parseInt(args[1]);
19         InetSocketAddress server=null, isa=null;
20         try{
21             server=
22                 new InetSocketAddress(InetAddress.getByName(serverHost),port);
23         }
24         catch(UnknownHostException e){
25             System.out.println("Unknown host : "+e.getMessage());
26             System.exit(1);
27         }
28         isa=new InetSocketAddress(0);
29         DatagramChannel dc=null;
30         try {
31             dc=DatagramChannel.open();
32             DatagramSocket socket = dc.socket();
33             socket.bind(isa);
34         }
35         catch (IOException e) {
36             System.err.println("Couldn't open the DatagramChannel "+
37                 e.getMessage());
38             System.exit(1);
39         }
40         long m,n,r;
41         Scanner scanner=new Scanner(System.in);
42         System.out.println("m=");
43         m=scanner.nextLong();
44         System.out.println("n=");
45         n=scanner.nextLong();
46         ByteBuffer bb=ByteBuffer.allocate(16);
47         // Varianta 1
48         bb.putLong(0,m).putLong(8,n);
49         // Varianta 2
50         // LongBuffer lb=bb.asLongBuffer();
51         // lb.put(0,m).put(1,n);
52         try{
53             dc.send(bb, server);
54             bb.clear();
55             dc.receive(bb);
56             // Varianta 1
57             r=bb.getLong(0);
58             // Varianta 2

```

```
59      // r=lb.get(0);  
60      System.out.println("Cmmdc = "+r);  
61  }  
62  catch(Exception e){  
63      System.err.println("Client error : "+e.getMessage());  
64  }  
65  finally{  
66      if(dc!=null) dc.disconnect();  
67  }  
68  }  
69  }
```

## 2.4.2 Recepție cu Selector

Programarea recepției cererilor clienților fără utilizarea firelor de execuție se poate face utilizând clasele

- `java.nio.channel.Selector`  
Gestionează obiectele `SocketChannel` înregistrate și transmite cererile care trebuie satisfăcute.
- `java.nio.channel.SelectionKey`  
Obiect utilizat de *selector* pentru sortarea cererilor. O cheie identifică un client și tipul cererii.

Figura 2.1 prezintă structura unei aplicații. <sup>1</sup>

### Clasa `java.nio.channel.Selector`

#### Metode

- `public static Selector open() throws IOException`  
Instanțiază un obiect de tip `Selector`.
- `public Set<SelectionKey> selectedKeys()`  
Returnează cheile înregistrate de selector.
- `public int select() throws IOException`  
Metodă blocantă până la selectarea unui canal.

---

<sup>1</sup>Imaginea este preluată din Naccarato G., *Introducing Nonblocking Sockets*, <http://www.onjava.com>, 2002.

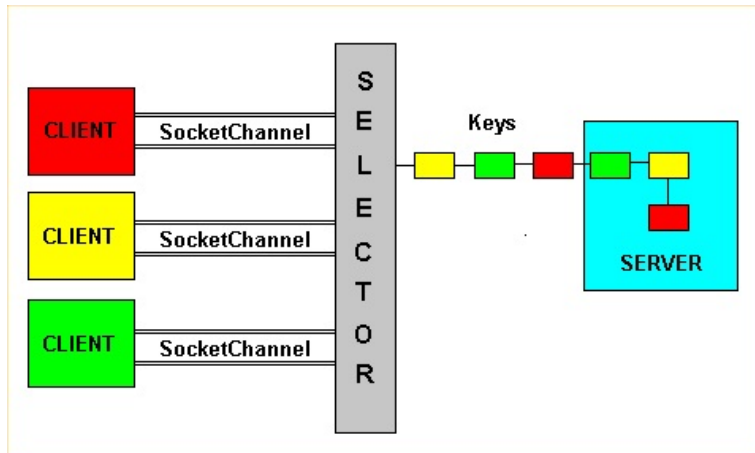


Figure 2.1: Structura unei aplicații.

Înregistrarea selectorului se face de către obiectul `SocketChannel` sau `ServerSocketChannel` prin metoda

```
public final SelectionKey register(Selector selector, int ops) throws
ClosedChannelException
```

Operațiile pot fi

```
SelectionKey.OP_CONNECT
SelectionKey.OP_ACCEPT
SelectionKey.OP_READ
SelectionKey.OP_WRITE
```

Canalul se configurează pe modul ne-blocant prin

```
configureBlocking(false) throws IOException
```

Clasa `java.nio.channel.SelectionKey`

Metode

- `public final boolean isConnectable()`
- `public final boolean isReadable()`
- `public final boolean isWritable()`
- `public final boolean isAcceptable()`
- `public void cancel()`



```

53         buffer.clear();
54         buffer.putLong(0,r);
55         client.write(buffer);
56         buffer.clear();
57         client.close();
58         key.cancel();
59     }
60 }
61 }
62 }
63 }
64 catch(Exception e){
65     System.err.println(" BufferOpException : "+e.getMessage());
66 }
67 finally{
68     try{
69         serverSocketChannel.close();
70     }
71     catch(Exception e){
72         System.out.println(" CloseException : "+e.getMessage());
73     }
74 }
75 }
76 }

```

Aplicația client nu suferă nici o modificare.

### 2.4.3 Comunicații asincrone prin canale

Utilizarea canalelor *asincrone*, introduse în Java 7, face apel la metode nebloccante.

Astfel sunt introduse clasele:

- `java.nio.channels.AsynchronousServerSocketChannel` și `java.nio.channels.AsynchronousSocketChannel`.

Tehnicile de programare utilizate se bazează pe utilizarea claselor:

- `java.util.concurrent.Future`

#### Interfața `java.util.concurrent.Future<V>`

##### Metode

- `V get()`
- `boolean isDone()`

Valoarea `true` rezultă dacă s-a obținut / generat obiectul de tip `V`.

Un obiect de tip  $V$  care rezultă în urma apelării metodei cu semnătura

*Future<V> metoda(...)*

se va utiliza doar după verificarea generării ei, de exemplu cu metoda `isDone()`.

## Clasa `AsynchronousServerSocketChannel`

Un obiect de tip `AsynchronousServerSocketChannel` se obține cu metoda statică `open()`.

### Metode

- `static AsynchronousServerSocketChannel open() throws IOException`
- `AsynchronousServerSocketChannel bind(SocketAddress addr)`  
Se precizează portul la care se leagă canalul asincron. Clasa `InetSocketAddress` este extinde clasa `SocketAddress`.
- `public Future<AsynchronousSocketChannel> accept()`  
Metoda neblocantă generează un soclu prin care se realizează conexiunea din partea serverului cu un client.

## Clasa `AsynchronousSocketChannel`

### Metodei

- `public static AsynchronousSocketChannel open() throws IOException`
- `public abstract Future<Void> connect(SocketAddress remote)`
- `public Future<Integer> read(ByteBuffer dst)`  
Metoda returnează numărul octeților încărcăți în *dst*.
- `public Future<Integer> write(ByteBuffer src)`  
Metoda returnează numărul octeților expediați din *src*.

### Exemplul 2.4.4

Utilizăm modelul aplicației dezvoltată pe baza clasei `ServerSocketChannel`:

1. *IMyMServer.java*

```

1 package iserver;
2 import java.nio.channels.AsynchronousServerSocketChannel;
3 public interface IMyMServer{
4     public AsynchronousServerSocketChannel
5         getAsynchronousServerSocketChannel(int port);
6     public void myAction(AsynchronousServerSocketChannel
7         asynchronousServerSocketChannel);
8 }

```

## 2. *MyMServer.java*

```

1 package server.impl;
2 import iserver.IMyMServer;
3 import java.net.InetSocketAddress;
4 import java.nio.channels.AsynchronousServerSocketChannel;
5 import java.nio.channels.AsynchronousSocketChannel;
6 import java.util.concurrent.ExecutorService;
7 import java.util.concurrent.Executors;
8 import java.io.IOException;
9 import java.util.concurrent.Future;

11 public class MyMServer implements IMyMServer{

13     public AsynchronousServerSocketChannel
14         getAsynchronousServerSocketChannel(int port){
15         AsynchronousServerSocketChannel asynchronousServerSocketChannel =
16             null;
17         try{
18             asynchronousServerSocketChannel =
19                 AsynchronousServerSocketChannel.open();
20             InetSocketAddress isa=new InetSocketAddress(port);
21             asynchronousServerSocketChannel.bind(isa);
22         }
23         catch(IOException e){
24             System.out.println(" AsynchronousServerSocketChannelError : " +
25                 e.getMessage());
26             System.exit(0);
27         }
28         System.out.println(" Server ready... ");
29         return asynchronousServerSocketChannel;
30     }

32     public void myAction(AsynchronousServerSocketChannel
33         asynchronousServerSocketChannel){
34         int NTHREADS=100;
35         ExecutorService exec=Executors.newFixedThreadPool(NTHREADS);
36         AppThread appThread=new AppThread();
37         while(true){
38             try{
39                 Future<AsynchronousSocketChannel> future =
40                     asynchronousServerSocketChannel.accept();
41                 exec.execute(appThread.action.service(future.get()));
42             }
43             catch(Exception e){
44                 System.err.println("MyActionException : "+e.getMessage());
45             }
46         }
47     }

```

```
48 }
```

### 3. *AppThread*

```
1 package server.impl;
2 import java.nio.channels.AsynchronousSocketChannel;
3 import java.nio.ByteBuffer;
4 import java.util.concurrent.Future;
5 import java.io.IOException;

6
7 public class AppThread{
8     interface ServerSocketAction{
9         Thread service(AsynchronousSocketChannel asc);
10    }

11
12    static ServerSocketAction action=(AsynchronousSocketChannel asc)->{
13        return new Thread()->{
14            try{
15                ByteBuffer bb = ByteBuffer.allocate(16);
16                //LongBuffer lb = bb.asLongBuffer();
17                Future<Integer> fr=asc.read(bb);
18                while(!fr.isDone());
19                // Varianta 1
20                long m=bb.getLong(0);
21                long n=bb.getLong(8);
22                // Varianta 2
23                // long m=lb.get(0);
24                // long n=lb.get(1);
25                App app=new App();
26                System.out.println(m+" <-> "+n);
27                long r=app.cmmdcService.cmmdc(m,n);
28                bb.clear();
29                // Varianta 1
30                bb.putLong(0,r);
31                // Varianta 2
32                // lb.put(r);
33                Future<Integer> fw=asc.write(bb);
34                while(!fw.isDone());
35                asc.close();
36            }
37            catch(IOException e){
38                e.printStackTrace();
39            }
40        });
41    };
42 }
```

### 4. *AppServer.java*

```
1 package server;
2 import java.nio.channels.AsynchronousServerSocketChannel;
3 import server.impl.MyMServer;
4 import iserver.IMyMServer;

5
6 public class AppServer{
7     public static void main(String[] args){
8         int port=7999;
```



```

9      if (args.length>0)
10         port=Integer.parseInt (args [0]);
11      IMyMServer myMServer=new MyMServer ();
12      AsynchronousServerSocketChannel asynchronousServerSocketChannel =
13         myMServer.getAsynchronousServerSocketChannel (port);
14      myMServer.myAction (asynchronousServerSocketChannel);
15   }
16 }

```

### 5. *CmmdcClient.java*

```

1  package client;
2  import java.net.UnknownHostException;
3  import java.net.InetSocketAddress;
4  import java.nio.channels.AsynchronousSocketChannel;
5  import java.nio.ByteBuffer;
6  import java.util.Scanner;
7  import java.io.IOException;
8  import java.util.concurrent.Future;

10 public class CmmdcClient {
11     public static void main(String [] args) {
12         String host="localhost";
13         int port=7999;
14         if (args.length>0)
15             host=args [0];
16         if (args.length>1)
17             port=Integer.parseInt (args [1]);
18         AsynchronousSocketChannel asc=null;
19         try{
20             InetSocketAddress isa=new InetSocketAddress (host ,port );
21             asc=AsynchronousSocketChannel.open ();
22             asc.connect (isa );
23         }
24         catch (UnknownHostException e) {
25             System.err.println ("Server necunoscut: "+host+" "+e.getMessage ());
26             System.exit (1);
27         }
28         catch (IOException e) {
29             System.err.println ("Conectare imposibila la: "+
30                 host+" pe portul "+port+" "+e.getMessage ());
31             System.exit (1);
32         }

34         Scanner scanner=new Scanner (System.in );
35         long m,n,r;
36         System.out.println ("m=");
37         m=scanner.nextLong ();
38         System.out.println ("n=");
39         n=scanner.nextLong ();

41         ByteBuffer bb=ByteBuffer.allocate (16);
42         bb.putLong (0,m).putLong (8,n);
43         try{
44             Future<Integer> fw=asc.write (bb);
45             while (!fw.isDone ());
46             bb.clear ();
47             Future<Integer> fr=asc.read (bb);

```

```
48         while (! fr . isDone () );
49         r=bb.getLong ( 0 );
50         System.out.println ("Cmmdc : " +r );
51         asc.close ();
52     }
53     catch (Exception e){
54         System.err.println ("Eroare de comunicatie"+e.getMessage ());
55     }
56 }
57 }
```

## Comentarii

Tehnologiile prezentate sunt incluse în distribuția Java standard, oferită de *Oracle*.

S-au dezvoltat instrumente de programare cu scopul eficientizării din diverse puncte de vedere a comunicațiilor bazate pe socluri:

- *apache-mina*
- *netty*
- *LMAX Disruptor*

## Comparare

Pe două calculatoare

1. Intel Core2 Duo T6600 2.20GHz, Windows 7
2. Intel I7-4700MQ 2.40 GHz, Windows 8.1

un client solicitând 8192 conexiuni și folosind aplicațiile prezentate mai sus, durata evaluată pe server a fost

Tehnologia	1		2	
	Timp (ms)	Rang	Timp (ms)	Rang
Socket	20264	4	2235	3
SocketChannel	26551	6	5735	6
Selector	19422	3	2437	5
Datagram	10015	1	1812	2
DatagramChannel	16692	2	1047	1
LMAX Disruptor	22994	5	2344	4

Durata a fost evaluată prin utilizarea metodei `System.currentTimeMillis()`.

## Întrebări recapitulative

1. Precizați termenul *socket* (*soclu*).
2. Precizați clasele Java necesare unei aplicații client-server cu socluri (*socket*).
3. Precizați diferența de simetrie privind instanțierea dintre un obiect de tip `Socket` și unul de tip `ServerSocket`.
4. Care este rolul unui obiect de tip `ServerSocket` și cum se utilizează?
5. Precizați metodele unui obiect `Socket`, necesare în transmiterea și recepția datelor.
6. Precizați termenul *multicast*.
7. În ce constă participarea serverului la transmisie *multicast* ?
8. În ce constă participarea unui client la recepția *multicast* ?
9. Precizați termenul *broadcast*.
10. În ce constă participarea serverului la transmisie *broadcast* ?
11. În ce constă participarea unui client la recepția *broadcast* ?
12. Care este rolul unui obiect de tip `DatagramPacket` ?

13. Precizați metodele clasei `DatagramSocket` utilizate la expedierea și la recepția unui datagram.
14. Ce parametri caracterizează un obiect de tip `Buffer` ?
15. Ce asemănare există între comunicația bazată de datagrame și cea prin intermediul canalelor ?
16. Care este rolul unui obiect de tip `ByteBuffer`?

#### Probleme

1. Aplicația cu `AsynchronousSocketChannel` nu funcționează corect cu mulți clienți.
2. Rezolvarea solicitării unui număr (foarte) mare de clienți (> 8192.)

## Capitolul 3

# Regăsirea obiectelor prin servicii de nume

Oricărui obiect îi sunt asociate un nume și o referință. Regăsirea / căutarea obiectului se face pornind de la numele obiectului, prin referință se ajunge la obiect.

**Exemple** date prin șablonul (Nume  $\Rightarrow$  Referință  $\Rightarrow$  Obiect)

1. Accesul la o carte într-o bibliotecă:

Titlul cărții  $\Rightarrow$  Referință cărții din bibliotecă  $\Rightarrow$  Carte

2. Contactul telefonic cu o persoană:

Nume persoană  $\Rightarrow$  Număr telefon  $\Rightarrow$  Persoană

Un serviciu de nume conține asocieri dintre nume de obiecte și obiecte și poate oferi facilități de regăsire a obiectelor.

### 3.1 *Java Naming and Directory Interface*

*Java Naming and Directory Interface* - *JNDI* este o interfață de programare (*Application Programming Interface* - *API*) care descrie funcționalitatea unui serviciu de nume.

Cuvântul *servici* este utilizat în sens comun, entitate care pune la dispoziție facilități de folosire.

Alături de interfața *JNDI*, arhitectura *JNDI* mai conține interfața *Service Provider Interface* - *SPI*. Implementarea interfeței *SPI* de un furnizor de servicii *JNDI* are ca efect independența programului Java de furnizorul de servicii *JNDI*.

JNDI este implementat de serviciile de nume:

- *Filesystem* are ca obiect asocierea dintre numele de fișier sau catalog cu obiectul corespunzător.
- *DNS - Domain Name System* are ca obiect asocierea dintre adresa Internet cu adresa IP.
- *RMI registry* utilizat în aplicații RMI, din Java. Are ca obiect asocierea între un nume de serviciu de invocare ale obiectelor la distanță cu un delegat al serviciului (stub).
- *COS - Common Object Service Naming* utilizat în aplicații CORBA. Are ca obiect asocierea între numele unui serviciu de invocare ale obiectelor la distanță cu referința la serviciu.
- *LDAP - Lightweight Directory Access Protocol* definește un protocol pentru accesarea datelor reținute într-un catalog LDAP (*LDAP directory, information directory*). Un catalog LDAP permite reținerea și regăsirea referințelor obiectelor definite pe un calculator.

## Interfața `javax.naming.Context`

Printr-un context se va înțelege o mulțime de asocieri *nume - obiect*. Corespunzător unui context, JNDI definește interfața **Context**, cu metodele

- `void bind(String nume, Object object)`
- `void rebind(String nume, Object object)`
- `void unbind(String nume)`
- `Object lookup(String nume)`
- `NamingEnumeration list(String nume)`

Returnează lista cu nume obiectelor împreună cu tipul lor.

- `NamingEnumeration listBindings(String nume)`

Returnează lista cu nume obiectelor împreună cu tipul și locația acestora.

Specificațiile JNDI prevăd definirea unui *context inițial*, implementat prin clasa `javax.naming.InitialContext`, clasă ce implementează interfața **Context**.  
**Constructorii.**

- `public InitialContext() throws NamingException`
- `public void InitialContext(Hashtable<?,?> environment) throws NamingException`

Pentru crearea *contextului inițial* trebuie specificată clasa care crează contextul inițial prin parametrul `java.naming.factory.initial` sau constanta `Context.INITIAL_CONTEXT_FACTORY`.

Acest parametru se poate da în mai multe moduri:

- Includerea în obiectul `Hashtable` care apare în constructorul clasei `InitialContext`.

```
Hashtable env=new Hashtable()
env.put("java.naming.factory.initial",...);
Context ctx=InitialContext(env);
```

sau

```
Hashtable env=new Hashtable()
env.put(Context.INITIAL_CONTEXT_FACTORY,...);
Context ctx=new InitialContext(env);
```

- Ca parametru de sistem furnizat la lansarea programului Java, prin

```
java -Djava.naming.factory.initial=... ClasaJava
```

Parametrul se poate da în interiorul programului prin

```
System.setProperty("java.naming.factory.initial",...);
```

sau

```
System.setProperty(Context.INITIAL_CONTEXT_FACTORY,...);
```

- Atribut în fișierul de proprietăți *jndi.properties*.

În aceste ultime două cazuri, contextul initial se crează prin

```
Context ctx=new InitialContext();
```

Funcție de serviciul de nume, clasa care crează contextul inițial este dat în tabelul

Serviciul de nume	Clasa
Filesystem	com.sun.jndi.fscontext.RefFSContextFactory
COS	com.sun.jndi.cosnaming.CNCTXFactory
RMI	com.sun.jndi.rmi.registry.RegistryContextFactory
DNS	com.sun.jndi.dns.DnsContextFactory
LDAP	com.sun.jndi.ldap.LdapCtxFactory

**Exemplul 3.1.1** *Utilizând serviciul JNDI Filesystem se crează un context prin intermediul căreia se afișează conținutul unui catalog indicat de client.*

```

1 import javax.naming.Context;
2 import javax.naming.InitialContext;
3 import javax.naming.Binding;
4 import javax.naming.NamingEnumeration;
5 import javax.naming.NamingException;
6 import javax.naming.NameClassPair;
7 import java.io.File;
8 import java.util.Hashtable;
9 import java.util.Scanner;

11 class Lookup{
12     public static void main(String[] args) {
13         Context ctx=null;
14         /*
15          // Varianta 1
16          Hashtable env = new Hashtable(11);
17          env.put(Context.INITIAL_CONTEXT_FACTORY,
18              "com.sun.jndi.fscontext.RefFSContextFactory");
19          try{
20              ctx = new InitialContext(env);
21          }
22          catch (NamingException e) {
23              System.out.println("InitialContextError : "+e.getMessage());
24          }
25          */
26          /*
27          // Varianta 2
28          System.setProperty("java.naming.factory.initial",
29              "com.sun.jndi.fscontext.RefFSContextFactory");
30          try{
31              ctx = new InitialContext();
32          }
33          catch (NamingException e) {
34              System.out.println("InitialContextError : "+e.getMessage());
35          }
36          */

38          // Varianta 3
39          try{
40              ctx = new InitialContext();

```



```

41     }
42     catch (NamingException e) {
43         System.out.println("InitialContextError : "+e.getMessage());
44     }

46     Scanner scanner=new Scanner(System.in);
47     System.out.println("Introduceti referinta absoluta a unui catalog : ");
48     String myName=scanner.next();
49     try{
50         System.out.println("\n ctx.lookup("+myName+" ) produce");
51         System.out.println(ctx.lookup(myName));

53         System.out.println("\nContinutul catalogului "+myName+" este:\n");
54         NamingEnumeration lst=ctx.list(myName);
55         while(lst.hasMore()){
56             NameClassPair nc = (NameClassPair)lst.next();
57             System.out.println(nc);
58         }

60         System.out.println("\nContinutul catalogului "+myName+" este:\n");
61         NamingEnumeration lst1 = ctx.listBindings(myName);
62         while (lst1.hasMore()) {
63             Binding bd = (Binding)lst1.next();
64             System.out.println(bd);
65         }
66         ctx.close();
67     }
68     catch (NamingException e) {
69         System.out.println("Lookup failed: " + e.getMessage());
70     }
71 }
72 }

```

Apelarea clasei, în varianta dată, constă din

```
java -Djava.naming.factory.initial=com.sun.jndi.fscontext.RefFSContextFactory Lookup
```

sau

```
java Lookup
```

caz, în care, în catalogul curent se găsește fișierul *jndi.properties* cu conținutul

```
java.naming.factory.initial=com.sun.jndi.fscontext.RefFSContextFactory
```

Serviciul de nume *Filesystem* este dat de fișierul *fscontext-\*.jar* (Unele distribuții sunt alcătuite din fișierele *fscontext.jar* și *providerutil.jar*).

### 3.1.1 LDAP

LDAP poate fi privit ca un sistem de gestiune a unei baze de date (ne relațional). Baza de date este alcătuită din attribute ale căror nume este precizat de protocolul LDAP.

Punctele de intrare (rădăcinile) sunt date de *DN* (*Distinguished Name*). Uzual *DN* se definește printr-una din variantele

1. `o="organization",c="country"`
2. `o="organization"`
3. `dc="domain_content_1",dc="domain_content_2"`  
De exemplu, `dc=example,dc=com`
4. `uid=user id,ou="organization unit"`  
De exemplu, `uid=admin,ou=system`

Un utilizator este caracterizat prin atributul *cn* - *common name* și are acces la LDAP prin precizarea simultană a atributelor *DN* și *cn*.

Vom utiliza *ApacheDS* (*Apache Directory Service*)<sup>1</sup>. Produse pentru administrarea serverului prin intermediul unei interfețe grafice sunt:

- *Apache Directory Studio*;
- *jxplorer*.

**Instalarea** constă din dezarhivarea fișierului descărcat din Internet.

Serverul utilizează portul 10389 și se instalează cu următoarele valori implicite:

Context.PROVIDER_URL	<code>ldap://localhost:10389/uid=admin,ou=system</code>
Context.SECURITY_PRINCIPAL	<code>uid=admin,ou=system</code>
Context.SECURITY_CREDENTIALS	<code>secret</code>

**Lansarea** serverului se realizează prin apelarea fișierului `apacheds.bat` din catalogul `apacheds-*\bin`.

Pentru oprire se tastează Ctrl+C.

Acțiunile care pot fi întreprinse de un client care interacționează cu serverul constau în

- autentificare: datele necesare sunt DN și parola;
- conectare (bind) / deconectare (unbind) la un punct de intrare. Conectarea implică crearea unui punct de intrare precizat prin **cn**;
- căutarea / localizarea (lookup) unui punct de intrare precizat prin **cn**.

**Exemplul 3.1.2** Program pentru înregistrarea și ștergerea referinței unui obiect de tip *Cmmdc*.

<sup>1</sup>[http://en.wikipedia.org/wiki/List\\_of\\_LDAP\\_software](http://en.wikipedia.org/wiki/List_of_LDAP_software).

```

1 import java.util.Hashtable;
2 import java.util.Scanner;
3 import javax.naming.Context;
4 import javax.naming.NamingException;
5 import javax.naming.directory.DirContext;
6 import javax.naming.directory.InitialDirContext;

8 public class LDAPServerCmmdc {
9     public static void main(String[] args) {
10         Hashtable env = new Hashtable();
11         env.put(Context.INITIAL_CONTEXT_FACTORY,
12             "com.sun.jndi ldap.LdapCtxFactory");
13         env.put(Context.PROVIDER_URL,
14             "ldap://localhost:10389/uid=admin,ou=system");
15         env.put(Context.SECURITY_PRINCIPAL, "uid=admin,ou=system");
16         env.put(Context.SECURITY_CREDENTIALS, "secret");
17         DirContext ctx = null;
18         Scanner scanner=new Scanner(System.in);
19         System.out.println("Alegeti operatia : 1- bind; 2- unbind");
20         int oper=scanner.nextInt();
21         System.out.println("Introduceti valoarea atributului \"cn\" "+
22             "a obiectului Cmmdc");
23         System.out.println("cn=");
24         String cmmdcObj=scanner.next().trim();
25         try {
26             ctx = new InitialDirContext(env);
27             if(oper==1){
28                 Cmmdc obj=new Cmmdc();
29                 String str="cn="+cmmdcObj;
30                 ctx.bind(str,obj);
31             }
32             else{
33                 ctx.unbind("cn="+cmmdcObj);
34             }
35             ctx.close();
36         }
37         catch (NamingException e) {
38             System.out.println("LDAPserverCmmdc : "+e.getMessage());
39         }
40     }
41 }

```

Codul clasei *Cmmdc* este

```

1 public class Cmmdc implements java.io.Serializable{
2     public long cmmdc(long a, long b) { . . . }
3 }

```

**Exemplul 3.1.3** *Utilizarea unui obiect de tip Cmmdc regăsit pe baza referintei din serverul LDAP.*

```

1 import java.util.Hashtable;
2 import javax.naming.Context;
3 import javax.naming.NamingException;
4 import javax.naming.directory.DirContext;

```

```
5 import javax.naming.directory.InitialDirContext;  
6 import java.util.Scanner;  
  
8 public class LDAPClientCmmdc {  
9     public static void main(String[] args) {  
10         Hashtable env = new Hashtable();  
11         env.put(Context.INITIAL_CONTEXT_FACTORY,  
12             "com.sun.jndi.ldap.LdapCtxFactory");  
13         env.put(Context.PROVIDER_URL,  
14             "ldap://localhost:10389/uid=admin,ou=system");  
15         env.put(Context.SECURITY_PRINCIPAL, "uid=admin,ou=system");  
16         env.put(Context.SECURITY_CREDENTIALS, "secret");  
17         DirContext ctx = null;  
18         try {  
19             ctx=new InitialDirContext(env);  
20             if (ctx != null) {  
21                 Scanner scanner=new Scanner(System.in);  
22                 System.out.println("Introduceti valoarea atributului \"cn\" "+  
23                     "a obiectului Cmmdc");  
24                 System.out.println("cn=");  
25                 String cmmdcObj=scanner.next().trim();  
26                 Object object = ctx.lookup("cn="+cmmdcObj);  
27                 System.out.println("Primul numar");  
28                 long a=scanner.nextInt();  
29                 System.out.println("Al doilea numar");  
30                 long b=scanner.nextInt();  
31                 Cmmdc obj=(Cmmdc) object;  
32                 System.out.println("Rezultatul cmmdc este: "+obj.cmmdc(a,b));  
33                 ctx.close();  
34             }  
35         }  
36         catch (NamingException e) {  
37             System.out.println("LDAPClientCmmdc : "+e.getMessage());  
38         }  
39     }  
40 }
```

## Întrebări recapitulative

1. Care este rolul unui serviciu JNDI?
2. Ce înseamnă abrevierea JNDI?
3. Care sunt tehnologiile care au utilizat servicii JNDI?

# Capitolul 4

## Invocarea procedurilor la distanță

Invocarea procedurilor la distanță (*Remote Procedure Call – RPC*) înseamnă apelarea unei metode a unui obiect aflat pe un alt calculator ca și cum acesta s-ar afla pe calculatorul local.

Se vor prezenta două cazuri:

- Invocarea procedurilor la distanță în cazul mediului omogen Java. Denumirea tehnologiei în acest caz este *Invocarea metodei la distanță – Remote Method Invocation (RMI)*. Prin mediu omogen se înțelege faptul că atât componenta server cât și componenta client sunt programate în același limbaj de programare, Java în cazul de față.
- Invocarea procedurilor la distanță în cazul medii neomogene. Soluția în acest caz este dat de *Common Object Request Broker Architecture (CORBA)*.

### 4.1 Remote Method Invocation

**Regăsirea obiectelor la distanță.** Ideea găsirii unui obiect la distanță este că serverul înscrie un reprezentant al său – numit *stub* (ciot) – într-un obiect *registry* - registru. Acest obiect se crează cu programul `rmiregistry.exe` din distribuția java și se lansează în execuție prin

```
start rmiregistry [port]
```

unde valoarea implicită a portului este 1099. Comanda `start` aparține sistemului de operare. *rmiregistry* este un serviciu de nume JNDI.

Un client obține din *registry* stub-ul serverului, prin intermediul căruia realizează comunicația cu programul server.

Când un obiect al clientului apelează o metodă a unui obiect aflat la distanță, se va face, de fapt, un apel de metodă a unui obiect care reprezintă serverul. Acesta este *stub*-ul, aflat pe aceeași mașină cu clientul.

Rolul acestui obiect este să împacheteze (marshalling) parametrii de apel ai metodei într-un mesaj ce va fi transferat prin rețea. Împachetarea se face într-o manieră independentă de calculator, mai precis șirurile de caractere și obiecte sunt transmise într-un format care nu se bazează pe referințe. Pentru obiecte se utilizează *serializarea obiectelor Java*.

Serializarea datelor reprezintă transformarea acestora din tipuri de date diferite într-un șir de octeți care va fi transportat prin rețea fără interpretare, dar care păstrează informațiile despre structura inițială a datelor.

Deserializarea este procesul invers de refacere a structurilor trimise prin rețea.

Mesajul asamblat este transmis către server, care știe să desfacă mesajul recepționat invocând în mod corespunzător metoda referită de client.

Atunci când clientul face apel la o metodă aflată pe o altă mașină, este invocat stub-ul client, care începe conversația cu serverul. Acest lucru este complet transparent utilizatorului, care are impresia că invocă o metodă locală.

Metodele apelate la distanță trebuie declarate ca aparținând unei interfețe ce extinde interfața `java.rmi.Remote`. Fiecare asemenea metodă trebuie să arunce o excepție `java.rmi.RemoteException`. Obiectele care circulă prin rețea trebuie să implementeze interfața `java.io.Serializable`.

**Structura unei aplicații RMI.** O aplicație RMI este alcătuită din trei componente:

1. O interfață la distanță (*remote*) în care se declară serviciile puse la dispoziție de server;
2. Aplicația server care poate implementa serviciile interfeței la distanță și înscrie în *registry* stub-ul corespunzător;
3. Aplicația client ce apelează unul sau mai multe servicii ale serverului.

Aplicația server trebuie să aibă acces la clasele interfeței și a celor care o implementează. Locația acestor clase se fixează prin atributul de nume `java.rmi.server.codebase`. Valoarea atributului trebuie indicat printr-unul din protocoalele `http`, `ftp`, `file`.

Utilizând protocoalele `http` sau `ftp` arhiva interfeței și a claselor care o implementează se depun într-un server Web (*Microsoft Internet Interchange*

*Server* (IIS), *apache-tomcat*, respectiv într-un server **ftp** (*apache-ftp*). În momentul lansării aplicației server, serverul **http / ftp** trebuie să fie activ.

Dacă se indică prin protocolul **file** calea către cataloagele care conțin clasele interfeței și ale implementării lor atunci ultimul caracter al căii este **/**.

Din punctul de vedere al execuției sunt implicate componentele:

- Serviciul de nume *rmiregistry*;
- Aplicația server;
- Aplicația client.

**Aplicația server și *rmiregistry* trebuie să ruleze pe același calculator.**

Registrul *rmiregistry* implementează interfața `java.rmi.registry.Registry`<sup>1</sup>. Metodele oferite sunt:

- `void bind(String numeServiciu, Remote obj) throws RemoteException, AlreadyBoundException, AccessException`  
Înregistrează în *registry* obiectul *obj* ce implementează interfața `Remote` sub numele *numeServiciu*.
- `void rebind(String numeServiciu, Remote obj) throws RemoteException, AccessException`  
Reînregistrează în *registry* obiectul *obj* ce implementează interfața `Remote` sub numele *numeServiciu*.  
Această metodă poate fi apelată doar dacă programul care face înregistrarea se află pe aceeași mașină ca registrul *registry*.
- `String[] list() throws RemoteException, AccessException`  
Returnează o listă a tuturor serviciilor înregistrate în *registry*.
- `Remote lookup(String numeServiciu) throws RemoteException, NotBoundException, AccessException`  
Returnează stub-ul serviciului înregistrat sub numele *numeServiciu*.
- `void unbind(String numeServiciu) throws RemoteException, NotBoundException, AccessException`  
Șterge din registru serviciul.

---

<sup>1</sup>Varianța directă, fără a folosi facilitățile JNDI pentru *rmiregistry*.

Localizarea registrului se obține utilizând metoda statică `getRegistry` a clasei `java.rmi.registry.LocateRegistry`.

- `public static Registry getRegistry() throws RemoteException`
- `public static Registry getRegistry(String host) throws RemoteException`
- `public static Registry getRegistry(int port) throws RemoteException`
- `public static Registry getRegistry(String host,int port) throws RemoteException`

Metoda `public static Registry createRegistry(int port) throws RemoteException` a clasei `LocateRegistry` crează un registru la portul specificat pe calculatorul local.

Ansamblul (*rmiregistry*, *port*) determină în mod univoc o aplicație / serviciu.

Interfața `Remote` servește la marcarea interfețelor ale căror metode urmează a fi apelate de pe altă mașină virtuală Java.

Un obiect de tip `java.rmi.server.UnicastRemoteObject` mijlocește transmiterea obiectelor și servește la generarea unui stub unui serviciu. Generarea stub-ului unui serviciu se obține prin intermediul metodei statice

```
static Remote UnicastRemoteObject.exportObject(Remote obj,int
                                              port)
```

**Înregistrarea stub-ului** unui serviciu descris de interfața `IService` și implementat de clasa `ServiceImpl` în *registry* se face prin

```
ServiceImpl obj=new ServiceImpl();
IService stub=(IService)UnicastRemoteObject.exportObject(obj,0);
```

```
// Varianta cu apel rmiregistry direct
/*
Registry registry=LocateRegistry.getRegistry(host,port);
registry.bind("MyServiceName",stub);
*/
// Varianta JNDI
String sPort=(new Integer(port)).toString();
System.setProperty(Context.INITIAL_CONTEXT_FACTORY,
```



```
"com.sun.jndi.rmi.registry.RegistryContextFactory");  
System.setProperty(Context.PROVIDER_URL, "rmi://" + host + ":" + sPort);  
Context ctx = new InitialContext();  
ctx.bind("MyServiceName", stub);
```

Un client care dorește să folosească trebuie să cunoască :

- calculatorul pe care se găsește obiectul *registry* și portul la care ascultă serviciul;
- numele sub care serviciul s-a înregistrat în *registry*;
- metodele puse la dispoziție de serviciu.

#### 4.1.1 Crearea unei aplicații RMI

Exemplificăm construirea unei aplicații RMI în care serviciul asigurat de server este calculul celui mai mare divizor comun a două numere naturale.

Pentru exemplele<sup>2</sup> care urmează, presupunem că textele sursă se rețin în subcataloage ale unui catalog **src**, la care are acces doar programatorul, iar clasele obținute prin compilare se depun în subcataloagele unui catalog **public\classes**, accesibil prin rețea.

Dezvoltarea unei aplicații se va face pe un calculator. În acest sens, celor 3 componente li se asociază cataloagele *\i* - pentru interfața la distanță, *\s* - pentru componenta server și *\c* - pentru componenta client. Pași necesari compilării și desfășurării componentelor aplicației se vor realiza prin intermediul lui *ant*.

Pentru fiecare componentă se vor executa succesiv obiectivele:

1. Install - crează o structură de cataloage **src** și **public\classes**.
2. Init - crează structura de cataloage specifică aplicației.
3. Compile - compilează programele sursă.

Dezvoltarea unei aplicații RMI constă din parcurgerea următorilor pași:

1. Definirea interfeței la distanță.

---

<sup>2</sup>Sistemul de operare utilizat este Windows

```

1 package cmmdc;
2 public interface ICmmdc extends java.rmi.Remote{
3     long cmmdc(long a, long b) throws java.rmi.RemoteException;
4 }

```

Presupunem ca programatorul interfeței reține textul sursă ICmmdc.java în catalogul \i\src\cmmdc.

## 2. Compilarea și arhivarea interfeței se poate realiza fișierul *ant-build*

```

1 <project name="Interface" default="Install" basedir=".">
2   <description> Interface actions </description>
3   <!-- set global properties for this build -->
4
5   <property name="package" value="cmmdc" />
6
7   <target name="Install">
8     <!-- Create the build directory structure used by compile -->
9     <delete dir="src" />
10    <mkdir dir="src" />
11    <delete dir="public" />
12    <mkdir dir="public" />
13    <delete dir="public/classes" />
14    <mkdir dir="public/classes" />
15  </target>
16
17  <target name="Init">
18    <!-- Create the time stamp -->
19    <tstamp/>
20    <mkdir dir="src/${package}" />
21    <mkdir dir="public/classes/${package}" />
22  </target>
23
24  <target name="Compile" depends="Init"
25    description="compile the source " >
26    <javac srcdir="src" includes="${package}/**"
27      destdir="public/classes" includeantruntime="false" />
28    <jar basedir="public/classes"
29      destfile="public/classes/${package}/${package}.jar"
30      includes="${package}/*.class" />
31  </target>
32 </project>

```

## 3. Implementarea interfeței *remote* prin construirea aplicației server.

```

1 package server;
2 import cmmdc.ICmmdc;
3 import java.rmi.server.UnicastRemoteObject;
4 // Varianta cu apel rmiregistry direct
5 import java.rmi.registry.Registry;
6 import java.rmi.registry.LocateRegistry;
7 // Varianta JNDI
8 import javax.naming.Context;

```

```

9 import javax.naming.InitialContext;
11 public class CmmdcImpl implements ICmmdc{
13     public long cmmdc(long a,long b){. . .}
15     public static void main(String args[]) {
16         String host="localhost";
17         int port=1099;
18         if(args.length>0)
19             host=args[0];
20         if(args.length>1)
21             port=Integer.parseInt(args[1]);
22         try {
23             CmmdcImpl obj=new CmmdcImpl();
24             ICmmdc stub=(ICmmdc)UnicastRemoteObject.exportObject(obj,0);
26
27             // Varianta directa
28             /*
29             Registry registry=LocateRegistry.getRegistry(host,port);
30             registry.bind("CmmdcServer",stub);
31             */
32
33             // Varianta JNDI
34             String sPort=(new Integer(port)).toString();
35             System.setProperty(Context.INITIAL_CONTEXT_FACTORY,
36                 "com.sun.jndi.rmi.registry.RegistryContextFactory");
37             System.setProperty(Context.PROVIDER_URL,"rmi://" + host + ":" + sPort);
38             Context ctx=new InitialContext();
39             ctx.bind("CmmdcServer",stub);
40
41             System.out.println("CmmdcServer ready");
42         } catch (Exception e) {
43             System.out.println("CmmdcImpl err: " + e.getMessage());
44         }
45     }
46 }

```

Textul sursă al programului server `CmmdcImpl.java` se reține în catalogul `\s\src\server`

4. (a) Compilarea programului server;
- (b) Crearea obiectului *registry*;
- (c) Lansarea serverului în lucru

se obțin cu *ant* cu

```

1 <project name="Server" default="Install" basedir=".">
2     <description>Server actions </description>
4
5     <property name="path" location="." />
6
7     <property name="package" value="server" />
8     <property name="interface-jar"
9         location="${path}/i/public/classes/cmmdc" />

```

```

9    <property name="jar-file" value="cmmdc.jar" />
10   <property name="service-class" value="CmmdcImpl" />
11   <property name="port" value="1099" />
12   <property name="hostRMIRegistry" value="localhost" />
13   <property name="hostCodebase" value="localhost" />

15   <target name="Install">
16       <!-- Create the time stamp -->
17       <tstamp/>
18       <!-- Create the build directory structure used by compile -->
19       <delete dir="src" />
20       <mkdir dir="src" />
21       <delete dir="public" />
22       <mkdir dir="public" />
23       <delete dir="public/classes" />
24       <mkdir dir="public/classes" />
25   </target>

27   <target name="Init">
28       <mkdir dir="src/${package}" />
29       <mkdir dir="public/classes/${package}" />
30       <copy file="${interface-jar}/${jar-file}" todir="public/classes" />
31   </target>

33   <target name="Compile" depends="Init"
34       description="compile the source " >
35       <javac srcdir="src"
36           includes="${package}/*" destdir="public/classes"
37           classpath="public/classes/${jar-file}"
38           includeantruntime="false" />
39       <unjar src="public/classes/${jar-file}" dest="public/classes" />
40   </target>

42   <target name="Rmi">
43       <exec executable="rmiregistry">
44           <env key="classpath" value="public/classes" />
45           <arg value="${port}" />
46       </exec>
47   </target>

49   <target name="Archive">
50       <jar destfile="cmmdc.jar" basedir="public/classes">
51           <include name="${package}/*" />
52           <include name="cmmdc/*" />
53       </jar>
54   </target>

56   <target name="Server">
57       <java classname="${package}.${service-class}"
58           classpath="${path}/s/public/classes" fork="true">
59
60           <jvmarg value=
61               "-Djava.rmi.server.codebase=file:${path}/s/public/classes/" />
62
63       <!--
64       <jvmarg value=
65           "-Djava.rmi.server.codebase=
66               http://${hostCodebase}:8080/rmi/cmmdc.jar" />
67       -->

```

```

68      <!--
69      <jvmarg value=
70          "-Djava.rmi.server.codebase=
71             ftp:/${hostCodebase}:2121/rmi/cmmdc.jar" />
72      -->
73      <arg line="${hostRMIRegistry} ${port}" />
74      </java>
75      </target>
76 </project>

```

Obiectivele *Rmi* și *Server* se lansează în ferestre **dos** distincte care rămân active pe durata de viață a aplicației server.

Obiectivul *Archive* crează arhiva **jar** necesară desfășurării interfeței și a claselor care o implementează într-un server **http / ftp**. Așa cum s-a amintit mai sus, această arhivă este preluată de aplicația server, la lansare, prin atributul **java.rmi.server.codebase**.

## Server ftp

Utilizăm serverul ftp *apache-ftpserver*. Instalarea constă din dezarhivarea arhivei descărcate. Lansarea serverului se obține prin

```

set FTP_SERVER_HOME=. . .
set JAVA_HOME=. . .
%FTP_SERVER_HOME%\bin\ftpd.bat res/conf/ftpd-typical.xml

```

Resursele / fișierele puse la dispoziție de serverul ftp sunt puse în catalogul **FTP\_SERVER\_HOME\res\home**.

**Serverul RMI cu server ftp.** Se crează arhiva *cmmdc.jar* cu conținutul

```

cmmdc
|    ICmmdc.class
server
|    CmmdcImpl.class

```

care se depune în serverul ftp, în catalogul *rmi*.

## 5. Realizarea programului client

```

1 package client;
2 import cmmdc.ICmmdc;
3 import java.util.Scanner;
4 // Varianta cu apel rmiregistry direct
5 import java.rmi.registry.Registry;

```

```

6 import java.rmi.registry.LocateRegistry;;
7 // Varianta JNDI
8 import javax.naming.Context;
9 import javax.naming.InitialContext;

11 public class CmmdcClient {
12     public static void main(String args[]) {
13         String host="localhost";
14         int port=1099;
15         if (args.length>0)
16             host=args[0];
17         if (args.length>1)
18             port=Integer.parseInt(args[1]);
19         Scanner scanner=new Scanner(System.in);
20         System.out.println("Primul numar :");
21         long m=Long.parseLong(scanner.next());
22         System.out.println("Al doilea numar :");
23         long n=Long.parseLong(scanner.next());
24         long x=0;
25         try {
26             // Varianta directa
27             /*
28             Registry registry=LocateRegistry.getRegistry(host,port);
29             ICmmdc obj=(ICmmdc)registry.lookup("CmmdcServer");
30             */
31             // Varianta JNDI
32             String sPort=(new Integer(port)).toString();
33             System.setProperty(Context.INITIAL_CONTEXT_FACTORY,
34                 "com.sun.jndi.rmi.registry.RegistryContextFactory");
35             System.setProperty(Context.PROVIDER_URL,"rmi://"+host+": "+sPort);
36             Context ctx=new InitialContext();
37             ICmmdc obj=(ICmmdc)ctx.lookup("CmmdcServer");

39             x=obj.cmmdc(m,n);
40             System.out.println("Cmmdc="+x);
41         }
42         catch (Exception e) {
43             System.out.println("CmmdcClient exception: "+e.getMessage());
44         }
45     }
46 }

```

Sursa programului client `CmmdcClient.java` aparține catalogului  
`\c\src\client`

#### 6. Compilarea programului client și lansarea acestuia în execuție.

```

1 <project name="Client" default="Install" basedir=".">
2   <description> Client actions </description>

4   <property name="path" location=". . ."/>

6   <property name="package" value="client" />
7   <property name="interface-jar"
8     location="${path}/i/public/classes/cmmdc"/>
9   <property name="jar-file" value="cmmdc.jar" />
10  <property name="host" value="localhost" />
11  <property name="port" value="1099" />

```

```

12 <property name="client-class" value="CmmdcClient" />
13
14 <target name="Install">
15   <!-- Create the build directory structure used by compile -->
16   <delete dir="src"/>
17   <mkdir dir="src"/>
18   <delete dir="classes"/>
19   <mkdir dir="classes"/>
20 </target>
21
22 <target name="Init">
23   <!-- Create the time stamp -->
24   <tstamp/>
25   <mkdir dir="src/${package}"/>
26   <mkdir dir="classes/${package}"/>
27   <copy file="${interface-jar}\${jar-file}" todir="classes" />
28 </target>
29
30 <target name="Compile" depends="Init"
31   description="compile the source">
32   <javac srcdir="src"
33     includes="${package}/*.java" destdir="classes"
34     classpath="classes\${jar-file}" includeantruntime="false" />
35 </target>
36
37 <target name="Run" depends="Compile"
38   description="Start thr client">
39   <java classname="${package}.${client-class}" fork="true">
40     <classpath>
41       <pathelement location="classes\${jar-file}" />
42       <pathelement path="classes" />
43     </classpath>
44     <arg line="${host} ${port}" />
45   </java>
46 </target>
47 </project>

```

## 4.1.2 Tipare de programare

### Fabrica de obiecte

Tiparul *fabrica de obiecte* va permite crearea dinamică a unui server. Prin aceea schemă, se va crea o clasă - fabrica de obiecte - care implementează câte o metodă *get*, ce returnează o instanță de server. Aceste metode sunt declarate într-o interfață la distanță.

Un client, apelând o asemenea metodă, va instanția serverul dorit și va obține stub-ul corespunzător.

În felul acesta se vor putea utiliza o mulțime de aplicații distincte prin intermediul unui singur *rmiregistry*. Fiecărei aplicații îi corespunde un server, instanțiat la apelul clientului de metoda *get* corespunzătoare.

Programarea unui server care poate fi lansat dinamic trebuie să satisfacă restricțiile:

- Clasa serverului extinde clasa `UnicastRemoteObject`.
- Există un constructor fără argumente ce *aruncă* excepția `java.rmi.RemoteException`.

Pentru exemplificarea tehnicii de lucru reluăm aplicația pentru calculul celui mai mare divizor comun a două numere naturale, considerând interfața

```
1 package cmmdc0;
2 public interface ICmmdc0 extends java.rmi.Remote{
3     public long compute(long m,long n) throws java.rmi.RemoteException;
4 }
```

Această interfață va fi implementată de clasa *ServerCmmdc*. Metoda *compute* este o metodă de calcul a celui mai mare divizor comun a două numere.

Obiecte de tip *ServerCmmdc* se crează, de la distanță prin fabrica de obiecte *FabObiecte*, o clasă ce implementează interfața

```
1 package cmmdc0;
2 public interface IFabObiecte extends java.rmi.Remote{
3     public ICmmdc0 getCmmdc() throws java.rmi.RemoteException;
4 }
```

Codurile claselor *ServerCmmdc* și *FabObiecte* sunt

```
1 package cmmdc0;
2 import java.rmi.RemoteException;
3 import java.rmi.server.UnicastRemoteObject;
4
5 public class ServerCmmdc extends UnicastRemoteObject implements ICmmdc0{
6
7     public ServerCmmdc()throws RemoteException{}
8
9     public long compute(long m,long n)throws RemoteException{
10         . . .
11     }
12 }
```

respectiv

```
1 package cmmdc0;
2 import java.rmi.RemoteException;
3 import java.rmi.server.UnicastRemoteObject;
4 import java.rmi.registry.Registry;
5 import java.rmi.registry.LocateRegistry;
6
7 public class FabObiecte implements IFabObiecte{
8
9     public ICmmdc0 getCmmdc() throws RemoteException{
10         ServerCmmdc cmmdc=new ServerCmmdc();
11         return cmmdc;
12     }
```



```

12 }
13
14 public static void main(String args[]) {
15     String host="localhost";
16     int port=1099;
17     if (args.length>0)
18         host=args[0];
19     try {
20         FabObiecte obj=new FabObiecte();
21         IFabObiecte stub=(IFabObiecte)UnicastRemoteObject.exportObject(obj,0);
22         Registry registry=LocateRegistry.getRegistry(host,port);
23         registry.bind("ObjectFactory",stub);
24         System.out.println("ObjectFactory ready");
25     }
26     catch(Exception e){
27         System.out.println("Factory err "+e.getMessage());
28     }
29 }
30 }

```

Aplicația client se compune din două clase

### 1. *RemoteClient*

Clientul obține stub-ul serviciului, prin intermediul căruia apelează metoda *getCmmdc()* a fabricii de obiecte, obținând un obiect *remote* de tip *ServerCmmdc*, ce implementează interfața la distanță *ICmmdc0*.

```

1 package cmmdc0;
2 import java.rmi.RemoteException;
3 import java.rmi.server.UnicastRemoteObject;
4 import java.rmi.registry.Registry;
5 import java.rmi.registry.LocateRegistry;
6
7 public class RemoteClient extends UnicastRemoteObject {
8
9     ICmmdc0 remote=null;
10
11     public RemoteClient() throws RemoteException {}
12
13     public RemoteClient(String host,int port) throws RemoteException {
14         try {
15             Registry registry=LocateRegistry.getRegistry(host,port);
16             IFabObiecte obj=(IFabObiecte)registry.lookup("ObjectFactory");
17             remote=obj.getCmmdc();
18         }
19         catch (Exception e) {
20             System.out.println("ClientException: "+e.getMessage());
21         }
22     }
23 }

```

### 2. *ClientCmmdc0*

Apelează metoda *compute* a obiectului *remote*.

```

1 package cmmdc0;
2 import java.util.Scanner;

4 public class ClientCmmdc0{

6     public static void main(String args[]){
7         String host="localhost";
8         int port=1099;
9         if(args.length>0)
10             host=args[0];
11         if(args.length>1)
12             port=Integer.parseInt(args[1]);
13         Scanner scanner=new Scanner(System.in);
14         try{
15             RemoteClient ct=new RemoteClient(host,port);
16             System.out.println("m=");
17             long m=scanner.nextLong();
18             System.out.println("n=");
19             long n=scanner.nextLong();
20             long x=ct.remote.compute(m,n);
21             System.out.println("Cmmdc="+x);
22         }
23         catch(Exception e){
24             System.out.println("ClientException: "+e.getMessage());
25         }
26         System.exit(0);
27     }
28 }

```

## Apelul invers – Callback

Se numește apel invers apelarea de către programul server a unei metode a clientului.

În RMI realizarea unui apel invers presupune:

1. definirea unei interfețe la distanță ce va fi implementat de programul client;
2. programul client ce implementează interfața extinde clasa `UnicastRemoteObject` și are un constructor ce aruncă o excepție `java.rmi.RemoteException`.

Extindem aplicația anterioară cu posibilitatea suplimentară a serverului (*ServerCmmdc*) de a alege metoda de calcul a celui mai mare divizor comun dintre varianta imperativă și cea recursivă.

Extindem interfața *ICmmdc0* cu metoda

```
public void setMethod(ICallbackCmmdc obj)throws RemoteException;
```

```

1 package cmmdc0;
2 public interface ICmmdc0 extends java.rmi.Remote{
3     public long compute(long m,long n) throws java.rmi.RemoteException;
4     public void setMethod(ICallbackCmmdc obj)throws java.rmi.RemoteException;
5 }

```

unde *ICallbackCmmdc* este interfața

```

1 package cmmdc0;
2 public interface ICallbackCmmdc extends java.rmi.Remote{
3     public String getMethod() throws java.rmi.RemoteException;
4 }

```

ce va fi implementată în clasa *RemoteClient*.

Programul *ServerCmmdc* devine

```

1 package cmmdc0;
2 import java.rmi.RemoteException;
3 import java.rmi.server.UnicastRemoteObject;
4
5 public class ServerCmmdc extends UnicastRemoteObject
6     implements ICmmdc0{
7     private String method=null;
8
9     public ServerCmmdc()throws RemoteException{}
10
11     public long compute(long m,long n){
12         long x=0;
13         if (method.equals("NERECURSIV"))
14             x=nerecursiv(m,n);
15         if (method.equals("RECURSIV"))
16             x=recursiv(m,n);
17         return x;
18     }
19
20     public void setMethod(ICallbackCmmdc obj)throws RemoteException{
21         method=obj.getMethod();
22     }
23
24     private long recursiv(long a,long b){
25         if (a==b)
26             return a;
27         else
28             if (a<b)
29                 return recursiv(a,b-a);
30             else
31                 return recursiv(a-b,b);
32     }
33
34     private long nerecursiv(long m,long n){
35         long r,c;
36         do{
37             c=n;
38             r=m%n;
39             m=n;
40             n=r;
41         }
42         while(r!=0);

```

```

43     return c;
44 }
45 }

```

După obținerea stub-ului, clientul apelează metoda *setMethod* a serverului *remote*, care, prin apel invers, apelează metoda *getMethod* din *RemoteClient*. Clientul stabilește metoda de calcul și apelează metoda *compute* a lui *remote*.

Programele client devin

```

1 package cmmdc0;
2 import java.rmi.RemoteException;
3 import java.rmi.server.UnicastRemoteObject;
4 import java.rmi.registry.Registry;
5 import java.rmi.registry.LocateRegistry;
6 import java.util.Scanner;

8 public class RemoteClient extends UnicastRemoteObject
9     implements ICallbackCmmdc{
10     ICmmdc0 remote=null;

12     public RemoteClient() throws RemoteException{}

14     public String getMethod(){
15         Scanner scanner=new Scanner(System.in);
16         System.out.println("Alegeti varianta algoritmului lui Euclid");
17         System.out.println("1 - algoritmul ne-recurziv");
18         System.out.println("2 - algoritmul recursiv");
19         int x=scanner.nextInt();
20         String method=null;
21         if(x==1)
22             method="NERECURSIV";
23         else
24             method="RECURSIV";
25         return method;
26     }

28     public RemoteClient(String host,int port)throws RemoteException{
29         try{
30             Registry registry=LocateRegistry.getRegistry(host,port);
31             IFabObiecte obj=(IFabObiecte)registry.lookup("ObjectFactory");
32             remote=obj.getCmmdc();
33         }
34         catch(Exception e){
35             System.out.println("ClientException: "+e.getMessage());
36         }
37     }
38 }

```

și

```

1 package cmmdc0;
2 import java.util.Scanner;

4 public class ClientCmmdc0{
5     public static void main(String args[]){
6         String host="localhost";
7         int port=1099;
8         if(args.length>0)

```

```

9      host=args[0];
10     if (args.length>1)
11         port=Integer.parseInt(args[1]);
12     Scanner scanner=new Scanner(System.in);
13     try{
14         RemoteClient ct=new RemoteClient(host,port);
15         ct.remote.setMethod(ct);
16         System.out.println("m=");
17         long m=scanner.nextLong();
18         System.out.println("n=");
19         long n=scanner.nextLong();
20         long x=ct.remote.compute(m,n);
21         System.out.println("Cmmdc="+x);
22     }
23     catch(Exception e){
24         System.out.println("ClientException: "+e.getMessage());
25     }
26     System.exit(0);
27 }
28 }

```

### 4.1.3 Obiect activabil la distanță

O instanță a clasei `UnicastRemoteObject` reprezintă un obiect la distanță care rulează în permanență, folosind resursele mașinii server.

Începând cu versiunea JDK 1.2, prin introducerea clasei `java.rmi.activation.Activatable` și a programului `rmid (... \jdk... \bin \rmid.exe)` se pot crea programe care înregistrează informații despre obiecte la distanță ce vor fi create și executate la cerere.

Invocarea de la distanță a unei metode aparținând unui asemenea obiect are ca efect activarea obiectului.

Pe fiecare mașină virtuală Java există un grup de activare (activation group), care realizează activarea. Activarea este făcută de un activator. Un activator conține o tabelă care face legătura dintre clasa obiectului cu URL-ul acestuia și eventual, cu date necesare inițializării obiectului. Atunci când activatorul constată că nu există un obiect referit, face apel la grupul de activare care va produce activarea obiectului.

Din punctul de vedere al clientului utilizarea mecanismului de activare la distanță nu implică modificări. Modificările intervin numai din punctul de vedere al serverului și al înregistrării sale.

Reluăm aplicația dezvoltată la începutul capitolului, privind calculul celui mai mare divizor comun a două numere naturale.

Aplicația server este compusă din două clase

1. o clasa ce implementează interfața `ICmmdc : CmmdcActivabil`;

2. clasa *Setup*, cu metoda **main**, care face posibilă mecanismul de activare și înscriere serviciul în *registry*. Această clasă este preluată din tutorialul dedicat activării a documentației ce însoțește distribuția Java.

Clasa ce implementează interfața la distanță trebuie

1. să extindă clasa **Activatable**;
2. să aibă un constructor ce are doi parametri
  - (a) un identificator al grupului de activare, de tip *ActivationID*, utilizat de *demonul* de activare *rmid*;
  - (b) un obiect de tip **MarshaledObject** cu date de inițializare a obiectului activabil. În cazul nostru, acest parametru nu va fi folosit.

Codul sursă al clasei *CmmdcActivabil* este

```

1 package acmmdc;
2 import java.rmi.RemoteException;
3 import java.rmi.MarshaledObject;
4 import java.rmi.activation.Activatable;
5 import java.rmi.activation.ActivationID;
6 import cmmdc.ICmmdc;

8 public class CmmdcActivabil extends Activatable
9     implements ICmmdc{

11     public CmmdcActivabil(ActivationID id, MarshaledObject data)
12         throws RemoteException{
13         super(id, 0);
14     }

16     public long cmmdc(long m, long n){. . .}
17 }
```

Clasa *Setup* necesită o serie de date furnizare ca proprietăți:

1. drepturile de securitate ale grupului de activare

```
myactivation.policy=group.policy;
```

cu conținutul

```

grant codeBase "${myactivation.impl.codebase}" {

    // permission to read and write object's file
    permission java.io.FilePermission "${myactivation.file}", "read,write";

    // permission to listen on an anonymous port
    permission java.net.SocketPermission "*:1024-", "accept";
};
```

2. `java.class.path=no.classpath`

ceea ce previne grupul de activare să încarce o clasă din *classpath*-ul local;

3. localizarea (URL) clasei ce implementează interfața

`myactivation.impl.codebase;`

4. localizarea unui fișier cu date de inițializare a obiectului activabil

`myactivation.file;`

5. numele sub care înregistrează serviciul în *registry*

`myactivation.name.`

Programul *Setup* realizează:

1. Construirea unui descriptor al grupului de activare, instanță a clasei `ActivationGroupDesc`. Grupul de activare este un container ce gestionează obiectele activabile conținute.
2. Înregistrarea descriptorului grupului de activare, instanță a clasei `ActivationDesc` și obținerea unui identificator al grupului de activare, instanță a clasei `ActivationGroupID`.
3. Construirea descriptorului de activare. Trebuie cunoscute
  - idendificatorul grupului de activare;
  - numele clasei ce implementează interfața la distanță;
  - localizarea (URL) clasei ce implementează interfața la distanță;
  - obiectul de tip `MarshallableObject`, cu datele de inițializare a obiectului activabil.
4. Înregistrarea descriptorului de activare, în urma căreia se obține stub-ul obiectului activabil.
5. Înregistrarea stub-ului împreună cu numele serviciului în *registry*.

Codul clasei *setup* este

```

1 package acmmdc;
2 import java.rmi.*;
3 import java.rmi.activation.*;
4 import java.rmi.registry.*;
5 import java.util.Properties;

7 public class Setup{
8     private Setup() {}

10     public static void main(String[] args) throws Exception {
11         // Argumentul liniei de comanda
12         String implClass = "";
13         if (args.length < 1) {
14             System.out.println("usage: ");
15             System.out.println("java [options] acmmdc.Setup <implClass>");
16             System.exit(1);
17         }
18         else {
19             implClass = args[0];
20         }

22         // Construirea descriptorului grupului de activare
23         String policy=System.getProperty("myactivation.policy","group.policy");
24         String implCodebase=System.getProperty("myactivation.impl.codebase");
25         String filename=System.getProperty("myactivation.file", "");
26         Properties props = new Properties();
27         props.put("java.security.policy", policy);
28         props.put("java.class.path", "no_classpath");
29         props.put("myactivation.impl.codebase", implCodebase);
30         if (filename != null && !filename.equals("")) {
31             props.put("myactivation.file", filename);
32         }
33         ActivationGroupDesc groupDesc = new ActivationGroupDesc(props, null);

35         // Inregistrarea grupului de activare pentru obtinerea
36         // identicatorului de activare
37         ActivationGroupID groupID=
38             ActivationGroup.getSystem().registerGroup(groupDesc);
39         System.err.println("Activation group descriptor registered.");

41         // Construirea descriptorului de activare
42         MarshalledObject data = null;
43         if (filename != null && !filename.equals("")) {
44             data = new MarshalledObject(filename);
45         }

47         ActivationDesc desc=
48             new ActivationDesc(groupID, implClass, implCodebase, data);

50         // Inregistrarea descriptorului de activare
51         Remote stub = Activatable.register(desc);
52         System.err.println("Activation descriptor registered.");

54         // Inregistrarea serviciului in registry
55         String name = System.getProperty("myactivation.name");
56         LocateRegistry.getRegistry().rebind(name, stub);
57         System.err.println("Stub bound in registry.");
58     }

```



59 | }

Sursele celor două programe `CmmdcActivabil.java` și respectiv `Setup.java` sunt în catalogul `\c\src\acmmdc`.

Lansarea serverului în execuție constă din

### 1. Pornirea *registry*-ului

```
start rmiregistry
```

### 2. Pornirea *demon*-ului *rmid*

```
start rmid -J-Djava.security.policy=rmid.policy
           -J-Dmyactivation.policy=group.policy
```

unde *rmid.policy* este

```
grant {
    // allow activation groups to use certain system properties
    permission com.sun.rmi.rmid.ExecOptionPermission
        "-Djava.security.policy=${myactivation.policy}";
    permission com.sun.rmi.rmid.ExecOptionPermission
        "-Djava.class.path=no_classpath";
    permission com.sun.rmi.rmid.ExecOptionPermission
        "-Dmyactivation.impl.codebase=*";
    permission com.sun.rmi.rmid.ExecOptionPermission
        "-Dmyactivation.file=*";};
```

iar *group.policy* are codul

```
grant codeBase "${myactivation.impl.codebase}" {
    // permission to read and write object's file
    permission java.io.FilePermission "${myactivation.file}", "read,write";

    // permission to listen on an anonymous port
    permission java.net.SocketPermission " *:1024-", "accept";
};
```

### 3. Lansarea programului *Setup*

```
set classpath=\s\public\classes\cmmdc.jar;\s\public\classes
java -Djava.rmi.server.codebase=file:/s/public/classes/
    -Dmyactivation.impl.codebase=file:/s/public/classes/
    -Dmyactivation.name=CmmdcServer
    -Dmyactivation.file=""
    -Dmyactivation.policy=group.policy
    acmmdc.Setup acmmdc.CmmdcActivabil
```

Localizarea (URL) interfeței la distanță se precizează prin  
`java.rmi.server.codebase;`

Drept client se utilizează programul realizat în secțiunea 4.1.

## 4.2 CORBA

Rețelele de calculatoare sunt eterogene în timp ce majoritatea interfețelor de programare a aplicațiilor sunt orientate spre platforme omogene.

Pentru a facilita integrarea unor sisteme dezvoltate separat, într-un singur mediu distribuit eterogen, OMG (Object Management Group – consorțiu cuprinzând peste 800 de firme) a elaborat standardul CORBA (Common Object Request Broker Architecture): un cadru de dezvoltare a aplicațiilor distribuite în medii eterogene.

La CORBA au aderat toate marile firme de software - cu excepția Microsoft, firmă care a dezvoltat propriul său model DCOM (Distributed Component Object Model), incompatibil CORBA.

Aplicația server se înregistrează într-un **Object Request Broker (ORB)** sub un nume simbolic - nume serviciu. Pe baza acestui nume de serviciu, un client accesând ORB va avea acces la funcțiile oferite de aplicația server.

ORB este un pachet de servicii, independent de aplicații, dar care permit aplicațiilor să interacționeze prin rețea. ORB face parte din *middleware* – un intermediar între softul de rețea și cel de aplicație.

Un ORB se poate executa local pe un singur calculator sau poate fi conectat cu oricare alt ORB din Internet, folosind protocolul **IIOP -- Internet Inter ORB Protocol**, definit de CORBA 2.

Distribuția `jdk` oferă un ORB utilizabil în Java sub forma unui serviciu JNDI.

CORBA face o separare între interfața unui obiect și implementarea sa și folosește un limbaj neutru pentru definirea interfețelor: **IDL -- Interface Definition Language**.

IDL permite realizarea descrierii de interfețe independent de limbajul de programare și de sistemul de operare folosit. O interfață IDL definește legătura dintre client și server.

### 4.2.1 Conexiunea RMI - CORBA

Firma *Oracle - Sun Microsystems* a dezvoltat o soluție prin care interfețele RMI pot fi implementate pentru a putea fi accesate ca obiecte CORBA. Aceasta

este cunoscută sub numele de soluția RMI-IIOP, concretizată printr-o serie de pachete din distribuția jdk. În acest fel nu mai este necesar utilizarea limbajului IDL pentru descrierea interfețelor la distanță.

Instrumente necesare utilizării soluției RMI-IIOP:

- compilatorul `rmic`

Opțiunea `-iiop` generează stub-ul și legătura (`tie`) din partea serverului.

Cu opțiunea `-d` se specifică catalogul în care aceste fișiere sunt scrise.

- serviciul ORB care asigură regăsirea resurselor CORBA. Acest server se lansează în execuție prin

```
start orbd -ORBInitialPort [port]
```

Programele `orbd`, `rmic` sunt în distribuția jdk.

## Dezvoltarea unei aplicații RMI-IIOP

Exemplificăm prin aplicația care implementează un serviciu de calcul a celui mai mare divizor comun a două numere naturale. Etapele realizării aplicației sunt:

1. Elaborarea interfeței este identică cu cea a aplicației RMI.
2. Implementarea interfeței

```
1 package cmmdciiop;
2 import javax.rmi.PortableRemoteObject;
3 import cmmdc.ICmmdc;
4 import java.rmi.RemoteException;
5
6 // Se extinde clasa PortableRemoteObject
7 // si nu UnicastRemoteObject
8
9 public class CmmdcImpl extends PortableRemoteObject implements ICmmdc{
10     // Constructorul clasei
11     public CmmdcImpl() throws RemoteException {}
12
13     public long cmmdc(long a, long b){. . .}
14 }
```

3. *Realizarea programului server.*

```

1 package cmmdciiop;
2 import javax.naming.InitialContext;
3 import javax.naming.Context;

4
5 public class CmmdcServer {
6     public static void main(String[] args) {
7         String host="localhost";
8         String port="1050";
9         if(args.length>0)
10             host=args[0];
11         if(args.length>1)
12             port=args[1];
13         try {
14             // 1: Crearea unei instante CmmdcImpl
15             CmmdcImpl cmmdcRef = new CmmdcImpl();

16
17             // 2: Inregistrarea unei referinte a serviciului
18             // utilizand JNDI API
19             System.setProperty("java.naming.factory.initial",
20                               "com.sun.jndi.cosnaming.CNCtxFactory");
21             System.setProperty("java.naming.provider.url",
22                               "iiop://"+host+":"+port);
23             Context ctx = new InitialContext();
24             ctx.rebind("CmmdcService", cmmdcRef);
25             System.out.println("Cmmdc Server: Ready...");
26         }
27         catch (Exception e) {
28             System.out.println("CmmdcServer: " + e.getMessage());
29         }
30     }
31 }

```

4. Compilarea programelor *CmmdcImpl.java* și *CmmdcServer.java*.
5. Generarea stub-ului *cmmdc.\_ICmmdc\_Stub.class* corespunzător interfeței *ICmmdc* și a fișierului *\_Tie cmmdciiop.\_CmmdcImpl.Tie.class* corespunzător clasei *CmmdcImpl*. Acestea se obțin rulând utilitarul **rmic** - din distribuția Java – cu opțiunea **-iiop**

```
rmic -iiop cmmdciiop.CmmdcImpl
```

6. Pornirea serverului CORBA de regăsire a serviciilor

```
start orbd -ORBInitialPort 1050
```

7. Lansarea serverului în execuție.

Activitățile legate de server se obțin prin **ant** cu fișierul *build*

```

1 <project name="Server" default="Install" basedir=". ">
2   <description>Server actions</description>

4   <property name="path" location=". . ." />

6   <property name="package" value="cmmhciop" />
7   <property name="interface-package" value="cmmhc" />
8   <property name="interface-jar"
9     location="${path}/i/public/classes/cmmhc" />
10  <property name="jar-file" value="cmmhc.jar" />
11  <property name="service-class" value="CmmhcServer" />
12  <property name="implementation-class" value="CmmhcImpl" />
13  <property name="host" value="localhost" />
14  <property name="port" value="1050" />

16  <target name="Install">
17    <!-- Create the time stamp -->
18    <tstamp/>
19    <!-- Create the build directory structure used by compile -->
20    <delete dir="src" />
21    <mkdir dir="src" />
22    <delete dir="public" />
23    <mkdir dir="public" />
24    <delete dir="public/classes" />
25    <mkdir dir="public/classes" />
26  </target>

28  <target name="Init">
29    <mkdir dir="src/${package}" />
30    <mkdir dir="public/classes/${package}" />
31    <copy file="${interface-jar}/${jar-file}" todir="public/classes" />
32    <unjar src="public/classes/${jar-file}" dest="public/classes" />
33    <delete dir="public/classes/META-INF" />
34  </target>

36  <target name="Compile" depends="Init"
37    description="compile the source " >
38    <javac srcdir="src" destdir="public/classes"
39      includes="${package}*"
40      classpath="public\classes\${jar-file}"
41      includeantruntime="false" />
42    <rmic classname="${package}.${implementation-class}"
43      sourcebase="src"
44      iio="yes"
45      base="public/classes"
46      classpath="public/classes" />
47  </target>

49  <target name="Orb">
50    <exec executable="orbd">
51      <arg line="-ORBInitialPort ${port} -ORBInitialHost ${host}" />
52    </exec>
53  </target>

55  <target name="Server" description="Start thr server" >
56    <java classname="${package}.${service-class}" fork="true">
57      <classpath>
58        <pathelement location="public/classes/${jar-file}" />

```

```

59         <pathelement path="public/classes" />
60     </classpath>
61     <arg line="{host} {port}" />
62 </java>
63 </target>
64 </project>

```

*orbd* și aplicația server se pot afla pe calculatoare distincte.

#### 8. Editarea programului client.

```

1 package cmmdciiop;
2 import javax.rmi.PortableRemoteObject;
3 import javax.naming.Context;
4 import javax.naming.InitialContext;
5 import java.util.Scanner;
6 import cmmdc.ICmmdc;

8 public class CmmdcClient {
9     public static void main( String args[] ) {
10         String host="localhost";
11         String port="1050";
12         if (args.length>0)
13             host=args[0];
14         if (args.length>1)
15             port=args[1];

17         Scanner scanner=new Scanner(System.in);
18         System.out.println("Primul numar :");
19         long m=Long.parseLong(scanner.next());
20         System.out.println("Al doilea numar :");
21         long n=Long.parseLong(scanner.next());
22         try {
23             System.setProperty("java.naming.factory.initial",
24                               "com.sun.jndi.cosnaming.CNCtxFactory");
25             System.setProperty("java.naming.provider.url",
26                               "iiop://" + host + ":" + port);
27             Context ctx = new InitialContext();

29             // STEP 1: Get the Object reference from the Name Service
30             // using JNDI call.
31             Object objref = ctx.lookup("CmmdcService");
32             System.out.println("Client: Obtained a ref. to Cmmdc server.");

34             // STEP 2: Narrow the object reference to the concrete type and
35             // invoke the method.
36             ICmmdc obj=(ICmmdc)PortableRemoteObject.narrow(objref,ICmmdc.class);
37             long x=obj.cmmdc(m,n);
38             System.out.println("Cmmdc="+x);
39         }
40         catch( Exception e ) {
41             System.out.println("Exception " + e.getMessage());
42         }
43     }
44 }

```

9. Compilarea și lansarea clientului în execuție. Clientul trebuie să dispună de fișierul stub (`cmmdc._ICmmdc_Stub.class`) și bineînțeles de interfața `ICmmdc.jar`.

Fișierul *buildfile* pentru executarea clientului:

```

1 <project name="Client" default="Install" basedir=". ">
2   <description>Client actions</description>

4   <property name="path" location=". . ." />

6   <property name="package" value="cmmdciop" />
7   <property name="interface-jar"
8     location="${path}/i/public/classes/cmmdc" />
9   <property name="jar-file" value="cmmdc.jar" />
10  <property name="server-package" value="cmmdciop" />
11  <property name="interface-package" value="cmmdc" />
12  <property name="interface-stub-location"
13    location="${path}/s/public/classes/${interface-package}" />
14  <property name="stub-class" value="*_Stub.class" />
15  <property name="host" value="localhost" />
16  <property name="port" value="1050" />
17  <property name="client-class" value="CmmdcClient" />

19  <target name="Install">
20    <!-- Create the build directory structure used by compile -->
21    <delete dir="src" />
22    <mkdir dir="src" />
23    <delete dir="classes" />
24    <mkdir dir="classes" />
25  </target>

27  <target name="Init">
28    <!-- Create the time stamp -->
29    <tstamp/>
30    <mkdir dir="src/${package}" />
31    <mkdir dir="classes/${package}" />
32    <delete dir="classes/${server-package}" />
33    <mkdir dir="classes/${server-package}" />
34    <delete dir="classes/${interface-package}" />
35    <mkdir dir="classes/${interface-package}" />
36    <copy file="${interface-jar}/${jar-file}" todir="classes" />
37    <copy todir="classes/${interface-package}" >
38      <fileset dir="${interface-stub-location}"
39        includes="${stub-class}" />
40    </copy>
41    <unjar src="classes/${jar-file}" dest="classes" />
42    <delete dir="classes/META-INF" />
43  </target>

45  <target name="Compile" depends="Init"
46    description="compile the source " >
47    <javac srcdir="src" destdir="classes"
48      includes="${package}/*.java" classpath="classes"
49      includeantruntime="false" />
50  </target>

52  <target name="Run" depends="Compile" description="Run the client" >
53    <java classname="${package}.${client-class}" fork="true">

```

```

54     <classpath>
55         <pathelement location="classes/${jar-file}" />
56         <pathelement path="classes" />
57     </classpath>
58     <arg line="${host} ${port}" />
59 </java>
60 </target>
61 </project>

```

### 4.2.2 Aplicație Java prin CORBA

Scopul acestei secțiuni este prezentarea dezvoltării unei aplicații pe baza unei interfețe bazat pe IDL.

Pentru dezvoltarea aplicațiilor în limbajul de programare Java, translatarea interfeței IDL în Java se realizează cu utilitarul `idlj` din distribuția *jdk*.

Correspondența între entitățile IDL și Java este dată în Tabelul 4.1

Tip IDL	Tip Java
module	package
boolean	boolean
char, wchar	char
octet	byte
string, wstring	java.lang.String
short, unsigned short	short
long, unsigned long	int
long long, unsigned long long	long
float	float
double	double
fixed	java.math.BigDecimal
enum, struct, union	class
sequence, array	array
interface (non-abstract)	signature interface, operations interface, helper class, holder class
Any	org.omg.CORBA.Any

Table 4.1: Entități IDL și Java

Legarea cererii unui client de codul serviciului care satisface cererea utilizează componenta CORBA *Portable Object Adapter* (POA).



## Model cu server temporal

Exemplificăm dezvoltarea unei aplicații distribuite CORBA în cazul în care serviciul pus la dispoziție de server este calculul celui mai mare divizor comun a două numere naturale.

Dezvoltarea unei aplicații distribuite CORBA cu mediul nativ Java presupune parcurgerea următoarelor pași:

### 1. Realizarea interfeței IDL care înseamnă

#### (a) Editarea programului de interfață:

```

1 module CmmdcApp{
2     interface Cmmdc{
3         long long cmmdc(in long long a,in long long b);
4     };
5 };

```

Salvăm acest text într-un fișier denumit `Cmmdc.idl`.

*Cmmdc* va fi numele interfeței utilizat de un client și implementat de server. Serviciul conține o singură metodă *cmmdc*.

#### (b) Traducerea interfeței în Java

```
idlj -fall Cmmdc.idl
```

Programul `idlj` crează un subcatalog `CmmdcApp` cu un pachet Java `CmmdcApp` conținând fișierele:

- `Cmmdc.java`
- `CmmdcPOA.java` ;
- `CmmdcOperations.java`;
- `_CmmdcStub.java`;
- `CmmdcHelper.java`;
- `CmmdcHolder.java`.

### 2. Realizarea programelor server. Punem în evidență programul servant *CmmdcImpl.java* ce implementează interfața *Cmmdc*.

```

1 import CmmdcApp.*;
2 import org.omg.CORBA.ORB;
3
4 public class CmmdcImpl extends CmmdcPOA {
5     private ORB orb;
6 }

```

```

7   public CmmdcImpl(ORB orb){
8       this.orb = orb;
9   }

11  public long cmmdc(long a, long b){. . .}
12 }

```

și programul *CmmdcServer.java*, care înscrie în registrul ORB referințele servantului. Activitățile ce trebuie întreprinse sunt declarate prin comentarii în textul sursă al programului

```

1  import CmmdcApp.*;
2  import org.omg.CosNaming.NameComponent;
3  import org.omg.CosNaming.NamingContextExtHelper;
4  import org.omg.CosNaming.NamingContextExt;
5  import org.omg.CORBA.ORB;
6  import org.omg.PortableServer.POA;
7  import org.omg.PortableServer.POAHelper;

9  public class CmmdcServer {
10     public static void main(String args[]) {
11         try{
12             // Creare si initializare ORB
13             ORB orb = ORB.init(args, null);

15             // Obtinerea unei referinte POA si
16             // activarea gestionarului POAManager
17             POA rootPOA =
18                 POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
19             rootPOA.the_POAManager().activate();

21             // Crearea unui servent
22             CmmdcImpl cmmdcImpl = new CmmdcImpl(orb);

24             // Obtinerea unei referinte pentru servent
25             org.omg.CORBA.Object ref=rootPOA.servant_to_reference(cmmdcImpl);
26             Cmmdc href = CmmdcHelper.narrow(ref);

28             // Obtinerea serviciului NameService
29             org.omg.CORBA.Object objRef =
30                 orb.resolve_initial_references("NameService");
31             NamingContextExt ncRef=NamingContextExtHelper.narrow(objRef);

33             // Legarea servantului in NameService
34             String name = "CmmdcService";
35             NameComponent path[] = ncRef.to_name(name);
36             ncRef.rebind(path, href);

38             System.out.println("CmmdcServer ready and waiting ...");

40             // Gata pentru satisfacerea clientilor
41             orb.run();
42         }
43         catch (Exception e) {
44             System.err.println("ERROR: " + e.getMessage());
45         }
46         System.out.println("CmmdcServer Exiting ...");

```

```

47 }
48 }

```

Această clasă corespunde unui șablon de programare adaptat exemplului tratat. În acest caz numele serviciului înregistrat în ORB va fi *Cmmdc-Service*. Evidența numelor serviciilor CORBA înregistrate în ORB este făcută de serviciul *NameService*.

### 3. Realizarea programului client *CmmdcClient.java*:

```

1 import CmmdcApp.*;
2 import org.omg.CosNaming.NamingContextExtHelper;
3 import org.omg.CosNaming.NamingContextExt;
4 import org.omg.CORBA.ORB;
5 import java.util.Scanner;

7 public class CmmdcClient{
8     static Cmmdc cmmdc;

10     public static void main(String args[]){
11         try{
12             // crearea si initializarea unui reprezentant ORB
13             ORB orb = ORB.init(args, null);

15             // obtinerea unei referinte pentru serviciul denumirilor
16             // serviciilor inregistrate
17             org.omg.CORBA.Object objRef =
18                 orb.resolve_initial_references("NameService");
19             NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

21             // obtinerea unei referinte la serviciul dorit
22             String name = "CmmdcService";
23             cmmdc = CmmdcHelper.narrow(ncRef.resolve_str(name));

25             System.out.println("Obtained a handle on server object: " +
26                 cmmdc);
27             Scanner scanner=new Scanner(System.in);
28             long m,n;
29             System.out.println("m=");
30             m=scanner.nextLong();
31             System.out.println("n=");
32             n=scanner.nextLong();
33             System.out.println("Cmmdc="+cmmdc.cmmdc(m,n));
34         }
35         catch (Exception e) {
36             System.out.println("ERROR : " + e) ;
37             e.printStackTrace(System.out);
38         }
39     }
40 }

```

Din nou clasa client conține șablonul de accesare a unui serviciu CORBA.

Programele se compilează

```
javac CmmdcApp\*.java
```

4. Pornirea serviciului de intregistrare a numelor cu programul `orbd.exe` din distribuția *jdk*.

```
start orbd -ORBInitialHost localhost -ORBInitialPort 1050
```

5. Pornirea programului server prin

```
start java CmmdcServer -ORBInitialHost localhost -ORBInitialPort 1050
```

6. Lansarea programului client prin

```
java CmmdcClient -ORBInitialHost localhost -ORBInitialPort 1050
```

## Model cu server persistent

Se consideră aceeași interfață *Cmmdc.idl*.

Partea de server este alcătuită din clasa servant *CmmdcImpl.java* care implementează interfața *Cmmdc* -prezentat în secțiunea anterioară și clasa *PersistentServer* care asigură

- legătura cu serviciile ORB;
- accesul la clasa servantului.

```
1 import java.util.Properties;
2 import org.omg.CORBA.ORB;
3 import org.omg.CORBA.Policy;
4 import org.omg.CosNaming.NamingContextExtHelper;
5 import org.omg.CosNaming.NamingContextExt;
6 import org.omg.CosNaming.NameComponent;
7 import org.omg.PortableServer.POA;
8 import org.omg.PortableServer.POAHelper;
9 import org.omg.PortableServer.LifespanPolicyValue;

11 public class PersistentServer{
12     public static void main( String args[] ) {
13         Properties properties = System.getProperties();
14         properties.put("org.omg.CORBA.ORBInitialHost","localhost");
15         properties.put("org.omg.CORBA.ORBInitialPort","1050");
16         try {
17             // Pas 1: Creare si initializare ORB
```

```

18     ORB orb = ORB.init(args, properties);

20     // Pas 2: Crearea unui servent
21     CmmdcImpl cmmdcImpl = new CmmdcImpl(orb);

23     // Pas 3: Obținerea unei referințe POA și
24     //         activarea gestionarului POAManager
25     // *****
26     // Pas 3-1: Obținerea radacinii rootPOA
27     POA rootPOA =
28         POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
29     // Pas 3-2: Create securitatii Persistent Policy
30     Policy[] persistentPolicy = new Policy[1];
31     persistentPolicy[0] = rootPOA.create_lifespan_policy(
32         LifespanPolicyValue.PERSISTENT);
33     // Pas 3-3: Crearea obiectului POA cu securitatea the Persistent Policy
34     POA persistentPOA = rootPOA.create_POA("childPOA", null, persistentPolicy);
35     // Pas 3-4: Activarea managerului PersistentPOA POAManager,
36     persistentPOA.the_POAManager().activate();
37     // *****

39     // Pas 4: Asocierea serventului cu PersistentPOA
40     persistentPOA.activate_object(cmmdcImpl);

42     // Pas 5: Fixarea contextului RootNaming și
43     //         legarea de numele serventului
44     // Numele serviciului de nume : 'NameService'
45     // Numele serventului 'PersistentCmmdcServer'

47     org.omg.CORBA.Object objRef=
48         orb.resolve_initial_references("NameService");
49     NamingContextExt rootContext=NamingContextExtHelper.narrow(objRef);
50     NameComponent[] nc = rootContext.to_name("PersistentCmmdcServer");
51     rootContext.rebind(nc, persistentPOA.servant_to_reference(cmmdcImpl));

53     // Pas 6: Gata pentru satisfacerea clientilor
54     orb.run();
55 }
56 catch (Exception e) {
57     System.err.println("Exception in Persistent Server Startup "+
58         e.getMessage());
59 }
60 }
61 }

```

### Programul client

```

1 import CmmdcApp.*;
2 import org.omg.CORBA.ORB;
3 import java.util.Scanner;

5 public class PersistentClient {
6     public static void main(String args[]) {
7         String host="localhost";
8         String port="1050";
9         if(args.length>0)
10             host=args[0];
11         if(args.length>1)
12             port=args[1];

```

```

13     try {
14         // Pas 1: Initializare ORB
15         ORB orb = ORB.init( args , null );

17         // Pas 2: Rezolvarea persistentei
18         // Serviciul NameService ruleaza pe host cu portul port
19         // Numele serviciului cerut lui NameService este
20         // "PersistentCmmdcServer"
21         org.omg.CORBA.Object obj = orb.string_to_object(
22             "corbaname::" + host + ":" + port + "#PersistentCmmdcServer" );
23         Cmmdc cmmdc = CmmdcHelper.narrow( obj );

25         // Pas 3: Utilizarea serviciului
26         Scanner scanner = new Scanner( System.in );
27         System.out.println( "Calling Persistent Server.." );
28         int m, n, r;
29         System.out.println( "m=" );
30         m = scanner.nextInt();
31         System.out.println( "n=" );
32         n = scanner.nextInt();
33         System.out.println( cmmdc.cmmdc(m, n) );
34     }
35     catch ( Exception e ) {
36         System.err.println( "Exception in PersistentClient.java..." +
37             e.getMessage() );
38     }
39 }
40 }

```

Serverul trebuie să fie pe același calculator pe care rulează `orbd`. Executarea aplicației presupune:

1. Pornirea serviciului de înregistrare a numelor cu programul `orbd.exe` din distribuția *jdk*.

```
orbd -ORBInitialPort 1050 -serverPollingTime 200
```

2. Activarea serverului:

- (a) Se lansează utilitarul `servertool`

```
servertool -ORBInitialPort 1050
```

- (b) Se înregistrează serverul împreună cu numele serviciului pe care îl îndeplinește:

```

servertool > register -server PersistentServer
-applicationName PersistentCmmdcServer
-classpath cale_catre_fisierele_server\

```

Îndeplinirea acțiunii de înregistrare a serverului este indicat printr-un mesaj de forma `server registered (serverid=257)`.

Alte comenzi `servertool`

Comanda	Semnificație
<code>servertool&gt; shutdown -serverid 257</code>	Oprirea serviciului
<code>servertool&gt; unregister -serverid 257</code>	Ștergerea serviciului
<code>servertool&gt; quit</code>	Închiderea utilitarului
<code>servertool&gt; help</code>	

**Observație 4.2.1** *Dacă se utilizează referința*

*`c:\Program Files\Java\jdk1.7.0_*\bin`*

*în variabila de sistem PATH (în loc de `c:\Program Files\Java\jdk1.7.0_*\bin`) atunci apelarea serviciilor `orbd` și `servertool` se va face prin*

```
c:\Program Files\Java\jdk1.7.0_*\bin\orbd -ORBInitialPort 1050 -serverPollingTime 200
c:\Program Files\Java\jdk1.7.0_*\bin\servertool -ORBInitialPort 1050
```

### 3. Executarea clientului

```
java PersistentClient [hostORB [portORB]]
```

## Întrebări recapitulative

1. Precizați abrevierea RMI.
2. Care sunt componentele unei aplicații client-server bazat pe RMI?
3. Câte și care sunt entitățile care participă la execuția unei aplicații client-server bazat pe RMI?
4. Care este rolul îndeplinit de *(rmi)registry*?
5. Ce asigură șablonul de programare *Fabrica de obiecte* în RMI?
6. Ce asigură șablonul de programare *Callback* în RMI?
7. Explicați caracterul sincron al comunicării în RMI.

8. Care sunt activitățile pe care un server RMI le are de îndeplinit ?
9. Care sunt componentele unei aplicații client server bazat pe CORBA?
10. Câte și care sunt entitățile care participă la execuția unei aplicații client-server bazat pe CORBA?
11. Care este ideea soluției CORBA pentru asigurarea interoperabilității între client și server realizate în limbaje de programare diferite.
12. Explicați caracterul sincron al comunicării în CORBA.
13. Ce oferă RMI-IIOP ?
14. Precizați modelele de aplicații CORBA prezentate în curs și diferența dintre ele din punctul de vedere al execuției.



# Capitolul 5

## Mesaje în Java

Comunicația prin apelul la distanță - inclusiv RMI - este sincron: programul apelant se blochează și așteaptă ca metoda apelată să se termine și să furnizeze rezultatul cerut. Cu alte cuvinte apelul de procedură la distanță cere atât clientului cât și serverului să fie simultan disponibile.

Comunicațiile asincrone<sup>1</sup> între programe permit realizarea unor sisteme de programe cu un grad mult mai scăzut de cuplare, în sensul că lansarea unei cereri și recepționarea rezultatului se pot executa în momente diferite de timp. Asemenea aplicații se pot realiza prin comunicații de mesaje, mesaje care sunt reținute de un intermediar (serviciu de mesagerie, server, broker, messaging middleware).

### 5.1 Java Message Service (JMS)

JMS definește un cadru de programare Java (API) pentru realizarea aplicațiilor bazate pe comunicații asincrone.

Prezentarea se bazează pe interfața de programare (API) JMS-2, pentru care există implementarea de referință *Open Message Queue 5.\** de la Oracle.

Pentru interfața de programare (API) JMS-1, există mai multe implementări JMS, dintre care semnalăm:

- *Open Message Queue \**
- *apache ActiveMQ*
- *apache qpid*

---

<sup>1</sup>În accepțiunea din acest capitol.

Aceste produse implementează una sau mai multe protocoale privind reprezentarea datelor unui mesaj și detalii privind transmiterea lor. *Advanced Message Queue Protocol* (AMQP), *Streaming Text Oriented Messaging Protocol* (STOMP) sunt exemple de asemenea protocoale. Scopul unui protocol este asigurarea interoperabilității între aplicații de mesagerie realizate în diferite limbaje de programare, produse sau platforme de calcul.

O aplicație JMS este alcătuită din

- *un furnizor JMS (provider JMS)* : un sistem de mesagerie ce implementează specificațiile JMS;
- *client JMS* : aplicație Java care trimite și recepționează mesaje;
- *mesaje* : obiecte utilizate în schimbul de informații de clienți JMS;
- *obiecte administrator* - obiecte (resurse) create de administrator pentru a fi utilizate de clienții JMS, precum
  - fabrica de conexiuni,
  - obiectele destinație pentru reținerea mesajelor.

Aplicațiile dezvoltate pe baza JMS 1/JMS 1.1 funcționează și în cazul utilizării unui furnizor de servicii realizat pentru JMS-2. În esență, JMS-2 simplifică programarea față de JMS-1.

#### **Modele de comunicație:**

- *Comunicații punctuale* : Un mesaj este generat de un producător (expeditor) și la care va avea acces un singur consumator (destinatar). Mesajul este depus într-o coadă, de unde este preluat de către consumatorul care s-a legat de coadă. Dacă de coadă nu se leagă nici un consumator, atunci mesajul este păstrat în coadă.
- *Comunicații axate pe subiect (topic)* : Mesajele sunt depuse (publicate) în destinații specifice subiectului. Consumatorii ce au subscris la acel subiect au acces la mesajele respective. Mai multi producători pot genera mesaje specifice unui subiect, mesaje care pot fi accesate de consumatorii care au subscris subiectului.

Consumatorii abonați pe un subiect pot fi

- abonat simplu - valabi, adică are acces la mesajele emise după momentul abonării, pe durata conexiunii la furnizorul de servicii JMS;

- abonament permanent - valabil și după o întrerupere a conexiunii la furnizorul de servicii JMS;
- abonament partajat - mai mulți abonați utilizează aceeași conexiune;
- abonament partajat și permanent.

**Structura unui mesaj** Un mesaj este alcătuit din

- *Antet (header)* : conține informații pentru identificarea destinației cât și pentru identificarea mesajului.
- *Proprietăți* : au caracter opțional și sunt sub forma (nume, valoare). Proprietățile ajută consumatorii să selecteze mesajele.
- *Corpul mesajului* : opțional. Potrivit specificațiilor JMS există 6 tipuri de mesaje.
  - `javax.jms.Message` : mesaj fără corp;
  - `javax.jms.StreamMessage` : corpul mesajului conține un flux Java de date de tip predefinit;
  - `javax.jms.BytesMessage` :
  - `javax.jms.MapMessage` : corpul mesajului conține o familie de perechi (nume, valoare);
  - `javax.jms.TextMessage` : corpul mesajului conține un string;
  - `javax.jms.ObjectMessage` : corpul mesajului conține un obiect serializat.
- *Atașament* (opțional).

## 5.2 Open Message Queue 5

**Instalarea variantei de sine stătătoare.** În mediul Windows, în funcție de resursa descărcată instalarea constă din:

- Se dezarchivează fișierul `openmq*-binary-Windows.zip`
- Se fixează atributele fișierului `etc\mq\mqenv.conf`.

**Lansarea serviciului JMS.** Orice aplicație JMS necesită funcționarea serviciului JMS. Serviciul JMS se instalează prin

`mqbrokerd`

Funcționarea corectă este indicată prin mesajul

*imqbroker@hostname:7676 ready*

Dacă se dorește schimbarea portului atunci se folosește opțiunea `-port port`.

Pe un calculator pot coexista mai multe servicii JMS doar dacă folosesc porturi distincte și au nume diferite. În acest caz, lansarea unui nou serviciu se face prin

```
imqbrokerd -port port -name name
```

Cu utilitarul `imqsvcadmin` putem

- dezinstala : `imqsvcadmin remove`
- verifica : `imqsvcadmin query`
- instala : `imqsvcadmin install`

serviciul JMS ca serviciu Windows. Dezinstalarea și instalarea are efect odată cu repornirea calculatorului.

**Compilarea și executarea** unui program necesită completarea variabilei sistem `classpath` cu fișierele

- `JMS_HOME\lib\jms.jar`
- `JMS_HOME\lib\imq.jar`

## 5.3 Elemente de programare - JMS-2

Semnalăm două modalități de programare / generare / regăsire a obiectelor administrator:

- Programat: obiectele administrator se instanțiază prin API-ul oferit de produsul informatic;
- Prin JNDI. Codul sursă al programelor este independent de furnizorul de servicii de mesagerie. Elementele specifice se declară în fișiere de proprietăți.

### 5.3.1 Modul programat: Trimiterea unui mesaj

Trimiterea unui mesaj necesită:

1. Generarea obiectelor administrator

(a) *Fabrici de conexiuni.*

```
com.sun.messaging.ConnectionFactory cf=  
    new com.sun.messaging.ConnectionFactory();
```

Clasa `com.sun.messaging.ConnectionFactory` are descendenții:

- `QueueConnectionFactory`
- `TopicConnectionFactory`

(b) *Obiectul destinație.* În cazul comunicațiilor punctuale obiectul destinație este de tip `javax.jms.Queue`, iar crearea se face prin

```
Queue q=new com.sun.messaging.Queue(numeCoadă);
```

Pentru comunicații axate pe subiect, obiectul destinație, de tip `javax.jms.Topic` se instanțiază prin

```
Topic t=new com.sun.messaging.Topic(subiect);
```

2. Instanțierea unui obiect de tip `javax.jms.JMSContext`

```
JMSContext ctx=cf.createContext();
```

În final, contextul se închide

```
ctx.close();
```

Prin intermediul unui obiect de tip `JMSContext` se obțin producătorii de mesaje, de tip `javax.jms.JMSProducer`, dar și consumatorii de mesaje, de tip `javax.jms.JMSConsumer`.

## 3. Instanțierea unui producător de mesaje

```
JMSProducer producer=ctx.createProducer();
```

Crearea și expedierea mesajelor se realizează cu metodele clasei `JMSProducer`

- `JMSProducer send(Destination destination, byte[] body)`
- `JMSProducer send(Destination destination, Serializable body)`
- `JMSProducer send(Destination destination, Map<String, Object> body)`

- `JMSProducer send(Destination destination, String body)`
- `JMSProducer send(Destination destination, Message message)`

Un obiect care implementează interfața `javax.jms.Message` se obține prin `ctx.createMessage()`.

**Exemplul 5.3.1** *Clasa următoare generează într-un fir de execuție, un număr de mesaje de tip **TextMessage** într-o comunicație punctuală. Sfârșitul expedierii mesajelor se indică prin generarea unui mesaj de tip **Message**. Numărul mesajelor este indicat de un parametru al constructorului.*

```

1 import javax.jms.Queue;
2 import javax.jms.JMSContext;
3 import javax.jms.JMSProducer;

5 public class MsgSenderT extends Thread{
6     int n;
7     String queueName;

9     MsgSenderT(String queueName,int n){
10         this.queueName=queueName;
11         this.n=n;
12     }

14     public void run(){
15         try{
16             com.sun.messaging.QueueConnectionFactory cf=
17                 new com.sun.messaging.QueueConnectionFactory();
18             //cf.setProperty("imqBrokerHostName","host");
19             //cf.setProperty("imqBrokerHostPort","7676");
20             Queue q=new com.sun.messaging.Queue(queueName);
21             JMSContext ctx=cf.createContext();
22             JMSProducer producer=ctx.createProducer();

24             for(int i=0;i<n;i++){
25                 producer.send(q,"Hello "+i);
26             }
27             producer.send(q,ctx.createMessage());
28             ctx.close();
29         }
30         catch(Exception e){
31             System.out.println("JMSEException : "+e.getMessage());
32         }
33         System.out.println("Sender finished");
34     }
35 }

```

### 5.3.2 Recepția sincronă a unui mesaj

Primele două acțiuni sunt identice cu cele de la trimiterea unui mesaj:

1. Generarea obiectelor administrator.

2. Instanțierea unui obiect de tip `javax.jms.JMSContext`.
3. Instanțierea unui consumator de mesaje

```
JMSConsumer consumer = ctx.createConsumer(q);
```

Recepția unui mesaj se obține cu una din metode

- `Message receive()`  
Recepție blocantă.
- `Message receive(long timeout)`  
Recepție într-un interval de timp.
- `Message receiveNoWait()`  
Recepție neblocantă.

**Exemplul 5.3.2** *Recepția de tip sincron a mesajelor într-un fir de execuție este efectuat de clasa următoare:*

```

1 import javax.jms.TextMessage;
2 import javax.jms.Message;
3 import javax.jms.Queue;
4 import javax.jms.JMSContext;
5 import javax.jms.JMSConsumer;

7 public class SyncMsgReceiverT extends Thread{
8     String queueName;

10     SyncMsgReceiverT(String queueName){
11         this.queueName=queueName;
12     }

14     public void run(){
15         try{
16             com.sun.messaging.QueueConnectionFactory cf=
17                 new com.sun.messaging.QueueConnectionFactory();
18             //cf.setProperty("imqBrokerHostName","host");
19             //cf.setProperty("imqBrokerHostPort","7676");
20             Queue q=new com.sun.messaging.Queue(queueName);
21             JMSContext ctx=cf.createContext();
22             JMSConsumer consumer = ctx.createConsumer(q);
23             Message msg=null;
24             while((msg=consumer.receive())!=null){
25                 if(msg instanceof TextMessage){
26                     TextMessage m=(TextMessage)msg;
27                     System.out.println(m.getText());
28                 }
29                 else
30                     break;
31             }

```

```

32     ctx.close();
33 }
34 catch(Exception e){
35     System.out.println("JMSEException : "+e.getMessage());
36 }
37 System.out.println("Consumer finished");
38 }
39 }

```

### Exemplul 5.3.3 *Trimiterea și recepția mesajelor se face prin aplicația*

```

1 class MsgHelloT{
2     public static void main(String [] args){
3         int n=3;
4         String queueName="MyQueue";
5         if(args.length>0)
6             queueName=args[0];
7         if(args.length>1)
8             n=Integer.parseInt(args[1]);
9         MsgSenderT sender=new MsgSenderT(queueName,n);
10        SyncMsgReceiverT receiver =new SyncMsgReceiverT(queueName);
11        receiver.start();
12        sender.start();
13    }
14 }

```

### 5.3.3 Recepția asincronă a unui mesaj

Recepția asincronă a mesajelor presupune implementarea interfeței `MessageListener` ce conține o singură metodă

```
public void onMessage(Message mesaj);
```

care fixează prelucrarea mesajului.

Metoda `setMessageListener(MessageListener obj)` a clasei `JMSConsumer` fixează obiectul ce implementează interfața `MessageListener`.

Astfel, caracterul asincron constă din faptul că mesajele sunt preluate de *ascultător* - adică obiectul ce implementează interfața `MessageListener` și nu de clientul JMS.

**Exemplul 5.3.4** *În ideea exemplelor anterioare, un consumator de tip asincron al mesajelor este dat în clasa următoare:*



```

1 import javax.jms.MessageConsumer;
2 import javax.jms.Queue;
3 import javax.jms.JMSContext;
4 import javax.jms.JMSConsumer;

6 public class AsyncMsgReceiverT extends Thread{
7     String queueName;

9     AsyncMsgReceiverT(String queueName){
10         this.queueName=queueName;
11     }

13     public void run(){
14         try{
15             com.sun.messaging.QueueConnectionFactory cf=
16                 new com.sun.messaging.QueueConnectionFactory();
17             //cf.setProperty("imqBrokerHostName","host");
18             //cf.setProperty("imqBrokerHostPort","7676");
19             Queue q=new com.sun.messaging.Queue(queueName);
20             JMSContext ctx=cf.createContext();
21             JMSConsumer consumer = ctx.createConsumer(q);

23             TextListener textListener=new TextListener();
24             consumer.setMessageListener(textListener);
25             textListener.run();
26             ctx.close();
27         }
28         catch(Exception e){
29             System.out.println(e.getMessage());
30         }
31         System.out.println("Consumer finished");
32     }
33 }

```

împreună cu *ascultătorul*

```

1 import javax.jms.MessageListener;
2 import javax.jms.JMSException;
3 import javax.jms.TextMessage;
4 import javax.jms.Message;

6 public class TextListener implements MessageListener{
7     boolean sfarsit=false;
8     public void onMessage(Message message){
9         if(message instanceof TextMessage){
10             TextMessage m=(TextMessage)message;
11             try{
12                 String s=m.getText();
13                 System.out.println(s);
14             }
15             catch(JMSException e){
16                 System.out.println(e.getMessage());
17             }
18         }
19         else
20             sfarsit=true;
21     }

23     public void run(){

```

```
24     while(!sfarsit){
25         try{
26             Thread.sleep(1);
27         }
28         catch(InterruptedException e){}
29     };
30 }
31 }
```

### 5.3.4 Publicarea mesajelor

Publicarea mesajelor corespunzătoare unui subiect se face asemănător cu transmiterea mesajelor în comunicația punctuală, dar folosind instanțe ale claselor dedicate acestui tip de comunicație.

#### Exemplul 5.3.5

```
1 import javax.jms.Topic;
2 import javax.jms.JMSContext;
3 import javax.jms.JMSProducer;

5 public class MsgPublisherT extends Thread{
6     int n;
7     String subiect;

9     MsgPublisherT(String subiect, int n){
10         super();
11         this.n=n;
12         this.subiect=subiect;
13     }

15     public void run(){
16         try{
17             com.sun.messaging.TopicConnectionFactory cf=
18                 new com.sun.messaging.TopicConnectionFactory();
19             //cf.setProperty("imqBrokerHostName","host");
20             //cf.setProperty("imqBrokerHostPort","7676");
21             Topic t=new com.sun.messaging.Topic(subiect);
22             JMSContext ctx=cf.createContext();
23             JMSProducer producer=ctx.createProducer();
24             for(int i=0;i<n;i++){
25                 producer.send(t,"Hello "+i);
26             }
27             producer.send(t,ctx.createMessage());
28             ctx.close();
29         }
30         catch(Exception e){
31             System.out.println("JMSEException : "+e.getMessage());
32         }
33         System.out.println("Publisher finished");
34     }
35 }
```

### 5.3.5 Abonare și recepția mesajelor

Dacă  $t$  este obiectul de tip `Topic`, clienții se abonează - subscriu - unui subiect prin

```
JMSConsumer consumer = ctx.createConsumer(t);
```

Subscrierea este valabilă atâta timp cât clientul este activ. Pentru a primi toate mesajele specifice subiectului, chiar și când clientul este inactiv, acesta trebuie să fie *durabil*, adică crearea consumatorului să se facă prin

```
ctx.setClientID(clientID);
JMSConsumer consumer = ctx.createDurableConsumer(t,clientName);
```

În ambele cazuri, clientul primește doar mesajele publicate din momentul subscrierii.

### Exemplul 5.3.6

```

1 import javax.jms.TextMessage;
2 import javax.jms.Message;
3 import javax.jms.Topic;
4 import javax.jms.JMSContext;
5 import javax.jms.JMSConsumer;

6
7 public class MsgSubscriberT extends Thread{
8     String subiect;
9     String clientID;
10    String clientName;

11
12    MsgSubscriberT(String subiect,String clientID , String clientName){
13        this.subiect=subiect;
14        this.clientID=clientID;
15        this.clientName=clientName;
16    }

17
18    public void run(){
19        try{
20            com.sun.messaging.TopicConnectionFactory cf=
21                new com.sun.messaging.TopicConnectionFactory();
22            //cf.setProperty("imqBrokerHostName","host");
23            //cf.setProperty("imqBrokerHostPort","7676");
24            Topic t=new com.sun.messaging.Topic(subiect);
25            JMSContext ctx=cf.createContext();
26            ctx.setClientID(clientID);
27            JMSConsumer consumer = ctx.createDurableConsumer(t,clientName);

28
29            Message msg=null;
30            while((msg=consumer.receive())!=null){
31                if(msg instanceof TextMessage){
32                    TextMessage m=(TextMessage)msg;
33                    System.out.println(clientName+" received : "+m.getText());
34                }
35            }
36        }
37    }
38 }

```

```

35         else
36             break;
37     }
38     ctx.close();
39 }
40 catch(Exception e){
41     System.out.println("JMSEException : "+e.getMessage());
42 }
43 System.out.println("Subscriber finished");
44 }
45 }

```

Apelarea celor două activități se face prin

### Exemplul 5.3.7

```

1 class MsgPS{
2     public static void main(String [] args){
3         String subiect="JMS";
4         int n=3,noAbonati=3;
5         if(args.length>0)
6             subiect=args[0];
7         if(args.length>1)
8             n=Integer.parseInt(args[1]);
9         MsgPublisherT publisher=new MsgPublisherT(subiect,n);
10        MsgSubscriberT [] abonati=new MsgSubscriberT[noAbonati];
11        publisher.start();
12        for(int i=0;i<noAbonati;i++){
13            abonati[i]=new MsgSubscriberT(subiect,"ID"+i,"id"+i);
14            abonati[i].start();
15        }
16    }
17 }

```

### 5.3.6 Cazul abonatului partajat

Mai mulți clienți JMS folosesc aceeași conexiune iar abonamentul se identifică printr-un nume (**String**). Dintre aceși clienți doar unul recepționează mesajele care sunt emise după momentul abonării.

### Exemplul 5.3.8

Față de exemplul anterior se vor expedia mai multe mesaje de tip **Message**.

```

1 import javax.jms.Topic;
2 import javax.jms.JMSContext;
3 import javax.jms.JMSProducer;

5 public class MsgPublisherT extends Thread{
6     int n;
7     String subiect;

```

```

9   MsgPublisherT(String subiect ,int n){
10       super();
11       this.n=n;
12       this.subiect=subiect;
13   }

15   public void run(){
16       try{
17           com.sun.messaging.TopicConnectionFactory cf=
18               new com.sun.messaging.TopicConnectionFactory();
19           //cf.setProperty("imqBrokerHostName","host");
20           //cf.setProperty("imqBrokerHostPort","7676");
21           Topic t=new com.sun.messaging.Topic(subiect);
22           JMSContext ctx=cf.createContext();
23           JMSProducer producer=ctx.createProducer();
24           for(int i=0;i<n;i++){
25               producer.send(t,"Hello "+i);
26           }
27           for(int i=0;i<n;i++)producer.send(t,ctx.createMessage());
28           ctx.close();
29       }
30       catch(Exception e){
31           System.out.println("JMSEException : "+e.getMessage());
32       }
33       System.out.println("Publisher finished");
34   }
35 }

```

Pentru a folosi aceeași conexiune recepția mesajelor este programată în două clase :

- Se instanțiază o fabrică de conexiuni și un număr de clienți care vor utiliza aceeași conexiune. Fiecare client este un fir de execuție care este gestionat de o cuvă (*pool*) de fire de execuție.

```

1   import javax.jms.TextMessage;
2   import javax.jms.Message;
3   import javax.jms.Topic;
4   import javax.jms.JMSContext;
5   import javax.jms.JMSConsumer;
6   import java.util.concurrent.ExecutorService;
7   import java.util.concurrent.Executors;

9   public class ReceiversT extends Thread{
10       String subiect;
11       String sharedSubscriptionName;
12       int noAbonati;
13       com.sun.messaging.TopicConnectionFactory cf;

15       ReceiversT(int noAbonati, String subiect ,
16           String sharedSubscriptionName){
17           this.subiect=subiect;
18           this.noAbonati=noAbonati;
19           this.sharedSubscriptionName=sharedSubscriptionName;
20       }

22       public void run(){
23           ExecutorService exec=Executors.newFixedThreadPool(noAbonati);

```

```

25     try{
26         cf=new com.sun.messaging.TopicConnectionFactory();
27         //cf.setProperty("imqBrokerHostName","host");
28         //cf.setProperty("imqBrokerHostPort","7676");

30         for (int i=0;i<noAbonati;i++){
31             String clientName="Client "+i;
32             MsgSubscriberT t=new MsgSubscriberT(cf,subiect ,
33                 sharedSubscriptionName,clientName);
34             exec.execute(t);
35         }
36         exec.shutdownNow();
37     }
38     catch(Exception e){
39         System.out.println("Receivers : "+e.getMessage());

41     }
42     System.out.println("Receivers finished");
43 }
44 }

```

### ● Clienții JMS propriu-ziși

```

1  import javax.jms.TextMessage;
2  import javax.jms.Message;
3  import javax.jms.Topic;
4  import javax.jms.JMSContext;
5  import javax.jms.JMSConsumer;

7  public class MsgSubscriberT extends Thread{
8      String subiect;
9      String clientID;
10     String clientName;
11     String sharedSubscriptionName;
12     com.sun.messaging.TopicConnectionFactory cf;

14     MsgSubscriberT(com.sun.messaging.TopicConnectionFactory cf,
15         String subiect,String sharedSubscriptionName,
16         String clientName){
17         this.cf=cf;
18         this.subiect=subiect;
19         this.sharedSubscriptionName=sharedSubscriptionName;
20         this.clientName=clientName;
21     }

23     public void run(){
24         try{
25             Topic t=new com.sun.messaging.Topic(subiect);
26             JMSContext ctx=cf.createContext();
27             JMSConsumer consumer = ctx.createSharedConsumer(t,
28                 sharedSubscriptionName);

30             Message msg=null;
31             while((msg=consumer.receive())!=null){
32                 if(msg instanceof TextMessage){
33                     TextMessage m=(TextMessage)msg;
34                     System.out.println(clientName+" received : "+m.getText());

```

```

35         }
36         else
37             break;
38     }
39     ctx.close();
40 }
41 catch (Exception e) {
42     System.out.println("MsgSubscriber : "+e.getMessage());
43 }
44 System.out.println("Subscriber "+clientName+" finished");
45 }
46 }

```

Aplicația este condusă de clasa

```

1 public class MsgPS{
2     public static void main(String [] args){
3         String subiect="JMS";
4         String sharedSubscriptionName="ourSubscription";
5         int n=3,noAbonati=3;
6         if(args.length>0)
7             subiect=args[0];
8         if(args.length>1)
9             n=Integer.parseInt(args[1]);
10        MsgPublisherT publisher=new MsgPublisherT(subiect,n);
11        ReceiversT abonati=new ReceiversT(noAbonati, subiect,
12            sharedSubscriptionName);
13        abonati.start();
14        publisher.start();
15    }
16 }

```

### 5.3.7 Obiecte administrator prin JNDI

Obiectele administrator, fabrica de conexiuni și obiectul destinație, se precizează într-un fișier de proprietăți - *jndi.properties*.

Pentru Open Message Queue acest fișier este  
Varianta *Oracle-Open Message Queue*

```

1 java.naming.factory.initial =
2   com.sun.jndi.fscontext.RefFSContextFactory
3 java.naming.provider.url=file:///d:/Temp
4
5 # use the following property to configure the default connector
6
7 # register some queues in JNDI using the form
8 # queue.[jndiName] = [physicalName]
9 queue.queue = myqueue
10
11 # register some topics in JNDI using the form
12 # topic.[jndiName] = [physicalName]
13 topic.topic = mytopic

```

În acest caz este nevoie de crearea în prealabil a obiectelor corespunzătoare conexiunii, a cozii (queue) și a subiectului (topic). Aceste obiecte se creează cu utilitarul *imqobjmgr* din distribuția *Oracle-Open Message Queue*.

Obiectele se pot crea cu fișierul de comenzi:

```

1 set PATH=d:\mq\bin;%PATH%
2 imqobjmgr add -t qf -l "ConnectionFactory"
3   -j java.naming.provider.url=file:///d:/Temp
4   -j java.naming.factory.initial=com.sun.jndi.fscontext.RefFSContextFactory
5 rem imqobjmgr add -t tf -l "ConnectionFactory"
6   -j java.naming.provider.url=file:///d:/Temp
7   -j java.naming.factory.initial=com.sun.jndi.fscontext.RefFSContextFactory
8 imqobjmgr add -t q -l "queue"
9   -j java.naming.provider.url=file:///d:/Temp
10  -j java.naming.factory.initial=com.sun.jndi.fscontext.RefFSContextFactory
11 rem imqobjmgr add -t t -l "topic"
12  -j java.naming.provider.url=file:///d:/Temp
13  -j java.naming.factory.initial=com.sun.jndi.fscontext.RefFSContextFactory

```

Obiectele se pot șterge cu fișierul de comenzi:

```

1 set PATH=d:\mq\bin;%PATH%
2 imqobjmgr delete -t qf -l "ConnectionFactory"
3   -j java.naming.provider.url=file:///d:/Temp
4   -j java.naming.factory.initial=com.sun.jndi.fscontext.RefFSContextFactory
5 imqobjmgr delete -t tf -l "ConnectionFactory"
6   -j java.naming.provider.url=file:///d:/Temp
7   -j java.naming.factory.initial=com.sun.jndi.fscontext.RefFSContextFactory
8 imqobjmgr delete -t q -l "queue"
9   -j java.naming.provider.url=file:///d:/Temp
10  -j java.naming.factory.initial=com.sun.jndi.fscontext.RefFSContextFactory
11 imqobjmgr delete -t t -l "topic"
12  -j java.naming.provider.url=file:///d:/Temp
13  -j java.naming.factory.initial=com.sun.jndi.fscontext.RefFSContextFactory

```

Reluăm aplicațiile anterioare privind transmiterea / recepția unui mesaj prin comunicație punctuală utilizând o coadă și publicarea și recepționarea unui mesaj prin comunicație bazată pe subiect într-o variantă independentă de suportul *middleware* de serviciu de mesagerie folosit. Regăsirea resurselor gestionate de serviciul de mesagerie se va face utilizând JNDI.

### 5.3.8 Comunicația prin coadă - queue

Aplicația este alcătuită din clasele *MsgHelloT*, *MsgSenderT*, *SyncMsgReceiverT*, *AsyncMsgReceiverT*, *TextListener*. Față de versiunea prezentată anterior se modifică doar clasele *MsgSenderT*, *SyncMsgReceiverT*, *AsyncMsgReceiverT*.

În clasa *MsgHelloT* câmpul *queueName* reprezintă numele JNDI al cozii. Clasa *MsgSenderT*

```

1 import javax.jms.QueueConnectionFactory;
2 import javax.jms.Queue;

```



```

3 import javax.jms.JMSContext;
4 import javax.jms.JMSProducer;
5 import javax.naming.InitialContext;
6 import javax.naming.NamingException;
7 import java.util.Properties;
8 import java.io.IOException;

10 public class MsgSenderT extends Thread{
11     final String CONNECTION_JNDI_NAME = "ConnectionFactory";
12     String QUEUE_JNDI_NAME = "queue";
13     private InitialContext ctx;
14     private int n;

16     MsgSenderT(String queueName, int n){
17         QUEUE_JNDI_NAME=queueName;
18         this.n=n;
19     }

21     public void run(){
22         try{
23             setupJNDI();
24             QueueConnectionFactory cf=
25                 (QueueConnectionFactory)ctx.lookup(CONNECTION_JNDI_NAME);
26             Queue q=(Queue)ctx.lookup(QUEUE_JNDI_NAME);
27             closeJNDI();

29             JMSContext jmsctx=cf.createContext();
30             JMSProducer producer=jmsctx.createProducer();
31             for(int i=0;i<n;i++){
32                 producer.send(q,"Hello "+i);
33             }
34             producer.send(q,jmsctx.createMessage());
35             jmsctx.close();
36         }
37         catch(Exception e){
38             System.out.println("JMSException : "+e.getMessage());
39         }
40         System.out.println("Sender finished");
41     }

43     private void setupJNDI(){
44         Properties properties = new Properties();
45         try{
46             properties.load(this.getClass().getResourceAsStream("jndi.properties"));
47         }
48         catch(IOException e){
49             System.out.println("JNDI-PropertiesError : "+e.getMessage());
50         }
51         try{
52             ctx = new InitialContext(properties);
53         }
54         catch (NamingException e){
55             System.err.println("Error Setting up JNDI Context:" + e);
56         }
57     }

59     private void closeJNDI(){
60         try{
61             ctx.close();

```

```

62     }
63     catch (NamingException e){
64         System.err.println("Unable to close JNDI Context : " + e);
65     }
66 }
67 }

```

### Clasa *SyncMsgReceiverT*

```

1  import javax.jms.QueueConnectionFactory;
2  import javax.jms.Queue;
3  import javax.jms.TextMessage;
4  import javax.jms.Message;
5  import javax.jms.JMSContext;
6  import javax.jms.JMSConsumer;
7  import javax.naming.InitialContext;
8  import javax.naming.NamingException;
9  import java.util.Properties;
10 import java.io.IOException;

12 public class SyncMsgReceiverT extends Thread{
13     final String CONNECTION_JNDI_NAME = "ConnectionFactory";
14     String QUEUE_JNDI_NAME = "queue";
15     private InitialContext ctx;

17     SyncMsgReceiverT(String queueName){
18         QUEUE_JNDI_NAME=queueName;
19     }

21     public void run(){
22         try{
23             setupJNDI();
24             QueueConnectionFactory cf=
25                 (QueueConnectionFactory)ctx.lookup(CONNECTION_JNDI_NAME);
26             Queue q=(Queue)ctx.lookup(QUEUE_JNDI_NAME);
27             closeJNDI();

29             JMSContext jmsctx=cf.createContext();
30             JMSConsumer consumer = jmsctx.createConsumer(q);
31             Message msg=null;
32             while((msg=consumer.receive())!=null){
33                 if(msg instanceof TextMessage){
34                     TextMessage m=(TextMessage)msg;
35                     System.out.println(m.getText());
36                 }
37                 else
38                     break;
39             }
40             jmsctx.close();
41         }
42         catch(Exception e){
43             System.out.println("JMSEException : "+e.getMessage());
44         }
45         System.out.println("Consumer finished");
46     }

48     private void setupJNDI(){. . .}

50     private void closeJNDI(){. . .}

```

51 | }

### 5.3.9 Comunicația pe bază de subiect - topic

Aplicația este alcătuită din clasele *MsgPS*, *MsgPublisherT*, *MsgSubscriberT*. Față de versiunea prezentată anterior se modifică doar clasele *MsgPublisherT* și *MsgSubscriberT*.

În clasa *MsgPS* câmpul *subiect* reprezintă numele JNDI atașat topic-ului.  
Clasa *MsgPublisherT*

```

1 import javax.jms.TopicConnectionFactory;
2 import javax.jms.Topic;
3 import javax.jms.JMSContext;
4 import javax.jms.JMSProducer;
5 import javax.naming.InitialContext;
6 import javax.naming.NamingException;
7 import java.util.Properties;
8 import java.io.IOException;

10 public class MsgPublisherT extends Thread{
11     final String CONNECTION_JNDI_NAME = "ConnectionFactory";
12     String TOPIC_JNDI_NAME = "topic";
13     private InitialContext ctx;
14     private int n;

16     MsgPublisherT(String subiect, int n){
17         this.n=n;
18         TOPIC_JNDI_NAME=subiect;
19     }

21     public void run(){
22         try{
23             setupJNDI();
24             TopicConnectionFactory cf=
25                 (TopicConnectionFactory)ctx.lookup(CONNECTION_JNDI_NAME);
26             Topic t=(Topic)ctx.lookup(TOPIC_JNDI_NAME);
27             closeJNDI();

29             JMSContext jmsctx=cf.createContext();
30             JMSProducer producer=jmsctx.createProducer();
31             for(int i=0;i<n;i++){
32                 producer.send(t,"Despre JMS"+ " "+i);
33             }
34             producer.send(t,jmsctx.createMessage());
35             jmsctx.close();
36         }
37         catch(Exception e){
38             System.out.println("JMSException : "+e.getMessage());
39         }
40         System.out.println("Publisher finished");
41     }

43     private void setupJNDI(){ . . . }

45     private void closeJNDI(){ . . . }

```

46 }

### Clasa *MsgSubscriberT*

```

1 import javax.jms.TopicConnectionFactory;
2 import javax.jms.Topic;
3 import javax.jms.TextMessage;
4 import javax.jms.Message;
5 import javax.jms.JMSContext;
6 import javax.jms.JMSConsumer;
7 import javax.naming.InitialContext;
8 import javax.naming.NamingException;
9 import java.util.Properties;
10 import java.io.IOException;

12 public class MsgSubscriberT extends Thread{
13     final String CONNECTION_JNDLNAME = "ConnectionFactory";
14     String TOPIC_JNDLNAME = "topic";
15     private InitialContext ctx;
16     private String clientID;
17     private String clientName;

19     MsgSubscriberT(String subiect, String clientID, String clientName){
20         TOPIC_JNDLNAME=subiect;
21         this.clientID=clientID;
22         this.clientName=clientName;
23     }

25     public void run(){
26         try{
27             setupJNDI();
28             TopicConnectionFactory cf=
29                 (TopicConnectionFactory)ctx.lookup(CONNECTION_JNDLNAME);
30             Topic t=(Topic)ctx.lookup(TOPIC_JNDLNAME);
31             JMSContext jmsctx=cf.createContext();
32             jmsctx.setClientID(clientID);
33             JMSConsumer consumer = jmsctx.createDurableConsumer(t, clientName);
34             closeJNDI();

36             Message msg=null;
37             while((msg=consumer.receive())!=null){
38                 if(msg instanceof TextMessage){
39                     TextMessage m=(TextMessage)msg;
40                     System.out.println(clientName+" received : "+m.getText());
41                 }
42                 else
43                     break;
44             }
45             jmsctx.close();
46         }
47         catch(Exception e){
48             System.out.println("JMSEException : "+e.getMessage());
49         }
50         System.out.println("Subscriber finished");
51     }

53     private void setupJNDI(){. . .}

55     private void closeJNDI(){. . .}

```

56 | }

### 5.3.10 Utilizarea mesajelor de tip StreamMessage

Fișiere grafice (jpg, png), audio (mp3), video pot fi procesate prin mesaje de tip `StreamMessage`

**Exemplul 5.3.9** *Aplicație JMS cu clasă pentru publicarea și consumarea unui mesaj care încorporează o imagine jpg.*

#### Clasa `MsgPublisherT`

```

1 import javax.jms.Topic;
2 import javax.jms.JMSContext;
3 import javax.jms.StreamMessage;
4 import javax.jms.BytesMessage;
5 import javax.jms.JMSProducer;
6 import java.io.File;
7 import java.io.ByteArrayOutputStream;
8 import java.awt.image.BufferedImage;
9 import javax.imageio.ImageIO;

11 public class MsgPublisherT extends Thread{
12     String subiect;

14     MsgPublisherT(String subiect){
15         this.subiect=subiect;
16     }

18     public void run(){
19         try{
20             com.sun.messaging.TopicConnectionFactory cf=
21                 new com.sun.messaging.TopicConnectionFactory();
22             //cf.setProperty("imqBrokerHostName","host");
23             //cf.setProperty("imqBrokerHostPort","7676");
24             Topic t=new com.sun.messaging.Topic(subiect);
25             JMSContext ctx=cf.createContext();

27             File file = new File("myimage.jpg");
28             BufferedImage img = ImageIO.read(file);
29             ByteArrayOutputStream baos=new ByteArrayOutputStream();
30             ImageIO.write(img,"jpg",baos);
31             baos.flush();
32             byte[] bytes=baos.toByteArray();
33             System.out.println(bytes.length);
34             baos.close();

36             StreamMessage msg=ctx.createStreamMessage();
37             msg.writeBytes(bytes);

39             JMSProducer producer=ctx.createProducer();
40             producer.send(t,msg);

42             ctx.close();

```

```

43     }
44     catch (Exception e){
45         System.out.println("JMSEException : "+e.getMessage());
46     }
47     System.out.println("Publisher finished");
48 }
49 }

```

### Clasa MsgSubscriberT

```

1  import javax.jms.Message;
2  import javax.jms.Topic;
3  import javax.jms.JMSContext;
4  import javax.jms.JMSConsumer;
5  import javax.jms.StreamMessage;
6  import javax.jms.BytesMessage;
7  import java.awt.image.BufferedImage;
8  import javax.imageio.ImageIO;
9  import java.io.ByteArrayInputStream;
10 import java.io.InputStream;

12 public class MsgSubscriberT extends Thread{
13     String subject;
14     String clientID;
15     String clientName;

17     MsgSubscriberT(String subject,String clientID , String clientName){
18         this.subject=subject;
19         this.clientID=clientID;
20         this.clientName=clientName;
21     }

23     public void run(){
24         try{
25             com.sun.messaging.TopicConnectionFactory cf=
26                 new com.sun.messaging.TopicConnectionFactory();
27             //cf.setProperty("imqBrokerHostName","host");
28             //cf.setProperty("imqBrokerHostPort","7676");
29             Topic t=new com.sun.messaging.Topic(subject);
30             JMSContext ctx=cf.createContext();
31             ctx.setClientID(clientID);
32             JMSConsumer consumer = ctx.createDurableConsumer(t,clientName);

34             Message msg=null;
35             while((msg=consumer.receive())!=null){
36                 if(msg instanceof StreamMessage){
37                     StreamMessage m=(StreamMessage)msg;
38                     m.reset();
39                     byte[] bytes=(byte[])m.readObject();
40                     System.out.println(clientName+" received : "+bytes.length);
41                     InputStream in = new ByteArrayInputStream(bytes);
42                     BufferedImage bi = ImageIO.read(in);
43                     ShowImage showImage=new ShowImage(bi);
44                     showImage.show();
45                     break;
46                 }
47             }
48             ctx.close();
49         }

```

```

50     catch(Exception e){
51         System.out.println("JMSEException : "+e.getMessage());
52     }
53     System.out.println("Subscriber finished");
54 }
55 }

```

Clasa *ShowImage* afișează imaginea pe monitor.

```

1  import java.awt.Canvas;
2  import java.awt.Image;
3  import java.awt.Graphics;
4  import java.awt.BorderLayout;
5  import javax.swing.JFrame;

7  class MyCanvas extends Canvas{
8      Image image=null;

10     MyCanvas(Image image){
11         this.image=image;
12     }

14     public void paint(Graphics g){
15         g.drawImage(image,0,0,this);
16     }
17 }

19 public class ShowImage{
20     MyCanvas mc=null;

22     ShowImage(Image image){
23         mc=new MyCanvas(image);
24     }

26     public void show(){
27         // Interfata swing
28         JFrame jframe = new JFrame("The received image");
29         jframe.addNotify();
30         jframe.getContentPane().setLayout(new BorderLayout());
31         jframe.getContentPane().add(mc, BorderLayout.CENTER);
32         jframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
33         jframe.setSize(600,600);
34         jframe.setVisible(true);
35     }
36 }

```

Producătorul și consumatorului se lansează din

```

1  class MsgPS{
2      public static void main(String[] args){
3          String subiect="streaming";
4          if(args.length>0)
5              subiect=args[0];
6          MsgSubscriberT abonat=new MsgSubscriberT(subiect,"myID","myName");
7          MsgPublisherT publisher=new MsgPublisherT(subiect);
8          abonat.start();

10         try{
11             Thread.sleep(1000);

```

```

12     }
13     catch (InterruptedException e){}

15     publisher.start();
16 }
17 }

```

### 5.3.11 Aplicație JMS slab cuplată

Aplicația va fi alcătuită din trei clienți:

- Client care preia și rezolvă cererile iar răspunsurile sunt trimise solicitantului într-un mesaj pe un subiect indicat de acesta. Acest client preia mesajele corespunzătoare unui subiect public.
- Client care lansează cererea. Acest client va crea un abonat durabil pe subiectul pe care se va prelua răspunsul. Subiectul face parte din cerere.
- Client care preia răspunsul.

**Exemplul 5.3.10** *Calculul celui mai mare divizor comun a două numere naturale.*

Clasa *MsgCmmdcServer*

```

1 import javax.jms.TextMessage;
2 import javax.jms.Message;
3 import javax.jms.Topic;
4 import javax.jms.JMSContext;
5 import javax.jms.JMSConsumer;
6 import javax.jms.JMSProducer;

8 public class MsgCmmdcServer{

10     public static void main(String [] args){
11         MsgCmmdcServer server=new MsgCmmdcServer();
12         server.service();
13     }

15     private void service(){
16         try{
17             com.sun.messaging.TopicConnectionFactory cf=
18                 new com.sun.messaging.TopicConnectionFactory();
19             //cf.setProperty("imqBrokerHostName","host");
20             //cf.setProperty("imqBrokerHostPort","7676");
21             Topic t=new com.sun.messaging.Topic("Cmmdc");
22             JMSContext ctx=cf.createContext();
23             JMSConsumer consumer = ctx.createSharedConsumer(t,"Cmmdc");
24             JMSProducer producer = ctx.createProducer();
25             while(true){
26                 Message msg=null;
27                 while((msg=consumer.receive())!=null){

```



```

28         if(msg instanceof TextMessage){
29             TextMessage tm=(TextMessage)msg;
30             String s=tm.getText();
31             String [] ss=s.split(" ");
32             long m=Long.parseLong(ss[0]);
33             long n=Long.parseLong(ss[1]);
34             String topic=ss[2];
35             long c=cmmdc(m,n);
36             Topic t1=new com.sun.messaging.Topic(topic);
37             producer.send(t1,(new Long(c)).toString());
38             System.out.println("Server sent "+c+" to "+topic);
39         }
40     }
41 }

43 }
44 catch(Exception e){
45     System.out.println("JMSEException : "+e.getMessage());
46 }
47 }

49 private long cmmdc(long m,long n){. . .}
50 }

```

### Clasa *MsgClientSender*

```

1 import javax.jms.Topic;
2 import javax.jms.JMSContext;
3 import javax.jms.JMSProducer;
4 import javax.jms.JMSConsumer;
5 import java.util.Scanner;

7 public class MsgClientSender{
8     private String msg,clientID,clientName,topicResult;

10     MsgClientSender(String clientID,String clientName){
11         this.clientID=clientID;
12         this.clientName=clientName;
13         Scanner scanner=new Scanner(System.in);
14         System.out.println("Introduceti m :");
15         long m=scanner.nextLong();
16         String sm=new Long(m).toString();
17         System.out.println("Introduceti n :");
18         long n=scanner.nextLong();
19         String sn=new Long(n).toString();
20         System.out.println("Introduceti 'Topic'-ul raspunsului");
21         topicResult=scanner.next();
22         msg=sm+" "+sn+" "+topicResult;
23     }

25     private void service(){
26         try{
27             com.sun.messaging.TopicConnectionFactory cf=
28                 new com.sun.messaging.TopicConnectionFactory();
29             //cf.setProperty("imqBrokerHostName","host");
30             //cf.setProperty("imqBrokerHostPort","7676");
31             Topic t=new com.sun.messaging.Topic("Cmmdc");
32             JMSContext ctx=cf.createContext();

```

```

34      Topic t1=new com.sun.messaging.Topic(topicResult);
35      ctx.setClientID(clientID);
36      JMSConsumer consumer = ctx.createDurableConsumer(t1, clientName);
37      JMSProducer producer=ctx.createProducer();
38      producer.send(t,msg);
39      ctx.close();
40  }
41  catch(Exception e){
42      System.out.println("JMSEException : "+e.getMessage());
43  }
44  }

46  public static void main(String[] args){
47      if(args.length<2){
48          System.out.println("Usage:");
49          System.out.println("java MsgClientSender clientID clientName");
50          System.exit(0);
51      }
52      MsgClientSender client=new MsgClientSender(args[0], args[1]);
53      client.service();
54  }
55  }

```

### Clasa *MsgClientReceiver*

```

1  import javax.jms.TextMessage;
2  import javax.jms.Message;
3  import javax.jms.Topic;
4  import javax.jms.JMSContext;
5  import javax.jms.JMSConsumer;
6  import java.util.Scanner;

8  public class MsgClientReceiver{
9      private String topicResult,clientID,clientName;

11     MsgClientReceiver(String clientID,String clientName){
12         this.clientID=clientID;
13         this.clientName=clientName;
14         Scanner scanner=new Scanner(System.in);
15         System.out.println("Introduceti 'Topic'-ul raspunsului");
16         topicResult=scanner.next();
17     }

19     public void service(){
20         try{
21             com.sun.messaging.TopicConnectionFactory cf=
22                 new com.sun.messaging.TopicConnectionFactory();
23             //cf.setProperty("imqBrokerHostName","host");
24             //cf.setProperty("imqBrokerHostPort","7676");
25             Topic t=new com.sun.messaging.Topic(topicResult);
26             JMSContext ctx=cf.createContext();
27             ctx.setClientID(clientID);
28             JMSConsumer consumer = ctx.createDurableConsumer(t,clientName);
29             Message msg=null;
30             while((msg=consumer.receive())!=null){
31                 if(msg instanceof TextMessage){
32                     TextMessage m=(TextMessage)msg;
33                     System.out.println("Cmmdc : "+m.getText());
34                     break;

```

```

35     }
36     }
37     ctx.close();
38     }
39     catch(Exception e){
40         System.out.println("JMSEException : "+e.getMessage());
41     }
42 }

44 public static void main(String[] args){
45     if(args.length<2){
46         System.out.println(" Usage:");
47         System.out.println(" java MsgSOAPClientReceiver clientID clientName");
48         System.exit(0);
49     }
50     MsgClientReceiver client=new MsgClientReceiver(args[0],args[1]);
51     client.service();
52 }
53 }

```

### 5.3.12 Programare JMS prin *glassfish*

Diferența față de versiunile amintite mai sus constau în

- obiectele administrator,
  - fabrica de conexiuni,
  - obiectul / sursa destinație

care se crează de administratorul *glassfish*.

1. Se lansează serverul de aplicații *glassfish*

```

set GLASSFISH_HOME=.. \glassfish*
set PATH=%GLASSFISH_HOME%\bin;%PATH%
asadmin start-domain domain1

```

Lansarea serverului de aplicație implică pornirea serverului JMS.

2. Dintr-un navigator se apelează administratorul <http://localhost:4848>.
3. – *Resources* → *JMS Resources* → *Destination Resources*

Se crează câte un obiect destinație cu datele:

JNDI Name	<i>myQueue</i>	<i>myTopic</i>
Physical Name	<i>myQueue</i>	<i>myTopic</i>
Resource Type	<i>javax.jms.Queue</i>	<i>javax.jms.Topic</i>

- *Resources* → *JMS Resources* → *Connection Factories*

Se crează câte o fabrică de conexiuni cu datele:

JNDI Name	<i>myQueueConnectionFactory</i> , respectiv <i>myTopicConnectionFactory</i>
Resource Type	<i>javax.jms.QueueConnectionFactory</i> , respectiv <i>javax.jms.TopicConnectionFactory</i>

- Aceste obiecte se injectează într-o clasă utilizând adnotarea `javax.annotation.Resource`

```
import javax.annotation.Resource;
. . .
public class . . .
    @Resource(lookup="myQueueConnectionFactory")
    private static QueueConnectionFactory cf;
    @Resource(lookup="myQueue")
    private static Queue q;
```

- O aplicație se arhivează cu jar cu indicarea clasei cu metoda main și se lansează prin intermediul procedurii acoperitoare glassfish\bin\appclient.bat

```
appclient -client numeArhiva.jar [-targetserver host:3700]
```

### Exemplul 5.3.11 *Aplicație cu comunicație punctuală.*

Clasa expeditorului

```
1 import javax.jms.QueueConnectionFactory;
2 import javax.jms.Queue;
3 import javax.jms.JMSContext;
4 import javax.jms.JMSProducer;
5 import javax.annotation.Resource;
6
7 public class MsgSender{
8     @Resource(lookup="myQueueConnectionFactory")
9     private static QueueConnectionFactory cf;
10    @Resource(lookup="myQueue")
11    private static Queue q;
12    private static int n=3;
13
14    public static void main(String[] args){
15        try{
16            JMSContext jmsctx=cf.createContext();
17            JMSProducer producer=jmsctx.createProducer();
18            for(int i=0;i<n;i++){
19                producer.send(q,"Hello "+i);
20            }
21            producer.send(q,jmsctx.createMessage());
22            jmsctx.close();
23        }
24        catch(Exception e){
25            //System.out.println("JMSException : "+e.getMessage());
26            e.printStackTrace();
27        }
28        System.out.println("Sender finished");
29    }
30 }
```

Clasa unui client sincron

```
1 import javax.jms.QueueConnectionFactory;
2 import javax.jms.Queue;
3 import javax.jms.JMSContext;
4 import javax.jms.JMSConsumer;
5 import javax.jms.Message;
6 import javax.jms.TextMessage;
7 import javax.annotation.Resource;

9 public class SyncMsgReceiver{
10     @Resource(lookup="myQueueConnectionFactory")
11     private static QueueConnectionFactory cf;
12     @Resource(lookup="myQueue")
13     private static Queue q;

15     public static void main(String [] args){
16         try{
17             JMSContext jmsctx=cf.createContext();
18             JMSConsumer consumer = jmsctx.createConsumer(q);
19             Message msg=null;
20             while((msg=consumer.receive())!=null){
21                 if(msg instanceof TextMessage){
22                     TextMessage m=(TextMessage)msg;
23                     System.out.println(m.getText());
24                 }
25                 else
26                     break;
27             }
28             jmsctx.close();
29         }
30         catch(Exception e){
31             //System.out.println("JMSEException : "+e.getMessage());
32             e.printStackTrace();
33         }
34         System.out.println("Consumer finished");
35     }
36 }
```

## Întrebări recapitulative

1. Explicați caracterul de cuplare slabă al unei aplicații bazat pe JMS.
2. Care este rolul furnizorului (provider) JMS?
3. Precizați și explicați modelele de comunicații cu mesaje în JMS.
4. Precizați modelele de programare al unui client JMS prezentate în curs.



**Partea II**

**TEHNOLOGII CU  
COMUNICAȚII PRIN  
INTERNET**





# Capitolul 6

## HyperText Transfer Protocol

Protocolul `http` - HyperText Transfer Protocol (`http`) este destinat schimburilor de informații în Internet.

Protocolul `http` a ajuns la versiunea 2 (`HTTP/2`), versiunea anterioară fiind 1.1 (`HTTP/1.1`). Scopul lui `HTTP/2` este reducerea latenței prin utilizarea mai eficientă a rețelei. Diferența dintre `HTTP/2` și `HTTP/1.1` este la nivelul de transport, adică a modului în care biții sunt schimbați prin rețea. `HTTP/2` păstrează structura mesajului din `HTTP/1.1`.

În vederea schimburilor dintre calculatoare prin Internet, reprezentarea datelor se face utilizând în principal:

- *eXtensible Markup Language* (XML) - subset al limbajului *Standard Generalized Markup Language* (SGML);
- *JavaScript Object Notation* (JSON).

*HyperText Markup Language* (HTML), XHTML, HTML 5 sunt principalele limbaje pentru publicarea informațiilor pe Web. Aceste limbaje derivă de asemenea din SGML.

### 6.1 Transacție `http`

Protocolul `http` este de tip client-server:

- Navigatoarele *Google Chrome*, *Mozilla Firefox*, *Microsoft Internet Explorer*, *Opera*, *Apple Safari* sunt aplicații client uzuale.
- Informațiile sunt găzduite / generate de servere Web de către site-uri și servicii Web. Exemple de servere Web sunt `apache HTTP Server`,

**Microsoft Internet Interchange Server.** O categorie aparte - esențială pentru platforma Java - este dată de servere Web container de servlet și Java Server Pages (JSP).

Transmiterea unui mesaj (cerere) conform protocolului http 1.1 constă din

1. Căutare DNS: Clientul încearcă să determine adresa IP a serverului Web:
  - (a) Se lansează o cerere DNS către serverul DNS al furnizorului de servicii Internet;
  - (b) Serverul DNS răspunde cu adresa IP a serverului Web.
2. Stabilirea unei conexiuni către serverul Web;
3. Expedierea mesajului cerere;
4. Se așteaptă răspunsul la cererea emisă;
5. Se recepționează și se încarcă mesajul de răspuns;
6. Închiderea conexiunii.

Acest ciclu de acțiuni se numește tranzacție http. Pași 3-4-5 se pot repeta. Un mesaj de tip cerere diferă de un mesaj de tip răspuns prin structura unor elemente constitutive.

Serverul nu reține informații între două tranzacții http. Acest comportament se exprimă prin terminologia: protocolul http este *fără stare* - *stateless*.

Transmisia datelor se realizează utilizând de obicei protocolul TCP.

Referințele resurselor se indică folosind URI / URL.

Cererile și răspunsurile sunt reprezentate ca linii de text separate de caracterul <CR><LF>, având structura

1. preambul - format dintr-o linie;
2. antete (*header*) - 0 sau mai multe atribute (nume:valoare);
3. o linie goală;
4. corpul mesajului - opțional.

**Preambulul unei cereri** conține:

1. numele metodei {GET, POST, PUT, DELETE, HEAD, CONNECT, TRACE, OPTIONS};

2. referința resursei (URL);
3. Versiunea protocolului http.

**Preambulul unui răspuns** conține:

1. Versiunea protocolului http;
2. codul răspunsului: număr natural format din trei cifre cu semnificațiile

Categoria	Indică
1**	mesaj de informare
2**	mesaj de succes
3**	redirectare către alt URL
4**	eroare în mesajul clientului
5**	eroare din partea serverului

Categoria este dată de prima cifră.

3. String explicitând codul răspunsului.

Un **antet** (*header*) este un atribut, adică o pereche (nume:valoare). Protocolul http definește o paletă largă de attribute. Exemple de antete sunt date în tabelul următor:

Antet	Semnificație
	Exemplu
host	Gazda și portul serverului Web
	localhost:8080
user-agent	Navigatorul care a lansat cererea
	Mozilla/5.0 (Windows NT 6.1; WOW64)
	AppleWebKit/535.2 (KHTML, like Gecko) Chrome/15.0.874.106 Safari/535.2
referer	URL-ul cererii
	http://localhost:8080/apphello/
accept-encoding	Tipuri de arhive acceptate
	gzip,deflate,sdch
accept-charset	Tipuri de codificare acceptate
	ISO-8859-1,utf-8;q=0.7,*;q=0.3
accept-language	Cea mai potrivită limbă pentru înțelegerea conținutului
	en-US,en;q=0.8
content-type	Tipul MIME al corpului mesajului
	http://application/x-www-form-urlencoded
content-length	Lungimea corpului mesajului
	22
Upgrade	Solicitare / Acord schimbare protocol
	websocket

**Corpul** mesajului este reprezentat printr-un text. Dacă conținutul este imagine, cod binar, etc., atunci acesta este codificat în text.

Codul *Multipurpose Internet Mail Exchange* (MIME) precizează natura conținutului unei resurse:

text/plain
text/html
text/xml
image/png
image/jpg
application/octet-stream
application/x-www-form-urlencoded

Ne propunem să punem în evidență mesajele http.

## Mesaj cerere http

Dintr-un navigator se va lansa prin intermediul unui document `html` o cerere către un ipotetic servlet dintr-un server Web. Cererea este interceptată de o clasă Java cu un soclu TCP, `ServerSocket`, care preia mesajele pe portul serverului Web.

Codul clasei Java este

```

1 import java.net.ServerSocket;
2 import java.net.Socket;
3 import java.io.InputStream;
4 import java.io.InputStreamReader;
5 import java.io.BufferedReader;
6 import java.io.IOException;
7 import java.io.BufferedWriter;
8 import java.io.FileWriter;

10 public class RequestHTTPMsg{
11     public static void main(String [] args){
12         Socket socket=null;
13         try{
14             ServerSocket serverSocket=new ServerSocket(8080);
15             socket=serverSocket.accept();
16         }
17         catch(Exception e){
18             System.out.println(e.getMessage());
19         }
20         try(
21             InputStream is=socket.getInputStream();
22             InputStreamReader isr=new InputStreamReader(is);
23             BufferedReader br=new BufferedReader(isr);
24             FileWriter fw=new FileWriter("output.txt",true);
25             BufferedWriter bw=new BufferedWriter(fw)){
26             for(;;){
27                 String s=br.readLine();
28                 if(s==null)break;
29                 System.out.println(s);
30                 bw.write(s);
31                 bw.newLine();
32             }
33         }
34         catch(IOException e){
35             System.out.println("Input Exception : "+e.getMessage());
36         }
37     }
38 }

```

Rezultatul cererii depinde de metoda GET sau POST utilizată în formulărul `html` folosit.

Pentru metoda GET cererea http este

```

GET /apphello/hello?name=mk&tip=text%2Fhtml HTTP/1.1
Host: localhost:8080
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64)
AppleWebKit/535.2 (KHTML, like Gecko)
Chrome/15.0.874.106 Safari/535.2

```

```

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3

```

iar pentru metoda POST cererea http este

```

POST /apphello/hello HTTP/1.1
Host: localhost:8080
Connection: keep-alive
Content-Length: 23
Cache-Control: max-age=0
Origin: null
User-Agent: Mozilla/5.0 (Windows NT 6.1;
    WOW64) AppleWebKit/535.2 (KHTML, like Gecko)
    Chrome/15.0.874.106 Safari/535.2
Content-Type: application/x-www-form-urlencoded
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3

name=mk&tip=text%2Fhtml

```

Formularul html utilizat are codul

```

1 <html>
2   <head><title> Servlet-ul Hello</title></head>
3   <body>
4     <center>
5       <h1> Pagina de apelare a servletului HelloServlet </h1>
6       <form method="post"
7         action="http://localhost:8080/apphello/hello">
8         <p>Introduceti numele:
9         <input type="text" name="name" size=20>
10        <p>
11        <input type="submit" value="Apeleaza">
12        <input type="hidden" name="tip" value="text/html" >
13      </form>
14    </center>
15  </body>
16 </html>

```

## Mesaj răspuns http

Un servlet (*apphello*) este activ într-un server Web. O clasă Java lansează o cerere http către acel servlet - chiar unul din mesajele obținute mai sus - după care recepționează răspunsul dat de servlet.

Codul clasei Java

```

1 import java.net.Socket;
2 import java.io.InputStreamReader;
3 import java.io.BufferedReader;
4 import java.io.IOException;
5 import java.io.BufferedWriter;

```

```

6 import java.io.FileWriter;
7 import java.io.PrintWriter;
8 import java.util.Scanner;

10 public class ResponseHTTPMsg{
11     public static void main(String [] args){
12         String reqGET="GET /apphello/hello?name=mk&tip=text%2Fhtml HTTP/1.1\r\n"+
13         "Host: localhost:8080\r\n"+
14         "Connection: keep-alive\r\n"+
15         "User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64)" +
16         "AppleWebKit/535.2 (KHTML, like Gecko)" +
17         "Chrome/15.0.874.106 Safari/535.2\r\n"+
18         "Accept: text/html,application/xhtml+xml,application/xml;" +
19         "q=0.9,*/*;q=0.8\r\n"+
20         "Accept-Encoding: gzip, deflate, sdch\r\n"+
21         "Accept-Language: en-US,en;q=0.8\r\n"+
22         "Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3\r\n";

24         String reqPOST="POST /apphello/hello HTTP/1.1\r\n"+
25         "Host: localhost:8080\r\n"+
26         "Connection: keep-alive\r\n"+
27         "Content-Length: 23\r\n"+
28         "Cache-Control: max-age=0\r\n"+
29         "Origin: null\r\n"+
30         "User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64)" +
31         "AppleWebKit/535.2 (KHTML, like Gecko)" +
32         "Chrome/15.0.874.106 Safari/535.2\r\n"+
33         "Content-Type: application/x-www-form-urlencoded\r\n"+
34         "Accept: text/html,application/xhtml+xml,application/xml;" +
35         "q=0.9,*/*;q=0.8\r\n"+
36         "Accept-Encoding: gzip, deflate, sdch\r\n"+
37         "Accept-Language: en-US,en;q=0.8\r\n"+
38         "Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3\r\n"+
39         "\r\n"+
40         "name=mk&tip=text%2Fhtml";

42         Scanner scanner=new Scanner(System.in);
43         System.out.println("Precizati metoda HTTP :");
44         System.out.println("1 : GET; 2: POST");
45         int metoda;
46         do{
47             metoda=scanner.nextInt();
48         }
49         while((metoda!=1)&&(metoda!=2));
50         try(
51             Socket socket=new Socket("localhost",8080);
52             InputStreamReader isr=new InputStreamReader(socket.getInputStream());
53             BufferedReader br=new BufferedReader(isr);
54             FileWriter fw=new FileWriter("output.txt",true);
55             BufferedWriter bw=new BufferedWriter(fw);
56             PrintWriter pw=new PrintWriter(socket.getOutputStream(),true)
57         ){
58             switch(metoda){
59                 case 1:
60                     //Lansarea unei cereri GET
61                     pw.println(reqGET);
62                     break;
63                 case 2:
64                     //Lansarea unei cereri POST

```

```

65         pw.println(reqPOST);
66     }
67     for(;;){
68         String s=br.readLine();
69         if(s==null)break;
70         System.out.println(s);
71         bw.write(s);
72         bw.newLine();
73         bw.flush();
74     }
75 }
76 catch(Exception e){
77     System.out.println("Exception : "+e.getMessage());
78 }
79 }
80 }

```

Indiferent de metoda cererii, mesajul http de răspuns este

```

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/html
Content-Length: 119
Date: Sun, 30 Oct 2011 16:35:47 GMT

<html>
<head><title>HelloServlet</title></head>
<body>
<h1>HelloServlet</h1>
<p>
Hi mk !
</p>
</body>
</html>

```

## 6.2 Server Web - container de servlet

În prezent sunt disponibile mai multe servere Web în care poate fi instalat un servlet și un fișier JSP *Java Server Pages*. Despre un asemenea server Web se spune că este container de servlet și JSP. Dintre produsele gratuite amintim

- *apache-tomcat*
- *jetty*
- *glassfish*

Acest server Web este utilizat în *java\_ee\_sdk* - o implementare JEE (Java Enterprise Edition) de la Oracle.

- *weblogic* (Oracle)



- *jboss application server* (RedHat)

*Geronimo*, o alta implementare JEE dezvoltată de fundația *apache*, poate utiliza containerul *apache-tomcat* sau *jetty*.

Serverul *Apache HTTP Server* și *Windows Internet Information Server* nu sunt servere Web continere de servlet și JSP.

## 6.3 Serverul Web *apache-tomcat*

Serverul Web *apache-tomcat*, (pe scurt *tomcat*) este distribuit gratuit și poate fi descărcat pornind de la adresa [www.apache.org](http://www.apache.org).

**Instalarea serverului în mediul Windows** revine la dezarhivarea fișierului descărcat *apache-tomcat-\*\*\**, într-un catalog *TOMCAT\_HOME*. Utilizarea serverului necesită fixarea a doi parametri sistem:

- *CATALINA\_HOME*= calea la catalogul în care s-a instalat produsul - *TOMCAT\_HOME*;
- *JAVA\_HOME*=calea la distribuția Java utilizată.

Pachetul conține cataloagele<sup>1</sup>: *bin*, *common*, *conf*, *logs*, *server*, *shared*, *temp*, *webapps*, *work*.

Serverul se lansează prin comanda

```
TOMCAT_HOME\bin\startup
```

și se oprește cu comanda

```
TOMCAT_HOME\bin\shutdown
```

Din catalogul *TOMCAT\_HOME*, lansarea se poate obține cu ajutorul fișierului de comenzi (\*.bat)

```
set CATALINA_HOME=. . .
set JAVA_HOME=. . .
bin\startup
```

Opțional se poate instala / activa *managerul* / *administratorul* serverului Web *tomcat*.

Verificarea funcționării serverului Web *tomcat* se face apelând dintr-un navigator pagina *http://host:port* unde

---

<sup>1</sup>Numele și numărul cataloagelor este dependent de distribuția *apache-tomcat*.

- *host* este numele calculatorului pe care rulează *tomcat*;
- Portul implicit este 8080.

Reușita este ilustrată de imaginea motanului



Toate aplicațiile se depun în catalogul `TOMCAT_HOME\webapps`.

Asemănător se procedează și în cazul serverului Web *jetty*. Nu este nevoie de nici o configurare suplimentară, lansarea făcându-se prin

```
cd . . . \jetty-distribution-*
java -jar start.jar
```

Din nou aplicațiile se depun în catalogul `\jetty-distribution-*\webapps`.

### ***Secure Socket Layer în tomcat***

Transmisia utilizând *Secure Socket Layer* - SSL (mai nou *Transport Layer Security* - TLS) înseamnă criptarea datelor care circulă între client și server.

Realizarea presupune

1. Generarea unui certificat de securitate cu utilitarul `keytool` din distribuția Java, de exemplu

```
keytool -genkey -alias tomcat -keyalg RSA
        -keystore {cale}/tomcatKeystore
        -dname "cn=SE, ou=cs, o=unitbv, l=brasov, c=RO"
        -keypass 1q2w3e -storepass 1q2w3e
```

Parametrul `keystore` fixează locația și denumirea fișierului certificatului de securitate.

Se definesc două parole

- `keypass` parola certificatului de securitate;
- `storepass` parola de protecție a locației certificatului de securitate.

Parametrul *cale* desemnează calea către catalogul unde în care se crează fișierul *tomcatKeystore* - certificatul de securitate. Uzual, certificatul de securitate se mută în catalogul `TOMCAT_HOME\conf`.

## 2. Modificarea fișierului `apache-tomcat-*\conf\server.xml`

- Se decommentează elementul

```
<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
    maxThreads="150" scheme="https" secure="true"
    clientAuth="false" sslProtocol="TLS" />
```

- Acest element `Connector` se completează cu atributele

```
keystoreFile="conf\tomcatKeystore"
keystorePass="1q2w3e"
```

## 3. Serverul Web se apelează prin

`https://localhost:8443`

# 6.4 Glassfish

**Instalarea și lansarea serverului.** Instalarea constă din dezarhivarea arhivei descărcate.

Este recomandat includerea în fișierul `GLASSFISH_HOME\glassfish\config\asenv.bat` a liniei

```
set AS_JAVA=c:\Progra~1\Java\jdk1.*
```

Utilizarea paginilor JSP necesită această setare.

Prin intermediul paginii de administrare se poate schimba parola administratorului (*admin*).

Containerele în care se depun aplicațiile se află într-un *domeniu*. Implicit, în catalogul `glassfish\domain` se crează domeniul cu numele `domain1`:

`GLASSFISH_HOME\glassfish\domains\domain1`.

Lansarea serverului se poate face prin:

- Din `GLASSFISH_HOME\glassfish\bin` se comandă

```
asadmin start-domain domain1
```

Oprirea serverului se poate face prin:

```
asadmin stop-domain domain1
```

Administrarea serverului se face prin

- pagina Web `http://localhost:4848`

- utilitarul `GLASSFISH_HOME\bin\asadmin.bat`

Administratorul serverului are posibilitatea să creeze domenii noi: Astfel crearea unui domeniu nou, având numele *numeDomeniu*, situat în catalogul *dirDomeniu* se obține prin

```
asadmin create-domain --adminport 4848 --domaindir dirDomeniu numeDomeniu
```

Ștergerea domeniului se obține prin

```
asadmin delete-domain --domaindir dirDomeniu numeDomeniu
```

Lansarea și oprirea serverului *glassfish* se comandă prin

```
asadmin start-domain --domaindir dirDomeniu numeDomeniu
```

```
asadmin stop-domain --domaindir dirDomeniu numeDomeniu
```

# Capitolul 7

## Conexiune simplă prin clase din `java.net`

Pachetul `java.net` oferă, prin intermediul claselor `URL`, `URLConnection`, `HttpURLConnection`, `HttpsURLConnection`, `JarURLConnection` o posibilitate elementară de acces a unor resurse din Internet.

### 7.1 Clasa `java.net.URL`

Clasa `java.net.URL` permite realizarea unei conexiuni cu un server Web<sup>1</sup> potrivit protocolurilor *http*, *https*, *ftp*, *file*, *jar*.

#### Constructori

- `URL(String spec)` throws `MalformedURLException`

Crează un obiect `URL` legat de resursa specificată de *spec*, care trebuie să fie o referință `URL` validă.

#### Metode

- `InputStream openStream()` throws `IOException`

Returnează fluxul de intrare pentru citirea resursei.

- `URLConnection openConnection()` throws `IOException`

Returnează o instanță a conexiunii cu obiectul definit de `URL`.

---

<sup>1</sup>apache-tomcat, apache, Microsoft IIS (Internet Information Services), etc.

**Exemplul 7.1.1** *Pe baza referinței către un fișier html dintr-un catalog vizibil al unui server Web, să se afișeze conținutul fișierului.*

În codul reprodus mai jos, fișierul *Hello.html* se află pe serverul *apache-tomcat*, în catalogul *webapps\url*.

```

1 import java.net.URL;
2 import java.io.InputStream;
3 import java.io.InputStreamReader;
4 import java.io.BufferedReader;

6 public class ReadHTTP{
7     public static void main(String [] args){
8         String adr="http://localhost:8080/url/Hello.html";
9         //System.setProperty("http.proxyHost","10.3.5.133");
10        //System.setProperty("http.proxyPort","3128");
11        URL url=null;
12        try{
13            url=new URL(adr);
14        }
15        catch(Exception e){
16            System.out.println(e.getMessage());
17        }
18        try(
19            InputStream in=url.openStream();
20            InputStreamReader isr=new InputStreamReader(in);
21            BufferedReader br=new BufferedReader(isr);
22        ){
23            String s;
24            do{
25                s=br.readLine();
26                if(s!=null)
27                    System.out.println(s);
28            }
29            while(s!=null);
30        }
31        catch(Exception e){
32            System.out.println(e.getMessage());
33        }
34    }
35 }

```

În cazul în care resursa - în exemplul dat reprezentat de fișierul *Hello.html* - este accesibil doar în urma autentificării *basic* sau *digest* (Cap. Servlet, Autentificare, 8.5.6) atunci codul de mai sus trebuie completat cu

```

final String username = "digest";
final String password = "digest";
Authenticator.setDefault(new Authenticator(){
    @Override
    protected PasswordAuthentication getPasswordAuthentication(){
        return new PasswordAuthentication(username,password.toCharArray());
    }
});

```

Un exemplu privind utilizarea clasei *java.net.HttpURLConnection* este dat în capitolul Servlet.

# Capitolul 8

## Servlet

Printre aplicațiile distribuite de tip client-server, în care comunicațiile se bazează pe protocolul *http*, se disting:

- *Aplicații Web (site)*: Cererea adresată serverului este lansată uzual de o persoană utilizând un program navigator: *Google Chrome*, *Mozilla Firefox*, *Microsoft Internet Explorer*, *Opera*, *Apple Safari*, etc.
- *Servicii Web*: Cererea către server se face de un program. Aplicația server și client se programează utilizând interfețe de programare specifice.

Un servlet este

- un program Java care se execută pe un server Web, compatibil;
- gestionat de serverul Web;
- capabil să recepționeze și să răspundă cererilor formulate de clienți.

De asemenea vom utiliza termenul servlet și pentru aplicația Web corespunzătoare.

Pe platforma de programare Java, servlet-ul este componenta care stă la baza majorității cadrelor de dezvoltare (*framework*) pentru aplicații și servicii Web.<sup>1</sup>

Programarea și utilizarea unui servlet necesită:

- Cunoașterea marcatului **html** `<form>` pentru realizarea formularelor de introducere a datelor;
- Utilizarea unui server Web, container de servleți.

---

<sup>1</sup>*vert.x*, *Play* sunt cadre de lucru care dezvoltă în Java aplicații distribuite fără să se bazeze pe servlet.

## 8.1 Marcajul <form>

Într-un document html introducerea datelor se poate obține utilizând marcajul <form> ...</form> .

**Atribute ale marcajului <form>** . Reamintim că atributele se prezintă ca perechi (nume, valoare) și se scriu în antetul marcajului sub forma `nume = valoare`.

Nume	Valoare	Semnificația valorii
<b>action</b>	adresa tip URL	Resursa care prelucrează formularul, cel puțin <i>/context/numeApel</i>
<b>method</b>	GET	Mesajul trimis serverului Web conține după adresa URL numele și valorile parametrilor introduși. Adăugarea se face potrivit sintaxei <code>?numeParam1=valoare&amp;numeParam2=valoare...</code> . Lungimea mesajului nu poate depăși 255 caractere.
	POST	Transmisia datelor se face în fluxuri de date. Permite transferul unor fișiere de pe mașina clientului pe mașina serverului.
<b>id</b>		Parametru de identificare a formularului (opțional).
<b>name</b>		Nume atribuit formularului (opțional).
<b>onSubmit</b>		Metodă JavaScript executată înaintea apelării serverului Web (opțional).

În **conținutul marcajului <form>** putem include elemente de control prin marcatele

- <input>
- <option>
- <select>
- <textarea>

Atributele marcajului < input> sunt:



Nume	Valoare	Semnificație
<b>type</b>	<b>text</b> <b>number</b> <b>date</b> <b>password</b> <b>submit</b> <b>reset</b> <b>file</b> <b>hidden</b>	Se așteaptă introducerea unui text Se așteaptă introducerea unui număr întreg Se așteaptă introducerea unei date calendaristice Se așteaptă introducerea unei parole Se marchează sfârșitul completării formularului Se reinițializează formularul Permite selectarea unui fișier Transmite mai departe un atribut fără vizualizarea lui
<b>name</b>		numele controlului
<b>value</b>		valoarea (inițială) a controlului
<b>size</b>		numărul caracterelor atașat controlului

În cazul marcajului `< select >` utilizarea este

```
<select name="nume">
  <option value="valoare"> Valoare
  . . . . .
</select>
```

Valoarea atributului *nume* este dată de valoarea selectată.

## 8.2 Realizarea unui servlet

Interfața de programare (API) pentru servlet nu face parte din JDK, fiind implementat de fiecare producător de server Web container de servlet.

Structura minimală a unui servlet este

```
catalogAppServlet
|--> WEB-INF
|   |--> classes
|   |   |   *.class
|   |--> lib
|   |   |   *.jar
|   index.html
```

Catalogul **classes** conține fișierele *class* ale aplicației servlet.

Catalogul **lib** este opțional și va conține resursele *jar* suplimentare cerute de clasele servlet-ului.

Prin intermediul fișierului *index.html* se apelează aplicația servlet. Adresa de apelare (URL - *Universal Resource Locator*) a aplicației servlet este

`http://host:port/catalogAppServlet`

Dacă în loc de *index* fișierul `html` de apelare are alt nume, de exemplu *xyz.html* atunci adresa de apelare va fi

`http://host:port/catalogAppServlet/xyz.html`

*host* este numele calculatorului pe care rulează serverul Web - *gazda* aplicației servlet. Portul implicit utilizat de un server Web container de servlet este 8080.

Catalogul aplicației este denumit *context*-ul servlet-ului.

Apelarea servlet-ului se poate face:

- din meniul *File/Open* al unui navigator;
- ca referință într-un document `html`

`<a href="URL-servlet">...</a>`

Din documentul `html` pomenit anterior, apelarea clasei servlet se face uzual prin intermediul atributului `action` a elementului `form`. Valoarea atributului este numele de apel al servlet-ului.

Plasând fișierul de apelare al aplicației servlet într-un server Web (apache-httpd, Microsoft-IIS) și aplicația servlet într-un servlet Web compatibil, servletul se va apela prin

`http://host/.../xyz.html`

iar clasa servlet se va apela din *xyz.html*.

Prin *Asynchronous JavaScript And Xml - AJAX* un servlet se poate apela dintr-o funcție Javascript.

Legătura cu clasa servlet-ului se poate realiza

- programat – prin adnotări în codul servlet-ului;
- descriptiv – în catalogul `WEB-INF` se editează fișierul `web.xml`.

În versiunile anterioare ale interfeței de programare servlet aceasta a fost unica opțiune.

Trebuie demarcată diferența dintre apelarea / lansarea în execuție a clasei servlet de apelarea aplicației Web.

**Modul programat** se bazează pe adnotarea `javax.servlet.annotation.WebServlet` cu elementele:

String	name
String[ ]	urlPatterns
@WebInitParams[ ]	initParams
boolean	asyncSupported
long	asyncTimeout

**Modul descriptiv** În fișierul `web.xml` apar elementele

1. `<servlet>` leagă *numele servlet-ului* definit în elementul `<servlet-name>` de clasa servlet-ului dat în elementul `<servlet-class>`.
2. `<servlet-mapping>` definește numele sub care servlet-ul identificat prin `<servlet-name>nume_servlet</servlet-name>` se invocă din programul navigator. Acest identificator - *numeApel* - se fixează în elementul `<url-pattern>`. Identificatorul are ca prefix caracterul / (slash).

Structura unui fișier `web.xml` este

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
4   <servlet>
5     <servlet-name>nume_servlet_1</servlet-name>
6     <servlet-class>Nume_clasa</servlet-class>
7   </servlet>
8   <servlet>
9     <servlet-name>nume_servlet_2</servlet-name>
10    <servlet-class>Nume_clasa</servlet-class>
11  </servlet>
12  . . .
13  <servlet-mapping>
14    <servlet-name>nume_servlet_1</servlet-name>
15    <url-pattern>/numeApel_1</url-pattern>
16  </servlet-mapping>
17  <servlet-mapping>
18    <servlet-name>nume_servlet_2</servlet-name>
19    <url-pattern>/numeApel_2</url-pattern>
20  </servlet-mapping>
21  . . .
22 </web-app>

```

Unui element `<servlet>` îi pot corespunde mai multe elemente `<servlet-mapping>`, prin utilizarea de *numeApel* diferite.

Opțional `web.xml` poate conține elementul

```

<welcome-file-list>
  <welcome-file>
    fisier.html sau jsp
  </welcome-file>
</welcome-file-list>

```

cu precizarea fișierelor html sau jsp care apelează aplicația Web. Declarația fișierului *index.html* este implicită.

**Compilarea** clasei servlet necesită completarea variabilei de mediu `classpath` cu fișierul `TOMCAT_HOME\lib\servlet-api.jar`.

Odată completată structura de cataloage și fișiere ale aplicației servlet această structură trebuie copiată în catalogul `TOMCAT_HOME\webapps`. Această operație se numește **desfășurarea** (*deployment*) sau instalarea servlet-ului. Copierea se poate executa și cu serverul Web pornit.

Pentru instalarea unui servlet există mai multe alternative:

- Din catalogul *catalogAppServlet* se realizează arhiva *catalogAppServlet.war*

```
jar cvf catAppServlet.war WEB-INF\* index.html
```

care se copiază în catalogul *TOMCAT\_HOME\webapps*.

Serverul Web *tomcat* va dezarchiveaza arhiva. Astfel servlet-ul este instalat.

Această instalare se numește *instalare dinamică - hot deployment*.

Dacă fișierul **war** este creat, atunci în locul copierii, instalarea se poate face prin componenta *manager* a lui *tomcat*.

- O aplicație servlet arhivată **war** se poate instala de la distanță prin produsele:
  - *apache-tomcat-deployer*
  - *cargo*

### 8.2.1 Codul unui servlet

Un servlet implementează interfața `Servlet` sau extinde una din clasele `GenericServlet` sau `HttpServlet`. `GenericServlet` implementează interfața `Servlet`, iar `HttpServlet` extinde clasa `GenericServlet`. Extinzând clasa `GenericServlet` nu este nevoie de rescrierea tuturor metodelor abstracte ale interfeței `Servlet`.

Metodele interfeței `Servlet` sunt:

- `abstract public void init(ServletConfig config)`

Se apelează o singură dată la lansarea servlet-ului.

- `abstract public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException`

Metoda este apelată de serverul Web pentru rezolvarea cererii unui client.

- `abstract public void destroy()`

Se apelează o singură dată la distrugerea servlet-ului.

- `public String getServletInfo()`

- `public ServletConfig getServletConfig()`

În cele ce urmează o clasă servlet va fi o clasă care extinde clasa `HttpServlet`. În locul metodei `service(...)`, programatorul suprascrie metodele `doGet(...)` sau `doPost(...)`, în funcție de metoda utilizată de client la lansarea cererii.

Practic, un servlet constă din scrierea metodelor

- `void init(ServletConfig config)`

Această metodă este opțională.

```
public void init(ServletConfig config) throws ServletException{
    super.init(config);
    // cod de initializare
}
```

Obiectul *config* are o metodă `String getInitParameter(String nameParam)` cu ajutorul căreia se pot recupera parametri de initializare asociați servlet-ului și care se dau fie prin adnotarea `WebInitParam` prin șablonul

```
import javax.servlet.annotation.WebInitParam;
```

```
@WebServlet(urlPatterns = "/numeApel",
    initParams = {
        @WebInitParam(name = "numeParam", value = "valoareParam"),
        . . .
    }
)
```

fie în fișierul `web.xml` prin elementele

```

<init-param>
    <param-name> NumeleParametrului </param-name>
    <param-value> Valoare </param-value>
</init-param>

```

cuprinse în elementul `<servlet>`.

- `protected void doGet(HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException`  
Tratează o cerere trimisă cu metoda GET (vezi marcajul `<form>`).
- `protected void doPost(HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException`  
Tratează o cerere trimisă cu metoda POST (vezi marcajul `<form>`).

Activitățile de întreprins într-o metodă `doGet()` sau `doPost()` sunt

1. Stabilirea naturii răspunsului:  
`res.setContentType(String tip)`  
unde *tip* specifică tipul *MIME* - *Multipurpose Internet Mail Extensions* al răspunsului:
  - "text/html" - pagină html;
  - "text/xml" - document xml;
  - "text/plain" - text;
  - "image/jpg" - imagine jpg;
  - "image/gif" - imagine gif;
  - "application/json" - date codificate JSON.
2. Se obține o referință către un obiect care realizează transmisia datelor către navigatorul clientului:  
`ServletOutputStream out = res.getOutputStream();`  
sau  
`PrintWriter out=res.getWriter();`
3. Se preiau datele cererii cu una din metodele interfeței `HttpServletRequest`:  
`String getParameter(String numeParametru)`  
`java.util.Enumeration getParameterNames()`  
Adițional se pot afla

- calculatorul cu serverul web: `getServerName()`;
  - portul: `getServerPort()`;
  - catalogul servlet-ului: `getContextPath()`.
4. Rezolvă cererea clientului;
  5. Formează și *scrie* răspunsul;
  6. Închide conexiunea obiectului prin care s-a realizat transmitia datelor către navigatorul clientului prin `out.close()`.

Un câmp (global) declarat în clasa servletului este comun fiecărei instanțe a servletului.

Un utilizator lansează o cerere către servlet. De obicei acest lucru se realizează prin completarea unui formular al unui document html. Programul navigator trimite cererea serverului Web prin intermediul căruia este lansat servlet-ul în acțiune.

**Ciclul de viață al unui servlet.** Când un servlet este apelat prima dată de către serverul Web se execută metoda `init`. După aceasta, fiecărei cereri lansate de un utilizator i se asociază un fir de execuție în care se apelează metoda `service`. Metoda `service` apelează apoi metodele `doGet()`, `doPost()`.

**Exemplul 8.2.1** *Servlet-ul Hello: Clientul transmite numele servlet-ului care îi răspunde cu mesajul de salut "Hi" + nume!.*

Formularul html prin care clientul introduce numele este (*index.html*)

```

1 <!doctype html>
2 <head>
3   <meta charset="utf-8">
4   <link rel="stylesheet" href="mycss.css">
5 </head>
6 <body>
7   <center>
8   <h1> Pagina de apelare a servletului HelloServlet </h1>
9   <form method="post"
10     action="/apphelloP/hello">
11     <table>
12       <tr>
13         <td><label>Introduceți numele </label></td>
14         <td>
15           <input type="text" name="name" size="20">
16         </td>
17       </tr>
18       <tr>
19         <td>
20           <input type="submit" value="Calculeaza">

```

```

21         </td>
22     </tr>
23 </table>
24 </form>
25 </center>
26 </body>
27 </html>

```

*apphelloP* corespunde contextului aplicației.

În modul programat servlet-ului are codul

```

1  import java.io.IOException;
2  import javax.servlet.ServletException;
3  import javax.servlet.http.HttpServlet;
4  import javax.servlet.http.HttpServletRequest;
5  import javax.servlet.http.HttpServletResponse;
6  import javax.servlet.ServletOutputStream;
7  import javax.servlet.annotation.WebServlet;

9  @WebServlet(urlPatterns = "/hello")
10 public class HelloServlet extends HttpServlet {
11     public void doGet(HttpServletRequest req, HttpServletResponse res)
12         throws ServletException, IOException {
13         res.setContentType("text/html");
14         ServletOutputStream out=res.getOutputStream();
15         String nume=req.getParameter("name");
16         out.println("<html>");
17         out.println("<head><title>HelloServlet </title></head>");
18         out.println("<body>");
19         out.println("<h1>HelloServlet </h1>");
20         out.println("<p>");
21         out.println("  Hi  "+ nume+"  !");
22         out.println("</p>");
23         out.println("</body>");
24         out.println("</html>");
25         out.close();
26     }

28     public void doPost(HttpServletRequest req, HttpServletResponse res)
29         throws ServletException, IOException {
30         doGet(req, res);
31     }
32 }

```

În modul descriptiv, în locul liniilor de cod 7-9 va fi folosit fișierul *web.xml*

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
4      <servlet>
5          <servlet-name>hello</servlet-name>
6          <servlet-class>HelloServlet</servlet-class>
7      </servlet>

9      <servlet-mapping>
10         <servlet-name>hello</servlet-name>
11         <url-pattern>/hello</url-pattern>
12     </servlet-mapping>

```



13 | &lt;/web-app&gt;

Compilarea și arhivarea servlet-ului o vom realiza prin intermediul lui *apache-ant*. În acest scop se crează structura:

```
hello
|      |--> src
|      |      HelloServlet.java
|      |--> web
|      |      |--> WEB-INF
|      |      |      |--> classes
|      |      |      |--> lib
|      |      |              web.xml
|      |      |              index.html
|      build.xml
```

și se utilizează fișierul *build.xml*

```

1 <project basedir="." default="generate.war">
2   <property name="TOMCATHOME" value="." />
3   <property name="dist.name" value="apphelloP" />
4   <property name="dist.dir" value="dist" />
5
6   <path id="myclasspath">
7     <fileset dir="web/WEB-INF/lib">
8       <include name="*.jar" />
9     </fileset>
10    <pathelement path="${TOMCATHOME}/lib/servlet-api.jar" />
11  </path>
12
13  <target name="init">
14    <delete dir="${dist.dir}" />
15    <delete dir="web/WEB-INF/classes" />
16    <mkdir dir="web/WEB-INF/classes" />
17    <mkdir dir="${dist.dir}" />
18  </target>
19
20  <target name="compile" depends="init">
21    <javac classpathref="myclasspath"
22      includeantruntime="false"
23      srcdir="${basedir}/src"
24      destdir="web/WEB-INF/classes" />
25  </target>
26
27  <target name="generate.war" depends="compile">
28    <jar destfile="${dist.dir}/${dist.name}.war" basedir="web" />
29  </target>
30 </project>

```

**Observație.** Valoarea parametrului *dist.name* definește catalogul aplicației (contextul) *catalogAppServlet*.

Se lansează în execuție serverul Web *tomcat*, se desfășoară servlet-ul în serverul Web și dintr-un navigator se deschide pagina <http://localhost:8080/apphelloP>

**Exemplul 8.2.2** *Servlet pentru calculul celui mai mare divizor comun a două numere.*

```

1 import java.io.IOException;
2 import java.io.PrintWriter;
3 import javax.servlet.ServletException;
4 import javax.servlet.http.HttpServlet;
5 import javax.servlet.http.HttpServletRequest;
6 import javax.servlet.http.HttpServletResponse;
7 import javax.servlet.annotation.WebServlet;

9 @WebServlet(urlPatterns = "/cmmdc")
10 public class CmmdcServlet extends HttpServlet{

12     public long cmmdc(long m, long n){. . .}

14     public void doGet(HttpServletRequest req,HttpServletResponse res)
15         throws ServletException,IOException{
16         String sm=req.getParameter("m"),sn=req.getParameter("n");
17         String tip=req.getParameter("tip");
18         long m=Long.parseLong(sm),n=Long.parseLong(sn);
19         long x=cmmdc(m,n);
20         PrintWriter out=res.getWriter();
21         if (tip.equals("text/html")){
22             String title="CmmdcServlet";
23             res.setContentType("text/html");
24             out.println("<HTML><HEAD><TITLE>");
25             out.println(title);
26             out.println("</TITLE></HEAD><BODY>");
27             out.println("<H1>"+title+"</H1>");
28             out.println("<P>Cmmdc is "+x);
29             out.println("</BODY></HTML>");
30         }
31         else{
32             res.setContentType("text/plain");
33             out.println(x);
34         }
35         out.close();
36     }

38     public void doPost(HttpServletRequest req,HttpServletResponse res)
39         throws ServletException,IOException{
40         doGet(req,res);
41     }
42 }

```

apelabil prin documentul html (*index.html*)

```

1 <!doctype html>
2 <head>
3     <meta charset="utf-8">
4     <link rel="stylesheet" href="mycss.css">
5 </head>
6 <body>
7     <center>
8     <h1> Pagina de apelare CmmdcServlet </h1>
9     <form method="get"
10         action="/myservlet/cmmdc">
11         <table>
12             <tr>
13                 <td><label> Primul numar </label></td>
14                 <td>

```

```

15         <input type="number" name="m" size="5"
16         required min="1">
17     </td>
18 </tr>
19 <tr>
20     <td><label> Al doilea numar </label></td>
21     <td>
22         <input type="number" name="n" size="5"
23         required min="1">
24     </td>
25 </tr>
26 <tr>
27     <td>
28         <p><input type="submit" value=" Calculeaza">
29     </td>
30 </tr>
31 </table>
32 <input type="hidden" name="tip" value="text/html" >
33 </form>
34 <center>
35 </body>
36 </html>

```

Pentru fixarea naturii răspunsului `text/html` sau `text/plain` s-a introdus variabila *tip*, care în fișierul de invocare *index.html* primește *pe ascuns* valoarea `text/html`. În cazul în care vom apela servlet-ul dintr-un program, va fi avantajos să primim răspunsul ca `text/plain`.

## 8.3 Procesare asincronă în servlet

Rezolvarea unei cereri adresat de un client unui servlet (metoda `doGet` / `doPost`) se execută de către serverul Web într-un fir de execuție. Firul de execuție este creat și lansat de serverul Web.

Interfața de programare Servlet 3.0 oferă posibilitatea unui execuții asincrone: satisfacerea cererii clientului se face într-un fir de execuție lansat de clasa servlet-ului. Terminarea activității clasei servlet nu mai este legată de rezolvarea cererii clientului și de trimiterea răspunsului.

Acest mod de execuție se indică prin adnotarea

```
@WebServlet(urlPatterns="/numeApel" ,asyncSupported=true)
```

Suportul asincron necesită un *context*, obiect de tip `javax.servlet.AsyncContext`. Acest obiect se obține prin

- `AsyncContext asyncCtx=req.startAsync();`
- `AsyncContext asyncCtx=req.startAsync(req,res);`

Activitățile ce trebuie îndeplinite pentru satisfacerea cererii clientului se lansează într-un fir de execuție prin

```
asyncCtx.start(Runnable fir);
```

Metoda `void complete()` a clasei `javax.servlet.AsyncContext` finalizează îndeplinirea activității asincrone.

Interfața `AsyncListener` permite notificarea evenimentelor în serverul Web

- `void onComplete(AsyncEvent ae)`
- `void onTimeout(AsyncEvent ae)`
- `void onError(AsyncEvent ae)`
- `void onStartAsync(AsyncEvent ae)`

Obiectul de tip `AsyncEvent` este creat în momentul satisfacerii cererii clientului, depășirii timpului alocat sau producerii unei erori.

**Exemplul 8.3.1** *Varianta asincronă a servlet-ului de calcul a celui mare divisor comun.*

Codul servlet-ului (*AsyncServlet*) este independent de activitățile ce satisfac cererea clientului. Acele activități sunt programate în clasa *AsyncAction* prin  $\lambda$ -expresii.

Clasa *AsyncServlet*

```

1 package myservlet;
2 import javax.servlet.ServletException;
3 import javax.servlet.http.HttpServlet;
4 import javax.servlet.http.HttpServletRequest;
5 import javax.servlet.http.HttpServletResponse;
6 import javax.servlet.annotation.WebServlet;
7 import javax.servlet.AsyncContext;
8 import java.io.IOException;
9 import listeners.MyAsyncListener;

11 @WebServlet(urlPatterns="/cmmdc", asyncSupported=true)
12 public class AsyncServlet extends HttpServlet{

14     public void doGet(HttpServletRequest req, HttpServletResponse res)
15         throws ServletException, IOException{
16         final AsyncContext asyncCtx=req.startAsync(req, res);
17         AsyncAction asyncAction=new AsyncAction();
18         asyncCtx.addListener(new MyAsyncListener());
19         asyncCtx.start(asyncAction.f.service(asyncCtx));
20     }

22     public void doPost(HttpServletRequest req, HttpServletResponse res)
23         throws ServletException, IOException{
24         doGet(req, res);
25     }
26 }
```

- Clasa *AsyncAction*

```

1 package myservlet;
2 import javax.servlet.AsyncContext;
3 import java.io.PrintWriter;
4 import javax.servlet.ServletException;
5 import javax.servlet.ServletResponse;
6 import java.io.IOException;

8 class AsyncAction{

10     interface CmmdcService {
11         long cmmdc(long m, long n);
12     }

14     static CmmdcService cmmdcService=(long m, long n) ->{. . .}

16     interface IAsyncAction{
17         Thread service(AsyncContext asyncCtx);
18     }

20     IAsyncAction f=(AsyncContext asyncCtx)->{
21         return new Thread()->{
22             ServletRequest req=asyncCtx.getRequest();
23             ServletResponse res=asyncCtx.getResponse();
24             String sm=req.getParameter("m");
25             String sn=req.getParameter("n");
26             long m=Long.parseLong(sm),n=Long.parseLong(sn);
27             long x=cmmdcService.cmmdc(m,n);
28             String result=new Long(x).toString();
29             System.out.println(x);
30             try{
31                 PrintWriter out=res.getWriter();
32                 String title="Cmmdc Servlet";
33                 res.setContentType("text/html");
34                 out.println("<HTML><HEAD><TITLE>");
35                 out.println(title);
36                 out.println("</TITLE></HEAD><BODY>");
37                 out.println("<H1>"+title+"</H1>");
38                 out.println("<P>Cmmdc is "+x);
39                 out.println("</BODY></HTML>");
40                 out.close();
41             }
42             catch(IOException e){
43                 System.out.println(e.getMessage());
44             }
45             asyncCtx.complete();
46         });
47     };
48 }

```

- Clasa *MyAsyncListener* implementează interfața *AsyncListener*

```

1 package listeners;
2 import javax.servlet.AsyncEvent;
3 import javax.servlet.AsyncListener;
4 import javax.servlet.ServletException;

6 public class MyAsyncListener implements AsyncListener{

```

```

8      public MyAsyncListener(){}

10     public void onComplete(AsyncEvent ae){
11         ServletRequest req=ae.getAsyncContext().getRequest();
12         String r=req.getParameter("m")+" <-> "+req.getParameter("n");
13         System.out.println(" AsyncListener: onComplete for request: "+r);
14     }

16     public void onTimeout(AsyncEvent ae){
17         ServletRequest req=ae.getAsyncContext().getRequest();
18         String r=req.getParameter("m")+" <-> "+req.getParameter("n");
19         System.out.println(" AsyncListener: onTimeout for request: "+r);
20     }

22     public void onError(AsyncEvent ae){
23         ServletRequest req=ae.getAsyncContext().getRequest();
24         String r=req.getParameter("m")+" <-> "+req.getParameter("n");
25         System.out.println(" AsyncListener: onError for request: "+r);
26     }

28     public void onStartAsync(AsyncEvent ae){
29         ServletRequest req=ae.getAsyncContext().getRequest();
30         String r=req.getParameter("m")+" <-> "+req.getParameter("n");
31         System.out.println(" AsyncListener: onStartAsync for request: "+r);
32     }
33 }

```

Serverul de aplicație *glassfish* oferă facilități pentru implementarea aplicației servlet:

- Firul de execuție va aparține unui *bazin* de fire de execuție al serverului de aplicație. Există un bazin inițial (implicit) pe care-l vom folosi.

Un bazin propriu se crează executând

Resources -> Concurrent Resources -> Managed Executor Services

1. New
2. Se completează
  - JNDI Name: *executorService*
  - Core Size: 10
3. OK

Resursa se indică prin adnotarea

```

import javax.annotation.Resource;

@Resource
private ManagedExecutorService executorService;

```

Includerea firului de execuție se programează cu metoda

```
Future<?> submit(Runnable task)
```

Codul complet este

### Exemplul 8.3.2

```

1 package myservlet;
2 import javax.servlet.ServletException;
3 import javax.servlet.http.HttpServlet;
4 import javax.servlet.http.HttpServletRequest;
5 import javax.servlet.http.HttpServletResponse;
6 import javax.servlet.annotation.WebServlet;
7 import javax.servlet.AsyncContext;
8 import java.io.IOException;
9 import javax.enterprise.concurrent.ManagedExecutorService;
10 import javax.annotation.Resource;
11 import java.io.PrintWriter;

13 @WebServlet(urlPatterns="/cmmdc", asyncSupported=true)
14 public class AsyncServlet extends HttpServlet{

16     @Resource private ManagedExecutorService executorService;

18     public void doGet(HttpServletRequest req, HttpServletResponse res)
19         throws ServletException, IOException{
20         final AsyncContext asyncCtx=req.startAsync();
21         executorService.submit(()->{
22             res.setContentType("text/html");
23             String sm=req.getParameter("m"), sn=req.getParameter("n");
24             String tip=req.getParameter("tip");
25             long m=Long.parseLong(sm), n=Long.parseLong(sn);
26             long x=cmmdc(m,n);
27             try{
28                 PrintWriter out=res.getWriter();
29                 if(tip.equals("text/html")){
30                     String title="Cmmdc Servlet";
31                     res.setContentType("text/html");
32                     out.println("<HTML><HEAD><TITLE>");
33                     out.println(title);
34                     out.println("</TITLE></HEAD><BODY>");
35                     out.println("<H1>"+title+"</H1>");
36                     out.println("<P>Cmmdc is "+x);
37                     out.println("</BODY></HTML>");
38                 }
39                 else{
40                     res.setContentType("text/plain");
41                     out.println(x);
42                 }
43                 out.close();
44                 asyncCtx.complete();
45             }
46             catch(IOException e){}
47         });
48     }

50     public void doPost(HttpServletRequest req, HttpServletResponse res)
51         throws ServletException, IOException{
52         doGet(req, res);
53     }

55     public long cmmdc(long m, long n){. . .}
56 }

```

- Acțiunea firului de execuție poate fi inclusă într-o componentă Java care va fi instanțiată și injectată de serverul de aplicație.

Injectarea este indicată prin

```
import javax.inject.Inject;

@Inject
private CmmdcAction obj;
```

Clasa *CmmdcAction* nu conține decât metoda *cmmdc* din codul de mai sus.

În plus este nevoie fișierul *beans.xml*

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <beans
3   xmlns="http://xmlns.jcp.org/xml/ns/javaee"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
6   http://xmlns.jcp.org/xml/ns/javaee/beans_1_1.xsd"
7   bean-discovery-mode="all">
8 </beans>
```

plasat în catalogul WEB-INF.

## 8.4 Dezvoltări în *servlet-api 3.1*

Implementarea de referință a interfeței de programare (API) *servlet-api 3.1* este conținută în produsul *glassfish4*.

### 8.4.1 Procesare asincronă neblocantă

Diferența majoră constă în faptul că solicitarea clientului este trimisă ne-mijlocit și îndeplinită de o clasă tip *listener*.

Clasa `javax.servlet.ServletInputStream`

**Metode**

- `public int readLine(byte[] b, int off, int len) throws IOException`
- `public void setReadListener(ReadListener readListener)`

Interfața `javax.servlet.ReadListener` declară metodele

- `void onDataAvailable() throws IOException`
- `void onAllDataRead() throws IOException`



- `void onError(Throwable t)`

Clasa care implementează această interfață satisface cererea clientului.

Se va mai utiliza interfața `javax.servlet.WriteListener` care declară metodele

- `void onWritePossible() throws java.io.IOException`
- `void onError(java.lang.Throwable throwable)`

### Exemplul 8.4.1

```

1 import java.io.IOException;
2 //import java.io.PrintWriter;
3 import java.net.URL;
4 import javax.servlet.AsyncContext;
5 import javax.servlet.ServletException;
6 import javax.servlet.ServletInputStream;
7 import javax.servlet.ServletOutputStream;
8 import javax.servlet.annotation.WebServlet;
9 import javax.servlet.http.HttpServlet;
10 import javax.servlet.http.HttpServletRequest;
11 import javax.servlet.http.HttpServletResponse;

13 @WebServlet(urlPatterns = {"/nonblock"}, asyncSupported=true)
14 public class NonBlockingServlet extends HttpServlet {
15     protected void doGet(HttpServletRequest req, HttpServletResponse res)
16         throws ServletException, IOException {
17         res.setContentType("text/html; charset=UTF-8");
18         ServletInputStream input = req.getInputStream();
19         ServletOutputStream output = res.getOutputStream();
20         output.println("<html>");
21         output.println("<head>");
22         output.println("<title>Servlet ReadTestServlet</title>");
23         output.println("</head>");
24         output.println("<body>");
25         output.println("<h1>Servlet NonBlockingServlet at " +
26             req.getContextPath() + "</h1>");
27         AsyncContext context = req.startAsync();
28         CmmdcListener listener=new CmmdcListener(input, context, output);
29         output.println("</body>");
30         output.println("</html>");
31     }

33     @Override
34     protected void doPost(HttpServletRequest req, HttpServletResponse res)
35         throws ServletException, IOException {
36         doGet(req, res);
37     }
38 }

```

împreună cu clasa care îndeplinește cererea clientului

```

1 import java.io.IOException;
2 import javax.servlet.AsyncContext;
3 import javax.servlet.ReadListener;
4 import javax.servlet.ServletOutputStream;
5 import javax.servlet.WriteListener;
6 import javax.servlet.ServletInputStream;
7 import java.io.IOException;

9 public class CmmdcListener implements ReadListener, WriteListener{

11     private ServletInputStream input = null;
12     private AsyncContext context = null;
13     private ServletOutputStream out = null;
14     private boolean readFinished=false;
15     private String rez=null;
16     private String data=null;

18     public CmmdcListener(ServletInputStream in,
19         AsyncContext ac, ServletOutputStream output) {
20         input = in;
21         context = ac;
22         out = output;
23         in.setReadListener(this);
24         out.setWriteListener(this);
25     }

27     @Override
28     public void onDataAvailable() {
29         try {
30             int len = 0;
31             byte b[] = new byte[1024];
32             while (input.isReady() && (len>-1)) {
33                 len=input.read(b);
34             }
35             data = new String(b, 0, len);
36             System.out.println(data);
37         }
38         catch (IOException e) {
39             System.out.println("onAvailableException : "+e.getMessage());
40         }
41     }

43     @Override
44     public void onAllDataRead() throws IOException{
45         System.out.println("onAllDataRead");
46         readFinished=true;
47         System.out.println(data);
48         rez=solver(data);
49         context.complete();
50         onWritePossible();
51     }

53     @Override
54     public void onWritePossible() throws IOException {
55         while(!readFinished);
56         out.println("<p>");
57         out.println(rez);
58     }

```

```

60  @Override
61  public void onError(Throwable t) {
62      t.printStackTrace();
63      context.complete();
64  }

66  private String solver(String data){
67      String [] s=data.split("&");
68      String [] s0=s[0].split("=");
69      long m=Long.parseLong(s0[1]);
70      String [] s1=s[1].split("=");
71      long n=Long.parseLong(s1[1]);
72      String r=(new Long(cmmdc(m,n)).toString());
73      return "Cmmdc : "+r;
74  }

76  private long cmmdc(long m, long n){. . .}
77  }

```

### 8.4.2 Modificarea protocolului http: *upgrade*

Schema este utilizată deja în protocolului *websocket*.

În apelul *http* trebuie inserat antetul (*header*) *Upgrade*. Odată această antet recunoscut, solicitarea clientului este rezolvată într-o clasă care implementează interfața *javax.servlet.http.HttpUpgradeHandler*.

Interfața declară metodele:

- void *init*(*WebConnection wc*)
- void *destroy*()

#### Interfața *javax.servlet.http.WebConnection*

Metode

- *ServletInputStream* *getInputStream*() throws *IOException*
- *ServletOutputStream* *getOutputStream*() throws *IOException*

Clasa care implementează interfața *HttpUpgradeHandler* se declară în *servlet* prin metoda *upgrade* a interfeței *HttpServletRequest*

<T extends *HttpUpgradeHandler*> T *upgrade*(Class<T> *handlerClass*)  
throws *IOException*, *ServletException*

În principiu clasa *servlet*-ului este independentă de specificul aplicației.

```

1 import javax.servlet.ServletException;
2 import javax.servlet.http.HttpServlet;
3 import javax.servlet.http.HttpServletRequest;
4 import javax.servlet.http.HttpServletResponse;
5 import javax.servlet.annotation.WebServlet;

7 @WebServlet(urlPatterns = {"/upgrade"})

9 public class UpgradeServlet extends HttpServlet{

11     public void doGet(HttpServletRequest req, HttpServletResponse res)
12         throws ServletException, IOException{
13         if ("upgrade".equals(req.getHeader("Upgrade"))){
14             res.setStatus(101);
15             res.setHeader("Upgrade", "upgrade");
16             res.setHeader("Connection", "Upgrade");
17             res.flushBuffer();
18             System.out.println("Upgrade OK "+req.getHeader("Upgrade"));
19             MyHttpUpgradeHandler handler = req.upgrade(MyHttpUpgradeHandler.class);
20         }
21         else{
22             System.out.println("No upgrade: ");
23         }
24     }

26     public void doPost(HttpServletRequest req, HttpServletResponse res)
27         throws ServletException, IOException{
28         doGet(req, res);
29     }
30 }

```

Pentru problema calculului celui mai mare divizor comun a două numere naturale clasa *MyHttpUpgradeHandler* este

```

1 import javax.servlet.ServletOutputStream;
2 import javax.servlet.ServletInputStream;
3 import javax.servlet.http.WebConnection;
4 import javax.servlet.http.HttpUpgradeHandler;

6 public class MyHttpUpgradeHandler implements HttpUpgradeHandler {
7     private WebConnection wc = null;

9     @Override
10    public void init(WebConnection wc) {
11        this.wc=wc;
12        try{
13            ServletInputStream input = wc.getInputStream();
14            ServletOutputStream output=wc.getOutputStream();
15            String CRLF = "\r\n";
16            String resStr = "update/1.0 " + CRLF;
17            resStr += "Server: Glassfish/ServerTest" + CRLF;
18            resStr += "Content-Type: text/html" + CRLF;
19            resStr += "Connection: Upgrade" + CRLF;
20            resStr += CRLF;
21            byte[] b=new byte[256];
22            input.read(b);
23            String data=new String(b).trim();
24            String rez=solver(data);
25            resStr +=rez + CRLF;

```

```

26         output.write(resStr.getBytes());
27         output.flush();
28     }
29     catch(Exception ex) {
30         throw new RuntimeException(ex);
31     }
32 }

34 @Override
35 public void destroy() {
36     try{
37         wc.close();
38     }
39     catch (Exception ex) {
40         System.out.println("Destroy wc Exception : "+ex.getMessage());
41     }
42 }

44 private String solver(String data){
45     System.out.println("Solver : "+data);
46     String [] s=data.split(" ");
47     String r="-1";
48     try{
49         long m=Long.parseLong(s[0]);
50         long n=Long.parseLong(s[1]);
51         r=(new Long(cmmdc(m,n))).toString();
52     }
53     catch(NumberFormatException e){
54         System.out.println("NumberFormatException : "+e.getMessage());
55     }
56     return "Cmmdc : "+r;
57 }

59 private long cmmdc(long m, long n){. . .}
60 }

```

Problema care rămâne de rezolvat este inserarea antetului **Update**. Rezolvarea consta în generarea unui mesaj **http** care se va transmite printr-un soclu TCP. Mesajul **http** folosește metoda **post**, iar datele din corpul mesajului sunt preluate prin interfața **HttpInputStream**.

Lansarea aplicației se face dintr-un program client, în care se generează mesajul **http** cu antetul **Upgrade**, mesaj expediat printr-un soclu TCP.

Client Java

```

1 import java.net.Socket;
2 import java.io.InputStream;
3 import java.io.OutputStream;
4 import java.io.InputStreamReader;
5 import java.io.BufferedReader;
6 import java.io.IOException;
7 import java.util.Scanner;

9 public class JClient{
10     public static void main(String[] args){
11         Scanner scanner=new Scanner(System.in);
12         System.out.println("m=");

```

```

13     long m=scanner.nextLong();
14     String sm=new Long(m).toString();
15     System.out.println("n=");
16     long n=scanner.nextLong();
17     String sn=new Long(n).toString();
18     String data=sm+" "+sn;
19     String host="localhost";
20     String port="8080";
21     String contextRoot="/upgrade";

23     String CRLF = "\r\n";
24     String reqStr = "POST " + contextRoot + "/upgrade HTTP/1.1" + CRLF;
25     reqStr += "User-Agent: Java/1.7" + CRLF;
26     reqStr += "Host: " + host + ":" + port + CRLF;
27     reqStr += "Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2" +
28         CRLF;
29     reqStr += "Upgrade: upgrade" + CRLF;
30     reqStr += "Connection: Upgrade" + CRLF;
31     reqStr += "Content-type: application/x-www-form-urlencoded" + CRLF;
32     reqStr += "Transfer-Encoding: chunked" + CRLF;
33     reqStr += CRLF;
34     reqStr += data + CRLF;

36     Socket socket=null;
37     try{
38         socket=new Socket(host,Integer.parseInt(port));
39     }
40     catch(Exception e){
41         System.out.println(e.getMessage());
42     }
43     try(
44         OutputStream out=socket.getOutputStream();
45         InputStream in=socket.getInputStream();
46         InputStreamReader isr=new InputStreamReader(in);
47         BufferedReader br=new BufferedReader(isr);
48     ){
49         out.write(reqStr.getBytes());
50         out.flush();
51         String s;
52         while((s=br.readLine())!=null){
53             System.out.println(s);
54         };
55         socket.close();
56         //System.exit(0);
57     }
58     catch(IOException e){
59         System.out.println("Input Exception : "+e.getMessage());
60     }
61 }
62 }

```

### Client servlet

```

1 import java.io.BufferedReader;
2 import java.io.IOException;
3 import java.io.InputStream;
4 import java.io.InputStreamReader;
5 import java.io.OutputStream;
6 import java.io.PrintWriter;

```

```

7 import java.net.Socket;

9 import javax.servlet.ServletException;
10 import javax.servlet.annotation.WebServlet;
11 import javax.servlet.http.HttpServlet;
12 import javax.servlet.http.HttpServletRequest;
13 import javax.servlet.http.HttpServletResponse;

15 @WebServlet(urlPatterns = {"/client"})
16 public class ClientServlet extends HttpServlet {

18     protected void doGet(HttpServletRequest req, HttpServletResponse res)
19         throws ServletException, IOException {
20         String m=req.getParameter("m");
21         String n=req.getParameter("n");

23         res.setContentType("text/html;charset=UTF-8");
24         PrintWriter out = res.getWriter();

26         final String CRLF = "\r\n";
27         final String host = req.getServerName(); // "localhost";
28         final int port = req.getServerPort(); // 8080;
29         final String contextRoot = "/upgrade";
30         final String data = m+" "+n;
31         InputStream input = null;
32         OutputStream output = null;
33         BufferedReader reader = null;
34         Socket s = null;
35         try{
36             out.println("<html>");
37             out.println("<head>");
38             out.println("<title>Servlet ClientTest</title>");
39             out.println("</head>");
40             out.println("<body>");
41             out.println("<h1>Http Upgrade Process</h1>");

43             // Setting the HTTP upgrade req header
44             String reqStr = "POST " + contextRoot + "/upgrade HTTP/1.1" + CRLF;
45             reqStr += "User-Agent: Java/1.7" + CRLF;
46             reqStr += "Host: " + host + ":" + port + CRLF;
47             reqStr += "Accept:
48                 text/html, image/gif, image/jpeg, *, q=.2, */*; q=.2" + CRLF;
49             reqStr += "Upgrade: upgrade" + CRLF;
50             reqStr += "Connection: Upgrade" + CRLF;
51             reqStr += "Content-type: application/x-www-form-urlencoded" + CRLF;
52             reqStr += "Transfer-Encoding: chunked" + CRLF;
53             reqStr += CRLF;
54             reqStr += data + CRLF;

56             s = new Socket(host, port);
57             input = s.getInputStream();
58             output = s.getOutputStream();
59             output.write(reqStr.getBytes());
60             output.flush();

62             out.println("<h2>Sending upgrade req to server.....</h2>");
63             out.println("<h3>Request header with data:</h3>");
64             out.println();

```

```

66     String reqStrDisplay = reqStr.replaceAll("\r\n", "<br>");
67     out.println(reqStrDisplay);
68     out.println("_____ </br></br>");
69     out.flush();

71     reader = new BufferedReader(new InputStreamReader(input));

73     out.println("<h2>Server accept upgrade req, send back the res:</h2>");
74     out.println("<h3>Response header:</h3>");
75     out.println();

77     // Reading the res, and displaying the header from server
78     printHeader(reader, out);
79     out.println("_____ </br></br>");
80     out.flush();

82     out.println("<h2>Server send back the res with new protocol and data:</h2>");
83     out.println("<h3>Response header with data:</h3>");

85     // Reading the res, and displaying the header from server
86     printHeader(reader, out);

88     // Reading the echo data
89     String dataOutput;
90     if ((dataOutput = reader.readLine()) != null) {
91         // Print out the data after header
92         out.println("</br>" + dataOutput + "</br>");
93         out.println("_____ </br></br>");
94         out.println("<h2>Connection with new protocol established</h2>");
95     }
96     out.flush();
97     out.println("</body>");
98     out.println("</html>");
99 }
100 catch (IOException e) {
101     System.out.println("ClientServlet Exception : "+e.getMessage());
102 }
103 finally{
104     if (reader != null) {
105         reader.close();
106     }
107     if (output != null) {
108         output.close();
109     }
110     if (input != null) {
111         input.close();
112     }
113     if (s != null) {
114         s.close();
115     }
116 }
117 }

119 protected void printHeader(BufferedReader reader, PrintWriter out)
120     throws IOException {
121     for(String line; (line = reader.readLine()) != null;) {
122         if(line.isEmpty()){
123             break;
124         }

```



```
125     out.println(line + "</br>");
126   }
127 }
129 @Override
130 protected void doPost(HttpServletRequest req, HttpServletResponse res)
131     throws ServletException, IOException {
132     doGet(req, res);
133 }
134 }
```

## 8.5 Facilități de programare cu servlet

### 8.5.1 Program client al unui servlet

Apelarea unui servlet dintr-un program Java – adică lansarea unei cereri și recepționarea răspunsului furnizat de servlet se poate obține

- sincron cu produsul *httpcomponents-client*,
- asincron cu unul din produsele
  - *httpcomponents-client*,
  - *httpcomponents-asyncclient*.

Ambele produse sunt dezvoltate de *apache*.

Caracterul sincron / asincron constă în faptul că recepția și prelucrarea răspunsului oferit de servlet are loc în metoda din care s-a lansat cererea, respectiv într-un alt obiect.

Într-un asemenea caz, din punctul de vedere al clientului este mai avantajos ca răspunsul servlet-ului fie `text/plain`, în loc de `text/html`.

#### Cazul sincron

Catalogul `lib` din *httpcomponents-client* conține, printre altele, fișierele *httpcore-\*.jar*, *httpclient-\*.jar*, *commons-logging-\*.jar*.

Pentru compilare trebuie declarată în variabila de sistem `classpath` referința către *httpclient-\*.jar* și *httpcore-\*.jar* dar pentru execuție este nevoie și de referința către *commons-logging-\*.jar*.

Dezvoltarea unui client presupune:

1. Crearea unui obiect de tip `org.apache.http.client.CloseableHttpClient`:

```
CloseableHttpClient httpClient = HttpClients.createDefault();
```

2. Declararea metodei de transmitere a datelor *get*, *post* cu inserarea datelor din cerere.

- GET

```
HttpGet httpget = new HttpGet(uri);
```

unde `uri` este `String`-ul de apelare a servlet-ului, de forma

*http://host:port/catalog/numeApel?numeParam=valParam&...*,

O soluție mai elegantă este

```
List<NameValuePair> qparams = new ArrayList<NameValuePair>();
qparams.add(new BasicNameValuePair("param_1", value_1));
qparams.add(new BasicNameValuePair("param_2", value_2));
. . .
try{
    URI uri = URIUtils.createURI("http", "localhost", 8080,
        "/catalog/numeApel",URLEncodedUtils.format(qparams,"UTF-8"),
        null);
    HttpGet httpget = new HttpGet(uri);
    . . .
}
catch(. . .){. . .}
```

- POST

```
List<NameValuePair> qparams = new ArrayList<NameValuePair>();
qparams.add(new BasicNameValuePair("param_1", value_1));
qparams.add(new BasicNameValuePair("param_2", value_2));
. . .
try{
    UrlEncodedFormEntity params=new UrlEncodedFormEntity(qparams,
        "UTF-8");
    HttpPost httppost=new HttpPost(uri);
    httppost.setEntity(params);
    . . .
}
catch(. . .){. . .}
```

cu `uri="http://host:port/catalog/numeApel"`.

Clasele `HttpGet`, `HttpPost` aparțin pachetului `org.apache.http.client.methods`.

### 3. Lansarea cererii.

```
HttpResponse response=httpclient.execute(httpget);
```

respectiv

```
HttpResponse response=httpclient.execute(httppost);
```

### 4. Preluarea răspunsului.

```
HttpEntity entity=response.getEntity();
if(entity!=null){
    InputStream is=entity.getContent();
    int l;
    byte[] tmp=new byte[2048];
    while((l=is.read(tmp))!=-1){}
    . . .
}
```

**Exemplul 8.5.1** *Dezvoltăm un program client pentru servlet-ul `CmmdcServlet` (`post`).*

```
1 import java.util.Scanner;
2 import org.apache.http.HttpEntity;

4 import org.apache.http.impl.client.CloseableHttpClient;
5 import org.apache.http.client.methods.HttpPost;
6 import org.apache.http.client.methods.CloseableHttpResponse;
7 import org.apache.http.impl.client.HttpClients;
8 import java.util.List;
9 import java.util.ArrayList;
10 import org.apache.http.NameValuePair;
11 import org.apache.http.message.BasicNameValuePair;

13 import org.apache.http.client.entity.UrlEncodedFormEntity;
14 import java.io.*;

16 public class ClientCmmdcServlet{
17     static String uri="http://localhost:8080/myservlet/cmmdc";

19     public static void main(String[] args) {
20         Scanner scanner=new Scanner(System.in);
21         System.out.println("m=");
22         String m=scanner.nextLine().trim();
```

```

23      System.out.println("n=");
24      String n=scanner.nextLine().trim();

26      CloseableHttpClient httpclient = HttpClients.createDefault();
27      List<NameValuePair> qparams = new ArrayList<NameValuePair>();
28      qparams.add(new BasicNameValuePair("m", m));
29      qparams.add(new BasicNameValuePair("n", n));
30      qparams.add(new BasicNameValuePair("tip", "text/plain"));
31      try{
32          UrlEncodedFormEntity params=new UrlEncodedFormEntity(qparams,"UTF-8");
33          HttpPost httppost=new HttpPost(uri);
34          httppost.setEntity(params);
35          CloseableHttpResponse response=httpclient.execute(httppost);
36          HttpEntity entity=response.getEntity();
37          if(entity!=null){
38              InputStream is=entity.getContent();
39              int l;
40              byte[] tmp=new byte[2048];
41              while((l=is.read(tmp))!=-1){}
42              System.out.println("Cmmdc = "+(new String(tmp).trim()));
43          }
44      }
45      catch(Exception e){
46          System.out.println("Exception : "+e.getMessage());
47      }
48  }
49 }

```

### Varianta *fluentă*

```

1  import java.util.Scanner;
2  import org.apache.http.client.fluent.Form;
3  import org.apache.http.client.fluent.Request;

5  public class ClientCmmdcServlet1{
6      static String uri="http://localhost:8080/myservlet/cmmdc";

8      public static void main(String[] args) {
9          Scanner scanner=new Scanner(System.in);
10         System.out.println("m=");
11         String m=scanner.nextLine().trim();
12         System.out.println("n=");
13         String n=scanner.nextLine().trim();

15         System.out.println("Post method request");
16         try{
17             String result=Request.Post(uri)
18                 .bodyForm(Form.form()
19                     .add("m",m)
20                     .add("n",n)
21                     .add("tip","text/plain")
22                     .build())
23                 .execute().returnContent().asString();
24             System.out.println("Cmmdc = "+result);
25         }
26         catch(Exception e){
27             System.out.println("Exception : "+e.getMessage());
28         }

```

```

30     System.out.println("Get method request");
31     String data=" ?m="+m+"&n="+n+"&tip=text/plain";
32     try{
33         String result=Request.Get(uri+data)
34             .execute().returnContent().asString();
35         System.out.println("Cmmdc = "+result);
36     }
37     catch(Exception e){
38         System.out.println("Exception : "+e.getMessage());
39     }
40 }
41 }

```

O soluție elementară bazată pe clasa `java.net.HttpURLConnection` este

```

1  import java.net.URL;
2  import java.net.HttpURLConnection;
3  import java.util.Scanner;
4  import java.io.InputStreamReader;
5  import java.io.BufferedReader;
6  import java.io.PrintWriter;

8  public class CmmdcClient{
9      public static void main(String [] args){
10         Scanner scanner=new Scanner(System.in);
11         System.out.println("m=");
12         long m=scanner.nextLong();
13         System.out.println("n=");
14         long n=scanner.nextLong();
15         String msg="m="+m+"&n="+n+"&tip=text/plain";
16         System.out.println("HttpURLConnection cu metoda GET");
17         try{
18             String urlGET="http://localhost:8080/myservlet/cmmdc?" +msg;
19             URL url=new URL(urlGET);
20             HttpURLConnection conn=(HttpURLConnection)url.openConnection();
21             conn.setRequestMethod("GET");
22             System.out.println(conn.getResponseCode());
23             System.out.println(conn.getResponseMessage());
24             BufferedReader br=
25                 new BufferedReader(new InputStreamReader(conn.getInputStream()));
26             String s;
27             while((s=br.readLine())!=null){
28                 System.out.println(s);
29             }
30             br.close();
31             conn.disconnect();
32         }
33         catch(Exception e){
34             System.out.println(e.getMessage());
35         }
36         System.out.println("HttpURLConnection cu metoda POST");
37         try{
38             String urlPOST="http://localhost:8080/myservlet/cmmdc";
39             URL url=new URL(urlPOST);
40             HttpURLConnection conn=(HttpURLConnection)url.openConnection();
41             conn.setRequestMethod("POST");
42             conn.setUseCaches(false);
43             conn.setDoInput(true);
44             conn.setDoOutput(true);

```

```

45     conn.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
46     PrintWriter pw=new PrintWriter(conn.getOutputStream());

48     pw.println(msg);
49     pw.flush();
50     System.out.println(conn.getResponseCode());
51     System.out.println(conn.getResponseMessage());
52     BufferedReader br=
53         new BufferedReader(new InputStreamReader(conn.getInputStream()));
54     String s;
55     while((s=br.readLine())!=null){
56         System.out.println(s);
57     }
58     br.close();
59     pw.close();
60     conn.disconnect();
61 }
62 catch(Exception e){
63     System.out.println(e.getMessage());
64 }
65 }
66 }

```

## Cazul asincron

### 1. *httpcomponent-client*

```

1  import java.util.Scanner;
2  import org.apache.http.client.fluent.Form;
3  import org.apache.http.client.fluent.Request;
4  import org.apache.http.client.fluent.Async;
5  import org.apache.http.client.fluent.Content;
6  import java.util.concurrent.ExecutorService;
7  import java.util.concurrent.Executors;
8  import java.util.concurrent.Future;
9  import org.apache.http.concurrent.FutureCallback; //HttpCore

11 public class AsyncClientCmmdcServlet{
12     static String uri="http://localhost:8080/myservlet/cmmdc";

14     public static void main(String[] args) {
15         Scanner scanner=new Scanner(System.in);
16         System.out.println("m=");
17         String m=scanner.nextLine().trim();
18         System.out.println("n=");
19         String n=scanner.nextLine().trim();

21         ExecutorService threadpool = Executors.newFixedThreadPool(2);
22         Async async = Async.newInstance().use(threadpool);

24         System.out.println("Post method request");
25         try{
26             Request request=Request.Post(uri)
27                 .bodyForm(Form.form()
28                     .add("m",m)
29                     .add("n",n)
30                     .add("tip","text/plain")

```

```

31         . build ( ) ;
32
33     // Varianta 1
34     /*
35     Future<Content> future=async.execute(request);
36     while (!future.isDone()){};
37     System.out.println("Cmmdc = "+future.get().asString());
38     */
39
40     // Varianta 2
41     Future<Content> future=async.execute(request ,
42     new FutureCallback<Content>(){
43     @Override
44     public void completed(final Content content){
45     System.out.println("Cmmdc : "+content.asString());
46     }
47     @Override
48     public void failed(final Exception e) {
49     System.out.println(e.getMessage() + ": " + request);
50     }
51     @Override
52     public void cancelled() {}
53     });
54     while (!future.isDone()){};
55
56     threadpool.shutdown();
57 }
58 catch (Exception e){
59     System.out.println("Exception : "+e.getMessage());
60 }
61 }
62 }

```

## 2. *httpcomponent-asyncclient*

Variabila `classpath` conține referențele către fișerele `jar` aflate în catalogul `lib` din *httpcomponent-asyncclient*.

Programarea constă din

- (a) Crearea unei instanțe a clasei  
`org.apache.http.impl.nio.client.CloseableHttpAsyncClient`  
`CloseableHttpAsyncClient httpclient =`  
`HttpAsyncClients.createDefault();`  
`httpclient.start();`
- (b) Lansarea unei cereri apelând o metodă `execute`:  
`<T> Future<T> execute(`  
`org.apache.http.nio.protocol.HttpAsyncRequestProducer`  
`requestProducer,`  
`org.apache.http.nio.protocol.HttpAsyncResponseConsumer<T>`

*responseConsumer*,  
`org.apache.http.concurrent.FutureCallback<T> callback)`

O instanță de tip `HttpAsyncRequestProducer` se obține prin

`HttpAsyncMethods.createGet(uri)`

Drept instanță a clasei `HttpAsyncResponseConsumer` poate fi o clasă ce extinde clasa `org.apache.http.nio.client.methods.AsyncCharConsumer`.

### Exemplul 8.5.2

```

1 import java.util.Scanner;

3 import org.apache.http.HttpResponse;
4 import org.apache.http.impl.nio.client.HttpAsyncClients;
5 import org.apache.http.nio.IOControl;
6 import org.apache.http.impl.nio.client.CloseableHttpAsyncClient;
7 import org.apache.http.nio.client.methods.AsyncCharConsumer;
8 import org.apache.http.nio.client.methods.HttpAsyncMethods;
9 import org.apache.http.protocol.HttpContext;

11 import java.nio.CharBuffer;
12 import java.io.IOException;
13 import java.util.concurrent.Future;

15 public class AsyncClientCmmdcServlet{
16     static String uri="http://localhost:8080/myservlet/cmmdc";

18     public static void main(String[] args) throws Exception {
19         Scanner scanner=new Scanner(System.in);
20         System.out.println("m=");
21         String m=scanner.nextLine().trim();
22         System.out.println("n=");
23         String n=scanner.nextLine().trim();
24         String requestData="?m="+m+"&n="+n+"&tip=text/plain";
25         System.out.println(requestData);
26         // Create an instance of HttpAsyncClient.

28         CloseableHttpAsyncClient httpclient =
29             HttpAsyncClients.createDefault();
30         httpclient.start();
31         try {
32             Future<Boolean> future = httpclient.execute(
33                 HttpAsyncMethods.createGet(uri+requestData),
34                 new MyResponseConsumer(), null);
35             Boolean result = future.get();
36             if (result != null && result.booleanValue()) {
37                 System.out.println("Request successfully executed");
38             }
39             else {
40                 System.out.println("Request failed");
41             }
42             System.out.println("Shutting down");

```



```

43     }
44     finally {
45         httpClient.close();
46     }
47     System.out.println("Done");
48 }

50 static class MyResponseConsumer extends AsyncCharConsumer<Boolean> {
51     @Override
52     protected void onResponseReceived(final HttpResponse response){}

54     @Override
55     protected void onCharReceived(final CharBuffer buf,
56         final IOControl ioctl) throws IOException {
57         System.out.println("Cmmdc : ");
58         while (buf.hasRemaining()) {
59             System.out.print(buf.get());
60         }
61     }

63     @Override
64     protected void releaseResources() {}

66     @Override
67     protected Boolean buildResult(final HttpContext context){
68         return Boolean.TRUE;
69     }
70 }
71 }

```

## 8.5.2 Servlete înlănțuite

Un servlet apelează la un moment dat alt servlet. Șablonul de lucru este

```

RequestDispatcher dispatcher = getServletContext()
    .getRequestDispatcher("/url_pattern_servlet_apelat");
if(dispatcher!=null)
    dispatcher.include(request,response);

```

**Exemplul 8.5.3** *Un servlet VerifServlet verifică parametri cererii. Pentru problema calculului celui mai mare divizor comun a două numere, dacă cei doi parametri sunt numere întregi, atunci se apelează servlet-ul ComputeServlet, altfel se formează un mesaj de eroare.*

Codurile celor două servlete sunt:  
*VerifServlet.java*

```

1 package cmmdc;
2 import java.io.IOException;
3 import java.io.PrintWriter;
4 import javax.servlet.ServletException;

```

```

5 import javax.servlet.http.HttpServlet;
6 import javax.servlet.http.HttpServletRequest;
7 import javax.servlet.http.HttpServletResponse;
8 import javax.servlet.RequestDispatcher;
9 import javax.servlet.annotation.WebServlet;

11 @WebServlet(urlPatterns = "/verif")
12 public class VerifServlet extends HttpServlet{
13     public void doGet(HttpServletRequest req, HttpServletResponse res)
14         throws ServletException, IOException{
15         PrintWriter out=res.getWriter();
16         res.setContentType("text/html");
17         String sm=req.getParameter("m"),sn=req.getParameter("n");
18         String message="";
19         long m,n;
20         if((sm==null)|| (sm.equals(""))){
21             message="Numar absent";
22         }
23         else{
24             try{
25                 m=Long.parseLong(sm);
26             }
27             catch(NumberFormatException e){
28                 message="Nu este numar";
29             }
30         }
31         if((sn==null)|| (sn.equals(""))){
32             message="Numar absent";
33         }
34         else{
35             try{
36                 n=Long.parseLong(sn);
37             }
38             catch(NumberFormatException e){
39                 message="Nu este numar";
40             }
41         }
42         out.println("<html><body>");
43         if(message.equals("")){
44             out.println("<h3> Rezultatul ob&#355;inut </h3>");
45             RequestDispatcher dispatcher=
46                 getServletContext().getRequestDispatcher("/calcul");
47             if(dispatcher!=null)
48                 dispatcher.include(req, res);
49         }
50         else{
51             out.println("<h3> Date eronate </h3>");
52             out.println(message);
53         }
54         out.println("</body></html>");
55         out.close();
56     }

58     public void doPost(HttpServletRequest req, HttpServletResponse res)
59         throws ServletException, IOException{
60         doGet(req, res);
61     }
62 }

```

*ComputeServlet.java*

```

1 package cmmdc;
2 import java.io.IOException;
3 import java.io.PrintWriter;
4 import javax.servlet.ServletException;
5 import javax.servlet.http.HttpServlet;
6 import javax.servlet.http.HttpServletRequest;
7 import javax.servlet.http.HttpServletResponse;
8 import javax.servlet.annotation.WebServlet;

10 @WebServlet(urlPatterns = "/calcul")
11 public class ComputeServlet extends HttpServlet{

13     public long cmmdc(long m, long n){. . .}

15     public void doGet(HttpServletRequest req, HttpServletResponse res)
16         throws ServletException, IOException{
17         long m=Long.parseLong(req.getParameter("m"));
18         long n=Long.parseLong(req.getParameter("n"));
19         PrintWriter out=res.getWriter();
20         out.println("<H1> Cmmdc = "+cmmdc(m,n)+"</H1>");
21     }

23     public void doPost(HttpServletRequest req, HttpServletResponse res)
24         throws ServletException, IOException{
25         doGet(req, res);
26     }
27 }

```

### 8.5.3 Sesiune de lucru

În cazul protocolului HTTP, de fiecare dată când un client deschide sau revine la o pagină Web se deschide o nouă conexiune cu serverul Web iar acesta nu reține informațiile referitoare la client pe perioada conexiunii respective. Perioada de timp cât un client este în conexiune cu o pagină Web se numește sesiune.

Există posibilitatea păstrării unor informații pe durata unei sesiuni prin intermediul unui obiect de tip `javax.servlet.http.HttpSession`.

Înainte satisfacerii unei cereri, servlet-ul verifică existența unui obiect `HttpSession`. Acest obiect se crează la prima apelare de către un client a servlet-ului prin

```
HttpSession sesiune=request.getSession(true);
```

Un obiect `HttpSession` poate reține atribute, adică perechi de forma (nume, valoare). Introducerea unui atribut se realizează prin

```
void setAttribute(String nume, Object valoare)
```

iar extragerea valorii unui atribut se obține prin

`Object getAttribute(String nume)`

Metoda `String nume[] getValueNames()` returnează numele tuturor atributelor definite.

Un atribut se elimină cu metoda `void removeAttribute(String nume)`.

**Exemplul 8.5.4** *Exemplul următor numără de câte ori se apelează servlet-ul într-o sesiune. Se definește un atribut `noAcces`, care la prima apelare este inițializat iar apoi este mărit cu câte o unitate la fiecare nouă apelare a servlet-ului.*

```

1 import java.io.IOException;
2 import java.io.PrintWriter;
3 import javax.servlet.ServletException;
4 import javax.servlet.http.HttpServlet;
5 import javax.servlet.http.HttpServletRequest;
6 import javax.servlet.http.HttpServletResponse;
7 import javax.servlet.http.HttpSession;
8 import javax.servlet.annotation.WebServlet;

10 @WebServlet(urlPatterns = "/sesiune")
11 public class Sesiune extends HttpServlet{

13     public void doGet(HttpServletRequest req, HttpServletResponse res)
14         throws ServletException, IOException{
15         res.setContentType("text/html");
16         String mesaj;
17         PrintWriter out=res.getWriter();
18         HttpSession session=req.getSession(true);
19         Integer contor=(Integer)session.getAttribute("noAcces");
20         if(contor==null){
21             contor=new Integer(1);
22             mesaj="Salut !";
23         }
24         else{
25             contor=new Integer(contor.intValue()+1);
26             mesaj="Bine ati revenit !";
27         }
28         session.setAttribute("noAcces", contor);
29         out.println("<html><body>");
30         out.println("<h1>"+mesaj+"</h1>");
31         out.println("Numarul de accesari al acestei pagini este " +
32             contor.intValue());
33         out.println("</body></html>");
34         out.close();
35     }

37     public void doPost(HttpServletRequest req, HttpServletResponse res)
38         throws ServletException, IOException{
39         doGet(req, res);
40     }
41 }

```

Apelarea servlet-ului se face din

```

1 <!doctype html>
2 <head>
3   <meta charset="utf-8">
4   <link rel="stylesheet" href="mycss.css">
5 </head>
6 <body>
7   <center>
8     <h1> Formular de acces </h1>
9     <form method="get"
10       action="/myservlet/sesiune">
11       <p>
12         <input type="submit" value="Acceseaza">
13       </form>
14     </center>
15 </body>
16 </html>

```

### 8.5.4 Cookie

Un *Cookie* este un fișier de dimensiune mică trimis de către programul server clientului ca parte a header-ului Http.

Acesta conține informații despre sesiunea curentă care salvează pe disc vor putea fi accesate în sesiuni ulterioare. Când un navigator emite o cerere către un server, cookie-urile anterioare primite de către client de la serverul respectiv sunt trimise din nou serverului ca parte a cererii formulată de client.

Cookie-urile sunt șterse automat în momentul expirării.

Unii clienți nu permit memorarea cookie-urilor. În acest caz, clientul este informat că acest fapt ar putea duce la imposibilitatea satisfacerii cererii sale / accesării paginii Web. Implicit, durata de viață a unui cookie este sesiunea curentă a navigatorului (până se închide navigatorul).

#### Clasa Cookie

##### Constructor

`Cookie(String nume, String valoare)`

##### Metode

- `void setDomain(String model)`

Domeniul este o adresă URL ce restricționează accesul cookie-urilor la acel domeniu. *model* trebuie să conțină cel puțin două caractere ".".

- `void setMaxAge(int durată)` Fixează durata de existență a cookie-ului - în secunde. Valoarea implicită este -1, adică cookie-ul există până la închiderea programului navigator.

- `void setComment(String comentariu)`
- `void setSecure(boolean flag)`

Valoarea implicită este `false`.

Trimiterea unui cookie clientului:

```
public void HttpServletResponse.addCookie(Cookie cookie)
```

Recunoașterea cookie-urilor de către servlet:

```
Cookie [ ] cookies=request.getCookies();
if(cookies!=null){
    for(int i=0;i<cookies.length;i++){
        String name=cookies[i].getName();
        String valoare=cookies[i].getValue();
        . . .
    }
}
```

**Exemplul 8.5.5** În exemplul următor se numără de câte ori se apelează servlet-ul pe durata de viață a cookie-ului.

```
1 package cookie;
2 import java.io.IOException;
3 import java.io.PrintWriter;
4 import javax.servlet.ServletException;
5 import javax.servlet.http.HttpServlet;
6 import javax.servlet.http.HttpServletRequest;
7 import javax.servlet.http.HttpServletResponse;
8 import javax.servlet.http.Cookie;
9 import javax.servlet.annotation.WebServlet;

11 @WebServlet(urlPatterns = "/cookie")

13 public class Apelari extends HttpServlet{

15     public void doGet(HttpServletRequest req,
16                       HttpServletResponse res)
17         throws ServletException, IOException{
18         res.setContentType("text/html");
19         String mesaj="";
20         PrintWriter out=res.getWriter();
21         Cookie myCookie=null;
22         int contor=0;
23         Cookie [ ] cookies=req.getCookies();
24         boolean sw=false;
25         if(cookies!=null){
26             for(int i=0;i<cookies.length;i++){
27                 String name=cookies[i].getName();
28                 if(name.equals("urmarire")){
```

```

29         sw=true;
30         contor=Integer.parseInt(cookies[i].getValue());
31         contor++;
32         mesaj="Bine ati revenit !";
33         System.out.println("Gasit "+contor);
34     }
35 }
36 }
37 if(sw){
38     myCookie=new Cookie("urmarire",(new Integer(contor)).toString());
39 }
40 else{
41     myCookie=new Cookie("urmarire", (new Integer(1)).toString());
42     contor=1;
43     mesaj="Salut !";
44 }
45 myCookie.setMaxAge(1000000);
46 res.addCookie(myCookie);
47 out.println("<html><body>");
48 out.println("<h1>"+mesaj+"</h1>");
49 out.println("Numarul de accesari al acestei pagini este "+contor);
50 out.println("</body></html>");
51 out.close();
52 }
53
54 public void doPost(HttpServletRequest req,
55                     HttpServletResponse res)
56     throws ServletException, IOException{
57     doGet(req, res);
58 }
59 }

```

### 8.5.5 Gestiunea butoanelor - *TimerServlet*

Într-o pagină html de apelare a unui servlet pot fi mai multe butoane

```
<input type="submit" value="text" name="numeBtn"/>
```

Ultimul atribut permite determinarea butonului accesat.

Șablonul de programare este

```

String button="";
for (Enumeration<String> e = req.getParameterNames(); e.hasMoreElements();){
    button=e.nextElement();
    switch(button){
        case "numeBtn1":
            . . .
            break;
        case "numeBtn2":
            . . .
            break;
        . . .
    }
}
}

```

**Exemplul 8.5.6** *Un servlet va executa sarcini la intervale de  $n$  (dat ca parametru) secunde. Câte un buton permite declanșarea, oprirea și aflarea stării curente.*

Programarea unor activități care se vor repeta se poate realiza utilizând clasele `java.util.Timer` și `java.util.TimerTask`.

Clasa `java.util.Timer`

### Constructori

- `public Timer()`

### Metode

- `public void schedule(TimerTask task, long delay, long period)`
- `public void cancel()`

Sarcina de executat se definește într-o clasă ce extinde `java.util.TimerTask`. Programatorul trebuie să realizeze metoda `void run()`.

În acest exemplu, sarcina constă în reținerea momentului (ora : minut : secundă).

```

1 package timer;
2 import java.io.IOException;
3 import javax.servlet.ServletException;
4 import javax.servlet.http.HttpServlet;
5 import javax.servlet.http.HttpServletRequest;
6 import javax.servlet.http.HttpServletResponse;
7 import java.io.PrintWriter;
8 import javax.servlet.annotation.WebServlet;
9 import java.util.Enumeration;

11 @WebServlet(urlPatterns = "/timer")

13 public class TimerServlet extends HttpServlet {
14     private static MyTimer myTimer;

16     public void doGet(HttpServletRequest req, HttpServletResponse res)
17         throws ServletException, IOException {
18         PrintWriter out=res.getWriter();
19         res.setContentType("text/html");
20         String button="";
21         for (Enumeration<String>e=req.getParameterNames(); e.hasMoreElements(); ) {
22             button=e.nextElement();
23             switch (button) {
24                 case "start":
25                     String ss=req.getParameter("seconds");
26                     int seconds=Integer.parseInt(ss);
27                     StringBuffer sb=new StringBuffer(1000);
28                     myTimer=new MyTimer(seconds, out, sb);
29                     break;

```



```

30         case "stop":
31             myTimer.timerStopped();
32             break;
33         case "state":
34             out.println("<html>");
35             out.println("<body>");
36             out.println("<h1>TimerServlet</h1>");
37             out.println("<p/>");
38             out.println(myTimer.getSb());
39             out.println("<p/>");
40             out.println(" <a href=\"index.html\">Start Page</a>");
41             out.println("</body>");
42             out.println("</html>");
43             break;
44     }
45 }
46 out.close();
47 }

49 public void doPost(HttpServletRequest req, HttpServletResponse res)
50     throws ServletException, IOException{
51     doGet(req, res);
52 }
53 }

```

unde clasa *MyTimer* are codul

```

1 package timer;
2 import java.util.Timer;
3 import java.util.TimerTask;
4 import java.io.PrintWriter;
5 import java.util.Calendar;

7 public class MyTimer{
8     private Timer timer;
9     private PrintWriter out;
10    private StringBuffer sb;

12    public MyTimer(int seconds, PrintWriter out, StringBuffer sb){
13        this.out=out;
14        this.sb=sb;
15        timer=new Timer();
16        long ms=1000;
17        timer.scheduleAtFixedRate(new MyTask(),0*ms,seconds*ms);
18        Calendar calendar=Calendar.getInstance();
19        String time = calendar.get(Calendar.HOUR_OF_DAY)+":"+
20            calendar.get(Calendar.MINUTE)+
21            ":"+calendar.get(Calendar.SECOND);
22        System.out.println("Timer is started : "+time);
23        sb.append("<br/>");
24        sb.append("Timer is started : "+time);
25        out.println("<html>");
26        out.println("<body>");
27        out.println("<h1>TimerServlet</h1>");
28        out.println("<p/>");
29        out.println(sb.toString());
30        out.println("<p/>");
31        out.println(" <a href=\"index.html\">Start Page</a>");
32        out.println("</body>");

```

```

33     out.println("</html>");
34 }

36 public void timerStopped(){
37     timer.cancel();
38     Calendar calendar=Calendar.getInstance();
39     String time = calendar.get(Calendar.HOUR_OF_DAY)+":"+
40         calendar.get(Calendar.MINUTE)+
41         ":"+calendar.get(Calendar.SECOND);
42     System.out.println("Timer is stopped : "+time);
43     sb.append("<br/>");
44     sb.append("Timer is stopped : "+time);
45     out.println("<html>");
46     out.println("<body>");
47     out.println("<h1>TimerServlet</h1>");
48     out.println("<p/>");
49     out.println(sb.toString());
50     out.println("<p/>");
51     out.println(" <a href=\"index.html\">Start Page</a>");
52     out.println("</body>");
53     out.println("</html>");
54 }

56 public String getSb(){
57     return sb.toString();
58 }

60 class MyTask extends TimerTask {
61     @Override
62     public void run() {
63         Calendar calendar=Calendar.getInstance();
64         String time = calendar.get(Calendar.HOUR_OF_DAY)+":"+
65             calendar.get(Calendar.MINUTE)+
66             ":"+calendar.get(Calendar.SECOND);
67         sb.append("<br/>");
68         sb.append("Current time : "+time);
69     }
70 }
71 }

```

Apelarea - clientul Web - este

```

1 <!doctype html>
2 <body>
3     <center>
4     <h1> Timer </h1>
5     <form method="post"
6         action="/timer/timer">
7         <table>
8             <tr>
9                 <td><label>Durata (secunde)</label></td>
10                <td>
11                    <input type="number" name="seconds" size="5" >
12                </td>
13                <td>
14                    <input type="submit" value="Start timer" name="start"/>
15                </td>
16                <td>
17                    <input type="submit" value="Current state" name="state"/>

```

```

18         </td>
19         <td>
20             <input type="submit" value="Stop timer" name="stop" />
21         </td>
22     </tr>
23 </table>
24 </form>
25 </center>
26 </body>
27 </html>

```

### 8.5.6 Autentificare

Serverul Web *apache-tomcat* poate executa autentificările *basic* și *digest*. Autentificarea se face pe baza perechii (nume\_utilizator, parola) (username,password). Aceste informații sunt reținute în serverul Web, în fișierul `conf\tomcat-users.xml` fiind asociate unui element `<role>`, de exemplu

```

<role rolename="BASIC_ROLE"/>
<role rolename="DIGEST_ROLE"/>
. . .
<user username="basic" password="basic" roles="BASIC_ROLE"/>
<user username="digest" password="digest" roles="DIGEST_ROLE"/>

```

Codul servlet-ului nu este implicat, iar cerința de autentificare solicitată unui client se precizează în fișierul `web.xml`

```

<security-role>
    <role-name>BASIC_ROLE</role-name>
</security-role>

<security-constraint>
    <web-resource-collection>
        <web-resource-name>Restricted Access - Members Only</web-resource-name>
        <url-pattern>/cmmdc</url-pattern>
        <http-method>GET</http-method>
        <http-method>POST</http-method>
    </web-resource-collection>
    <auth-constraint>
        <role-name>BASIC_ROLE</role-name>
    </auth-constraint>
    <user-data-constraint>
        <transport-guarantee>NONE</transport-guarantee>
    </user-data-constraint>
</security-constraint>

<login-config>
    <auth-method>BASIC</auth-method>
    <!--
    <realm-name>My realm name</realm-name>
    -->
</login-config>

```

Opțional poate fi definit un domeniu (realm).

În cazul apelării dintr-un program Java a unui servlet, la care accesul presupune autentificare, aceasta se realizează prin intermediul claselor pachetului *httpcomponents-client*:

```
import org.apache.http.auth.AuthScope;
import org.apache.http.auth.UsernamePasswordCredentials;
import org.apache.http.impl.auth.BasicScheme;
import org.apache.http.protocol.BasicHttpContext;

. . .
DefaultHttpClient httpclient = new DefaultHttpClient();
// Activitati pentru autentificare
httpclient.getCredentialsProvider().setCredentials(
    new AuthScope(host,Integer.parseInt(port)),
    // new AuthScope(host,Integer.parseInt(port),realm),
    new UsernamePasswordCredentials(username,password)
);
BasicHttpContext context=new BasicHttpContext();
BasicScheme schema = new BasicScheme();
// DigestScheme schema = new DigestScheme();
context.setAttribute("preemptive-auth", schema);
. . .
HttpResponse response=httpclient.execute(httppost,context);
. . .
```

În cazul autentificării *digest*, datele de identificare sunt criptate iar in cazul autentificării *basic* se utilizează codarea *base64*.

### 8.5.7 Servlet cu conexiune la o bază de date

Folosind Anexa **H** considerăm

**Exemplul 8.5.7** *Consultarea unei agende de adrese e-mail. Se utilizează o bază de date AgendaEMail alcătuită dintr-un singur tabel adrese (id int, nume varchar(20), email varchar(30)).*

Utilizând SGBD *derby* servlet-ul este

```
1 import java.io.IOException;
2 import javax.servlet.ServletException;
3 import javax.servlet.ServletOutputStream;
4 import javax.servlet.ServletConfig;
5 import javax.servlet.http.HttpServlet;
6 import javax.servlet.http.HttpServletRequest;
7 import javax.servlet.http.HttpServletResponse;
8 import javax.servlet.annotation.WebServlet;
9 import java.sql.Statement;
10 import java.sql.Connection;
11 import java.sql.DriverManager;
12 import java.sql.SQLException;
13 import java.sql.ResultSet;
```

```

15 @WebServlet(urlPatterns = "/adrese")
16 public class AgendaEMailServlet extends HttpServlet {
17     Statement instructiune=null;
18     Connection con=null;

20     public void init(ServletConfig config) throws ServletException{
21         super.init(config);
22         // SGBD Derby
23         String jdbcDriver="org.apache.derby.jdbc.ClientDriver";
24         String URLBazaDate="jdbc:derby://localhost:1527/AgendaEMail";
25         // SGBD Mysql
26         //String jdbcDriver="com.mysql.jdbc.Driver";
27         //String URLBazaDate="jdbc:mysql://localhost:3306/AgendaEMail?user=root";

29         try{
30             Class.forName(jdbcDriver).newInstance();
31             con=DriverManager.getConnection(URLBazaDate);
32             instructiune=con.createStatement();
33         }
34         catch(ClassNotFoundException e){
35             System.out.println("Driver inexistent JDBC: "+jdbcDriver);
36         }
37         catch(SQLException e){
38             System.out.println("Baza de date inexistentă "+URLBazaDate);
39         }
40         catch(Exception e){
41             System.out.println("Eroare : "+e.getMessage());
42         }
43     }

45     public void destroy(){
46         try{
47             if(con!=null) con.close();
48         }
49         catch(SQLException e){
50             System.out.println(e.getMessage());
51         }
52     }

54     public void doGet(HttpServletRequest req, HttpServletResponse res)
55         throws ServletException, IOException{
56         String myAtribut,myVal;
57         res.setContentType("text/html");
58         ServletOutputStream out = res.getOutputStream();

60         myAtribut=req.getParameter("criteriu");
61         myVal=req.getParameter("termen");
62         myVal='\''+myVal+'\'';
63         try{
64             String sql="select * from adrese where "+myAtribut+" = "+myVal;
65             ResultSet rs=instructiune.executeQuery(sql);
66             out.println("<html>");
67             out.println("<head><title>AgendaEMail</title></head>");
68             out.println("<body>");
69             out.println("<h1>Agenda de Adrese e-mail </h1>");
70             out.println("<p>");
71             out.println("<b>      Nume      <—>      Adresa e-mail </b>");
72             out.println("<br/>");
73             while(rs.next()){

```

```

74         out.print(rs.getString("nume")+ " <—> "+rs.getString("email"));
75         out.println("<br/>");
76     }
77     out.println("</body>");
78     out.println("</html>");
79     out.close();
80 }
81 catch(SQLException e){
82     System.out.println("SQLException: "+e.getMessage());
83 }
84 catch(Exception e){
85     System.out.println("Eroare : "+e.getMessage());
86 }
87 }

89 public void doPost(HttpServletRequest req, HttpServletResponse res)
90     throws ServletException, IOException{
91     doGet(req, res);
92 }
93 }

```

Apelarea servlet-ului se face din

```

1 <html>
2 <body>
3   <h1> Cautare in baza de date AgendaEMail</h1>
4   <form method="get"
5       action="/agendae/adrese">
6       <p>Criteriu de cautare:
7       <select name="criteriu" >
8         <option value="nume">dupa Nume
9         <option value="email">dupa Email
10      </select>
11      <br>
12      <p>Entitatea cautata
13      <input type="text" name="termen" size=30>
14      <p><input type="submit" value="Cauta">
15  </form>
16 </body>
17 </html>

```

Testarea aplicației presupune:

1. Realizarea bazei de date:
2. Testarea servlet-ului:
  - (a) Pornirea serverului bazei de date.
  - (b) Se verifică prezența în catalogul servlet-ului ...\\WEB-INF\\lib a fișierului `derbyclient.jar` sau `mysql-connector-java-*-bin.jar`.
  - (c) Pornirea serverului *tomcat* sau reîncărcarea servlet-ului.
  - (d) Apelarea servlet-ului din pagina Web.

### 8.5.8 Imagini furnizate de servlet

Indicăm două modalități prin care un client obține o imagine furnizată de un servlet. Imaginea poate proveni dintr-un fișier extern sau poate fi creată de servlet.

- Imaginea este transmisă direct clientului în fluxul de ieșire de tip `ServletOutputStream`, tipul MIME al răspunsului fiind

```
response.setContentType("image/ ext");
```

$ext \in \{gif, jpg, png, \dots\}$ .

Textul sursă al servlet-ului este:

```

1 package graphgif;
2 import java.io.IOException;
3 import javax.servlet.ServletException;
4 import javax.servlet.http.HttpServlet;
5 import javax.servlet.http.HttpServletRequest;
6 import javax.servlet.http.HttpServletResponse;
7 import javax.servlet.ServletOutputStream;
8 import javax.servlet.annotation.WebServlet;

10 import java.nio.file.Path;
11 import java.nio.file.Paths;
12 import java.nio.file.Files;

14 @WebServlet(urlPatterns = "/graphgif")

16 public class MyGraphG extends HttpServlet {
17     public void doGet(HttpServletRequest req, HttpServletResponse res)
18         throws ServletException, IOException {
19         String fs=System.getProperty("file.separator");
20         ServletOutputStream out = res.getOutputStream();
21         /*
22         String pathTomcat = new java.io.File(".").getCanonicalPath();
23         String contextPath=req.getContextPath();
24         Path path=Paths.get(pathTomcat+fs+"webapps"+fs+
25             contextPath+fs+"walking_santa.gif");
26         */
27         String pathApp=
28             req.getSession().getServletContext().getRealPath("/") + fs;
29         Path path=Paths.get(pathApp+"walking_santa.gif");
30         try{
31             res.setContentType("image/gif");
32             Files.copy(path, out);
33         }
34         catch(Exception e){
35             res.setContentType("text/plain");
36             System.out.println(e.getMessage());
37             out.println("Cererea d-voastra nu poate fi satisfacuta");
38         }

```

```

39     out.close();
40 }
41 }

```

- Pe calculatorul serverului Web, imaginea se salvează într-un fișier grafic, după care servlet-ul scrie în fluxul de ieșire un document *html* cu o legătură (link) către fișierul cu imaginea creată anterior. Navigatorul clientului va descărca și vizualiza imaginea.

```

1 package graphjpg;
2 import java.io.IOException;
3 import java.io.PrintWriter;
4 import java.io.File;
5 import javax.servlet.ServletException;
6 import javax.servlet.http.HttpServlet;
7 import javax.servlet.http.HttpServletRequest;
8 import javax.servlet.http.HttpServletResponse;

10 import javax.servlet.annotation.WebServlet;
11 import javax.imageio.ImageIO;
12 import java.awt.Frame;
13 import java.awt.Font;
14 import java.awt.Graphics;
15 import java.awt.image.BufferedImage;

17 @WebServlet(urlPatterns = "/graphjpg")

19 public class MyGraphP extends HttpServlet {

21     public void doGet(HttpServletRequest req, HttpServletResponse res)
22         throws ServletException, IOException {
23         String fs=System.getProperty("file.separator");
24         res.setContentType("text/html");
25         PrintWriter out=res.getWriter();
26         String pathTomcat = new java.io.File(".").getCanonicalPath();
27         String contextPath=req.getContextPath();

29         String fileRef=pathTomcat+fs+"webapps"+fs+contextPath+fs;
30         String numeFis="desen";
31         String ext="jpg"; // sau "png","gif"
32         // Formarea imaginii
33         Frame frame = null;
34         Graphics g = null;
35         BufferedImage image=null;
36         try{
37             frame = new Frame();
38             frame.addNotify();
39             image = (BufferedImage)frame.createImage(800, 60);
40             g = image.getGraphics();
41             // Fixarea fontului
42             g.setFont(new Font("Serif", Font.ITALIC, 48));
43             // Editarea unui text
44             g.drawString("Tehnologii distribuite in Java", 10, 50);

```



```

46      // Salvarea imaginii intr-un fisier jpg sau png
47
48      File f=new File(fileRef+numeFis+"."+ext);
49      ImageIO.write(image,ext,f);
50
51      // Raspunsul catre client
52      out.println("<HTML><BODY>");
53      out.println("<h2>Imagine preluata de pe server </h2>");
54      out.println("<p><a href=\"http://localhost:8080/myservlet/\"+
55          numeFis+\"."+ext+\">");
56      out.println("Vizualizarea imaginii </a>");
57      out.println("</BODY></HTML>");
58      out.close();
59  }
60  finally {
61      // Eliberarea resurselor
62      if (g != null) g.dispose();
63      if (frame != null) frame.removeNotify();
64  }
65 }
66 }

```

### 8.5.9 Servlet cu RMI

Un servlet poate fi client al unei aplicații RMI. Interfața la distanță se depune în catalogul WEB-INF\lib al servlet-ului. Apelarea programului server RMI se face prin

```

import java.rmi.Naming;
. . .
InterfataDistanța obj=(InterfataDistanța)
    Naming.lookup("//"+host+": "+port+"/NumeServiciuRMI");

```

**Exemplul 8.5.8** *Client servlet pentru aplicația RMI de calcul al celui mai mare divizor comun a două numere naturale.*

```

1 import java.io.IOException;
2 import java.io.PrintWriter;
3 import javax.servlet.ServletException;
4 import javax.servlet.http.HttpServlet;
5 import javax.servlet.http.HttpServletRequest;
6 import javax.servlet.http.HttpServletResponse;
7 import javax.servlet.annotation.WebServlet;
8 import java.rmi.Naming;
9 import cmmdc.ICmmdc;
10
11 @WebServlet(urlPatterns = "/servletrmi")
12 public class ServletRMI extends HttpServlet {
13
14     public void doGet(HttpServletRequest req, HttpServletResponse res)
15         throws ServletException, IOException {

```

```

16     res.setContentType("text/html");
17     PrintWriter out = res.getWriter();

19     String sm=req.getParameter("m"),sn=req.getParameter("n");
20     long m=(new Long(sm)).longValue(),n=(new Long(sn)).longValue();
21     long x=0;

23     String host=req.getParameter("host").trim();
24     String sPort=req.getParameter("port");
25     int port=Integer.parseInt(sPort);

27     try{
28         ICmmdc obj=(ICmmdc)Naming.lookup("//"+host+": "+port+"/CmmdcServer");
29         x=obj.cmmdc(m,n);
30     }
31     catch (Exception e) {
32         System.out.println("CmmdcClient exception: "+e.getMessage());
33     }
34     String title="CmmdcServlet";
35     res.setContentType("text/html");
36     out.println("<HTML><HEAD><TITLE>");
37     out.println(title);
38     out.println("</TITLE></HEAD><BODY>");
39     out.println("<H1>"+title+"</H1>");
40     out.println("<P>Cmmdc : "+x);
41     out.println("</BODY></HTML>");
42     out.close();
43 }

45 public void doPost(HttpServletRequest req,HttpServletResponse res)
46     throws ServletException,IOException{
47     doGet(req,res);
48 }
49 }

```

### 8.5.10 Servlet cu JMS

De data aceasta cererea clientului se rezolva asincron:

- In prima fază se apelează un servlet care apelează un server JMS. Soluția este reținută de furnizorul serviciului de mesagerie pe o destinație cu un subiect furnizat de client. Pentru regăsirea rezultatului, servlet-ul crează clientului un abonament durabil, funcție de numele subiectului.
- În faza a doua, clientul apelează un alt servlet, a cărei funcție este preluarea rezultatului. Clientul trebuie sa furnizeze subiectul destinației rezultatului.

#### Exemplul 8.5.9

Serverului JMS al aplicației este dat de clasa *MsgCmmdcServer* prezentat în 5.3.11.

Codul servlet-ului care transmite datele problemei serverului JMS

```

1 package javax.jms;
2 import javax.jms.Topic;
3 import javax.jms.JMSContext;
4 import javax.jms.JMSProducer;
5 import javax.jms.JMSConsumer;
6 import java.io.IOException;
7 import javax.servlet.ServletOutputStream;
8 import javax.servlet.ServletException;
9 import javax.servlet.http.HttpServlet;
10 import javax.servlet.http.HttpServletRequest;
11 import javax.servlet.http.HttpServletResponse;
12 import javax.servlet.annotation.WebServlet;

14 @WebServlet(urlPatterns = "/sender")

16 public class JMSSenderServlet extends HttpServlet{

18     public void doGet(HttpServletRequest req, HttpServletResponse res)
19         throws ServletException, IOException {
20         res.setContentType("text/html");
21         ServletOutputStream out = res.getOutputStream();
22         String m=req.getParameter("m");
23         String n=req.getParameter("n");
24         String topic=req.getParameter("topic");
25         String clientID=req.getParameter("clientID");
26         String clientName=req.getParameter("clientName");
27         String msg=m+" "+n+" "+topic;
28         try{
29             // Varianta Oracle-Sun Message Topic
30             com.sun.messaging.TopicConnectionFactory cf =
31                 new com.sun.messaging.TopicConnectionFactory();
32             //cf.setProperty("imqBrokerHostName","host");
33             //cf.setProperty("imqBrokerHostPort","7676");
34             Topic t=new com.sun.messaging.Topic("Cmmdc");
35             JMSContext ctx=cf.createContext();

37             Topic t1=new com.sun.messaging.Topic(topic);
38             ctx.setClientID(clientID);
39             JMSConsumer consumer = ctx.createDurableConsumer(t1,clientName);

41             JMSProducer producer=ctx.createProducer();
42             producer.send(t,msg);

44             ctx.close();
45             out.println("<html><body bgcolor=\"#ccbbcc\"><center>");
46             out.println("<h1> JSP Cmmdc </h1>");
47             out.println("<p>");
48             out.println("Datele au fost expediate serverului");
49             out.println("</center></body></html>");
50         }
51         catch(Exception e){
52             out.println(e.getMessage());
53         }
54         out.close();
55         System.out.println("Publisher finished");
56     }

58     public void doPost(HttpServletRequest req, HttpServletResponse res)

```

```

59     throws ServletException, IOException {
60     doGet(req, res);
61     }
62 }

```

Codul servlet-ului care preia rezultatul

```

1  package jms;
2  import javax.jms.Topic;
3  import javax.jms.JMSContext;
4  import javax.jms.JMSConsumer;
5  import javax.jms.TextMessage;
6  import java.io.IOException;
7  import javax.servlet.ServletOutputStream;
8  import javax.servlet.ServletException;
9  import javax.servlet.http.HttpServlet;
10 import javax.servlet.http.HttpServletRequest;
11 import javax.servlet.http.HttpServletResponse;
12 import javax.servlet.annotation.WebServlet;

14 @WebServlet(urlPatterns = "/receiver")

16 public class JMSReceiverServlet extends HttpServlet{

18     public void doGet(HttpServletRequest req, HttpServletResponse res)
19         throws ServletException, IOException {
20         res.setContentType("text/html");
21         ServletOutputStream out = res.getOutputStream();
22         String topic=req.getParameter("topic");
23         String clientID=req.getParameter("clientID");
24         String clientName=req.getParameter("clientName");
25         //String clientID="JMScmmdc";
26         //String clientName="JMScmmdc";
27         try{
28             // Varianta Oracle-Sun Message Topic
29             com.sun.messaging.TopicConnectionFactory cf
30                 = new com.sun.messaging.TopicConnectionFactory();
31             //cf.setProperty("imqBrokerHostName","host");
32             //cf.setProperty("imqBrokerHostPort","7676");
33             Topic t=new com.sun.messaging.Topic(topic);
34             JMSContext ctx=cf.createContext();
35             ctx.setClientID(clientID);
36             JMSConsumer consumer = ctx.createDurableConsumer(t,clientName);
37             TextMessage txtMsg=(TextMessage)consumer.receive();
38             String cmmdc=txtMsg.getText();
39             ctx.close();
40             out.println("<html><body bgcolor=\"#ccbbcc\"><center>");
41             out.println("<h1> JSP Cmmdc </h1>");
42             out.println("<p>");
43             out.println("Rezultatul obtinut : "+cmmdc);
44             out.println("</center></body></html>");
45         }
46         catch(Exception e){
47             out.println(e.getMessage());
48         }
49         out.close();
50         System.out.println("Subscriber finished");
51     }

```

```

53 public void doPost(HttpServletRequest req, HttpServletResponse res)
54     throws ServletException, IOException {
55     doGet(req, res);
56 }
57 }

```

În catalogul `lib` al servlet-ului trebuie incluse fișierele *jar* ale serviciului JMS folosit.

### 8.5.11 Servlet cu jurnalizare

**Exemplul 8.5.10** *Servlet cu jurnalizare. Fișierul log se va afla în catalogul aplicației și va fi oferit clientului spre consultare.*

```

1 package logtest;
2 import java.io.IOException;
3 import javax.servlet.ServletException;
4 import javax.servlet.ServletContext;
5 import javax.servlet.http.HttpServlet;
6 import javax.servlet.http.HttpServletRequest;
7 import javax.servlet.http.HttpServletResponse;
8 import javax.servlet.ServletOutputStream;
9 import java.util.logging.Logger;
10 import java.util.logging.FileHandler;
11 import java.util.logging.SimpleFormatter;
12
13 import javax.servlet.annotation.WebServlet;
14
15 @WebServlet(urlPatterns = "/logging")
16
17 public class LoggerServlet extends HttpServlet {
18     private static Logger logger=Logger.getLogger("logtest.LoggerServlet");
19
20     public void init(){
21         try{
22             FileHandler loggingFile = new FileHandler("webapps/logger/results.log");
23             loggingFile.setFormatter(new SimpleFormatter());
24             logger.addHandler(loggingFile);
25         }
26         catch(IOException e){
27             System.out.println(e.getMessage());
28         }
29     }
30
31     public void doGet(HttpServletRequest req, HttpServletResponse res)
32         throws ServletException, java.io.IOException {
33         String fileSep = System.getProperty("file.separator");
34
35         logger.info("INFO : Hello");
36         logger.warning("WARN : Hello");
37         logger.severe("ERROR : Hello");
38
39         res.setContentType("text/html");
40         java.io.PrintWriter out = res.getWriter();
41         out.println("<html><head><title>Servlet logging</title></head><body>");

```

```

42     out.println("<h2>Hello from LoggerServlet</h2>");
43     out.println("<br/>");
44     out.println("<a href=\"http://"+
45         req.getServerName()+":"+
46         req.getLocalPort()+fileSep+
47         "logger"+fileSep+"results.log\">Vizualizati fisierul log</a>");
48     out.println("</body></html>");
49     out.close();
50 }

52 public void doPost(HttpServletRequest req, HttpServletResponse res)
53     throws ServletException, java.io.IOException {
54     doGet(req, res);
55 }
56 }

```

## 8.6 FileUpload

Deseori clientul trebuie să furnizeze unui servlet un volum mare de date, depozitate într-un fișier. Un produs care ne ajută să îndeplinim acest obiectiv este *commons-fileupload* - dezvoltat de *apache*.

*Commons-FileUpload* - dezvoltat în cadrul *apache* - este un produs care simplifică transferul unui fișier de la un client la programul server (file upload). Interfața de programare a produsului se referă la partea de server - în cazul de față reprezentat prin servlet.

Instalarea produsului constă din dezarhivarea fișierului descărcat din Internet.

În plus este nevoie de

- *commons-io*

Fișierele

- *commons-fileupload-\*.jar*
- *commons-io-\*.jar*

se depun în catalogul `lib` al servlet-ului.

Transferarea unui fișier, din partea clientului nu ridică nici o problemă. În fișierul html de apelare, se definește un formular

```

<form
    action=. . .
    enctype="multipart/form-data"
    method="post">

```

iar un fișier de încărcat se fixează prin intermediul marcajului

```
<input type="file" name=. . . size=. . .>
```

Programul navigator afișează o fereastră de căutare, prin care clientul selectează fișierul pe care dorește să-l încarce.

Dacă partea de client este un program, atunci se utilizează *commons-httpclient*, [8.5.1](#)

Programarea încărcării revine la

1. Crearea unei *fabrici* pentru manipularea fișierelor pe disc

```
FileItemFactory factory = new DiskFileItemFactory();
```

2. Crearea unei unelte de încărcare

```
ServletFileUpload upload = new ServletFileUpload(factory);
```

3. Analiza (parsarea) mesajului furnizat de client

```
List fileItems = upload.parseRequest(req);
```

Fiecare element al listei implementează interfața `FileItem`.

Se pot fixa parametrii

- dimensiunea zonei de pe disc destinată datelor de încărcat

```
DiskFileItemFactory factory = new DiskFileItemFactory();  
factory.setSizeThreshold(maxMemorySize);
```

- catalogul temporar de reținere a datelor de încărcat

```
factory.setRepositoryPath(tempDirectory);
```

sau direct

```
DiskFileItemFactory factory = new DiskFileItemFactory(  
    maxMemorySize, tempDirectory);
```

- dimensiunea maximă a unui fișier

```
upload.setSizeMax(maxRequestSize);
```

4. Prelucrarea elementelor încărcate

```
Iterator iter=fileItems.iterator();
while (iter.hasNext()) {
    FileItem item = (FileItem) iter.next();
    if (item.isFormField()) {
        // Prelucrarea elementului item care corespunde unei
        // date din formularul html care nu este de tip fisier
    }
    else{
        // Prelucrarea elementului item de tip fisier
    }
}
```

5. În cazul unui element care nu este de tip fișier putem obține numele și valoarea atributului furnizat de client

```
String name = item.getFieldName();
String value = item.getString();
```

6. În cazul unui fișier putem afla numele câmpului input, numele fișierului, dimensiunea fișierului

```
String fieldName = item.getFieldName();
String fileName = item.getName();
long sizeInBytes = item.getSize();
```

7. Dacă dorim să salvăm fișierul pe calculatorul server atunci prelucrarea este

```
File uploadedFile = new File(...);
item.write(uploadedFile);
```

8. Dacă datele fișierului se încarcă în memoria calculatorului atunci prelucrarea este

```
InputStream in = item.getInputStream();
//preluarea datelor din fluxul in
. . .
in.close();
```

Alternativ, datele se pot reține ca un șir de octeți prin



```
byte[] data = item.get();
```

**Exemplul 8.6.1** *Să se obțină în memoria serverului matricea conținută într-un fișier text. În fișierul text, fiecare linie conține o linie a matricei, iar elementele sunt separate prin spații.*

Metoda *getMatrix* utilizată va reface matricea din datele fișierului.

```

1 package upload;
2 import java.io.IOException;
3 import java.io.InputStream;
4 import java.io.InputStreamReader;
5 import java.io.BufferedReader;
6 import javax.servlet.ServletException;
7 import javax.servlet.http.HttpServlet;
8 import javax.servlet.http.HttpServletRequest;
9 import javax.servlet.http.HttpServletResponse;
10 import javax.servlet.ServletOutputStream;
11 import javax.servlet.annotation.WebServlet;
12 import java.util.List;
13 import java.util.Iterator;
14 import java.util.Vector;
15 import org.apache.commons.fileupload.disk.DiskFileItemFactory;
16 import org.apache.commons.fileupload.servlet.ServletFileUpload;
17 import org.apache.commons.fileupload.FileItemFactory;
18 import org.apache.commons.fileupload.FileItem;

20 @WebServlet(urlPatterns = "/upload")
21 public class FileUploadServlet extends HttpServlet{

23     public void doPost(HttpServletRequest req, HttpServletResponse res)
24     throws ServletException, IOException {
25         res.setContentType("text/plain");
26         ServletOutputStream out = res.getOutputStream();
27         try{
28             FileItemFactory factory = new DiskFileItemFactory();
29             ServletFileUpload upload = new ServletFileUpload(factory);
30             List items = upload.parseRequest(req);
31             upload.setSizeMax(1000000);
32             Iterator iter=items.iterator();
33             while (iter.hasNext()) {
34                 FileItem item = (FileItem) iter.next();
35                 if (!item.isFormField()) {
36                     String fileName = item.getName();
37                     out.println(fileName);
38                     long sizeInBytes = item.getSize();
39                     out.println(sizeInBytes);
40                     InputStream in=item.getInputStream();
41                     InputStreamReader isr=new InputStreamReader(in);
42                     BufferedReader br=new BufferedReader(isr);
43                     double [][] matrix=getMatrix(br);
44                     int m=matrix.length;
45                     int n=matrix[0].length;
46                     for(int i=0;i<m;i++){
47                         for(int j=0;j<n;j++){
48                             out.print(matrix[i][j]+" ");
49                             out.println();

```

```

50         }
51         br.close();
52         isr.close();
53         in.close();
54         out.close();
55     }
56 }
57 }
58 catch(Exception e){
59     System.out.println("Exception: "+e.getMessage());
60 }
61 }

63 private double [][] getMatrix(BufferedReader br) throws Exception{
64     Vector<Double> v=new Vector<Double>(10);
65     double [][] matrix=null;
66     try{
67         String line,s;
68         int m=0,n,mn;
69         do{
70             line=br.readLine();
71             if(line!=null){
72                 m++;
73                 String[] st=line.split(" ");
74                 for(String s:st){
75                     v.addElement(new Double(s));
76                 }
77             }
78         } while(line!=null);
79         if(v.size()>0){
80             mn=v.size();
81             n=mn/m;
82             matrix=new double[m][n];
83             for(int i=0;i<mn;i++){
84                 for(int j=0;j<n;j++){
85                     matrix[i][j]=((Double)v.elementAt(i*n+j)).doubleValue();
86                     System.out.print(matrix[i][j]+" ");
87                 }
88                 System.out.println();
89             }
90         }
91     }
92 }
93 catch(Exception e){
94     throw new Exception(e.getMessage());
95 }
96 return matrix;
97 }
98 }

```

Pentru compilare se completează variabila de sistem `classpath` cu referința către fișierul `commons-fileupload-*.jar`.

Formularul clientului este

```

1 <!doctype html>
2 <body bgcolor="#bbccbb">
3   <h1> Incărcarea unui fișier </h1>
4   <form

```

```

5      action="/upload/upload"
6      enctype="multipart/form-data"
7      method="post"
8      name="linear" onSubmit="return checkIt()">
9  <p>
10 Selecta&#355;i fi&#351;ierul
11 <p>
12 <input type="file" name="myfile" size=30 required>
13 <p>
14 <input type="submit" value="Expediaza fisierul">
15 </form>
16 </body>
17 </html>

```

## 8.7 Descărcarea unui fișier

Considerăm cazul:

**Exemplul 8.7.1** *Fișierul ales de client dintr-o lista disponibilă este descărcat fiind transmis navigatorului.*

```

1 import java.io.IOException;
2 import javax.servlet.ServletException;
3 import javax.servlet.http.HttpServlet;
4 import javax.servlet.http.HttpServletRequest;
5 import javax.servlet.http.HttpServletResponse;
6 import javax.servlet.ServletOutputStream;
7 import javax.servlet.annotation.WebServlet;
8 import java.nio.file.Path;
9 import java.nio.file.Paths;
10 import java.nio.file.Files;
11
12 @WebServlet(urlPatterns = "/download")
13
14 public class DownloadServlet extends HttpServlet {
15     public void doGet(HttpServletRequest req, HttpServletResponse res)
16         throws ServletException, IOException{
17         ServletOutputStream out=res.getOutputStream();
18         String file=req.getParameter("file");
19         System.out.println(file);
20         Path cale=Paths.get("webapps/download/resources/"+file);
21         try{
22             System.out.println(cale+file);
23             res.setContentType("Application/Octet-stream");
24             res.addHeader("Content-Disposition", "attachment; filename="+ file);
25             Files.copy(cale, out);
26         }
27         catch(Exception e){
28             res.setContentType("text/plain");
29             out.println("Cererea d-voastra nu poate fi satisfacuta");
30         }
31         out.close();
32     }

```

```

34 public void doPost(HttpServletRequest req, HttpServletResponse res)
35     throws ServletException, IOException{
36     doGet(req, res);
37 }
38 }

```

Rândul 18 are ca efect păstrarea numelui și a extensiei pentru fișierul selectat a se descărca.

## 8.8 Filtru

Un filtru se aseamănă unui servlet, dar activitatea întreprinsă vizează uzual *contextul*, adică ansamblul servleților care fac parte din aplicația Web.

Din nou tehnica de programare poate fi:

- Descriptiv. Filtrul se declară în fișierul `web.xml` prin

```

<web-app>
    . . .
    <filter>
        <filter-name>nume_filtru</filter-name>
        <filter-class>clasa_filtrului</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>nume_filtru</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
    . . .

</web-app>

```

- Programat. Se utilizează adnotarea `WebFilter`

```

. . .
import javax.servlet.annotation.WebFilter;

@WebFilter(filterName="MyFilterDispatcher",urlPatterns={"/*"})
public class MyFilterDispatcher implements Filter {. . .}

```

Clasa filtrului implementează interfața `Filter`, adică metodele

- `public void init(FilterConfig filterConfig) throws ServletException`
- `public void destroy()`
- `public void doFilter(ServletRequest request, ServletResponse response, FilterChain filterChain) throws IOException, ServletException`

**Exemplul 8.8.1** Contextul filtrud conține doi servleți *HelloServlet* și *CmmdcServlet*, apelabili respectiv din *hello.html*, *cmmddc.html*. Să se programeze un filtru care

- Redirectează solicitarea *“/filtrud/hello”* către *“/cmmddc.html”*.
- Dacă se cere ca natura răspunsului să fie *“text/xml”* atunci invalidează cererea.

Servleții *HelloServlet* și *CmmdcServlet* sunt cei dezvoltati la începutul acestui capitol. Filtrul (în varianta descriptivă) are codul

```

1 import java.io.IOException;
2 import javax.servlet.ServletException;
3 import javax.servlet.ServletRequest;
4 import javax.servlet.ServletResponse;
5 import javax.servlet.RequestDispatcher;
6 import javax.servlet.Filter;
7 import javax.servlet.FilterConfig;
8 import javax.servlet.FilterChain;
9 import javax.servlet.ServletException;
10 import javax.servlet.http.HttpServletRequest;
11 import javax.servlet.http.HttpServletResponse;
12
13 public class MyFilterDispatcher implements Filter {
14     private FilterConfig filterConfig;
15
16     public void init(FilterConfig filterConfig) throws ServletException {
17         this.filterConfig = filterConfig;
18     }
19
20     public void destroy() {
21         this.filterConfig = null;
22     }
23
24     public void doFilter(ServletRequest request, ServletResponse response,
25         FilterChain filterChain) throws IOException, ServletException {
26         HttpServletRequest req = (HttpServletRequest) request;
27         HttpServletResponse res = (HttpServletResponse) response;
28         String uri = req.getRequestURI();
29         System.out.println("Filter URI= " + uri);
30         System.out.println(uri);
31
32         if(uri.equals("/filtrud") || uri.equals("/filtrud/"))
33             filterChain.doFilter(request, response);
34         else{

```

```

34     if(uri.equals("/filtrud/hello")){
35         String dispatcherUri="/cmmdc.html";
36         RequestDispatcher rd=request.getRequestDispatcher(dispatcherUri);
37         rd.forward(request,response);
38     }
39     else{
40         if(uri.equals("/filtrud/cmmdc")){
41             String tip=req.getParameter("tip");
42             if(tip.equals("text/xml")){
43                 res.sendError(HttpServletResponse.SC.FORBIDDEN);
44             }
45         }
46         filterChain.doFilter(request,response);
47     }
48 }
49 }
50 }

```

Fișierul *web.xml* este

```

1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <!DOCTYPE web-app
3     PUBLIC "-//Sun Microsystems, Inc./DTD Web Application 2.3//EN"
4     "http://java.sun.com/dtd/web-app_2_3.dtd">
5
6 <web-app>
7     <filter>
8         <filter-name>filterDispatcher</filter-name>
9         <filter-class>MyFilterDispatcher</filter-class>
10    </filter>
11
12    <filter-mapping>
13        <filter-name>filterDispatcher</filter-name>
14        <url-pattern>*/</url-pattern>
15    </filter-mapping>
16 </web-app>

```

Pentru înțelegerea aplicației prezentăm și codul fișierului *cmmdc.html*

```

1 <html>
2 <body bgcolor="#ccbbcc">
3     <center>
4         <h1> Cmmdc Servlet </h1>
5         <form method="get"
6             action="/myservlet/cmmdc">
7             <table border="1">
8                 <tr>
9                     <td> Primul numar </td>
10                    <td> <input type="text" name="m" size=10> </td>
11                </tr>
12                <tr>
13                    <td> Al doilea numar </td>
14                    <td> <input type="text" name="n" size=10> </td>
15                </tr>
16                <tr>
17                    <td> Natura raspunsului </td>
18                    <td> <select name="tip" >
19                        <option value="text/html"> text/html
20                        <option value="text/plain"> text/plain

```

```

21         <option value="text/xml"> text/xml
22     </select> </td>
23 </tr>
24 <tr>
25     <td> <input type="submit" value=" Calculeaza"> </td>
26     <td></td>
27 </tr>
28 </table>
29 </form>
30 </center>
31 </body>
32 </html>

```

## 8.9 Eveniment și auditor

Tehnologia Servlet permite urmărirea și intervenția de către serverul Web în:

- ciclul de viață al unui servlet, prin interfața `javax.servlet.ServletContextListener`
- evoluția obiectului `javax.servlet.http.HttpSession` prin interfața `javax.servlet.http.HttpSessionListener`

Interfața `ServletContextListener` declară metodele

- `void contextInitialized(ServletContextEvent sec)`
- `void contextDestroyed(ServletContextEvent sec)`

Interfața `HttpSessionListener` declară metodele

- `void sessionCreated(HttpSessionEvent hse)`
- `void sessionDestroyed(HttpSessionEvent hse)`

Clasa care implementează una din aceste interfețe se declară în fișierul `web.xml` printr-un element `<listener>`

```

<listener>
    <listener-class> clasa_listener </listener-class>
</listener>

```

**Exemplul 8.9.1** Auditor care sesizează încărcarea și dispariția unui servlet afișând în fereastra DOS a serverului Web un mesaj.

```

1 import javax.servlet.ServletContextListener;
2 import javax.servlet.ServletContextEvent;
3 import javax.servlet.ServletContext;

5 public class FirstContextListener
6     implements ServletContextListener {
7     public void contextDestroyed(ServletContextEvent event) {
8         System.out.println("Web app was removed.");
9     }
10    public void contextInitialized(ServletContextEvent event) {
11        System.out.println("Web app is ready.");
12        ServletContext sc=event.getServletContext();
13        System.out.println(sc.getContextPath());
14        System.out.println(sc.getEffectiveMajorVersion());
15        System.out.println(sc.getEffectiveMinorVersion());
16    }
17 }

```

### Exemplul 8.9.2

```

1 import javax.servlet.http.HttpSessionListener;
2 import javax.servlet.http.HttpSessionEvent;

4 public class FirstSessionListener implements HttpSessionListener {
5     static int users = 0;

7     public void sessionCreated(HttpSessionEvent e) {
8         users++;
9     }
10    public void sessionDestroyed(HttpSessionEvent e) {
11        users--;
12    }
13    public static int getConcurrentUsers() {
14        return users;
15    }
16 }

```

## 8.10 Server *apache-tomcat* încorporat

Într-o clasă Java se poate încorpora un server *tomcat* în care pot fi instalate una sau mai mulți serveți. Resursele necesare sunt conținute în *apache-tomcat\*-embedded*, iar fișierele *jar* conținute trebuie declarate în variabila de sistem `classpath`.

Șablonul de programare este:

```

1 import org.apache.catalina.startup.Tomcat;
2 import org.apache.catalina.Context;
3 import java.io.File;

5 public class EmbeddedTomcat{

```



```
6  public static void main(String[] args) {
7      try {
8          Tomcat tomcat = new Tomcat();
9          tomcat.setBaseDir(".");
10         tomcat.setPort(9090);           // portul serverului

12         File docBase = new File(".");
13         Context ctxt = tomcat.addContext("/", docBase.getAbsolutePath());

15         Tomcat.addServlet(ctxt, "numeServlet", new ClasaServlet());
16         ctxt.addServletMapping("/numeApel", "numeServlet");

18         tomcat.start();
19         tomcat.getServer().await();
20     }
21     catch (Exception e) {
22         e.printStackTrace();
23     }
24 }
25 }
```

### Probleme

- Servlet API 3.1 : Upgrade prin tomcat

## Întrebări recapitulative

1. Precizați sensurile și conținutul cuvântului *servlet*.
2. Unde se instalează o aplicație servlet în serverul Web *apache-tomcat*?
3. Cum se poate apela aplicația servlet, dar servlet-ul propriu zis?
4. Extinzând clasa *HttpServlet*, ce trebuie să facă programatorul?
5. Care sunt modurile de programare a unui servlet și precizați diferența dintre ele.
6. Care sunt sarcinile de indeplinit în metoda *doGet*?
7. Cum se rezolvă solicitatarea clientului într-un servlet asincron conform Servlet-API 3.0 ?
8. Cum se rezolvă solicitatarea clientului într-un servlet asincron conform Servlet-API 3.1 ?
9. Ce posibilitate de prelucrare oferă un filtru?

10. Ce posibilitate oferă clasa `javax.servlet.http.Cookie`?
11. Ce posibilitate oferă clasa `javax.servlet.http.HttpSession`?
12. Care sunt metodele de programare asincronă a unui servlet?

# Capitolul 9

## Java Server Page – JSP

### 9.1 Tehnologia JSP

Tipul tehnologiei JSP este denumit *procesare de șabloane (template engine)*. JSP este o tehnologie similară cu PHP, ASP.NET, apache-velocity, etc. JSP permite includerea de cod Java într-un document `html`. Un asemenea document se depozitează într-un server Web, container de servlet, cu extensia `jsp`, eventual `jspx`.

Apelarea documentului JSP se realizează prin

- meniul File/Open a unui navigator, cu

`http://host:port/cale/doc.jsp`

- referință html

`<a href="http://host:port/cale/doc.jsp">`

- valoare a atributului `action` într-un marcaj `form`

`<form action="http://host:port/cale/doc.jsp" ... >`

Prin *cale* se înțelege calea de la catalogul `webapps` până la catalogul ce conține fișierul `jsp`.

Vom depozita fișierele JSP într-un catalog *jsp* din arborele

```
webapps
|--> JSPApp
|   |--> WEB-INF
|       |--> classes
|           web.xml
```

```
|      |--> jsp
|      |      |      doc.jsp
```

caz în care *cale=JSPApp/jsp*.

Astfel, schimbând numele fișierului *Hello.html*

```
1 <html>
2   <body>
3     Hello
4   </body>
5 </html>
```

în *Hello.jsp* și plasându-l în catalogul *jsp* se obține același efect, dar prelucrarea paginilor / documentelor este diferită. Fișierul html este prelucrat doar de programul navigator și poate fi deschis ca fișier, în timp ce fișierul JSP este prelucrat de serverul Web cu afișarea prin intermediul navigatorului. Prelucrarea efectuată de serverul Web constă din transformarea paginii / documentului JSP într-un servlet, care este compilat și lansat în execuție. Din aceeașă cauză prima invocare a paginii / documentului JSP durează mai mult decât apelările ulterioare.

Fișierul JSP poate fi construit pe un document *xhtml*, având preambulul

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

sau document *html* 5. În acest caz preambulul este

```
<!doctype html>
```

Apelarea paginii / documentului JSP se face prin *jsp/Hello.jsp*

Există două moduri de a include elemente JSP într-un text html:

- prin elemente specifice JSP.

Fișierul are extensia *jsp* și se numește *pagină JSP*.

- prin elemente xml aparținând spațiului de nume

<http://java.sun.com/JSP/Page>

Fișierul poate avea extensia *jsp* sau *jspx* și se numește *document JSP*.

Comentariile JSP se scriu de forma

```
<%-- Comentariu --%>
```

Considerăm următorul exemplu introductiv:

**Exemplul 9.1.1** *Putem afișa valoarea unei variabile Java (de exemplu data calendaristică) prin*

- Varianta paginii JSP

```

1 <html>
2   <body>
3     <p>
4       Data calendaristica 1:
5       <%= new java.util.Date() %>
6
7     <p>
8       <% java.util.Date data1=new java.util.Date(); %>
9       Data calendaristica 2:
10      <%= data1 %>
11
12    <p>
13      <% java.util.Date data2=new java.util.Date(); %>
14      Data calendaristica 3:
15      <% out.print(data2); %>
16    </body>
17 </html>

```

Dacă se utilizează operatorul de afișare = atunci după expresia de afișat nu se pune ;.

Variabila predefinită `out` este de tip `javax.servlet.jsp.JspWriter`.

- Varianta documentului JSP

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <html xmlns:jsp="http://java.sun.com/JSP/Page">
3   <body>
4     <jsp:scriptlet>
5       java.util.Date d=new java.util.Date();
6     </jsp:scriptlet>
7     <jsp:expression> d </jsp:expression>
8   </body>
9 </html>

```

Codul Java înglobat într-un text html se numește *scriptlet*. Sintaxa utilizată este

- Varianta paginii JSP

`<% cod Java %>`

- Varianta documentului JSP

```
<jsp:scriptlet> codJava </jsp:scriptlet>
```

**Domeniul de valabilitate.** Domeniul de valabilitate definește intervalul de timp, de existență al unui obiect, fiind definit prin valorile:

Valoare	Domeniu de valabilitate
page request	pagina curentă în pagina curentă, în paginile incluse și în paginile către care se face o redirectare
session	în sesiunea curentă
application	pe durata rulării aplicației

În orice pagină / document JSP sunt predefinite obiectele:

Variabila	Tip/Clasa
out	javax.servlet.jsp.JspWriter
request	javax.servlet.ServletRequest
response	javax.servlet.ServletResponse
session	javax.servlet.http.HttpSession
page	java.lang.Object, this
pageContext	javax.servlet.jsp.PageContext
application	javax.servlet.ServletContext ServletContext.getServletConfig().getContext()
exception	java.lang.Throwable

Astfel

```
String request.getParameter(String numeParametru)
```

furnizează valoarea parametrului *numeParametru* dintr-un formular html.

**Exemplul 9.1.2** *Pagina JSP Hello: Clientul transmite numele paginii care îi răspunde cu mesajul de salut "Hi " + nume + " !".*

Codul paginii JPS (*hello.jsp*) este

```

1 <html>
2   <head>
3     <title> jsphello </title>
4   </head>
5   <body>
```

```

6      <center>
7      <h1> Pagina de r&#259;spuns </h1>
8      <p>
9      <%
10     out.println("Hi "+request.getParameter("name")+" !" );
11     %>
12     </center>
13 </body>
14 </html>

```

apelat din (*index.html*)

```

1 <html>
2 <head>
3   <title> JSP Hello </title>
4 </head>
5 <body bgcolor="#bbeebb">
6   <center>
7     <h1> Pagina de apelare JSP </h1>
8     <form method="post"
9       action="jsp/hello.jsp">
10      <p> Numele:
11      <input type="text" name="name" size=20>
12      <p>
13      <input type="submit">
14    </form>
15  </center>
16 </body>
17 </html>

```

Compilarea și arhivarea servlet-ului o vom realiza prin intermediul lui *apache-ant*. În acest scop se crează structura:

```

jsphello
|      |--> src
|      |--> web
|      |      |      |--> jsp
|      |      |      |      hello.jsp
|      |      |--> WEB-INF
|      |      |      |--> classes
|      |      |      |--> lib
|      |      |      |      web.xml
|      |      |      |      index.html

```

Fișierul *build.xml* este simplu

```

1 <project basedir="." default="generate.war">
2   <property name="dist.name" value="JSPApp" />
3   <property name="dist.dir" value="dist" />
4
5   <path id="myclasspath">
6     <fileset dir="web/WEB-INF/lib">
7       <include name="*.jar" />
8     </fileset>
9   </path>
10
11  <target name="init">
12    <delete dir="${dist.dir}" />

```

```

13     <delete dir="web/WEB-INF/classes" />
14     <mkdir dir="web/WEB-INF/classes" />
15     <mkdir dir="${dist.dir}" />
16 </target>

18 <target name="compile" depends="init">
19     <javac classpathref="myclasspath"
20           srcdir="src"
21           destdir="web/WEB-INF/classes"
22           includeantruntime="false" />
23 </target>

25 <target name="generate.war" depends="compile">
26     <jar destfile="${dist.dir}/${dist.name}.war" basedir="web" />
27 </target>
28 </project>

```

### 9.1.1 Declarații JSP

Printr-o *declarație JSP*, putem defini câmpuri(variabale) și metode Java ce pot fi apoi folosite, respectiv apelate în documentul respectiv. O declarație JSP se definește printr-un marcaj

<%! . . . %>

sau, în format XML

<jsp:declaration> . . . </jsp:declaration>

**Exemplul 9.1.3** *Calculul celui mai mare divizor comun a două numere naturale cu metoda de calcul este definită într-o declarație JSP.*

Pagina JSP a aplicației *cmmdc.jsp*:

```

1 <html>
2 <body>
3   <H1> CMMDC </H1>
4   <%!
5       long cmmdc(long m,long n){. . .}
6   %>
7   Rezultatul este
8   <%
9       String sm=request.getParameter("m");
10      String sn=request.getParameter("n");
11      long m=Long.parseLong(sm),n=Long.parseLong(sn);
12      out.println(cmmdc(m,n));
13   %>
14 </body>
15 </html>

```

apelat din documentul *cmmdc.html*



```

1 <!doctype html>
2 <head>
3   <meta charset="utf-8">
4 </head>
5 <body bgcolor="#bbccbb">
6   <center>
7     <h1> Pagina de apelare CmmdcServlet </h1>
8     <form method="get"
9       action="jsp/cmmdc.jsp">
10      <table>
11        <tr>
12          <td><label> Primul numar </label></td>
13          <td>
14            <input type="number" name="m" size="5"
15              required min="1">
16          </td>
17        </tr>
18        <tr>
19          <td><label> Al doilea numar </label></td>
20          <td>
21            <input type="number" name="n" size="5"
22              required min="1">
23          </td>
24        </tr>
25        <tr>
26          <td>
27            <p><input type="submit" value="Calculeaza">
28          </td>
29        </tr>
30      </table>
31    </form>
32    <center>
33  </body>
34 </html>

```

Există două metode `jspInit()` și `jspDestroy()` care dacă sunt declarate de programator atunci sunt executate la începutul și la sfârșitul ciclului de viață a paginii / documentului JSP.

**Exemplul 9.1.4** *Metoda `jspInit` inițializează un număr cu 0 iar metoda `jspDestroy` afișează numărul pe calculatorul serverului. Un client introduce un număr care este adunat la cel reținut de pagina JSP.*

Efectul metodei `jspDestroy` are loc în urma opririi aplicației JSP.

Codul paginii JSP este

```

1 <html>
2   <head>
3     <title> Init </title>
4   </head>
5   <body>
6     <%!
7       int numar;
8       public void jspInit(){

```

```

9      numar=0;
10     }
11     public void jspDestroy(){
12         System.out.println(numar);
13     }
14     %>
15     <center>
16     <h1> Pagina de r&#259;spuns </h1>
17     <p>
18     <%
19         String sn=request.getParameter("numar");
20         int n=Integer.parseInt(sn);
21         numar+=n;
22         out.println("Numarul este : "+numar);
23     %>
24     </center>
25 </body>
26 </html>

```

### 9.1.2 Directive JSP

*Directivele JSP* fixează informații pentru tot documentul jsp. O directivă jsp se indică prin marcajul

`<%@ directivă atribut1 atribut2 ... %>`

sau în format XML

`<jsp:directive.directiva atribut1 atribut2 ... />`

unde fiecare atribut are sintaxa *nume=valoare*.

Directivele pot fi: **page**, **include**, **taglib**.

- Directiva **page**. Menționăm attributele

```

import=      "listă de pachete separate prin ,"
info=        "text" Informația se poate regăsi apelând metoda
              getServletInfo()
errorPage=   "adresa url a paginii ce tratează excepția"
isErrorPage= "true | false"

```

- Directiva **include** permite includerea unor fișiere .html sau .jsp în document

`<%@ include file="fișier html, jsp" %>`

Includerea are loc în locul în care apare directiva.

Referința la fișierul `html` sau `jsp` se face relativ la catalogul paginii JSP inițiale (adică cea în care apare directiva).

- Directiva `taglib` indică bibliotecile de marcaje utilizate în documentul `jsp`, având atributele

```
uri=      "uri - Universal Resource Identifier - a bibliotecii de marcaje"
prefix=   "prefixul marcajului"
```

### 9.1.3 Marcaje JSP predefinite

Un marcaj JSP definește o acțiune care se execută în timpul procesării paginii `jsp`. Sintaxa marcajelor JSP seamănă cu cea a marcajelor `html` sau `xml`

```
<prefix : marcaj atribut />
```

Dintre marcajele JSP predefinite – adică cu prefixul JSP – amintim:

- `<jsp : include page="numeFișier jsp sau html" />`
- `<jsp : forward page="numeFișier jsp sau html" />`

Prelucrarea care urmează va fi cea din fișierul menționat. Diferența dintre cele două elemente constă în faptul că `include` prevede revenirea în pagina JSP inițială iar `forward` nu.

Referința la fișierul `html` sau `jsp` se face relativ la catalogul paginii JSP inițiale.

În cazul marcajelor `<jsp: include>`, `<jsp: forward>` se pot transmite parametri prin marcajele incluse

```
<jsp:param name="nume" value=valoare "/>
```

sau

```
<jsp:params>
  <jsp:param name="nume" value=valoare "/>
  . . . . .
</jsp:params>
```

- `<jsp : useBean id="numeComponentăJava"  
class="numeClasa"  
scope="domeniu" />`

unde *domeniu* precizează domeniul de valabilitate al componentei Java, adică `page`, `request`, `session`, `application`.

Crează un obiect "*numeComponentăJava*" de tip "*numeClasa*" având domeniul de valabilitate dată de "*domeniu*".

- `<jsp : setProperty name="numeComponentăJava"  
property="numeProp"  
value="valoare" />`

Acest marcaj este echivalent cu codul Java  
*numeComponentăJava.setNumeProp(valoare)*.

`<jsp : setProperty name="numeComponentăJava" property="*" />` vizează toate proprietățile componentei Java, fixarea valorilor făcându-se cu datele unui formular. Numele parametrilor din formularele de introducere a datelor trebuie să coincidă cu identificatorii câmpurilor din componenta Java corespunzătoare.

- `<jsp : getProperty name="numeComponentăJava"  
property="numeProp" />`

Preia și afișează valoarea câmpului *numeComponentăJava.numeProp*.

#### 9.1.4 Pagini JSP cu componente Java

Clasa componentei Java care se va utiliza într-o pagină JSP trebuie inclusă într-un pachet.

Reluăm exemplul 9.1.2 cu o componentă Java corespunzătoare numelui din formularul *index.html*.

##### Exemplul 9.1.5

```

1 package jsp;
2 public class HelloBean {
3     private String name="";
4     public String getName() {
5         return name;
6     }
7     public void setName(String name) {
8         this.name=name;
9     }
10 }
```

În acest caz, pagina JSP este (*hello.jsp*)

```

1 <jsp:useBean id="obj" class="jsp.HelloBean" scope="request" />
2 <jsp:setProperty name="obj" property="*" />
3 <html>
4   <head>
5     <title> jsphello </title>
6   </head>
7   <body>
8     <h1> Pagina de r&#259;spuns </h1>
9     <center>
10      <%
11        out.println("Hi "+obj.getName()+" !");
12      %>
13    </center>
14  </body>
15 </html>

```

Mai mult, se poate include formularul în pagina JSP, bineînțeles ștergându-l din fișierul `html`:

```

1 <jsp:useBean id="obj" class="jsp.HelloBean" scope="request" />
2 <jsp:setProperty name="obj" property="*" />
3 <html>
4   <head>
5     <title> jsphello </title>
6   </head>
7   <body>
8     <center>
9       <h1> Pagina JSP - aplica&#355;ia Hello </h1>
10      <form method="post">
11        <p> <h3> Introduceți numele: </h3>
12        <input type="text" name="name" size=20>
13        <p>
14        <input type="submit">
15      </form>
16      <p>
17      <%
18        out.println("Hi "+obj.getName()+" !");
19      %>
20    </center>
21  </body>
22 </html>

```

**Exemplul 9.1.6** *Pagină JSP pentru calculul celui mai mare divizor comun cu metoda de calcul definită într-o componentă Java.*

Utilizând documentului `html` din Exemplul 9.1.3 se definește componenta Java

```

1 package cmmdc;
2 public class CmmdcBean{
3   private String m="";
4   private String n="";
5   private String cmmdc;

```

```

7  public void setM(String m){
8      this.m=m;
9  }
10 public void setN(String n){
11     this.n=n;
12 }
13 public String getM(){
14     return m;
15 }
16 public String getN(){
17     return n;
18 }

20 public String getCmmdc(){
21     long a=Long.parseLong(m);
22     long b=Long.parseLong(n);
23     return (new Long(cmmdc(a,b))).toString();
24 }

26 long cmmdc(long m,long n){ . . . }

28 }

```

Instantîem o componenta Java și îi fixăm proprietățile (adică îi transmitem parametri problemei) după care apelăm metoda ce calculează rezultatul dorit în pagina JSP:

```

1  <jsp:useBean id="obj" class="cmmdc.CmmdcBean" scope="application" />
2  <jsp:setProperty name="obj" property="*" />
3  <html>
4  <body>
5      Cel mai mare divizor comun al numerelor
6      <p>
7          <%=obj.getM() %> si <%=obj.getN() %>
8          este <%=obj.getCmmdc() %>
9  </body>
10 </html>

```

### Exemplul 9.1.7 Generarea unei excepții (errhandler.jsp):

```

1  <%@ page errorPage="errorpage.jsp" %>
2  <html>
3  <body>
4      <%
5          String materia=request.getParameter("materia").trim();
6          if (materia.equals("AN")) {
7              out.println("<hr><font color=red>Alegere corecta !</font>");
8          }
9          else{
10             throw new Exception("N-ati facut alegerea corecta ");
11         }
12     %>
13 </body>
14 </html>

```

cu pagina de tratare a excepției (*errorpage.jsp*)

```

1 <!--
2 Aceste comentarii sunt foarte importante in cazul utilizarii
3 navigatorului IE si a lui apache-tomcat-5.*.*./6.*.*. In lipsa
4 lor nu se genereaza saltul la exceptie prin pagina jsp.
5 Rolul comentariilor este marirea lungimii fisierului de fata.

7 O alternativa este ca din IE6 . . . Options sa se dezactiveze
8 optiunea "Show friendly HTTP error message"

10 Cu navigatorul Firefox nu exista aceasta problema.
11 —>

13 <%@ page isErrorPage="true" %>
14 <html>
15   <body>
16     <div align="center">
17       <%= exception.getMessage() %>
18     </div>

20   </body>
21 </html>

```

apelate prin

```

1 <html>
2   <body>
3     <form method=post
4       action="jsp/errhandler.jsp">
5       Care este materia preferata din anii de studiu universitar ?
6       <p>
7         Algoritmica si programare
8         <input type="radio" name="materia" value="AP" checked>
9       <p>
10        Analiza numerica
11        <input type="radio" name="materia" value="AN">
12      <p>
13        Inteligenta artificiala
14        <input type="radio" name="materia" value="IA">
15      <p>
16        <input type=submit>
17    </form>
18  </body>
19 </html>

```

## 9.2 JSP Standard Tag Library JSTL

JSTL este o familie de biblioteci de marcaje ce oferă o serie de facilități activității de realizare a paginilor Web. JSTL ajută la separarea activității de programare de proiectarea (design) paginii Web.

JSTL este alcătuită din 5 biblioteci:

URI	Descriere
<a href="http://java.sun.com/jsp/jstl/core">http://java.sun.com/jsp/jstl/core</a>	Biblioteca de bază
<a href="http://java.sun.com/jsp/jstl/xml">http://java.sun.com/jsp/jstl/xml</a>	Biblioteca de prelucrare a documentelor xml
<a href="http://java.sun.com/jsp/jstl/fmt">http://java.sun.com/jsp/jstl/fmt</a>	Biblioteca de formatare a datelor
<a href="http://java.sun.com/jsp/jstl/sql">http://java.sun.com/jsp/jstl/sql</a>	Biblioteca de lucru cu baze de date
<a href="http://java.sun.com/jsp/jstl/functions">http://java.sun.com/jsp/jstl/functions</a>	Biblioteca de funcții ajutătoare

**Instalarea bibliotecilor.** Bibliotecile sunt livrate împreună cu *apache-tomcat-\** în catalogul *apache-tomcat-\*/webapps/examples/WEB-INF/lib* prin fișierele *taglibs-standard-spec-\*.jar* și *taglibs-standard-impl-\*.jar*.

**Utilizarea bibliotecilor.** În vederea utilizării, cele două fișiere trebuie copiate în catalogul *lib* al aplicației care utilizează bibliotecile.

În pagina / documentul JSP, o bibliotecă utilizată trebuie declarată printr-o directivă **taglib**.

### 9.2.1 Biblioteca de bază

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

Marcaje din biblioteca de bază:

- **c:set** Fixează o valoare într-o variabilă.

Atribute ale marcajului:

Atribut	Fel	Descriere
<b>var</b>	obligatoriu	Numele variabilei ce va stoca valoarea expresiei.
<b>value</b>	opțional	Expresia care va fi evaluată și atribuită variabilei
<b>scope</b>	opțional	Domeniul de valabilitate al variabilei. Unul din valorile: <b>page, request, session, application.</b>

Referirea la o variabilă se face prin sintaxa ***\${numeVariabilă}***

Referirea la valoarea unui câmp dintr-un formular se face prin ***\${param.numeCâmp}***

Alături de obiectul **param**, alte obiecte predefinite sunt **cookie, header, initParam, pageContext**.



Se pot defini variabile cu același nume dar având domenii de valabilitate diferită. Referirea se face prin `${pageScope.numeVariabilă}`, `${requestScope.numeVariabilă}`, `${sessionScope.numeVariabilă}`, `${applicationScope.numeVariabilă}`.

Plasând clauza **empty** înaintea unei variabile, `${empty numeVariabilă}`, se obține **false** sau **true** după cum variabila are sau nu atribuită o valoare.

- **c:remove** Șterge o variabilă.

Atribute ale marcajului:

Atribut	Fel	Descriere
<b>var</b> <b>scope</b>	obligatoriu opțional	Numele variabilei ce se șterge. Domeniul de valabilitate al variabilei. Unul din valorile: <b>page, request, session, application.</b>

- **c:out** Afișează o valoare.

Atribute ale marcajului:

Atribut	Fel	Descriere
<b>value</b> <b>default</b>	obligatoriu opțional	Valoarea ce se evaluează și se afișează. Cea ce se afișează în cazul în care expresia nu poate fi evaluată.
<b>escapeXml</b>	opțional	<b>true / false.</b> Valoarea implicită este <b>true</b> . Pe <b>false</b> interpretează caracterele din <b>value</b> ca și cod html.

- **c:if** Test, verificarea unei condiții.

Atribute ale marcajului:

Atribut	Fel	Descriere
<b>test</b>	obligatoriu	Condiția de test.
<b>var</b>	opțional	Numele variabilei ce va stoca valoarea testului.
<b>scope</b>	opțional	Domeniul de valabilitate al variabilei definită anterior.

În cazul în care condiția are valoarea **true** se prelucrează corpul marcajului, în caz contrar, acesta este ignorat.

**Exemplul 9.2.1** *Preluarea datelor unui formular cu câmpurile de intrare nume, prenume și email se face prin pagina JSP*

```

1 <HTML>
2 <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
3 <BODY>
4   <p>
5     <c:if test="${empty param.name}" var="testNume" >
6       <c:out value="Numele lipseste !" />
7     </c:if>
8     <c:if test="${not testNume}" >
9       Nume:<c:out value="${param.name}" />
10    </c:if>
11    <p>
12      <c:if test="${empty param.prenume}" var="testPrenume" >
13        <c:out value="Prenumele lipseste !" />
14      </c:if>
15      <c:if test="${not testPrenume}" >
16        Prenume:<c:out value="${param.prenume}" />
17      </c:if>
18      <p>
19        <c:if test="${empty param.email}" var="testEmail">
20          <c:out value="Adresa E-Mail lipseste !" />
21        </c:if>
22        <c:if test="${not testEmail}" >
23          E-mail:<c:out value="${param.email}" />
24        </c:if>
25      </BODY>
26 </HTML>

```

- **c:choose** Marcajul de selecție poate conține oricâte marcaje **c:when** și cel mult un marcaj **c:otherwise**. Fiecare marcaj **c:when** conține obligatoriu atributul **test**. Dacă într-un marcaj **c:when** condiția are valoarea **true**, atunci se prelucrează corpul acelui marcaj. În cazul în care toate marcajele **c:when** au fost evaluate cu **false** atunci se va prelucra marcajul **c:otherwise** (marcaj fără attribute).
- **c:forEach** Ciclu.

Atribute ale marcajului:

Atribut	Fel	Descriere
<b>items</b>	opțional	Colecția care se parcurge.
<b>var</b>	opțional	Numele variabilei în care se stochează valoarea elementului curent.
<b>begin</b>	opțional	Valoarea inițială a variabilei <b>var</b> .
<b>end</b>	opțional	Valoarea finală a variabilei <b>var</b> .
<b>step</b>	opțional	Valoarea pasului de iterare. Implicit este 1.
<b>varStatus</b>	opțional	Informații despre elementul curent.

Variabila `varStatus` are câmpurile:

- `index` valoarea curentă a elementului după care se realizează ciclarea;
- `count` numărul iterației curente;
- `first` are valoarea `true` dacă este primul element al ciclului;
- `last` are valoarea `true` dacă este ultimul element al ciclului;

**Exemplul 9.2.2** *Lista parametrilor formularului de apelare a exemplului anterior se afișează prin:*

```
<h2> Lista parametrilor din formular </h2>
<ul>
  <c:forEach items="${param}" var="p">
    <li>
      <c:out value="${p.key}"/> = <c:out value="${p.value}" />
    </li>
  </c:forEach>
</ul>
```

**Exemplul 9.2.3** *Lista parametrilor unui header se afișează prin:*

```
<h2> Lista campurilor din antet </h2>
<ul>
  <c:forEach items="${header}" var="h">
    <li>
      <c:out value="${h.key}"/> = <c:out value="${h.value}"/>
    </li>
  </c:forEach>
</ul>
```

**Exemplul 9.2.4** *Evidențierea fontului cu care se scriu titlurile într-un document html:*

```
<c:forEach begin="1" end="6" var="i" >
  <c:out value="<h${i}> Heading ${i} </h${i}>" escapeXml="false" />
</c:forEach>
```

- `c:forTokens` Asigură aceeași funcționalitate ca și clasei `java.util.StringTokenizer`.

Atribute ale marcajului:

Atribut	Fel	Descriere
<b>value</b>	obligatoriu	Valoarea ce se evaluează și se afișează. expresiei.
<b>default</b>	opțional	Cea ce se afișează în cazul în care expresia nu poate fi evaluată.
<b>escapeXml</b>	opțional	<b>true</b> / <b>false</b> . Valoarea implicită este <b>true</b> . Pe <b>false</b> interpretează caracterele din <b>value</b> ca și cod html.

- **c:import** Permite includerea altor pagini JSP în pagina curentă.

Atribute ale marcajului:

Atribut	Fel	Descriere
<b>url</b>	obligatoriu	Adresa documentului importat.
<b>context</b>	opțional	Context-ul paginii / documentului importat. Simbolul /, urmat de numele unei aplicații de pe același server.
<b>var</b>	opțional	Numele variabilei în care va fi stocat documentul importat.
<b>scope</b>	opțional	Domeniul de valabilitate al variabilei <b>var</b> . Unul din valorile: <b>page, request, session, application</b> .

Cu marcajul **c:param** se pot fixa parametri pentru pagina importată. Acest marcaj are două atribute **name** și **value**. Acești parametri se transmit cu metoda **get**.

- **c:redirect** Redirectarea activitatea către o altă pagină.

Atribute ale marcajului:

Atribut	Fel	Descriere
<b>url</b>	obligatoriu	Adresa paginii către care se face redirectarea.
<b>context</b>	opțional	Context-ul paginii către care se face redirectarea. Simbolul /, urmat de numele unei aplicații de pe același server.

Prin redirectare, parametrii nu sunt retransmiși automat mai departe.

- **c:url** Reține adrese URL.

Atribute ale marcajului:

Atribut	Fel	Descriere
<b>value</b>	obligatoriu	Adresa documentului de reținut.
<b>context</b>	opțional	Context-ul documentului. Simbolul /, urmat de numele unei aplicații de pe același server.
<b>var</b>	opțional	Numele variabilei în care va fi stocată adresa documentului.
<b>scope</b>	opțional	Domeniul de valabilitate al variabilei <b>var</b> . Unul din valorile: <b>page, request, session, application.</b>

### 9.2.2 Biblioteca de lucru cu baze de date

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

Marcaje din biblioteca de bază:

- **sql:setDataSource** Fixează referința la baza de date.

Atribute ale marcajului:

Atribut	Fel	Descriere
<b>dataSource</b>	opțional	Referința la baza de date
<b>driver</b>	opțional	Driver-ul bazei de date
<b>url</b>	opțional	url-ul bazei de date
<b>username</b>	opțional	nume utilizatorului bazei de date
<b>password</b>	opțional	parola de acces la baza de date
<b>var</b>	opțional	variabila cu referința la baza de date
<b>scope</b>	opțional	Domeniul de valabilitate al variabilei <b>var</b> .

- **sql:query** O interogare a bazei de date.

Atribute ale marcajului:

Atribut	Fel	Descriere
<b>sql</b>	obligatoriu	Fraza sql
<b>dataSource</b>	opțional	Referința la baza de date
<b>startRow</b>	opțional	Linia de la care se începe interogarea
<b>maxRows</b>	opțional	Numărul maxim de rezultate acceptate
<b>var</b>	obligatoriu	Variabila cu rezultatele interogării bazei de date
<b>scope</b>	opțional	Domeniul de valabilitate al variabilei <b>var</b> .

- `sql:update` Actualizarea bazei de date.

Atribute ale marcajului:

Atribut	Fel	Descriere
<code>sql</code>	obligatoriu	Fraza sql
<code>dataSource</code>	optional	Referința la baza de date

**Exemplul 9.2.5** *Să se afișeze lista din agenda de adrese e-mail creată în exemplul din Cap. Servlet.*

```

1 <HTML>
2 <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
3 <%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>
4 <BODY>
5   <p>
6     <sql:setDataSource
7       driver="org.apache.derby.jdbc.ClientDriver"
8       url="jdbc:derby://localhost:1527/AgendaEMail"
9       var="db" />
10    <sql:query
11      dataSource="${db}"
12      var="result"
13      sql="select * from adrese" />
14    <c:if test="${result.rowCount gt 0}" >
15      <table>
16        <tr>
17          <c:forEach items="${result.columnNames}" var="col">
18            <th>
19              <c:out value="${col}" />
20            </th>
21          </c:forEach>
22        </tr>
23        <c:forEach items="${result.rowsByIndex}" var="line" >
24          <tr>
25            <c:forEach items="${line}" var="elem" >
26              <td>
27                <c:out value="${elem}" />
28              </td>
29            </c:forEach>
30          </tr>
31        </c:forEach>
32      </table>
33    </c:if>
34 </BODY>
35 </HTML>

```

## 9.3 Marcaje JSP personale

Programatorul poate crea marcaje JSP proprii care se grupează în colecții numite biblioteci de marcaje. O bibliotecă de marcaje este reprezentată de un identificator, pe care-l vom denumi identificatorul bibliotecii de marcaje.

### 9.3.1 Marcaje fără attribute și fără corp.

Pentru a crea unui asemenea marcaj JSP propriu este necesară definirea următoarelor componente:

1. O clasă de definiție a comportamentului marcajului JSP (*tag handler class*).
2. Descriptorul bibliotecii de marcaje JSP, care leagă clasa de definiție a marcajului cu identificatorul bibliotecii de marcaje. Acest descriptor este un fișier cu extensia `tld`. Serverul Web va depista descriptorul bibliotecii de marcaje în catalogul aplicației.
3. Fișierul JSP ce utilizează marcajul JSP (clientul).

Exemplificăm această tehnologie prin

**Exemplul 9.3.1** *Să se realizeze un marcaj `dateTag`, a cărui efect să fie afișarea datei calendaristice.*

1. Clasa de definiție a comportamentului marcajului. Programul constă din:

- (a) Importul pachetelor

```
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
```

Aceste pachete se găsesc în fișierul `jsp-api.jar`.

- (b) Un marcaj fără attribute și fără corp trebuie să extindă clasa `TagSupport` și să suprascrie metoda `doStartTag`, care definește activitatea întreprinsă când este întâlnit marcajul într-un document *jsp*. Metoda trebuie să returneze constanta `SKIP_BODY`.

```
public class NumeClasa extends TagSupport{
    public int doStartTag(){
        . . .
        return SKIP_BODY;
    }
}
```

- (c) Scrierea în fluxul de ieșire se face cu un obiect `JspWriter`, care se obține cu `pageContext.getOut()`. Metoda `print` a clasei `JspWriter` poate genera o excepție `IOException`.

Textul sursă al clasei de definiție a comportamentului marcajului *dateTag* este:

```

1 package jsp;
2 import javax.servlet.jsp.JspWriter;
3 import javax.servlet.jsp.tagext.TagSupport;
4 import java.io.IOException;
5 import java.util.Date;

7 public class DateTag extends TagSupport{
8     public int doStartTag(){
9         try{
10             JspWriter out=pageContext.getOut();
11             out.println(new Date());
12         }
13         catch(IOException e){
14             System.out.println("DateTagException "+e.getMessage());
15         }
16         return SKIP_BODY;
17     }
18 }

```

2. Descriptorul bibliotecii de marcaje JSP este dependent de versiunea *tomcat* folosită. Acest fișier trebuie să aibă extensia `tld` (Taglib Language Definition).

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!-- a tag library descriptor -->
3 <taglib xmlns="http://java.sun.com/xml/ns/j2ee"
4         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
6         http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd"
7         version="2.0">

9     <tlib-version> 1.0 </tlib-version>
10    <jsp-version> 2.0 </jsp-version>
11    <short-name> mytagslibrary </short-name>
12    <uri>mytags</uri>
13    <description> Librarie de marcaje </description>

15    <tag>
16        <name> dateTag </name>
17        <tag-class> jsp.DateTag </tag-class>
18        <body-content> EMPTY </body-content>
19        <description> furnizeaza data curenta </description>
20    </tag>
21 </taglib>

```



Elementul *uri* conține identificatorul bibliotecii de marcaje, *mytags*.

Pentru fiecare marcaj propriu se completează un marcaj **tag**. Pentru un marcaj propriu fără attribute elementele acestui marcaj sunt

- (a) **name** Numele simbolic al marcajului.
- (b) **tag-class** Referința la fișierul class al clasei de definiție a comportamentului marcajului propriu. Referința se face relativ la catalogul `...\WEB-INF\classes`
- (c) **description** Descrierea marcajului propriu.
- (d) **body-content** În cazul nostru are valoarea **EMPTY**. În cazul unui marcaj cu corp se dă valoarea **JSP**.

Astfel elementele constitutive se vor găsi în:

```
webapps
|--> mytag
|   |--> WEB-INF
|   |   |--> classes
|   |   |   |--> jsp
|   |   |   |   |--> DateTag.class
|   |   |   |   mylibtag.tld
|   |--> jsp
|   |   |   dateTag.jsp
```

3. Marcajele proprii se utilizează cu sintaxa

*<prefix : NumeMarcaj />*

Referința la identificatorul bibliotecii de marcaje și prefixul se fixează în directiva **taglib**.

Un fișier *jsp* care utilizează marcajul realizat este (*dateTag.jsp*):

```
1 <html>
2   <head>
3     <title>
4       Tag pentru data calendaristica curenta
5     </title>
6   </head>
7   <body>
8     <%@ taglib uri="mytags"
```

```

9         prefix="mk" %>
10        <p>
11        Data curenta este:
12        <mk:dateTag/>
13    </body>
14 </html>

```

apelat din

```

1 <html>
2 <body>
3   <form method="get"
4     action="jsp/dateTag.jsp">
5
6     Data calendaristic&#259;:
7     <p><input type="submit" value="Afiseaza">
8   </form>
9 </body>
10</html>

```

### 9.3.2 Marcaje cu atribute și fără corp.

Realizăm un marcaj *ziuaTag* cu un atribut *ziua* care va fi afișat în momentul prelucrării marcajului.

1. Pentru fiecare atribut clasa ce definește acțiunea marcajului trebuie să conțină o metodă

```
public void setNumeAtribut(String value){...}
```

care preia valoarea atributului dată de parametrul *value*.

#### Exemplul 9.3.2

Pentru exemplul enunțat această clasă este

```

1 package jsp;
2 import javax.servlet.jsp.JspWriter;
3 import javax.servlet.jsp.tagext.TagSupport;
4 import java.io.IOException;
5
6 public class ZiuaTag extends TagSupport{
7     String ziua;
8
9     public void setZiua(String value){
10         ziua=value;
11     }

```

```

13 public int doStartTag(){
14     try{
15         JspWriter out=pageContext.getOut();
16         out.println(ziua);
17     }
18     catch(IOException e){
19         System.out.println("ZiuaTagException "+e.getMessage());
20     }
21     return SKIP_BODY;
22 }
23 }

```

2. În descriptorul bibliotecii de marcaje pentru fiecare atribut se definește un marcaj `<attribute>...</attribute>` având incluse marcajele

Nume marcaj	Semnificație	Fel
<b>name</b>	numele atributului	obligatoriu
<b>required</b>	true   false după cum atributul e obligatoriu sau nu	obligatoriu
<b>rtexprvalue</b>	true   false după cum atributul se poate utiliza într-o expresie < %= numeAtribut % >	opțional

Marcajul `<tag>` din descriptorul bibliotecii de marcaje devine

```

<tag>
  <name> ziuaTag </name>
  <tag-class> jsp.ZiuaTag </tag-class>
  <body-content> EMPTY </body-content>
  <description> furnizeaza argumentul ziua curenta </description>
  <attribute>
    <name>ziua</name>
    <required>true</required>
    <rtexprvalue>true</rtexprvalue>
  </attribute>
</tag>

```

3. Utilizarea acestui marcaj este exemplificat în

```

1 <html>
2   <body>
3     <form method="get"
4       action="jsp/ziuaTag.jsp">
5       <p>Astazi, este
6       <select name="ziua">
7         <option value="luni"> Luni
8         <option value="marti"> Marti
9         <option value="miercuri"> Miercuri

```

```

10         <option value="joi"> Joi
11         <option value="vineri"> Vineri
12         <option value="simbata"> Simbata
13         <option value="duminica"> Duminica
14     </select>
15     <p><input type="submit" value=" Afiseaza">
16 </form>
17 </body>
18 </html>

```

unde *ziuaTag.jsp* este

```

1 <html>
2 <head>
3   <title> Tag cu marcaj </title>
4 </head>
5 <body>
6   <%@ taglib uri="mytags"
7     prefix="mk" %>
8   <p>
9     <%
10       String zi=request.getParameter("ziua");
11     %>
12     Ziua este:
13     <mk:ziuaTag ziua="<%= zi %>" />
14   </body>
15 </html>

```

### 9.3.3 Marcaje cu corp.

În metoda `doStartTag` valoarea returnată trebuie să fie `EVAL_BODY_INCLUDE`, în loc de `SKIP_BODY`.

În descriptorul bibliotecii de marcaje apare

```
<body-content> JSP </body-content>
```

în loc de `EMPTY`.

Dacă se dorește ca marcajul să execute acțiuni după interpretarea corpului, atunci acele activități sunt definite în metoda `doEndTag`. Această metodă returnează valoarea `EVAL_PAGE` sau `SKIP_PAGE` după cum se dorește sau nu continuarea procesării paginii jsp.

**Exemplul 9.3.3** *Fie marcajul `modTag` care modifică un text în caractere mari sau mici după valoarea atributului `trans`. Acest marcaj poate include ale elemente.*

Codul clasei ce prelucrează marcajul este

```

1 package jsp;
2 import javax.servlet.jsp.JspWriter;
3 import javax.servlet.jsp.tagext.TagSupport;
4 import java.io.IOException;

6 public class ModTag extends TagSupport{
7     String text;
8     boolean toUpperCase;

10    public void setText(String value){
11        text=value;
12    }

14    public void setTrans(String value){
15        toUpperCase=(new Boolean(value)).booleanValue();
16    }

18    public int doStartTag(){
19        try{
20            JspWriter out=pageContext.getOut();
21            if(toUpperCase)
22                out.println(text.toUpperCase());
23            else
24                out.println(text.toLowerCase());
25        }
26        catch(IOException e){
27            System.out.println("ModTagException "+e.getMessage());
28        }
29        return EVAL_BODY_INCLUDE;
30    }
31 }

```

Descriptorul bibliotecii de marcate se completează cu

```

<tag>
  <name> modTag </name>
  <tag-class> jsp.ModTag </tag-class>
  <body-content> JSP </body-content>
  <description> modifica caracterele </description>
  <attribute>
    <name>text</name>
    <required>true</required>
    <rtexprvalue>true</rtexprvalue>
  </attribute>
  <attribute>
    <name>trans</name>
    <required>true</required>
    <rtexprvalue>true</rtexprvalue>
  </attribute>
</tag>

```

O pagină de utilizare a marcajului *modTag* cu un corp nevid este

```

1 <html>
2   <body>
3     <form method="get">

```

```

4      action="jsp/modtextTag.jsp">
5      Introducere o fraz&#259;
6      <input type="text" name="text" size="40" >
7      <p>
8      Se transform&#259; &#238;n litere
9      <select name="trans">
10         <option value="upperCase"> mari
11         <option value="lowerCase"> mici
12     </select>
13     <p><input type="submit" value=" Afiseaza">
14 </form>
15 </body>
16 </html>

```

împreună cu *modtextTag.jsp*

```

1 <html>
2 <body>
3     <%@ taglib uri="mytags" prefix="mk" %>
4     <%
5         String text=request.getParameter("text");
6         String trans=request.getParameter("trans");
7         String t;
8         if(trans.equals("upperCase"))
9             t="true";
10        else
11            t="false";
12    %>
13    <p>
14    <mk:modTag text="<%= text %>" trans="<%=t%>">
15        <mk:dateTag/>
16    </mk:modTag>
17 </body>
18 </html>

```

## 9.4 Autentificare și autorizare cu *apache-shiro*

*Apache-shiro* este un produs care permite autentificarea și autorizarea unei aplicații informatice ca o entitate independentă de aplicația în cauză. Datele de autentificare sunt înregistrate într-un fișier *shiro.ini*. În momentul de față nu este posibilă actualizarea dinamică a fișierului *shiro.ini*.

### Termeni

- *Credential* informație pe baza căreia se realizează autentificarea unui utilizator / subiect (de exemplu: o parolă);
- *Principal* Autentificarea se asigură printr-una sau mai multe *credentials* printre care se află *principal*, (de exemplu: username);

- *Realm* (tărâm/domeniu) entitatea care reține datele de identificare a unui utilizator;
- *Subject* termen utilizat pentru un utilizator (om sau program).

Fișierul de configurare **shiro.ini** pentru aplicație Web conține:

1. **[main]** shiro.loginUrl = /login.jsp

Se indică pagina de autentificare.

2. **[users]**

Declararea utilizatorilor împreună cu parola de autentificare și *rolul* / *rolurile*. Un rol fixează activitățile (*permissions*) de care dispune utilizatorul.

Sintaxa utilizată este

*numeUtilizator* = *parola*, *rol\_1*, *rol\_2*, ...

3. **[roles]**

Se declară activitățile permise de rol, mai precis pentru care se asigură autorizarea.

Sintaxa utilizată este

*rol* = *activitatea\_1*, *activitatea\_2*, ...

\* desemnează orice activitate.

4. **[urls]**

Se declară referințele din serverul Web la care se asigură accesul doar în urma autentificării, care sunt *filtrate* de *apache-shiro*.

Sintaxa utilizată este

*/fișier.jsp* = **authc**

Deconectarea se indică prin

*/logout* = **logout**

### Exemplul 9.4.1

```
1 [main]
2 shiro.loginUrl = /login.jsp
4 [users]
5 # format: username = password, role1, role2, ..., roleN
6 root = secret,admin
```





```
14     <filter -name>ShiroFilter</filter -name>
15     <filter -class>
16         org.apache.shiro.web.servlet.ShiroFilter
17     </filter -class>
18 </filter>

20 <filter -mapping>
21     <filter -name>ShiroFilter</filter -name>
22     <url-pattern>/*</url-pattern>
23 </filter -mapping>
24 </web-app>
```

**Marcaje definite de biblioteca** prefix=shiro, url=<http://shiro.apache.org/tags>.

- **<shiro:principal/ >**  
Furnizează utilizatorul.
- **<shiro:guest>**  
Execută marcajul interior dacă utilizatorul (*Subject*) nu este autentificat.
- **<shiro:user>**  
Execută marcajul interior dacă utilizatorul (*Subject*) este autentificat.
- **<shiro:hasRole name="rol">**  
Execută marcajul interior dacă utilizatorul are rolul *rol*.
- **<shiro:lacksRole name="rol">**  
Execută marcajul interior dacă utilizatorul nu are rolul *rol*.
- **<hasAnyRole name=rol1,rol2,...>**  
Execută marcajul interior dacă utilizatorul are unul din rolurile din listă.
- **<shiro:hasPermission name="acțiune">**  
Execută marcajul interior dacă rolului îi este atribuit activitatea *acțiune*.
- **<shiro:hasPermission name="acțiune">**  
Execută marcajul interior dacă rolului nu îi este atribuit activitatea *acțiune*.

**Exemplul 9.4.2** *Aplicație Web utilizând fișierul shiro.ini cu acces către*

- *act1: cmmdc1pagina.jsp*
- *act2: hello1pagina.jsp*

Aplicația are structura descisă mai sus.

Fișierul `index.html` este

```

1 <html>
2   <head>
3     <META HTTP-EQUIV="Refresh" CONTENT="0;URL=home.jsp">
4   </head>
5   <body>
6     <p>Apelarea aplicatiei...</p>
7   </body>
8 </html>

```

Fișierul `home.jsp` este

```

1 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
2 <%@ taglib prefix="shiro" uri="http://shiro.apache.org/tags" %>
3
4 <html>
5 <body>
6   <h1>Autentificare (Apache Shiro) </h1>
7
8   <p>Salut
9     <shiro:guest>Guest</shiro:guest>
10    <shiro:user><shiro:principal/></shiro:user>!
11
12    <shiro:user>
13      <table>
14        <tr>
15          <td>
16            <a href="<c:url value="/accesAutorizat/alegeActiune.jsp"/>">
17              Ac&#355;iuni</a>
18          </td>
19        </tr>
20        <tr>
21          <td>
22            <a href="<c:url value="/logout"/>">Log out</a>
23          </td>
24        </tr>
25      </table>
26    </shiro:user>
27    <shiro:guest>
28      <a href="<c:url value="/login.jsp"/>">Log in</a>
29    </shiro:guest>
30  </p>
31
32  <h3>Rolurile d-voastr&#259;</h3>
33  <p>
34    <shiro:hasRole name="admin">admin<br/></shiro:hasRole>
35    <shiro:hasRole name="roll">roll<br/>
36    <%
37      session.setAttribute("rol","roll");
38    %>
39    </shiro:hasRole>

```

```

40     <shiro:hasRole name="rol2">rol2<br/>
41     <%
42         session.setAttribute("rol", "rol2");
43     %>
44     </shiro:hasRole>
45 </p>

47 <h3>Roluri pe care nu le ave&#355;i</h3>
48 <p>
49     <shiro:lacksRole name="admin">admin<br/></shiro:lacksRole>
50     <shiro:lacksRole name="rol1">rol1<br/></shiro:lacksRole>
51     <shiro:lacksRole name="rol2">rol2<br/></shiro:lacksRole>
52 </p>

54 <h3>Activit&#259;&#355;ile d-voastr&#259;</h3>
55 <p>
56     <shiro:hasPermission name="act1">act1<br/>
57     <%
58         session.setAttribute("act1", "act1");
59     %>
60     </shiro:hasPermission>
61     <shiro:hasPermission name="act2">act2<br/>
62     <%
63         session.setAttribute("act2", "act2");
64     %>
65     </shiro:hasPermission>
66 </p>

68 <h3>Activit&#259;&#355;i de care nu dispune&#355;i</h3>
69 <p>
70     <shiro:lacksPermission name="act1">act1<br/></shiro:lacksPermission>
71     <shiro:lacksPermission name="act2">act2<br/></shiro:lacksPermission>
72 </p>
73 </body>
74 </html>

```

Fișierul `login.jsp` este

```

1 <html>
2 <body>
3 <h2>Pagina de conectare</h2>
4 <form method="post">
5     <table align="left" border="0" cellspacing="0" cellpadding="3">
6         <tr>
7             <td>Utilizator:</td>
8             <td>
9                 <input type="text" name="username" maxlength="30">
10            </td>
11        </tr>
12        <tr>
13            <td>Parola:</td>
14            <td>
15                <input type="password" name="password" maxlength="30">
16            </td>
17        </tr>
18        <tr>
19            <td colspan="2" align="right">
20                <input type="submit" name="submit" value="Login">
21            </td>

```

```

22     </tr>
23 </table>
24 </form>
25 </body>
26 </html>

```

Fișierul `alegeRol.jsp` este

```

1 <html>
2 <body bgcolor="#aaceaa">
3     <%
4         String rol=(String)session.getAttribute("rol");
5         if (rol=="rol1"){
6             %>
7             <jsp:forward page="alegeActiune.jsp" />
8             <%
9             }
10            if (rol=="rol2"){
11                %>
12                <jsp:forward page="alegeActiune.jsp" />
13                <%
14                }
15            %>
16        </body>
17 </html>

```

Fișierul `alegeActiune.jsp` este

```

1 <html>
2 <body bgcolor="#aaceaa">
3     <table>
4         <%
5             String act=(String)session.getAttribute("act1");
6             if (act=="act1"){
7                 %>
8                 <tr>
9                     <td>
10                        <a href="cmmdc1pagina.jsp">Cmmdc</a>
11                    </td>
12                </tr>
13                <%
14                }
15                act=(String)session.getAttribute("act2");
16                if (act=="act2"){
17                    %>
18                    <tr>
19                        <td>
20                            <a href="hello1pagina.jsp">Hello</a>
21                        </td>
22                    </tr>
23                <%
24                }
25            %>
26        </table>
27 </body>
28 </html>

```

Aplicația propriu-zisă (*cmmdc1pagina.jsp*) se completează la început și sfârșit cu

```
<%  
String act=(String)session.getAttribute("act1");  
if(act!="act1"){  
%>  
    <jsp:forward page="/home.jsp"/>  
%>  
}
```

respectiv

```
<p><a href="<c:url value="/home.jsp"/>">Return to the home page.</a></p>  
<p><a href="<c:url value="/logout"/>">Log out.</a></p>
```

Se procedează analog și pentru *hello1pagina.jsp*.

## Întrebări recapitulative

1. Care este tipul tehnologiei JSP ?
2. Precizați diferența dintre o pagină JSP și un document JSP.
3. Unde se instalează o pagină JSP?
4. Care este rolul unei declarații JSP ?
5. Care este rolul unei directive JSP ?
6. Cum apelează o pagină JSP ?
7. Cum prelucrează un server Web o pagină / document JSP ?
8. Care sunt trăsăturile unei componente Java (bean) ?
9. Care este rolul unui element `<jsp:useBean>` ?



# Capitolul 10

## Desfășurarea în *nor*

Dezvoltarea Internetului, nevoia de a reduce costurile legate de realizarea și întreținerea infrastructurii care oferă servicii pe Internet, concomitent cu nevoia de creștere a calității serviciilor a condus la *servicii în nor* (*Cloud Computing*).

Avantajele oferite de serviciile *serviciile în nor* sunt:

- Reducerea costurilor
- Agilitate (*Agility*)

Reducerea duratei:

- de așteptare în cazul apariției unei disfuncționalități din partea furnizorului *serviciului în nor*;
- de actualizare și întreținere din partea realizatorului *serviciului în nor*.

- Elasticitate (*Elasticity*)

Posibilitatea de creștere / descreștere a resurselor (în principal hard) alocate pentru a satisface cerințele clienților într-un interval de timp.

Se face distincție de *scalabilitate*, termen care desemnează nevoia de creștere / descreștere a resurselor alocate legată de dezvoltarea aplicațiilor care compun serviciul.

Tipuri de *servicii în nor*:

- Aplicații ca serviciu (*Software as a Service - SaaS*)  
*Skype, Google's Docs, Gmail, Yahoo Messenger, Microsoft Office 365, etc.*

- Infrastructură ca serviciu (*Infrastructure as a Service - IaaS*)  
*Amazon's Elastic Compute Cloud - (EC2)*
- Platformă ca serviciu (*Platform as a Service - PaaS*)  
PaaS poate fi
  - Ne-portabilă : aplicația va avea o structură predefinită.  
*Google AppEngine (GAE), Microsoft Azure, OpenShift*
  - Portabilă  
*Heroku*

În cele ce urmează ne interesează doar platformele PaaS care acceptă desfășurarea de aplicații Java, în mod gratuit, sau oferă un simulator local.

## 10.1 Servlet și JSP în *Google App Engine*

*Google AppEngine* permite:

- încărcarea unei aplicații Web pe un simulator local al platformei de Cloud Computing;
- încărcarea unei aplicații Web pe platforma Google de Cloud Computing.

În prezent, pe platforma GAE se pot încărca aplicații realizate în Java, Python, PHP și Go, alături de care pot apărea fișiere *http*, *css*, *js*. Există câte o distribuție distinctă pentru fiecare din aceste limbaje de programare.

### Utilizarea simulatorului local

Încărcarea pe simulatorul local al platformei GAE, în versiunea Java este construit peste serverul Web *jetty*.

**Instalarea** produsului constă din dezarhivarea fișierului *appengine-java-sdk-\**.

**Utilizarea.** În vederea încărcării unui servlet pe simulatorul local se crează structura de cataloage și fișiere

```
war
|--> WEB-INF
|   |--> classes
|   |   | *.class
|   |--> lib
|   |   | *.jar
|   |   web.xml
|   |   appengine-web.xml
|   index.html
```



Singurul fișier specific GAE este *appengine-web.xml*. Codul acestui fișier este

```

1 <appengine-web-app xmlns="http://appengine.google.com/ns/1.0">
2   <!-- Replace this with your application id from
3       http://appengine.google.com -->
4   <application></application>
6   <version>1</version>
8 </appengine-web-app>

```

Datele fișierului *web.xml* corespund servlet-ului, iar prin fișierul *index.html* se apelează aplicația Web. Parametrul **action** al elementului **form** are forma simplificată **action=/numeApel**, unde *numeApel* coincide cu valoarea atributului **urlPattern**.

Dacă aplicația se încarcă pe platforma *Google de Cloud Computing* atunci trebuie completat elementul **<application>**.

Lansarea simulatorului și încărcarea se poate face prin comenzile

```

set GAE_HOME=. .\appengine-java-sdk-*
%GAE_HOME%\bin\dev_appserver war

```

lansate într-o fereastră DOS, în catalogul care-l conține pe *war*. Aplicația se apelează prin **http://localhost:8080**. Dacă în loc de *index.html* se utilizează alt nume, atunci apelarea aplicației este **http://localhost:8080/fișier.html**.

O aplicație JSP se tratează asemănător.

Distribuția GAE pentru Java conține șablonul unei aplicații împreună cu un fișier *build.xml* (*appengine-java-sdk-\*\demos\new\_project\_template*) prin intermediul căruia, cu ajutorul lui *ant*, se construiește catalogul *war* descris anterior.

### Exemplul 10.1.1 *Servlet-ul CmmdcServlet instalat în platforma Google App Engine de Cloud Computing.*

Șablonul se copiază în zona de lucru sub numele *appcmmdc* și se completează cu fișierele servlet-ului (*CmmdcServlet.java*, *cmmdc.html*) rezultând:

```

appcmmdc
|--> src
|   |   CmmdcServlet.java
|--> war
|   |--> WEB-INF
|   |   |   appengine-web.xml
|   |   |   web.xml
|   |   |   cmmdc.html
|   |   |   build.xml

```

Se execută cu *ant* obiectivul implicit din `build.xml`, urmat de lansarea simulatorului din interiorul catalogului *appcmmddc*. Deoarece numele fișierului html diferă de *index*, aplicația se apelează prin `http://localhost:8080/cmmddc.html`.

Alternativ, aplicația se poate construi cu *Eclipse* folosind o componentă (*plug-in*) specifică.

*GAE* conține în plus un serviciu de autentificare și autorizare *UserService*, un sistem de persistență a datelor *Datastore*, *Task-Queue*.

## Desfășurarea în GAE

Acest pas necesită din partea dezvoltatorului cont Google.

Desfășurarea gratuită (cel mult 10 aplicații) în GAE presupune:

1. Înregistrarea aplicației:

- (a) Se accesează pagina Web <https://appengine.google.com/>.

- (b) Înregistrarea propriu-zisă:

- i. Application Identifier

Se fixează *identificatorul aplicației* – *app\_id*, care se trece și în elementul `<application>` din `appengine-web.xml`;

- ii. Check Availability

Verificarea disponibilității identificatorului;

- iii. Application title

Fixarea titlului aplicației;

- iv. Create Application

2. Încărcarea în nor (upload)

```
%GAE_HOME%\bin\appcfg.cmd --oauth2 update locatia_aplicatiei_web
```

Prin *locatia\_aplicatiei\_web* se înțelege catalogul `war` construit de GAE.

Aplicația va fi disponibilă prin

[http://app\\_id.appspot.com/](http://app_id.appspot.com/) sau

[http://app\\_id.appspot.com/fisier.html](http://app_id.appspot.com/fisier.html)

## 10.2 Heroku

*Heroku* este o PaaS care oferă suport pentru mai aplicații dezvoltate în mai multe limbaje de programare, printre care și Java.

## Instalarea resurselor

1. Punctul de pornire este crearea unui cont *Heroku* la <https://devcenter.heroku.com> prin *Getting Started*. Parametrii contului sunt (adresă de email, parolă).
2. Se va instala *Heroku Toolbelt*. Cu acest prilej se instalează *Ruby* și *Git* în `c:\Program Files (x86)`.<sup>1</sup> *Ruby* se instalează în *Heroku*. Variabila de sistem `PATH` se actualizează cu căile la *Heroku* și *Git* (dar nu la `Git\bin\ssh-keygen.exe`).
3. Login pentru generarea cheii de identificare.

```
heroku login
```

La conectări ulterioare nu se mai generează această cheie.

## Utilizarea.

Gestionarea aplicațiilor se realizează la <https://dashboard.heroku.com/apps>.

Dezvoltarea aplicațiilor se bazează pe *maven* iar desfășurarea pe *Git*.

### 10.2.1 JSP în *Heroku*

1. Generarea aplicației:

```
set GroupID=myGroupId
set ArtifactID=myArtifactId
set Version=1.0
mvn archetype:generate -DgroupId=%GroupID%
-DartifactId=%ArtifactID%
-Dversion=%Version%
-DarchetypeArtifactId=maven-archetype-webapp
-DinteractiveMode=false
```

2. Completarea fișierului `pom.xml`:

```
1 <project xmlns="http://maven.apache.org/POM/4.0.0"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4     http://maven.apache.org/maven-v4_0_0.xsd">
5   <modelVersion>4.0.0</modelVersion>
```

<sup>1</sup>Se are în vedere sistemul de operare Windows.

```

6   <groupId>myGroupId</groupId>
7   <artifactId>myArtifactId</artifactId>
8   <packaging>war</packaging>
9   <version>1.0-SNAPSHOT</version>
10  <name>MyArtifactId Maven Webapp</name>
11  <url>http://maven.apache.org</url>
12  <dependencies>
13    <dependency>
14      <groupId>org.eclipse.jetty</groupId>
15      <artifactId>jetty-servlet</artifactId>
16      <version>9.1.0.v20131115</version>
17    </dependency>
18    <dependency>
19      <groupId>junit</groupId>
20      <artifactId>junit</artifactId>
21      <version>3.8.1</version>
22      <scope>test</scope>
23    </dependency>
24  </dependencies>
25  <build>
26    <finalName>myArtifactId</finalName>
27    <plugins>
28      <plugin>
29        <groupId>org.apache.maven.plugins</groupId>
30        <artifactId>maven-dependency-plugin</artifactId>
31        <version>2.3</version>
32        <executions>
33          <execution>
34            <phase>package</phase>
35            <goals><goal>copy</goal></goals>
36            <configuration>
37              <artifactItems>
38                <artifactItem>
39                  <groupId>org.eclipse.jetty</groupId>
40                  <artifactId>jetty-runner</artifactId>
41                  <version>9.1.0.v20131115</version>
42                  <destFileName>jetty-runner.jar</destFileName>
43                </artifactItem>
44              </artifactItems>
45            </configuration>
46          </execution>
47        </executions>
48      </plugin>
49    </plugins>
50  </build>
51 </project>

```

3. Completarea aplicației cu fișierele `jsp`.
4. Opțional aplicația se poate arhiva și verifica.
5. Completarea cu fișierele

- Procfile

```
1 web: java $JAVA_OPTS -jar target/dependency/jetty-runner.jar
2 --port $PORT target/*.war
```

- **system.properties**

```
1 java.runtime.version=1.7
```

- **.gitignore**

```
1 target
```

Structura aplicației este

```
myapp
|--> src
|   |--> main
|   |   |--> resources
|   |   |--> webapp
|   |   |   |--> WEB-INF
|   |   |   |   web.xml
|   |   |   |   fisiere.jsp
|--> target
|   |   *
|   .gitignore
|   Procfile
|   system.properties
|   pom.xml
```

## 6. Pregătirea *Git*

```
git init
git add .
git commit -m "myapp"
```

## 7. Generarea și desfășurarea aplicației

```
heroku create
git push heroku master
```

*Heroku* atribuie un nume aplicației. Dintr-un navigator aplicația se va apela prin acest nume:

<http://numeDatDeHeroku.herokuapp.com/>

## 8. Calibrare

```
heroku ps:scale web=1
```

cu verificarea calibrării

```
heroku ps
```

Această setare fixează resursele atribuite aplicației la 1 *dynos*. Un număr mai mare de resurse presupune un cost.

#### 9. Lansarea aplicației în linia de comandă din catalogul aplicației

```
heroku open
```

#### 10. Opțional se poate consulta fișierul de jurnalizare

```
heroku logs
```

### 10.2.2 Servlet în *Heroku*

Aplicația servlet se bazează pe interfața de programare `servlet-api 2.5`.

Clasa servletului are metoda `main` prin care se lansează un server Web încorporat.

```

1 package hello.heroku;
2 import java.io.IOException;
3 import javax.servlet.ServletException;
4 import javax.servlet.http.HttpServlet;
5 import javax.servlet.http.HttpServletRequest;
6 import javax.servlet.http.HttpServletResponse;
7 import javax.servlet.ServletOutputStream;
8 import org.eclipse.jetty.server.Server;
9 import org.eclipse.jetty.servlet.ServletContextHandler;
10 import org.eclipse.jetty.servlet.ServletHolder;

12 public class HelloWorld extends HttpServlet {
13     @Override
14     protected void doGet(HttpServletRequest req, HttpServletResponse resp)
15         throws ServletException, IOException {
16         resp.getWriter().print("Hello from Java!\n");
17     }

19     public static void main(String[] args) throws Exception{
20         Server server = new Server(Integer.valueOf(System.getenv("PORT")));
21         ServletContextHandler context =
22             new ServletContextHandler(ServletContextHandler.SSESSIONS);
23         context.setContextPath("/");
24         server.setHandler(context);
25         context.addServlet(new ServletHolder(new HelloWorld()), "/*");
26         server.start();
27         server.join();
28     }
29 }
```

Fișierele `html`, `css` care asigură interfața grafică a aplicației servlet nu fac parte din ce se încarcă în nor.

Servletul se apelează prin

`http://numeDatDeHeroku.herokuapp.com/numeApel`

Desfășurarea și apelarea constă din

1. Generarea aplicației.
2. Completarea fișierului `pom.xml`:

```

1 <project xmlns="http://maven.apache.org/POM/4.0.0"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4     http://maven.apache.org/maven-v4_0_0.xsd">
5   <modelVersion>4.0.0</modelVersion>
6   <groupId>myGroupId</groupId>
7   <artifactId>myArtifactId</artifactId>
8   <packaging>war</packaging>
9   <version>1.0-SNAPSHOT</version>
10  <name>myArtifactId Maven Webapp</name>
11  <url>http://maven.apache.org</url>
12  <dependencies>
13    <dependency>
14      <groupId>org.eclipse.jetty</groupId>
15      <artifactId>jetty-servlet</artifactId>
16      <version>9.1.0.v20131115</version>
17    </dependency>
18    <dependency>
19      <groupId>javax.servlet</groupId>
20      <artifactId>servlet-api</artifactId>
21      <version>2.5</version>
22    </dependency>
23  </dependencies>
24  <build>
25    <finalName>myArtifactId</finalName>
26    <plugins>
27      <plugin>
28        <groupId>org.apache.maven.plugins</groupId>
29        <artifactId>maven-compiler-plugin</artifactId>
30        <version>3.1</version>
31        <configuration>
32          <source>1.7</source>
33          <target>1.7</target>
34        </configuration>
35      </plugin>
36      <plugin>
37        <groupId>org.apache.maven.plugins</groupId>
38        <artifactId>maven-dependency-plugin</artifactId>
39        <version>2.4</version>
40        <executions>
41          <execution>
42            <id>copy-dependencies</id>
43            <phase>package</phase>
44            <goals><goal>copy-dependencies</goal></goals>
45          </execution>

```

```

46         </executions>
47     </plugin>
48     <plugin>
49         <groupId>org.eclipse.jetty</groupId>
50         <artifactId>jetty-maven-plugin</artifactId>
51         <version>9.1.0.v20131115</version>
52     </plugin>
53 </plugins>
54 </build>
55 </project>

```

3. Completarea aplicației cu fișierele `java` și `web.xml`.

4. Opțional aplicația se poate arhiva și verifica.

Verificarea se face prin

```

set PORT=5000
java -cp myArtifactId\target\classes;"myArtifactId\target\dependency\*"
    clasaServlet

```

5. Completarea cu fișierele

- Procfile

```

1 web: java $JAVA_OPTS -cp target/classes:target/dependency/*
2     clasaServlet

```

- system.properties

- .gitignore

Structura aplicației devine

```

myapp
|--> src
|   |--> main
|       |--> java
|           *.java
|       |--> resources
|       |--> webapp
|           |--> WEB-INF
|               web.xml
|--> target
|   *
|   .gitignore
|   Procfile
|   system.properties
|   pom.xml

```

6. Pregătirea *Git*



```
git init
git add .
git commit -m "myapp"
```

#### 7. Generarea și desfășurarea aplicației

```
heroku create
git push heroku master
```

*Heroku* atribuie un nume aplicației. Dintr-un navigator aplicația se va apela prin acest nume.

#### 8. Calibrare

```
heroku ps:scale web=1
```

cu verificarea calibrării

```
heroku ps
```

Această setare fixează resursele atribuite aplicației la 1 *dynos*. Un număr mai mare de resurse presupune un cost.

#### 9. Dacă se transmit date aplicației printr-un fișier `html` atunci lansarea aplicației se face din navigator completând atributul `action` cu adresa furnizată de heroku iar în caz contrar din linie de comandă se apelează

```
heroku open
```

#### 10. Opțional se poate consulta fișierul de jurnalizare

```
heroku logs
```

## 10.3 *OpenShift*

*OpenShift* este produs de *RedHat* cu suport pentru aplicații distribuite în Java dar și pentru (python, ruby, javascript, php).

Comunicațiile unei aplicații se fac exclusiv potrivit protocoalelor și porturilor din tabelul următor:

Protocolul	Portul
HTTP	80
HTTPS	443
SSH- <i>SecureShell</i>	22
WebSocket HTTP	8000
WebSocket HTTPS	8443

O aplicație aparține unui cartuş - **cartridge** - dar care poate conține și alte componente ale aplicației (de exemplu un SGBD). Unui cartuş i se asociază un angrenaj - **gear** - ce trebuie privit ca un container al aplicației. Angrenajul poate fi mic (512 MB), mediu (1 GB) sau mare (2 GB).

Cartuşul definește natura aplicației (servlet sau JSP în tomcat 6 sau 7, aplicație JEE, etc).

Un utilizator trebuie să se înregistreze (gratuit) la [www.openshift.com](http://www.openshift.com), moment în care se fixează codul de identificare - o adresă de e-mail - și o parolă.

Unui dezvoltator i se asociază un *domeniu* - *namespace* - indicat printr-un cuvânt. Apelarea unei aplicații se face după schema

`http://numeApp-domeniu.rhcloud.com`

Gratuit un dezvoltator poate instala până la 3 aplicații într-un angrenaj mic.

O aplicație se poate dezvolta:

- în linie de comandă prin clientul *OpenShift* **rhc**;
- într-o consola Web;
- în Eclipse cu o componentă (plug-in) specifică.

În continuare considerăm doar dezvoltarea în linie de comandă.

## Instalarea clientului **rhc**

.

1. Se instalează *Ruby* (<http://rubyinstaller.org/downloads>). Este suficientă o versiune fără procedura propriu-zisă de instalare (7z).

Verificarea instalării:

```
ruby -e 'puts "Welcome to Ruby"'
```

2. Se instalează *Git* (<http://git-scm.com/download/win>).

Verificarea instalării:

```
git --version
```

3. Descărcarea clientului *OpenShift*

```
gem install rhc
```

Locația este `RUBY_HOME\include`.

4. Instalarea clientului *OpenShift*

```
rhc setup
```

## Dezvoltarea unei aplicații

1. `rhc app create numeApp cartuş`

Cartușe pentru aplicații Java (gratuite)

Denumire cartuş	Conținut
jbossews-1.0	Tomcat 6
jbossews-2.0	Tomcat 7
jbossas-7	JBoss Application Server 7
jenkins-1	Jenkins Server

Pentru `jbossews-2.0`, în catalogul de lucru se generează structura

```
numeApp
|--> .git (Catalog ascuns - hidden)
|   |   * - resurse Git
|--> .openshift
|   |   * - resurse OpenShift
|--> src
|   |--> main
|   |   |--> java
|   |   |--> resources
|   |   |--> webapp
|   |   |   |--> images
|   |   |   |--> WEB-INF
```

```
|      |      |      |      |      web.xml
|      |      |      |      |      index.html
|      |      |      |      |      snoop.jsp
|--> webapps
|      README.md
|      pom.xml
```

Dezvoltatorul completează catalogul `src` cu fișierele sursă ale aplicației Web.

### Încărcarea în nor

```
git add .
git commit -m "text"
git push
```

### Operații rhc

- `rhc --help`
- `rhc --help comandă_rhc`
- `rhc cartridge list`  
Lista cartușelor disponibile.
- `rhc app delete numeApp`  
Ștergerea aplicației.
- `rhc apps`  
Lista aplicațiilor.

### Întrebări recapitulative

1. Precizați tipurile principale de servicii în nor.
2. Enumerați servicii în nor de tip PaaS cu suport pentru Java.

# Capitolul 11

## Java Web Start

### 11.1 Java Web Start

O aplicație Java destinată a fi utilizată pe un calculator poate fi valorificată și într-o rețea, pe baza protocolului *Java Network Launching Protocol - JNLP*. Tehnologia poartă numele de *Java Web Start*.

Aplicația Java trebuie să satisfacă o serie de restricții:

- Aplicația trebuie arhivată cu `jar`. În fișierul `MANIFEST.MF` trebuie să apară atributul

`Permissions: all-permissions`

- Arhiva `jar` trebuie certificată. Certificarea se realizează cu utilitarele `keytool.exe` și `jarsigner.exe` din distribuția `jdk`;
- Eventualele date necesare aplicației se introduc prin intermediul unei interfețe grafice (*JavaFX*, *swing*, *SWT*, *apache-pivot*);
- Acces limitat la proprietățile și resursele sistemului client.

Aplicației i se atașează un fișier `xml`, dar cu extensia `jnlp`, care se va apela dintr-un navigator. Pentru acest fișier folosim terminologia de fișier *jnlp*.

Fișierul *jnlp* fixează:

- referința URL a aplicație (prin atributul `codebase`);
- arhivele `jar` utilizate (prin atributul `href`);
- clasa cu metoda `main` (prin atributul `main-class`).

Ansamblul de resurse formează o aplicație Web care poate fi desfășurată într-un server Web (container de servlet) sau poate fi incorporat într-un servlet.

Arătăm activitățile care trebuie întreprinse în cazul unui exemplu simplu dat de clasa *VisualCmmdc.java*.

```

1 public class VisualCmmdc extends javax.swing.JFrame {
2     public VisualCmmdc(){
3         initComponents();
4     }
5
6     private long cmmdc(long m,long n){. . .}
7
8     private void initComponents() {
9         // cod generat de Netbeans
10    }
11
12    private void cmmdcButtonMouseClicked(java.awt.event.MouseEvent evt){
13        try{
14            String sm=mTextField.getText();
15            String sn=nTextField.getText();
16            long m=Long.parseLong(sm);
17            long n=Long.parseLong(sn);
18            long c=cmmdc(m,n);
19            String s=(new Long(c)).toString();
20            cmmdcTextField.setText(s);
21        }
22        catch(Exception e){
23            System.err.println("Exception : "+e.getMessage());
24        }
25    }
26
27    private void exitForm(java.awt.event.WindowEvent evt){
28        System.exit(0);
29    }
30
31    public static void main(String args[]) {
32        new VisualCmmdc().setVisible(true);
33    }
34
35    private javax.swing.JButton cmmdcButton;
36    private javax.swing.JLabel mLabel;
37    private javax.swing.JLabel nLabel;
38    private javax.swing.JTextField mTextField;
39    private javax.swing.JTextField nTextField;
40    private javax.swing.JTextField cmmdcTextField;
41 }

```

Interfața grafică a programului este ilustrată în Fig. 11.1

## Instalare într-un server Web

Pentru instalarea aplicației într-un server Web, se parcurg pașii:



Figure 11.1: Interfața grafică a aplicației.

## 1. Se arhivează aplicația

```
jar cmfv myManifest.mf cmmdc0.jar *.class
```

unde fișierul *myManifest.mf* conține doar linia

```
Permissions: all-permissions
```

## 2. Certificarea se obține prin

- (a) `keytool -genkey -keystore myKeystore -alias myself  
-dname "cn=XYZ, ou=cs, o=unitbv, l=brasov, c=RO"  
-keypass abc123 -storepass 123abc`
- (b) `keytool -selfcert -alias myself -keystore myKeystore  
-keypass abc123 -storepass 123abc`
- (c) `jarsigner -keystore myKeystore -signedjar cmmdc.jar  
-keypass abc123 -storepass 123abc cmmdc0.jar myself  
jarsigner -verify cmmdc.jar`

Primele două acțiuni au ca rezultat obținerea certificatului *myKeystore* iar ultima acțiune reprezintă înglobarea certificării în arhiva *resursa.jar* - în cazul exemplului *cmmdc.jar*.

3. Se editează fișierul *launch.jnlp*

```

1 <?xml version="1.0" encoding="UTF-8"?>
2   <jnlp codebase="http://host:8080/cmmdc">
3     <information>
4       <title> . . . </title>
5       <vendor> . . . </vendor>
6       <description> . . . </description>
7     </information>
8     <resources>
9       <j2se version="1.2+" />
10      <jar href="cmmdc.jar" />
11    </resources>
12    <security>
13      <all-permissions/>
14    </security>
15    <application-desc main-class="VisualCmmdc" />
16  </jnlp>

```

*host* se înlocuiește cu numele calculatorului care găzduiește serverul Web.

4. Se crează fișier de apelare a aplicației (*cmmdc.html*)

```

1 <html>
2   <body bgcolor="#AEEFAA">
3     <center>
4       <h3>Visual Cmmdc Application </h3>
5       <a href="http://host:8080/cmmdc/launch.jnlp">
6         Launch the application </a>
7     </center>
8   </body>
9 </html>

```

5. Ansamblul

```

cmmdc
|   cmmdc.jar
|   cmmdc.html
|   launch.jnlp

```

se copiază în serverul Web.

Dintr-un navigator, apelarea aplicației este *http://host:port/cmmdc/cmmdc.html*.

Fișierul *jnlp* se descarcă pe calculatorul clientului și se procesează cu *bin\javaws.exe* din Java.

## Aplicația Java ca servlet

Pentru includerea aplicație Java într-un servlet, primii trei pași prezentați mai sus coincid. Singura diferență constă în conținutul referintelor, după *host* va apare portul utilizat de serverul Web container de servlet, uzual *host:8080*.

4. Într-un catalog de lucru se crează structura de cataloage și fișiere:

```

app
|   cmmdc.jar
|   launch.jnlp
WEB-INF
|---> lib
|       |   jnlp-servlet.jar
|       |   web.xml
cmmdc.html

```

Fișierul *jnlp-servlet.jar* se preia din distribuția JDK, din catalogul *sample/jnlp/servlet*.

Fișierul *web.xml* este



```

1 ?xml version="1.0" encoding="iso-8859-1"?>
2 <!DOCTYPE web-app
3     PUBLIC "-//Sun Microsystems, Inc./DTD Web Application 2.3//EN"
4     "http://java.sun.com/dtd/web-app_2_3.dtd">
5 <web-app>
6     <!-- Standard Action Servlet Configuration (with debugging) -->
7     <web-app>
8         <servlet>
9             <servlet-name>JnlpDownloadServlet</servlet-name>
10            <servlet-class>
11                jnlp.sample.servlet.JnlpDownloadServlet
12            </servlet-class>
13        </servlet>
14        <servlet-mapping>
15            <servlet-name>JnlpDownloadServlet</servlet-name>
16            <url-pattern>*.jnlp</url-pattern>
17        </servlet-mapping>
18    </web-app>
19 </web-app>

```

5. Conținutul catalogului de lucru se arhivează într-un fișier **war**.

```
jar cvf cmmdc.war app/* WEB-INF/* cmmdc.html
```

6. Utilizând *apache-tomcat* sau *jetty*, de exemplu, fișierul **war** se copiază în catalogul **webapps** al serverului web.

Dintr-un navigator, apelarea aplicației se obține prin

*http://host:8080/cmmdc/cmmdc.html*.

- Dacă aplicația Java utilizează imagini grafice acestea trebuie arhivate împreună cu aplicația, de exemplu într-un catalog *images*.

Încărcarea și afișarea unei imagini programându-se prin

```

ClassLoader cl=this.getClass().getClassLoader();
URL file=cl.getResource("images/pic1.jpg");
Image img=Toolkit.getDefaultToolkit().getImage(file);
graphics.drawImage(img,x,y,this);

```

- 1. Dacă aplicația Java utilizează alte resurse date prin fișiere **jar** atunci toate fișierele **jar** trebuie să poarte aceeași certificare.
- 2. Fișierele **jar** certificate se depun în catalogul *app\lib*.
- 3. În fișierul *jnlp*, în elementul **resources**, pentru fiecare fișier **jar** utilizat se introduce declarația

```
<jar href="lib/resource.jar"/>
```

- Dacă arhiva `jar` a aplicației Java este executabilă atunci, în fișierul `jnlp`, elementul `application-desc` poate lipsi.

Într-o arhivă `jar` executabilă, fișierul `MANIFEST.MF` indică clasa cu metoda `main` și eventualele resurse externe utilizate (fișiere `jar`) prin proprietățile

```
Main-class: numeClasa  
Class-path: lib/resursa1.jar lib/resursa2.jar . . .
```

Pentru a obține fișierul `MANIFEST.MF` cu aceste proprietăți se editează un fișier text cu conținutul de mai sus, denumit de exemplu `myManifest.mf`, și se arhivează prin

```
jar cfvm numeArhiva.jar myManifest.mf *.class lib
```

În mod asemănător se procedează și pentru introducerea proprietății `Permissions: all-permissions.`

## Întrebări recapitulative

1. Ce posibilitate oferă tehnologia Java Web Start ?
2. O aplicație Java urmează să fie valorificat pe Internet prin Java Web Start. Dacă aplicația necesită date ale clientului ce restricție există și cum se satisface restricția ?
3. Ce înseamnă abrevierea JNLP?
4. Unde se utilizează JNLP?
5. În ce constă utilizarea protocolului JNLP?
6. Care este programul executabil care precesează un fișier `jnlp`?

# Capitolul 12

## WebSocket

HTML5 introduce protocolul *WebSocket* pentru comunicații bidirecționale între un client HTML5 dintr-un navigator și un server Web. Utilizarea protocolului WebSocket conduce la reducerea semnificativă a traficului în rețea.

Toate navigatoarele importante suportă protocolul WebSocket.

### 12.1 Protocolul WebSocket

Printr-un mesaj `http` trimis de client se indică trecerea de la protocolul `http` către protocolul WebSocket. Schimbarea protocolului este indicat de antetul `Upgrade: websocket`. Exemplul unui asemenea mesaj este

```
GET /HelloWebSocket/hello HTTP/1.1
Upgrade: websocket
Connection: Upgrade
Host: localhost:9090
Origin: null
Sec-WebSocket-Key: tLZ8VGZ8Cw8kt0BvhuV6Vw==
Sec-WebSocket-Version: 13
Sec-WebSocket-Extensions: x-webkit-deflate-frame
Cookie: JSESSIONID=2BCFF666164139524DD92D573C3859F7;
JSESSIONID=bf12d2417e8eb1bcd6da6137af9d;
treeForm_tree-hi=treeForm:tree:applications
```

Mesajul de răspuns este

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: ZDE5NmS4T9spIby8/vo/V+rbNAs=
```

Dacă între cei doi parteneri se stabilește acordul (*handshake*) atunci restul comunicațiilor au loc prin intermediul unui soclu TCP pe portul 80.

Astfel ciclul de viață al procesului de comunicație este:

1. Un client solicită printr-un mesaj **http** acordul pentru trecerea la protocolul WebSocket.
2. Serverul răspunde acceptând acordul.
3. Odată stabilită conexiunea, acesta devine bidirecțională (simetrică), clientul și serverul transmit și recepționează mesaje.
4. Una din părți închide conexiunea.

## 12.2 Interfața de programare HTML5 de client WebSocket

Interfața de programare se utilizează în JavaScript.

### Constructor

- `WebSocket(in String uri)`  
unde *uri* este de forma `ws://host:port/context/numeApel`

### Funcții

- `attribute function onopen(evt)`
- `attribute function onmessage(evt)`
- `attribute function onerror(evt)`
- `attribute function onclose(evt)`
- `boolean send(in String data)`
- `close()`

### Atribute

- `readyState`

Starea conexiunii:

Valoare	Semnificația
0	Nu s-a stabilit conexiunea
1	Conexiune pregătită pentru comunicații
2	Conexiune în pragul <i>confirmării</i> ( <i>handshake</i> )
3	Conexiune închisă și nu mai poate fi redeschisă

- **bufferedAmount**

Numărul octeților trimiși de funcția **send**. Datele sunt codificate UTF-8.

Un șablon de utilizare poate fi

```
<script language="javascript" type="text/javascript">
    var wsUri = "ws://host:8080/context/numeApel";
    var websocket = new WebSocket(wsUri);
    websocket.onopen = function(evt) { . . . };
    websocket.onmessage = function(evt) { . . . };
    websocket.onerror = function(evt) { . . . };
    websocket.onclose = function(evt) { . . . };
    . . .
</script>
```

### Expedierea datelor.

Considerăm formularul HTML

```
<form name="myform">
    <input type="text" name="xyz" . . . />
    . . .
    <input type="button" onclick="send"/>
</form>
```

Șablonul funcției *send* este

```
function send(){
    var xyz=document.myform.xyz.value;
    websocket.send(xyz);
}
```

**Recepția unui rezultat furnizat de server.** Funcția **onmessage** permite recuperarea rezultatului din **evt.data**.

## 12.3 WebSocket în Java

*Servlet-API 3.1* conține specificațiile JSR 356 (*Java Specification Request*) pentru WebSocket. O implementare de referință a fost realizată de *Oracle* în pachetul *tyrus* inclusă în *glassfish 4*.

Pentru server două moduri de programare sunt definite :

- Prin utilizarea adnotărilor (*Annotation driven*);
- Fără utilizarea adnotărilor (*Interface driven*).

### 12.3.1 Programare prin adnotări

Șablonul de programare este

```
@ServerEndpoint(value="/urlPattern")
public class EndpointWebSocketServer {
    private static Set<Session> sessions =
        Collections.synchronizedSet(new HashSet<Session>());

    @OnMessage
    public void myTask(String msg, Session session)
        throws IOException, EncodeException{
        . . .
    }

    @OnOpen
    public void onOpen(Session session){
        sessions.add(session);
    }

    @OnClose
    public void onClose(Session session){
        sessions.remove(session);
    }
}
```

Metoda cu adnotareae `@OnMessage` asigură accesul la datele cererii unui client.

#### Interfața `javax.websocket.Session`

##### Metode

- `void addMessageHandler(MessageHandler handler)`

- `RemoteEndpoint.Basic getBasicRemote()`

Prin intermediul unui obiect de tip `RemoteEndpoint.Basic` se programează expedierea răspunsului către client.

- `RemoteEndpoint.Async getAsyncRemote()`

- `void close()`

Interfața `RemoteEndpoint` are subinterfețele

`RemoteEndpoint.Async`  
`RemoteEndpoint.Basic.`

## Interfața javax.websocket.RemoteEndpoint.Basic

### Metode

- void sendObject(Object data) throws IOException, EncodeException
- void sendText(String data) throws IOException
- void sendBinary(ByteBuffer data) throws IOException
- OutputStream getSendStream() throws IOException

Structura aplicației Web cu desfășurare în *glassfish* este

### AppSerser

```
|--> WEB-INF
|      |--> classes
|      |      |      *.class
|      |      index.html
```

**Exemplul 12.3.1** *Calculul celui mai mare divizor comun a două numere naturale.*

Aplicația *server* are codul

```
1 package websocket.cmmdc;
2 import javax.websocket.OnMessage;
3 import javax.websocket.server.ServerEndpoint;
4 import javax.websocket.RemoteEndpoint;
5 import javax.websocket.Session;
6 import javax.websocket.OnOpen;
7 import javax.websocket.OnClose;
8 import javax.websocket.EncodeException;
9 import java.io.IOException;
10 import java.util.Set;
11 import java.util.Collections;
12 import java.util.HashSet;
13
14 @ServerEndpoint(value="/cmmdc")
15 public class CmmdcWebSocketServer {
16     private static Set<Session> sessions =
17         Collections.synchronizedSet(new HashSet<Session>());
18
19     @OnMessage
20     public void myTask(String msg, Session session)
21         throws IOException, EncodeException{
22         String [] elem=msg.split(":");
23         long m=Long.parseLong(elem[0]);
24         long n=Long.parseLong(elem[1]);
25         long r=cmmdc(m,n);
26         for (Session peer : sessions) {
27             if(peer.equals(session)){
```

```

28         peer.getBasicRemote().sendText(new Long(r).toString());
29     }
30 }
31 }
32
33 public long cmmdc(long m, long n){ . . . }
34
35 @OnOpen
36 public void onOpen(Session session){
37     sessions.add(session);
38 }
39
40 @OnClose
41 public void onClose(Session session){
42     sessions.remove(session);
43 }
44 }

```

În Java 8 codul serverului poate fi

```

1 package websocket.cmmdc;
2 import javax.websocket.OnMessage;
3 import javax.websocket.server.ServerEndpoint;
4 import javax.websocket.Session;
5 import javax.websocket.OnOpen;
6 import javax.websocket.OnClose;
7 import javax.websocket.EncodeException;
8 import java.io.IOException;
9 import java.util.Set;
10 import java.util.Collections;
11 import java.util.HashSet;
12 import java.util.stream.Stream;
13 import javax.websocket.RemoteEndpoint;
14
15 @ServerEndpoint(value="/cmmdc")
16 public class CmmdcWebSocketServer{
17     interface CmmdcService {
18         long cmmdc(long m, long n);
19     }
20
21     static CmmdcService cmmdcService=(long m, long n) ->
22     {
23         long r,c;
24         do{
25             c=n;
26             r=m%n;
27             m=n;
28             n=r;
29         }
30         while (r!=0);
31         return c;
32     };
33
34     private static Set<Session> sessions =
35         Collections.synchronizedSet(new HashSet<Session>());
36
37     @OnMessage
38     public void myTask(String msg,Session session)
39         throws IOException ,EncodeException{

```



```

40     String [] elem=msg.split(":");
41     long m=Long.parseLong(elem[0]);
42     long n=Long.parseLong(elem[1]);
43     System.out.println(m+" : "+n);
44     long r=cmmdcService.cmmdc(m,n);
45     String rez=new Long(r).toString();
46     Stream<Session> stream=sessions.stream();
47     stream
48         .filter(s->s.equals(session))
49         .forEach(s->{
50             RemoteEndpoint.Basic endpoint=s.getBasicRemote();
51             try{
52                 endpoint.sendText(rez);
53             }
54             catch(IOException e){};
55         });
56 }

58 @OnOpen
59 public void onOpen(Session session){
60     sessions.add(session);
61 }

63 @OnClose
64 public void onClose(Session session){
65     sessions.remove(session);
66 }
67 }

```

Clientul HTML5/Javascript este

```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
5     <title>Cmmdc WebSocket</title>

7     <script language="javascript" type="text/javascript">
8       var wsUri = "ws://localhost:8080/CmmdcWebSocket/cmmdc";
9       var websocket = new WebSocket(wsUri);
10      websocket.onopen = function(evt) { onOpen(evt) };
11      websocket.onmessage = function(evt) { onMessage(evt) };
12      websocket.onerror = function(evt) { onError(evt) };
13      websocket.onclose = function(evt) { onClose(evt) };

15      function init() {
16        output = document.getElementById("result");
17      }

19      function send() {
20        var sm=document.cmmdc.m.value;
21        var sn=document.cmmdc.n.value;
22        var msg=sm+" : "+sn;
23        websocket.send(msg);
24        writeToScreen("SENT: " + msg);
25      }

27      function onOpen(evt) {
28        writeToScreen("CONNECTED");

```

```

29     }
30
31     function onMessage(evt) {
32         writeToScreen("RECEIVED: " + evt.data);
33     }
34
35     function onError(evt) {
36         writeToScreen('<span style="color: red;">ERROR:</span> ' +
37             evt.data);
38     }
39
40     function onClose(evt) {
41         writeToScreen("CLOSED");
42     }
43
44     function myclose(){
45         websocket.close();
46     }
47
48     function writeToScreen(message) {
49         var pre = document.createElement("p");
50         pre.style.wordWrap = "break-word";
51         pre.innerHTML = message;
52         output.appendChild(pre);
53     }
54
55     window.addEventListener("load", init, false);
56 </script>
57 </head>
58 <body>
59     <h1>CMMDC with WebSocket!!</h1>
60     <div style="text-align: center;">
61         <form name="cmmdc">
62             <table border="1">
63                 <tr>
64                     <td> Primul num&#259;r </td>
65                     <td> <input type="number" name="m" value="1"
66                         required min="1"/> </td>
67                 </tr>
68                 <tr>
69                     <td> Al doilea num&#259;r </td>
70                     <td> <input type="number" name="n" value="1"
71                         required min="1"/> </td>
72                 </tr>
73                 <tr>
74                     <td>
75                         <input type="button" value="Calculeaza" onclick="send()" />
76                     </td>
77                 </tr>
78                 <tr>
79                     <td>
80                         <input type="button" value="Inchide" onclick="myclose()" />
81                     </td>
82                 </tr>
83             </table>
84         </form>
85     </div>
86
87

```

```

88 |     <div id="result"></div>
89 | </body>
90 | </html>

```

### 12.3.2 Programare fără adnotări

Clasa `javax.websocket.Endpoint`

#### Metode

- `void onClose(Session session, CloseReason closeReason)`
- `void onError(Session session, Throwable th)`
- `abstract void onOpen(Session session, EndpointConfig config)`

Interfața `javax.websocket.MessageHandler.Whole<T>`

#### Metode

- `void onMessage(T message)`

În funcție de tipul mesajului `T` acesta poate fi

Mesaj text	Mesaj binar
String	ByteBuffer
Reader	byte[]
	InputStream

Pentru același exemplu, fără modificarea clientului componenta server va fi alcătuită din două clase:

- *MyApplicationConfig*  
în care se declară clasa care tratează solicitarea clientului și *numele de apel* (`urlPattern`);
- *CmmdcWebSocketEndpoint*  
clasa declarată anterior și care rezolvă solicitarea clientului.

Clasa extinde clasa `javax.websocket.Endpoint`, programatorul implementează metodele `onOpen`, `onClose`, `onError`. În metoda `onOpen`, se utilizează o clasă care implementează interfața `MessageHandler.Whole` prin care avem acces la parametrii cererii.

Codurile sunt:

```

1 import java.util.Set;
2 import javax.websocket.Endpoint;
3 import javax.websocket.server.ServerApplicationConfig;
4 import javax.websocket.server.ServerEndpointConfig;

6 public class MyApplicationConfig implements ServerApplicationConfig {

8     @Override
9     public Set<ServerEndpointConfig> getEndpointConfigs(
10         Set<Class<? extends Endpoint>> set) {
11         return new HashSet<ServerEndpointConfig>() {
12             {
13                 add(ServerEndpointConfig.Builder
14                     .create(CmmdcWebSocketEndpoint.class, "/cmmdc")
15                     .build());
16             }
17         };
18     }

20     @Override
21     public Set<Class<?>> getAnnotatedEndpointClasses(Set<Class<?>> set){
22         return Collections.emptySet();
23     }
24 }

```

și

```

1 package websocket.cmmdc;
2 import javax.websocket.CloseReason;
3 import javax.websocket.RemoteEndpoint;
4 import javax.websocket.Session;
5 import javax.websocket.Endpoint;
6 import javax.websocket.EndpointConfig;
7 import javax.websocket.MessageHandler;
8 import javax.websocket.EncodeException;
9 import java.io.IOException;

11 public class CmmdcWebSocketEndpoint extends Endpoint {

13     public void onOpen(Session session, EndpointConfig config) {
14         final RemoteEndpoint.Basic remote = session.getBasicRemote();
15         session.addMessageHandler(new MessageHandler.Whole<String>() {
16             public void onMessage(String msg) {
17                 try {
18                     String[] elem=msg.split(":");
19                     long m=Long.parseLong(elem[0]);
20                     long n=Long.parseLong(elem[1]);
21                     System.out.println(m+" : "+n);
22                     long r=cmmdc(m,n);
23                     remote.sendText(new Long(r).toString());
24                 }
25                 catch (Exception ioe) { }
26             }
27         });
28     }

30     public void onClose(Session session, CloseReason closeReason) {
31         System.out.println("Closing: " + closeReason.getReasonPhrase());

```

```

32 }
34 public void onError (Session session , Throwable throwable) {
35     System.out.println("Error: " + throwable.getLocalizedMessage());
36 }
38 public long cmmdc(long m, long n){. . .}
39 }

```

### 12.3.3 Client Java pentru WebSocket

Programarea unui client Java necesită

- clasa `javax.websocket.Endpoint`
- interfața `javax.websocket.Session`

și în plus

Clasa `javax.websocket.ContainerProvider`

Metode

- `public static WebSocketContainer getWebSocketContainer()`

Interfața `javax.websocket.WebSocketContainer`

Metode

- `Session connectToServer(Endpoint endpointInstance, ClientEndpointConfig cec, URI path) throws DeploymentException, IOException`

Codul unui client Java pentru una din aplicațiile anterioare este

```

1 import java.io.IOException;
2 import java.net.URI;
3 import java.util.Scanner;
4 import java.net.URISyntaxException;
5 import javax.websocket.ContainerProvider;
6 import javax.websocket.DeploymentException;
7 import javax.websocket.WebSocketContainer;
8 import javax.websocket.Session;
9 import javax.websocket.Endpoint;
10 import javax.websocket.EndpointConfig;
11 import javax.websocket.MessageHandler;
13 public class WebSocketClient extends Endpoint{
14     private static boolean sfarsit=false;
15     private static String SERVER =

```

```

16      "ws://localhost:8080/CmmdcWebSocketAD/cmmdc" ;
18      public static void main(String[] args){
19          Scanner scanner=new Scanner(System.in);
20          System.out.println("m=");
21          long m=scanner.nextLong();
22          String sm=new Long(m).toString();
23          System.out.println("n=");
24          long n=scanner.nextLong();
25          String sn=new Long(n).toString();
26          String data=sm+"."+sn;
27          WebSocketContainer container =
28              ContainerProvider.getWebSocketContainer();
29
30          try {
31              Session session = container.connectToServer(
32                  WebSocketClient.class, null, new URI(SERVER));
33              session.getBasicRemote().sendText(data);
34              while(! sfarsit){System.out.println(" ");};
35              session.close();
36          }
37          catch(Exception ex){
38              System.out.println("LocalEndPoint Exception : "+ex.getMessage());
39          }
40      }
41
42      public void onOpen(Session session, EndpointConfig config) {
43          session.addMessageHandler(new MessageHandler.Whole<String>() {
44              public void onMessage(String text){
45                  System.out.println("Cmmdc : "+text);
46                  sfarsit=true;
47              }
48          });
49      }
50  }

```

### 12.3.4 Transmiterea datelor prin adnotarea PathParam

În varianta cu adnotare utilizând `ServerEndpoint` de forma

```
@ServerEndpoint("/urlPattern/{param}")
```

un parametru dat este accesibil prin

```

@OnOpen
public void onOpen(Session session,
    @PathParam("param") String param) {
    . . .
}

```

### Exemplul 12.3.2

```

1 package websocket.cmmdc;
2 import javax.websocket.OnMessage;
3 import javax.websocket.server.ServerEndpoint;
4 import javax.websocket.Session;
5 import javax.websocket.OnOpen;
6 import javax.websocket.OnClose;
7 import javax.websocket.OnError;
8 import javax.websocket.RemoteEndpoint;
9 import javax.websocket.EncodeException;
10 import javax.websocket.server.PathParam;
11 import java.io.IOException;
12 import java.util.Set;
13 import java.util.Collections;
14 import java.util.HashSet;

16 @ServerEndpoint(value="/cmmdc/{msg}")
17 public class CmmdcWebSocketServer{
18     private static Set<Session> sessions =
19         Collections.synchronizedSet(new HashSet<Session>());

21     interface CmmdcService {
22         long cmmdc(long m, long n);
23     }

25     static CmmdcService cmmdcService=(long m, long n) -> { . . . };

27     @OnMessage
28     public void onMessage(String message, Session session){}

30     @OnOpen
31     public void onOpen(Session session, @PathParam("msg") String msg)
32         throws IOException, EncodeException{
33         sessions.add(session);
34         String [] elem=msg.split(":");
35         long m=Long.parseLong(elem[0]);
36         long n=Long.parseLong(elem[1]);
37         long r=cmmdcService.cmmdc(m,n);
38         String rez=new Long(r).toString();
39         sessions.stream()
40             .filter(s->s.equals(session))
41             .forEach(s->{
42                 RemoteEndpoint.Basic endpoint=s.getBasicRemote();
43                 try{
44                     endpoint.sendText(rez);
45                 }
46                 catch(IOException e){}
47             });
48     }

50     @OnClose
51     public void onClose(Session session){
52         sessions.remove(session);
53     }
54 }

```

cu apelarea din

```

1 <!DOCTYPE html>
2 <html>

```

```

3  <head>
4  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
5  <title>Cmmdc WebSocket</title>

7  <script language="javascript" type="text/javascript">
8      var websocket;

10     function init() {
11         output = document.getElementById("result");
12     }

14     function send() {
15         var sm=document.cmmdc.m.value;
16         var sn=document.cmmdc.n.value;
17         var msg=sm+":"+sn;
18         var wsUri = "ws://localhost:8080/CmmdcWebSocketAD/cmmdc/"+msg;
19         websocket = new WebSocket(wsUri);
20         websocket.onopen = function(evt) { onOpen(evt) };
21         websocket.onmessage = function(evt) { onMessage(evt) };
22         websocket.onerror = function(evt) { onError(evt) };
23         websocket.onclose = function(evt) { onClose(evt) };
24     }

26     function myclose(){
27         websocket.close();
28     }

30     function onOpen(evt) {
31         writeToScreen("CONNECTED");
32     }

34     function onMessage(evt) {
35         writeToScreen("RECEIVED: " + evt.data);
36     }

38     function onError(evt) {
39         writeToScreen('<span style="color: red;">ERROR:</span> ' +
40             evt.data);
41     }

43     function onClose(evt) {
44         writeToScreen("CLOSED");
45     }

47     function writeToScreen(message) {
48         var pre = document.createElement("p");
49         pre.style.wordWrap = "break-word";
50         pre.innerHTML = message;
51         output.appendChild(pre);
52     }

54     window.addEventListener("load", init, false);
55 </script>
56 </head>
57 <body>
58 <h1>CMMDc with WebSocket!!</h1>
59 <div style="text-align: center;">
60     <form name="cmmdc">
61         <table border="1">

```



```

62     <tr>
63         <td> Primul num&#259;r </td>
64         <td> <input type="number" name="m" value="1"
65             required min="1" /> </td>
66     </tr>
67     <tr>
68         <td> Al doilea num&#259;r </td>
69         <td> <input type="number" name="n" value="1"
70             required min="1" /> </td>
71     </tr>
72     <tr>
73         <td>
74             <input type="button" value="Calculeaza" onclick="send()" />
75         </td>
76     </tr>
77     <tr>
78         <td>
79             <input type="button" value="Inchide" onclick="myclose()" />
80         </td>
81     </tr>
82 </table>
83 </form>
84 </div>
85 <div id="result"></div>
86 </body>
87 </html>

```

### 12.3.5 Conversie și deconversie prin JSON / XML

Interfața de programare pentru WebSocket oferă posibilitatea conversiei unei componente Java într-un obiect JSON sau document XML<sup>1</sup> și transformarea inversă.

Programatorul trebuie să

- Implementeze interfețele `Encoder.Text<T>` și `Decoder.Text<T>`;
- Declare clasele de conversie și deconversie în adnotarea `@ServerEndpoint`, sub forma

```

@ServerEndpoint(value="/numeApel",
    encoders={ClasaConversie.class, . . .},
    decoders={ClasaDeconversie.class, . . .})

```

Șablonul de implementare al interfeței `Encoder.Text` este

<sup>1</sup>Parcursarea acestei secțiuni necesită familiarizarea cu *Java Script Object Notation* JSON și JAXB. Pentru JSON se va utiliza pachetul `javax.json` din JEE.

```

import javax.websocket.Encoder;
import javax.websocket.EndpointConfig;
import javax.websocket.EncodeException;

public class MyEncoder implements Encoder.Text<T> {
    @Override
    public void init(EndpointConfig ec) { }

    @Override
    public void destroy() { }

    @Override
    public String encode(T obj) throws EncodeException{
        // Cod de conversie al campurilor obiectului obj in JSON
        return objJsonString;
    }
}

```

Șablonul de implementare al interfeței `Decoder.Text` este

```

import javax.websocket.Decoder;
import javax.websocket.EndpointConfig;
import javax.websocket.DecodeException;

public class MyDecoder implements Decoder.Text<T>{
    @Override
    public void init(EndpointConfig ec) { }

    @Override
    public void destroy() { }

    @Override
    public T decode(String string) throws DecodeException{
        // Instantierea unui obiect de tip T cu datele din string.
    }

    @Override
    public boolean willDecode(String string) {
        return true;
    }
}

```

### Exemplul 12.3.3

Fie componenta Java

```

1 import javax.xml.bind.annotation.XmlRootElement;
2
3 @XmlRootElement(name="date")
4 public class CmmdcBean{
5     private long m;
6     private long n;
7     public void setM(long m){
8         this.m=m;
9     }
10
11     public long getM(){
12         return m;

```

```

13 }
15 public void setN(long n){
16     this.n=n;
17 }
19 public long getN(){
20     return n;
21 }
22 }

```

Aplicația server utilizează doar decodorul - transformarea din JSON / XML în obiect de tip *CmmdcBean*. Transformarea este realizată de serverul Web. Codul serverului este

```

1 package websocket.cmmdc;
2 import javax.websocket.OnMessage;
3 import javax.websocket.server.ServerEndpoint;
4 import javax.websocket.Session;
5 import javax.websocket.OnOpen;
6 import javax.websocket.OnClose;
7 import javax.websocket.EncodeException;
8 import java.io.IOException;
9 import java.util.Set;
10 import java.util.Collections;
11 import java.util.HashSet;
12
13 @ServerEndpoint(value="/cmmdc",
14     decoders={CmmdcBeanXMLDecoder.class, CmmdcBeanJSONDecoder.class})
15 public class CmmdcWebSocketServer{
16     private static Set<Session> sessions =
17         Collections.synchronizedSet(new HashSet<Session>());
18
19     @OnMessage
20     public void myTask(Session session, CmmdcBean obj)
21         throws IOException, EncodeException{
22         long m=obj.getM();
23         long n=obj.getN();
24         long r=cmmdc(m,n);
25         for (Session peer : sessions) {
26             if(peer.equals(session)){
27                 peer.getBasicRemote().sendText(new Long(r).toString());
28             }
29         }
30     }
31
32     @OnOpen
33     public void onOpen(Session session){
34         sessions.add(session);
35     }
36
37     @OnClose
38     public void onClose(Session session){
39         sessions.remove(session);
40     }
41
42     public long cmmdc(long m, long n){. . .}
43 }

```

Codul decodorului din JSON este

```

1 package websocket.cmmdc;
2 import javax.websocket.Decoder;
3 import javax.websocket.EndpointConfig;
4 import javax.websocket.DecodeException;
5 import javax.json.Json;
6 import javax.json.JsonArray;
7 import javax.json.JsonObject;
8 import javax.json.JsonReader;
9 import java.io.StringReader;

11 public class CmmdcBeanJSONDecoder implements Decoder.Text<CmmdcBean>{
12     @Override
13     public void init(EndpointConfig ec) { }

15     @Override
16     public void destroy() { }

18     @Override
19     public CmmdcBean decode(String string) throws DecodeException{
20         StringReader sr=new StringReader(string);
21         JsonReader jsonReader = Json.createReader(sr);
22         JsonObject obj=jsonReader.readObject();
23         jsonReader.close();
24         long m=0,n=0;
25         String sm=obj.getString("m");
26         m=Long.parseLong(sm);
27         String sn=obj.getString("n");
28         n=Long.parseLong(sn);
29         CmmdcBean bean=new CmmdcBean();
30         bean.setM(m);
31         bean.setN(n);
32         return bean;
33     }

35     @Override
36     public boolean willDecode(String string) {
37         return true;
38     }
39 }

```

iar codul decodorului din XML este

```

1 package websocket.cmmdc;
2 import javax.websocket.Decoder;
3 import javax.websocket.EndpointConfig;
4 import javax.websocket.DecodeException;
5 import javax.xml.bind.JAXBContext;
6 import javax.xml.bind.Unmarshaller;
7 import java.io.StringReader;

9 public class CmmdcBeanXMLDecoder implements Decoder.Text<CmmdcBean>{
10     @Override
11     public void init(EndpointConfig ec) { }

13     @Override
14     public void destroy() { }

16     @Override

```

```

17 public CmmdcBean decode(String string) throws DecodeException{
18     StringReader sr=new StringReader(string);
19     CmmdcBean bean=null;
20     JAXBContext jaxbContext;
21     try{
22         jaxbContext=JAXBContext.newInstance(CmmdcBean.class);
23         Unmarshaller unmarshaller=jaxbContext.createUnmarshaller();
24         bean=(CmmdcBean) unmarshaller.unmarshal(sr);
25     }
26     catch(Exception e){
27         e.printStackTrace();
28     }
29     return bean;
30 }

32 @Override
33 public boolean willDecode(String string) {
34     return (string!=null);
35 }
36 }

```

Clasa clientului utilizează doar conversia datelor în obiect JSON sau XML. Codorul în JSON este

```

1 import javax.websocket.Encoder;
2 import javax.websocket.EndpointConfig;
3 import javax.websocket.EncodeException;
4 import javax.json.Json;
5 import javax.json.JsonObject;

7 public class CmmdcBeanJSONEncoder implements Encoder.Text<CmmdcBean> {
8     @Override
9     public void init(EndpointConfig ec) { }

11     @Override
12     public void destroy() { }

14     @Override
15     public String encode(CmmdcBean obj) throws EncodeException{
16         long m=obj.getM();
17         long n=obj.getN();
18         JsonObject jsonObject=Json.createObjectBuilder()
19             .add("m",new Long(m).toString())
20             .add("n",new Long(n).toString())
21             .build();
22         String objJsonString=jsonObject.toString();
23         return objJsonString;
24     }
25 }

```

iar codorul în XML este

```

1 import javax.websocket.Encoder;
2 import javax.websocket.EndpointConfig;
3 import javax.websocket.EncodeException;
4 import javax.xml.bind.JAXBContext;
5 import javax.xml.bind.Marshaller;
6 import java.io.StringWriter;

```

```

8 public class CmmdcBeanXMLEncoder implements Encoder.Text<CmmdcBean> {
9     @Override
10    public void init(EndpointConfig ec) { }

12    @Override
13    public void destroy() { }

15    @Override
16    public String encode(CmmdcBean obj) throws EncodeException{
17        JAXBContext jaxbContext=null;
18        StringWriter st=null;
19        try{
20            jaxbContext=JAXBContext.newInstance(CmmdcBean.class);
21            Marshaller marshaller=jaxbContext.createMarshaller();
22            st=new StringWriter();
23            marshaller.marshal(obj,st);
24            System.out.println("Results : "+st.toString());
25        }
26        catch(Exception e){
27            e.printStackTrace();
28        }
29        return st.toString();
30    }
31 }

```

Codul clasei client este

```

1 import java.net.URI;
2 import javax.websocket.ContainerProvider;
3 import javax.websocket.WebSocketContainer;
4 import javax.websocket.Session;
5 import java.util.Scanner;
6 import javax.websocket.Endpoint;
7 import javax.websocket.EndpointConfig;
8 import javax.websocket.MessageHandler;

10 public class WebSocketClient extends Endpoint{
11     private static boolean sfarsit=false;

13     public static void main(String[] args){
14         Scanner scanner=new Scanner(System.in);
15         System.out.println("m=");
16         long m=scanner.nextLong();
17         System.out.println("n=");
18         long n=scanner.nextLong();
19         CmmdcBean obj=new CmmdcBean();
20         obj.setM(m);
21         obj.setN(n);
22         int tip=1;
23         String server="";
24         do{
25             System.out.println("Encoder Type");
26             System.out.println("1: JSON");
27             System.out.println("2: XML");
28             tip=scanner.nextInt();
29         }
30         while((tip!=1) && (tip!=2));

32         WebSocketContainer container =

```

```

33     ContainerProvider.getWebSocketContainer();
34     String request=null;
35     try {
36         if (tip==1){
37             server="ws://localhost:8080/JsonCmmdcWebSocket/cmmdc";
38             CmmdcBeanJSONEncoder encoderJSON=new CmmdcBeanJSONEncoder();
39             request=encoderJSON.encode(obj);
40         }
41         else{
42             server="ws://localhost:8080/XmlCmmdcWebSocket/cmmdc";
43             CmmdcBeanXMLEncoder encoderXML=new CmmdcBeanXMLEncoder();
44             request=encoderXML.encode(obj);
45         }
46         System.out.println(request);
47         Session session=
48             container.connectToServer(WebSocketClient.class, null,
49                 new URI(server));
50         session.getBasicRemote().sendText(request);
51         while(! sfarsit){};
52     }
53     catch(Exception ex){
54         System.out.println("LocalEndPoint Exception : "+ex.getMessage());
55     }
56 }

58 public void onOpen(Session session, EndpointConfig config) {
59     session.addMessageHandler(new MessageHandler.Whole<String>() {
60         public void onMessage(String text){
61             System.out.println("Cmmdc : "+text);
62             sfarsit=true;
63             System.exit(0);
64         }
65     });
66 }
67 }

```

## Verificare prin *tyrus-client*

Verificarea funcționării unui server WebSocket se poate realiza cu aplicația *tyrus-client-cli-\*.jar* prin

```
java -jar tyrus-client-cli-*.jar ws://host:8080/context/numeApel
```

Comenzile care se pot utiliza se obțin prin comanda **help**.

În cazul aplicației *CmmdcWebSocketAD*, dacă  $m = 56$  și  $n = 42$  atunci se va cere

```

send 56:42
close
quit

```

### 12.3.6 Streaming

Prin socluri se pot transmite fluxuri de date (*streaming*). Detalii de programare depind de natura datelor ce trebuie vehiculate (text, sunet, grafică).

Expedierea datelor de către server se realizează prin intermediul unui flux dat de metoda

`OutputStream getSendStream()`

a clasei `javax.websocket.RemoteEndpoint.Basic`.

**Exemplul 12.3.4** *Clientul solicită serverului transmisia unui fișier text, cerere care va fi satisfăcută de server. Clientul cunoaște lista fișierelor pe care le poate trimite serverul.*

Atât pentru programul server cât și pentru programul client indicăm două soluții care se corespund.

Clasa server *TextStreaming*

```

1 import javax.websocket.OnMessage;
2 import javax.websocket.server.ServerEndpoint;
3 import javax.websocket.Session;
4 import javax.websocket.OnOpen;
5 import javax.websocket.OnClose;
6 import javax.websocket.EncodeException;
7 import java.io.IOException;
8 import java.io.OutputStream;
9 import java.util.Set;
10 import java.util.Collections;
11 import java.util.HashSet;
12 import java.nio.file.Paths;

14 // Varianta 1
15 /*
16 import java.io.File;
17 import java.io.FileReader;
18 import java.io.BufferedReader;
19 import java.io.DataOutputStream;
20 */
21 // Varianta 2
22 import java.nio.file.Path;
23 import java.nio.file.Files;

25 @ServerEndpoint(value="/text")
26 public class TextStreaming{
27     private static Set<Session> sessions =
28         Collections.synchronizedSet(new HashSet<Session>());

30     @OnMessage
31     public void myTask(String msg,Session session)
32         throws IOException,EncodeException{
33         String fs=System.getProperty("file.separator");
34         String pathServer=new java.io.File(".").getCanonicalPath();

```



```

35     int x=pathServer.indexOf("glassfish");
36     String path="";
37     String path0=fs+"TextStreaming"+
38         fs+"WEB-INF"+fs+"text"+fs;
39     if(x== -1){
40         // apache-tomcat
41         path=Paths.get(pathServer).toString()+
42             fs+"webapps"+path0;
43     }
44     else{
45         // glassfish
46         x=pathServer.lastIndexOf(fs);
47         String pathGlassfish=pathServer.substring(0,x);
48         path=Paths.get(pathGlassfish).toString()+
49             fs+"applications"+path0;
50     }
51     int noFile=Integer.parseInt(msg);
52     String fileName="";
53     switch(noFile){
54     case 1:
55         fileName="capitol.txt";
56         break;
57     case 2:
58         fileName="junit.tex";
59         break;
60     }
61     System.out.println(path+fileName);
62     // Varianta 1
63     /*
64     File inputFile=new File(path+fileName);
65     FileReader fr=new FileReader(inputFile);
66     BufferedReader br= new BufferedReader(fr);
67     for (Session peer : sessions) {
68         if(peer.equals(session)){
69             OutputStream out=peer.getBasicRemote().getSendStream();
70             DataOutputStream dos=new DataOutputStream(out);
71             String s;
72             while((s=br.readLine())!=null) dos.writeUTF(s);
73             dos.writeUTF("endOfFile");
74             dos.flush();
75             br.close();
76             fr.close();
77             dos.close();
78         }
79     }
80     */

82     // Varianta 2
83     Path cale=Paths.get(path+fileName);
84     for (Session peer : sessions) {
85         if(peer.equals(session)){
86             OutputStream out=peer.getBasicRemote().getSendStream();
87             Files.copy(cale, out);
88             out.close();
89         }
90     }
91 }

93 @OnOpen

```

```

94 public void onOpen(Session session){
95     sessions.add(session);
96 }
97
98 @OnClose
99 public void onClose(Session session){
100     sessions.remove(session);
101 }
102 }

```

### Clasa clientului Java *WebSocketClient*

```

1 import java.io.IOException;
2 import java.net.URI;
3 import javax.websocket.ContainerProvider;
4 import javax.websocket.DeploymentException;
5 import javax.websocket.WebSocketContainer;
6 import javax.websocket.Session;
7 import java.util.Scanner;
8 import javax.websocket.Endpoint;
9 import javax.websocket.EndpointConfig;
10 import javax.websocket.MessageHandler;
11 import java.io.DataInputStream;
12 import java.io.InputStream;
13 import java.io.ByteArrayInputStream;

14
15 public class WebSocketClient extends Endpoint{
16     private static boolean sfarsit=false;
17     private static String SERVER="ws://localhost:8080/TextStreaming/text";

18
19     public static void main(String [] args){
20         Scanner scanner=new Scanner(System.in);
21         System.out.println("Fisierul de accesat:");
22         System.out.println("1: capitol.txt");
23         System.out.println("2: junit.tex");
24         int noFile=scanner.nextInt();
25         String data=new Integer(noFile).toString();
26         WebSocketContainer container =
27             ContainerProvider.getWebSocketContainer();
28         try {
29             Session session=
30                 container.connectToServer(WebSocketClient.class, null,
31                     new URI(SERVER));
32             session.getBasicRemote().sendText(data);
33             while(! sfarsit){};
34         }
35         catch(Exception ex){
36             System.out.println("LocalEndPoint Exception : "+ex.getMessage());
37         }
38     }

39
40     // Varianta 1
41     /*
42     public void onOpen(Session session, EndpointConfig config){
43         session.addMessageHandler(new MessageHandler.Whole<byte[]>() {
44             public void onMessage(byte [] msg){
45                 try(InputStream is=new ByteArrayInputStream(msg);
46                     DataInputStream in = new DataInputStream(is)){
47                     String s;

```

```

48         while (!(s=in.readUTF()).equals("endOfFile"))
49             System.out.println(s);
50     }
51     catch(IOException e){
52         e.printStackTrace();
53     }
54     sfarsit=true;
55 }
56 });
57 }
58 */
59
60 // Variana 2
61 public void onOpen(Session session, EndpointConfig config){
62     session.addMessageHandler(new MessageHandler.Whole<byte[]>() {
63         public void onMessage(byte[] msg){
64             System.out.println(new String(msg));
65             sfarsit=true;
66         }
67     });
68 }
69 }

```

**Exemplul 12.3.5** *Clientul solicită serverului transmiterea unui fișier grafic, cerere care va fi satisfăcută de server. Clientul cunoaște lista fișierelor pe care le poate trimite serverul.*

Clasa server *ImageStreaming*

```

1  import javax.websocket.OnMessage;
2  import javax.websocket.server.ServerEndpoint;
3  import javax.websocket.Session;
4  import javax.websocket.OnOpen;
5  import javax.websocket.OnClose;
6  import javax.websocket.EncodeException;
7  import java.io.IOException;
8  import java.io.OutputStream;
9  import java.util.Set;
10 import java.util.Collections;
11 import java.util.HashSet;
12 import java.nio.file.Paths;
13 import java.nio.file.Path;
14 import java.nio.file.Files;
15
16 @ServerEndpoint(value="/image")
17 public class ImageStreaming{
18     private static Set<Session> sessions =
19         Collections.synchronizedSet(new HashSet<Session>());
20
21     @OnMessage
22     public void myTask(String msg, Session session)
23         throws IOException, EncodeException{
24         String fs=System.getProperty("file.separator");
25         String pathServer=new java.io.File(".").getCanonicalPath();
26         int x=pathServer.indexOf("glassfish");
27         String path="";

```

```

28     String path0=fs+"ImageStreaming"+
29         fs+"WEB-INF"+fs+"images"+fs;
30     if(x==1){
31         // apache-tomcat
32         path=Paths.get(pathServer).toString()+
33         fs+"webapps"+path0;
34     }
35     else{
36         // glassfish
37         x=pathServer.lastIndexOf(fs);
38         String pathGlassfish=pathServer.substring(0,x);
39         path=Paths.get(pathGlassfish).toString()+
40         fs+"applications"+path0;
41     }
42     int noFile=Integer.parseInt(msg);
43     String fileName="";
44     switch(noFile){
45     case 1:
46         fileName="brasov.jpg";
47         break;
48     case 2:
49         fileName="xml-pic.jpg";
50         break;
51     }
52     Path cale=Paths.get(path+fileName);
53     for (Session peer : sessions) {
54         if(peer.equals(session)){
55             OutputStream out=peer.getBasicRemote().getSendStream();
56             Files.copy(cale,out);
57             out.close();
58         }
59     }
60 }

62 @OnOpen
63 public void onOpen(Session session){
64     sessions.add(session);
65 }

67 @OnClose
68 public void onClose(Session session){
69     sessions.remove(session);
70 }
71 }

```

### Clasa client *WebSocketClient.java*

```

1  import java.io.IOException;
2  import java.net.URI;
3  import javax.websocket.ContainerProvider;
4  import javax.websocket.DeploymentException;
5  import javax.websocket.WebSocketContainer;
6  import javax.websocket.Session;
7  import java.util.Scanner;

9  import javax.websocket.Endpoint;
10 import javax.websocket.EndpointConfig;
11 import javax.websocket.MessageHandler;

```

```

13 import java.awt.image.BufferedImage;
14 import javax.imageio.ImageIO;
15 import java.io.InputStream;
16 import java.io.ByteArrayInputStream;
17 import java.awt.Image;

19 public class WebSocketClient extends Endpoint{
20     private static boolean sfarsit=false;
21     private static String SERVER =
22         "ws://localhost:8080/ImageStreaming/image";

24     public static void main(String[] args){
25         Scanner scanner=new Scanner(System.in);
26         System.out.println(" Fisierul de descarcat:");
27         System.out.println(" 1: Imagine din Brasov");
28         System.out.println(" 2: pic-xml");
29         int noFile=scanner.nextInt();
30         String data=new Integer(noFile).toString();
31         WebSocketContainer container =
32             ContainerProvider.getWebSocketContainer();

34         try {
35             Session session=
36                 container.connectToServer(WebSocketClient.class, null,
37                     new URI(SERVER));
38             session.getBasicRemote().sendText(data);
39             while(! sfarsit){};
40         }
41         catch(Exception ex){
42             System.out.println("LocalEndPoint Exception : "+ex.getMessage());
43         }
44     }

46     public void onOpen(Session session, EndpointConfig config){
47         session.addMessageHandler(new MessageHandler.Whole<byte[]>() {
48             public void onMessage(byte[] msg){
49                 try{
50                     InputStream in=new ByteArrayInputStream(msg);
51                     BufferedImage bi=ImageIO.read(in);
52                     Image image=(Image)bi;
53                     ShowImage s=new ShowImage(image);
54                     s.show();
55                 }
56                 catch(IOException e){
57                     e.printStackTrace();
58                 }
59                 sfarsit=true;
60             }
61         });
62     }
63 }

```

Clasa *ShowImage* afișează imaginea descărcată.



## Capitolul 13

### *Enterprise Java Beans*

În prezent se evidențiază două abordări privind realizarea aplicațiilor de complexitate mai mare (*aplicațiilor de întreprindere*):

- *Java Enterprise Edition - JEE* este o extensie a interfeței de programare (*API*) Java, pentru care există mai multe implementări. Principalul promotor este al modelului este *Oracle*, dar este susținut și de *RedHat* - *JBoss*;
- *Spring* a cărei dezvoltare ține de firma *VMware*. *Spring* este alcătuit din mai multe cadre de lucru. Acest model de dezvoltare este susținut de *Google*.

*Java Enterprise Edition* este un cadru de lucru care înglobează pe o serie implementări ale interfețelor de programare (*JDBC, JMS, JNDI, JSF, RMI-IIOP, JPA, JTA, JAAS, etc*), resurse care sunt utilizate prin intermediul unei componente *Enterprise Java Bean- EJB*.

O componentă EJB este o clasă Java - de obicei de tip *POJO* - care face parte din aplicația server, conține metodele care rezolvă o problemă (*business logic*) și este conținut într-un server de aplicații. Serverul de aplicații asigură unei componente EJB o serie de funcționalități, ca injectarea dependențelor, conexiunea cu bazele de date, gestiunea tranzacțiilor, etc. Serverul de aplicații poate fi interpretat și ca un container de componente EJB.

Se spune că serverul de aplicație JEE asigură accesul la contextul în care rulează aplicația, adică la resursele serverului, și la injectarea dependențelor (*Context and Dependency Injection - CDI*).

Servere de aplicații gratuite:

- *glassfish* - Oracle.

- *geronimo* - apache, sprijinit de I.B.M.
- *JOnAS*.

*Glassfish*, *geronimo* sunt implementări ale extensiei *Java Enterprise Edition* - *JEE*. În același timp, aceste servere Web sunt containere EJB, de servlet și JSP. În cele ce urmează vom utiliza *glassfish*.

Tipuri de componente EJB:

- *Session* - sesiune EJB.
- *Message Driven* - preia mesaje de tipul Java Message Service (JMS). Vom folosi prescurtarea MDB - Message Driven Bean.
- *Entity*

## 13.1 Session EJB

Starea unui obiect Java este dată de valorile câmpurilor sale. Din punctul de vedere al reținerii stării, există următoarele tipuri de componente sesiune EJB:

- *stateless* - fără reținerea stării;
- *stateful* - cu reținerea stării pe parcursul sesiunii serverului;
- *singleton* - există o singură instanță a componentei EJB. Durata de viață a componentei coincide cu intervalul de timp în care componenta EJB este activă în serverul de aplicații.

Aplicațiile cu componentă EJB constau din:

- componenta EJB - desfășurată în serverul de aplicații. Această componentă poate fi totodată și un serviciu Web de tip JAX-WS;
- aplicație client care poate fi
  - client Web - servlet sau client al serviciului Web de tip JAX-WS;
  - client RMI-IIOP (Internet Inter ORB Protocol).



În terminologia JEE componenta EJB, clientul Web și clientul RMI-IIOP formează câte un modul. Componenta EJB și clientul Web formează o aplicație JEE care se instalează în serverul de aplicații JEE.

Modulele EJB și RMI-IIOP se arhivează cu extensia **jar** iar modulul Web se arhivează cu extensia **war**. Aplicația JEE se arhivează cu extensia **ear** -*Enterprise ARchive*- sau **war**.

Un client RMI-IIOP apelează modulul EJB prin intermediul programului `glassfish\bin\appclient.exe`

```
appclient -client modul-iiop.jar [-targetserver host:port] arg1 . . .
```

Portul implicit este 3700.

### 13.1.1 Componentă EJB sesiune stateless

Șablonul pentru crearea unei componente EJB de tip *stateless session* este

```
import javax.ejb.Stateless;

@Stateless
public class Componenta{
    public tip metoda(. . .){. . .}
    . . .
}
```

**Exemplul 13.1.1** *Cel mai mare divizor comun a două numere naturale.*

Componentei EJB are codul

```
1 package cmmdc.ejb;
2 import javax.ejb.Stateless;

4 @Stateless
5 public class CmmdcBean{
6     public long cmmdc(long m, long n){. . .}
7 }
```

Aplicația client va fi un servlet găzduit de același server de aplicație. Accesul la componenta EJB se poate programa:

- prin injectare, utilizând adnotări;
- prin JNDI.

În ambele cazuri serverul Web este responsabil de instanțierea componentei EJB.

## Injectare cu adnotare

Serverul de aplicație *injectează* în servlet o instanță a componentei EJB sau altfel explicat

@EJB		@EJB(name="xyz")
Tip var;		Tip var;

injectează (realizează o referință) pentru *var* către un obiect de tipul *Tip* specificat. Acțiunea este declanșată de adnotarea EJB.

Codul servlet-ului este

```

1 package cmmdc.web;
2 import java.io.IOException;
3 import java.io.PrintWriter;
4 import javax.servlet.ServletException;
5 import javax.servlet.http.HttpServlet;
6 import javax.servlet.http.HttpServletRequest;
7 import javax.servlet.http.HttpServletResponse;
8 import cmmdc.ejb.CmmdcBean;
9 import javax.ejb.EJB;
10 import javax.servlet.annotation.WebServlet;

12 @WebServlet(urlPatterns = "/cmmdc")
13 public class CmmdcServlet extends HttpServlet{
14     @EJB
15     CmmdcBean cb;

17     public void doGet(HttpServletRequest req,HttpServletResponse res)
18         throws ServletException,IOException{
19         String sm=req.getParameter("m"),sn=req.getParameter("n");
20         long m=Long.parseLong(sm),n=Long.parseLong(sn);
21         long x=cb.cmmdc(m,n);
22         PrintWriter out=res.getWriter();

24         String title="Cmmdc Servlet";
25         res.setContentType("text/html");
26         out.println("<HTML><HEAD><TITLE>");
27         out.println(title);
28         out.println("</TITLE></HEAD><BODY>");
29         out.println("<H1>"+title+"</H1>");
30         out.println("<P>Cmmdc : "+x);
31         out.println("</BODY></HTML>");

33         out.close();
34     }

36     public void doPost(HttpServletRequest req,HttpServletResponse res)
37         throws ServletException,IOException{
38         doGet(req,res);
39     }
40 }
```

## Desfășurarea aplicației

```

ejbcmmdc
|--> WEB-INF
|   |--> classes
|   |   |--> cmmdc
```

```

|   |   |   |--> ejb
|   |   |   |   CmmdcBean.class
|   |   |   |--> web
|   |   |   |   CmmdcServlet.class
|   index.html

```

Această structură se arhivează cu `jar`, dar cu extensia `war`.

## Acces la componenta EJB prin JNDI

Referința JNDI are structura

*prefix* / [*nume\_app*] / [*nume\_modul*] / *nume\_EJB*

unde *prefix* poate fi

Prefix	Vizibilitatea rezultatului
java:global	Componenta se poate utiliza de oriunde. Coincide cu numele arhivei <code>ear</code> .
java:app	Componenta se poate utiliza în aplicație. Coincide cu numele arhivei <code>jar</code> sau <code>war</code> .
java:module	Componenta se poate utiliza în modul

În cazul exemplului codul servlet-ului va fi

```

1 package cmmdc.web;
2 import java.io.IOException;
3 import java.io.PrintWriter;
4 import javax.servlet.ServletException;
5 import javax.servlet.ServletConfig;
6 import javax.servlet.http.HttpServlet;
7 import javax.servlet.http.HttpServletRequest;
8 import javax.servlet.http.HttpServletResponse;
9 import cmmdc.ejb.CmmdcBean;
10 import javax.naming.Context;
11 import javax.naming.InitialContext;
12 import javax.servlet.annotation.WebServlet;

14 @WebServlet(urlPatterns = "/cmmdc")
15 public class CmmdcServlet extends HttpServlet{
16     CmmdcBean cb;

18     public void init(ServletConfig config) throws ServletException{
19         super.init(config);
20         Context ctx=null;
21         try{
22             ctx=new InitialContext();
23             cb=(CmmdcBean)ctx.lookup("java:module/CmmdcBean");
24         }
25         catch(Exception e){
26             System.out.println("Eroare : "+e.getMessage());
27         }
28     }

```

```

30 public void doGet(HttpServletRequest req, HttpServletResponse res)
31     throws ServletException, IOException { . . . }

33 public void doPost(HttpServletRequest req, HttpServletResponse res)
34     throws ServletException, IOException { . . . }
35 }
36 }

```

### 13.1.2 Componentă cu metode asincrone

În cazul apelării unei metode a unei componente EJB, clientul este blocat până la furnizarea rezultatului de către serverul de aplicație. Programând metode asincrone, programul client va deține controlul imediat după apelare sau altfel explicat apelarea nu mai este blocantă.

#### Definirea unei metode asincrone

Se îndeplinesc două condiții:

1. Metoda asincronă este adnotată *@Asynchronous* (javax.ejb.Asynchronous);
2. Rezultatul metodei va fi de tip `java.util.concurrent.Future<V>`.

#### Exemplul 13.1.2

În cazul exemplului tratat anterior, codul componentei EJB sesiune stateless devine

```

1 package cmmdc.ejb;
2 import javax.ejb.Stateless;
3 import javax.ejb.Asynchronous;
4 import javax.ejb.AsyncResult;
5 import java.util.concurrent.Future;

7 @Stateless
8 public class CmmdcBean{
9     @Asynchronous
10    public Future<Long> cmmdc(long m, long n){
11        long r, c;
12        do{
13            c=n;
14            r=n%n;
15            m=n;
16            n=r;
17        }while(r!=0);
18        Long result=new Long(c);
19        return new AsyncResult<Long>(result);
20    }
21 }

```

Clasa `AsyncResult` implementează interfața `Future`.

## Apelarea unei metode asincrone

Din nou, pentru exemplul anterior apelarea metodei `cmmdc` se programează prin

```
Future<Long> result=cb.cmmdc(m,n);
long x=0;
try{
    while(! result.isDone()){};
    x=result.get().longValue();
}
catch(Exception e){
    e.printStackTrace();
}
```

### 13.1.3 Aplicație JEE cu module EJB, Web și client RMI-IIOP

Modificăm arhitectura aplicației anterioare definind

- un modul EJB pentru componta EJB.

```
modul-ejb
|--> numePachet
|      |
|      | * .class
```

- un modul Web, a cărei arhivă va avea extensia `war`.

```
modul-Web
|--> WEB-INF
|      |--> classes
|      |      |
|      |      | * .class
|      |      |
|      |      | index.html
```

Desfășurarea aplicației va fi

```
--> META-INF
|      | application.xml
|      | modul-ejb.jar
|      | modul-Web.war
```

Instalarea aplicație, adică a ansamblului alcătuit din cele 2 module se face prin intermediul administratorului.

Fișierul `application.xml` precizează structura aplicației

```
<?xml version="1.0" encoding="UTF-8"?>
<application version="5" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/application_5.xsd">
  <display-name>nume-aplicatie</display-name>
  <module>
    <ejb>modul-ejb.jar</ejb>
  </module>
  <module>
    <web>
      <web-uri>modul-web.war</web-uri>
      <context-root>/context</context-root>
    </web>
  </module>
</application>
```

Această organizare necesită modificarea componentei EJB din secțiunea 13.1.1 în sensul definirii unei interfețe cu adnotarea **Remote**. Componenta EJB implementează interfața și în plus, dacă interfața are numele *XYZ* atunci componenta EJB va avea numele *XYZBean*.

Pentru exemplul discutat codurile sunt:

```
1 package cmmdc.ejb;
2 import javax.ejb.Remote;

4 @Remote
5 public interface Cmmdc{
6     public long cmmdc(long m, long n);
7 }
```

și

```
1 package cmmdc.ejb;
2 import javax.ejb.Stateless;

4 @Stateless
5 public class CmmdcBean implements Cmmdc{
6     public long cmmdc(long m, long n){. . .}
7 }
```

Modulul / clientul Web este cel prezentat anterior în secțiunea 13.1.1, cu diferența că referința la componenta EJB se face prin interfață.

În cazul exemplului tratat, desfășurarea clientului RMI-IIOP va fi

```
cmmdc-iiop
|--> cmmdc
|      |--> client
```

```

|      |      |      CmmdcClient.class
|      |--> ejb
|      |      |      Cmmdc.class

```

Codul clientului prin injectare este

```

1 package cmmdc.client;
2 import javax.ejb.EJB;
3 import cmmdc.ejb.Cmmdc;
4 import java.util.Scanner;
5
6 public class CmmdcClient{
7     @EJB
8     private static Cmmdc cb;
9
10    public static void main(String[] args)throws Exception{
11        Scanner scanner=new Scanner(System.in);
12        System.out.println("m=");
13        long m=scanner.nextLong();
14        System.out.println("n=");
15        long n=scanner.nextLong();
16        long x=cb.cmmdc(m,n);
17        System.out.println("Cmmdc : "+x);
18    }
19 }

```

Dacă se utilizează referința prin JNDI atunci adresarea componentei EJB va fi:

`java:global/cmmdc-ear/cmmdc-ejb/CmmdcBean`

Dacă *dist* este catalogul cu arhiva clientului atunci apelarea acestuia este

```

set GLASSFISH_HOME=.. \glassfish4
%GLASSFISH_HOME%\glassfish\bin\appclient -client
dist\cmmdc-iiop.jar -targetserver localhost:3700

```

Apelarea modulului Web se face în mod obișnuit, prin:

`http://host:8080/context`

unde *context* a fost definit în fișierul `application.xml`.

### 13.1.4 Componentă EJB sesiune singleton

Serverul de aplicație crează o singură instanțiază a componentei EJB care este utilizată de toți clienții. Diferența constă în utilizarea adnotării `@Singleton` în loc de `@Stateless`.

Utilizăm modelul din secțiunea 13.1.3. Considerăm doar varianta bazată pe adnotări.

**Exemplul 13.1.3** *Numărarea solicitărilor de utilizare a componentei EJB.*

Codul componentei EJB singleton

```

1 package single.ejb;
2 import javax.ejb.Remote;

4 @Remote
5 public interface Single{
6     public int getIndex();
7 }

```

```

1 package single.ejb;
2 import javax.ejb.Singleton;

4 @Singleton
5 public class SingleBean implements Single{
6     private int index=0;
7     public int getIndex(){
8         return ++index;
9     }
10 }

```

Servlet-ul aplicației client are codul

```

1 package single.web;
2 import java.io.IOException;
3 import java.io.PrintWriter;
4 import javax.servlet.ServletException;
5 import javax.servlet.http.HttpServlet;
6 import javax.servlet.http.HttpServletRequest;
7 import javax.servlet.http.HttpServletResponse;
8 import single.ejb.Single;
9 import javax.ejb.EJB;
10 import javax.servlet.annotation.WebServlet;

12 @WebServlet(urlPatterns = "/single")
13 public class SingleServlet extends HttpServlet{
14     @EJB
15     Single sb;

17     public void doGet(HttpServletRequest req,HttpServletResponse res)
18         throws ServletException,IOException{
19         int x=sb.getIndex();
20         PrintWriter out=res.getWriter();
21         res.setContentType("text/html");
22         String title="Single Servlet";
23         out.println("<HTML><HEAD><TITLE>");
24         out.println(title);
25         out.println("</TITLE></HEAD><BODY>");
26         out.println("<H1>"+title+"</H1>");
27         out.println("<p>Valoarea index : "+x);
28         out.println("</BODY></HTML>");
29         out.close();
30     }

32     public void doPost(HttpServletRequest req,HttpServletResponse res)
33         throws ServletException,IOException{
34         doGet(req,res);
35     }
36 }

```



Apelarea servlet-ului se face din fișierul *index.html*

```

1 <html>
2   <body bgcolor="#ccbbcc">
3     <center>
4       <h1> EJB Single Servlet </h1>
5       <form method="get"
6         action="single">
7         <p><input type="submit" value="Apeleaza">
8       </form>
9     </center>
10  </body>
11 </html>

```

Contextul aplicației definit în META-INF\application.xml este *ejbsingle*.

Aplicația client are codul

```

1 package single.client;
2 import javax.ejb.EJB;
3 import single.ejb.Single;
4 import java.util.Scanner;
5
6 public class SingleClient{
7     @EJB
8     private static Single nb;
9
10    public static void main(String [] args)throws Exception{
11        Scanner scanner=new Scanner(System.in);
12        String ch="Y";
13        int index=0;
14
15        while(ch.equals("Y")){
16            index=nb.getIndex();
17            System.out.println("Numarul de apelari : "+index);
18            System.out.println("Continuati ? (Y/N)");
19            do{
20                ch=scanner.next().trim().toUpperCase();
21            }
22            while ((!ch.startsWith("Y"))&&(!ch.startsWith("N")));
23        }
24    }
25 }

```

### 13.1.5 Componentă EJB sesiune stateful

Diferența formală față de modelele *Stateless* și *Singleton* constă în utilizarea adnotării `@Stateful`.

Considerăm un exemplu ce derivă din cel precedent.

**Exemplul 13.1.4** *Numărarea solicitărilor de utilizare a componentei EJB.*

Componenta EJB are interfața

```

1 package numara.ejb;
2 import javax.ejb.Remote;

4 @Remote
5 public interface Numara{
6     public int getIndex();
7     public void remove();
8 }

```

și implementarea

```

1 package numara.ejb;
2 import javax.ejb.Stateful;
3 import javax.ejb.Remove;

5 @Stateful
6 public class NumaraBean implements Numara{
7     private int index=0;
8     public int getIndex(){
9         return ++index;
10    }

12    @Remove
13    public void remove() {}
14 }

```

Metoda cu adnotarea **Remove** are ca efect ștergerea instanței componente EJB din serverul de aplicații. Din acest motiv, apelarea acestei metode într-un modul Web, are ca efect compromiterea servlet-ului.

Injectarea componente EJB are loc în momentul activării servlet-ului. Astfel fiecare client Web va utiliza aceeași componentă EJB.

Modulul IIOP este mult mai util. La fiecare apelare a modulului IIOP are loc injectarea unei componente EJB. Serverul de aplicații păstrează starea componente pe durata de utilizare a modulului IIOP.

Codul modulului Web

```

1 package numara.web;
2 import java.io.IOException;
3 import java.io.PrintWriter;
4 import javax.servlet.ServletException;
5 import javax.servlet.http.HttpServlet;
6 import javax.servlet.http.HttpServletRequest;
7 import javax.servlet.http.HttpServletResponse;
8 import numara.ejb.Numara;
9 import javax.ejb.EJB;

11 public class NumaraServlet extends HttpServlet{
12     @EJB
13     Numara nb;

15     public void doGet(HttpServletRequest req,HttpServletResponse res)
16         throws ServletException,IOException{
17         String oper=req.getParameter("oper");
18         PrintWriter out=res.getWriter();
19         res.setContentType("text/html");

```

```

20 String title="Numara Servlet";
21 out.println("<HTML><HEAD><TITLE>");
22 out.println(title);
23 out.println("</TITLE></HEAD><BODY>");
24 out.println("<H1>"+title+"</H1>");
25 switch(oper){
26     case "index" :
27         int x=nb.getIndex();
28         out.println("<p>Valoarea index : "+x);
29         break;
30     case "exit" :
31         out.println("<p>OK: Pe raspunderea d-voastra");
32         nb.remove();
33         break;
34 }
35 out.println("</BODY></HTML>");
36 out.close();
37 }

39 public void doPost(HttpServletRequest req,HttpServletResponse res)
40     throws ServletException,IOException{
41     doGet(req,res);
42 }
43 }

```

apelat din

```

1 <html>
2   <body bgcolor="#ccbbcc">
3     <center>
4       <h1> EJB Numar&#259; Servlet </h1>
5       <form method="get"
6         action="numara">
7         <table>
8           <tr>
9             <td>Selecta&#355;i opera&#355;ia</td>
10            <td>
11              <select name="oper">
12                <option value="index"> Num&#259;r de apel&#259;ri
13                <option value="exit"> Sterge componenta EJB
14              </select>
15            </td>
16          </tr>
17          <tr>
18            <td> <input type="submit" value="Apeleaza"></td>
19            <td/>
20          </tr>
21        </table>
22      </form>
23    </center>
24  </body>
25 </html>

```

Contextul aplicației Web este *ejbnumara*.  
Codul modului IIOP

```

1 package numara.client;
2 import javax.ejb.EJB;
3 import numara.ejb.Numara;

```

```

4 import java.util.Scanner;

6 public class NumaraClient{
7     @EJB
8     private static Numara nb;

10    public static void main(String [] args)throws Exception{
11        Scanner scanner=new Scanner(System.in);
12        String ch="Y",msg="";
13        int op,index=0;

15        while(ch.equals("Y")){
16            System.out.println("Operatii : 1. Numara 2. Sterge componenta EJB");
17            System.out.println("Operatia : ");
18            op=scanner.nextInt();
19            switch(op){
20                case 1:
21                    index=nb.getIndex();
22                    msg="Numarul de apelari : "+(new Integer(index)).toString();
23                    break;
24                case 2:
25                    nb.remove();
26                    msg="S-a sters componenta EJB";
27                    break;
28                default:
29                    msg="Cod operatie eronat";
30                    break;
31            }
32            System.out.println(msg);

34            System.out.println("Continuati ? (Y/N)");
35            do{
36                ch=scanner.next().trim().toUpperCase();
37            }
38            while((!ch.startsWith("Y"))&&(!ch.startsWith("N")));
39        }
40    }
41 }

```

Rulând modulele IIOP ale aplicațiilor corespunzătoare adnotărilor **Stateful** și **Singleton** observați diferența rezultatelor furnizate.

## 13.2 Componentă EJB *MessageDriven*

O componentă EJB *MessageDriven* (MDB) preia mesaje care sunt trimise către un obiect destinație predefinit de tip *Queue* sau *Topic*. Componenta EJB este adnotată cu

**@MessageDriven(mappedName=*nume\_Destinație*)**

Prelucrarea mesajelor primite se programează în metoda **onMessage** a interfeței **MessageListener**.

Afișarea unor mesaje se obține prin intermediul unei jurnalizări `java.util.logging.Logger`. Textele apar în fișierul `server.log` aflat în catalogul `log` al *domeniului*.

Obiectele administrator - fabrica de conexiuni, obiectul destinație - se crează înaintea desfășurării componentei EJB.

## Crearea obiectelor administrator

Dintr-un navigator, apelând `http://localhost:4848` se accesează administratorul grafic (web). Din **Resources** | **JMS Resources** se vor crea obiectele

### 1. Destination Resources

- *myQueue*  
JNDI Name : *myQueue*  
Physical Destination Name : `javax.jms.Queue`
- *myTopic*  
JNDI Name : *myTopic*  
Physical Destination Name : `javax.jms.Topic`

### 2. Connection Factories

- *myQueueConnectionFactory*  
JNDI Name : *myQueueConnectionFactory*  
Resource Type : `javax.jms.QueueConnectionFactory`
- *myTopicConnectionFactory*  
JNDI Name : *myTopicConnectionFactory*  
Resource Type : `javax.jms.TopicConnectionFactory`

**Exemplul 13.2.1** *O aplicație client trimite un număr de mesaje către o componentă MDB.*

### Codul componentei EJB

```

1 package mdb;
2 import javax.ejb.MessageDriven;
3 import javax.jms.Message;
4 import javax.jms.TextMessage;
5 import javax.jms.MessageListener;
6 import java.util.logging.Logger;

8 @MessageDriven(mappedName="myQueue")
9 public class MessageBean implements MessageListener {
10     private static final Logger logger =
11         Logger.getLogger(MessageBean.class.getName());

13     public void onMessage(Message msg) {
14         TextMessage txtMsg = null;
15         try {
16             if(msg instanceof TextMessage){
17                 txtMsg = (TextMessage) msg;

```

```

18         logger.info("MESSAGE.BEAN: " + txtMsg.getText());
19         System.out.println (txtMsg.getText());
20     }
21 }
22 catch (Throwable th) {
23     th.printStackTrace();
24 }
25 }
26 }

```

Desfășurarea componentei MDB va fi

```

simple-mdb
|--> META-INF
|    |    application.xml
|    simple-mdb.jar

```

Fișierul simple-mdb.jar nu conține decât clasa *mdb.MessageBean*, reprodusă mai sus.

Codul clientului cu regăsirea obiectelor prin adnotare

```

1 package mdb;
2 import javax.jms.QueueConnectionFactory;
3 import javax.jms.Destination;
4 import javax.annotation.Resource;
5 import javax.jms.JMSContext;
6 import javax.jms.JMSProducer;

8 public class MessageClient{
9     @Resource(lookup="myQueueConnectionFactory")
10    private static QueueConnectionFactory cf;
11    @Resource(lookup="myQueue")
12    private static Destination q;

14    public static void main(String [] args){
15        try{
16            JMSContext ctx=cf.createContext();
17            JMSProducer producer=ctx.createProducer();
18            for(int i=0;i<5;i++){
19                producer.send(q,"Hello "+i);
20            }
21            producer.send(q,ctx.createMessage());
22            ctx.close();
23        }
24        catch (Exception e) {
25            e.printStackTrace();
26        }
27    }
28 }

```

și prin JNDI

```

1 package mdb;
2 import javax.jms.QueueConnectionFactory;
3 import javax.jms.Destination;
4 import javax.naming.Context;
5 import javax.naming.InitialContext;
6 import javax.jms.JMSContext;

```

```

7 import javax.jms.JMSProducer;
9 public class MessageClient {
10     private static QueueConnectionFactory cf;
11     private static Destination q;
13     public static void main(String [] args){
14         Context ctx=null;
15         try{
16             ctx=new InitialContext ();
17             cf=(QueueConnectionFactory) ctx.lookup("myQueueConnectionFactory");
18             q=(Destination) ctx.lookup("myQueue");
19         }
20         catch(Exception e){
21             System.out.println("Eroare : "+e.getMessage());
22             System.exit(1);
23         }
25         try {
26             JMSContext jmsCtx=cf.createContext();
27             JMSProducer producer=jmsCtx.createProducer();
28             for(int i=0;i<5;i++){
29                 producer.send(q,"Hello "+i);
30             }
31             producer.send(q,jmsCtx.createMessage());
32             jmsCtx.close();
33         }
34         catch (Exception e) {
35             e.printStackTrace();
36         }
37     }
38 }

```

Clientul se execută prin intermediul lui `appclient`.

## 13.3 Componentă EJB *Entity*

O componenta de tip *entity* simplifică accesul la o bază de date într-o aplicație JEE.

### Crearea bazei de date

*Glassfish* conține SGBD *JavaDB (derby)*. Vom lansa serverul *glassfish* și prin intermediul utilitarului `ij` vom crea baza de date *AgendaEMail*. Crearea se face în modul descris în anexa dedicată utilizării unei SGBD într-un program Java.

### Definirea elementelor de conexiune dintre *glassfish* și baza de date

Dintr-un navigator, apelând `http://localhost:4848` se accesează administratorul grafic (web). Din **Resources** | **JDBC** se vor crea

#### 1. JDBC Connection Pool

cuvă de conexiuni la baza de date;

## 2. JDBC Resources

legătura dintre numele JNDI asociat bazei de date cu cuva de conexiuni creată anterior.

**Creare JDBC Connection Pool**

1. Pool Name *adresePool* un nume simbolic atașat cuvei
2. Resource Type Se selectează *java.sql.Driver*
3. Database Driver Vendor Se selectează *Derby*
4. Introspect Se bifează *Enabled*
5. Next
6. Driver Classname Se selectează *org.apache.derby.jdbc.ClientDriver*
7. Additional Properties Se bifează *URL* și se completează coloana **Value** cu *jdbc:derby:\. . . calea către locația bazei de date. . . \AgendaEMail*
8. Finish

**Creare JDBC Resources**

1. JNDI Name *jdbc/Adrese*, un nume simbolic atașat resursei
2. Pool Name Se selectează *adresePool*, numele cuvei de conexiuni
3. OK

**Aplicație cu componentă Entity**

O componenta *Entity* va fi adnotată `@Entity` și corespunde unui tabel al bazei de date relaționale.

**Exemplul 13.3.1** *Aplicație de consultarea a bazei de date AgendaEMail.*

Desfășurarea aplicației este

```
agendae-ear
|--> META-INF
|    |    persistence.xml
|    agenda-ejb.jar
```

unde *agenda-ejb.jar* are structura

```
--> ejb
|    |    AgendaEMail.class
|    |    AgendaEMailBean.class
|--> entity
|    |    Adrese.class
|--> META-INF
|    |    persistence.xml
```



Clasa *Adrese.java* este componenta *Entity* și corespunde tabelii *adrese*.

```

1 package entity;
2 import java.io.Serializable;
3 import javax.persistence.Entity;
4 import javax.persistence.Id;
5 import javax.persistence.GeneratedValue;
6 import javax.persistence.GenerationType;
7 import java.io.Serializable;

9 @Entity
10 public class Adrese implements Serializable{
11     @Id
12     @GeneratedValue(strategy=GenerationType.IDENTITY)
13     private int id;
14     private String nume;
15     private String email;

17     public String getNume() {
18         return nume;
19     }
20     public void setNume(String nume) {
21         this.nume = nume;
22     }
23     public String getEmail() {
24         return email;
25     }
26     public void setEmail(String email) {
27         this.email = email;
28     }
29     public int getId() {
30         return id;
31     }
32     public void setId(int id) {
33         this.id = id;
34     }
35 }

```

Accesul la baza de date se realizează printr-o componentă EJB *Session stateless* Codul interfeței *AgendaEMail.java* este

```

1 package ejb;
2 import javax.ejb.Remote;
3 import entity.Adrese;
4 import java.util.List;

6 @Remote
7 public interface AgendaEMail{
8     public List<Adrese> getEmail(String nume);
9     public List<Adrese> getNume(String email);
10 }

```

implementat de clasa *AgendaEMailBean.java*

```

1 package ejb;
2 import javax.ejb.Stateless;
3 import javax.persistence.PersistenceContext;
4 import javax.persistence.EntityManager;
5 import javax.persistence.Query;

```

```

6 import entity.Adrese;
7 import java.util.List;

9 @Stateless
10 public class AgendaEMailBean implements AgendaEMail{
11     @PersistenceContext(unitName="agendae_persistence_ctx")
12     EntityManager em;

14     public List<Adrese> getEmail(String nume){
15         String nm="\'"+nume+"\'";
16         String sql="SELECT entity FROM Adrese entity WHERE entity.nume="+nm;
17         System.out.println(sql);
18         Query query=em.createQuery(sql);
19         List<Adrese> list=(List<Adrese>)query.getResultList();
20         return list;
21     }

23     public List<Adrese> getNume(String email){
24         String eml="\'"+email+"\'";
25         String sql="SELECT entity FROM Adrese entity WHERE entity.email="+eml;
26         System.out.println(sql);
27         Query query=em.createQuery(sql);
28         List<Adrese> list=(List<Adrese>)query.getResultList();
29         return list;
30     }
31 }

```

Fișierul de configurare **persistence.xml** este

```

1 <persistence xmlns="http://java.sun.com/xml/ns/persistence" version="1.0">
2   <persistence-unit name="agendae_persistence_ctx">
3     <jta-data-source>jdbc/Adrese</jta-data-source>
4     <properties>
5       <!-- Use the java2db feature -->
6       <property name="toplink.ddl-generation" value="none"/>
7       <!-- Generate the sql specific to Derby database -->
8       <property name="toplink.jdbc.driver"
9         value="org.apache.derby.jdbc.ClientDriver"/>
10    </properties>
11  </persistence-unit>
12 </persistence>

```

Se observă următoarele corelații

- Valoarea atributului **name** a elementului **persistence-unit** (*agendae\_persistence\_ctx*) este utilizat în clasa *AgendaEMailBean* în adnotarea **@PersistenceContext**
- Elementul **jta-data-surse** conține numele JNDI a bazei de date definit în *glassfish*.

## Aplicația client

Desfășurarea aplicației client este

```

agenda-client
|--> ejb
|   |   AgendaEMail.class
|--> entity
|   |   Adrese.class
|--> META-INF
|   |   MANIFEST.MF
|   |   Client.class

```

Fișierul MANIFEST.MF este completat cu

```

Main-Class: Client
Class-Path: agenda-client.jar

```

Clasa *Client.java* are codul

```

1 import ejb.AgendaEMail;
2 import javax.ejb.EJB;
3 import java.util.List;
4 import java.util.Iterator;
5 import java.util.Scanner;
6 import java.util.InputMismatchException;
7 import entity.Adrese;

9 public class Client {
10     @EJB
11     private static AgendaEMail bean;
12     private static List<Adrese> list=null;

14     public static void main(String[] args) {
15         int prel,no;
16         String ch="Y",nume="",email="";
17         Scanner scanner=new Scanner(System.in);
18         Iterator<Adrese> iter=null;
19         Adrese inreg=null;

21         try {
22             while(ch.startsWith("Y")){
23                 do{
24                     System.out.println("Continue ? (Y/N)");
25                     ch=scanner.next().toUpperCase();
26                 }
27                 while((!ch.startsWith("Y"))&&(!ch.startsWith("N")));
28                 if(ch.startsWith("Y")){
29                     System.out.println("Natura interogarii ?");
30                     System.out.println("(Dupa nume:1,Dupa email:2)");
31                     do{
32                         prel=0;
33                         try{
34                             prel=scanner.nextInt();
35                         }
36                         catch(InputMismatchException e){}
37                     }
38                     while((prel<1)&&(prel>2));
39                     switch(prel){
40                         case 1 :
41                             System.out.println("Numele");
42                             nume=scanner.next().trim();
43                             list=bean.getEmail(nume);
44                             iter=list.iterator();

```

```
45         System.out.println("Adresele de email pentru : "+nume);
46         while(iter.hasNext()){
47             inreg=(Adrese)iter.next();
48             System.out.println(inreg.getEmail());
49         }
50         break;
51     case 2 :
52         System.out.println("Email");
53         email=scanner.next().trim();
54         list=bean.getNume(email);
55         iter=list.iterator();
56         System.out.println("Inregistrarile pentru adresa : "+email);
57         while(iter.hasNext()){
58             inreg=(Adrese)iter.next();
59             System.out.println(inreg.getNume());
60         }
61         break;
62     default: System.out.println("Comanda eronata");
63 }
64 }
65 }
66 }
67 catch (Exception e) {
68     e.printStackTrace();
69 }
70 }
71 }
```

## Întrebări recapitulative

1. Care sunt tipurile de componente EJB studiate?
2. Care sunt posibilitățile de apelare / utilizare al unei componente EJB?
3. Care este tehnologia de comunicație utilizată pentru apelarea unei componente EJB de un client program?
4. Cum se apelează un client - program al unei componente EJB?
5. Care sunt tipurile de componente sesiune EJB din JEE?

# Partea III

## ANEXE



# Anexa A

## Unelte de dezvoltare

Începem această secțiune prin a introduce câteva elemente privind sintaxa unui document `xml`.

### A.1 XML

EXtensible Markup Language (XML) reprezintă un limbaj pentru definirea marcajelor de semantică, care împart un document în părți identificabile în document. Din 1998, XML este un standard World Wide Web Consortium (W3C).

Totodată XML este un meta-limbaj pentru definirea sintaxei de utilizat în alte domenii.

XML descrie structura și semantica și nu formatarea.

Structura unui document XML este

```
<?xml version="1.0" encoding="tip_codare" [standalone="yes"]?>
    corpul documentului alcatuit din elemente
```

Prima linie - preambulul - reprezintă declarația de document XML. *Tipul codării* poate fi *utf-8*, *iso-8859-1*.

Corpul documentului este alcătuit din elemente. Începutul unui element este indicat printr-un marcaj. Textul marcajului constituie denumirea elementului. Elementele pot fi cu corp, alcătuit din alte elemente, având sintaxa

```
<marcaj>
    corpul elementului
</marcaj>
```

sau fără corp, caz în care sintaxa este

```
<marcaj/>
```

Un marcaj poate avea atribute date prin sintaxa

```
numeAtribut="valoareAtribut"
```

Valoarea unui atribut este cuprinsă între ghilimele ("").

```
<marcaj numeAtribut="valoareAtribut" . . .>
    corpul elementului
</marcaj>
```

Există un singur element rădăcină. Elementele unui document XML formează un arbore. Fiecărui marcaj de început al unui element trebuie să-i corespundă un marcaj de sfârșit. Caracterele mari și mici din denumirea unui element sunt distincte (*case sensitive*).

Elementele încuibărite (*nested*)- incluse într-un alt element - nu se pot amesteca, adică un marcaj de sfârșit corespunde ultimului marcaj de început.

Un comentariu se indică prin

```
<!--
    Text comentariu
-->
```

**Exemplul A.1.1** *Fișier XML - denumirile elementelor și conținutul lor permit înțelegerea simplă a semanticii introduse în document.*

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <cursuri>
3   <disciplina fel="obligatoriu">
4     <nume> Analiza numerica </nume>
5     <fond-de-timp>
6       <curs> 2 </curs>
7       <seminar> 1 </seminar>
8       <laborator> 1 </laborator>
9     </fond-de-timp>
10  </disciplina>
11  <disciplina fel="obligatoriu">
12    <nume> Programare distribuita </nume>
13    <fond-de-timp>
14      <curs> 2 </curs>
15      <seminar> 0 </seminar>
16      <laborator> 2 </laborator>
17    </fond-de-timp>
18  </disciplina>
19  <disciplina fel="obligatoriu">
```



```
20      <nume> Soft matematic </nume>
21      <fond-de-timp>
22          <curs> 2 </curs>
23          <seminar> 0 </seminar>
24          <laborator> 1 </laborator>
25      </fond-de-timp>
26 </disciplina>
27 </cursuri>
```

Un document XML este *bine formatat* dacă:

- există preambul;
- fiecare element este închis sau îi corespunde un marcaj de închidere;
- marcajele încuibărite nu se amestecă.

Un document XML este *valid* dacă

- este *bine formatat*;
- în cazul că există o referință către XML Schema atunci documentul XML este conform schemei.

Un produs informatic accesibil gratuit, care permite verificarea bine formatării și a validității unui document `xml`, este *XML Copy Editor*.

Pentru validare, dacă fișierul XML Schema se află în alt catalog decât cel al fișierului `xml` atunci acestuia i se asociază XML Schema prin *XML*  $\rightarrow$  *Associate*  $\rightarrow$  *XML Schema*...

Prelucrarea unui document XML în Java se poate face pe baza interfețelor de programare:

- *Document Object Model* - DOM;
- *Simple API for XML* - SAX;
- *Stream API for XML* - StAX;

Amintim faptul că reprezentarea obiectelor matematice prin elemente XML constituie subiectul a două proiecte *MathML* și *OpenMath*. Obiectivul limbajului *MathML* este reprezentarea unui text matematic într-un document HTML, în timp ce obiectivul proiectului *OpenMath* este reprezentarea semantică a datelor matematice pentru realizarea de aplicații cooperante între sisteme de calcul simbolic - CAS (*Computer Algebra System*).

Formatul XML se utilizează și pentru serializarea unui obiect, adică reprezentarea câmpurilor împreună cu valorile aferente sub formă de text, în cadrul transmisiilor de date. Interfața de programare este *Java Architecture for XML Binding* - JAXB.

## A.2 *apache-ant*

Utilitarul *apache-ant* asigură executarea unui șir de comenzi de operare. Aceste comenzi se înregistrează într-un fișier de tip `xml`, cu denumirea `build.xml`. Astfel, *apache-ant* se substituie unui fișier de comenzi `bat` în Windows sau unui script shell din Linux/Unix. Avantajul obținut constă în independența față de platforma de calcul (Windows, Linux).

**Instalarea** constă în dezarhivarea fișierului descărcat din Internet.

**Lansarea în execuție** necesită fixarea variabilei de mediu `JAVA_HOME`, ce conține calea la distribuția Java. Lansarea se poate face prin următorul fișier de comenzi

```
set JAVA_HOME=. . .
set ANT_HOME=. . .
%ANT_HOME%\bin\ant.bat %1
```

Parametrul `%1` acestui fișier de comenzi reprezintă obiectivul care se dorește a fi atins. Dacă se modifică denumirea sau locația fișierului `build.xml` atunci fișierul de comenzi se invocă cu opțiunea `-buildfile`.

Un fișier `build.xml` corespunde unui proiect (project), alcătuit din unul sau mai multe obiective (target). Atingerea fiecărui obiectiv constă din îndeplinirea uneia sau mai multor sarcini (task). *Apache-ant* conține o familie predefinită de sarcini. Programatorul are datoria fixării atributelor sarcinilor. Manualul din documentația produsului conține descrierea atributelor cât și exemple. În general, o sarcină reprezintă o operație executată uzual în linia de comandă.

Atributele se dau, respectând sintaxa **XML**

numeAtribut = "valoareAtribut"

Astfel, un proiect apare sub forma

```
<project name="numeProiect"
        default="obiectiv"
        basedir="catalogDeReferinta">

  <target name="numeObiectiv">
    sarcini
  </target>
  . . . . .
</project>
```

Dacă la apelarea lui *Apache-ant* lipsește parametrul opțional atunci se va executa obiectivul **default**.

Într-un proiect se pot defini variabile prin marcajul

```
<property name="numeVariabila" value="valoareVariabila" />
```

O variabilă definită se va utiliza cu sintaxa `${numeVariabila}`.

### Exemplul A.2.1 Fișierul *build* pentru execuția programelor din §1.2.

```

1 <project name="Socket" default="Server" basedir=".">
2   <description> Socluri TCP </description>

4   <!-- set global properties for this build -->
5   <property name="build" location="work"/>
6   <property name="src" location="." />
7   <property name="host" value="localhost" />
8   <property name="port" value="7999" />

10  <target name="init">
11    <!-- Create the time stamp -->
12    <tstamp/>
13    <!-- Create the build directory structure used by compile -->
14    <delete dir="${build}" />
15    <mkdir dir="${build}" />
16  </target>

18  <target name="Compile" depends="init" description="compile the source " >
19    <javac srcdir="${src}" destdir="${build}" includeantruntime="false" />
20  </target>

22  <target name="Server" depends="Compile">
23    <java classname="MyMServer" classpath="${build}" fork="true" />
24  </target>

26  <target name="Client">
27    <java classname="VisualCmmdcClient" classpath="${build}" fork="true">
28      <arg line="${host} ${port}" />
29    </java>
30  </target>
31 </project>

```

Utilizarea lui *apache-ant* presupune că toate resursele utilizate, cuprinse uzual în fișiere cu extensia **jar** (*java archive*) sunt disponibile pe calculatorul local. Dacă calculatorul este conectat la Internet, atunci resursele publice pot fi descărcate împreună cu toate dependențele și utilizate prin *apache-ant*, folosind suplimentar *apache-ivy* <sup>1</sup>.

În vederea utilizării lui *apache-ivy* se copiază fișierul **ivy-\*.jar** din distribuția *apache-ivy* în **ANT\_HOME\lib**.

Fișierul *build.xml* conține în plus

<sup>1</sup>Asemănător cu *maven*, un alt produs de dezvoltare.

```

<project name="Proiect" basedir="." default="obiectiv_final"
  xmlns:ivy="antlib:org.apache.ivy.ant">

  . . .
  <!-- =====
        target: resolve
        ===== -->
  <target name="resolve" depends="init"
    description="--> retrieve dependencies with ivy">
    <ivy:retrieve/>
  </target>

  <!-- =====
        target: report
        ===== -->
  <target name="report" depends="resolve"
    description="--> generates a report of dependencies">
    <ivy:report todir="${report.dir}"/>
  </target>

  . . .
</project>

```

la care se adaugă un fișier *ivy.xml*, cu dependențele necesare aplicației care urmează a fi preluate dintr-unul din depozitele *ivy* - *local*, *shared*, *public*

```

<ivy-module version="2.0">
  <info organisation="cs.unitbv.ro" module="Biblioteca-Ivy"/>
  <dependencies>
    <dependency org="commons-fileupload" name="commons-fileupload" rev="1.2"/>
    . . .
  </dependencies>
</ivy-module>

```

### A.3 *apache-maven*

*Apache-maven*<sup>2</sup> este un alt cadru de dezvoltare și gestiune a proiectelor (Project management framework). Calculatorul pe care se dezvoltă proiectul / aplicația trebuie să fie conectat la internet. Resursele necesare îndeplinirii diferitelor sarcini (*maven artifacts*) sunt preluate din internet și depuse într-un depozit local *maven* (*local repository*). În prezent sunt întreținute depozite

<sup>2</sup>Maven – acumulator de cunoștințe (Idiș).

publice de resurse soft necesare dezvoltării de aplicații cu *maven*<sup>3</sup> iar dezvoltatorii de instrumente soft au posibilitatea de a-și promova produsele prin depunerea într-un depozit *maven*. Dintr-un asemenea depozit public resursele necesare sunt descărcate în depozitul local.

## Gestiunea proiectelor cu *apache-maven*

**Instalarea produsului** constă în dezarhivarea fișierului descărcat din internet într-un catalog `MAVEN_HOME`.

**Utilizarea produsului** necesită

- Completarea variabilei de sistem `PATH` cu calea `MAVEN_HOME\bin`.
- Declararea variabilei `JAVA_HOME` având ca valoare calea către distribuția `jdk` folosită.

În mod obișnuit depozitul local *maven* este

`C:\Documents and Settings\client\.m2\repository`

Locația depozitului local se poate modifica, introducând elementul

`<localRepository>volum:/cale/catalog_depozit</localRepository>`

în fișierul `MAVEN_HOME\conf\settings.xml`.

Potrivit principiului separării preocupărilor (Separation of Concerns), un proiect *maven* produce o singură ieșire.

Declararea unui proiect se face printr-un fișier `pom.xml` (Project Object Model). Este sarcina programatorului să completeze fișierul `pom.xml`, creat la generarea structurii de cataloage ale proiectului, cu specificarea resurselor suplimentare sau a condiționărilor în efectuarea unor operații (de exemplu, prezența adnotărilor necesită utilizarea versiunii Java  $\geq 1.5$ ).

Dezvoltarea unei aplicații / proiect prin *maven* presupune generarea unei structuri de cataloage (Standard directory layout for projects). Această structură de cataloage este specifică tipului / șablonului de aplicație (*archetype*, în limbajul *maven*).

Șabloane uzuale de aplicații:

Nume șablon	Semnificația
<code>maven-archetype-quickstart</code>	aplicație simplă (șablonul implicit)
<code>maven-archetype-webapp</code>	aplicație Web
<code>maven-archetype-j2ee-simple</code>	aplicație JEE

<sup>3</sup>De exemplu [repo1.maven.org/maven2](http://repo1.maven.org/maven2).

Șablonul se specifică în parametrul `-DarchetypeArtifactId` al comenzii `mvn archetype:create`.

Îndeplinirea diferitelor obiective (generarea unui proiect, compilare, arhivare, testare, etc) se obțin prin comenzi *maven*.

Comenzile *maven* sunt de două tipuri:

- Comenzi pentru gestiunea ciclului de viață al unui proiect (lifecycle commands):

Comanda <i>maven</i>	Semnificația
<code>mvn -version</code>	afișează versiunea <i>maven</i> (utilă pentru verificarea funcționării lui <i>maven</i> )
<code>mvn clean</code>	șterge fișierele <i>maven</i> generate
<code>mvn compile</code>	compilarea sursele Java
<code>mvn test-compile</code>	compilează sursele Java care realizează testele <code>junit</code>
<code>mvn test</code>	execută testul <code>junit</code>
<code>mvn package</code>	crează o arhivă jar sau war
<code>mvn install</code>	depune arhiva jar sau war în depozitul local

- Comenzi de operare inserate (plugin commands):

Comanda <i>maven</i>	Semnificația
<code>mvn archetype:create</code>	generează structura de cataloage a proiectului <code>mvn archetype:create \</code> <code>-DgroupId=numelePachetuluiAplicației \</code> <code>-DartifactId=numeleProiectului \</code> <code>-DarchetypeArtifactId=numeȘablon</code>
<code>mvn archetype:generate</code>	generează structura de cataloage ai proiectului. Șablonul se alege dintr-o listă
<code>mvn clean:clean</code>	șterge fișierele generate în urma compilării
<code>mvn compiler:compile</code>	compilarea sursele Java
<code>mvn surefire:test</code>	execută testul <code>junit</code>
<code>mvn jar:jar</code>	crează o arhivă jar
<code>mvn install:install-file</code>	depune o arhivă jar în depozitul local <code>mvn install:install-file \</code> <code>-Dfile=numeFișier \</code> <code>-DgroupId=numePachet \</code> <code>-DartifactId=numeProiect \</code> <code>-Dversion=versiunea \</code> <code>-Dpackaging=tipArhivă</code>
<code>mvn exec:java</code>	execută metoda <code>main</code> a unei clase <code>mvn exec:java -Dexec.mainClass=</code> <code>"clasaMetodeiMain"</code> <code>-Dexec.args="listă argumente"</code>
<code>mvn dependency:copy-dependencies</code>	descarcă în catalogul <code>target</code> resursele declarate în <code>dependencies</code> .

Astfel comanda

```
set GroupID=simple.app.helloworld
set ArtifactID=helloworld
set Version=1.0
mvn archetype:generate -DgroupId=%GroupID%
-DartifactId=%ArtifactID%
-Dversion=%Version%
-DarchetypeArtifactId=maven-archetype-quickstart
-DinteractiveMode=false
```

generează arborescența

```
hello
|--> src
```

```

|      |--> main
|      |      |--> java
|      |      |      |--> unitbv
|      |      |      |      |--> cs
|      |      |      |      |      |--> calcul
|      |      |      |      |      |      App.java
|      |--> test
|      |      |--> java
|      |      |      |--> unitbv
|      |      |      |      |--> cs
|      |      |      |      |      |--> calcul
|      |      |      |      |      |      AppTest.java
|      pom.xml

```

Descrierea proiectului este dată în fișierul `pom.xml` generat

```

1 <project xmlns="http://maven.apache.org/POM/4.0.0"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4   http://maven.apache.org/maven-v4_0_0.xsd">
5   <modelVersion>4.0.0</modelVersion>
6   <groupId>unitbv.cs.calcul</groupId>
7   <artifactId>hello</artifactId>
8   <packaging>jar</packaging>
9   <version>1.0-SNAPSHOT</version>
10  <name>hello</name>
11  <url>http://maven.apache.org</url>
12  <dependencies>
13    <dependency>
14      <groupId>junit</groupId>
15      <artifactId>junit</artifactId>
16      <version>3.8.1</version>
17      <scope>test</scope>
18    </dependency>
19  </dependencies>
20 </project>

```

*App.java* este programul Java *HelloWorld* iar *AppTest.java* este un program de verificare bazat pe *junit*.

Pentru testarea aplicației, din catalogul *hello* se execută comenzile

```

mvn compile
mvn exec:java -Dexec.mainClass="simple.app.helloworld.App"
mvn test

```

Execuția programului se poate lansa prin intermediul unui profil (*profile*)

```
mvn exec:java -PnumeProfil
```

În prealabil în fișierul `pom.xml` se introduce secvența

```

<profiles>
  <profile>
    <id>numeProfil</id>
    <properties>
      <target.main.class>clasaCuMetodaMain</target.main.class>
    </properties>
  </profile>
</profiles>

```



```

<build>
  <plugins>
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>exec-maven-plugin</artifactId>
      <configuration>
        <mainClass>${target.main.class}</mainClass>
        <includePluginDependencies>false</includePluginDependencies>
      </configuration>
    </plugin>
  </plugins>
</build>

```

Această variantă este avantajoasă în cazul în care proiectul include mai multe clase cu metoda `main`.

Sarcina programatorului este acela de a înlocui aceste programe cu cele care rezolvă sarcinile proiectului. Pentru orice prelucrare toate dependențele trebuie să se găsească în depozitul local *maven*. Dacă o dependență (resursă `jar`) nu se găsește în depozitul local atunci depandința este căutată într-un depozit global și este descărcată în depozitul global. Este sarcina programatorului să declare toate dependențele necesare unei aplicații. Declararea se face într-un element `<dependency>`. Dacă resursa este inaccesibilă atunci *maven* termină prelucrarea.

Programatorul are posibilitatea să specifice depozite globale unde să se găsească resursele necesare, de exemplu

```

<repositories>
  <repository>
    <id>java.net-promoted</id>
    <url>https://maven.java.net/content/groups/promoted/</url>
  </repository>
</repositories>

```

**Exemplul A.3.1** *Dezvoltarea aplicației de calcul al celui mai mare divizor comun a două numere naturale utilizând maven.*

Generăm proiectul *maven*

```

set GroupID=simple.app.cmmdc
set ArtifactID=cmmdc
set Version=1.0
mvn archetype:generate -DgroupId=%GroupID%
-DartifactId=%ArtifactID%
-Dversion=%Version%
-DarchetypeArtifactId=maven-archetype-quickstart
-DinteractiveMode=false

```

Înlocuim clasa *App* cu clasele

```

1.
1 package simple.app.cmmdc;
2 import java.util.Scanner;

4 public class App{
5     public static void main(String[] args){
6         Scanner scanner=new Scanner(System.in);
7         System.out.println("m=");
8         long m=scanner.nextLong();
9         System.out.println("n=");
10        long n=scanner.nextLong();
11        MyCmmdc obj=new MyCmmdc();
12        System.out.println("cmmdc = "+obj.cmmdcService.cmmdc(m,n));

14    }
15 }

```

```

2.
1 package simple.app.cmmdc;
2 public class MyCmmdc{
3     interface CmmdcService {
4         long cmmdc(long m, long n);
5     }

7     static CmmdcService cmmdcService=(long m, long n) -> {. . .}

9 }

```

iar programul de testare *AppTest.java* prin

```

1 package simple.app.cmmdc;
2 import org.junit.*;
3 import static org.junit.Assert.*;

5 public class TestProiect{
6     MyCmmdc obj;
7     long rez;

9     @Before
10    public void setUp(){
11        obj=new MyCmmdc();
12    }

14    @Test
15    public void testCmmdc1(){
16        rez=obj.cmmdcService.cmmdc(56,42);
17        assertEquals(141, rez);
18    }

20    public static void main(String args[]){
21        org.junit.runner.JUnitCore.main("proiect.TestProiect");
22    }
23 }

```

Completarea fișierului *pom.xml*:

1. Aplicația necesită Java 8. Această cerință se declară prin secvența de cod

```

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.2</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>
  </plugins>
</build>

```

2. Întrucât dorim să folosim cea mai recentă versiune a produsului `junit` modificăm numărul versiunii.

Fișierul `pom.xml` devine

```

1 <project xmlns="http://maven.apache.org/POM/4.0.0"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4     http://maven.apache.org/xsd/maven-4.0.0.xsd">
5   <modelVersion>4.0.0</modelVersion>
6
7   <groupId>simple.app.cmmdc</groupId>
8   <artifactId>cmmdc</artifactId>
9   <version>1.0-SNAPSHOT</version>
10  <packaging>jar</packaging>
11
12  <name>cmmdc</name>
13  <url>http://maven.apache.org</url>
14
15  <properties>
16    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
17  </properties>
18
19  <dependencies>
20    <dependency>
21      <groupId>junit</groupId>
22      <artifactId>junit</artifactId>
23      <version>4.11</version>
24      <scope>test</scope>
25    </dependency>
26  </dependencies>
27
28  <build>
29    <plugins>
30      <plugin>
31        <groupId>org.apache.maven.plugins</groupId>
32        <artifactId>maven-compiler-plugin</artifactId>

```

```

33         <configuration>
34             <!-- or whatever version you use -->
35             <source>1.8</source>
36             <target>1.8</target>
37         </configuration>
38     </plugin>
39 </plugins>
40 </build>
41 </project>

```

Operarea poate fi

```

mvn archetype:create -DgroupId=simple.app.cmmdc -DartifactId=cmmdc
-DarchetypeArtifactId=maven-archetype-quickstart
cd cmmdc
mvn compile
mvn test
mvn exec:java -Dexec.mainClass="simple.app.cmmdc.App"

```

## maven cu ant

În *maven* se pot integra sarcini *apache-ant* pentru orice etapă al evoluției unei aplicații. Utilizarea constă în completarea fișierului `pom.xml` cu

```

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-antrun-plugin</artifactId>
      <executions>
        <execution>
          <phase>
            <!-- etapa de viata : compile, package, install, test -->
          </phase>
          <configuration>
            <tasks>
              <!-- Exemplu
              <property name="compile_classpath" refid="maven.compile.classpath"/>
              <property name="runtime_classpath" refid="maven.runtime.classpath"/>
              <property name="test_classpath" refid="maven.test.classpath"/>
              <property name="plugin_classpath" refid="maven.plugin.classpath"/>

              <echo message="compile classpath: ${compile_classpath}"/>
              <echo message="runtime classpath: ${runtime_classpath}"/>
              <echo message="test classpath:    ${test_classpath}"/>
              <echo message="plugin classpath:  ${plugin_classpath}"/>
              -->
            </tasks>
          </configuration>
          <goals>
            <goal>run</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>

```

În elementul `<tasks>` se definesc sarcinile **ant** care se doresc executate la comanda corespunzătoare prelucrării etapei din evoluția proiectului *maven* - compile, package, install, test, etc.

Mai precis, elementele `<echo>` se înlocuiesc cu sarcinile **ant** care se doresc efectuate.

Proprietățile se definesc în elementul

```
<properties>
  <nume.proprietate>valoare</nume.proprietate>
  .
  .
  .
</properties>
```

## RMI prin *maven*

### Exemplul A.3.2 *Aplicația de calcul a celui mai mare divizor comun.*

Corespunzător serverului și clientului se generează într-un catalog comenzi maven

```
mvn archetype:create -DgroupId=cmmdc -DartifactId=server
-DarchetypeArtifactId=maven-archetype-quickstart
```

```
mvn archetype:create -DgroupId=cmmdc -DartifactId=client
-DarchetypeArtifactId=maven-archetype-quickstart
```

Clasa *App.java* se înlocuiește cu

Catalog	Clase
s	cmmdc.ICmmdc cmmdc.server.CmmdcImpl.java
c	cmmdc.ICmmdc cmmdc.client.CmmdcClient.java

În catalogul aplicației se introduce fișierul *pom.xml*

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project>
3   <modelVersion>4.0.0</modelVersion>
4   <artifactId>cmmdc</artifactId>
5   <groupId>cmmdc</groupId>
6   <version>1.0.0</version>
7   <name>RMI</name>
8   <packaging>pom</packaging>
9   <modules>
10     <module>server</module>
11     <module>client</module>
12   </modules>
13 </project>
```

Se execută:

1. `mvn clean package`
2. Din *server* se lansează `rmiregistry` prin

```
set classpath=. . .\rmi\cmmdc\server\target\classes
rmiregistry
```

urmat de

```
mvn -e exec:java -Dexec.mainClass="cmmdc.server.CmmdcImpl"
```

3. Din *client*

```
mvn exec:java -Dexec.mainClass="cmmdc.client.CmmdcClient"
```

## Dezvoltarea unui servlet prin *maven*

Dezvoltarea unui servlet prin *maven*<sup>4</sup> se va exemplifica prin aplicația în care servletul răspunde clientului cu mesajul *Hi nume\_client*, *nume\_client* fiind parametru transmis de către client la apelarea servlet-ului.

Container de servleți cu care lucrează *maven* este *jetty*.

Dezvoltarea revine la parcurgerea pașilor:

1. **Generarea aplicației:**

```
mvn archetype:create -DgroupId=hello
                    -DartifactId=helloname
                    -DarchetypeArtifactId=maven-archetype-webapp
```

Se crează structura de cataloage și fișiere

```
helloname
|--> src
|   |--> main
|   |   |--> resources
|   |   |--> webapp
|   |   |   |--> WEB-INF
|   |   |   |   |--> web.xml
|   |   |   |   |--> index.jsp
|   |   |   |   |--> pom.xml
```

unde fișierul *pom.xml* este

---

<sup>4</sup>Iverson W., *Building Web Applications with maven 2*, <http://www.java.net/author/will-iverson>

```

1 <project xmlns="http://maven.apache.org/POM/4.0.0"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4     http://maven.apache.org/maven-v4_0_0.xsd">
5   <modelVersion>4.0.0</modelVersion>
6   <groupId>hello</groupId>
7   <artifactId>helloname</artifactId>
8   <packaging>war</packaging>
9   <version>1.0-SNAPSHOT</version>
10  <name>helloname Maven Webapp</name>
11  <url>http://maven.apache.org</url>
12  <dependencies>
13    <dependency>
14      <groupId>junit</groupId>
15      <artifactId>junit</artifactId>
16      <version>3.8.1</version>
17      <scope>test</scope>
18    </dependency>
19  </dependencies>
20  <build>
21    <finalName>helloname</finalName>
22  </build>
23 </project>

```

## 2. Completarea aplicației.

- Se completează structura creată anterior cu

```
helloname
|--> src
|   |--> main
|   |   |--> java
|   |   |   |--> hello
|   |   |   |   HelloServlet.java
|   |   |--> resources
|   |   |--> webapp
|   |   |   |--> WEB-INF
|   |   |   |   web.xml
|   |   |   |   index.html
|   pom.xml
```

*HelloServlet.java* este cel utilizat în capitolul *Servlet* din cursul de *Programare distribuită*.

- Fișierul `web.xml` este completat cu elementele specifice servlet-ului (servlet și servlet-mapping).

```

1 <!DOCTYPE web-app PUBLIC
2   "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
3   "http://java.sun.com/dtd/web-app-2.3.dtd" >
4
5 <web-app>
6   <display-name>Archetype Created Web Application</display-name>
7   <servlet>
8     <servlet-name>hello</servlet-name>
9     <servlet-class>hello.HelloServlet</servlet-class>
10  </servlet>

```

```

11 <servlet-mapping>
12   <servlet-name>hello</servlet-name>
13   <url-pattern>/hello</url-pattern>
14 </servlet-mapping>
15 </web-app>

```

- Fișierul *index.jsp* se înlocuiește cu fișierul *index.html* utilizat la servletul menționat anterior<sup>5</sup>.

- Fișierul *pom.xml* se completează cu

- Referințele la resursele *javax.servlet.servlet-api*, necesare compilării.

```

<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>servlet-api</artifactId>
  <version>3.0.1</version>
  <scope>provided</scope>
</dependency>

```

- Referințele serverului Web *jetty*

```

<plugins>
  <plugin>
    <groupId>org.eclipse.jetty</groupId>
    <artifactId>jetty-maven-plugin</artifactId>
    <version>9.1.0.v20131115</version>
  </plugin>
</plugins>

```

Codul fișierului *pom.xml* devine

```

1 <project xmlns="http://maven.apache.org/POM/4.0.0"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4     http://maven.apache.org/maven-v4_0_0.xsd">
5   <modelVersion>4.0.0</modelVersion>
6   <groupId>hello</groupId>
7   <artifactId>helloname</artifactId>
8   <packaging>war</packaging>
9   <version>1.0-SNAPSHOT</version>
10  <name>helloname Maven Webapp</name>
11  <url>http://maven.apache.org</url>
12  <dependencies>
13    <dependency>
14      <groupId>junit</groupId>
15      <artifactId>junit</artifactId>
16      <version>3.8.1</version>
17      <scope>test</scope>
18    </dependency>
19    <dependency>
20      <groupId>javax.servlet</groupId>
21      <artifactId>javax.servlet-api</artifactId>
22      <version>3.0.1</version>
23      <scope>provided</scope>
24    </dependency>

```

<sup>5</sup>Atenție la parametri de apelare a servlet-ului, care eventual trebuie adaptați. Contextul aplicației coincide cu parametrul *artifactId*



```

25 | </dependencies>
26 | <build>
27 |   <finalName>helloname</finalName>
28 |   <plugins>
29 |     <plugin>
30 |       <groupId>org.eclipse.jetty</groupId>
31 |       <artifactId>jetty-maven-plugin</artifactId>
32 |       <version>9.1.0.v20131115</version>
33 |     </plugin>
34 |   </plugins>
35 | </build>
36 | </project>

```

Clauza **provided** din elementul **<scope>** implică neincluderea resursei în arhiva **war**.

### 3. Prelucrarea revine la

- (a) `mvn clean package`
- (b) lansarea serverului Web *jetty*: `mvn jetty:run`  
Serverul se oprește cu **Ctrl+C**.
- (c) testarea servlet-ului: într-un navigator se deschide pagina `http://localhost:8080/index.html`.

## Aplicație JSP prin *maven*

Ne propunem să calculăm cel mai mare divizor comun a două numere naturale într-o pagină JSP utilizând o componentă Java pentru calculul propriu-zis (cf. cursului de Programare distribuită).

Realizarea aplicației JSP constă din

### 1. Generarea cadrului:

```

mvn archetype:create -DgroupId=jsp -DartifactId=cmmdcjsp
-DarchetypeArtifactId=maven-archetype-webapp

```

- 2. Se completează aplicația cu fișierele *CmmdcBean.java*, *cmmdc.jsp* și *index.html* din cursul Programare distribuită, desfășurarea fiind

```

cmmdcjsp
|--> src
|   |--> main
|   |   |--> java
|   |   |   |   CmmdcBean.java
|   |   |--> resources
|   |   |--> webapp
|   |   |   |--> jsp
|   |   |   |   cmmdc.jsp

```

```
|   |   |   |--> WEB-INF
|   |   |   |   | web.xml
|   |   |   index.html
|   pom.xml
```

3. Fișierul `pom.xml` se completează cu referința pentru *Jetty*, la fel cum s-a procedat la servlet.
4. Prelucrarea constă din
  - (a) `mvn clean package`
  - (b) `mvn jetty:run`
  - (c) Din navigator se apelează `http://localhost:8080/index.html`.

## Anexa B

# Fir de execuție prin $\lambda$ -expresie

Java introduce firele de execuție prin interfețele

- `java.lang.Runnable`

Interfața declară metoda `public void run()`.

Clasa `java.lang.Thread` implementează interfața `Runnable`.

- `java.util.concurrent.Callable<V>`

Interfața declară metoda `public V call()`.

Șablon pentru definirea unui fir de execuție de tip `Thread` printr-o lambda expresie

```
interface MyThread{
    Thread service(semnatura datelor de intrare);
}

static MyThread action=(semnatura datelor de intrare)->{
    return new Thread()->{
        codul metodei run
    };
};
```

### Exemplul B.0.3

```
interface MyThread{
    Thread scrie(String txt);
}

static MyThread f=(String txt)->{
```

```

return new Thread()->{
    System.out.println(txt);
}; // Varianta fara lansare implicita
// Varianta cu lansare implicita
// }).start();
};

```

Variante de lansare în execuție:

- ```

public static void main(String args[]){
    f.scrie("Primul fir de executie").start();
    f.scrie("Al doilea fir de executie").start();
    f.scrie("Al treilea fir de executie").start();
}

```
- ```

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
. . .
static final int NTHREADS=100;
static ExecutorService exec=Executors.newFixedThreadPool(NTHREADS);
. . .
public static void main(String args[]){
    exec.execute(f.scrie("Primul fir de executie"));
    exec.execute(f.scrie("Al doilea fir de executie"));
    exec.execute(f.scrie("Al treilea fir de executie"));
    exec.shutdown();
}

```

Șablon pentru definirea unui fir de execuție de tip `java.util.concurrent.Callable<T>` printr-o lambda expresie

```

interface MyCallable{
    Callable<T> service(semnatura datelor de intrare);
}

static MyCallable action=(semnatura datelor de intrare)->{
    Callable<T> c=()->{
        T var;
        codul metodei call
        return var;
    };
    return c;
};

```

#### Exemplul B.0.4

```

import java.util.concurrent.Callable;
. . .
interface MyCallable{
    Callable<Integer> scrie(int index) throws Exception;
}

```

```

}

static MyCallable f=(int index)->{
    Callable<Integer> c=()->{
        System.out.println("I am "+index);
        return index;
    };
    return c;
};

```

Variante de lansare în execuție:

```

● public static void main(String args[]){
    int numarFire=3;
    try{
        for(int i=0;i<numarFire;i++){
            Integer r=f.scrie(i).call();
            System.out.println("Returned : "+r);
        }
    }
    catch(Exception e){
        System.out.println("Exception : "+e.getMessage());
    }
}

● import java.util.concurrent.ExecutorService;
import java.util.concurrent.Future;
import java.util.concurrent.Executors;
import java.util.concurrent.Callable;
import java.util.List;
import java.util.ArrayList;
. . .
    static final int NTHREADS=100;
    static ExecutorService exec=Executors.newFixedThreadPool(NTHREADS);
. . .
public static void main(String args[]){
    List<Callable<Integer>> tasks =
        new ArrayList<Callable<Integer>>(NTHREADS);
    try{
        for(int i=0;i<NTHREADS;i++){
            tasks.add(f.scrie(i));
        }
        List<Future<Integer>> results=exec.invokeAll(tasks);
        for(int i=0;i<NTHREADS;i++){
            Future<Integer> r=results.get(i);
            if(r.isDone()) System.out.println("Returned : "+r.get());
        }
        exec.shutdown();
    }
    catch(Exception e){
        System.out.println("Exception : "+e.getMessage());
    }
}

```



# Anexa C

## Testare cu *junit*

Verificarea / testarea automată a programelor Java se poate face cu produsul informatic *junit*. Deseori se formulează probleme de test ale căror rezultate sunt cunoscute, cu rolul de a verifica funcționarea unui program de rezolvare, pentru depistarea unor greșeli.

S-a dezvoltat și o metodologie de lucru *Test Driven Development* - (*TDD*) care presupune pentru orice clasă elaborată realizarea unui program de testare, chiar a priori.

Un alt produs care are același scop de verificare a rezultatelor este *TestNG*.

*junit* permite verificarea automată a rezultatelor furnizate de un program, pentru o mulțime de date de test.

**Instalarea produsului** constă din dezarhivarea fișierului descărcat într-un catalog `JUNIT_HOME`. Pentru compilare și execuție, variabila de sistem `classpath` trebuie să conțină referința `JUNIT_HOME\junit-*.jar`.

**Utilizarea produsului** într-un program Java constă din:

1. Declararea resurselor pachetului *junit* prin

```
import org.junit.*;
import static org.junit.Assert.*;
```
2. Declararea clasei cu testele *junit* - uzual în metoda `main`.

```
org.junit.runner.JUnitCore.main("AppClass");
```
3. Eventuale operații necesare înainte sau după efectuarea testelor se precizează respectiv, în câte o metodă care a fost declarată cu adnotarea `@org.junit.Before` și respectiv, `@org.junit.After`.
4. Testele se definesc în metode declarate cu adnotarea `@org.junit.Test`.  
Clasa `Assert` posedă metodele de verificare ale unui rezultat:

- `static void assertEquals(Tip așteptat, Tip actual)`  
unde Tip poate fi double, int, long, Object.
- `static void assertEquals(double așteptat, double actual, double delta)`  
Testul reușește dacă  $|așteptat - actual| < delta$ .
- `static void assertEquals(Tip[] așteptat, Tip[] actual)`  
unde Tip poate fi byte, char, int, long, short, Object.
- `static void assertTrue(boolean condiție)`
- `static void assertFalse(boolean condiție)`
- `static void assertNull(Object object)`
- `static void assertNotNull(Object object)`

În cazul exemplului

```

1 import org.junit.*;
2 import static org.junit.Assert.*;

4 public class Exemplu{
5     public double rezultat=1.0;
6     public double eps=1e-6;

8     double getValue(){
9         return 1.0000001;
10    }

12    @Test
13    public void test(){
14        assertEquals(rezultat,getValue(),eps);
15    }

17    public static void main(String[] args){
18        org.junit.runner.JUnitCore.main("Exemplu");
19    }
20 }
```

se obține

```

JUnit version 4.5
.
Time: 0.03

OK (1 test)
```

**Exemplul C.0.5** Testarea clasei App ( 2.2.1).



```

1 package server;
2 import org.junit.*;
3 import static org.junit.Assert.*;

5 public class TestApp{
6     private App app;

8     @Before
9     public void initializare(){
10         app=new App();
11     }

13     @Test
14     public void test(){
15         assertEquals(8,app.cmmdc(56,24));
16     }

19     public static void main(String [] args){
20         org.junit.runner.JUnitCore.main("server.TestApp");
21     }
22 }

```

### Exemplul C.0.6 Testarea clasei *MyMServer* ( 2.2.1).

Primul test se referă la metoda *getServerSocket*. Se verifică afirmațiile:

1. Metoda returnează un obiect  $\neq \text{null}$ .
2. Obiectul returnat este de tip **ServerSocket**.

Al doilea test verifică metoda *myAction*. În acest scop se definește într-o clasă un fir de execuție care în metoda *run()* apelează metoda *myAction*. Acțiunile metodei de testare sunt:

1. Se obține un obiect **ServerSocket**.
2. Se lansează firul de execuție amintit mai sus.
3. Se simulează activitatea unui client.

```

1 package server.impl;
2 import server.impl.MyMServer;
3 import org.junit.Before;
4 import org.junit.Test;
5 import static org.junit.Assert.*;
6 import java.net.ServerSocket;
7 import iserver.IMyMServer;
8 import java.net.Socket;
9 import java.io.DataInputStream;
10 import java.io.DataOutputStream;

```

```

12 public class TestMyMServer{
13     private IMyMServer obj;
14     private static int port=7999;
15     public static final long M=12;
16     public static final long N=15;
17     public static final long RESULT=3;

19     @Before
20     public void initialize(){
21         obj=new MyMServer();
22     }

24     @Test
25     public void test(){
26         int port=8999;
27         Object result=obj.getServerSocket(port);
28         assertNotNull("Must not return a null response",result);
29         assertEquals(ServerSocket.class,result.getClass());
30     }

32     @Test
33     public void testMyAction(){
34         long r=0;
35         ServerSocket ss=obj.getServerSocket(port);
36         EmbeddedThread thread=new EmbeddedThread(ss);
37         thread.start();
38         try(Socket cmmdcSocket = new Socket("localhost",port);
39             DataInputStream in=new DataInputStream(cmmdcSocket.getInputStream());
40             DataOutputStream out=
41                 new DataOutputStream(cmmdcSocket.getOutputStream())){
42             out.writeLong(M);
43             out.writeLong(N);
44             r=in.readLong();
45         }
46         catch(Exception e){
47             System.err.println("Client communication error : "+e.getMessage());
48         }
49         assertEquals(r,RESULT);
50     }

52     public static void main(String[] args){
53         org.junit.runner.JUnitCore.main("server.impl.TestMyMServer");
54     }

56     class EmbeddedThread extends Thread{
57         ServerSocket ss;

59         EmbeddedThread(ServerSocket ss){
60             this.ss=ss;
61         }

63         public void run(){
64             obj.myAction(ss);
65         }
66     }
67 }

```

# Anexa D

## Jurnalizare

*Jurnalizarea* adică afișarea / reținerea rezultatelor sau evenimentelor într-un fișier. Deseori prezintă interes evoluția procesului de calcul prin prisma unor rezultate intermediare. În acest sens se pot utiliza:

- pachetul `java.util.logging` din `jdk`.
- *apache-log4j-2*.
- *slf4j* (Simple Logging Facade for Java), ([www.QOS.ch](http://www.QOS.ch), Quality of Open Software).
- *logback* ([logback.qos.ch](http://logback.qos.ch)).

### Jurnalizare prin `java.util.logging`

Șablonul de programare cu afișarea mesajelor pe ecranul monitorului este

```
1 import java.util.logging.Logger;
2
3 public class Exemplu{
4     static Logger logger = Logger.getLogger(Exemplu.class.getName());
5
6     public static void main(String args[]) {
7         logger.severe("SEVERE : Hello");
8         logger.warning("WARNING : Hello");
9         logger.info("INFO : Hello");
10    }
11 }
```

Programul afișează

```
Jan 23, 2013 2:34:40 PM Exemplu main
SEVERE: SEVERE : Hello
Jan 23, 2013 2:34:40 PM Exemplu main
```

```
WARNING: WARNING : Hello
Jan 23, 2013 2:34:40 PM Exemplu main
INFO: INFO : Hello
```

Dacă dorim ca rezultatele să fie înscrise într-un fișier, de exemplu *logging.txt* atunci clasa de mai sus se modifică în

```
1 import java.util.logging.Logger;
2 import java.util.logging.FileHandler;
3 import java.util.logging.SimpleFormatter;
4 import java.io.IOException;

6 public class Exemplu{
7     static Logger logger = Logger.getLogger(Exemplu.class.getName());

9     public static void main(String[] args) {
10         try{
11             FileHandler loggingFile = new FileHandler("logging.txt");
12             loggingFile.setFormatter(new SimpleFormatter());
13             logger.addHandler(loggingFile);
14         }
15         catch(IOException e){
16             System.out.println(e.getMessage());
17         }
18         logger.severe("SEVERE : Hello");
19         logger.warning("WARNING : Hello");
20         logger.info("INFO : Hello");
21     }
22 }
```

## Jurnalizare prin logback

Jurnalizare cu reținerea rezultatelor în fișier

```
1 package logtest;
2 import org.slf4j.Logger;
3 import org.slf4j.LoggerFactory;

5 public class Exemplu{
6     static Logger logger=LoggerFactory.getLogger("Exemplu");
7     public static void main(String args[]) {
8         logger.trace("TRACE : Hello");
9         logger.debug("DEBUG : Hello");
10        logger.info("INFO : Hello");
11        logger.warn("WARN : Hello");
12        logger.error("ERROR : Hello");
13    }
14 }
```

cu fișierul de configurare

```
1 <configuration>
2   <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
3     <encoder>
4       <pattern>
5         %d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n
6       </pattern>
```

```

7     </encoder>
8 </appender>

10 <appender name="FILE" class="ch.qos.logback.core.FileAppender">
11   <file>results.log</file>
12   <encoder>
13     <pattern>
14       %date %level [%thread] %logger{10} [%file:%line] %msg%n
15     </pattern>
16   </encoder>
17 </appender>

19 <root level="trace">
20   <appender-ref ref="STDOUT" />
21   <appender-ref ref="FILE" />
22 </root>
23 </configuration>

```

## Jurnalizare prin apache-log4j-2

Jurnalizare cu reținerea rezultatelor în fișier

```

1 import org.apache.logging.log4j.LogManager;
2 import org.apache.logging.log4j.Logger;

4 public class Exemplu{
5     static Logger logger = LogManager.getLogger(Exemplu.class);

7     public static void main(String args[]) {
8         logger.warn("WARN : Hello");
9         logger.debug("DEBUG : Hello");
10        logger.info("INFO : Hello");
11        logger.fatal("FATAL : Hello");
12    }
13 }

```

cu fișierul de configurare

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <configuration status="OFF">
3   <properties>
4     <property name="filename">results.log</property>
5   </properties>
6   <appenders>
7     <Console name="Console" target="SYSTEM.OUT">
8       <PatternLayout
9         pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n" />
10    </Console>
11    <File name="File" fileName="${filename}">
12      <PatternLayout
13        pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n" />
14    </File>
15  </appenders>
16  <loggers>
17    <root level="trace">
18      <appender-ref ref="Console" />
19      <appender-ref ref="File" />

```

```
20 |     </root>
21 | </loggers>
22 | </configuration>
```

## Anexa E

# Componentă Java (Java *Bean*)

O componentă Java este o clasă care poate interacționa cu alte componente Java, cu un document `jsp`, etc.

O componenta Java conține cel puțin

- Un constructor fără nici un argument;
- O mulțime de câmpuri declarate `private`;
- Pentru fiecare asemenea câmp

```
private Tip xyz;
```

trebuie definite cel puțin una din metodele

```
public void setXyz(Tip xyz){  
    this.xyz=xyz;  
}
```

```
public Tip getXyz(){  
    return xyz;  
}
```





# Anexa F

## Serializare fără XML

Scopul serializării unor date are ca scop obținerea unei reprezentări text în vederea transmiterii prin rețea. Orice soluție trebuie să ofere și posibilitatea refacerii datelor din forma serializată. Serializarea se poate obține prin

- Prin încorporarea datelor într-un document XML. În acest sens amintim *Java Architecture for XML Binding - JAXB*.
- Metode / tehnologii fără XML:
  - *YAML Ain't Markup Language - YAML*<sup>1</sup>
  - *JavaScript Object Notation - JSON*

Se consideră că JSON este o restricție de YAML.

---

<sup>1</sup>Acronimul YAML are și altă semnificație *Yet Another Multicolumn Layout*, desemnând o tehnologie JavaScript-CSS.

## F.1 YAML Ain't Markup Language - YAML

Marcaj YAML	Tip Java
Marcaj standard YAML	
!!null	null
!!bool	Boolean
!!int	Integer, Long, BigInteger
!!float	Double
!!binary	String
!!timestamp	java.util.Date, java.sql.Date, java.sql.Timestamp
!!omap, !!pairs	List of Object[ ]
!!set	Set
!!str	String
!!seq	List
!!map	Map

În Java vom exemplifica cu produsul *snakeyaml*.

### Exemplul F.1.1

Pornim de la componenta Java

```

1 public class Disciplina{
2     private String nume;
3     Disciplina(){
4
5     Disciplina(String nume){
6         this.nume=nume;
7     }
8     public String getNume(){
9         return nume;
10    }
11    public void setNume(String nume){
12        this.nume=nume;
13    }
14 }
```

Se instanțiază 3 obiecte care vor fi convertite în reprezentări YAML - stringuri ce sunt reținute într-un obiect de tip `java.util.Map`. În final acest obiect este reprezentat YAML, care este salvat într-un fișier.

```

1 import java.util.Map;
2 import java.util.HashMap;
3 import org.yaml.snakeyaml.Yaml;
4 import org.yaml.snakeyaml.TypeDescription;
5 import java.io.PrintWriter;
6
7 public class Generare{
8     //@SuppressWarnings("unchecked")
```

```

9  public static void main(String[] args) {
10     Yaml yaml = new Yaml();
11     Map<String,String> data = new HashMap<String,String>();

13     Disciplina an=new Disciplina("Analiza numerica");
14     String san=yaml.dump(an);
15     data.put("an",san);

17     Disciplina pd=new Disciplina("Programare distribuita");
18     String spd=yaml.dump(pd);
19     data.put("pd",spd);

21     Disciplina sm=new Disciplina("Soft matematic");
22     String ssm=yaml.dump(sm);
23     data.put("sm",ssm);

25     System.out.println("Serializarea datelor");
26     System.out.println("Continutul obtinut:\n");
27     try{
28         PrintWriter pw=new PrintWriter("file.yaml");
29         String objYAML=yaml.dump(data);
30         System.out.println(objYAML);
31         pw.write(objYAML);
32         pw.flush();
33     }
34     catch(Exception e){
35         e.printStackTrace();
36     }
37 }
38 }

```

Fișierul creat anterior este preluat, conținutul reconvertit în obiect de tip Map, iar componentele înmagazinate sunt retransformate în obiecte.

```

1  import java.io.File;
2  import java.io.FileInputStream;
3  import java.io.InputStream;
4  import java.util.Map;
5  import org.yaml.snakeyaml.Yaml;
6  import java.io.PrintWriter;
7  import org.yaml.snakeyaml.constructor.Constructor;
8  import java.util.Collection;
9  import java.util.Iterator;

11 public class Utilizare{
12     //@SuppressWarnings("unchecked")
13     public static void main(String[] args) {
14         Constructor constructor=new Constructor(Disciplina.class);

16         try{
17             InputStream input = new FileInputStream("file.yaml");
18             Yaml yaml = new Yaml();
19             Object data = yaml.load(input);
20             System.out.println("Continutul incarcat:\n");
21             System.out.println(data.toString());

23             Map<String,String> map = (Map)data;
24             Collection<String> discipline=map.values();
25             Iterator<String> iter=discipline.iterator();

```

```
26     Yaml yaml1 = new Yaml(constructor);
27     System.out.println("Regasirea datelor:\n");
28     while(iter.hasNext()){
29         String sobj=iter.next();
30         // Abordare urata !!
31         sobj=sobj.substring(14,sobj.length()-2);
32         Disciplina obj=(Disciplina)yaml1.load(sobj);
33         System.out.println(obj.getNume());
34     }
35 }
36 catch(Exception e){
37     e.printStackTrace();
38 }
39 }
40 }
```

## F.2 JavaScript Object Notation - JSON

JSON oferă o modalitate simplă (mai simplă chiar decât XML) pentru schimbul de date dintre un server și un client.

Pentru reprezentarea datelor în JSON se utilizează structurile de date:

- colecție de atribute, adică perechi (nume, valoare). O colecție de atribute este denumit obiect JSON.
- șir de valori.

Aceste structuri de date sunt prezente în toate limbajele de programare de uz general.

O colecție de atribute se reprezintă prin

```
{numeAtribut:valAtribut,numeAtribut:valAtribut,...}
```

Un șir de valori se reprezintă prin

```
[valoare,valoare,...]
```

*valAtribut*, *valoare* poate fi un string, număr, true, false, null, o colecție sau un șir.

## JSON în JavaScript

Utilizarea entităților JSON în javascript este exemplificat în aplicația următoare.

### Exemplul F.2.1

```

1 <HTML>
2 <HEAD>
3   <TITLE>Primul exemplu JavaScript</TITLE>
4   <SCRIPT LANGUAGE="JavaScript">
5     <!--
6     var myJSONObj=[{"disciplina":"Analiza Numerica"},
7     {"disciplina":"Programare distribuita"},
8     {"disciplina":"Soft matematic"}];
9     for (var i=0;i<myJSONObj.length;i++){
10      document.writeln("<br>");
11      document.writeln(myJSONObj[i].disciplina);
12    }
13    document.writeln("<br>");
14    var myObj=eval(myJSONObj);
15    document.writeln(myObj.toString());
16    for (var i=0;i<myObj.length;i++){
17      document.writeln("<br>");
18      document.writeln(myObj[i].disciplina);
19    }
20    //-->
21  </SCRIPT>
22 </HEAD>
23 <BODY>
24 </BODY>
25 </HTML>

```

## JSON în Java

### *google-gson*

Analogul aplicației javascript de mai sus, poate fi

### Exemplul F.2.2

```

1 import com.google.gson.Gson;
2 import com.google.gson.reflect.TypeToken;
3 import java.lang.reflect.Type;
4 import java.util.Collection;
5 import java.util.Iterator;
6
7 class Disciplina{
8   private String nume;
9   Disciplina(){
10
11   Disciplina(String nume){
12     this.nume=nume;
13   }
14   public String getNume(){
15     return nume;
16   }
17 }

```

```

19 public class TestGSON{
20     public static void main(String [] args){
21         Gson gson=new Gson();
22         Disciplina an=new Disciplina("Analiza numerica");
23         Disciplina pd=new Disciplina("Programare distribuita");
24         Disciplina sm=new Disciplina("Soft matematic");
25         Disciplina [] discipline={an,pd,sm};
26         String json=gson.toJson(discipline);
27         System.out.println(json);
28         Type collectionType = new TypeToken<Collection<Disciplina>>().getType();
29         Collection<Disciplina> d = gson.fromJson(json, collectionType);
30         Iterator<Disciplina> iter=d.iterator();
31         while(iter.hasNext()){
32             Disciplina dis=iter.next();
33             System.out.println(dis.getNume());
34         }
35     }
36 }

```

### *javax.json*

Interfața `javax.json.JsonValue` introduce obiectele nemodificabile (*immutable*) `JsonArray`, `JsonObject`, `JsonString`, `JsonNumber`, `JsonValue.TRUE`, `JsonValue.FALSE`, `JsonValue.NULL`.

Obiectele se instanțiază prin intermediul unor metode statice ale clasei `javax.json.Json`.

Există structura de interfețe

<code>JsonValue</code>	<code>JsonStructure</code>	<code>JsonArray</code>
		<code>JsonObject</code>
	<code>JsonString</code>	
	<code>JsonNumber</code>	acoperă tipurile de date Java numerice <code>BigDecimal</code> , <code>BigInteger</code> , <code>int</code> , <code>long</code> , <code>double</code>

Clasa `javax.json.Json`

Metode

- `static JsonObjectBuilder createObjectBuilder()`
- `static JsonArrayBuilder createArrayBuilder()`
- `static JsonWriter createWriter(java.io.Writer writer)`
- `static JsonReader createReader(java.io.Reader reader)`

## Interfața javax.json.JsonObjectBuilder

### Metode

- `JsonObjectBuilder add(String name, TipJson value)`  
*TipJson* ∈ {BigDecimal, BigInteger, int, long, double, boolean, JsonObjectBuilder, JsonArrayBuilder, JsonValue, String}.
- `JsonObjectBuilder addNull()`
- `JsonObject build()`

### Șablon de utilizare

```
JsonObject jsonObject=Json.createObjectBuilder()  
    .add("name", value)  
    .  
    .  
    .build();
```

## Interfața javax.json.JsonArrayBuilder

### Metode

- `JsonArrayBuilder add(TipJson value)`  
*TipJson* ∈ {BigDecimal, BigInteger, int, long, double, boolean, JsonObjectBuilder, JsonArrayBuilder, JsonValue, String}.
- `JsonArrayBuilder addNull()`
- `JsonArray build()`

### Șablon de utilizare

```
JsonArray jsonArray=Json.createArrayBuilder()  
    .add(value)  
    .  
    .  
    .build();
```

## Interfața javax.json.JsonWriterBuilder

### Metode

- `void writeArray(JsonArray array)`
- `void writeObject(JsonObject object)`
- `void close()`

### Șablon de utilizare

```
PrintWriter printWriter=new PrintWriter(System.out)  
JsonWriter jsonWriter=Json.createWriter(printWriter);  
jsonWriter.writeArray(jsonArray);  
jsonWriter.close();
```

## Interfața `javax.json.ReaderBuilder`

### Metode

- `void readArray()`
- `void readObject()`
- `void close()`

### Șablon de utilizare

```
String string=. . .
JsonReader jsonReader = Json.createReader(new StringReader(string));
JsonArray array = jsonReader.readArray();
jsonReader.close();
```

### Exemplul F.2.3 *Crearea unui fișier json.*

```
1 import javax.json.JsonArray;
2 import javax.json.JsonArrayBuilder;
3 import javax.json.JsonObject;
4 import javax.json.JsonObjectBuilder;
5 import javax.json.JsonWriter;
6 import javax.json.Json;
7 import java.io.PrintWriter;
8 import java.io.IOException;

10 public class GenerateJSON{
11     public static void main(String[] args){
12         JsonArray jsonArray=Json.createArrayBuilder()
13             .add(Json.createObjectBuilder()
14                 .add("nume", "Analiza numerica"))
15             .add(Json.createObjectBuilder()
16                 .add("nume", "Programare distribuita"))
17             .add(Json.createObjectBuilder()
18                 .add("nume", "Soft matematic"))
19             .add(100)
20             .add("javax.json")
21             .add(Json.createArrayBuilder()
22                 .add(1)
23                 .add(2)
24                 .add(3))
25             .add(Json.createArrayBuilder()
26                 .add(4)
27                 .add(5)
28                 .add(6))
29             .build();
30         System.out.println("System.out : "+jsonArray);
31         String fileName="exemplu.json";

33         try{
34             JsonWriter jsonWriter=Json.createWriter(new PrintWriter(fileName));
35             jsonWriter.writeArray(jsonArray);
36             jsonWriter.close();
37         }
```



```

38     catch(Exception e){
39         System.out.println(e.getMessage());
40     }
41     JsonWriter jsonWriter=Json.createWriter(new PrintWriter(System.out));
42     jsonWriter.writeArray(jsonArray);
43     jsonWriter.close();
44 }
45 }

```

**Exemplul F.2.4** *Consultarea fişierului json creat în exemplul anterior.*

```

1  import javax.json.JsonArray;
2  import javax.json.JsonObject;
3  import javax.json.JsonReader;
4  import javax.json.Json;
5  import javax.json.JsonValue;
6  import javax.json.JsonString;
7  import javax.json.JsonNumber;
8  import java.io.FileReader;
9  import java.io.IOException;
10 import java.util.Iterator;
11 import java.util.Map;
12 import java.util.Set;

14 public class ReadJSON{
15     public static void main(String[] args){
16         String fileName="exemplu.json";
17         String path="d:\\mk\\DISTR2\\GSON\\JEE\\ex1\\";
18         JsonArray array=null;
19         try{
20             JsonReader jsonReader =
21                 Json.createReader(new FileReader(path+fileName));
22             array = jsonReader.readArray();
23             jsonReader.close();
24         }
25         catch(IOException e){
26             System.out.println("Ex : "+e.getMessage());
27         }
28         analyse(array);
29     }

31     private static void analyse(JsonArray v){
32         Iterator<JsonValue> iterator=v.iterator();
33         while(iterator.hasNext()){
34             JsonValue value=(JsonValue)iterator.next();
35             if(value instanceof JsonArray){
36                 JsonArray array=(JsonArray)value;
37                 analyse(array);
38             }
39             if(value instanceof JsonObject){
40                 JsonObject obj=(JsonObject)value;
41                 analyseJsonObject(obj);
42             }
43             if(value instanceof JsonString){
44                 JsonString string=(JsonString)value;
45                 String s=string.getString();
46                 System.out.println(s);

```

```
47     }
48     if (value instanceof JsonNumber) {
49         JsonNumber number = (JsonNumber) value;
50         double d = number.doubleValue();
51         System.out.println(d);
52     }
53 }
54 }

56 private static void analyseJsonObject(JsonObject obj) {
57     Map<String, JsonValue> object = (Map) obj;
58     Set<String> keys = object.keySet();
59     Iterator<String> iter = keys.iterator();
60     while (iter.hasNext()) {
61         String name = iter.next();
62         System.out.println();
63         System.out.println("JsonObject name : " + name);
64         JsonValue vv = (JsonValue) object.get(name);
65         if (vv instanceof JsonArray) {
66             JsonArray array = (JsonArray) vv;
67             analyse(array);
68         }
69         if (vv instanceof JsonObject) {
70             JsonObject o = (JsonObject) vv;
71             analyseJsonObject(o);
72         }
73         if (vv instanceof JsonString) {
74             JsonString string = (JsonString) vv;
75             String s = string.getString();
76             System.out.println(s);
77         }
78         if (vv instanceof JsonNumber) {
79             JsonNumber number = (JsonNumber) vv;
80             double d = number.doubleValue();
81             System.out.println(d);
82         }
83     }
84 }
85 }
```

# Anexa G

## Adnotări

O *adnotare* este o completare, o notă sau o însemnare care explică sau întregeste un text.

O *metadată* este o adnotare a unei date.

În Java o adnotare (*annotation*) este o metadata a unui element de cod (identificator al unei entități din codul Java).

O adnotare poate să-și facă efectul:

- asupra codului sursă, înaintea compilării;
- asupra codului obiect, după compilare, dar înaintea executării;
- în timpul execuției codului.

Din punctul de vedere al sintaxei interesează

- definirea unei adnotări;
- declararea unei adnotări;
- procesarea unei adnotări.

### G.1 Definirea unei adnotări

Sintaxa utilizată este

```
import java.lang.annotation.*;
```

```
modifier @interface NumeAdnotare{  
    declarare.Element_1
```

```
        declarare_Element_2  
        . . .  
    }
```

unde *declarare\_Element* poate fi:

```
tip numeElement();  
tip numeElement() default valoare;
```

iar *tip* poate fi

- predefinit (int, short, long, byte, char, float, double, boolean);
- String
- Class
- enum
- adnotare
- tablou ale cărei elemente sunt de un tip precizat anterior

Adnotarea se salvează ca fișier text, sub numele *NumeAdnotare.java*. După numărul elementelor declarate într-o adnotare, acesta poate fi

- 0 - caz în care adnotarea este de tip *marker*;
- 1 sau mai multe elemente (*single-element* / *multi-value annotation*).

## G.2 Declararea unei adnotări

O adnotare se poate referi la un pachet, clasă, interfață, metodă, câmp.

Înainte de declararea elementului asupra căruia acționează, adnotarea se indică prin

```
@NumeAdnotare(numeElement_1=valoare,numeElement_2=valoare,...)
```

## G.3 Procesarea unei adnotări

Java declară adnotările

Adnotări regulate	Meta-adnotări
Override	Target
Deprecated	Retention
SuppressWarnings	Documented
	Inherited

### Adnotarea Override

Adnotarea **Override** precizează faptul că se suprascrie un element al clasei părinte.

```

1 import java.util.Date;
2 public class TestOverride extends Date{
3     @Override
4     public String toString(){
5         return super.toString()+" TestOverride";
6     }
7
8     public static void main(String [] args){
9         Date d=new TestOverride();
10        System.out.println(d.toString());
11    }
12 }
```

Dacă în locul liniei 9 se pune `Date d=new Date();` atunci nu mai apare mesajul *TestOverride*.

### Adnotarea Deprecated

Adnotarea **Deprecated** are ca efect afișarea unui mesaj de avertisment în momentul compilării.

Exemplificăm cu clasele

```

1 public class MyDeprecated{
2     @Deprecated
3     public void doJob(){
4         System.out.println("This is deprecated");
5     }
6 }
```

```

1 public class TestDeprecated{
2     public static void main(String [] args){
3         MyDeprecated obj=new MyDeprecated();
4         obj.doJob();
5     }
6 }
```

Mesajul de avertisment este

Note: TestDeprecated.java uses or overrides a deprecated API.  
Note: Recompile with -Xlint:deprecation for details.

Comentând linia adnotării are ca efect la o nouă compilare dispariția mesajelor de avertisment.

## Adnotarea SuppressWarnings

Adnotarea `SuppressWarnings` inhibă afișarea mesajelor de avertisment. Reluăm exemplul anterior schimbând clasa *Test*. În urma compilării nu mai apar mesajele de avertisment menționate mai sus.

```
1 @SuppressWarnings("deprecation")
2 public class TestSuppressWarnings{
3
4     public static void main(String [] args){
5         MyDeprecated obj=new MyDeprecated();
6         obj.doJob();
7     }
8 }
```

## Adnotarea Target

Adnotarea `Target` precizează elementul asupra căreia acționează:

- `ElementType.TYPE`
- `ElementType.FIELD`
- `ElementType.METHOD`
- `ElementType.PARAMETER`
- `ElementType.CONSTRUCTOR`
- `ElementType.LOCAL_VARIABLE`
- `ElementType.ANNOTATION_TYPE`

## Adnotarea Retention

Adnotarea `Retention` precizează momentul acțiunii adnotării:

- `RetentionPolicy.SOURCE`
- `RetentionPolicy.CLASS`
- `RetentionPolicy.RUNTIME`

## Adnotarea Documented

Adnotarea `Documented` are ca efect menționarea adnotării în documentul obținut prin `javadoc`.

Fie clasele

```
1 import java.lang.annotation.Documented;
3 @Documented
4 public @interface MyDocumented{}
```

și

```
1 public class TestDocumented{
2     public static void main(String [] args){
3         new TestDocumented().doDocumented();
4     }
6     @MyDocumented
7     public void doDocumented(){
8         System.out.println("Test Documented");
9     }
10 }
```

`javadoc` se lansează prin

```
md docs
javadoc -d docs *Documented.java
```

Procesarea unei adnotări se referă în primul rând la regăsirea în momentul execuției a valorilor elementelor adnotării. În funcție de valorile regăsite se pot implementa acțiuni specifice.

Procesarea unei adnotări se bazează pe metodele interfeței `AnnotationElement`, implementată de clasele `Class`, `Constructor`, `Field`, `Method`, `Package`:

- `<T extends Annotation> T getAnnotation(Class<T> annotationClass)`
- `Annotation[] getAnnotations()`
- `Annotation[] getDeclaredAnnotations()`
- `boolean isAnnotationPresent(Class<? extends Annotation> annotationClass)`

Considerăm adnotarea

```
1 import java.lang.annotation.Retention;
2 import java.lang.annotation.Target;
3 import java.lang.annotation.RetentionPolicy;
4 import java.lang.annotation.ElementType;
6 @Retention(RetentionPolicy.RUNTIME)
```

```

7 | @Target (ElementType.METHOD)
8 | public @interface MyAnnotation{
9 |     String doAction() default "";
10 |     int index() default 0;
11 | }

```

pe care o utilizăm în clasa

```

1 | import java.lang.reflect.Method;
2 |
3 | public class TestAnnotation{
4 |     public static void main(String [] args){
5 |         TestAnnotation obj=new TestAnnotation ();
6 |         obj.verif(obj);
7 |     }
8 |
9 |     @MyAnnotation(doAction="XYZ",index=1)
10 |    public void verif(Object o){
11 |        try{
12 |            Class cl=o.getClass ();
13 |            Method m=cl.getMethod("verif",
14 |                new Class [] {(new Object()).getClass ()});
15 |            if(m.isAnnotationPresent(MyAnnotation.class)){
16 |                MyAnnotation a=m.getAnnotation(MyAnnotation.class);
17 |                if(a!=null){
18 |                    String numeElement=a.doAction ();
19 |                    System.out.println (numeElement);
20 |                    int index=a.index ();
21 |                    System.out.println (index);
22 |                }
23 |            }
24 |        }
25 |        catch(Exception e){
26 |            System.out.println ("MyEx : "+e.getMessage ());
27 |        }
28 |    }
29 | }

```

Clasa `java.lang.Class` furnizează o reprezentare (reflectare) a unei clase în timpul execuției. Metoda `public Method getMethod(String name, Class<?>... parameterTypes) throws NoSuchMethodException, SecurityException` furnizează o reprezentare (reflectare) a unei metode în timpul execuției unui program.



# Anexa H

## Utilizarea SGBD în Java

Scopul acestei anexe este prezentarea bazelor utilizării unui Sistem de Gestiune a Bazelor de Date (SGBD - *Data Bases Management System* - DBMS) din Java. Exemplificăm modul de operare și utilizare pentru crearea și exploatarea unei baze de date corespunzătoare unei agende de adrese e-mail.

### H.1 *Derby / Javadb*

**Instalarea produsului** constă în dezarhivarea fișierului descărcat.

**Utilizarea produsului.** Va fi utilizată varianta de *rețea* bazată pe un server al SGBD care utilizează implicit portul 1527.

Se întreprind următoarele operații:

1. Lansarea serverului Derby / Javadb:

```
set JAVA_HOME=. . .
set DB_HOME=. . .
set PATH=%DB_HOME%\bin;%PATH%
startNetworkServer.bat -h 0.0.0.0 -noSecurityManager
```

Prezența opțiunii `-h 0.0.0.0` asigură accesul la serverul SGBD de pe orice calculator.

2. Crearea bazei de date se va face utilizând utilitarul *ij* din distribuția Derby / Javadb. Acesta se lansează prin:

```
set JAVA_HOME=. . .
set DB_HOME=. . .
set PATH=%DB_HOME%\bin;%PATH%
ij.bat
```

### Exemplul H.1.1

Baza de date *AgendaEMail* se crează executând

`run 'CreateAgendaE.sql';`

unde fișierul *CreateAgendaE.sql* este

```

1 connect 'jdbc:derby:AgendaEMail;create=true';
2 create table adrese(
3   id int generated always as identity(start with 1, increment by 1)
4   primary key,
5   nume char(20) not null,
6   email char(30) not null
7 );

```

și încărcarea cu date

`run 'ValuesAgendaE.sql';`

cu fișierul *ValuesAgendaE.sql*

```

1 insert into adrese(nume,email) values('aaa','aaa@yahoo.com'),
2   ('bbb','bbb@gmail.com'),('ccc','ccc@unitbv.ro'),
3   ('aaa','xyz@unitbv.ro');

```

## H.2 *mysql*

**Instalarea produsului.** Pentru instalare s-a descărcat varianta fără instalare automată *mysql-\*. \*-win\*.zip*. Acest fișier se dezarchivează într-un catalog `MYSQL_HOME`.

Dezarchivarea este urmată de inițializare

- Varianta nesecurizată, adică fără utilizarea parolelor:

```

set MYSQL_HOME=. . .
set PATH=%MYSQL_HOME%\bin;%PATH%
mysqld --initialize-insecure

```

- Varianta securizată:

```

set MYSQL_HOME=. . .
set PATH=%MYSQL_HOME%\bin;%PATH%
mysqld --initialize

```

Pentru *root* se va genera o parolă inițială și efemeră care trebuie modificată

```
ALTER USER 'root'@'localhost' IDENTIFIED BY 'new_password';
```

în cadrul unei sesiuni mysql

```
set MYSQL_HOME=. . .
set PATH=%MYSQL_HOME%\bin;%PATH%
mysql -u root -p
```

Implicit, serverul mysql utilizează portul 3306, iar fișierele bazelor de date vor fi găzduite în catalogul `MYSQL_HOME\data`.

Pentru utilizarea în aplicații Java trebuie descărcat un conector *mysql-connector-java-\*.\*.tar.gz*, conținând fișierul *mysql-connector-java-\*.\*.bin.jar*.

### Utilizarea produsului.

Se întreprind următoarele operații:

1. Lansarea serverului *mysql*:

```
set MYSQL_HOME=. . .
set PATH=%MYSQL_HOME%\bin;%PATH%
mysqld
```

### 2. Exemplul H.2.1

Crearea bazei de date *AgendaEMail* se va face prin intermediul fișierului de comenzi

```
set MYSQL_HOME=d:\mysql-*win*\bin
set path=%MYSQL_HOME%;%PATH%
mysql -u root < CreateAgendaE.sql
mysql -u root < ValuesAgendaE.sql
```

unde scriptul *CreateAgendaE.sql* este

```
1 create database AgendaEMail;
2 use AgendaEMail;

4 create table adrese(
5     id int primary key auto_increment not null,
6     nume char(20) not null,
7     email char(30) not null
8 );
```

iar scriptul de populare cu date (*ValuesAgendaE.sql*) este

```
1 use AgendaEMail;
2 insert adrese values(1,"aaa","aaa@yahoo.com");
3 insert adrese values(2,"bbb","bbb@gmail.com");
4 insert adrese values(3,"ccc","ccc@unitbv.ro");
5 insert adrese values(1,"aaa","xyz@unitbv.ro");
```

### 3. Serverul *mysql* se oprește prin

```
set MYSQL_HOME=d:\mysql-*--win*
set PATH=%MYSQL_HOME%\bin;%PATH%
mysqladmin -u root shutdown
```

Dacă se utilizează varianta securizată atunci comenzile *mysql* de la pct. 2 și 3 trebuie să conțină opțiunea *-p*. Prin această opțiune se cere autentificarea prin introducerea parolei.

## H.3 Șablonul de utilizare a unei baze de date într-un program Java

Interacțiunea cu baze de date relaționale implică:

1. Stabilirea corespondenței dintre date aflate în obiecte și atribute / tabele (*object to relational database mapping*).
2. Apelarea acțiunilor CRUD (*Create, Read, Update, Delete*).

Limbaajul SQL (*Structured Query Language*) este dependent de SGBD utilizat. Dezvoltarea interacțiunii dintre un program Java (client) cu o bază de date dintr-un SGBD s-a născut din dorința de a asigura independența *stratului* Java de SGBD. Soluția găsită constă în introducerea unui strat suplimentar, între Java și SGBD care asigură corespondența între obiectele Java cu tabelele unei baze de date și permite o configurare simplă la schimbarea SGBD. Astfel se folosește terminologia de aplicație multistrat.

Materializarea acestor idei (*Object Relational Mapping* - ORM) se află în

- interfața de programare (API) *Java Persistence API* (JPA);

Există mai multe implementări JPA.

- interfața de programare (API) *Java Data Object* (JDO);
- produsul *Hibernate*.

*Hibernate* conține și o implementare JPA.

- *apache-empire*

- *ebean*

Scopul urmărit este realizarea trecerii de la un SGBD la altul prin modificări minime în stratul intermediar.

În cele ce urmează vom face abstracție de modelele menționate anterior și vom trata în modul cel mai simplu realizarea unei aplicații care interacționează cu o bază de date gestionată de un SGBD.

Pentru a avea acces la o bază de date trebuie stabilită o conexiune la acea bază de date. În acest sens este necesar cunoașterea:

- driver-ului de acces la sistemul de gestiune a bazei de date (SGBD)

Tip SGBD	Driver	Fișierul driver-ului
access	sun.jdbc.odbc.JdbcOdbcDriver	
mysql	com.mysql.jdbc.Driver	mysql-connector-java-*. *-bin.jar (www.mysql.com)
derby javadb	org.apache.derby.jdbc.ClientDriver	derbyclient.jar (distribuția derby)
postgresql	org.postgresql.Driver	postgresql-*. *-jdbc4.jar
hypeqsql	org.hsqldb.jdbcDriver	hsqldb.jar
oraclexe	oracle.jdbc.driver.OracleDriver	ojdbc14.jar

Dacă într-un servlet se realizează o conexiune la o bază de date atunci fișierul driver-ului trebuie copiat în catalogul `lib` al aplicației.

- adresa URL a bazei de date (String *URLBazaDate*), sub forma

Tip SGBD	Referință Baza de Date
mysql	jdbc:mysql://host:3306/numeBazaDate
derby / javadb	jdbc:derby://host:1527/numeBazaDate
postgresql	jdbc:postgresql://host:5432/numeBazaDate
hypersql	jdbc:hsqldb:hsq://host/numeBazaDate
oracle	jdbc:oracle:thin:@host:1521:XE

Șablonul de prelucrare este

```
String URLBazaDate = . . .
String jdbcDriver = . . .
Connection con=null;
try{
    Class.forName(jdbcDriver).newInstance();
    con=DriverManager.getConnection(URLBazaDate);
    ...
}
```

```

    }
    catch(ClassNotFoundException e){. . .}
    catch(SQLException e){. . .}

```

Anumite SGBD asigură accesul la o bază de date dacă sunt fixați parametrii *username* și *password*. În acest caz se programează

```
con=DriverManager.getConnection(URLBazaDate,username,password);
```

Odată conexiunea cu baza de date stabilită se generează un obiect de tip **Statement** prin intermediul căruia se execută interogarea SQL.

```

Statement instructiune=con.createStatement();
String sql=. . . //fraza select;

```

Rezultatele interogării bazei de date se obține prin

```

try{
    ResultSet rs=instructiune.executeQuery(sql);
    while(rs.next()){
        prelucrarea rezultatului
    }
}
catch(SQLException e){...}

```

### Exemplul H.3.1

O interogare simplă a bazei de date *AgendaEMail* se realizează cu programul

```

1 import java.sql.Connection;
2 import java.sql.DriverManager;
3 import java.sql.ResultSet;
4 import java.sql.Statement;
5 import java.util.Scanner;
6 import java.util.InputMismatchException;

8 public class AgendaE{
9     private static String jdbcURLDerby =
10         "jdbc:derby://localhost:1527/AgendaEMail";
11     private static String jdbcDriverDerby =
12         "org.apache.derby.jdbc.ClientDriver";

14     private static String jdbcURLMysql =
15         "jdbc:mysql://localhost:3306/AgendaEMail?user=root";
16     private static String jdbcDriverMysql =
17         "com.mysql.jdbc.Driver";

19     public static void main(String [] args){

```

```

20 String dbms=null, username=null, password=null, jdbcURL=null;
21 switch( args.length ){
22     case 0:
23         System.out.println("At least one argument required");
24         System.out.println("DBMS username password");
25         System.out.println("DBMS derby, mysql");
26         System.exit(0);
27         break;
28     case 1:
29         dbms=args[0];
30         break;
31     case 2:
32         dbms=args[0];
33         username=args[1];
34         password="";
35         break;
36     default:
37         dbms=args[0];
38         username=args[1];
39         password=args[2];
40 }
41 Connection conn = null;
42 try {
43     switch(dbms){
44         case "derby":
45             Class.forName(jdbcDriverDerby).newInstance();
46             jdbcURL=jdbcURLDerby;
47             break;
48         case "mysql":
49             Class.forName(jdbcDriverMysql).newInstance();
50             jdbcURL=jdbcURLMysql;
51             break;
52         default:
53             System.out.println("Unknown DBMS...");
54             System.exit(0);
55     }
56     System.out.println("jdbcURL="+jdbcURL);
57     if(password==null)
58         conn = DriverManager.getConnection(jdbcURL);
59     else
60         conn=DramaManager.getConnection(jdbcURL, username, password);
61
62     Statement instructiune=conn.createStatement();
63     Scanner scanner=new Scanner(System.in);
64     int prel,no;
65     String ch="Y",nume="",email="",sql="";
66     ResultSet rs=null;
67
68     while(ch.startsWith("Y")){
69         do{
70             System.out.println("Continue ? (Y/N)");
71             ch=scanner.next().toUpperCase();
72         }
73         while((!ch.startsWith("Y"))&&(!ch.startsWith("N")));
74         if(ch.startsWith("Y")){
75             System.out.println("Natura interogarii ?");
76             System.out.println("(Dupa nume:1,Dupa email:2)");
77             do{
78                 prel=0;

```

```

79         try{
80             prel=scanner.nextInt();
81         }
82         catch(InputMismatchException e){}
83     }
84     while((prel<1)&&(prel>2));
85     switch(prel){
86         case 1 :
87             System.out.println("Numele");
88             nume='\''+scanner.next().trim()+'\'';
89             sql="select * from adrese where nume="+nume;
90             rs=instrutiune.executeQuery(sql);
91             break;
92         case 2 :
93             System.out.println("Email");
94             email='\''+scanner.next().trim()+'\'';
95             sql="select * from adrese where email="+email;
96             rs=instrutiune.executeQuery(sql);
97             break;
98         default: System.out.println("Comanda eronata");
99     }
100     if(rs!=null){
101         System.out.println("Results :");
102         while(rs.next()){
103             System.out.println("id=" + rs.getInt("id"));
104             System.out.println("nume=" + rs.getString("nume"));
105             System.out.println("email=" + rs.getString("email"));
106             System.out.println("_____");
107         }
108     }
109     else{
110         System.out.println("No item found !");
111     }
112 }
113 }
114 }
115 catch(Exception e) {
116     // handle the exception
117     e.printStackTrace();
118 }
119 }
120 }

```

Fraza *select* și interogarea se mai putea programa prin

```

String sql="select * from adrese where nume =?";
PreparedStatement prepStmt=con.prepareStatement(sql);
prepStmt.setString(1,nume);
ResultSet rs=prepStmt.executeQuery();
. . .
prepStmt.close();

```

În acest caz, valoarea variabilei *nume* este fără apostroafe.



Compilarea și execuția programului necesită declararea în variabila `classpath` a fișierelor

- Varianta *Derby*  
`derbyclient.jar` din catalogul `%DERBY_INSTALL%\lib`.
- Varianta *mysql*  
`%MYSQL_CONNECTOR_JAVA_HOME%\mysql-connector-java-*.*. *-bin.jar`

Execuția programului presupune serverul SGBD activ.



# Anexa I

## Injectarea dependențelor

Injectarea dependențelor (*Dependency Injection - DI*) costă în oferirea spre utilizare a unor obiecte instanțiate de mediul de lucru (server Web, server de aplicații, container specializat) de către o clasă.

Injectarea dependențelor este o tehnică uzuală în JEE, dar poate fi programată și utilizată și înafara unui cadru JEE. Produse informatice ce asigură această facilitare sunt:

- *Weld* realizat de *Jboss - RedHat* și utilizat de *Glassfish*.
- *Guice* realizat de *Google*.

### I.1 Weld

Exemplificăm pe aplicația simplă de calcul a celui mai mare divizor comun. Obiectul ce se va injecta este instanță a clasei

```
1 package cmmdc;  
3 public class Cmmdc{  
4     public long cmmdc(long m,long n){. . .}  
5 }
```

### Aplicație de sine stătătoare

Structura aplicației este

```
catalogul_aplicatiei  
|--> cmmdc  
|    | Cmmdc.class  
|    | ApelCmmdc.class
```

```
|--> META-INF
|    |    beans.xml
|    Client.class
```

Codurile claselor:

- Clasa *ApelCmmdc*

```
1 package cmmdc;
2 import javax.inject.Inject;

4 public class ApelCmmdc{

6     @Inject
7     Cmmdc obj;

9     public long compute(long m, long n) {
10         return obj.cmmdc(m, n);
11     }
12 }
```

- Clasa *Client*

```
1 import java.util.Scanner;
2 import org.jboss.weld.environment.se.Weld;
3 import org.jboss.weld.environment.se.WeldContainer;
4 import cmmdc.ApelCmmdc;

6 public class Client{
7     public static void main(String[] args){
8         Scanner scanner=new Scanner(System.in);
9         long m,n,r;
10        System.out.println("m=");
11        m=scanner.nextLong();
12        System.out.println("n=");
13        n=scanner.nextLong();
14        WeldContainer weld = new Weld().initialize();
15        ApelCmmdc obj = weld.instance()
16            .select(ApelCmmdc.class)
17            .get();
18        r=obj.compute(m,n);
19        System.out.println("Cmmdc : "+r);
20    }
21 }
```

Fișierul `beans.xml` este

```
1 <beans></beans>
```

## Servlet

Structura aplicației Web:

```

contextul_Web
|--> WEB-INF
|   |--> classes
|   |   |   Cmmdc.class
|   |   |   CmmdcWebServlet.class
|   |--> lib
|   |   |   weld-servlet.jar
|   |   |   web.xml
|   |   |   beans.xml
|   |   |   index.html

```

Servletul are codul

```

1 import java.io.IOException;
2 import javax.servlet.ServletException;
3 import javax.servlet.http.HttpServlet;
4 import javax.servlet.http.HttpServletRequest;
5 import javax.servlet.http.HttpServletResponse;
6 import javax.servlet.ServletConfig;
7 import javax.servlet.annotation.WebServlet;
8 import java.io.PrintWriter;
9 import javax.inject.Inject;
10 import cmmdc.Cmmdc;

12 @WebServlet(urlPatterns = "/cmmdc")

14 public class CmmdcWeldServlet extends HttpServlet{
15     @Inject
16     private Cmmdc obj;

18     public void doGet(HttpServletRequest req, HttpServletResponse res)
19         throws ServletException, IOException{
20         String sm=req.getParameter("m"),sn=req.getParameter("n");
21         String tip=req.getParameter("tip");
22         long m=Long.parseLong(sm),n=Long.parseLong(sn);
23         long x=obj.cmmdc(m,n);
24         PrintWriter out=res.getWriter();
25         if(tip.equals("text/html")){
26             String title="Cmmdc Servlet";
27             res.setContentType("text/html");
28             out.println("<HTML><HEAD><TITLE>");
29             out.println(title);
30             out.println("</TITLE></HEAD><BODY>");
31             out.println("<H1>"+title+"</H1>");
32             out.println("<P>Cmmdc is "+x);
33             out.println("</BODY></HTML>");
34         }
35         else{
36             res.setContentType("text/plain");
37             out.println(x);
38         }
39         out.close();
40     }

42     public void doPost(HttpServletRequest req, HttpServletResponse res)
43         throws ServletException, IOException{
44         doGet(req, res);
45     }
46 }

```

iar fişierele de configurare sunt

- *web.xml*

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xmlns="http://java.sun.com/xml/ns/javaee"
4   xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
5   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
6     http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
7   id="WebApp_ID" version="3.0">
8
9   <display-name>CDI Web Application</display-name>
10  <listener>
11    <listener-class>
12      org.jboss.weld.environment.servlet.Listener
13    </listener-class>
14  </listener>
15</web-app>
```

- *beans.xml*

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://java.sun.com/xml/ns/javaee"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5     http://java.sun.com/xml/ns/javaee/beans_1_0.xsd">
6 </beans>
```

**Partea IV**

**TEME DE LABORATOR**





# Anexa J

## Teme de aplicații

### J.1 Probleme propuse

1. Să se realizeze conversia temperaturii exprimată în grade Celsius în grade Fahrenheit și invers. Formula de transformare este

$$t_F = 1.8t_C + 32^\circ F$$

2. Să se realizeze conversia a unei sume de bani între USD, EURO și RON.  
Rata zilnică de schimb preia dinamic sub forma unui fișier `xml`, apelând <http://www.bnr.ro/nbrfxrates.xml>.
3. Să se realizeze conversia unui număr natural, din cifre arabe în cifre romane și invers.

Pentru numere  $x \in (3999, 3999999]$ ,  $x = a * 1000 + b$  se va folosi scrierea sub forma  $(A)B$ , unde  $A, B$  reprezintă conversiile lui  $a$ , respectiv  $b$ .

Exemplu: Conversia numărului 2289463 este (MMCCLXXXIX)CDLXIII.

Pentru numere  $x \in (4000000, 3999999999]$ ,  $x = a * 1000000 + b * 1000 + c$  se va folosi scrierea sub forma  $[A](B)C$ , unde  $A, B, C$  reprezintă conversiile lui  $a, b, c$ .

4. Să se realizeze conversia unui număr dintr-o bază în alta.
5. Un server adună la un număr (inițializat cu 0) o valoare trimisă de un client returnând rezultatul. Există o singură instanță a numărului, același pentru orice client.

Să se programeze serviciul descris mai sus.

6. Să se determine zodia (chinezească) corespunzătoare unei date calendaristice. Obs. Începutul anului nou chinezesc nu coincide cu începutul anului nou calendaristic.

(a) Varianta 1

Se va crea baza de date *AN\_CHINEZESC* alcătuită din tabelul

INCEPUT
AN
LUNA
ZI

(b) Varianta 2

Datele dintr-un fișier text se introduc într-o colecție Java. Rezolvarea cererii unui client se face utilizând facilitățile de programare oferite de interfața `java.util.Collection` din `jdk`.

7. Se consideră baza de date *UNITAȚI-DE-MĂSURĂ* formată din tabelul

CONVERSIE
UM_SI
UM_SIMBOL
DENUMIRE (ROM)
VAL

Exemplu:

1	M	INCH	Țol	0.0254
2	M	FEET (FT)	Picior	0.3048
3	KG	UK_ONCE	Uncie (UK)	0.031103
4	KG	UK_POUND	Livra (UK)	0.373

Să se realizeze o aplicație pentru conversia unităților de măsură extinzând conținutul bazei de date.

8. Se consideră baza de date *NORME-DIDACTICE* formată din tabelele

CADRU-DIDACTIC	MATERIA	CURSURI
COD-CADRU-DIDACTIC	COD-MATERIE	COD-CURS
NUME	DENUMIRE	COD-CADRU-DIDACTIC
		COD-MATERIE

Să se elaboreze programe de întreținere și interogare a bazei de date.

9. Se consideră baza de date *APROVIZIONARE* formată din tabelele

RESURSA	FURNIZOR	CONTRACT
COD-RESURSA	COD-FURNIZOR	COD-CONTRACT
DENUMIRE	NUME	COD-FURNIZOR
		COD-RESURSA
		CANTITATE

Să se elaboreze programe de întreținere și interogare a bazei de date.

10. Se consideră baza de date *DESFACERE* formată din tabelele

PRODUSE	BENEFICIAR	CONTRACT
COD-PRODUS	COD-BENEFICIAR	COD-CONTRACT
DENUMIRE	NUME	COD-BENEFICIAR
		COD-PRODUS
		CANTITATE

Să se elaboreze programe de întreținere și interogare a bazei de date.

11. Validarea codului numeric personal.

Codul Numeric Personal (CNP) constituie numărul de ordine atribuit de Evidența populației unei persoane la naștere, de cetățenie română, care se înscrie în actele și certificatele de stare civilă și se preia în celelalte acte cu caracter oficial.

Structura unui CMP este

```

S AA LL ZZ JJ ZZZ C
| | | | | | | -> cifra de control
| | | | | | -> numar de ordine atribuit persoanei
| | | | | -> codul judetului
| | | -> ziua nasterii
| | -> Luna nasterii
| -> Anul nasterii
|-> Cifra sexului (M/F)
    1/2 nascuti intre 1 ian 1900 si 31 dec 1999
    3/4 nascuti intre 1 ian 1800 si 31 dec 1899
    5/6 nascuti intre 1 ian 2000 si 31 dec 2099
    7/8 rezidenti

```

Verificarea cifrei de control: Se folosește cheia de testare 279146358279. Primele 12 cifre ale CNP se înmulțesc pe rând de la stânga spre dreapta cu cifra corespunzătoare a cheii de testare. Cele 12 produse se adună, iar suma se împarte la 11. Dacă restul împărțirii este mai mic decât 10, atunci acesta va fi cifra de control. Dacă restul împărțirii este 10 atunci cifra de control este 1.

Să se elaboreze un program pentru verificarea CNP.

12. Informațiile unui aeroport civil privind decolările(plecări) și aterizări(sosiri) pe parcursul unei perioade sunt cuprinse în tabelul *activitate* al bazei de date *aeroport*

ACTIVITATE	
ID	
FEL	aterizare/decolare
ZI	
TIMP	ora/minut
COMPANIE	
OBSERVATIE	de la/spre

Să se realizeze o aplicație de furnizare a datelor către clienți (interogarea bazei de date).

13. Se dau  $n$  tipuri de monede  $M_{x_1}, M_{x_2}, \dots, M_{x_n}$ ,  $x_i$  reprezentând valoarea monedei,  $i = 1, 2, \dots, n$ . Să se determine variantele de plată a unei sume  $S$  cu un număr minim de monede din cele disponibile.

Se vor trata cazurile:

- (a) Numărul monedelor de fiecare tip este nelimitat.
- (b) Numărul monedelor disponibile este mărginit, respectiv de numerele  $k_1, k_2, \dots, k_n$  ( $k_1x_1 + \dots + k_nx_n \geq S$ ).

14. Globul pământesc este împărțit în 24 de fuse orare de câte  $15^\circ$  ( $24 * 15 = 360$ ) pornind de la meridianul 0 spre stânga și dreapta.

- Dându-se longitudinile a 2 puncte de pe glob să se calculeze diferența de fus orar.
- Fixând ora în primul punct să se indice ora în al doilea punct.

15. Să se calculeze unghiurile unui triunghi dat prin coordonatele vârfurilor.

16. Fixând coeficienții  $a, b, c, d, e, f$  ale conicei

$$ax^2 + 2bxy + cy^2 + 2dx + 2ey + f = 0$$

să se reducă conica la forma canonică.

17. Să se calculeze suma multiplilor de 3 sau 5 mai mici decât un număr dat  $n$ .

18. Fie  $p \in \mathbb{N}^*$  fixat și  $n \in \mathbb{N}, 0 \leq n \leq 2^p - 1$  cu reprezentarea binară  $n = \overline{a_{p-1}a_{p-2} \dots a_1a_0}_2$ . Să se determine numărul  $m$  care are reprezentarea binară  $m = \overline{a_0a_1 \dots a_{p-2}a_{p-1}}_2$ .

De exemplu, pentru  $p = 6, n = 17 = \overline{010001}_2$  numărul căutat este  $m = \overline{100010}_2 = 34$ .

19. Să se determine numărul de apariții al fiecărui cuvânt dintr-un text dat.

20. Să se calculeze  $B_n$  (Numerele lui Bernoulli) definit prin

$$B_n = \begin{cases} 1 & \text{dacă } n = 0 \\ -\frac{\sum_{k=0}^{n-1} \binom{n-1}{k} B_k}{n} & \text{dacă } n > 0 \end{cases}$$

Calculul se fac în  $\mathbb{Q}$ .

21. **Problema generalilor bizantini.**<sup>1</sup> Sunt  $n$  generali dintre care unul este comandant și ceilalți locotenenți. Generalii pot fi loiali sau trădători. Numărul generalilor trădători este  $m$ . Comandantul (loial sau trădător) lansează o comandă (atac sau retragere). Toți locotenenții loiali urmează ordinele comandantului și ei trebuie să ajungă la aceeași concluzie privind acțiunea de urmat.

Algoritmul: După ce un locotenent primește ordinul comandantului el retransmite ordinul primit tuturor celorlalți locotenenți. Comanda majoritară este cea care va fi urmată.

Ipoteze: Transmisia mesajelor este sigură. Mesajele nu pot fi interceptate și modificate. Un trădător nu poate trimite decât un singur mesaj și este obligat să urmeze algoritmul.

Se arată că

---

<sup>1</sup>Aplicația distribuită nu este de tip client - server.

- Dacă  $n = 3m$  atunci problema n-are soluție.
- Dacă  $n \geq 3m + 1$  atunci problema are soluție.

Să se simuleze problema generalilor bizantini printr-o aplicație distribuită utilizând tehnologia soclurilor.

22. **Problema**  $3x + 1$ . Fie șirul de numere naturale  $(a_n)_{n \in \mathbb{N}}$  definit prin formula de recurență

$$a_{n+1} = \begin{cases} 3a_n + 1 & \text{dacă } a_n \text{ impar} \\ \frac{a_n}{2} & \text{dacă } a_n \text{ par} \end{cases}$$

Pentru orice  $a_0$  există  $n \in \mathbb{N}$  astfel încât  $a_n = 1$ .

Numim pas trecerea de la un termen impar la următorul termen impar ( $a_n$  și  $a_{n+p}$  determină un pas dacă sunt numere impare și  $a_{n+1}, \dots, a_{n+p-1}$  sunt numere pare).

Pentru un  $a_0$  dat să se determine numărul de pași până la atingerea lui 1.

Exemplu. Pentru  $a_0 = 7$  șirul este

$$7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1$$

iar numărul de pași este 5.

23. Se definesc tablourile  $(T_n^d)$  și  $(P_n^d)$  pentru  $n \in \mathbb{N}^*$ ,  $d \in \mathbb{N}$  prin formulele de recurență:

$$\begin{aligned} P_1^d &= 1, \quad d \in \mathbb{N} \\ P_{n+1}^d &= T_n^d, \quad d \in \mathbb{N}, \quad n \in \mathbb{N}^* \\ T_n^d &= \sum_{j=0}^d P_n^j \quad d \in \mathbb{N}, \quad n \in \mathbb{N}^* \end{aligned}$$

Să se calculeze  $T_n^d$ .

Exemplu.

$$P_n^d = \begin{array}{c|ccccc} & d & & & & \\ n & 0 & 1 & 2 & 3 & 4 \\ \hline 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 1 & 2 & 3 & 4 & 5 \\ 3 & 1 & 3 & 6 & 10 & 15 \end{array} \quad T_n^d = \begin{array}{c|ccccc} & d & & & & \\ n & 0 & 1 & 2 & 3 & 4 \\ \hline 1 & 1 & 2 & 3 & 4 & 5 \\ 2 & 1 & 3 & 6 & 10 & 15 \\ 3 & 1 & 4 & 10 & 20 & 35 \end{array}$$

# Bibliografie

- [1] ALBOAIE L., BURAGA S., 2006, *Servicii Web*. Ed. Polirom, Iași.
- [2] ATHANASIU I., COSTINESCU B., DRĂGOI O.A., POPOVICI F.I., 1998, *Limbajul Java. O perspectivă pragmatică*. Ed. Computer Libris Agora, Cluj-Napoca.
- [3] BOIAN F.M., BOIAN R. F., 2004, *Tehnologii fundamentale Java pentru aplicații Web*. Ed. Albastră, Cluj-Napoca.
- [4] BOIAN F.M., 2011, *Servicii Web; Modele, Platforme, Aplicații*. Ed. Albastră, Cluj-Napoca.
- [5] BURAGA S.C., 2001, *Tehnologii Web*. Ed. Matrix Rom, București.
- [6] BURAGA S. (ed), 2007, *Programarea în Web 2.0.*, Ed. Polirom, Iași.
- [7] CARLSON L., 2013, *Programming for PaaS*. O'Reilly, Sebastopol CA.
- [8] COULOURIS G., DOLLIMORE J., KINDBERG T., *Distributed Systems. Concepts and Design*. Addison Wesley, 2005.
- [9] JURCĂ I., 2000, *Programarea rețelelor de calculatoare*. Ed. de Vest, Timișoara.
- [10] LETȚIA S.T., 2002, *Programare avansată în Java*. Ed. Albastră, Cluj-Napoca.
- [11] SCHEIBER E., 2007, *Programare concurentă și paralel distribuită în Java*. Ed. Albastră, Cluj-Napoca.
- [12] TANASĂ Ș., ANDREI Ș., OLARU C., 2011, *Java de la 0 la extert*. Ed. Polirom, Iași.
- [13] TANASĂ Ș., OLARU C., 2005, *Dezvoltarea aplicațiilor Web folosind Java*. Ed. Polirom, Iași.