ERNEST SCHEIBER

PROGRAMARE DISTRIBUITĂ ÎN JAVA

Volumul I

Braşov

Prefaţă

Scopul acestui curs este prezentarea tehnologiilor de programare de pe platforma Java:

- *Java Platform Standard Edition Development Kit* - (JDK);
- *Eclipse Enterprise for Java* (EE4J) redenumirea platformei *Java Enterprise Edition* - (JEE)

care permit realizarea aplicaţiilor client - server:

- socluri Java - JDK;
- apelarea metodelor de la distanţă
  - *Remote Method Invocation (RMI)* - JDK;
  - *Common Object Request Brocker Arhitecture (CORBA)* - JDK;
- mesageria Java (*Java Message Service (JMS)* - JEE ;
- servlet - JEE;
- *Java Server Pages (JSP)* - JEE;
- *Enterprise Java Bean (EJB)* - JEE.

Accentul cade pe detaliile tehnice de realizare a comunicaţiilor şi pe arhitectura programelor / aplicaţiilor care le utilizează. Trebuie semnalat faptul că exemplele date nu încorporează aspecte indispensabile unei aplicaţii informatice la standardele zilei:

- securitate, autentificare şi autorizare;
- interfaţă grafică pentru componenta client;
- utilizarea bazelor de date relaţionale / orientate obiect / NoSQL.

Metodele şi 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   }

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  };
```

O variantă mai elaborată este

```
1  import java.util.function.BiFunction;
2  . . .
3  static long cmmdc(long a,long b){
4      BiFunction<Long,Long,Long> f=(m,n)->{
5        long r,c;
6        do{
7          c=n;
8          r=m % n;
9          m=n;
10         n=r;
11       }
12       while(r!=0);
13       return Long.valueOf(c);
14     };
15     return f.apply(a,b).longValue();
16   }
```

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(...)`.

O variantă extinsă a metodei *cmmdc* imperativă este

```
1  public long[] cmmdc(long m,long n){
2       long r,c,u0=0,u1=0,v0=0,v1=0,q,u=0,v=0;
3       int k=0;
4       boolean sw=true;
5       if(n>m){
6         sw=false;
7         c=m;m=n;n=c; //  m <-> n
8       }
9       do{
10        k++;
11        c=n;
12        r=m % n;
13        q=m / n;
14        if(k==1){
15          if(r==0){
16            u=1;
17            v=1-q;
18          }
19          else{
20            u0=1;
21            v0=-q;
22          }
23        }
24        if(k==2){
25          if(r==0){
26            u=u0;
27            v=v0;
28          }
29          else{
30            u1=-u0*q;
31            v1=1-v0*q;
32          }
33        }
34        if(k>2){
35          if(r==0){
36            u=u1;
37            v=v1;
38          }
39          else{
40            u=u0-u1*q;
41            v=v0-v1*q;
42            u0=u1;
43            v0=v1;
44            u1=u;
45            v1=v;
```

```
46            }
47          }
48          m=n;
49          n=r;
50        }while(r!=0);
51        if(!sw){
52          q=u;u=v;v=q;  // u <-> v
53        }
54        return new long[]{c,u,v};
55    }
```

Dacă $[d, u, v] = \text{cmmdc}(a, b)$ atunci $d = (a, b)$ şi $d = ua + vb$.

Pentru aplicaţiile 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 învăţare propus este

1. Se instalează toate resursele necesare (Se exemplifică la laborator).
2. Se execută aplicaţia / aplicaţiile din curs (Se exemplifică la laborator).
3. Pentru fiecare tehnologie, pe suportul oferit de curs, se programează o altă aplicaţie.

   Propunem următoarele teme:

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

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

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

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

Sursele programelor din curs sunt disponibile prin `git`:
`https://github.com/e-scheiber/DistributedProgramming1.git`

Probleme:

- http2
- Docker, Vagrant, Kubernetes
- microservicii
- Autentificare şi autorizare prin LDAP, Kerberos, OAuth2

Produsele informatice utilizate

Pe durata existenţei, produsele informatice evoluează prin versiunile pe care producătorii ni le pun la dispoziţie. Nu de puţine 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-1.10.1-bin.tar.gz
2	apache-commons-fileupload	commons-fileupload-1.3.3-bin.tar.gz
3	apache-ftpserver	ftpserver-1.1.1.tar.gz
4	apache-maven	apache-maven-3.5.2-bin.tar.gz
5	apache-shiro	apache-shiro-1.3.2
6	apache-tomcat	apache-tomcat-9.0.2.tar.gz
7	apacheds	apacheds-2.0.0-M24.tar.gz
8	glassfish	glassfish-5.0.zip, payara-4.1.2.174.zip
9	gradle	gradle-4.2.1-all.zip
10	google appengine	appengine-java-sdk-1.9.59.zip
11	db-derby	db-derby-10.14.1.0-bin.tar.gz
12	httpcomponents-client	httpcomponents-client-4.5.4-bin.tar.gz
13	httpcomponents-asyncclient	httpcomponents-asyncclient-4.1.3-bin.tar.gz
14	Java (jdk)	jdk-9.0.1-windows-x64_bin.exe jdk-8u151-windows-{i586;x64}.exe
15	junit	junit4.12.zip
16	mysql	mysql-5.6.37-win{32;x64}.zip
17	mysql-connector	mysql-connector-java-5.1.43.tar.gz
18	openmq	openmq5_1_1-binary-windows.zip

Cuprins

Capitolul 1

Introducere

Occam's razor (William Ockam, cca 1285-1349)
entia non sunt multiplicanda praeter necessitatem
Lama lui Occam
entitile nu trebuie să fie multiplicate dincolo de necesar

Reţelele locale, internetul, răspândirea pe o arie geografică a resurselor şi a locaţiilor în care se petrec acţiuni ce ţin de o activitate bine definită sau sunt urmărite, gestionate din alte locuri au drept consecinţă existenţa aplicaţiilor distribuite. Termenul distribuit se referă tocmai la faptul că componente ale aplicaţiei se află pe calculatoare diferite dar între care au loc schimburi de date. Dacă părţile unei aplicaţii sau resursele utilizate se găsesc pe calculatoare distincte atunci aplicaţia se numeşte distribuită.

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

Astfel se vorbeşte de programare distribuită ca mijloc de realizare a aplicaţiilor distribuite. Pe lângă *algoritm, structuri de date, limbaj de programare,* la realizarea unei aplicaţii distribuite intervin *comunicaţiile:* 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 comunicaţiilor, transmisia de date presupune transformarea acestora din formatul tipizat în şir de octeţi şi invers - serializarea datelor. În acest

scop tehnologiile de programare utilizează modalităţi diverse, de la atribuirea acestei sarcini către programator până la asigurarea unui cadru fix de realizare a serializării.

Punem în evidenţă două modele de aplicaţii distribuite:

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

  O aplicaţie client – server se compune din:

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

  Serverul şi clientul (clienţii) rulează, de obicei, pe calculatoare distincte. Un server trebuie să satisfacă cererile mai multor clienţi.

  Aplicaţia server se va găsi (desfăşura) într-o aplicaţie de tip server de aplicaţie, server Web. Aceste aplicaţii au de multe ori un caracter integrator în sensul că înglobează resursele necesare diverselor tehnologii de programare distribuită. Tendinţa curentă constă din găzduirea aplicaţiilor server în *nor* (*cloud*).

  Durata de viaţă a unei aplicaţii client – server este dată de durata funcţionării 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 şi până la deconectare poartă numele de sesiune. În această perioadă, clientul poate invoca mai multe servicii ale serverului. Un client poate iniţia mai multe sesiuni.

  Un client trebuie să-şi regăsească datele în cadrul unei sesiuni, între sesiuni, pe toată durata de viaţă a aplicaţiei. Astfel se pune problema reţinerii / persistenţei datelor pentru fiecare client în parte.

  Amintim următoarele tehnologii Java pentru realizarea aplicaţiilor client-server:

  - Tehnologii destinate reţelelor locale, nebazate pe protocolul `http`.
    - *Socluri*
    - *Remote Method Invocation* (RMI)
    - *Common Object Request Brocker Arhitecture* (CORBA)

    - *Java Message Service* (JMS)

    Comunicaţiile utilizează porturi care nu trebuie să fie închise de eventuale aplicaţii de tip *firewall.*

  - Tehnologii cu comunicaţii prin Internet, bazate pe protocolul `http`.
    - *Servlet* şi *Java Server Pages* (JSP)
    - *Java Web Start*
    - *WebSocket*

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

Există diferenţe mari între o aplicaţie care rulează pe un calculator şi o aplicaţie distribuită[1]:

- latenţa (*latency*) - există mai multe definiţii:
  - diferenţa în timp între o operaţie executată pe un calculator la distanţă de execuţia ei pe calculatorul local.
  - diferenţa în timp între momentul recepţionării răspunsului la o cerere şi momentul lansării ei.
  - diferenţa în timp între momentul recepţionării unei cereri şi momentul transmiterii răspunsului.

  Recepţia rezultatului unei operaţii executate pe un calculator la distanţă se poate programa

  - sincron – de obicei recepţia blochează firul de execuţie al apelului până la sosirea rezultatului;
  - asincron – concept care are mai multe materializări:
    - recepţia se obţine într-un obiect dedicat care se execută în afara firului de execuţie al apelului.
      (JMS, servlet)
    - rezultatul solicitării este un obiect de tip `Future`$< T >$ şi a cărui procesare se poate face după instanţierea obiectului T.
      (socluri cu canale de comunicaţii, client servlet)

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

  - se utilizează metode neblocante.

  O preocupare continuă este dezvoltarea de tehnologii hard şi soft pentru micşorarea latenţei.

- accesul la memorie (*memory access*). Instrumente de programare - cadre de lucru (*framework*) care mijlocesc realizarea aplicaţiilor asigură accesul la resursele aflate în memoria calculatoarelor (*read, write, send, receive*).

- interoperabilitate. Aplicaţiile distribuite pot rula pe platforme diferite (de exemplu Linux şi Windows sau Java şi .NET) iar componentele ei pot fi realizate în limbaje de programare diferite.

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

- prăbuşirea parţială (*partial failure*) constă în încetarea funcţionării unei părţi a aplicaţiei 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 prăbuşirea parţială, disponibilitatea aplicaţiei şi consistenţa datelor.

Precizăm sensul unor noţiuni utilizate în lucrare:

- *protocol* - pachet de reguli, şablon utilizat în comunicaţii, în accesarea unor resurse. Exemple de protocoale standard utilizate sunt:
  - *http - HyperText Transfer Protocol* este principalul protocol utilizat în comunicaţiile prin Internet;
  - *https* protocol http securizat;
  - *file* - protocol pentru specificarea fişierelor aflate pe calculatorul local;
  - *ftp - File Transfer Protocol* - protocol pentru transferul fişierelor între două calculatoare;
  - *smtp - Simple Message Transfer Protocol* utilizat de poşta elecronică.

- *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 aplicaţia responsabilă de prelucrarea solicitării. Implicit, anumitor protocoale le sunt asociate porturi. Dintre acestea amintim:

Port	Utilizat de
80	http
443	https
25	smtp

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

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

Pe de altă parte comunicaţia dintre server şi client ale unei aplicaţii se bazează pe protocoale proprii aplicaţiei. Un asemenea protocol poate fi

- implicit

  Protocolul nu este materializat printr-un cod. În specificaţiile aplicaţiei se fixează serviciile / metodele serverului şi în particular semnăturile acestora.

- explicit

  Protocolul este dat de o interfaţă sau de o clasă.

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

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

În urma standardizării, pentru o tehnologie de prelucrare se obţine o interfaţă de programare (*Application Programming Interface* - (API)), uzual publică şi care poate fi implementată de diverşi producători.

Prin standardizare se urmăreşte independenţa unei aplicaţii de implementarea interfeţelor de programare. Reamintim faptul că platforma de programare Java este independentă de sistemul de operare, orice program Java se prelucrează şi rulează fără nici o modificare pe Linux, Windows, Mac OS X.

Întrebări recapitulative

1. Explicaţi noţiunea de latenţă.
2. Explicaţi termenul de protocol şi daţi exemple de protocoale de comunicaţie.
3. Explicaţi noţiunea de prăbuşire parţială.
4. Ce desemnează denumirea *Java Community Process*?
5. Precizaţi sintaxa unui URI.
6. Care este rolul standardizării?

Java 9

Java 9 se caracterizează prin următoarele *categorii* de funcţionalităţi:

- *Funcţionalitate implicită*: Codul funcţionează fără modificări faţă de JDK 8.
- *Funcţionalitate nouă.*
- *Funcţionalitate specializată*: Necesită modificarea codului Java pentru a putea rula în JDK 9 impusă de noile tehnologii implementate.
- *Funcţionalitate în dezvoltare*: Poate suferi modificări în versiunile Java viitoare.
- *Functionalitate eliminată*: Facilitate prezentă în JDK 8 dar nu mai apare în JDK 9.

Java 9 introduce `jdk-modular` (proiectul *jigsaw* - fierăstrău mecanic - *Jdk Enhancement Proposal* - JEP). Pachetele din `jdk` sunt grupate în module şi un modul este încărcat doar la solicitare. Criteriile de grupare a pachetelor în module au fost funcţionalitatea oferită şi frecvenţa de utilizare.

Lista modulelor se obţine cu comanda

`java --list-modules`

Lista pachetelor exportate de un modul se obţine cu comanda

`java --list-modules` *numeModul*

Dezvoltarea unei aplicaţii se poate face în două moduri:

- Modul obişnuit de dezvoltare cu regăsirea resurselor, adică a pachetelor Java, prin variabila de sistem `classpath`.

Clasele accesibile din `jdk` sunt doar cele care nu sunt declarate *Deprecated*.

Nu toate pachetele Java sunt *visibile* la compilare sau executare, chiar dacă s-au declarat prin `import`, de exemplu `java.corba`. În acest caz un asemenea pachet trebuie explicitat în comenzile `javac, java` prin atributul

`--add-module` *nume_pachet, ...*

- În manieră modulară.

Indiferent de modul de dezvoltare, codurile claselor Java sunt identice, cu precizarea că în maniera modulară orice clasă aparţine unui pachet.

Dezvoltarea modulară

O aplicaţie este alcătuită din unul sau mai multe module. Sursele unui modul sunt găzduite de un catalog, în care se găsesc:

1. Fişierul `module-info.java` cu descrierea modulului.

   Modulul `java.base` se încarcă implicit, el nu trebuie declarat în fişierul `module.info.java`.

2. Sursele claselor[2], incluse într-un catalog sau într-o structură de cataloage.

Definirea unui modul

1. Cea mai simplă formă a fişierului `module-info.java` este

```
1 module numeModul{}
```

   În acest caz modulul definit foloseşte doar pachetele modulului implicit `java.base` şi oferă o funcţionalitate imediată care nu va fi inclusă în alte module.

2. Utilizarea altor module şi posibilitatea refolosirii unor pachete se indică prin

[2]Orice clasă Java aparţine unui pachet

```
module numeModulX{
    requests numeModulA;
    requests transitive numeModulB;
    . . .
    exports numePachet;
    . . .
}
```

Declaraţia `request transitive` asigură accesul la modulele şi pachetele modulului *numeModulB* către orice modul care utilizează *numeModulX*.

3. Modificatorul `open`

```
open module numeModul{
  . . .
}
```

permite accessul la resursele modulului care se defineşte de către modulele utilizate.

Prin modularizare se întăreşte încapsularea: importul unei clase exterioare pachetului nu asigură accesul la acea clasă în lipsa instrucţiunii `requires` cu pachetul corespunzător.

Această proprietate se numeşte *dubla încapsulare.*

Anumite resurse se pot indica prin `classpath.`

Compilare în linia de comandă

`javac -d` *catalogDestinaţie* `*.java`

`javac --module-path` *caleCătreModule* `-d` *catalogDestinaţie* `*.java`

Pentru catalogul destinaţie se va folosi şi denumirea catalogul modulului.

Lansare în execuţie în linia de comandă

`java ---module-path` *caleCătreModule* `-m` *numeModul/clasăCuMetodaMain*

Accesul la unele pachete ale unui modul care nu sunt *visibile* se indică prin

`--add-opens=`*modul/packet*`=ALL-UNNAMED`

Manipularea resurselor exterioare ne-modulare

Lista dependinţelor unei arhive `jar` se obţine cu comanda
`jdeps -s` *cale/numeArhiva.jar*
`jdeps` *cale/catalog*

Dacă se folosesc resurse dintr-o arhivă, de exemplu *xyz-*.jar,* atunci se modifică denumirea fişierului în *xyz.jar* şi în `module-info.java` va apărea instrucţiunea `requires` *xyz;* Modulul *xyz* se numeşte modul automat.

În linie de comandă, pentru compilare şi execuţie referinţa la fişierul *xyz.jar* se indică cu opţiunea `--module-path`.

Probleme

- `--add-module, --add-opens` în `apache-ant, apache-maven.`

Partea I

TEHNOLOGII PENTRU REŢELE LOCALE

Capitolul 2

Programare cu socluri Java

Comunicaţia bazată pe socluri Java constituie modalitatea de nivelul inferior pentru realizarea aplicaţiilor distribuite. Din punct de vedere fizic un soclu este o interfaţă de conectare la procesor. Soclurile se află pe placa de bază. Noţiunea 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 reţea 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 comunicaţii independente de platforma de calcul. Pentru a alege clasa Java care să fie utilizată trebuie cunoscută diferenţa dintre TCP şi UDP.

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

UDP Utilizarea protocolului UDP presupune trimiterea unor pachete de date – numite datagrame – de la o aplicaţie la alta fără să se asigure faptul că datagramele ajung la destinaţie şi nici ordinea lor de sosire. Acest tip de comunicaţie seamănă cu trimiterea scrisorilor prin poştă.

2.2 Soclu TCP

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

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

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

Clientul cunoaşte numele calculatorului pe care rulează serverul cât şi portul la care acesta este conectat. Pentru stabilirea conexiunii, clientul încearcă un rendez-vous cu serverul de pe maşina serverului şi 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 aşa fel încât ascultarea cererilor la soclul iniţial să poată continua în timp ce sunt satisfăcute cererile clientului conectat. Din partea clientului, după acceptarea conexiunii soclul este creat şi este utilizat pentru comunicaţia 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 ataşat soclului, pentru citirea (preluarea) informaţiilor de la soclu.

- `public OutputStream getOutputStream() throws IOException`

  Returnează un flux de ieşire ataşat soclului, pentru scrierea (transmiterea) informaţiilor la soclu.

- `public synchronized void close() throws IOException`

  închide soclul de referinţă.

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 şirului (tamponului) de aşteptare 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 şirului (tamponului) de aşteptare.

Metode

- `public Socket accept() throws IOException`

  Metoda blochează procesul (firul de execuţie) apelant până la sosirea unei cereri de conectare şi crează un soclu client prin care se va desfăşura comunicarea cu solicitantul acceptat.

- `public synchronized void close() throws IOException`

  închide soclul de referinţă.

2.2.1 Aplicaţie client – server cu socluri

Serverul trebuie să satisfacă simultan solicitările mai multor clienţi. Fiecare client apelează programul server la acelaşi port şi în consecinţă cererile de conectare sunt recepţionate de acelaşi ServerSocket. Serverul recepţionează apelurile secvenţial. La un apel, se crează de partea severului un soclu prin care se va face schimbul de date cu clientul. Cererile clienţilor pot fi satisface concurent/paralel, utilizând fire de execuţie ce implementează serviciul oferit sau secvenţial - 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.*

Protocolul de comunicaţie este implicit: serverul trimite două numere întregi şi recepţionează rezultatul. Tipul numerelor este `long`.

Programul client *CmmdcClient* se conectează la server, transmite serverului cele două numere naturale şi recepţionează rezultatul pe care apoi îl afişează.

În esenţă orice program client trebuie să execute:

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

Sursele sunt:

- Clasa *cmmdc.socket.client.CmmdcClient*

```
1  package cmmdc.socket.client;
2  import java.io.DataInputStream;
3  import java.io.DataOutputStream;
4  import java.io.IOException;
5  import java.net.Socket;
6  import java.util.Scanner;

8  public class CmmdcClient {
9    public static void main(String[] args) throws IOException {
10     String host="localhost";
11     int port=7999;
12     if (args.length>0)
13        host=args[0];
14     if (args.length>1)
15        port=Integer.parseInt(args[1]);
16     Scanner scanner=new Scanner(System.in);
17     System.out.println("m=");
18     long m=scanner.nextLong();
19     System.out.println("n=");
20     long n=scanner.nextLong();
21     try(Socket cmmdcSocket = new Socket(host, port);
22       DataInputStream in=
23         new DataInputStream(cmmdcSocket.getInputStream());
24       DataOutputStream out=
25         new DataOutputStream(cmmdcSocket.getOutputStream())){
26         out.writeLong(m);
27         out.writeLong(n);
```

```
28            long r=in.readLong();
29            System.out.println("Required result : "+r);
30        }
31        catch(Exception e){
32          System.err.println("Client comunication error : "+e.getMessage());
33        }
34    }
35 }
```

- `module-info.java`

```
1 module client{}
```

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

Partea server este alcătuită din mai multe clase:

- Clasa *cmmdc.socket.server.MyMServer,* independentă de un serviciu anume, preia apelurile clienţilor şi lansează satisfacerea cererii.

```
1  package cmmdc.socket.server;
2  import java.net.ServerSocket;
3  import java.util.concurrent.ExecutorService;
4  import java.util.concurrent.Executors;
5  import java.io.IOException;

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

11   public static void main(String[] args){
12     int port=7999;
13     boolean listening=true;
14     AppThread appThread=new AppThread();
15     try(ServerSocket serverSocket=new ServerSocket(port)) {
16       System.out.println("The server is listening on port 7999");
17       while(listening){
18         exec.execute(appThread.service.apply(serverSocket.accept()));
19       }
20     }catch (IOException e){
21       e.printStackTrace();
22     }
23   }
24 }
```

În ciclul `while,` la recepţia unei solicitări de conexiune se crează şi se lansează un fir de execuţie a cărei metodă *run* conţine acţiunile ce răspund solicitării.

- Clasa *cmmdc.socket.server.AppThread* - fir de execuţie responsabil de preluarea datelor şi de transmitere a rezultatului.

```
1  package cmmdc.socket.server;
2  import java.net.Socket;
3  import java.io.DataInputStream;
4  import java.io.DataOutputStream;
5  import java.io.IOException;
6  import java.util.function.Function;

8  public class AppThread{
9    // Firul de executie lansat de server
10   Function<Socket,Thread> service=socket->{
11     return new Thread(()->{
12       try(DataOutputStream out =
13             new DataOutputStream(socket.getOutputStream());
14         DataInputStream in =
15             new DataInputStream(socket.getInputStream())){
16         long m=0,n=0,r;
17         m=in.readLong();
18         n=in.readLong();
19         r=App.cmmdc(m,n);
20         out.writeLong(r);
21         socket.close();
22       }
23       catch(IOException e){
24          e.printStackTrace();
25       }
26     });
27   };
28 }
```

- Clasa *cmmdc.socket.server.App* corespunzătoare calculului celui mai mare divizor comul a două numere naturale.

```
1  package cmmdc.socket.server;
2  import java.util.function.BiFunction;

4  public class App{
5    static public long cmmdc(long a,long b){
6      BiFunction<Long,Long,Long> f=(m,n)->{
7        long r,c;
8        do{
9          c=n;
10         r=m % n;
11         m=n;
12         n=r;
13       }
14       while(r!=0);
15       return Long.valueOf(c);
16     };
17     return f.apply(a,b).longValue();
18   }
19 }
```

- `module-info.java`

```
1 module server{}
```

Observaţie 2.2.1

În aceasta variantă activitatea pentru rezolvarea solicitării clientului este externalizată într-un fir de execuţie (clasa *AppThread*). Pentru activităţi de durată foarte scurtă, activitatea se poate programa chiar în clasa serverului (*MyMServer*). Astfel metoda `main` poate fi

```
public static void main(String[] args){
  int port=7999;
  boolean listening=true;
  try(ServerSocket serverSocket=new ServerSocket(port)) {
    System.out.println("The server is listening on port 7999");
    while(listening){
      Socket socket=serverSocket.accept();
      try(DataOutputStream out = new DataOutputStream(socket.getOutputStream());
        DataInputStream in = new DataInputStream(socket.getInputStream())){
        long m=in.readLong();
        long n=in.readLong();
        long r=App.cmmdc(m,n);
        out.writeLong(r);
        socket.close();
      }
      catch(IOException e){
         e.printStackTrace();
      }
    }
  }catch (IOException e){
    e.printStackTrace();
  }
}
```

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

Comenzile de operare pentru aplicaţia server sunt

```
javac -d mods src/*.java src/server/*.java

java --module-path mods -m server/cmmdc.socket.server.MyMServer
```

iar pentru aplicaţia client sunt

```
javac -d mods src/*.java src/client/*.java

java --module-path mods -m client/cmmdc.socket.client.CmmdcClient
```

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

Aplicaţia distribuită se va compune din:

- Aplicaţia server alcătuită din:
  - Interfaţă

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

  - Implementarea interfeţei

```
1  package cmmdc.socket.server.impl;

3  import cmmdc.socket.server.i.IMyMServer;
4  import java.net.ServerSocket;
5  import java.io.IOException;
6  import java.util.concurrent.ExecutorService;
7  import java.util.concurrent.Executors;
8  import java.io.IOException;

10 public class MyMServer implements IMyMServer{

12   public ServerSocket getServerSocket(int port){
13     ServerSocket serverSocket = null;
14     try{
15       serverSocket = new ServerSocket(port);
16     }
17     catch (IOException e) {
18       System.err.println("Could not listen on port: "+port);
19       System.err.println(e.getMessage());
20       System.exit(1);
21     }
22     System.out.println("ServerSocket is ready ...");
23     return serverSocket;
24   }

26   public void myAction(ServerSocket serverSocket){
27     int NTHREADS=8192;
28     ExecutorService exec=Executors.newFixedThreadPool(NTHREADS);
29     AppThread appThread=new AppThread();
30     while(true){
31       try{
32         exec.execute(appThread.service.apply(serverSocket.accept()));
33       }
34       catch(IOException e){
35         e.printStackTrace();
36       }
37     }
38   }
39 }
```

Clasele *AppThread, App* sunt cele utilizate anterior.

– Clasă de lansare a serverului

```
1  package cmmdc.socket.server;
2  import java.net.ServerSocket;
3  import cmmdc.socket.server.impl.MyMServer;
4  import cmmdc.socket.server.i.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 }
```

- Aplicaţia 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 fişier text, cerere care va fi satisfăcută de server. Clientul cunoaşte lista fişierelor pe care le poate trimite serverul.*

Structura aplicaţiei 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;

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 comunication error : "+e.getMessage());
31      }
32    }
33  }
```

module-info

```
1  module streamingtext{}
```

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.OutputStream;
9  import java.io.IOException;

11 public class AppThread extends Thread{
12   Socket socket=null;

14   public AppThread(Socket socket){
15     this.socket=socket;
16   }

18   public void run(){
19     String path="../../resources/text/";
20     try(DataInputStream in = new DataInputStream(socket.getInputStream());
21       DataOutputStream out=new DataOutputStream(socket.getOutputStream())){
22       int noFile=in.readInt();
23       System.out.println(noFile);

25       String fileName="";
26       if(noFile==1)
27         fileName="capitol.txt";
28       else
29         fileName="junit.tex";
30       File inputFile=new File(path+fileName);
31       FileReader fr=new FileReader(inputFile);
```

```
32        BufferedReader br= new BufferedReader(fr);

34        String s;
35        while((s=br.readLine())!=null) out.writeUTF(s);
36        out.writeUTF("endOFfile");
37        out.flush();
38        br.close();
39        fr.close();
40        System.out.println("Activity Finished !");
41        socket.close();
42      }
43      catch(IOException e){
44         System.err.println("Server comunication error : "+e.getMessage());
45      }
46    }
47  }
```

module-info

```
1 module streamingtext{}
```

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

Pe aceaşi 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 comunication error : "+e.getMessage());
33        }
34     }
35  }
```

Clasa *ShowImage* afişează imaginea pe monitor.

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

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

11    MyCanvas(Image image){
12       this.image=image;
13    }

15    public void paint(Graphics g){
16       g.drawImage(image,0,0,this);
17    }
18 }

20 public class ShowImage{
21    MyCanvas mc=null;

23    ShowImage(Image image){
24       mc=new MyCanvas(image);
25    }

27    public void show(){
28    // Interfata swing
29       JFrame jframe = new JFrame("The received image");
30       jframe.addNotify();
31       jframe.getContentPane().setLayout(new BorderLayout());
32       jframe.getContentPane().add(mc, BorderLayout.CENTER);
33       jframe.setDefaultCloseOperation(jframe.EXIT_ON_CLOSE);
34       jframe.setSize(600,600);
35       jframe.setVisible(true);
36    }
37 }
```

module-info

```
1  module streamingimage{
2     // Swing
3     requires javafx.swing;
4  }
```

Clasa *AppThread.java*

```
1  package streamingimage;
2  import java.net.Socket;
3  import java.io.OutputStream;
4  import java.io.DataInputStream;
5  import java.io.IOException;
6  import java.nio.file.Paths;
7  import java.nio.file.Path;
8  import java.nio.file.Files;


11 public class AppThread extends Thread{
12   Socket socket=null;

14   public AppThread(Socket socket){
15     this.socket=socket;
16   }

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

25       String fileName="";
26       if(noFile==1)
27         fileName="xml-pic.jpg";
28       else
29         fileName="brasov.jpg";
30       Path cale=Paths.get(path+fileName);
31       Files.copy(cale,out);
32       socket.close();
33     }
34     catch(Exception e){
35        System.err.println("Server comunication error : "+e.getMessage());
36     }
37   }
38 }
```

module-info

```
1  module streamingimage{
2  }
```

2.3 Datagrame

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

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

O aplicaţie trimite şi recepţionează pachete `DatagramPacket` prin intermediul unui `DatagramSocket.` Un pachet `DatagramPacket` poate fi trimis la mai mulţi destinatari prin intermediul unui `MulticastSocket.`

Reamintim că o datagramă este un mesaj trimis prin reţea 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 conţine corpul mesajului şi adresa destinaţiei. Apoi acest obiect `DatagramPacket` poate fi pus în reţea în vederea trimiterii sale. Primirea unui pachet UDP necesită crearea unui obiect `DatagramPacket` şi apoi acceptarea unui pachet UDP din reţea. După primire, se poate extrage din obiectul `DatagramPacket` adresa sursă şi conţinutul mesajului.

Constructori

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

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

  Acest contructor este folosit pentru primirea pachetelor. Un pachet se memorează în tamponul *buffer* având *lung* octeţi. Dacă lungimea pachetului depăşeşte această lungime, atunci pachetul este trunchiat iar octeţii 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 conţinut în tamponul *buffer* având *lung* octeţi. Pachetul va fi trimis către adresa şi 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 acelaşi port.

Metode

- `InetAddress getAddress()`

  returnează adresa IP a expeditorului.

- `int getPort()`

  returnează portul expeditorului.

- `byte[] getData()`

  returnează conţinutul 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 conţinutul pachetului.

- `void setLength(int` *lung*`)`

  fixează lungimea pachetului.

Clasa `java.net.DatagramSocket`

Această clasă se foloseşte atât pentru trimiterea, cât şi pentru primirea obiectelor `DatagramPacket`. Un obiect `DatagramSocket` ascultă la un port cuprins între 1 şi 65535 (porturile cuprinse între 1 şi 1023 sunt rezervate pentru aplicaţiile sistem). Deoarece UDP nu este orientat pe conexiune, se va crea un singur obiect `DatagramSocket` pentru trimiterea pachetelor către diferite destinaţii şi 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` conţine metode pentru trimiterea şi primirea de obiecte `DatagramPacket`, închiderea soclului, determinarea informaţiilor adresei locale şi setarea timpului de primire.

- `void send(DatagramPacket` *pachet*`) throws IOException`

  Trimite *pachet*ul prin reţea. Dacă se trimit pachete la o destinaţie necunoscută sau care nu ascultă, în cele din urmă se generează o excepţie `IOException`.

- `void receive(DatagramPacket` *pachet*`) throws IOException`

  Metoda primeşte un singur pachet UDP în obiectul *pachet* specificat. Apoi, pachetul poate fi inspectat pentru determinarea adresei IP sursă, portul sursă şi lungimea mesajului. Execuţia metodei este blocată până când se primeşte cu succes un pachet sau se scurge timpul de aşteptare.

- `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` *timpDeAşteptere*`) throws SocketException`

  Metoda fixează timpul de aşteptare (în milisecunde) a soclului. Metoda `receive()` se va bloca pentru timpul de aşteptare specificat pentru primirea unui pachet UDP, după care va arunca o excepţie `InterruptedException`. Dacă valoarea parametrului este 0, atunci soclul este blocat.

- `int getSoTimeout() throws SocketException`

  Returnează timpul de aşteptare.

- `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 getInetAddress()`

  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 reţea indicând adresa IP corespunzătoare maşinii destinaţie. Clasa `InetAddress` furnizează acces la adresele IP.

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

- `InetAddress getLocalHost() throws UnknownHostException`

  Returnează un obiect `InetAddress` corespunzător maşinii locale.

- `InetAddress getByName(String` *host*`) throws UnknownHostException`

  Returnează un obiect `InetAddress` corespunzător maşinii *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 şir de obiecte `InetAddress` corespunzător fiecărei adrese IP a maşinii *host*.

Metodele clasei `InetAddress`

- `byte [ ] getAddress()`

  Returnează şirul de octeţi corespunzător obiectului *InetAddress* de referinţă.

- `String getHostName()`

  Returnează numele maşinii gazdă.

- `String getHostAddress()`

  Returnează adresa IP a maşinii gazdă.

- `boolean isMulticastAddress()`

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

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

Exemplul 2.3.1

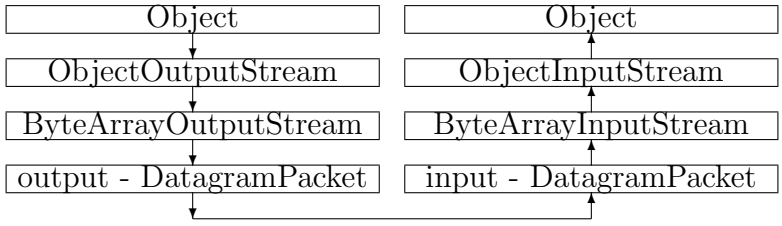
```
1  import java.net.InetAddress;
2  import java.net.UnknownHostException;
3
4  public class AdreseIP{
5    public static void main(String arg[]){
6      InetAddress adresa=null;
7      try{
8        adresa=InetAddress.getLocalHost();
9        System.out.println("Calculatorul gazda are:");
10       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 Aplicaţii client – server cu datagrame.

Un pachet de tip `DatagramPacket` este alcătuit dintr-un şir de octeţi. Este datoria programatorului să transforme datele (mesajul) în şiruri de octeţi la expediere şi la recepţie.

Transformarea unui obiect serializabil într-un şir de octeţi şi invers se poate realiza cu schema



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

În principiu expedierea şi transformarea obiectului *obj* într-un şir de octeţi se poate realiza cu secvenţa 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* şi *port* sunt adresa şi portul destinatarului. Dacă *packet* este un obiect `DatagramPacket` recepţionat atunci metodele `getAddress()` şi `getPort()` furnizează adresa şi portul expeditorului

Recepţionarea şi 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 şir de octeţi cu metoda `byte[] String.getBytes()` şi invers, el se obţine prin `new String(bin)` sau `new String(packet.getData())`.

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

În vederea transportului definim clasa

```
1  package cmmdc.datagram.p;
2  import java.io.Serializable;

4  public class Protocol implements Serializable{
5    static final long serialVersionUID = 1L;
6    long x,y;
7    Protocol(long x,long y){
8      this.x=x;
9      this.y=y;
10   }
11 }
```

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

Se va utiliza arhitectura de aplicaţie dezvoltată în finalul secţiunii anterioare. Aplicaţia se va compune din:

- Aplicaţia server alcătuită din:
  - Interfaţă

```
ackage cmmdc.datagram.i;
import java.net.DatagramSocket;
public interface IMyMServer{
  public DatagramSocket getDatagramSocket(int port);
  public void myAction(DatagramSocket datagramSocket);
}
```

– Implementarea interfeţei

```
package cmmdc.datagram.s.impl;
import cmmdc.datagram.p.Protocol;
import cmmdc.datagram.i.IMyMServer;
import java.net.DatagramSocket;
import java.net.DatagramPacket;
import java.net.InetAddress;
import java.io.IOException;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.function.Consumer;

public class MyMServer implements IMyMServer{
  static Consumer<DatagramSocket> service=datagramSocket->{
    DatagramPacket packet=null;
    Protocol p=null;
    try{
      byte[] bin=new byte[4048];
      packet=new DatagramPacket(bin,bin.length);
      datagramSocket.receive(packet);
      ByteArrayInputStream bais=new ByteArrayInputStream(bin);
      ObjectInputStream in=new ObjectInputStream(bais);
      p=(Protocol)in.readObject();
      p.x=App.cmmdc(p.x,p.y);
      p.y=0;
      ByteArrayOutputStream baos=new ByteArrayOutputStream(256);
      ObjectOutputStream out=new ObjectOutputStream(baos);
      out.writeObject(p);
      byte[] bout=baos.toByteArray();
      InetAddress address=packet.getAddress();
      int port=packet.getPort();
      packet=new DatagramPacket(bout,bout.length,address,port);
      datagramSocket.send(packet);
    }
    catch(Exception e){
      e.printStackTrace();
    }
  };

  public DatagramSocket getDatagramSocket(int port){
    DatagramSocket datagramSocket = null;
    try{
      datagramSocket = new DatagramSocket(port);
    }
    catch (IOException e) {
      e.printStackTrace();
    }
    System.out.println("DatagramSocket is ready ...");
```

```
50      return datagramSocket;
51    }

53    public void myAction(DatagramSocket datagramSocket){
54      while(true){
55        service.accept(datagramSocket);
56      }
57    }
58  }
```

– Clasă de lansare a serverului

```
1  package cmmdc.datagram.s;
2  import java.net.DatagramSocket;
3  import cmmdc.datagram.s.impl.MyMServer;
4  import cmmdc.datagram.i.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 }
```

– *module-info*

```
1  module server{}
```

Şi în acest caz este valabilă Observaţia 2.2.1 privind implementarea serverului.

- Aplicaţia client

```
1  package cmmdc.datagram.c;
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 cmmdc.datagram.p.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  }
```

- *module-info*

```
1  module client{}
```

2.3.2 Multicast vs. Broadcast

Clasa `java.net.MulticastSocket`

Prin intermediul unui soclu de tip `MulticastSocket` se pot recepţiona datagrame expediate de un server către toţi clienţii cu un asemenea soclu.

Constructori

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

Metode

- `void joinGroup(InetAddress` *adresă*`) throws SocketException`

  Soclul se "conectează" la grupul definit de adresa IP (de tip D, adică cuprins între 224.0.0.0 şi 239.255.255.255).

- `void leaveGroup(InetAddress` *adresă*`) throws SocketException`
  Soclul se "deconectează" la grupul definit de adresa IP.
- `void close()`

Pregătirea clientului în vederea recepţionării 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 şi 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 şi la un port fixat. Pachetele emise de server sunt recepţionate de orice client ce crează un soclu de tip `MulticastSocket` pentru portul la care emite serverul şi care se alătură grupului.

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

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

Orice client care îşi crează un soclu la portul la care emite serverul recepţionează datagramele trimise de server. Adresa utilizată de server la crearea datagramelor trebuie să identifice reţeaua.

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

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

2.4 Canale de comunicaţie

Odată cu versiunea j2sdk1.4 apar clase noi pentru operaţii de intrare - ieşire în pachetele `java.nio` şi `java.nio.channels`. Pachetul `java.nio.channels` conţine clase pentru comunicaţia în reţea, şi anume canalele de comunicaţie.

MulticastServer

```
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());
}
}
}
```

BroadcastServer

```
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());
}
}
}
```

Table 2.1: Clasele server.

MulticastClient

```
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<5);
socket.leaveGroup(address);
socket.close();
}
}
```

BroadcastClient

```
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<5);

socket.close();
}
}
```

Table 2.2: Clasele client.

2.4.1 `SocketChannel` & `DatagramChannel`

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

Clasa `java.nio.Buffer`

Ierarhia claselor `Buffer`

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

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

Un obiect de tip `Buffer` este caracterizat de

- *capacitate (capacity)* numărul elementelor care pot fi înmagazitate în tampon;
- *limită (limit)* marginea superioră 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 $limita = capacitate$ şi $indice = 0$.
- *flip()* pregăteşte obiectul de tip `Buffer` pentru consultare (citire). Fixează *limita* =numărul elementelor din tampon şi $indice = 0$.
- *rewind()* pregăteşte obiectul de tip `Buffer` pentru re-citire.
- *remaining()* returnează număul octeţilor cuprinşi între valoarea indicelui şi limită.

Clasa `java.nio.ByteBuffer`

Instanţierea unui obiect se obţine prin metoda statică

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

Introducerea şi extragerea datelor din tampon se poate face în mod

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

Metode

Introducerea datelor în mod relativ:

- `ByteBuffer put(byte` *b*`)`
- `ByteBuffer put`*Tip*`(`*tip x*`)`

Extragerea datelor în mod relativ:

- `byte get()`
- *tip* `get`*Tip*`()`

Introducerea datelor în mod absolut:

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

Extragerea datelor în mod absolut:

- `byte get(int` *index*`)`
- *tip* `get`*Tip*`(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 existenţa în tampon a unor octeţi.

- `static ByteBuffer wrap(byte[]` *array*`)`

- `public static ByteBuffer wrap(byte[]` *array*`,int` *offset*`,int` *length*`)`

  Converteşte şirul de octeţi într-un obiect `ByteBuffer`.

- `public final byte[] array()`

  Transformarea inversă, obiectul `ByteBuffer` este convertit într-un şir de octeţi.

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` şi încapsulează adresa unui calculator din Internet împreună cu un port în vederea *legării* la un `ServerSocket`.

Constructori:

- `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 comunicaţie prin metoda

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

Şablonul 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 `Socket Channel` prin care se vor derula comunicaţiile cu clientul. Acest canal de comunicaţie se obţine 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*`)`

şi se recepţionează prin metoda

`int read(ByteBuffer` *destinaţie*`)`

Valoarea returnată de cele două metode reprezintă numărul octeţilor trimişi / recepţionaţi.

Doar octeţii unui obiect de tip `ByteBuffer` cuprinşi între indice şi limită sunt transmişi 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.*

Aplicaţie client-server bazat pe canale de comunicaţie prin socluri se compune din:

- Aplicaţia server alcătuită din:

– Interfaţă

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

– Implementarea interfeţei
Clasa *MyMServer*

```
1  package cmmdc.socketchannel.s.impl;
2  import cmmdc.socketchannel.i.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.service.apply(
33           serverSocketChannel.accept()));
34       }
35       catch(IOException e){
36         e.printStackTrace();
37       }
38     }
39   }
40 }
```

– Clasa *AppThread*

```
1  package cmmdc.socketchannel.s.impl;
2  import java.io.IOException;
3  import java.nio.channels.SocketChannel;
```

```
4  import java.nio.ByteBuffer;
5  import java.util.function.Function;

7  public class AppThread extends Thread{
8    Function<SocketChannel,Thread> service=socketChannel->{
9      return new Thread(()->{
10       try{
11         ByteBuffer bb = ByteBuffer.allocate(16);
12         //LongBuffer lb = bb.asLongBuffer();
13         socketChannel.read(bb);
14         // Varianta 1
15         long m=bb.getLong(0);
16         long n=bb.getLong(8);
17         // Varianta 2
18         // long m=lb.get(0);
19         // long n=lb.get(1);
20         long r=App.cmmdc(m,n);
21         bb.clear();
22         // Varianta 1
23         bb.putLong(0,r);
24         // Varianta 2
25         // lb.put(r);
26         socketChannel.write(bb);
27         socketChannel.close();
28       }
29       catch(IOException e){
30         e.printStackTrace();
31       }
32     });
33   };
34 }
```

– Clasă de lansare a serverului

```
1  package cmmdc.socketchannel.s;
2  import java.nio.channels.ServerSocketChannel;
3  import cmmdc.socketchannel.s.impl.MyMServer;
4  import cmmdc.socketchannel.i.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 }
```

- Aplicaţia client

```
1  package cmmdc.socketchannel.c;
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       InetSocketAddress isa=new InetSocketAddress(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 utilizeară clasa acoperitoare `LongBuffer`.

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

```
. . .
      ByteBuffer bb = ByteBuffer.allocate(1024);
      socketChannel.read(bb);
      byte[] bin=bb.array();
      ByteArrayInputStream bais=new ByteArrayInputStream(bin);
      ObjectInputStream in=new ObjectInputStream(bais);
      Protocol p=(Protocol)in.readObject();
      p.x=App.cmmdc(p.x,p.y);
      p.y=0;
      ByteArrayOutputStream baos=new ByteArrayOutputStream(256);
      ObjectOutputStream out=new ObjectOutputStream(baos);
      out.writeObject(p);
      byte[] bout=baos.toByteArray();
      bb=ByteBuffer.wrap(bout);
      socketChannel.write(bb);
      socketChannel.close();
. . .
```

şi, respectiv,

```
. . .
    ByteBuffer bb=ByteBuffer.allocate(2048);
    Protocol p=new Protocol(m,n);
    try{
      ByteArrayOutputStream baos=new ByteArrayOutputStream(256);
      ObjectOutputStream out=new ObjectOutputStream(baos);
      out.writeObject(p);
      byte[] bout=baos.toByteArray();
      bb=ByteBuffer.wrap(bout);
      sc.write(bb);
      bb.clear();
      sc.read(bb);
      byte[] bin=bb.array();
      ByteArrayInputStream bais=new ByteArrayInputStream(bin);
      ObjectInputStream in=new ObjectInputStream(bais);
      p=(Protocol)in.readObject();
      System.out.println("Cmmdc : "+p.x);
      sc.close();
. . .
```

Clasa `java.nio.channels.DatagramChannel`

Şablonul de programare pentru instanţierea 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 octeţi expediaţi. Recepţia unui `ByteBuffer` se obţine 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.*

- Aplicaţia server este alcătuită din:
  - Interfaţă

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

  - Implementarea interfeţei

```
1  package cmmdc.datagramchannel.s.impl;
2  import cmmdc.datagramchannel.i.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;
9  import java.util.function.Consumer;

11 public class MyMServer implements IMyMServer{
12   static Consumer<DatagramChannel> service=datagramChannel->{
13     try{
14       ByteBuffer bb = ByteBuffer.allocate(16);
15       //LongBuffer lb = bb.asLongBuffer();
16       SocketAddress sa=datagramChannel.receive(bb);
```

```
17        // Varianta 1
18        long m=bb.getLong(0);
19        long n=bb.getLong(8);
20        // Varianta 2
21        // long m=lb.get(0);
22        // long n=lb.get(1);
23        long r=App.cmmdc(m,n);
24        bb.clear();
25        // Varianta 1
26        bb.putLong(0,r);
27        // Varianta 2
28        // lb.put(r);
29        datagramChannel.send(bb,sa);
30      }
31      catch(Exception e){
32        e.printStackTrace();
33      }
34    };

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

52    public void myAction(DatagramChannel datagramChannel){
53      while(true){
54        service.accept(datagramChannel);
55      }
56    }
57  }
```

– Clasă de lansare a serverului

```
1  package cmmdc.datagramchannel.s;
2  import java.nio.channels.DatagramChannel;
3  import cmmdc.datagramchannel.s.impl.MyMServer;
4  import cmmdc.datagramchannel.i.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 }
```

- Aplicaţia client

```
1  package cmmdc.datagramchannel.c;
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 Recepţie cu `Selector`

Programarea recepţiei cererilor clienţilor fără utilizarea firelor de execuţie se poate face utilizând clasele

- `java.nio.channel.Selector`

  Gestionează obiectele `SocketChannel` înregistrate şi transmite cererile care trebuie satisfăcute.

- `java.nio.channel.SelectionKey`

  Obiect utilizat de *selector* pentru sortarea cererilor. O cheie identifică un client şi tipul cererii.

Figura 2.1 prezintă structura unei aplicaţii. [1]

Clasa `java.nio.channel.Selector`

Metode

- `public static Selector open() throws IOException`

  Instaţiază un obiect de tip `Selector`.

- `public Set<SelectionKey> selectedKeys()`

  Returnează cheile înregistrate de selector.

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

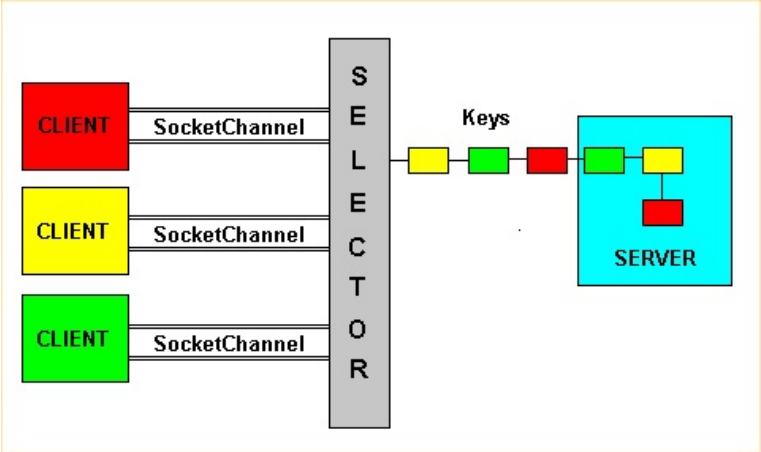


Figure 2.1: Structura unei aplicaţii.

- `public int select() throws IOException`

  Metodă blocantă până la selectarea unui canal.

Înregistrarea selectorului se face de către obiectul `SocketChannel` sau `ServerSocketChannel` prin metoda
`public final SelectionKey register(Selector` *selector*, `int` *ops*`) throws ClosedChannelException`
Operaţiile 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()`

Exemplul 2.4.3 *Aplicaţia server pentru aplicaţia de calcul a celui mai mare divizor comun.*

Aplicaţia server va fi formată de clasele *AppServer* şi *App,* cea utilizată în acest capitol.

```
1  package selector.server;
2  import java.nio.channels.ServerSocketChannel;
3  import java.nio.channels.SocketChannel;
4  import java.nio.channels.SelectionKey;
5  import java.nio.channels.Selector;
6  import java.net.InetSocketAddress;
7  import java.net.ServerSocket;
8  import java.nio.ByteBuffer;
9  import java.util.Iterator;
10 import java.util.Set;
11 import java.io.IOException;

13 public class AppServer{
14   public static void main(String[] args){
15     int port=7999;
16     if(args.length>0)
17       port=Integer.parseInt(args[0]);
18     ServerSocketChannel serverSocketChannel=null;
19     try{
20       serverSocketChannel = ServerSocketChannel.open();
21       InetSocketAddress isa=new InetSocketAddress(port);
22       ServerSocket ss=serverSocketChannel.socket();
23       ss.bind(isa);
24       serverSocketChannel.configureBlocking(false);
25       System.out.println("Server ready... ");

27       Selector selector = Selector.open();
28       SelectionKey serverkey =
29         serverSocketChannel.register(selector, SelectionKey.OP_ACCEPT);
30       ByteBuffer buffer = ByteBuffer.allocate(16);
31       while(true){
32         selector.select();
33         Set<SelectionKey> keys = selector.selectedKeys();
34         for (Iterator<SelectionKey> i = keys.iterator(); i.hasNext();){
35           SelectionKey key = i.next();
36           i.remove();
37           if (key == serverkey) {
38             if (key.isAcceptable()){
39               SocketChannel client = serverSocketChannel.accept();
40               client.configureBlocking(false);
41               SelectionKey clientkey =
42                 client.register(selector, SelectionKey.OP_READ);
43             }
44           }
45           else {
46             if (key.isReadable()){
47               SocketChannel client = (SocketChannel) key.channel();
48               int bytesread = client.read(buffer);
49               long m=buffer.getLong(0);
```

```
50                  long n=buffer.getLong(8);
51                  long r=App.cmmdc(m,n);
52                  buffer.clear();
53                  buffer.putLong(0,r);
54                  client.write(buffer);
55                  buffer.clear();
56                  client.close();
57                  key.cancel();
58                }
59              }
60            }
61          }
62        }
63        catch(Exception e){
64          System.err.println("BufferOpException : "+e.getMessage());
65        }
66        finally{
67          try{
68            serverSocketChannel.close();
69          }
70          catch(Exception e){
71            System.out.println("CloseException : "+e.getMessage());
72          }
73        }
74      }
75  }
```

Aplicaţia client nu suferă nici o modificare.

2.4.3 Comunicaţii asincrone prin canale

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

Astfel sunt introduse clasele:

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

Tehnicile de programare utilizate se bazează pe utilizarea claselor:

- `java.util.concurrent.Future`

Interfaţa `java.util.concurrent.Future<V>`

Metode

- `V get()`

  Metoda este blocantă până la obţinerea rezultatului.

- `boolean isDone()`

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

Un obiect de tip V care rezultă în urma apelării metodei neblocante 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 obţine 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 octeţilor încărcaţi în *dst*.

- `public Future<Integer> write(ByteBuffer` *src*`)`

  Metoda returnează numărul octeţilor expediaţi din *src*.

- `SocketAddress getRemoteAddress()`

  Returnează adresa la distanţă la care s-a conectat. Metoda serveşte la verificarea conectării. Dacă rezultatul este `null` şi se apelează metode de trimitere / recepţie date atunci se generează eroarea `java.nio.channels.NotYetConnectedException`.

Exemplul 2.4.4

Utilizăm modelul aplicaţiei dezvoltată pe baza clasei `ServerSocketChannel`:

1. *IMyMServer.java*

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

30   public void myAction(AsynchronousServerSocketChannel
31         asynchronousServerSocketChannel){
```

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

3. *AppThread*

```
1  package channel.cmmdc.s.impl;
2  import java.nio.channels.AsynchronousSocketChannel;
3  import java.nio.ByteBuffer;
4  import java.util.concurrent.Future;
5  import java.io.IOException;
6  import java.util.function.Function;

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

4. *AppServer.java*

```
1  package server;
2  import java.nio.channels.AsynchronousServerSocketChannel;
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     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       while(asc.getRemoteAddress()==null);
24     }
25     catch (UnknownHostException e) {
26       System.err.println("Server necunoscut: "+host+" "+e.getMessage());
27       System.exit(1);
28     }
29     catch (IOException e) {
30       System.err.println("Conectare imposibila la: "+
31                 host+" pe portul "+port+" "+e.getMessage());
32       System.exit(1);
33     }

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

```
39      System.out.println("n=");
40      n=scanner.nextLong();

42      ByteBuffer bb=ByteBuffer.allocate(16);
43      bb.putLong(0,m).putLong(8,n);
44      try{
45        Future<Integer> fw=asc.write(bb);
46        while(!fw.isDone());
47        bb.clear();
48        Future<Integer> fr=asc.read(bb);
49        while(!fr.isDone());
50        r=bb.getLong(0);
51        System.out.println("Cmmdc : "+r);
52        asc.close();
53      }
54      catch(Exception e){
55        System.err.println("Eroare de comunicatie"+e.getMessage());
56      }
57    }
58  }
```

Comentarii

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

S-au dezvoltat instrumente de programare cu scopul eficientizării din diverse puncte de vedere a comunicaţiilor 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 şi folosind aplicaţiile prezentate mai sus, durata evaluată pe server a fost

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

13. Precizati metodele clasei `DatagramSocket` utilizate la expedierea şi la recepţia unui datagram.

14. Ce parametri caracterizează un obiect de tip `Buffer` ?

15. Ce asemănare există între comunicaţia bazată de datagrame şi cea prin intermediul canalelor ?

16. Care este rolul unui obiect de tip `ByteBuffer`?

Probleme

1. Rezolvarea solicitării unui număă (foarte) mare de clienţi (> 8192.)

2. Editarea diacriticelor în interfeţele grafice (desktop - sub Windows , Web).

Capitolul 3

Regăsirea obiectelor prin servicii de nume

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

Exemple date prin şablonul (Nume ⇒ Referinţă ⇒ Obiect)

1. Accesul la o carte într-o bibliotecă:

   Titlul cărţii ⇒ Referintă cărţii din bibliotecă ⇒ Carte

2. Contactul telefonic cu o persoana:

   Nume persoană ⇒ Număr telefon ⇒ Persoană

Un serviciu de nume conţine asocieri dintre nume de obiecte şi obiecte şi poate oferi facilităţi de regăsire a obiectelor.

3.1 *Java Naming and Directory Interface*

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

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

Alături de interfaţa JNDI, arhitectura JNDI mai conţine interfaţa *Service Provider Interface - SPI.* Implementarea interfeţei SPI de un furnizor de servicii JNDI are ca efect independenţa programului Java de furnizorul de servicii JNDI.

JNDI este implementat de serviciile de nume:

- *Filesystem* are ca obiect asocierea dintre numele de fişier sau catalog cu obiectul corespunzător.
- *DNS - Domain Name System* are ca obiect asocierea dintre adresa Internet cu adresa IP.
- *RMI registry* utilizat în aplicaţii RMI, din Java. Are ca obiect asocierea între un nume de serviciu de invocare ale obiectelor la distanţă cu un delegat al serviciului (stub).
- *COS - Common Object Service Naming* utilizat în aplicaţii CORBA. Are ca obiect asocierea între numele unui serviciu de invocare ale obiectelor la distanţă cu referinţa la serviciu.
- *LDAP - Lightweight Directory Access Protocol* defineşte un protocol pentru accesarea datelor reţinute într-un catalog LDAP (*LDAP directory, information directory*). Un catalog LDAP permite reţinerea şi regăsirea referinţelor obiectelor definite pe un calculator.

Interfaţa `javax.naming.Context`

Printr-un context se va înţelege o mulţime de asocieri *nume - obiect.* Corespunzător unui context, JNDI defineşte interfaţa `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 împreuna cu tipul lor.
- `NamingEnumeration listBindings(String` *nume*`)`

  Returnează lista cu nume obiectelor împreuna cu tipul şi locaţia acestora.

Specificaţiile JNDI prevăd definirea unui *context iniţial,* implementat prin clasa `javax.naming.InitialContext,` clasă ce implementează interfaţa `Context.`

Constructori.

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

Pentru crearea *contextului iniţial* trebuie specificată clasa care crează contextul iniţial 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 `Inital Context.`

  ```
  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 fişierul de proprietăţi *jndi.properties.*

  În aceste ultime două cazuri, contextul initial se crează prin

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

Funcţie de serviciul de nume, clasa care crează contextul iniţial 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 afişează conţinutul unui catalog indicat de client.*

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

12 class Lookup{
13   public static void main(String[] args) {
14     Context ctx=null;
15     /*
16     //  Varianta 1
17     Hashtable env = new Hashtable(11);
18     env.put(Context.INITIAL_CONTEXT_FACTORY,
19         "com.sun.jndi.fscontext.RefFSContextFactory");
20     try{
21       ctx = new InitialContext(env);
22     }
23     catch (NamingException e) {
24       System.out.println("InitialContextError : "+e.getMessage());
25     }
26     */
27     /*
28     // Varianta 2
29     System.setProperty("java.naming.factory.initial",
30       "com.sun.jndi.fscontext.RefFSContextFactory");
31     try{
32       ctx = new InitialContext();
33     }
34     catch (NamingException e) {
35       System.out.println("InitialContextError : "+e.getMessage());
36     }
37     */

39     // Varianta 3
40     try{
```

```
41        ctx = new InitialContext();
42      }
43      catch (NamingException e) {
44        System.out.println("InitialContextError : "+e.getMessage());
45      }

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

54        System.out.println("\nContinutul catalogului "+myName+" este:\n");
55        NamingEnumeration<NameClassPair> lst=ctx.list(myName);
56        while(lst.hasMore()){
57          NameClassPair nc = lst.next();
58          System.out.println(nc);
59        }

61        System.out.println("\nContinutul catalogului "+myName+" este:\n");
62        NamingEnumeration<Binding> lst1 = ctx.listBindings(myName);
63        while (lst1.hasMore()) {
64          Binding bd = lst1.next();
65          System.out.println(bd);
66        }
67        ctx.close();
68      }
69      catch (NamingException e) {
70        System.out.println("Lookup failed: " + e.getMessage());
71      }
72    }
73  }
```

```
1  module fs{
2    requires java.naming;
3  }
```

Prelucrarea în linie de comandă constă din:

1. `javac -d mods --module-path lib *.java`
2. Apelarea clasei, în varianta dată, constă din

```
java --module-path mods
  -cp fscontext-4.4.2.jar
  -Djava.naming.factory.initial=com.sun.jndi.fscontext.RefFSContextFactory
  -m fs/fs.Lookup
```

Dacă în catalogul `mods\fs` se găseşte fişierul *jndi.properties* cu conţinutul

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

atunci apelarea va fi

```
java --module-path mods -cp fscontext-4.4.2.jar -m fs/fs.Lookup
```

Serviciul de nume *Filesystem* este dat de fişierul *fscontext-*.jar*.

3.1.1 LDAP

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

Punctele de intrare (rădăcinile) sunt date de *DN* (*Distinguished Name*). Uzual *DN* se defineşte 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* şi are acces la LDAP prin precizarea simultană a atributelor *DN* şi *cn*.

Vom utiliza *Apache Directory Service*[1] - *ApacheDS*. Produse pentru administrarea serverului prin intermediul unei interfeţe grafice sunt:

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

Instalarea constă din dezarhivarea fişierului descărcat din Internet.

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

`Context.PROVIDER_URL`	*ldap://localhost:10389/uid=admin, ou=system*
`Context.SECURITY_PRINCIPAL`	*uid=admin,ou=system*
`Context.SECURITY_CREDENTIALS`	*secret*

[1] http://en.wikipedia.org/wiki/List_of_LDAP_software.

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

Pentru oprire se tastează Ctrl+C.

Acţiunile care pot fi întreprinse de un client care interacţionează cu serverul constau în

- autentificare: datele necesare sunt DN şi 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 şi ştergerea referinţei unui obiect de tip Cmmdc.*

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

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

```
38      catch (NamingException e) {
39        System.out.println("LDAPserverCmmdc :  "+e.getMessage());
40      }
41    }
42  }
```

Codul clasei *Cmmdc* este

```
1  package ldap;
2  private static final long serialVersionUID = 1L;

4  public class Cmmdc implements java.io.Serializable{
5    public long cmmdc(long a, long b) {. . .}
6  }
```

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

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

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

```
39      }
40    }
41  }
```

```
1  module ldap{
2    requires java.naming;
3  }
```

Execuţia în linie de comanda revine la comenzile:

```
javac -d mods *.java

java --module-path mods -m ldap/ldap.LDAPServerCmmdc

java --module-path mods -m ldap/ldap.LDAPClientCmmdc
```

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 distanţă

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

Se vor prezenta două cazuri:

- Invocarea procedurilor la distanţă în cazul mediului omogen Java. Denumirea tehnologiei în acest caz este *Invocarea metodei la distanţă – Remote Method Invocation (RMI).* Prin mediu omogen se înţelege faptul că atât compontenta server cât şi componenta client sunt programate în acelaşi limbaj de programare, Java în cazul de faţă.

- Invocarea procedurilor la distanţă în cazul medii neomogene. Soluţia în acest caz este dat de *Common Object Request Broker Arhitecture (CORBA).*

4.1 Remote Method Invocation

Regăsirea obiectelor la distanţă. Ideea găsirii unui obiect la distanţă 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 distribuţia java şi se lansează în execuţie prin

`rmiregistry` [*port*]

unde valoarea implicită a portului este 1099.

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

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

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

Serializarea datelor reprezintă transformarea acestora din tipuri de date diferite într-un şir de octeţi care va fi transportat prin reţea fără interpretare, dar care păstrează informaţiile despre structura iniţială a datelor.

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

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

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

Metodele apelate la distanţă trebuie declarate ca aparţinând unei interfeţe ce extinde interfaţa `java.rmi.Remote.` Fiecare asemenea metodă trebuie să arunce o excepţie `java.rmi.RemoteException.` Obiectele care circulă prin reţea trebuie să implementeze interfaţa `java.io.Serializable.`

Structura unei aplicaţii RMI. O aplicaţie RMI este alcătuită din trei componente:

1. O interfaţă la distanţă (*remote*) în care se declară serviciile puse la dispoziţie de server;

2. Aplicaţia server care poate implementa serviciile interfeţei la distanţă şi înscrie în *registry* stub-ul corespunzător;

3. Aplicaţia client ce apelează unul sau mai multe servicii ale serverului.

Aplicaţia server trebuie să aibă acces la clasele interfeţei şi a celor care o implementează. Locaţia 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 interfeţei şi 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 aplicaţiei server, serverul `http / ftp` trebuie să fie activ.

Dacă se indică prin protocolul `file` calea către cataloagele care conţin clasele interfeţei şi ale implementării lor atunci ultimul caracter al căii este /.

Din punctul de vedere al execuţiei sunt implicate componentele:

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

Aplicaţia server şi *rmiregistry* trebuie să ruleze pe acelaşi calculator.

Registrul *rmiregistry* implementează interfaţa `java.rmi.registry.Registry`[1]. Metodele oferite sunt:

- `void bind(String` *numeServiciu*`, Remote` *obj*`)throws RemoteException, AlreadyBoundException, AccessException`

  Înregistrează în *registry* obiectul *obj* ce implementează interfaţa `Remote` sub numele *numeServiciu.*

- `void rebind(String` *numeServiciu*`, Remote` *obj*`)throws RemoteException, AccessException`

  Reînregistrează în *registry* obiectul *obj* ce implementează interfaţa `Remote` sub numele *numeServiciu.*

  Această metodă poate fi apelată doar dacă programul care face înregistrarea se află pe aceaşi maşină 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`

  Şterge din registru serviciul.

[1] Varianta directă, fără a folosi facilităţile JNDI pentru *rmiregistry.*

Localizarea registrului se obţine 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 aplicaţie / serviciu.

Interfaţa `Remote` serveşte la marcarea interfeţelor ale căror metode urmează a fi apelate de pe altă maşină virtuală Java.

Un obiect de tip `java.rmi.server.UnicastRemoteObject` mijloceşte transmiterea obiectelor şi serveşte la generarea unui stub unui serviciu. Generarea stub-ului unui serviciu se obţine prin intermediul metodei statice

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

Înregistrarea stub-ului unui serviciu descris de interfaţa *IService* şi 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 doreşte să folosească trebuie să cunoască :

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

4.1.1 Crearea unei aplicaţii RMI

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

Dezvoltarea unei aplicaţii se va face pe un calculator. În acest sens, celor 3 componente li se asociază cataloagele *i* - pentru interfaţa la distanţă, *s* - pentru componenta server şi *c* - pentru componenta client. Paşi necesari compilării şi desfăşurării componentelor aplicaţiei se vor realiza prin intermediul lui *ant.*

Dezvoltarea unei aplicaţii RMI constă din parcurgerea următorilor paşi:

1. Definitea interfeţei la distanţă.

```
1 package cmmdc.rmi.i;

3 public interface ICmmdc extends java.rmi.Remote {
4     long cmmdc(long a,long b) throws java.rmi.RemoteException;
5 }
```

   Presupunem ca programatorul interfeţei reţine textul sursă `ICmmdc.java` în catalogul `\i\src`

2. Compilarea şi arhivarea interfeţei se poate realiza fişierul *ant-build*

```
1 <project name="Interface" default="Compile" basedir=".">
2   <property name="build.dir" location="build"/>
3   <property name="archive.name" value="icmmdc"/>

5   <target name="Init">
6     <!-- Create the time stamp -->
7     <tstamp/>
8     <delete dir="${build.dir}"/>
```

```
9       <mkdir dir="${build.dir}"/>
10    </target>

12    <target name="Compile" depends="Init">
13      <javac srcdir="src" destdir="${build.dir}"
14        includeantruntime="false">
15        <compilerarg value="-Xlint"/>
16      </javac>
17      <jar basedir="${build.dir}" destfile="${archive.name}.jar"/>
18    </target>
19  </project>
```

Arhiva interfeţei se distribuie celor care realizează aplicaţia server şi client.

3. Implementarea interfeţei *remote* prin construirea aplicaţiei server.

```
1  package cmmdc.rmi.s;

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

12 public class CmmdcImpl  implements ICmmdc{

14     public long cmmdc(long a,long b){. . .}

16     public static void main(String args[]) {
17       String host="localhost";
18       int port=1099;
19       if(args.length>0)
20          port=Integer.parseInt(args[0]);
21       try {
22         CmmdcImpl obj=new CmmdcImpl();
23         ICmmdc stub=(ICmmdc)UnicastRemoteObject.exportObject(obj,0);
24         // Varianta cu apel rmiregistry direct
25         /*
26         Registry registry=LocateRegistry.getRegistry(host,port);
27         registry.bind("CmmdcServer",stub);
28         */
29         // Varianta JNDI
30         String sPort=Integer.valueOf(port).toString();
31         System.setProperty(Context.INITIAL_CONTEXT_FACTORY,
32           "com.sun.jndi.rmi.registry.RegistryContextFactory");
33         System.setProperty(Context.PROVIDER_URL,"rmi://"+host+":"+sPort);
34         Context ctx=new InitialContext();
35         ctx.bind("CmmdcServer",stub);
36         System.out.println("CmmdcServer ready");
37         System.out.println("Press CTRL+C to finish !");
38       }
```

```
39        catch (Exception e) {
40          System.out.println("CmmdcImpl err: " + e.getMessage());
41        }
42      }
43  }
```

```
1  module scmmdc{
2    requires java.rmi;
3    requires java.naming;
4    exports cmmdc.rmi.i;
5  }
```

Textul sursă al programelor se reţin în catalogul `\s\src`.

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

   se obţin cu *ant* cu

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

4    <property name="path" location=".."/>
5    <property name="interface.archive.name" value="icmmdc.jar"/>
6    <property name="interface.jar.location" location="${path}/i" />
7    <property name="service-class" value="cmmdc.rmi.s.CmmdcImpl"  />
8    <property name="port" value="1099"/>
9    <property name="hostResources" value="localhost"/>

11   <target name="Init">
12     <delete dir="mods"/>
13     <mkdir dir="mods"/>
14     <copy file="${interface.jar.location}/${interface.archive.name}"
15           todir="${basedir}"/>
16     <unjar src="${interface.archive.name}" dest="mods"/>
17   </target>

19   <target name="Compile" depends="Init">
20     <javac srcdir="src" destdir="mods" includeantruntime="false">
21       <compilerarg value="-Xlint"/>
22     </javac>
23   </target>

25   <target name="Archive">
26      <jar destfile="cmmdc.jar" basedir="${build.dir}"/>
27   </target>

29   <target name="Rmi">
30      <exec executable="rmiregistry">
31        <env key="CLASSPATH" value="mods"/>
32        <arg value="${port}" />
33      </exec>
```

```
34    </target>

36    <target name="Server">
37       <java classname="${service-class}" fork="true"
38          modulepath="mods" module="scmmdc">
39          <jvmarg value="-Djava.rmi.server.codebase=file:${path}/s/mods"/>
40          <!--
41          <jvmarg value="-Djava.rmi.server.codebase=
42             http://${hostResources}:8080/rmi/cmmdc.jar" />
43          -->
44          <!--
45          <jvmarg value="-Djava.rmi.server.codebase=
46              ftp://${hostResources}:2121/rmi/cmmdc.jar" />
47          -->
48          <arg line="${port}"/>
49       </java>
50    </target>
51 </project>
```

Obiectivele *Rmi* şi *Server* se lansează în ferestre `dos` distincte care rămân active pe durata de viaţă a aplicaţiei server.

Obiectivul *Archive* crează arhiva `jar` necesară desfăşurării interfeţei şi a claselor care o implementează într-un server `http / ftp`. Aşa cum s-a amintit mai sus, acestă arhivă este preluată de aplicaţia 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 obţine prin

- Sistemul de operare Windows

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

- Sistemul de operare Linux

```
FTP_SERVER_HOME=. . .
$FTP_SERVER_HOME/bin/ftpd.sh res/conf/ftpd-typical.xml
```

Resursele / fişierele puse la dispoziţie de serverul ftp sunt puse în catalogul `FTP_SERVER_HOME\res\home`.

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

```
cmmdc
|      ICmmdc.class
server
|      CmmdcImpl.class
```

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

5. Realizarea programului client

```
1  package cmmdc.rmi.c;
2  import cmmdc.rmi.i.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 cu apel rmiregistry direct
27       /*
28       Registry registry=LocateRegistry.getRegistry(host,port);
29       ICmmdc obj=(ICmmdc)registry.lookup("CmmdcServer");
30       */
31       // Varianta JNDI
32       String sPort=Integer.valueOf(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 }
```

```
1  module scmmdc{
2    requires java.rmi;
3    requires java.naming;
```

```
4 }
```

Sursele programelor client se reţin în catalogul `\c\src`.

6. Compilarea programului client şi lansarea acestuia în execuţie.

```
<project name="Client" default="Run" basedir=".">
    <description>Client actions</description>

  <property name="path" location=".."/>
  <property name="build.dir" value="mods"/>
  <property name="interface.archive.name" value="icmmdc.jar"/>
  <property name="interface.jar.location" location="${path}/i" />
  <property name="host" value="localhost" />
  <property name="port" value="1099" />
  <property name="client-class" value="cmmdc.rmi.c.CmmdcClient" />

  <target name="Init">
    <!-- Create the time stamp -->
    <tstamp/>
    <delete dir="${build.dir}"/>
    <mkdir dir="${build.dir}"/>
    <copy file="${interface.jar.location}/${interface.archive.name}"
          todir="${basedir}"/>
    <unjar src="${basedir}/${interface.archive.name}" dest="mods"/>
  </target>

  <target name="Compile" depends="Init">
    <javac srcdir="src" destdir="${build.dir}" modulepath="mods"
        includeantruntime="false">
        <compilerarg value="-Xlint"/>
    </javac>
  </target>

  <target name="Run" depends="Compile">
    <java classname="${client-class}" fork="true"
       modulepath="mods" module="ccmmdc">
      <arg line="${host} ${port}" />
    </java>
  </target>
</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 instanţă de server. Aceste metode sunt declarate într-o interfaţă la distanţă.

Un client, apelând o asemenea metodă, va instanţia serverul dorit şi va obţine stub-ul corespunzător.

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

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

- Clasa serverului extinde clasa `UnicastRemoteObject.`
- Există un constructor fără argumente ce *aruncă* excepţia `java.rmi.Remote Exception.`

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

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

Acească interfaţă 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 distanţă prin fabrica de obiecte *FabObiecte,* o clasă ce implementează interfaţa

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

Codurile claselor *ServerCmmdc* şi *FabObiecte* sunt

```
1 package cmmdc.rmi.s;
2 import java.rmi.RemoteException;
3 import java.rmi.server.UnicastRemoteObject;
4 import cmmdc.rmi.i.ICmmdc0;

6 public class ServerCmmdc extends UnicastRemoteObject implements ICmmdc0{

8   public ServerCmmdc()throws RemoteException{}

10  public long compute(long m,long n)throws RemoteException{
11    . . .
12  }
13 }
```

respectiv

```
1 package cmmdc.rmi.s;
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 cmmdc.rmi.i.IFabObiecte;
```

```
7  import cmmdc.rmi.i.ICmmdc0;

9  public class FabObiecte implements IFabObiecte{

11   public ICmmdc0 getCmmdc() throws RemoteException{
12     ServerCmmdc cmmdc=new ServerCmmdc();
13     return cmmdc;
14   }

16   public static void main(String args[]){
17     String host="localhost";
18     int port=1099;
19     if (args.length>0)
20        host=args[0];
21     try{
22       FabObiecte obj=new FabObiecte();
23       IFabObiecte stub=(IFabObiecte)UnicastRemoteObject.exportObject(obj,0);
24       Registry registry=LocateRegistry.getRegistry(host,port);
25       registry.bind("ObjectFactory",stub);
26       System.out.println("ObjectFactory ready");
27     }
28     catch(Exception e){
29       System.out.println("Factory err "+e.getMessage());
30     }
31   }
32 }
```

Aplicaţia client se compune din două clase

1. *RemoteClient*

   Clientul obţine stub-ul serviciului, prin intermediul căruia apelează metada *getCmmdc()* a fabricii de obiecte, obţinând un obiect *remote* de tip *ServerCmmdc,* ce implementează interfaţa la distanţă *ICmmdc0.*

```
1  package cmmdc.rmi.c;
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 cmmdc.rmi.i.IFabObiecte;
7  import cmmdc.rmi.i.ICmmdc0;

9  public class RemoteClient extends UnicastRemoteObject{

11   ICmmdc0 remote=null;

13   public RemoteClient() throws RemoteException{}

15   public RemoteClient(String host,int port)throws RemoteException{
16     try{
17       Registry registry=LocateRegistry.getRegistry(host,port);
18       IFabObiecte obj=(IFabObiecte)registry.lookup("ObjectFactory");
19       remote=obj.getCmmdc();
20     }
21     catch(Exception e){
22       System.out.println("ClientException: "+e.getMessage());
23     }
```

```
24    }
25 }
```

2. *ClientCmmdc0*
   Apelează metoda *compute* a obiectului *remote.*

```
1  package cmmdc.rmi.c;
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 numeşte apel invers apelarea de către programul server a unei metode a clientului.

În RMI realizarea unui apel invers presupune:

1. definirea unei interfeţe la distanţă ce va fi implementat de programul client;

2. programul client ce implementează interfaţa extinde clasa `UnicastRemote Object` şi are un constructor ce aruncă o excepţie `java.rmi.RemoteException.`

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

Extindem interfaţa *ICmmdc0* cu metoda

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

```
1  package cmmdc.rmi.i;
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 interfaţa

```
1  package cmmdc.rmi.i;
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 cmmdc.rmi.s;
2  import java.rmi.RemoteException;
3  import java.rmi.server.UnicastRemoteObject;
4  import cmmdc.rmi.i.ICmmdc0;
5  import cmmdc.rmi.i.ICallbackCmmdc;

7  public class ServerCmmdc extends UnicastRemoteObject
8                            implements ICmmdc0{
9    private String method=null;

11   public ServerCmmdc()throws RemoteException{}

13   public long compute(long m,long n){
14     long x=0;
15     if(method.equals("NERECURSIV"))
16       x=nerecursiv(m,n);
17     if(method.equals("RECURSIV"))
18       x=recursiv(m,n);
19     return x;
20   }

22   public void setMethod(ICallbackCmmdc obj)throws RemoteException{
23     method=obj.getMethod();
24   }

26   private long recursiv(long a,long b){
27     if (a==b)
28       return a;
29     else
30       if (a<b)
31         return recursiv(a,b-a);
32       else
33         return recursiv(a-b,b);
34   }

36   private long nerecursiv(long m,long n){
37     long r,c;
38     do{
39       c=n;
40       r=m%n;
```

```
41        m=n;
42        n=r;
43      }
44      while(r!=0);
45      return c;
46    }
47  }
```

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

Programele client devin

```
1  package cmmdc.rmi.c;
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;
7  import cmmdc.rmi.i.ICallbackCmmdc;
8  import cmmdc.rmi.i.IFabObiecte;
9  import cmmdc.rmi.i.ICmmdc0;

11 public class RemoteClient extends UnicastRemoteObject
12                          implements ICallbackCmmdc{
13   ICmmdc0 remote=null;

15   public RemoteClient() throws RemoteException{}

17   public String getMethod(){
18     Scanner scanner=new Scanner(System.in);
19     System.out.println("Alegeti varianta algoritmului lui Euclid");
20     System.out.println("1 - algoritmul ne-recursiv");
21     System.out.println("2 - algoritmul recursiv");
22     int x=scanner.nextInt();
23     String method=null;
24     if(x==1)
25       method="NERECURSIV";
26     else
27       method="RECURSIV";
28     return method;
29   }

31   public RemoteClient(String host,int port)throws RemoteException{
32     try{
33       Registry registry=LocateRegistry.getRegistry(host,port);
34       IFabObiecte obj=(IFabObiecte)registry.lookup("ObjectFactory");
35       remote=obj.getCmmdc();
36     }
37     catch(Exception e){
38       System.out.println("ClientException: "+e.getMessage());
39     }
40   }
41 }
```

şi

```
1  package cmmdc.rmi.c;
```

```
2  import java.util.Scanner;
3
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 distanţă

O instanţă a clasei `UnicastRemoteObject` reprezintă un obiect la distanţă care rulează în permanenţă, folosind resursele maşinii server.

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

Invocarea de la distanţă a unei metode aparţinând unui asemenea obiect are ca efect activarea obiectului.

Pe fiecare maşină virtuală Java există un grup de activare (activation group), care realizează activarea. Activarea este făcută de un activator. Un activator conţine o tabelă care face legătura dintre clasa obiectului cu URL-ul acestuia şi eventual, cu date necesare iniţializării 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 distanţă nu implică modificări. Modificările intervin numai din punctul de vedere al serverului şi al înregistrării sale.

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

Aplicaţia server este compusă din două clase

1. o clasa ce implementează interfaţa *ICmmdc : CmmdcActivabil;*
2. clasa *Setup,* cu metoda `main,` care face posibilă mecanismul de activare şi înscrie serviciul în *registry.* Această clasă este preluată din tutorialul dedicat activării a documentaţiei ce însoţeşte distribuţia Java.

Clasa ce implementează interfaţa la distanţă trebuie

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

Codul sursă al clasei *CmmdcActivabil* este

```
1  package cmmdc.rmi.s;
2  import java.rmi.RemoteException;
3  import java.rmi.MarshalledObject;
4  import java.rmi.activation.Activatable;
5  import java.rmi.activation.ActivationID;
6  import cmmdc.rmi.i.ICmmdc;

8  public class CmmdcActivabil extends Activatable
9      implements ICmmdc{

11   public CmmdcActivabil(ActivationID id,MarshalledObject 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 proprietăţi:

1. drepturile de securitate ale grupului de activare

   `myactivation.policy=group.policy;`

   cu conţinutul

```
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 interfaţa

   `myactivation.impl.codebase;`

4. localizarea unui fişier cu date de iniţializare 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, instanţă a clasei `ActivationGroupDesc.` Grupul de activare este un container ce gestionează obiectele activabile conţinute.

2. Înregistrarea descriptorului grupului de activare, instanţă a clasei `ActivationDesc` şi obţinerea unui identificator al grupului de activare, instanţă a clasei `ActivationGroupID.`

3. Construirea descriptorului de activare. Trebuie cunoscute

   - idendificatorul grupului de activare;
   - numele clasei ce implementează interfaţa la distanţă;
   - localizarea (URL) clasei ce implementează interfaţa la distanţă;
   - obiectul de tip `MarshalledObject,` cu datele de iniţializare a obiectului activabil.

4. Înregistrarea descriptorului de activare, în urma căreia se obţine stub-ul obiectului activabil.

5. Înregistrarea stub-ului împreună cu numele serviciului în *registry*.

Codul clasei *Setup* este

```
1  package cmmdc.rmi.s;
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     // identificatorului 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` şi respectiv `Setup.java` sunt în catalogul `\c\src\acmmdc`.

Lansarea serverului în execuţie constă din

1. Pornirea *registry*-ului (*startrmiregistry*)

   - Sistemul de operare Windows

     ```
     set cale=. . ./acmmdc
     set classpath=%cale%/s/public/classes/cmmdc.jar;
       %cale%/s/public/classes
     start rmiregistry
     ```

   - Sistemul de operare Linux

     ```
     #!/bin/bash
     cale=. . ./acmmdc
     export CLASSPATH=$cale/s/public/classes/cmmdc.jar:
        $cale/s/public/classes
     rmiregistry
     ```

2. Pornirea *demon*-ului `rmid`

   ```
   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* (*eacmmdc*)

   - Sistemul de operare Windows
     - Varianta nemodulară

```
set cale=. . ./catalogul_aplicatiei
set classpath=%cale%/s/icmmdc.jar;%cale%/s/build
java -Djava.rmi.server.codebase=file:%cale%/s/build/
    -Dmyactivation.impl.codebase=file:%cale%/s/build/
    -Dmyactivation.name=CmmdcServer
    -Dmyactivation.file=""
     -Dmyactivation.policy=group.policy
     cmmdc.rmi.s.Setup cmmdc.rmi.s.CmmdcActivabil
```

     - Varianta modulara

```
set cale=. . ./catalogul_aplicatiei
java -Djava.rmi.server.codebase=file:%cale%/s/mods/
    -Dmyactivation.impl.codebase=file:%cale%/s/mods/
    -Dmyactivation.name=CmmdcServer -Dmyactivation.file=""
    -Dmyactivation.policy=group.policy
    --module-path %cale%/s/mods
    -m scmmdc/cmmdc.rmi.s.Setup cmmdc.rmi.s.CmmdcActivabil
```

   - Sistemul de operare Linux

```
#!/bin/bash
cale=. . ./acmmdc
export CLASSPATH=$cale/s/public/classes/cmmdc.jar:$cale/s/public/classes
java -Djava.rmi.server.codebase=file:$cale/s/public/classes/
      -Dmyactivation.impl.codebase=file:$cale/s/public/classes/
      -Dmyactivation.name=CmmdcServer -Dmyactivation.file=""
      -Dmyactivation.policy=group.policy
      acmmdc.Setup acmmdc.CmmdcActivabil
```

   Localizarea (URL) interfeţei la distanţă se precizează prin `java.rmi.server.codebase;`

   Drept client se utilizează programul realizat în secţiunea 4.1.

4.2 CORBA

Reţelele de calculatoare sunt eterogene în timp ce majoritatea interfeţelor de programare a aplicaţiilor sunt orientate spre platforme omogene.

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

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

Aplicaţia 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 funcţiile oferite de aplicaţia server.

ORB este un pachet de servicii, independent de aplicaţii, dar care permit aplicaţiilor să interacţioneze prin reţea. ORB face parte din *middleware* – un intermediar între softul de reţea şi cel de aplicaţie.

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.

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

CORBA face o separare între interfaţa unui obiect şi implementarea sa şi foloseşte un limbaj neutru pentru definirea interfeţelor: `IDL -- Interface Definition Language.`

IDL permite realizarea descrierii de interfeţe independent de limbajul de programare şi de sistemul de operare folosit. O interfaţă IDL defineşte legătura dintre client şi server.

4.2.1 Conexiunea RMI - CORBA

Firma *Oracle - Sun Microsystems* a dezvoltat o soluţie prin care interfeţele RMI pot fi implementate pentru a putea fi accesate ca obiecte CORBA. Aceasta este cunoscută sub numele de soluţia RMI-IIOP, concretizată printr-o serie de pachete din distribuţia jdk. În acest fel nu mai este necesar utilizarea limbajului IDL pentru descrierea interfeţelor la distanţă.

Instrumente necesare utilizării soluţiei RMI-IIOP:

- compilatorul `rmic`

  Opţiunea `-iiop` genereaza stub-ul şi legătura (`tie`) din partea serverului.

  Cu opţiunea `-d` se specifică catalogul în care aceste fişiere sunt scrise.

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

  `orbd -ORBInitialPort` [*port*]

Programele `orbd, rmic` sunt în distribuţia jdk.

Dezvoltarea unei aplicaţii RMI-IIOP

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

1. Elaborarea interfeţei este identică cu cea a aplicaţiei RMI.

2. Implementarea interfeţei

```
1  package cmmdc.iiop.s;
2  import javax.rmi.PortableRemoteObject;
3  import cmmdc.ICmmdc;
4  import java.rmi.RemoteException;

6  // Se extinde clasa PortableRemoteObject
7  // si nu UnicastRemoteObject

9  public class CmmdcImpl extends PortableRemoteObject implements ICmmdc{
10   // Constructorul clasei
11   public CmmdcImpl() throws RemoteException {}

13   public long cmmdc(long a,long b){. . .}
14 }
```

3. *Realizarea programului server.*

```
1  package cmmdc.iiop.s;
2  import javax.naming.InitialContext;
3  import javax.naming.Context;

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();

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* şi *CmmdcServer.java.*

5. Generarea stub-ului *cmmdc.rmi.i._ICmmdc_Stub.class* corespunzător interfeţei *ICmmdc* şi a fişierului _Tie *cmmdc.rmi.s._CmmdcImpl_Tie.class* corespunzător clasei *CmmdcImpl.* Acestea se obţin rulând utilitarul `rmic` - din distrubuţia Java – cu opţiunea `-iiop`

```
rmic -iiop cmmdc.iiop.s.CmmdcImpl
```

6. Pornirea serverului CORBA de regăsire a serviciilor

```
orbd -ORBInitialPort 1050
```

7. Lansarea serverului în execuţie.

   Activităţile legate de server se obţin prin `ant` cu fişierul *build*

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

4    <property name="path" location=".."/>
5    <property name="build.dir" value="mods"/>
6    <property name="interface.archive.name" value="icmmdc.jar"/>
7    <property name="interface.jar.location" location="${path}/i" />
8    <property name="interface-impl" value="cmmdc.iiop.s.CmmdcImpl"  />
9    <property name="service-class" value="cmmdc.iiop.s.CmmdcServer"  />
10   <property name="host" value="localhost" />
11   <property name="port" value="1050" />

13   <path id="myclasspath">
14     <pathelement path="${build.dir}"/>
15   </path>

17   <target name="Init">
18     <delete dir="${build.dir}"/>
19     <mkdir dir="${build.dir}"/>
20     <copy file="${interface.jar.location}/${interface.archive.name}"
21           todir="${basedir}"/>
22     <unjar src="${basedir}/${interface.archive.name}" dest="${build.dir}"/>
23   </target>

25   <target name="Compile" depends="Init">
26     <javac srcdir="src" destdir="${build.dir}"
27         includeantruntime="false" classpathref="myclasspath">
28        <compilerarg value="-Xlint"/>
```

```
29      </javac>
30      <rmic classname="${interface-impl}"
31            sourcebase="src"
32            iiop="yes"
33            base="${build.dir}"
34            classpath="${build.dir}"/>
35    </target>

37    <target name="Orb">
38       <exec executable="orbd">
39         <arg line="-ORBInitialPort ${port} -ORBInitialHost ${host}" />
40       </exec>
41    </target>

43    <target name="Server">
44       <java classname="${service-class}"
45           modulepathref="myclasspath" fork="true" module="iiop">
46          <arg line="${host} ${port}"/>
47       </java>
48    </target>
49  </project>
```

orbd şi aplicaţia server se pot afla pe calculatoare distincte.

8. *Editarea programului client.*

```
1  package cmmdc.iiop.c;
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 Servctxe
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 şi lansarea clientului în execuţie. Clientul trebuie să dispună de fişierul stub (`cmmdc.rmi.i._ICmmdc_Stub.class`) şi bineînţeles de interfaţa `cmmdc.rmi.i.ICmmdc.jar`.

   Fişierul *buildfile* pentru compilare:

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

4    <property name="path" location=".."/>
5    <property name="build.dir" value="mods"/>
6    <property name="stub.location" location="${path}/s/mods/cmmdc/rmi/i" />
7    <property name="host" value="localhost" />
8    <property name="port" value="1050" />
9    <property name="client-class" value="cmmdc.iiop.c.CmmdcClient" />

11   <path id="myclasspath">
12     <pathelement path="${build.dir}"/>
13   </path>

15   <target name="Init">
16     <!-- Create the time stamp -->
17     <tstamp/>
18     <delete dir="${build.dir}"/>
19     <mkdir dir="${build.dir}"/>
20     <mkdir dir="${build.dir}/cmmdc/rmi/i"/>
21     <copy todir="${build.dir}/cmmdc/rmi/i">
22       <fileset dir="${stub.location}"
23         includes="*.class"/>
24     </copy>
25   </target>

27   <target name="Compile" depends="Init">
28     <javac srcdir="src" destdir="${build.dir}" classpathref="myclasspath"
29         includeantruntime="false">
30        <compilerarg value="-Xlint"/>
31     </javac>
32   </target>
33 </project>
```

   şi lansare

```
java --add-modules java.corba -cp mods cmmdc.iiop.c.CmmdcClient
```

4.2.2 Aplicaţie Java prin CORBA

Scopul acestei secţiuni este prezentarea dezvoltării unei aplicaţii pe baza unei interfeţe bazat pe IDL.

Pentru dezvoltarea aplicaţiilor în limbajul de programare Java, translatarea interfeţei IDL în Java se realizează cu utilitarul `idlj` din distribuţia *jdk*.

Corespondenţa între entităţile IDL şi 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: Entităţi IDL şi 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 aplicaţii distribuite CORBA în cazul în care serviciul pus la dispoziţie de server este calculul celui mai mare divizor comun a două numere naturale.

Dezvoltarea unei aplicaţii distribuite CORBA cu mediul nativ Java presupune parcurgerea următoarelor paşi:

1. Realizarea interfeţei IDL care înseamnă

   (a) Editarea programului de interfaţă:

```
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 fişier denumit `Cmmdc.idl`.
   Cmmdc va fi numele interfeţei utilizat de un client şi implementat de server. Serviciul conţine o singură metodă *cmmdc.*

   (b) Translatarea interfeţei în Java

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

   Programul `idlj` crează un subcatalog `CmmdcApp` cu un pachet Java `CmmdcApp` conţinând fişierele:

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

2. Realizarea programelor server. Punem în evidenţa programul servant *CmmdcImpl.java* ce implementează interfaţa *Cmmdc.*

```
1  package CmmdcApp;
2  import org.omg.CORBA.ORB;

4  public class CmmdcImpl extends CmmdcPOA {
5    private ORB orb;

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

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

şi programul *CmmdcServer.java,* care înscrie în registrul ORB referinţele servantului. Activităţile ce trebuie întreprinse sunt declarate prin comentarii în textul sursă al programului

```
1  package 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       // Crease 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 servant
22       CmmdcImpl cmmdcImpl = new CmmdcImpl(orb);

24       // Obtinerea unei referinte pentru servant
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 şablon de programare adaptat exemplului tratat. În acest caz numele serviciului inregistrat în ORB va fi *Cmmdc-Service.* Evidenţa numelor serviciilor CORBA înregistrate în ORB este

făcută de serviciul *NameService.*

3. *module-info.java*

```
module tmp{
  requires java.corba;
  requires java.naming;
}
```

4. Realizarea programului client *CmmdcClient.java*:

```
package CmmdcApp;
import org.omg.CosNaming.NamingContextExtHelper;
import org.omg.CosNaming.NamingContextExt;
import org.omg.CORBA.ORB;
import java.util.Scanner;

public class CmmdcClient{
  static Cmmdc cmmdc;

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

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

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

      System.out.println("Obtained a handle on server object: " +
        cmmdc);
      Scanner scanner=new Scanner(System.in);
      long m,n;
      System.out.println("m=");
      m=scanner.nextLong();
      System.out.println("n=");
      n=scanner.nextLong();
      System.out.println("Cmmdc="+cmmdc.cmmdc(m,n));
    }
    catch (Exception e) {
      System.out.println("ERROR : " + e) ;
      e.printStackTrace(System.out);
    }
  }
}
```

Din nou clasa client conţine şablonul de accesare a unui serviciu CORBA.

5. *module-info.java*

```
module tmp{
```

```
2    requires java.corba;
3    requires java.naming;
4 }
```

Programele se compilează

- Sistemul de operare Windows
  `javac -d mods   CmmdcApp\*.java src\*.java`
- Sistemul de operare Linux
  `javac -d mods ./CmmdcApp/*.java src/*.java`

6. Pornirea serviciului de intregistrare a numelor cu programul `orbd.exe` din distribuţia *jdk.*

   - Sistemul de operare Windows
     `start orbd -ORBInitialHost localhost -ORBInitialPort 1050`
   - Sistemul de operare Linux
     `xterm -e orbd -ORBInitialPort 1050 -ORBInitialHost localhost &`

7. Pornirea programului server prin

```
java --module-path mods -m tmp/CmmdcApp.CmmdcServer
    -ORBInitialPort 1050 -ORBInitialHost localhost
```

8. Lansarea programului client prin

```
java --module-path mods -m client/CmmdcApp.CmmdcClient
    -ORBInitialPort 1050 -ORBInitialHost localhost
```

Model cu server persistent

Se consideră aceaşi interfaţă *Cmmdc.idl.*

Partea de server este alcătuită din clasa servant *CmmdcImpl.java* care implementează interfaţa *Cmmdc* -prezentat în secţiunea anterioară şi 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: Crease si initializare ORB
18       ORB orb = ORB.init(args, properties);

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

23       // Pas 3: Obtinerea unei referinte POA si
24       //        activarea gestionarului POAManager
25       // ******************
26       // Pas 3-1: Obtinerea 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 servantului cu PersistentPOA
40       persistentPOA.activate_object(cmmdcImpl);

42       // Pas 5: Fixarea contextului RootNaming si
43       //        legarea de  numele servantului
44       // Numele serviciului de nume : 'NameService'
45       // Numele servantului '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.getrMessage() );
38     }
39   }
40 }
```

Serverul trebuie să fie pe acelaşi calculator pe care rulează `orbd`. Executarea aplicaţiei presupune:

1. Pornirea serviciului de intregistrare a numelor cu programul `orbd.exe` din distribuţia *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 îndeplineşte:

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

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

Alte comenzi `servertool`

Comanda	Semnificaţie
`servertool> shutdown -serverid 257`	Oprirea serviciului
`servertool> unregister -serverid 257`	Ştergerea serviciului
`servertool> quit`	Închiderea utilitarului
`servertool> help`	

Observaţie 4.2.1 *Dacă se utilizează referinţa*

c:\Program Files\Java\jdk1.8.0_*\bin

în variabila de sistem `PATH` *(în loc de* `c:\Progra ~1\Java\jdk1.8.0_*\bin`*) atunci apelarea serviciilor* `orbd` *şi* `servertool` *se va face prin*

```
c:\Progra~1\Java\jdk1.8.0_*\bin\orbd -ORBInitialPort 1050 -serverPollingTime 200
c:\Progra~1\Java\jdk1.8.0_*\bin\servertool -ORBInitialPort 1050
```

3. Executarea clientului

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

Întrebări recapitulative

1. Precizaţi abrevierea RMI.
2. Care sunt componentele unei aplicaţii client-server bazat pe RMI?
3. Câte şi care sunt entităţile care participă la execuţia unei aplicaţii client-server bazat pe RMI?
4. Care este rolul îndeplinit de *(rmi)registry*?
5. Ce asigură şablonul de programare *Fabrica de obiecte* în RMI?
6. Ce asigură şablonul de programare *Callback* în RMI?
7. Explicaţi caracterul sincron al comunicării în RMI.
8. Care sunt activităţile pe care un server RMI le are de indeplinit ?
9. Care sunt componentele unei aplicaţii client server bazat pe CORBA?
10. Câte şi care sunt entităţile care participă la execuţia unei aplicaţii client-server bazat pe CORBA?
11. Care este ideea solutiei CORBA pentru asigurarea interoperabilităţii între client şi server realizate în limbaje de programare diferite.
12. Explicaţi caracterul sincron al comunicării în CORBA.
13. Ce oferă RMI-IIOP ?
14. Precizaţi modelele de aplicaţii CORBA prezentate în curs şi diferenţa dintre ele din punctul de vedere al execuţiei.

Probleme

1. Eliminarea mesajelor de avertisment.
2. IIOP, modelul persistent cu Java 9.

Capitolul 5

Mesaje în Java

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

Comunicaţiile asincrone[1] între programe permit realizarea unor sisteme de programe cu un grad mult mai scăzut de cuplare, în sensul că lansarea unei cereri şi recepţionarea rezultatului se pot executa în momente diferite de timp. Asemenea aplicaţii se pot realiza prin comunicaţii de mesaje, mesaje care sunt reţinute de un intermediar (serviciu de mesagerie, server, broker, messaging middleware).

5.1 Java Message Service (JMS)

JMS defineşte un cadru de programare Java (API) pentru realizarea aplicaţiilor slab cuplate.

Prezentarea se bazează pe interfaţa de programare (API) JMS-2 implementată de

- *Open Message Queue* implementarea de referinţă, Oracle.

Produsele *hornetq* (RedHat - jboss), *qpid* (apache) implementează de asemenea interfaţa JMS-2.

Pentru interfaţa de programare (API) JMS-1, există mai multe implementări JMS, dintre care amintim: *Open Message Queue* (Oracle), *ActiveMQ* (apache), *qpid* (apache).

[1] În accepţiunea din acest capitol.

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

O aplicaţie JMS este alcătuită din

- *un furnizor JMS (provider JMS)* : un sistem de mesagerie ce implementează specificaţiile JMS;
- *client JMS* : aplicaţie Java care trimite şi recepţionează mesaje;
- *mesaje* : obiecte utilizate în schimbul de informaţii de clienţi JMS;
- *obiecte administrator* - obiecte (resurse) create de administrator pentru a fi utilizate de clienţii JMS, precum
  - fabrica de conexiuni,
  - obiectele destinaţie pentru reţinerea mesajelor.

Aplicaţiile dezvoltate pe baza JMS 1/JMS 1.1 functionează şi în cazul utilizării unui furnizor de servicii realizat pentru JMS-2. În esenţă, JMS-2 simplifică programarea faţă de JMS-1.

Modele de comunicaţie:

- *Comunicaţii punctuale* : Un mesaj este generat de un producător (expeditor) şi 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.
- *Comunicaţii axate pe subiect (topic)* : Mesajele sunt depuse (publicate) în destinaţii 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 abonaţi pe un subiect pot fi
  - abonat simplu - are acces la mesajele emise după momentul abonării, pe durata conexiunii la furnizorul de servicii JMS;

  - abonat permanent - valabil şi după o întrerupere a conexiunii la furnizorul de servicii JMS;
  - abonat partajat - mai multi abonaţi utilizează aceaşi conexiune;
  - abonat partajat şi permanent.

Structura unui mesaj Un mesaj este alcătuit din

- *Antet (header)* : conţine informaţii pentru identificarea destinaţiei cât şi pentru identificarea mesajului.
- *Proprietăţi* : au caracter opţional şi sunt sub forma (nume, valoare). Proprietăţile ajută consumatorii să selecteze mesajele.
- *Corpul mesajului* : opţional. Potrivit specificaţiilor JMS există 6 tipuri de mesaje.
  - `javax.jms.Message` : mesaj fără corp;
  - `javax.jms.StreamMessage` : corpul mesajului conţine un flux Java de date de tip predefinit;
  - `javax.jms.ByteMessage` :
  - `javax.jms.MapMessage` : corpul mesajului conţine o familie de perechi (nume, valoare);
  - `javax.jms.TextMessage` : corpul mesajului conţine un string;
  - `javax.jms.ObjectMessage` : corpul mesajului conţine un obiect serializat.
- *Ataşament* (opţional).

5.2 *Open Message Queue 5*

Open Message Queue 5 este integrat în produsul *Glassfish,* o implementare JEE. În acelaşi timp *Glassfish* este şi un server de aplicaţii. Totodată este disponibilă o versiune de sine stătătoare.

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

- Se dezarhivează resursa descărcată.

- Se fixează atributele `IMQ_DEFAULT_JAVAHOME, IMQ_DEFAULT_VARHOME` în fişierul `etc\mq\imqenv.conf`.

Lansarea serviciului JMS. Orice aplicaţie JMS necesită funcţionarea serviciului JMS. Serviciul JMS se instalează prin

- *Windows*

```
set IMQ_JAVAHOME=. . .
set MQ_HOME=. . .
%MQ_HOME%\bin\imqbrokerd
```

- *Linux*

```
#!/bin/bash
MQ_HOME=. ..
export MQ_HOME
$MQ_HOME/bin/imqbrokerd
```

Funcţionarea corectă este indicată prin mesajul

imqbroker@hostname:7676 ready

Dacă se doreşte schimbarea portului atunci se foloseşte opţiunea `-port` *port*.

Pe un calculator pot coexista mai multe servicii JMS doar dacă folosesc porturi distincte şi 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 şi instalarea are efect odată cu repornirea calculatorului.

Compilarea şi executarea unui program necesită completarea variabilei sistem `classpath` cu fişierele `jms.jar` şi `imq.jar` iar dacă se va utiliza JNDI atunci va fi nevoie şi de `fscontext.jar`. Toate aceste fişiere se găsesc în catalogul `JMS_HOME/mq/lib`.

5.3 Elemente de programare - JMS-2

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

- Programat: obiectele administrator se instanţiază 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 fişiere de proprietăţi.

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 descendenţii:
   - `QueueConnectionFactory`
   - `TopicConnectionFactory`

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

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

   Pentru comunicaţii axate pe subiect, obiectul destinaţie, de tip `javax.jms.Topic` se instanţiază prin

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

2. Instanţierea 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 obţin producătorii de mesaje, de tip `javax.jms.JMSProducer,` dar şi consumatorii de mesaje, de tip `javax.jms.JMSConsumer.`

3. Instanţierea unui producător de mesaje

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

   Crearea şi 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*)

4. Interfaţa `javax.jms.Message` este implementată de clasele (`javax.jms.`) `BytesMessage, MapMessage, ObjectMessage, StreamMessage, TextMessage.`

   Un obiect care implementează interfaţa `javax.jms.Message` se obţine prin `ctx.createMessage(),` respectiv `ctx.create`*Tip*`Message`, unde $Tip \in$ { Bytes,Map,Object,Stream,Text}.

   Unui mesaj i se poate ataşa un atribut (cheie : String, valoare: T) unde T poate fi String, int, float, double, byte, boolean.

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

```
1  package queueprogramat;
2  import javax.jms.Queue;
3  import javax.jms.JMSContext;
4  import javax.jms.JMSProducer;

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

10   MsgSenderT(String queueName,int n){
11     this.queueName=queueName;
```

```
12      this.n=n;
13    }

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

25        for(int i=0;i<n;i++){
26            producer.send(q,"Hello "+i);
27        }
28        producer.send(q,ctx.createMessage());
29        ctx.close();
30      }
31      catch(Exception e){
32        System.out.println("JMSException : "+e.getMessage());
33      }
34      System.out.println("Sender finished");
35    }
36  }
```

5.3.2 Recepţia sincronă a unui mesaj

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

1. Generarea obiectelor administrator.
2. Instanţierea unui obiect de tip `javax.jms.JMSContext`.
3. Instanţierea unui consumator de mesaje

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

   Recepţia unui mesaj se obţine cu una din metode

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

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

```
1  package queueprogramat;
2  import javax.jms.TextMessage;
3  import javax.jms.Message;
4  import javax.jms.Queue;
5  import javax.jms.JMSContext;
6  import javax.jms.JMSConsumer;

8  public class  SyncMsgReceiverT extends Thread{
9    String queueName;

11   SyncMsgReceiverT(String queueName){
12     this.queueName=queueName;
13   }

15   public void run(){
16     try{
17       com.sun.messaging.QueueConnectionFactory cf=
18           new com.sun.messaging.QueueConnectionFactory();
19       //cf.setProperty("imqBrokerHostName","host");
20       //cf.setProperty("imqBrokerHostPort","7676");
21       Queue q=new com.sun.messaging.Queue(queueName);
22       JMSContext ctx=cf.createContext();
23       JMSConsumer consumer = ctx.createConsumer(q);
24       Message msg=null;
25       while((msg=consumer.receive())!=null){
26         if(msg instanceof TextMessage){
27           TextMessage m=(TextMessage)msg;
28           System.out.println(m.getText());
29         }
30         else
31           break;
32       }
33       ctx.close();
34     }
35     catch(Exception e){
36       System.out.println("JMSException : "+e.getMessage());
37     }
38     System.out.println("Consumer finished");
39   }
40 }
```

Exemplul 5.3.3 *Trimiterea şi recepţia mesajelor se face prin aplicaţia*

```
1  package queueprogramat;
2  class MsgHelloT{
3    public static void main(String[] args){
4      int n=3;
5      String queueName="MyQueue";
6      if(args.length>0)
7        queueName=args[0];
8      if(args.length>1)
9        n=Integer.parseInt(args[1]);
```

```
10      MsgSenderT sender=new MsgSenderT(queueName,n);
11      SyncMsgReceiverT receiver =new SyncMsgReceiverT(queueName);
12      receiver.start();
13      sender.start();
14    }
15  }
```

5.3.3 Recepţia asincronă a unui mesaj

Recepţia asincronă a mesajelor presupune implementarea interfeţei `MessageListener` ce conţine o singură metodă

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

care fixează prelucrarea mesajului.

Metoda `setMessageListener(MessageListener` *obj*`)` a clasei `JMSConsumer` fixează obiectul ce implementează interfaţa `Messagelistener.`

Astfel, caracterul asincron constă din faptul că mesajele sunt preluate de *ascultător* - adică obiectul ce implementează interfaţa `MessageListener` şi 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  package queueprogramat;
2  import javax.jms.MessageConsumer;
3  import javax.jms.Queue;
4  import javax.jms.JMSContext;
5  import javax.jms.JMSConsumer;

7  public class  AsyncMsgReceiverT extends Thread{
8    String queueName;

10   AsyncMsgReceiverT(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);

24       TextListener textListener=new TextListener();
25       consumer.setMessageListener(textListener);
26       textListener.run();
```

```
27        ctx.close();
28      }
29      catch(Exception e){
30        System.out.println(e.getMessage());
31      }
32      System.out.println("Consumer finished");
33    }
34  }
```

împreună cu *ascultătorul*

```
1  package queueprogramat;
2  import javax.jms.MessageListener;
3  import javax.jms.JMSException;
4  import javax.jms.TextMessage;
5  import javax.jms.Message;

7  public class  TextListener implements MessageListener{
8    boolean sfarsit=false;
9    public void onMessage(Message message){
10     if(message instanceof TextMessage){
11       TextMessage m=(TextMessage)message;
12       try{
13         String s=m.getText();
14         System.out.println(s);
15       }
16       catch(JMSException e){
17         System.out.println(e.getMessage());
18       }
19     }
20     else
21       sfarsit=true;
22   }

24   public void run(){
25     while(!sfarsit){
26       try{
27         Thread.sleep(1);
28       }
29       catch(InterruptedException e){}
30     };
31   }
32 }
```

5.3.4 Publicarea mesajelor

Publicarea mesajelor corespunzătoare unui subiect se face asemănător cu transmiterea mesajelor în comunicaţia punctuală, dar folosind instanţe ale claselor dedicate acestui tip de comunicaţie.

Exemplul 5.3.5

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

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

10   MsgPublisherT(String subiect,int n){
11     super();
12     this.n=n;
13     this.subiect=subiect;
14   }

16   public void run(){
17     try{
18       com.sun.messaging.TopicConnectionFactory cf=
19         new com.sun.messaging.TopicConnectionFactory();
20       //cf.setProperty("imqBrokerHostName","host");
21       //cf.setProperty("imqBrokerHostPort","7676");
22       Topic t=new com.sun.messaging.Topic(subiect);
23       JMSContext ctx=cf.createContext();
24       JMSProducer producer=ctx.createProducer();
25       for(int i=0;i<n;i++){
26           producer.send(t,"Hello "+i);
27       }
28       producer.send(t,ctx.createMessage());
29       ctx.close();
30     }
31     catch(Exception e){
32       System.out.println("JMSException : "+e.getMessage());
33     }
34     System.out.println("Publisher finished");
35   }
36 }
```

5.3.5 Abonare şi recepţia mesajelor

Dacă t este obiectul de tip `Topic`, clienţii 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 şi 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 primeşte doar mesajele publicate din momentul subscrierii.

Exemplul 5.3.6

```
1  package topicprogramat;
2  import javax.jms.TextMessage;
3  import javax.jms.Message;
4  import javax.jms.Topic;
5  import javax.jms.JMSContext;
6  import javax.jms.JMSConsumer;

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

13   MsgSubscriberT(String subiect,String clientID, String clientName){
14     this.subiect=subiect;
15     this.clientID=clientID;
16     this.clientName=clientName;
17   }

19   public void run(){
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(subiect);
26       JMSContext ctx=cf.createContext();
27       ctx.setClientID(clientID);
28       JMSConsumer consumer = ctx.createDurableConsumer(t,clientName);

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("JMSException : "+e.getMessage());
43     }
44     System.out.println("Subscriber finished");
45   }
46 }
```

Apelarea celor două activităţi se face prin

Exemplul 5.3.7

```
1  package topicprogramat;
2  class MsgPS{
3    public static void main(String[] args){
4      String subiect="JMS";
```

```
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      MsgSubscriberT[] abonat=new MsgSubscriberT[noAbonati];
12      publisher.start();
13      for(int i=0;i<noAbonati;i++){
14        abonat[i]=new MsgSubscriberT(subiect,"ID"+i,"id"+i);
15        abonat[i].start();
16      }
17    }
18  }
```

5.3.6 Cazul abonatului partajat

Mai mulţi clienţ JMS folosesc aceaşi conexiune iar abonamentul se identifică printr-un nume (`String`). Dintre aceşi clienţi doar unul recepţionează mesajele care sunt emise după momentul abonării.

Exemplul 5.3.8

Faţă de exemplul anterior se vor expedia mai multe mesaje de tip `Message`.

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

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

10   MsgPublisherT(String subiect,int n){
11     super();
12     this.n=n;
13     this.subiect=subiect;
14   }

16   public void run(){
17     try{
18       com.sun.messaging.TopicConnectionFactory cf=
19         new com.sun.messaging.TopicConnectionFactory();
20       //cf.setProperty("imqBrokerHostName","host");
21       //cf.setProperty("imqBrokerHostPort","7676");
22       Topic t=new com.sun.messaging.Topic(subiect);
23       JMSContext ctx=cf.createContext();
24       JMSProducer producer=ctx.createProducer();
25       for(int i=0;i<n;i++){
26         producer.send(t,"Hello "+i);
27       }
28       for(int i=0;i<n;i++)producer.send(t,ctx.createMessage());
```

```
29          ctx.close();
30        }
31        catch(Exception e){
32          System.out.println("JMSException : "+e.getMessage());
33        }
34        System.out.println("Publisher finished");
35      }
36  }
```

Pentru a folosi aceaşi conexiune recepţia mesajelor este programată în două clase :

- Se instanţiază o fabrică de conexiuni şi un număr de clienţi care vor utiliza aceaşi conexiune. Fiecare client este un fir de execuţie care este gestionat de o cuvă (*pool*) de fire de execuţie.

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

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

16   ReceiversT(int noAbonati, String subiect,
17       String sharedSubscriptionName){
18     this.subiect=subiect;
19     this.noAbonati=noAbonati;
20     this.sharedSubscriptionName=sharedSubscriptionName;
21   }

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

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

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

42     }
```

```
43      System.out.println("Receivers finished");
44    }
45  }
```

- Clienţii JMS propriu-zişi

```
1  package sharedprogramat;
2  import javax.jms.TextMessage;
3  import javax.jms.Message;
4  import javax.jms.Topic;
5  import javax.jms.JMSContext;
6  import javax.jms.JMSConsumer;

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

15   MsgSubscriberT(com.sun.messaging.TopicConnectionFactory cf,
16       String subiect,String sharedSubscriptionName,
17       String clientName){
18     this.cf=cf;
19     this.subiect=subiect;
20     this.sharedSubscriptionName=sharedSubscriptionName;
21     this.clientName=clientName;
22   }

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

31       Message msg=null;
32       while((msg=consumer.receive())!=null){
33         if(msg instanceof TextMessage){
34           TextMessage m=(TextMessage)msg;
35           System.out.println(clientName+" received : "+m.getText());
36         }
37         else
38           break;
39       }
40       ctx.close();
41     }
42     catch(Exception e){
43       System.out.println("MsgSubscriber : "+e.getMessage());
44     }
45     System.out.println("Subscriber "+clientName+" finished");
46   }
47 }
```

Aplicaţia este condusă de clasa

```
1  package sharedprogramat;
```

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

5.3.7 Obiecte administrator prin JNDI

Obiectele administrator, fabrica de conexiuni şi obiectul destinaţie, se precizează într-un fişier de proprietăţi - *jndi.properties.*

Pentru Open Message Queue acest fişier 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

5  # use the following property to configure the default connector

7  # register some queues in JNDI using the form
8  # queue.[jndiName] = [physicalName]
9  queue.queue = myqueue

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) şi a subiectului (topic). Aceaste obiecte se crează cu utilitarul *imqobjmgr* din distribuţia *Oracle-Open Message Queue.*

Obiectele se pot crea cu fişierul 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 şterge cu fişierul 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 aplicaţiile anterioare privind transmiterea / recepţia unui mesaj prin comunicaţie punctuală utilizând o coadă şi publicarea şi recepţionarea unui mesaj prin comunicaţie bazată pe subiect într-o variantă independententă de suportul *middleware* de serviciu de mesagerie folosit. Regăsirea resurselor gestionate de serviciul de mesagerie se va face utilizând JNDI.

5.3.8 Comunicaţia prin coadă - queue

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

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

Clasa *MsgSenderT*

```
1  package queuejndi;
2  import javax.jms.QueueConnectionFactory;
3  import javax.jms.Queue;
4  import javax.jms.JMSContext;
5  import javax.jms.JMSProducer;
6  import javax.naming.InitialContext;
7  import javax.naming.NamingException;
8  import java.util.Properties;
9  import java.io.IOException;

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

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

```
20    }

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

30        JMSContext jmsctx=cf.createContext();
31        JMSProducer producer=jmsctx.createProducer();
32        for(int i=0;i<n;i++){
33            producer.send(q,"Hello "+i);
34        }
35        producer.send(q,jmsctx.createMessage());
36        jmsctx.close();
37      }
38      catch(Exception e){
39        System.out.println("JMSException : "+e.getMessage());
40      }
41      System.out.println("Sender finished");
42    }

44    private void setupJNDI(){
45      Properties properties = new Properties();
46      try{
47        properties.load(this.getClass().getResourceAsStream("jndi.properties"));
48      }
49      catch(IOException e){
50        System.out.println("JNDI-PropertiesError : "+e.getMessage());
51      }
52      try{
53        ctx = new InitialContext(properties);
54      }
55      catch (NamingException e){
56        System.err.println("Error Setting up JNDI Context:" + e);
57      }
58    }

60    private void closeJNDI(){
61      try{
62        ctx.close();
63      }
64      catch (NamingException e){
65        System.err.println("Unable to close JNDI Context : " + e);
66      }
67    }
68  }
```

Clasa *SyncMsgReceiverT*

```
1  package queuejndi;
2  import javax.jms.QueueConnectionFactory;
3  import javax.jms.Queue;
4  import javax.jms.TextMessage;
5  import javax.jms.Message;
6  import javax.jms.JMSContext;
7  import javax.jms.JMSConsumer;
```

```
8  import javax.naming.InitialContext;
9  import javax.naming.NamingException;
10 import java.util.Properties;
11 import java.io.IOException;

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

18   SyncMsgReceiverT(String queueName){
19     QUEUE_JNDI_NAME=queueName;
20   }

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

30       JMSContext jmsctx=cf.createContext();
31       JMSConsumer consumer = jmsctx.createConsumer(q);
32       Message msg=null;
33       while((msg=consumer.receive())!=null){
34         if(msg instanceof TextMessage){
35           TextMessage m=(TextMessage)msg;
36           System.out.println(m.getText());
37         }
38         else
39           break;
40       }
41       jmsctx.close();
42     }
43     catch(Exception e){
44       System.out.println("JMSException : "+e.getMessage());
45     }
46     System.out.println("Consumer finished");
47   }

49   private void setupJNDI(){. . .}

51   private void closeJNDI(){. . .}
52 }
```

5.3.9 Comunicaţia pe bază de subiect - topic

Aplicaţia este alcătuită din clasele *MsgPS, MsgPublisherT, MsgSubscriberT*. Faţă de versiunea prezentată anterior se modifică doar clasele *MsgPublisherT* şi *MsgSubscriberT*.

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

```
1 package topicjndi;
```

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

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

17   MsgPublisherT(String subiect,int n){
18     this.n=n;
19     TOPIC_JNDI_NAME=subiect;
20   }

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

30       JMSContext jmsctx=cf.createContext();
31       JMSProducer producer=jmsctx.createProducer();
32       for(int i=0;i<n;i++){
33         producer.send(t,"Despre JMS"+" "+i);
34       }
35       producer.send(t,jmsctx.createMessage());
36       jmsctx.close();
37     }
38     catch(Exception e){
39       System.out.println("JMSException : "+e.getMessage());
40     }
41     System.out.println("Publisher finished");
42   }

44   private void setupJNDI(){. . .}

46   private void closeJNDI(){. . .}
47 }
```

Clasa *MsgSubscriberT*

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

```
11 import java.io.IOException;

13 public class  MsgSubscriberT extends Thread{
14   final String CONNECTION_JNDI_NAME = "ConnectionFactory";
15   String TOPIC_JNDI_NAME = "topic";
16   private InitialContext ctx;
17   private String clientID;
18   private String clientName;

20   MsgSubscriberT(String subiect,String clientID, String clientName){
21     TOPIC_JNDI_NAME=subiect;
22     this.clientID=clientID;
23     this.clientName=clientName;
24   }

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

37       Message msg=null;
38       while((msg=consumer.receive())!=null){
39         if(msg instanceof TextMessage){
40           TextMessage m=(TextMessage)msg;
41           System.out.println(clientName+" received : "+m.getText());
42         }
43         else
44           break;
45       }
46       jmsctx.close();
47     }
48     catch(Exception e){
49       System.out.println("JMSException : "+e.getMessage());
50     }
51     System.out.println("Subscriber finished");
52   }

54   private void setupJNDI(){. . .}

56   private void closeJNDI(){. . .}
57 }
```

5.3.10 Fluxuri prin mesaje

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

Exemplul 5.3.9 *Aplicaţie JMS cu clasă pentru publicarea şi consumarea unui mesaj care incorporează o imagine jpg.*

Clasa `MsgPublisherT`

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

12 public class  MsgPublisherT extends Thread{
13   String subiect;

15   MsgPublisherT(String subiect){
16     this.subiect=subiect;
17   }

19   public void run(){
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(subiect);
26       JMSContext ctx=cf.createContext();

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

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

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

43       ctx.close();
44     }
45     catch(Exception e){
46       System.out.println("JMSException : "+e.getMessage());
47     }
48     System.out.println("Publisher finished");
49   }
50 }
```

Clasa `MsgSubscriberT`

```
1  package msgimage;
2  import javax.jms.Message;
3  import javax.jms.Topic;
4  import javax.jms.JMSContext;
5  import javax.jms.JMSConsumer;
```

```
6  import javax.jms.StreamMessage;
7  import javax.jms.BytesMessage;
8  import java.awt.image.BufferedImage;
9  import javax.imageio.ImageIO;
10 import java.io.ByteArrayInputStream;
11 import java.io.InputStream;

13 public class  MsgSubscriberT extends Thread{
14   String subiect;
15   String clientID;
16   String clientName;

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

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

35       Message msg=null;
36       while((msg=consumer.receive())!=null){
37         if(msg instanceof StreamMessage){
38           StreamMessage m=(StreamMessage)msg;
39           m.reset();
40           byte[] bytes=(byte[])m.readObject();
41           System.out.println(clientName+" received : "+bytes.length);
42           InputStream in = new ByteArrayInputStream(bytes);
43           BufferedImage bi = ImageIO.read(in);
44           ShowImage showImage=new ShowImage(bi);
45           showImage.show();
46           break;
47         }
48       }
49       ctx.close();
50     }
51     catch(Exception e){
52       System.out.println("JMSException : "+e.getMessage());
53     }
54     System.out.println("Subscriber finished");
55   }
56 }
```

Clasa *ShowImage* (2.2.2) afişează imaginea pe monitor.

Producătorul şi consumatorului se lansează din

```
1  package msgimage;
2  class MsgPS{
3    public static void main(String[] args){
```

```
4       String subiect="streaming";
5       if(args.length>0)
6         subiect=args[0];
7       MsgSubscriberT abonat=new MsgSubscriberT(subiect,"myID","myName");
8       MsgPublisherT publisher=new MsgPublisherT(subiect);
9       abonat.start();

11      try{
12        Thread.sleep(1000);
13      }
14      catch(InterruptedException e){}

16      publisher.start();
17    }
18  }
```

5.3.11 Aplicaţie JMS slab cuplată

Posibilitatea de a crea obiecte *destinaţie* în mod dinamic permite realizarea unei aplicaţii în care clientul declară în momentul lansării unei solicitări destinaţia obiectului în care doreşte să primească rsspunsul. Clientul va prelua răspunsul într-un moment ulterior.

O asemenea aplicaţie este denumită aplicaţie *slab cuplată.*

Aplicaţia va fi alcătuită din trei clienţi:

- Client care preia şi 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  package cmmdc;
2  import javax.jms.TextMessage;
3  import javax.jms.Message;
4  import javax.jms.Topic;
5  import javax.jms.JMSContext;
6  import javax.jms.JMSConsumer;
7  import javax.jms.JMSProducer;

9  public class  MsgCmmdcServer{
```

```
11    public static void main(String[] args){
12      MsgCmmdcServer server=new MsgCmmdcServer();
13      server.service();
14    }

16    private void service(){
17      try{
18        com.sun.messaging.TopicConnectionFactory cf=
19          new com.sun.messaging.TopicConnectionFactory();
20        //cf.setProperty("imqBrokerHostName","host");
21        //cf.setProperty("imqBrokerHostPort","7676");
22        Topic t=new com.sun.messaging.Topic("Cmmdc");
23        JMSContext ctx=cf.createContext();
24        JMSConsumer consumer = ctx.createSharedConsumer(t,"Cmmdc");
25        JMSProducer producer = ctx.createProducer();
26        while(true){
27          Message msg=null;
28          while((msg=consumer.receive())!=null){
29            if(msg instanceof TextMessage){
30              TextMessage tm=(TextMessage)msg;
31              String s=tm.getText();
32              String[] ss=s.split(" ");
33              long m=Long.parseLong(ss[0]);
34              long n=Long.parseLong(ss[1]);
35              String topic=ss[2];
36              long c=cmmdc(m,n);
37              Topic t1=new com.sun.messaging.Topic(topic);
38              producer.send(t1,Long.valueOf(c).toString());
39              System.out.println("Server sent "+c+" to "+topic);
40            }
41          }
42        }

44      }
45      catch(Exception e){
46        System.out.println("JMSException : "+e.getMessage());
47      }
48    }

50    private long cmmdc(long m,long n){. . .}
51  }
```

Clasa *MsgClientSender*

```
1   package cmmdc;
2   import javax.jms.Topic;
3   import javax.jms.JMSContext;
4   import javax.jms.JMSProducer;
5   import javax.jms.JMSConsumer;
6   import java.util.Scanner;

8   public class  MsgClientSender{
9     private String msg,clientID,clientName,topicResult;

11    MsgClientSender(String clientID,String clientName){
12      this.clientID=clientID;
13      this.clientName=clientName;
14      Scanner scanner=new Scanner(System.in);
15      System.out.println("Introduceti m :");
```

```
16      long m=scanner.nextLong();
17      String sm=Long.valueOf(m).toString();
18      System.out.println("Introduceti n :");
19      long n=scanner.nextLong();
20      String sn=Long.valueOf(n).toString();
21      System.out.println("Introduceti 'Topic'-ul raspunsului");
22      topicResult=scanner.next();
23      msg=sm+" "+sn+" "+topicResult;
24    }

26    private void service(){
27      try{
28        com.sun.messaging.TopicConnectionFactory cf=
29          new com.sun.messaging.TopicConnectionFactory();
30        //cf.setProperty("imqBrokerHostName","host");
31        //cf.setProperty("imqBrokerHostPort","7676");
32        Topic t=new com.sun.messaging.Topic("Cmmdc");
33        JMSContext ctx=cf.createContext();

35        Topic t1=new com.sun.messaging.Topic(topicResult);
36        ctx.setClientID(clientID);
37        JMSConsumer consumer = ctx.createDurableConsumer(t1,clientName);
38        JMSProducer producer=ctx.createProducer();
39        producer.send(t,msg);
40        ctx.close();
41      }
42      catch(Exception e){
43        System.out.println("JMSException : "+e.getMessage());
44      }
45    }

47    public static void main(String[] args){
48      if(args.length<2){
49         System.out.println("Usage:");
50         System.out.println("java MsgClientSender clientID clientName");
51         System.exit(0);
52      }
53      MsgClientSender client=new MsgClientSender(args[0],args[1]);
54      client.service();
55    }
56  }
```

Clasa *MsgClientReceiver*

```
1   package cmmdc;
2   import javax.jms.TextMessage;
3   import javax.jms.Message;
4   import javax.jms.Topic;
5   import javax.jms.JMSContext;
6   import javax.jms.JMSConsumer;
7   import java.util.Scanner;

9   public class  MsgClientReceiver{
10    private String topicResult,clientID,clientName;

12    MsgClientReceiver(String clientID,String clientName){
13      this.clientID=clientID;
14      this.clientName=clientName;
15      Scanner scanner=new Scanner(System.in);
```

```
16      System.out.println("Introduceti 'Topic'-ul raspunsului");
17      topicResult=scanner.next();
18    }

20    public void service(){
21      try{
22        com.sun.messaging.TopicConnectionFactory cf=
23          new com.sun.messaging.TopicConnectionFactory();
24        //cf.setProperty("imqBrokerHostName","host");
25        //cf.setProperty("imqBrokerHostPort","7676");
26        Topic t=new com.sun.messaging.Topic(topicResult);
27        JMSContext ctx=cf.createContext();
28        ctx.setClientID(clientID);
29        JMSConsumer consumer = ctx.createDurableConsumer(t,clientName);
30        Message msg=null;
31        while((msg=consumer.receive())!=null){
32          if(msg instanceof TextMessage){
33            TextMessage m=(TextMessage)msg;
34            System.out.println("Cmmdc : "+m.getText());
35            break;
36          }
37        }
38        ctx.close();
39      }
40      catch(Exception e){
41        System.out.println("JMSException : "+e.getMessage());
42      }
43    }

45    public static void main(String[] args){
46      if(args.length<2){
47        System.out.println("Usage:");
48        System.out.println("java MsgSOAPClientReceiver clientID clientName");
49        System.exit(0);
50      }
51      MsgClientReceiver client=new MsgClientReceiver(args[0],args[1]);
52      client.service();
53    }
54  }
```

5.3.12 Programare JMS în JEE(*glassfish*)

Detalii privind serverul de aplicaţii sunt prezentate în secţiunea 6.4. Toate aplicaţiile dezvoltate anterior funcţionează fără nici o modificare. Bineînţeles, în prealabil trebuie pornit serverul de aplicaţie, prilej cu care se lansează şi furnizorul JMS.

Prezentăm o variantă de aplicaţie în care

- obiectele administrator,
  - fabrica de conexiuni,
  - obiectul / sursa destinaţie

se crează de administratorul *glassfish;*

1. Se lansează serverul de aplicaţii *glassfish*
   - Sistemul de operare Windows

     ```
     set GLASSFISH_HOME=. . .\glassfish*
     set PATH=%GLASSFISH_HOME%\bin;%PATH%
     asadmin start-domain domain1
     ```
   - Sistemul de operare Linux

     ```
     #!/bin/bash
     GLASSFISH_HOME=. . .
     $GLASSFISH_HOME/bin/asadmin start-domain domain1
     ```

   Lansarea serverului de aplicaţie 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 destinaţie cu datele:

`JNDI Name`	*myQueue*	*myTopic*
`Physical Name`	*myQueue*	*myTopic*
`Resource Type`	*javax.jms.Queue*	*javax.jms.Topic*

   - *Resources* → *JMS Resources* → *Connection Factories*
     Se crează câte o fabrică de conexiuni cu datele:

`JNDI Name`	*myQueueConnectionFactory*, respectiv *myTopicConnectionFactory*
`Resource Type`	*javax.jms.QueueConnectionFactory*, respectiv *javax.jms.TopicConnectionFactory*

- Aceste obiecte se injectează într-o clasă utilizînd 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 aplicaţie se arhivează cu `jar` cu indicarea clasei cu metoda `main` şi se lansează prin intermediul procedurii acoperitoare `glassfish\bin\appclient.bat`

  ```
  appclient -client numeArhiva.jar  [-targetserver host:3700]
  ```

Exemplul 5.3.11 *Aplicaţie cu comunicaţie 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;

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;

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       e.printStackTrace();
26     }
27     System.out.println("Sender finished");
28   }
29 }
```

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          e.printStackTrace();
32        }
33        System.out.println("Consumer finished");
34      }
35  }
```

Aplicaţia slab cuplată se obţine folosind doar două obiecte destinaţie *Topic*, o destinaţie pentru mesajele cu cereri şi una pentru mesajele cu răspunsuri. Expeditorul unei cereri specifică un identificator (String *messageId*) care este trecut de server ca atribut (*key : messageId*) al mesajului de răspuns. La recepţie, mesajul aşteptat se recunoaşte pe baza atributului.

Codurile aplicaţiei:

1. 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;
7  import javax.jms.JMSException;
8  import javax.jms.TopicConnectionFactory;
9  import javax.naming.InitialContext;
10 import javax.naming.NamingException;
11 import java.util.Properties;
12 import java.io.IOException;
13 import javax.annotation.Resource;

15 public class  MsgCmmdcServer{
16   final static String CONNECTION_JNDI_NAME =
17     "myTopicConnectionFactory";
18   final static String TOPIC_JNDI_NAME_1 = "cmmdc";
19   final static String TOPIC_JNDI_NAME_2 = "results";
20   private InitialContext ctx;

22   @Resource(lookup=CONNECTION_JNDI_NAME)
23   private static TopicConnectionFactory cf;
24   @Resource(lookup=TOPIC_JNDI_NAME_1)
25   private static Topic reqTopic;
26   @Resource(lookup=TOPIC_JNDI_NAME_2)
27   private static Topic resTopic;

29   public static void main(String[] args){
30     MsgCmmdcServer server=new MsgCmmdcServer();
31     server.service();
32   }

34   private void service(){
35     try{
36       JMSContext jmsctx=cf.createContext();
37       JMSConsumer consumer = jmsctx.createConsumer(reqTopic);
38       JMSProducer producer = jmsctx.createProducer();
```

```
39        Message msg=null;
40        while(true){
41          while((msg=consumer.receive())!=null){
42            if(msg instanceof TextMessage){
43              TextMessage tm=(TextMessage)msg;
44              String s=tm.getText();
45              System.out.println(s);
46              String[] ss=s.split(" ");
47              long m=Long.parseLong(ss[0]);
48              long n=Long.parseLong(ss[1]);
49              String messageId=ss[2];
50              long c=cmmdc(m,n);
51              tm=jmsctx.createTextMessage();
52              tm.setText(Long.valueOf(c).toString());
53              tm.setStringProperty("key",messageId);
54              producer.send(resTopic,tm);
55            }
56          }
57        }
58      }
59      catch(Exception e){
60        e.printStackTrace();
61      }
62    }

64    private long cmmdc(long m,long n){. . .}
65  }
```

2. 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;
6  import javax.jms.TopicConnectionFactory;
7  import javax.naming.InitialContext;
8  import javax.naming.NamingException;
9  import java.util.Properties;
10 import java.io.IOException;
11 import javax.annotation.Resource;

13 public class  MsgClientSender{
14   final static String CONNECTION_JNDI_NAME =
15     "myTopicConnectionFactory";
16   final static String TOPIC_JNDI_NAME_1 = "cmmdc";
17   final static String TOPIC_JNDI_NAME_2 = "results";
18   private String msg,clientID,clientName;
19   private InitialContext ctx;

21   @Resource(lookup=CONNECTION_JNDI_NAME)
22   private static TopicConnectionFactory cf;
23   @Resource(lookup=TOPIC_JNDI_NAME_1)
24   private static Topic reqTopic;
25   @Resource(lookup=TOPIC_JNDI_NAME_2)
26   private static Topic resTopic;

28   MsgClientSender(String clientID,String clientName){
```

```
29      this.clientID=clientID;
30      this.clientName=clientName;
31    }

33    private void service(){
34      Scanner scanner=new Scanner(System.in);
35      System.out.println("Introduceti m :");
36      long m=scanner.nextLong();
37      String sm=new Long(m).toString();
38      System.out.println("Introduceti n :");
39      long n=scanner.nextLong();
40      String sn=new Long(n).toString();
41      System.out.println("Introduceti 'messageId'-ul raspunsului");
42      String messageId=scanner.next();
43      msg=sm+" "+sn+" "+messageId;
44      try{
45        JMSContext jmsctx=cf.createContext();
46        jmsctx.setClientID(clientID);
47        JMSProducer producer=jmsctx.createProducer();
48        JMSConsumer consumer =
49          jmsctx.createDurableConsumer(resTopic,clientName);
50        producer.send(reqTopic,msg);
51        jmsctx.close();
52      }
53      catch(Exception e){
54        e.printStackTrace();
55      }
56    }

58    public static void main(String[] args){
59      if(args.length<2){
60        System.out.println("Usage:");
61        System.out.println("java MsgSOAPClientSender clientID clientName");
62        System.exit(0);
63      }
64      MsgClientSender client=new MsgClientSender(args[0],args[1]);
65      client.service();
66    }
67  }
```

3. 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;
7   import javax.jms.TopicConnectionFactory;
8   import javax.naming.InitialContext;
9   import javax.naming.NamingException;
10  import java.util.Properties;
11  import java.io.IOException;
12  import javax.annotation.Resource;

14  public class  MsgClientReceiver{
15    final static String CONNECTION_JNDI_NAME =
16      "myTopicConnectionFactory";
```

```
17    final static String TOPIC_JNDI_NAME_2 = "results";
18    private String messageId,clientID,clientName;
19    private InitialContext ctx;

21    @Resource(lookup=CONNECTION_JNDI_NAME)
22    private static TopicConnectionFactory cf;
23    @Resource(lookup=TOPIC_JNDI_NAME_2)
24    private static Topic resTopic;

26    MsgClientReceiver(String clientID,String clientName){
27      this.clientID=clientID;
28      this.clientName=clientName;
29    }

31    public void service(){
32      Scanner scanner=new Scanner(System.in);
33      System.out.println("Introduceti 'messageId'-ul raspunsului");
34      messageId=scanner.next();
35      try{
36        JMSContext jmsctx=cf.createContext();
37        jmsctx.setClientID(clientID);
38        JMSConsumer consumer =
39          jmsctx.createDurableConsumer(resTopic,clientName);
40        Message msg=null;
41        while((msg=consumer.receive())!=null){
42          if(msg instanceof TextMessage){
43            TextMessage m=(TextMessage)msg;
44            if(m.propertyExists("key")){
45              String msgId=m.getStringProperty("key");
46              if(msgId.equals(messageId)){
47                System.out.println("Cmmdc : "+m.getText());
48                break;
49              }
50            }
51          }
52        }
53        jmsctx.close();
54      }
55      catch(Exception e){
56        e.printStackTrace();
57      }
58    }

60    public static void main(String[] args){
61      if(args.length<2){
62        System.out.println("Usage:");
63        System.out.println("java MsgSOAPClientReceiver clientID clientName");
64        System.exit(0);
65      }
66      MsgClientReceiver client=new MsgClientReceiver(args[0],args[1]);
67      client.service();
68    }
69  }
```

Întrebări recapitulative

1. Explicaţi caracterul de cuplare slabă al unei aplicaţii bazat pe JMS.
2. Care este rolul furnizorului (provider) JMS?
3. Precizaţi şi explicaţi modelele de comunicaţii cu mesaje în JMS.
4. Precizaţi modelele de programare al unui client JMS prezentate în curs.

Probleme:

- Qpid cu JMS2

Partea II

TEHNOLOGII CU COMUNICAŢII PRIN INTERNET

Capitolul 6

HyperText Transfer Protocol

Protocolul `http - HyperText Transfer Protocol` (http) este destinat schimburilor de informaţii 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 latenţei prin utilizarea mai eficientă a reţelei. Diferenţa dintre HTTP/2 şi HTTP/1.1 este la nivelul de transport, adică a modului în care biţii sunt schimbaţi prin reţea. 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 informatiilor pe Web. Aceste limbaje derivă de asemenea din SGML.

6.1 Transacţie http

Protocolul http este de tip client-server:

- Navigatoarele *Google Chrome, Mozilla Firefox, Opera, Apple Safari* sunt aplicaţii client uzuale.
- Informaţiile sunt găzduite / generate de servere Web de către site-uri şi servicii Web. Exemple de servere Web sunt `apache HTTP Server,`

`Microsoft Internet Interchange Server.` O categorie aparte - esenţială pentru platforma Java - este dată de servere Web container de servlet şi Java Server Pages (JSP).

Transmiterea unui mesaj (cerere) conform protocolului http 1.1 constă din

1. Căutare *Domain Name System* (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 aşteaptă răspunsul la cererea emisă;
5. Se recepţionează şi se încarcă mesajul de răspuns;
6. Închiderea conexiunii.

Acest ciclu de acţiuni se numeşte tranzacţie http. Paşi 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 reţine informaţii între două tranzacţii http. Acest comportament se exprimă prin terminologia: protocolul http este *fără stare - stateless.*

Transmisia datelor se realizează utilizând de obicei protocolul TCP.

Referinţele resurselor se indică folosind URI / URL.

Cererile şi 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 - opţional.

Preambulul unei cereri conţine:

1. numele metodei {`GET, POST, PUT, DELETE, HEAD, CONNECT, TRACE, OPTIONS`};

2. referinţa resursei (URL);

3. Versiunea protocolului http.

Preambulul unui răspuns conţine:

1. Versiunea protocolului http;

2. codul răspunsului: număr natural format din trei cifre cu semnificaţiile

Categoria	Indică
1**	mesaj de informare
2**	mesaj de succes
3**	redirectare către alt URL
4**	eroare in 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 defineşte o paletă largă de atribute. Exemple de antete sunt date în tabelul următor:

Antet	Semnificaţie
	Exemplu
`host`	Gazda şi 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 înţelegerea conţinutului
	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 conţinutul este imagine, cod binar, etc., atunci acesta este codificat în text.

Codul *Multipurpose Internet Mail Exchange* (MIME) precizează natura conţinutului 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 evidentă 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 formularul html folosit[1].

Pentru metoda GET cererea http este

```
GET /apphello/hello?name=mk&tip=text%2Fhtml HTTP/1.1
Host: localhost:8080
Connection: keep-alive
```

[1] Mulţimea antetelor depinde de navigatorul utilizat.

```
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

```
<html>
  <head><title> Servlet-ul Hello</title></head>
  <body>
    <center>
    <h1> Pagina de apelare a servletului HelloServlet </h1>
    <form method="post"
          action="http://localhost:8080/apphello/hello">
      <p>Introduceti numele:
      <input type="text" name="name" size=20>
      <p>
      <input type="submit" value="Apeleaza">
      <input type="hidden" name="tip" value="text/html" >
    </form>
    </center>
  </body>
</htm
```

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 obţinute mai sus - după care recepţionează răspunsul dat de servlet.

Codul clasei Java

```
import java.net.Socket;
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 şi un fişier JSP *Java Server Pages.* Despre un asemenea server Web se spune că este container de servlet şi 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.

- *apache-tomee* este construit peste *apache-tomcat* şi integrează o serie de tehnologii JEE
- *weblogic* (Oracle)
- *jboss application server* (RedHat)

Serverul *Apache HTTP Server* şi *Windows Internet Information Server* nu sunt servere Web containere de servlet şi JSP.

6.3 Serverele Web *apache-tomcat* şi *jetty*

Serverul Web *apache-tomcat,* (pe scurt *tomcat*) este distribuit gratuit şi poate fi descărcat pornind de la adresa `www.apache.org`.

Instalarea serverului în mediul Windows revine la dezarhivarea fişierului 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 distribuţia Java utilizată.

Pachetul conţine cataloagele[2]: `bin, common, conf, logs, server, shared, temp, webapps, work`.

Serverul se lansează prin comanda

`TOMCAT_HOME\bin\startup`

şi se opreşte cu comanda

`TOMCAT_HOME\bin\shutdown`

sau, simplu, `Ctrl-C`

Din catalogul `TOMCAT_HOME`, lansarea se poate obţine cu ajutorul fişierului de comenzi

- Sistemul de operare Windows

```
set CATALINA_HOME=. . .
set JAVA_HOME=. . .
bin\startup
```

- Sistemul de operare Linux

[2]Numele şi numărul cataloagelor este dependent de distribuţia *apache-tomcat.*

```
#!/bin/bash
CATALINA_HOME=~/JavaApp/apache-tomcat-8.5.5
$CATALINA_HOME/bin/startup.sh
```

Opţional se poate instala / activa *managerul / administatorul* serverului Web *tomcat.*

Verificarea funcţionării serverului Web *tomcat* se face apelând dintr-un navigator pagina *http://host:port* unde

- *host* este numele calculatorului pe care rulează *tomcat;*
- Portul implicit este 8080.

Reuşita este ilustrată de imaginea motanului

Toate aplicaţiile se depun în catalogul `TOMCAT_HOME\webapps.`

Asemănător se procedează şi 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 aplicaţiile se depun în catalogul `\jetty-distribution-*\webapps.`

Transport Layer Security în *tomcat*

Transmisia utilizând protocolul *Transport Layer Security* - (TLS) care extinde *Secure Socket Layer* -(SSL) presupune criptarea datelor care circulă între client şi server.

Realizarea presupune:

1. Generarea unui certificat de securitate cu utilitarul `keytool` din distribuţia Java, de exemplu

```
keytool -genkey -alias tomcat -keyalg RSA
  -keystore {cale}/tomcatKeystore.jks
  -dname "cn=XYZ, ou=cs, o=unitbv, l=brasov, c=RO"
  -keypass 1q2w3e -storepass 1q2w3e
```

Parametrul `keystore` fixează locaţia şi denumirea fişierului certificatului de securitate.

Se definesc două parole

- `keypass` parola certificatului de securitate;
- `storepass` parola de protecţie a locaţiei certificatului de securitate.

Parametrul *cale* desemnează calea către catalogul unde în care se crează fişierul *tomcatKeystore.jks* - certificatul de securitate. Uzual, certificatul de securitate se mută în catalogul `TOMCAT_HOME\conf`.

2. Modificarea fişierului `apache-tomcat-*\conf\server.xml`

   - Se decomentează / modifică elementul

   ```
   <Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
              maxThreads="150" SSLEnabled="true">
       <SSLHostConfig>
            <Certificate certificateKeystoreFile="conf/localhost-rsa.jks"
                 type="RSA" />
       </SSLHostConfig>
   </Connector>
   ```

   unde *tomcatKeystore.jks* înlocuieşte *localhost-rsa.jks*

   - Elementul `Certificate` se completează cu atributele

   ```
   certificateKeystorePassword="1q2w3e"
   certificateKeyAlias="tomcat"
   ```

3. Serverul Web se poate apela prin

```
https://localhost:8443
http://localhost:8080
```

Transport Layer Security în *jetty*

Transmisia utilizând *Secure Socket Layer* - SSL (mai nou *Transport Layer Security* - TLS) înseamnă criptarea datelor care circulă între client şi server.

Realizarea presupune

1. Generarea unui certificat de securitate cu utilitarul `keytool` din distribuţia Java, de exemplu

```
keytool -genkey -alias jetty -keyalg RSA
  -keystore {cale}/keystore
  -dname "cn=SE, ou=cs, o=unitbv, l=brasov, c=RO"
  -keypass 1q2w3e -storepass 1q2w3e
```

Se definesc două parole

- `keypass` parola certificatului de securitate;
- `storepass` parola de protecţie a locaţiei certificatului de securitate.

Parametrul *cale* desemnează calea către catalogul unde în care se crează fişierul *keystore* - certificatul de securitate. Certificatul de securitate se mută în catalogul `JETTY_HOME\etc`.

2. Criptarea parolelor

```
java -cp %JETTY_HOME%\lib\jetty-util-*.jar
  org.eclipse.jetty.util.security.Password <parola>
```

Pentru exemplul certificatului generat mai sus, pentru *parola=1q2w3e* rezultatul este *OBF:1irv1lml1mii1mmc1lj51iur*

3. Completarea fişierului %JETTY_HOME%/start.ini cu secvenţa

```
# Module: https
--module=https

# Module: ssl
--module=ssl
jetty.ssl.host=0.0.0.0
jetty.sslContext.securePort=8443
jetty.sslContext.keyStorePath=etc/keystore
jetty.sslContext.trustStorePath=etc/keystore
jetty.sslContext.keyStorePassword=OBF:1irv1lml1mii1mmc1lj51iur
jetty.sslContext.trustStorePassword=OBF:1irv1lml1mii1mmc1lj51iur
jetty.sslContext.keyManagerPassword=OBF:1irv1lml1mii1mmc1lj51iur
jetty.sslContext.trustStoreType=JKS
```

4. După lansarea serverului Web apelarea poate fi

```
https://localhost:8443
http://localhost:8080
```

6.4 Glassfish

Instalarea şi lansarea serverului. Instalarea constă din dezarhivarea arhivei descărcate.

Este recomandat includerea în fişierul `GLASSFISH_HOME\grassfish\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 aplicaţiile 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 obţine prin
`asadmin create-domain --adminport 4848 --domaindir` *dirDomeniu numeDomeniu*

Ştergerea domeniului se obţine prin
`asadmin delete-domain --domaindir` *dirDomeniu numeDomeniu*

Lansarea şi oprirea serverului *glassfish* se comandă prin
`asadmin start-domain --domaindir` *dirDomeniu numeDomeniu*
`asadmin stop-domain --domaindir` *dirDomeniu numeDomeniu*

Un server de aplicaţii integrează o serie de tehnologii ca suport pentru aplicaţii. Astfel *glassfish* integrează java Message Service (JMS), WebSocket, servlet, Java Server Pages (JSP), Java Server Faces (JSF), Context and Dependency Injection (CDI), Java API for XML Wev Services (JAX-WS), Java API for XML Restful Services (JAX-RS).

Produsul *payara* (`www.paraya.fish`) se dezvoltă pornind de la *glassfish.*

Capitolul 7

Conexiune simplă prin clase din `jdk`

Pachetul `java.net` oferă, prin intermediul claselor `URL, URLConnection, HttpURLConnection, HttpsURLConnection, JarURLConnection` o posibilitate elementară de acces a unor resurse din Internet.

Clasa `javafx.scene.web.WebView` permite vizualizarea paginilor Web.

7.1 Clasa `java.net.URL`

Clasa `java.net.URL` permite realizarea unei conexiuni cu un server Web[1] 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 referinţă URL validă.

Metode

- `InputStream openStream() throws IOException`

  Returnează fluxul de intrare pentru citirea resursei.

- `URLConnection openConnection() throws IOException`

  Returnează o instanţă a conexiunii cu obiectul definit de URL.

[1] apache-tomcat, apache, Micrsoft IIS (Internet Information Services), etc.

Exemplul 7.1.1 *Pe baza referinţei către un fişier html dintr-un catalog vizibil al unui server Web, să se afişeze conţinutul fişierului.*

În codul reprodus mai jos, fişierul *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 }
```

7.2 Clasa `javafx.scene.web.WebView`

```
1  import javafx.scene.web.WebView;
2  import javafx.scene.web.WebEngine;
3  import javafx.application.Application;
4  import javafx.scene.Scene;
5  import javafx.geometry.HPos;
6  import javafx.geometry.VPos;
7  import javafx.scene.paint.Color;
8  import javafx.stage.Stage;
9  import javafx.scene.layout.Region;
```

```
11 public class Wv extends Application {
12   private Scene scene;

14   public static void main(String[] args) {
15     Application.launch(Wv.class, args);
16   }

18   @Override
19   public void start(Stage primaryStage) {
20     primaryStage.setTitle("Show Web Page");
21     scene = new Scene(new Browser(), 600,500, Color.LIGHTGREEN);

23     primaryStage.setScene(scene);
24     primaryStage.show();
25   }
26 }

28 class Browser extends Region {
29   final WebView webView = new WebView();
30   final WebEngine webEngine = webView.getEngine();

32   public Browser(){
33     webEngine.load("http://www.unitbv.ro");
34     getChildren().add(webView);
35   }
36 }
```

Capitolul 8

Servlet

Printre aplicaţiile distribuite de tip client-server, în care comunicaţiile se bazează pe protocolul *http,* se disting:

- *Aplicaţii Web (site):* Cererea adresată serverului este lansată uzual de o persoană utilizând un program navigator: *Google Chrome, Mozilla Firefox, Opera, Apple Safari, etc.*

- *Servicii Web:* Cererea către server se face de un program. Aplicaţia server şi client se programează utilizând interfeţe de programare specifice.

Componenta server a unei aplicaţii Web

- conţine o clasă Java care se execută de un server Web, compatibil;

- este gestionată de serverul Web;

- este capabilă să recepţioneze şi să răpundă cererilor formulate de clienţi.

Structura minimală a unei aplicaţii Web este

```
catalogAplicatiei
|--> WEB-INF
|    |--> classes
|    |    |    *.class
|    |--> lib
|    |    |    *.jar
|    index.html
```

Catalogul `classes` conţine fişierele *class* ale aplicaţiei Web.

Catalogul `lib` este opţional şi va conţine resursele *jar* suplimentare cerute de clasele aplicaţiei Web.

Prin intermediul fişierului *index.html* se apelează aplicaţia Web. Acest fişier este totodată şi client Web. Adresa de apelare (URL - *Universal Resource Locator*) a aplicaţiei Web este

`http://`*host:port/catalogAplicaţieiWeb*

Dacă în loc de *index* fişierul `html` de apelare are alt nume, de exemplu *xyz.html* atunci adresa de apelare va fi

`http://`*host:port/catalogAplicaţieiWeb/xyz.html*

host este numele calculatorului pe care rulează serverul Web - *gazda* aplicaţiei Web. Portul implicit utilizat de un server Web container de servlet este 8080.

Catalogul aplicaţiei este denumit *context*-ul aplicaţiei Web.

Programarea şi utilizarea unei aplicaţii Web necesită:

- Cunoaşterea elementului (marcaj, *tag*) `html` `<form>` pentru realizarea formularelor de introducere a datelor;

- Utilizarea unui server Web, container de servleţi. Dintre produsele gratuite amintim: *apache-tomcat, jetty, glassfish.*

8.1 Marcajul <form>

Într-un document html introducerea datelor se poate obţine utilizând marcajul <form> ...</form> .

Atribute ale marcajului <form> . Reamintim că atributele se prezintă ca perechi (nume, valoare) şi se scriu în antetul marcajului sub forma nume = valoare.

Nume	Valoare	Semnificaţia valorii
`action`	adresa tip URL	Resursa care prelucrează formularul, cel puţin */context/numeApel*
`method`	`GET`	Mesajul trimis serverului Web conţine după adresa URL numele şi valorile parametrilor introduşi. Adăugarea se face potrivit sintaxei ?numeParam1=valoare&numeParam2=valoare... Lungimea mesajului nu poate depăşi 255 caractere.
	`POST`	Transmisia datelor se face în fluxuri de date. Permite transferul unor fişiere de pe maşina clientului pe maşina serverului.
`id`		Parametru de identificare a formularului (opţional).
`name`		Nume atribuit formularului (opţional).
`onSubmit`		Metodă JavaScript executată înaintea apelării serverului Web (opţional).

În **conţinutul marcajului** <form> putem include elemente de control prin marcajele

- <`input`>
- <`option`>
- <`select`>
- <`textarea`>

Atributele marcajului < `input`> sunt:

Nume	Valoare	Semnificaţie
`type`	`text`	Se aşteaptă introducerea unui text
	`number`	Se aşteaptă introducerea unui număr întreg
	`date`	Se aşteaptă introducerea unei date calendaristice
	`password`	Se aşteaptă introducerea unei parole
	`submit`	Se marchează sfârşitul completării formularului
	`reset`	Se reiniţializează formularul
	`file`	Permite selectarea unui fişier
	`hidden`	Transmite mai departe un atribut fără vizualizarea lui
`name`		numele controlului
`value`		valoarea (iniţială) a controlului
`size`		numărul caracterelor ataşat 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

Pe platforma de programare Java, servlet-ul este componenta care stă la baza majorităţii cadrelor de dezvoltare (*framework*) pentru aplicaţii şi servicii Web.[1]

Termenul servlet se utilizează atât pentru aplicaţie cât şi pentru clasa gestionată de serverul Web.

Interfaţa de programare (API) pentru servlet nu face parte din `JDK`, fiind implementat de fiecare producător de server Web container de servlet.

Apelarea servlet-ului se poate face:

- din meniul *File/Open* al unui navigator;
- ca referinţă într-un document `html`

$$<\texttt{a href="}\textit{URL-servlet}\texttt{"}>\ldots</\texttt{a}>$$

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, *urlPattern*. Sintaxa utilizată este

$$/numeleAplicaţieiWeb/urlPattern$$

Plasând fişierul de apelare al aplicaţiei servlet într-un server Web (apache-httpd, Microsoft-IIS) şi aplicaţia servlet într-un servlet Web compatibil, servletul se va apela prin

$$\texttt{http://}host/\ldots/xyz.html$$

iar clasa servlet se va apela din *xyz.html*.

Legătura serverului Web cu clasa servlet-ului se poate realiza

[1] *vert.x, Play, dukescript* sunt cadre de lucru care dezvoltă în Java aplicaţii Web fără să se bazeze pe servlet.

- programat – prin adnotări în codul servlet-ului;
- descriptiv – în catalogul `WEB-INF` se editează fişierul `web.xml`.

  În versiunile anterioare ale interfeţei de programare servlet aceasta a fost unica opţiune.

Trebuie demarcată diferenţa dintre apelarea / lansarea în execuţie a clasei servlet de apelarea aplicaţiei 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 fişierul `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`> defineşte 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 fişier `web.xml` este

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <servlet>
     <servlet-name>nume_servlet_1</servlet-name>
     <servlet-class>Nume_clasa</servlet-class>
  </servlet>
  <servlet>
     <servlet-name>nume_servlet_2</servlet-name>
     <servlet-class>Nume_clasa</servlet-class>
  </servlet>
  . . .
  <servlet-mapping>
     <servlet-name>nume_servlet_1</servlet-name>
     <url-pattern>/numeApel_1</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
     <servlet-name>nume_servlet_2</servlet-name>
     <url-pattern>/numeApel_2</url-pattern>
  </servlet-mapping>
  . . .
</web-app>
```

Unui element <servlet> îi pot corespunde mai multe elemente <servlet-mapping>, prin utilizarea de *numeApel* diferite.

Opţional `web.xml` poate conţine elementul

```
<welcome-file-list>
  <welcome-file>
    fisier.html sau jsp
  </welcome-file>
</welcome-file-list>
```

cu precizarea fişierelor html sau jsp care apelează aplicaţia Web. Declaraţia fişierului *index.html* este implicită.

Compilarea clasei servlet necesită completarea variabilei de mediu `classpath` cu fişierul `TOMCAT_HOME\lib\servlet-api.jar`.

Odată completată structura de cataloage şi fişiere ale aplicaţiei servlet această structură trebuie copiată în catalogul `TOMCAT_HOME\webapps`. Această operaţie se numeşte **desfăşurarea** (*deployment*) sau instalarea servlet-ului. Copierea se poate executa şi cu serverul Web pornit.

Pentru instalarea unui servlet există mai multe alternative:

- Din catalogul *catalogAppServlet* se realizează arhiva *catalogAppServlet.war*

  `jar cfv` *catAppServlet.war* `WEB-INF\* index.html`

  care se copiază în catalogul *TOMCAT_HOME\webapps.*

  Serverul Web *tomcat* va dezarhiveaza arhiva. Astfel servlet-ul este instalat.

  Această instalare se numeşte *instalare dinamică - hot deployment.*

  Dacă fişierul `war` este creat, atunci în locul copierii, instalarea se poate face prin componenta *manager* a lui *tomcat.*

- O aplicaţie servlet arhivată `war` se poate instala de la distaţă prin produsele:

  – *apache-tomcat-deployer*

  – *cargo*

8.2.1 Codul unui servlet

Un servlet implementează interfaţa `Servlet` sau extinde una din clasele `GenericServlet` sau `HttpServlet`. `GenericServlet` implementează interfaţa `Servlet`, iar `HttpServlet` extinde clasa `GenericServlet`. Extinzând clasa

`GenericServlet` nu este nevoie de rescrierea tuturor metodelor abstracte ale interfeţei `Servlet`.

Metodele interfeţei `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 clasa servlet va fi o clasă care extinde clasa `HttpServlet`. În locul metodei `service(...)`, programatorul suprascrie metodele `doGet(...)` sau `doPost(...)`, în funcţie de metoda utilizată de client la lansarea cererii.

Practic, un servlet constă din scrierea metodelor

- `void init(ServletConfig` *config*`)`

  Această metodă este opţională.

```
public void init(ServletConfig config) throws ServletException{
  super.init(config);
  // cod de initializare
}
```

  Obiectul *config* are o metodă `String getInitParameter(String` *numeParam*`)` cu ajutorul căreia se pot recupera parametri de initializare asociaţi servlet-ului şi care se dau fie prin adnotarea `WebInitParam` prin şablonul

```
import javax.servlet.annotation.WebInitParam;

@WebServlet(urlPatterns = "/numeApel",
  initParams = {
```

```
    @WebInitParam(name = "numeParam", value = "valoareParam"),
    . . .
  }
)
```

fie în fişierul `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>).

Activităţile 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 obţine o referinţa 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 interfeţei `HttpServletRequest`:
   `String getParameter(String` *numeParametru*`)`
   `java.util.Enumeration getParameterNames()`

   Adiţional se pot afla

   - calculatorul cu serverul web: `getServerName()`;
   - portul: `getServerPort()`;
   - catalogul servlet-ului: `getContextPath()`.

4. Rezolvă cererea clientului;

5. Formează şi *scrie* răspunsul;

6. Închide conexiunea obiectului prin care s-a realizat transmisia datelor către navigatorul clientului prin `out.close()`.

Un câmp (global) declarat în clasa servletului este comun fiecărei instanţe 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 acţiune.

Ciclul de viaţă 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 execuţie 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>Introduceti 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 aplicaţiei.
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 fişierul *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 </web-app>
```

Compilarea şi 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
```

şi se utilizează fişierul *build.xml*

```
1 <project basedir="." default="generate.war">
2   <property name="TOMCAT_HOME" value=". . ."/>
3   <property name="dist.name" value="apphelloP" />
4   <property name="dist.dir" value="dist" />

6   <path id="myclasspath">
7     <fileset dir="web/WEB-INF/lib">
8       <include name="*.jar"/>
9     </fileset>
10    <pathelement path="${TOMCAT_HOME}/lib/servlet-api.jar" />
11  </path>

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>

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>

27  <target name="generate.war" depends="compile">
28    <jar destfile="${dist.dir}/${dist.name}.war" basedir="web" />
29  </target>
30 </project>
```

Observaţie. Valoarea parametrului *dist.name* defineşte catalogul aplicaţiei (contextul) *catalogAppServlet.*

Se lansează în execuţie serverul Web *tomcat*, se desfăşoară servlet-ul în serverul Web şi 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 fişierul de invocare *index.html* primeşte *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 execuţie. Firul de execuţie este creat şi lansat de serverul Web.

Interfaţa de programare Servlet 3.0 oferă posibilitatea unui execuţii asincrone: satisfacerea cererii clientului se face într-un fir de execuţie lansat de clasa servlet-ului. Terminarea activităţii clasei servlet nu mai este legată de rezolvarea cererii clientului şi de trimiterea răspunsului.

Acest mod de execuţie se indică prin adnotarea

`@WebServlet(urlPaterns="/`*numeApel*`" ,asyncSupported=true)`

Suportul asincron necesită un *context,* obiect de tip `javax.servlet.Async`

`Context`. Acest obiect se obţine prin

- `AsyncContext asyncCtx=req.startAsync();`
- `AsyncContext asyncCtx=req.startAsync(req,res);`

Activităţile ce trebuie îndeplinite pentru satisfacerea cererii clientului se lansează într-un fir de execuţie prin

```
asyncCtx.start(Runnable fir);
```

Metoda `void complete()` a clasei `javax.servlet.AsyncContext` finalizează îndeplinirea activităţii asincrone.

Interfaţa `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, depăşirii timpului alocat sau producerii unei erori.

Exemplul 8.3.1 *Varianta asincronă a servlet-ului de calcul a celui mare divizor comun.*

Codul servlet-ului (*AsyncServlet*) este independent de activităţile ce satisfac cererea clientului. Acele activităţi sunt programate în clasa *AsyncAction* prin λ-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.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.ServletRequest;
5  import javax.servlet.ServletResponse;
6  import java.io.IOException;
7  import java.util.function.BiFunction;
8  import java.util.function.Function;

10 class AsyncAction{
11   static public long cmmdc(long a,long b){
12     BiFunction<Long,Long,Long> f=(m,n)->{
13       long r,c;
14       do{
15         c=n;
16         r=m % n;
17         m=n;
18         n=r;
19       }
20       while(r!=0);
21       return Long.valueOf(c);
22     };
23     return f.apply(a,b).longValue();
24   }

26   static public Thread service(AsyncContext ac){
27     Function<AsyncContext,Thread> f=(asyncCtx)->{
28       return new Thread(()->{
29         ServletRequest req=asyncCtx.getRequest();
30         ServletResponse res=asyncCtx.getResponse();
31         String sm=req.getParameter("m");
32         String sn=req.getParameter("n");
33         long m=Long.parseLong(sm),n=Long.parseLong(sn);
34         long x=cmmdc(m,n);
35         String result=Long.valueOf(x).toString();
36         System.out.println(x);
37         try{
38           PrintWriter out=res.getWriter();
39           String title="Cmmdc Servlet";
40           res.setContentType("text/html");
41           out.println("<HTML><HEAD><TITLE>");
42           out.println(title);
43           out.println("</TITLE></HEAD><BODY>");
```

```
44              out.println("<H1>"+title+"</H1>");
45              out.println("<P>Cmmdc is "+x);
46              out.println("</BODY></HTML>");
47              out.close();
48            }
49            catch(IOException e){
50              System.out.println(e.getMessage());
51            }
52            asyncCtx.complete();
53          });
54      };
55      return f.apply(ac);
56    }
57  }
```

- Clasa *MyAsyncListener* implementează interfaţa `AsyncListener`

```
1  package listeners;
2  import javax.servlet.AsyncEvent;
3  import javax.servlet.AsyncListener;
4  import javax.servlet.ServletRequest;

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 aplicaţie *glassfish* oferă facilităţi pentru implementarea aplicaţiei servlet:

- Firul de execuţie va aparţine unui *bazin* de fire de execuţie al serverului de aplicaţie. Există un bazin iniţial (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 execuţie 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  }
```

- Acţiunea firului de execuţie poate fi inclusă într-o componentă Java care va fi instanţiată şi injectată de serverul de aplicaţie.

  Injectarea este indicată prin

```
import javax.inject.Inject;

@Inject
private CmmdcAction obj;
```

  Clasa *CmmdcAction* nu conţine decât metoda *cmmdc* din codul de mai sus.

  În plus este nevoie fişierul *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 referinţă a interfeţei de programare (API) *servlet-api 3.1* este conţinută în produsul `glassfish4`.

8.4.1 Procesare asincronă neblocantă

Diferenţa majoră constă în faptul că solicitarea clientului este trimisă nemijlocit şi 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*`)`

Interfaţa `javax.servlet.ReadListener` declară metodele

- `void onDataAvailable() throws IOException`
- `void onAllDataRead() throws IOException)`
- `void onError(Throwable t)`

Clasa care implementează această interfaţă satisface cererea clientului.

Se va mai utiliza interfaţa `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 javax.servlet.AsyncContext;
3  import javax.servlet.ServletException;
4  import javax.servlet.ServletInputStream;
5  import javax.servlet.ServletOutputStream;
6  import javax.servlet.annotation.WebServlet;
7  import javax.servlet.http.HttpServlet;
8  import javax.servlet.http.HttpServletRequest;
9  import javax.servlet.http.HttpServletResponse;

11 @WebServlet(urlPatterns = {"/nonblock"}, asyncSupported=true)
12 public class NonBlockingServlet extends HttpServlet {
13   protected void doGet(HttpServletRequest req, HttpServletResponse res)
```

```
14          throws ServletException, IOException {
15      res.setContentType("text/html;charset=UTF-8");
16      ServletInputStream input = req.getInputStream();
17      ServletOutputStream output = res.getOutputStream();
18      AsyncContext context = req.startAsync();
19      CmmdcListener listener=new CmmdcListener(input, context, output);
20    }

22    @Override
23    protected void doPost(HttpServletRequest req, HttpServletResponse res)
24            throws ServletException, IOException {
25      doGet(req, res);
26    }
27  }
```

împreună cu clasa care îndeplineşte 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{
10   private ServletInputStream input = null;
11   private AsyncContext context = null;
12   private ServletOutputStream out = null;
13   private boolean readFinished=false;
14   private String rez=null;
15   private String data=null;

17   public CmmdcListener(ServletInputStream in, AsyncContext ac,
18       ServletOutputStream output) throws IOException{
19     input = in;
20     context = ac;
21     out = output;
22           in.setReadListener(this);
23     out.setWriteListener(this);
24   }

26   @Override
27   public void onDataAvailable() {
28     try {
29       int len = 0;
30       byte b[] = new byte[1024];
31       StringBuffer sb=new StringBuffer(1024);
32       while (input.isReady() && (len>-1)) {
33         len=input.read(b);
34         if(len>0) sb.append(new String(b,0,len));
35       }
36       data = sb.toString();
37       System.out.println(data);
38     }
39     catch (IOException e) {
40       System.out.println("onAvailableException : "+e.getMessage());
41     }
42   }
```

```
44    @Override
45    public void onAllDataRead() throws IOException{
46      System.out.println("onAllDataRead");
47      readFinished=true;
48      System.out.println(data);
49      rez=solver(data);
50      context.complete();
51      onWritePossible();
52    }

54    @Override
55    public void onWritePossible() throws IOException {
56      while(!readFinished);
57      StringBuffer sb=new StringBuffer(1024);
58      sb.append("<html><head><title>Servlet ReadTestServlet</title></head>");
59      sb.append("<body><h1>Servlet NonBlockingServlet</h1><p>");
60      sb.append(rez);
61      sb.append("</p></body></html>");
62      out.print(sb.toString());
63    }

65    @Override
66    public void onError(Throwable t) {
67      t.printStackTrace();
68      context.complete();
69    }

71    private String solver(String data){
72      String[] s=data.split("&");
73      String[] s0=s[0].split("=");
74      long m=Long.parseLong(s0[1]);
75      String[] s1=s[1].split("=");
76      long n=Long.parseLong(s1[1]);
77      String r=Long.valueOf(cmmdc(m,n)).toString();
78      return "Cmmdc : "+r;
79    }

81    private long cmmdc(long m, long n){. . .}
82  }
```

8.4.2 Modificarea protocolului `http`: *upgrade*

Schema este utilizată deja în protocolului `websocket`.

În apelul `http` trebuie inserat antetul (*header*) `Upgrade`. Odată aceast antet recunoscut, solicitarea clientului este rezolvată într-o clasă care implementează interfaţa `javax.servlet.http.HttpUpgradeHandler`.

Interfaţa declară metodele:

- `void init(WebConnection` *wc*`)`
- `void destroy()`

Interfaţa `javax.servlet.http.WebConnection`

`Metode`

- `ServletInputStream getInputStream() throws IOException`
- `ServletOutputStream getOutputStream() throws IOException`

Clasa care implementează interfaţa `HttpUpgradeHandler` se declară în servlet prin metoda `upgrade` a interfeţei `HttpServletRequest`

`<T extends HttpUpgradeHandler> T upgrade(Class<T>` *handlerClass*`) throws IOException, ServletException`

În principiu clasa servlet-ului este independentă de specificul aplicaţiei.

```
1  package cmmdc;
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;

8  @WebServlet(urlPatterns = {"/upgrade"})

10 public class UpgradeServlet extends HttpServlet{
11   private static final long serialVersionUID = 3834711619672506478L;

13   public void doGet(HttpServletRequest req,HttpServletResponse res)
14       throws ServletException,IOException{
15     if ("distrJava".equals(req.getHeader("Upgrade"))){
16       res.setStatus(101);
17       res.setHeader("Upgrade", "distrJava");
18       res.setHeader("Connection", "Upgrade");
19       res.flushBuffer();
20       System.out.println("Upgrade OK "+req.getHeader("Upgrade"));
21       MyHttpUpgradeHandler handler = req.upgrade(MyHttpUpgradeHandler.class);
22     }
23     else{
24       System.out.println("No upgrade: ");
25     }
26   }

28   public void doPost(HttpServletRequest req,HttpServletResponse res)
29       throws ServletException,IOException{
30     doGet(req,res);
31   }
32 }
```

Pentru problema calculului celui mai mare divizor comun a două numere naturale clasa *MyHttpUpgradeHandler* este

```
1  package cmmdc;
2  import javax.servlet.ServletOutputStream;
3  import javax.servlet.ServletInputStream;
4  import javax.servlet.http.WebConnection;
5  import javax.servlet.http.HttpUpgradeHandler;
```

```
7  public class MyHttpUpgradeHandler implements HttpUpgradeHandler {
8    private WebConnection wc = null;

10   @Override
11   public void init(WebConnection wc) {
12     this.wc=wc;
13     try{
14       ServletInputStream input = wc.getInputStream();
15       ServletOutputStream output=wc.getOutputStream();
16       String CRLF = "\r\n";
17       String resStr = "distrJava/1.0 " + CRLF;
18       resStr += "Server: Glassfish/ServerTest" + CRLF;
19       resStr += "Content-Type: text/html" + CRLF;
20       resStr += "Connection: Upgrade" + CRLF;
21       resStr += CRLF;
22       byte[] b=new byte[256];
23       input.read(b);
24       String data=new String(b).trim();
25       String rez=solver(data);
26       resStr +=rez + CRLF;
27       output.write(resStr.getBytes());
28       output.flush();
29     }
30     catch(Exception ex) {
31         throw new RuntimeException(ex);
32     }
33   }

35   @Override
36   public void destroy() {
37     try{
38       wc.close();
39     }
40     catch (Exception ex) {
41       System.out.println("Destroy wc Exception : "+ex.getMessage());
42     }
43   }

45   private String solver(String data){
46     System.out.println("Solver : "+data);
47     String[] s=data.split(" ");
48     String r="-1";
49     try{
50       long m=Long.parseLong(s[0]);
51       long n=Long.parseLong(s[1]);
52       r=Long.valueOf(cmmdc(m,n)).toString();
53     }
54     catch(NumberFormatException e){
55       System.out.println("NumberFormatException : "+e.getMessage());
56     }
57     return "Cmmdc : "+r;
58   }

60   private long cmmdc(long m, long n){. . .}
61 }
```

Problema care rămâne de rezolvat este inserarea antetului `Update`. Re-

zolvarea consta în generarea unui mesaj `http` care se va transmite printr-un soclu TCP. Mesajul `http` foloseşte metoda `post,` iar datele din corpul mesajului sunt preluate prin interfaţa `HttpInputStream.`

Lansarea aplicaţiei se face dintr-un program client, în care se generează mesajul `http` cu anterul `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: distrJava" + 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  package cmmdc;
2  import java.io.BufferedReader;
3  import java.io.IOException;
4  import java.io.InputStream;
5  import java.io.InputStreamReader;
6  import java.io.OutputStream;
7  import java.io.PrintWriter;
8  import java.net.Socket;

10 import javax.servlet.ServletException;
11 import javax.servlet.annotation.WebServlet;
12 import javax.servlet.http.HttpServlet;
13 import javax.servlet.http.HttpServletRequest;
14 import javax.servlet.http.HttpServletResponse;

16 @WebServlet(urlPatterns = {"/client"})
17 public class ClientServlet extends HttpServlet {
18   private static final long serialVersionUID = 3473283187293545302L;

20   protected void doGet(HttpServletRequest req,HttpServletResponse res)
21           throws ServletException, IOException {
22     String m=req.getParameter("m");
23     String n=req.getParameter("n");

25     res.setContentType("text/html;charset=UTF-8");
26     PrintWriter out = res.getWriter();

28     final String CRLF = "\r\n";
29     final String host = req.getServerName();// "localhost";
30     final int port = req.getServerPort(); // 8080;
31     final String contextRoot = "/upgrade";
32     final String data = m+" "+n;
33     InputStream input = null;
34     OutputStream output = null;
35     BufferedReader reader = null;
36     Socket s = null;
37     try{
38       out.println("<html>");
39       out.println("<head>");
40       out.println("<title>Servlet ClientTest</title>");
41       out.println("</head>");
42       out.println("<body>");
43       out.println("<h1>Http Upgrade Process</h1>");
```

```
45        // Setting the HTTP upgrade req header
46        String reqStr = "POST " + contextRoot + "/upgrade HTTP/1.1" + CRLF;
47        reqStr += "User-Agent: Java/1.7" + CRLF;
48        reqStr += "Host: " + host + ":" + port + CRLF;
49        //reqStr += "Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2" + CRLF;
50        reqStr += "Upgrade: distrJava" + CRLF;
51        reqStr += "Connection: Upgrade" + CRLF;
52        reqStr += "Content-type: application/x-www-form-urlencoded" + CRLF;
53        //reqStr += "Transfer-Encoding: chunked" + CRLF;
54        reqStr += CRLF;
55        reqStr += data + CRLF;

57        s = new Socket(host, port);
58        input = s.getInputStream();
59        output = s.getOutputStream();
60        output.write(reqStr.getBytes());
61        output.flush();
62        reader = new BufferedReader(new InputStreamReader(input));
63        // Reading the res, and displaying the header from server
64        printHeader(reader, out);
65        // Reading the res, and displaying the header from server
66        printHeader(reader, out);

68        // Reading the echo data
69        String dataOutput;
70        if ((dataOutput = reader.readLine()) != null) {
71          // Print out the data after header
72          out.println("</br>" + dataOutput + "</br>");
73        }
74        out.flush();
75        out.println("</body>");
76        out.println("</html>");
77      }
78      catch (IOException e) {
79        System.out.println("ClientServlet Exception : "+e.getMessage());
80      }
81      finally{
82        if (reader != null) {
83            reader.close();
84        }
85        if (output != null) {
86            output.close();
87        }
88        if (input != null) {
89            input.close();
90        }
91        if (s != null) {
92            s.close();
93        }
94      }
95    }

97    protected void printHeader(BufferedReader reader, PrintWriter out)
98        throws IOException {
99      for(String line; (line = reader.readLine()) != null;) {
100       if(line.isEmpty()){
101         break;
102       }
103       out.println(line + "</br>");
```

```
104       }
105     }

107     @Override
108     protected void doPost(HttpServletRequest req,HttpServletResponse res)
109       throws ServletException, IOException {
110       doGet(req, res);
111     }
112   }
```

8.5 Dezvoltări în *servlet-api 4.0*

Implementarea interfeţei de programare *servlet-api 4.0* funcţionează în `glassfish5`.

8.5.1 Accelerare prin *Server push*

Server push este o metodă care asigură transferul anticipat al unor resurse (imagini, fişiere CSS, JavaScript, etc) către client înaintea terminării prelucrării cererii. Atunci când clientul va recepţiona răspunsul toate resursele necesare paginii web de răspuns se vor afla în *cache*-ul navigatorului, gata pentru a fi utilizate.

Astfel creşte viteza de încărcare a paginii Web de răspuns.

Metoda solicită protocolul *Transfer Layer Security* (TLS), prin `https`.

Transferul anticipat are la baza interfaţa `javax.servlet.http.PushBuilder`. O instanţă care implementează interfaţa se obţine prin intermediul obiectului `request`

```
request.getPushBuilder()
```

Acţiunile care se intreprind sunt:

1. `path(String` *resursaDeTransferat*`)`
2. `push()`

Exemplul 8.5.1 *Servlet pentru vizualizarea unui fişier gif animat.*

Structura aplicaţiei este

```
push
|--> resources
|    |    walking_santa.gif
|--> WEB-INF
|    |--> classes
|    |    |    PushGif.class
|    image.html
|    index.html
```

unde

- *PushGif.java*

```
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet("/push")
public class PushGif extends HttpServlet {

  @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse res)
      throws ServletException, IOException {
    req.newPushBuilder()
      .path("resources/walking_santa.gif")
      .push();
    getServletContext().getRequestDispatcher("/image.html")
      .forward(req, res);
  }

  public void doPost(HttpServletRequest req,HttpServletResponse res)
     throws ServletException,IOException{
    doGet(req,res);
  }
}
```

- *image.html*

```
<html>
<head>
  <title>Servlet 4.0 ServerPush Example</title>
</head>
<body>
  <img src='resources/walking_santa.gif'>
</body>
</htm
```

- *index.html*

```
<html>
  <head>
    <title> Servlet-ul Push </title>
  </head>
  <body>
    <center>
    <h1> Pagina de apelare a servletului PushServlet </h1>
    <form method="post"
          action="push">
      <input type="submit" value="Push">
    </form>
    </center>
  </body>
</html>
```

8.6 Facilităţi de programare cu servlet

8.6.1 Program client al unui servlet

Apelarea unui servlet dintr-un program Java – adică lansarea unei cereri şi recepţionarea răspunsului furnizat de servlet se poate obţine

- sincron cu
  - *apache : httpcomponents-client*
  - *jetty*
  - modulul *jdk.incubator.httpclient* din Java 9.
- asincron cu
  - *apache : httpcomponents-client*
  - *apache httpcomponents-asyncclient*

Caracterul sincron / asincron constă în faptul că recepţia şi 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* conţine, printre altele, fişierele *httpcore-*.jar, httpclient-*.jar, commons-logging-*.jar.*

Pentru compilare trebuie declarată în variabila de sistem `classpath` referinţa către *httpclient-*.jar* şi *httpcore-*.jar* dar pentru execuţie este nevoie şi de referinţa 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 soluţie 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` aparţin pachetului `org.apache.http.client.methods`.

3. Lansarea cererii.

```
CloseableHttpResponse response=httpclient.execute(httpget);
```

respectiv

```
CloseableHttpResponse 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 8.6.1 *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  }
```

În cazul utilizării protocolului `https` - care presupune utilizarea criptării prin SSL / TSL, metoda `main` are la început secvenţa de cod

```
SSLContext sslcontext = SSLContexts.custom()
   .loadTrustMaterial(new File("tomcatKeystore.jks"), "1q2w3e".toCharArray(),
   new TrustSelfSignedStrategy())
   .build();
  // Allow TLSv1 protocol only
SSLConnectionSocketFactory sslsf = new SSLConnectionSocketFactory(
   sslcontext,
   new String[] { "TLSv1" },
   null,
   SSLConnectionSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER);
CloseableHttpClient httpclient = HttpClients.custom()
  .setSSLSocketFactory(sslsf)
  .build();
```

Varianta *fluentă* a programului client anterior

```
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")
```

```
            .build())
          .execute().returnContent().asString();
        System.out.println("Cmmdc = "+result);
      }
      catch(Exception e){
        System.out.println("Exception : "+e.getMessage());
      }

      System.out.println("Get method request");
      String data="?m="+m+"&n="+n+"&tip=text/plain";
      try{
        String result=Request.Get(uri+data)
          .execute().returnContent().asString();
        System.out.println("Cmmdc = "+result);
      }
      catch(Exception e){
        System.out.println("Exception : "+e.getMessage());
      }
    }
}
```

O soluţie elementară bazată pe clasa `java.net.HttpURLConnection` este

```
import java.net.URL;
import java.net.HttpURLConnection;
import java.util.Scanner;
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.PrintWriter;

public class CmmdcClient{
  public static void main(String[] args){
    Scanner scanner=new Scanner(System.in);
    System.out.println("m=");
    long m=scanner.nextLong();
    System.out.println("n=");
    long n=scanner.nextLong();
    String msg="m="+m+"&n="+n+"&tip=text/plain";
    System.out.println("HttpURLConnection cu metoda GET");
    try{
      String urlGET="http://localhost:8080/myservlet/cmmdc?"+msg;
      URL url=new URL(urlGET);
      HttpURLConnection conn=(HttpURLConnection)url.openConnection();
      conn.setRequestMethod("GET");
      System.out.println(conn.getResponseCode());
      System.out.println(conn.getResponseMessage());
      BufferedReader br=
         new BufferedReader(new InputStreamReader(conn.getInputStream()));
      String s;
      while((s=br.readLine())!=null){
        System.out.println(s);
      }
      br.close();
      conn.disconnect();
    }
    catch(Exception e){
      System.out.println(e.getMessage());
    }
    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());

33        // Varianta 1
34        /*
35        Future<Content> future=async.execute(request);
36        while(!future.isDone()){;}
37        System.out.println("Cmmdc = "+future.get().asString());
38        */

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()){;}

56        threadpool.shutdown();
57      }
58      catch(Exception e){
59        System.out.println("Exception : "+e.getMessage());
60      }
61    }
62  }
```

2. *httpcomponent-asyncclient*

   Variabila `classpath` conţine referenţele către fişerele `jar` aflate în catalogul `lib` din *httpcomponent-asyncclient.*

   Programarea constă din

   (a) Crearea unei instanţe 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 instanţă de tip `HttpAsyncRequestProducer` se obţine prin

`HttpAsyncMethods.createGet(uri)`

Drept instanţă a clasei `HttpAsyncResponseConsumer` poate fi o clasă ce extinde clasa `org.apache.http.nio.client.methods.AsyncCharConsumer`.

Exemplul 8.6.2

```
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 ioctrl) 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  }
```

Experimental Java 9 introduce o interfaţă de programare pentru clienţi.

```
1  package client;
2  import jdk.incubator.http.HttpClient;
3  import jdk.incubator.http.HttpRequest;
4  import jdk.incubator.http.HttpRequest.BodyProcessor;
5  import jdk.incubator.http.HttpResponse;
6  import java.net.URI;
7  import java.net.URISyntaxException;
8  import java.util.Scanner;

10 public class MyHttpClient{
11   public static void main(String[] args){
12     Scanner scanner=new Scanner(System.in);
13     System.out.println("m=");
14     String m=scanner.nextLine().trim();
15     System.out.println("n=");
16     String n=scanner.nextLine().trim();
17     String tip="text/plain";
```

```
18      String params="?m="+m+"&n="+n+"&tip="+tip;

20      URI uri=null;
21      try{
22        uri=new URI("http://localhost:8080/myservlet/cmmdc"+params);
23      }
24      catch(URISyntaxException e){
25        e.printStackTrace();
26      }
27      HttpClient client = HttpClient.newBuilder()
28       .followRedirects(HttpClient.Redirect.ALWAYS)
29       .build();
30      System.out.println(client.version());
31      HttpRequest request = HttpRequest.newBuilder(uri)
32       .GET()
33       .build();
34      try{
35        HttpResponse<String> response =
36         client.send(request, HttpResponse.BodyHandler.asString());
37        System.out.println(response.statusCode());
38        System.out.println(response.body());
39      }
40      catch(Exception e){
41        e.printStackTrace();
42      }
43    }
44  }
```

```
1  module client{
2    requires jdk.incubator.httpclient;
3  }
```

8.6.2 Servlete înlănţuite

Un servlet apelează la un moment dat alt servlet. Şablonul de lucru este

```
RequestDispatcher dispatcher = getServletContext()
  .getRequestDispatcher("/url_pattern_servlet_apelat");
if(dispatcher!=null)
  dispatcher.include(request,response);
```

Exemplul 8.6.3 *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 }
```

8.6.3 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 reţine informaţiile referitoare la client pe perioada conexiunii respective. Perioada de timp cât un client este în conexiune cu o pagină Web se numeşte sesiune.

Există posibilitatea păstrării unor informaţii pe durata unei sesiuni prin intermediul unui obiect de tip `javax.servlet.http.HttpSession`.

Înaintea satisfacerii unei cereri, servlet-ul verifică existenţa 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 reţine 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 obţine 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 8.6.4 *Exemplul următor numără de câte ori se apelează servletul într-o sesiune. Se defineşte un atribut noAcces, care la prima apelare este iniţializat iar apoi este mărit cu câte o unitate la fiecare nouă apelare a servletului.*

```
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=Integer.valueOf(1);
22       mesaj="Salut !";
23     }
24     else{
25       contor=Integer.valueOf(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 aceastei 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>
```

8.6.4 Cookie

Un *Cookie* este un fişier de dimensiune mică trimis de către programul server clientului ca parte a header-ului Http.

Acesta conţine informaţii despre sesiunea curentă care salvate 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 sterse automat în momentul expirării.

Unii clienţi nu permit memorarea cookie-urilor. În acest caz, clientul este informat că acest fapt ar putea duce la imposibilitatea satisfacerii cereri sale / accesării paginii Web. Implicit, durata de viaţă a unui cookie este sesiunea curentă a navigatorului (până se închide navigatorul).

Clasa `Cookie`

Contructor

`Cookie(String` *nume*`, String` *valoare*`)`

Metode

- `void setDomain(String` *model*`)`

  Domeniul este o adresă URL ce restricţionează accesul cookie-urilor la acel domeniu. *model* trebuie să conţină cel puţin două caractere ".".

- `void setMaxAge(int` *durată*`)` Fixează durata de existenţă 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*`)`

Recunoaşterea 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 8.6.5 *În exemplul următor se numără de câte ori se apelează servletul pe durata de viaţă 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",Integer.valueOf(contor).toString());
39      }
40      else{
41         myCookie=new Cookie("urmarire", Integer.valueOf(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 aceastei pagini este "+contor);
50      out.println("</body></html>");
51      out.close();
52   }

54   public void doPost(HttpServletRequest  req,
55                      HttpServletResponse res)
56   throws ServletException,IOException{
57      doGet(req,res);
58   }
59 }
```

8.6.5 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.

Şablonul 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 8.6.6 *Un servlet va executa sarcini la intervale de n (dat ca parametru) secunde. Câte un buton permite declanşarea, oprirea şi aflarea starii curente.*

Programarea unor activităţi care se vor repeta se poate realiza utilizând clasele `java.util.Timer` şi `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 defineşte într-o clasă ce extinde `java.util.TimerTask`. Programatorul trebuie să realizeze metoda `void run()`.

În acest exemplu, sarcina constă în reţinerea 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>
```

8.6.6 Autentificare

Serverul Web *apache-tomcat* poate executa autentificările *basic* şi *digest.* Autenticarea se face pe baza perechii (nume_utilizator, parola) (username,password). Aceste informaţii sunt reţinute în serverul Web, în fişierul `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 cerinţa de autentificare solicitată unui client se precizează în fişierul `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>
```

Opţional 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 autentificarii *basic* se utilizează codarea `base64`.

8.6.7 Servlet cu conexiune la o bază de date

Folosind Anexa H considerăm

Exemplul 8.6.7 *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 inexistenta "+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 aplicaţiei presupune:

1. Realizarea bazei de date:
2. Testarea servlet-ului:
   (a) Pornirea serverului bazei de date.
   (b) Se verifică prezenţa în catalogul servlet-ului `...\WEB-INF\lib` a fişierului `derbyclient.jar` sau `mysql-connector-java-*-bin.jar`.

(c) Pornirea serverului *tomcat* sau reîncărcarea servlet-ului.

(d) Apelarea servlet-ului din pagina Web.

8.6.8 Imagini furnizate de servlet

Indicăm două modalităţi prin care un client obţine o imagine furnizată de un servlet. Imaginea poate proveni dintr-un fişier extern sau poate fi creată de servlet.

- Imaginea este transmisă direct clientului în fluxul de ieşire de tip `ServletOutputStream`, tipul MIME al răspunsului fiind

  `response.setContentType("image/`ext`");`

  $ext \in \{gif, jpg, png, \ldots\}$.

  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     String pathApp=
22       req.getSession().getServletContext().getRealPath("/")+fs;
23     Path path=Paths.get(pathApp+"walking_santa.gif");
24     try{
25       res.setContentType("image/gif");
26       Files.copy(path,out);
27     }
28     catch(Exception e){
29       res.setContentType("text/plain");
30       System.out.println(e.getMessage());
31       out.println("Cererea d-voastra nu poate fi satisfacuta");
32     }
```

```
33      out.close();
34    }
35  }
```

Liniile de cod 21-23 determină calea absolută până la fişierul grafic.

- Pe calculatorul serverului Web, imaginea se salvează într-un fişier grafic, după care servlet-ul scrie în fluxul de ieşire un document *html* cu o legătură (link) către fişierul cu imaginea creată anterior. Navigatorul clientului va descărca şi 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.Font;
13 import java.awt.image.BufferedImage;
14 import java.awt.Color;
15 import java.awt.Font;
16 import java.awt.Graphics2D;

18 @WebServlet(urlPatterns = "/graphjpg")

20 public class MyGraphP extends HttpServlet {

22   public void doGet(HttpServletRequest req,HttpServletResponse res)
23       throws ServletException, IOException {
24     String fs=System.getProperty("file.separator");
25     res.setContentType("text/html");
26     PrintWriter out=res.getWriter();
27     String fileRef=
28       req.getSession().getServletContext().getRealPath("/")+fs;
29     String numeFis="desen";
30     String ext="jpg";  // sau "png"
31     Graphics2D g=null;
32     BufferedImage image=null;
33     try{
34       image=new BufferedImage(800, 60, BufferedImage.TYPE_BYTE_INDEXED);
35       g=image.createGraphics();
36       g.setColor(Color.WHITE);
37       g.fillRect(0, 0, 800, 60);
38       g.setPaint(Color.BLUE);
39       // Fixarea fontului
40       Font font = new Font("Serif", Font.ITALIC, 48);
41       g.setFont(font);
42       // Editarea unui text
43       g.drawString("Tehnologii distribuite in Java!", 10, 50);
```

```
45          // Salvarea imaginii intr-un fisier jpg sau png

47          File f=new File(fileRef+numeFis+"."+ext);
48          ImageIO.write(image,"jpg",f);

50          // Raspunsul catre client
51          out.println("<HTML><BODY>");
52          out.println("<h2>Imagine preluata de pe server </h2>");
53          out.println("<p><a href=\"http://localhost:8080/myservlet/"+
54            numeFis+"."+ext+"\">");
55          out.println("Vizualizarea imaginii </a>");
56          out.println("</BODY></HTML>");
57          out.close();
58        }
59        finally {
60          // Eliberarea resurselor
61          if (g != null) g.dispose();
62          //if (frame != null) frame.removeNotify();
63        }
64      }
65  }
```

8.6.9 Servlet cu RMI

Un servlet poate fi client al unei aplicaţii RMI. Interfaţa la diatanţă se depune în catalogul `WEB-INF\lib` al servlet-ului. Apelarea programului server RMI se face prin

```
import java.rmi.Naming;
. . .
InterfataDistanta obj=(InterfataDistanta)
    Naming.lookup("//"+host+":"+port+"/NumeServiciuRMI");
```

Exemplul 8.6.8 *Client servlet pentru aplicaţia 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;

11 @WebServlet(urlPatterns = "/servletrmi")
12 public class ServletRMI extends HttpServlet {

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  }
```

8.6.10 Servlet cu JMS

De data aceasta cererea clientului se rezolva asincron:

- In prima fază se apelează un servlet care apelează un server JMS. Soluţia este reţinută de furnizorul serviciului de mesagerie pe o destinaţie cu un subiect furnizat de client. Pentru regăsirea rezultatului, servlet-ul crează clientului un abonament durabil, funcţie de numele subiectului.

- În faza a doua, clientul apelează un alt servlet, a cărei funcţie este preluarea rezultatului. Clientul trebuie sa furnizeze subiectul destinaţiei rezultatului.

Exemplul 8.6.9

Serverului JMS al aplicaţiei este dat de clasa *MsgCmmdcServer* prezentat în 5.3.11.

Codul servlet-ului care transmite datele problemei serverului JMS

```
1  package 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 fişierele *jar* ale serviciului JMS folosit.

8.6.11 Servlet cu jurnalizare

Exemplul 8.6.10 *Servlet cu jurnalizare. Fişierul log se va afla în catalogul aplicaţiei şi 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;

13 import javax.servlet.annotation.WebServlet;

15 @WebServlet(urlPatterns = "/logging")

17 public class LoggerServlet extends HttpServlet {
18   private static Logger logger=Logger.getLogger("logtest.LoggerServlet");

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   }

31   public void doGet(HttpServletRequest req, HttpServletResponse res)
32       throws ServletException, java.io.IOException {
33     String fileSep = System.getProperty("file.separator");

35     logger.info("INFO : Hello");
36     logger.warning("WARN : Hello");
37     logger.severe("ERROR : Hello");
```

```
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  }
```

8.7 FileUpload

Deseori clientul trebuie să furnizeze unui servlet un volum mare de date, depozitate într-un fişier. 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 fişier de la un client la programul server (file upload). Interfaţa de programare a produsului se referă la partea de server - în cazul de faţă reprezentat prin servlet.

Instalarea produsului constă din dezarhivarea fişierului descărcat din Internet.

În plus este nevoie de

- *commons-io*

Fişierele

- *commons-fileupload-*.*.jar*
- *commons-io-*.*.jar*

se depun în catalogul `lib` al servlet-ului.

Transferarea unui fişier, din partea clientului nu ridică nici o problemă. În fişierul html de apelare, se defineşte un formular

```
<form
   action=. . .
```

```
enctype="multipart/form-data"
method="post">
```

iar un fişier de încărcat se fixează prin intermediul marcajului

```
<input type="file" name=. . . size=. . .>
```

Programul navigator afişează o fereastră de căutare, prin care clientul selectează fişierul pe care doreşte să-l încarce.

Dacă partea de client este un program, atunci se utilizează *commons-httpclient*, 8.6.1

Programarea încărcării revine la

1. Crearea unei *fabrici* pentru manipularea fişierelor 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 interfaţa `FileItem`.

   Se pot fixa parametrii

   - dimensiunea zonei de pe disc destinată datelor de încărcat

     ```
     DiskFileItemFactory factory = new DiskFileItemFactory();
     factory.setSizeThreshold(maxMemorySize);
     ```

   - catalogul temporar de reţinere a datelor de încărcat

     ```
     factory.setRepositoryPath(tempDirectory);
     ```

     sau direct

     ```
     DiskFileItemFactory factory = new DiskFileItemFactory(
       maxMemorySize, tempDirectory);
     ```

   - dimensiunea maximă a unui fişier

```
    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 fişier putem obţine numele şi valoarea atributului furnizat de client

```
String name = item.getFieldName();
String value = item.getString();
```

6. În cazul unui fişier putem afla numele câmpului `input`, numele fişierului, dimensiunea fişierului

```
String fieldName = item.getFieldName();
String fileName = item.getName();
long sizeInBytes = item.getSize();
```

7. Dacă dorim să salvăm fişierul pe calculatorul server atunci prelucrarea este

```
File uploadedFile = new File(...);
item.write(uploadedFile);
```

8. Dacă datele fişierului se încarcă în memoria calculatorului atunci prelucrarea este

```
InputStream in = item.getInputStream();
  //preluarea datelor din fluxul in
  . . .
in.close();
```

Alternativ, datele se pot reţine ca un şir de octeţi prin

```
byte[] data = item.get();
```

Exemplul 8.7.1 *Să se obţină în memoria serverului matricea conţinută într-un fişier text. În fişierul text, fiecare linie conţine o linie a matricei, iar elementele sunt separate prin spaţii.*

Metoda *getMatrix* utilizată va reface matricea din datele fişierului.

```
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        }
79        while(line!=null);
80        if(v.size()>0){
81          mn=v.size();
82          n=mn/m;
83          matrix=new double[m][n];
84          for(int i=0;i<m;i++){
85            for(int j=0;j<n;j++){
86              matrix[i][j]=((Double)v.elementAt(i*n+j)).doubleValue();
87              System.out.print(matrix[i][j]+" ");
88            }
89            System.out.println();
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 referinţa către fişierul `commons-fileupload-*.*.jar`.

Formularul clientului este

```
1 <!doctype html>
```

```
<body bgcolor="#bbccbb">
  <h1> Inc&#259;rcarea unui fi&#351;ier </h1>
  <form
       action="/upload/upload"
       enctype="multipart/form-data"
       method="post"
       name="linear" onSubmit="return checkIt()">
  <p>
   Selecta&#355;i fi&#351;ierul
  <p>
  <input type="file" name="myfile" size=30 required>
  <p>
  <input type="submit" value="Expediaza fisierul">
  </form>
</body>
</html>
```

8.8 Descărcarea unui fişier

Considerăm cazul:

Exemplul 8.8.1 *Fişierul ales de client dintr-o lista disponibilă este descărcat fiind transmis navigatorului.*

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.ServletOutputStream;
import javax.servlet.annotation.WebServlet;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.Files;

@WebServlet(urlPatterns = "/download")

public class DownloadServlet extends HttpServlet {
  public void doGet(HttpServletRequest req,HttpServletResponse res)
      throws ServletException,IOException{
    ServletOutputStream out=res.getOutputStream();
    String file=req.getParameter("file");
    System.out.println(file);
    Path cale=Paths.get("webapps/download/resources/"+file);
    try{
      System.out.println(cale+file);
      res.setContentType("Application/Octet-stream");
      res.addHeader("Content-Disposition", "attachment; filename="+ file);
      Files.copy(cale,out);
    }
    catch(Exception e){
      res.setContentType("text/plain");
      out.println("Cererea d-voastra nu poate fi satisfacuta");
    }
```

```
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 şi a extensiei pentru fişierul selectat a se descărca.

8.9 Filtru

Un filtru se asemeană unui servlet, dar activitatea întreprinsă vizează uzual *context*ul, adică ansamblul servleţilor care fac parte din aplicaţia Web.

Din nou tehnica de programare poate fi:

- Descriptiv. Filtrul se declară în fişierul `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 interfaţa `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 8.9.1 *Contextul filtrud conţine doi servleţi HelloServlet şi CmmdcServlet, apelabili respectiv din hello.html, cmmdc.html. Să se programeze un filtru care*

- *Redirectează solicitarea "/filtrud/hello" către "/cmmdc.html".*
- *Dacă se cere ca natura răspunsului să fie "text/xml" atunci invalidează cererea.*

Servleţii *HelloServlet* şi *CmmdcServlet* sunt cei dezvoltaţi la începutul acestui capitol. Filtrul (în varianta descriptivă) are codul

```
1  import java.io.IOException;
2  import javax.servlet.ServletRequest;
3  import javax.servlet.ServletResponse;
4  import javax.servlet.RequestDispatcher;
5  import javax.servlet.Filter;
6  import javax.servlet.FilterConfig;
7  import javax.servlet.FilterChain;
8  import javax.servlet.ServletException;
9  import javax.servlet.http.HttpServletRequest;
10 import javax.servlet.http.HttpServletResponse;

12 public class MyFilterDispatcher implements Filter {
13   private FilterConfig filterConfig;

15   public void init(FilterConfig filterConfig) throws ServletException {
16     this.filterConfig = filterConfig;
17   }

19   public void destroy() {
20     this.filterConfig = null;
21   }

23   public void doFilter(ServletRequest request, ServletResponse response,
24       FilterChain filterChain) throws IOException, ServletException{
25     HttpServletRequest req = (HttpServletRequest) request;
26     HttpServletResponse res = (HttpServletResponse) response;
27     String uri = req.getRequestURI();
28     System.out.println("Filter URI= "+uri);
29     System.out.println(uri);

31     if(uri.equals("/filtrud")||uri.equals("/filtrud/"))
```

```
32        filterChain.doFilter(request, response);
33      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 }
```

Fişierul *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">

6 <web-app>
7   <filter>
8       <filter-name>filterDispatcher</filter-name>
9       <filter-class>MyFilterDispatcher</filter-class>
10  </filter>

12  <filter-mapping>
13      <filter-name>filterDispatcher</filter-name>
14      <url-pattern>/*</url-pattern>
15  </filter-mapping>
16 </web-app>
```

Pentru înţelegerea aplicaţiei prezentăm şi codul fişierului *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>
```

8.10 Eveniment şi auditor

Tehnologia Servlet permite urmărirea şi intervenţia de către serverul Web în:

- ciclul de viaţă al unui servlet, prin interfaţa

  `javax.servlet.ServletContextListener`

- evoluţia obiectului `javax.servlet.http.HttpSession` prin interfaţa

  `javax.servlet.http.HttpSessionListener`

Interfaţa `ServletContextListener` declară metodele

- `void contextInitialized(ServletContextEvent` *sec*`)`
- `void contextDestroyed(ServletContextEvent` *sec*`)`

Interfaţa `HttpSessiontListener` declară metodele

- `void sessionCreated(HttpSessionEvent` *hse*`)`
- `void sessionDestroyed(HttpSessionEvent` *hse*`)`

Clasa care implementează una din aceste interfeţe se declară în fişierul `web.xml` printr-un element `<listener>>`

```
<listener>
   <listener-class> clasa_listener </listener-class>
</listener>
```

Exemplul 8.10.1 *Auditor care sesizează încărcarea şi dispariţia unui servlet afişând în fereastra DOS a serverului Web un mesaj.*

```
import javax.servlet.ServletContextListener;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContext;

public class FirstContextListener
    implements ServletContextListener {
  public void contextDestroyed(ServletContextEvent event) {
    System.out.println("Web app was removed.");
  }
  public void contextInitialized(ServletContextEvent event) {
    System.out.println("Web app is ready.");
    ServletContext sc=event.getServletContext();
    System.out.println(sc.getContextPath());
    System.out.println(sc.getEffectiveMajorVersion());
    System.out.println(sc.getEffectiveMinorVersion());
  }
}
```

Exemplul 8.10.2

```
import javax.servlet.http.HttpSessionListener;
import javax.servlet.http.HttpSessionEvent;

public class FirstSessionListener implements HttpSessionListener {
  static int users = 0;

  public void sessionCreated(HttpSessionEvent e) {
    users++;
  }
  public void sessionDestroyed(HttpSessionEvent e) {
    users--;
  }
  public static int getConcurrentUsers() {
    return users;
  }
}
```

8.11 Server *apache-tomcat* încorporat

Într-o clasă Java se poate încorpora un server *tomcat* în care pot fi instalate una sau mai mulţi serveţi. Resursele necesare sunt conţinute în *apache-tomcat-*-embedded,* iar fişierele `jar` conţinute trebuie declarate în variabila de sistem `classpath`.

Şablonul 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 }
```

8.12 Dezvoltarea unui servlet prin *maven*

Dezvoltarea unui servlet prin *maven* se va exemplifica prin aplicaţia în care servletul răspunde clientului cu mesajul *Hi* `nume_client`, *nume_client* fiind parametru transmis de către client la apelarea servlet-ului.

Container de servleţi cu care lucrează *maven* este *jetty.*

Dezvoltarea revine la parcurgerea paşilor:

1. **Generarea aplicaţiei:**

```
set GroupID=hello
set ArtifactID=helloname
set Version=1.0
mvn -B archetype:generate
  -DgroupId=%GroupID%
  -DartifactId=%ArtifactID%
  -Dversion=%Version%
  -DarchetypeArtifactId=maven-archetype-webapp
```

Se crează structura de cataloage şi fişiere

```
helloname
|--> src
|    |--> main
```

```
|    |    |--> resources
|    |    |--> webapp
|    |    |    |--> WEB-INF
|    |    |    |    |   web.xml
|    |    |    |    index.jsp
|    pom.xml
```

2. **Completarea aplicaţiei.**

   - 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*.

   - Adaptarea fişierului `web.xml`.

     – Varianta descriptivă:
     Fişierul `web.xml` este completat cu elementele specifice servletului (`servlet` şi `servlet-mapping`)

     ```
     <!DOCTYPE web-app PUBLIC
      "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
      "http://java.sun.com/dtd/web-app_2_3.dtd" >

     <web-app>
       <display-name>Archetype Created Web Application</display-name>
       <servlet>
         <servlet-name>hello</servlet-name>
         <servlet-class>hello.HelloServlet</servlet-class>
       </servlet>
       <servlet-mapping>
         <servlet-name>hello</servlet-name>
         <url-pattern>/hello</url-pattern>
       </servlet-mapping>
     </web-app>
     ```

     – Varianta programată

     ```
     <?xml version="1.0" encoding="UTF-8"?>
     <web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

       <display-name>Archetype Created Web Application</display-name>
     </web-app>
     ```

- Fişierul *index.jsp* se înlocuieşte cu fişierul *index.html* utilizat la servletul menţionat anterior[2].
- Fişierul `pom.xml` se completează cu
  - Referinţele 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>
```

  - Referinţele serverului Web *jetty*

```
<plugin>
     <groupId>org.eclipse.jetty</groupId>
     <artifactId>jetty-maven-plugin</artifactId>
     <version>9.4.7.v20170914</version>
</plugin>
```

  - Fixarea versiunii compilatorului Java

```
<plugin>
     <groupId>org.apache.maven.plugins</groupId>
     <artifactId>maven-compiler-plugin</artifactId>
     <version>3.7.0</version>
     <configuration>
          <source>1.8</source>
          <target>1.8</target>
      </configuration>
</plugin>
```

  Codul fişierului `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</version>
10   <name>helloname Maven Webapp</name>
11   <url>http://maven.apache.org</url>

13   <dependencies>
14     <dependency>
15       <groupId>javax.servlet</groupId>
16       <artifactId>javax.servlet-api</artifactId>
17       <version>3.0.1</version>
18       <scope>provided</scope>
19     </dependency>
20   </dependencies>
21   <build>
22     <finalName>helloname</finalName>
```

[2]Atenţie la parametri de apelare a servlet-ului, care eventual trebuie adaptaţi. Contextul aplicaţiei coincide cu parametrul *artifactId*

```
23      <plugins>
24        <plugin>
25          <groupId>org.eclipse.jetty</groupId>
26          <artifactId>jetty-maven-plugin</artifactId>
27          <version>9.4.7.v20170914</version>
28        </plugin>
29        <plugin>
30          <groupId>org.apache.maven.plugins</groupId>
31          <artifactId>maven-compiler-plugin</artifactId>
32          <version>3.7.0</version>
33          <configuration>
34            <source>1.8</source>
35            <target>1.8</target>
36          </configuration>
37        </plugin>
38      </plugins>
39    </build>
40  </project>
```

Se va completa peste tot cu versiunile curente ale resurselor utilizate.

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 opreşte cu `Ctrl+C`.

   (c) testarea servlet-ului: într-un navigator se deschide pagina `http://localhost:8080`

Întrebări recapitulative

1. Precizaţi sensurile şi conţinutul cuvântului *servlet*.
2. Unde se instalează o aplicaţie servlet în serverul Web *apache-tomcat*?
3. Cum se poate apela aplicaţia servlet, dar servlet-ul propriu zis?
4. Extinzând clasa HttpServlet, ce trebuie să facă programatorul?
5. Care sunt modurile de programare a unui servlet şi precizaţi diferenţa 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?
13. În ce constă tehnologia *upgrade* ?
14. Ce oferă tehnologia *Server Push* ?

Capitolul 9

AJAX vs. JSONP

Scopul acestui capitol este prezentarea posibilităţii apelării unui servlet prin funcţii JavaScript. O trasătură este faptul că răspunsul furnizat de un program server reface doar o parte din pagina html şi nu întrega pagină, aşa cum, de exemplu, este cazul utilizării obişnuite a unui servlet.

- *Asynchronous JavaScript And Xml* -(AJAX) - permite, pe partea de client, un schimb de date cu un program server Web, prin funcţii JavaScript. La bază se află o interfaţă de programare (API - Application Programming Interface) *XMLHttpRequest* (XHR), iniţiată de Microsoft, ce poate fi utilizată de un limbaj de scripting (JavaScript, JScript, VBScript, etc) pentru
  - transfer de date către un server Web utilizând protocolul HTTP;
  - manipularea datelor XML sau JSON (JavaScript Object Notation).
- *JSON with Padding* - (JSONP)[1], (JSON ca umplutură).

  În JSONP se va încărca o expresie JSON ca o funcţie. Astfel extresia JSON

  {"nume":"valoare"}

  se utilizează în JSOP sub /forma

  myFct({"nume":"valoare"})

[1] Acronimul *JSON* desemnează pe de-o parte *JSON with Padding* dar şi *JSON Processing,* reprezentat în Java, de exemplu prin pachetul `javax.json`.

În AJAX nu se pot încărca date din alt domeniu. Prin JSONP prin marcajul <`script`> se obţin referinţe la funcţii din alt domeniu.

În AJAX rezultatul se prelucrează de o funcţie anonimă, necunoscută serverului spre deosebire de JSONP unde se va utiliza o funcţie cu nume fixat.

9.1 *AJAX* – Java

Există două implementări a interfeţei *XMLHttpRequest*:

- `ActiveXObject` în navigatorul *MS InternetExplorer;*
- `XMLHttpRequest` în celelate navigatoare.

Metodele interfeţei XMLHttpRequest.

- `open(`*method,URL*`)`
  `open(`*method,URL,async*`)`
  `open(`*method,URL,async,userName*`)`
  `open(`*method,URL,async,userName,password*`)`

  method poate fi `get` sau `post.`

  async fixează natura comunicaţiei - `true` pentru comunicaţie asincronă.
- `send(`*content*`)`
- `abord()`
- `getAllResponseHeaders()`
- `getResponseHeader(`*headerName*`)`
- `setRequestHeader(`*label,value*`)`

Proprietăţile interfeţei XMLHttpRequest.

- `onreadystatechange`

  Conţine numele funcţiei script care prelucrează răspunsul.
- `readyState`

  Indicatorul obiectului *XMLHttpRequest*: 0 - neiniţializat; 1 - deschis; 2 - trimis; 3 - recepţionat; 4 - încărcat.

- `responseText / responseXML`

  Conţine răspunsul sub forma text / xml.

- `status / statusText`

  404 - Not Found; 200 - OK.

Punctul de pornire al unei aplicaţii AJAX - Java este o pagină Web - `html`. La generarea unui eveniment legat de un element grafic al paginii Web se apelează o serie de funcţii JavaScript a căror execuţie realizează comunicaţia cu un program server - servlet sau jsp. Uzual, programul server formulează un răspuns sub forma unui fişier `xml`, care este prelucrat de o funcţie JavaScript oferind date clientului.

Simplificând, punem în evidenţă 3 funcţii JavaScript

1. Generarea unui obiect `XMLHTTPRequest`

```
function initRequest() {
  if (window.XMLHttpRequest) {
    return new XMLHttpRequest();
  }
  else
    if (window.ActiveXObject){
      return new ActiveXObject("Microsoft.XMLHTTP");
    }
}
```

2. Funcţia apelată de eveniment şi care lansează comunicaţia AJAX

```
function XXX() {
  var idField=document.getElementById("numeCimp");
  var url=
   "http://host:port/context/numeServlet?numeCimp="+
    escape(idField.value);
  var req = initRequest();
  req.onreadystatechange = function() {
    if (req.readyState == 4) {
      if (req.status == 200) {
        functiaPrelucrareRaspuns(req.responseXML);
      } else {
        alert(req.status+" : "+req.statusText);
```

```
13          }
14        }
15    };
16    req.open("GET", url, true);
17    req.send(null);
18  }
```

Câmpul `responseXML` se utilizează pentru înmagazinarea unui răspuns în format XML, iar `responseText` se utilizează pentru preluarea unui răspuns JSON.

3. Funcţia de prelucrare a răspunsului.

Exemplul 9.1.1 *Aplicaţie Web de alegere a unei oferte. Pagina Web a aplicaţiei afişează o listă de oferte de cursuri opţionale. Un student - client - selectează cursul dorit iar selecţia este transmisă unui servlet care centralizează alegerile.*

Lista cursurilor opţionale este încărcată în momentul apelării paginii Web utilizând AJAX. Pentru AJAX, pe partea de server este un alt servlet care trimite lista cursurilor opţionale sub forma unui fişier XML.

Găzduită de *apache-tomcat* desfăşurarea aplicaţiei este

```
webapps
   |--- ajax
          |--- WEB-INF
          |      |--> classes
          |      |      |--- AJAXAlegereServlet.class
          |      |      |--- AJAXCompletareServlet.class
          |      |--> lib
          |      |      |--- javax.json.jar
          |      |--- web.xml
          |--- index.html
```

Codul Java al programului *AJAXCompletareServlet* este

```
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.json.JsonArray;
8 import javax.json.JsonWriter;
9 import javax.json.Json;
```

```
11 public class AJAXCompletareServlet extends HttpServlet{
12   public void doGet(HttpServletRequest req,HttpServletResponse res)
13     throws ServletException,IOException{
14     PrintWriter out=res.getWriter();
15     String tip=req.getParameter("tip");
16     res.setHeader("Cache-Control","no-cache");
17     if(tip.equals("xml")){
18       res.setContentType("text/xml");
19       out.print("<?xml version=\"1.0\" ?>");
20       out.print("<optionale>");
21       out.print("<disciplina>");
22       out.print("<denumire> Calcul Paralel </denumire>");
23       out.print("</disciplina>");
24       out.print("<disciplina>");
25       out.print("<denumire> Tehnologii distribuite </denumire>");
26       out.print("</disciplina>");
27       out.print("<disciplina>");
28       out.print("<denumire> Rezolvarea numerica a e.d.o. </denumire>");
29       out.print("</disciplina>");
30       out.print("</optionale>");
31     }
32     else{
33       res.setContentType("application/json");
34       // jee
35       JsonArray jsonArray=Json.createArrayBuilder()
36        .add(Json.createObjectBuilder()
37            .add("nume","Analiza numerica"))
38        .add(Json.createObjectBuilder()
39          .add("nume","Programare distribuita"))
40        .add(Json.createObjectBuilder()
41          .add("nume","Soft matematic"))
42       .build();
43       JsonWriter jsonWriter=Json.createWriter(out);
44       jsonWriter.writeArray(jsonArray);
45       jsonWriter.close();
46     }
47     out.close();
48   }

50   public void doPost(HttpServletRequest req,HttpServletResponse res)
51     throws ServletException,IOException{
52     doGet(req,res);
53   }
54 }
```

Răspunsul nu se stochează la recepţie

```
response.setHeader("Cache-Control","no-cache");
```

În varianata XML, natura răspunsului este ”text/xml”

```
response.setContentType("text/xml");
```

iar în varianta JSON acesta este ”text/plain”.

În cazul exemplului, în varianta XML răspunsul la apelarea servlet-ului este fişierul `xml`

```
<?xml version=\"1.0\" ?>
<optionale>
  <disciplina>
     <denumire> Calcul paralel </denumire>
  </disciplina>
  <disciplina>
     <denumire> Tehnologii distribuite </denumire>
  </disciplina>
  <disciplina>
     <denumire> Rezolvarea numerica a e.d.o. </denumire>
  </disciplina>
</optionale>
```

iar, în varianta JSON, răspunsul este stringul

```
[{"nume":"Analiza numerica"},{"nume":"Programare distribuita"},{"nume":"Soft matematic"}]
```

Servlet-ul aplicaţiei (*AJAXalegereServlet*) este banal: confirmă clientului alegerea făcută

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class AJAXAlegereServlet extends HttpServlet{
  public void doGet(HttpServletRequest req,HttpServletResponse res)
    throws ServletException,IOException{
    String materia=req.getParameter("alegere");
    PrintWriter out=res.getWriter();
    res.setContentType("text/html");
    out.println("<html><body>");
    out.println("<h1> Disciplina optionala selectata </h1>");
    out.println("<p>"+materia);
    out.println("</body></html>");
    out.close();
  }

  public void doPost(HttpServletRequest req,HttpServletResponse res)
    throws ServletException,IOException{
    doGet(req,res);
  }
}
```

Pagina Web de apelare a aplicaţiei (*indexXMLAlegere.html*) este

```
<html>
<head>

<script language="javascript">
<!--
function initRequest() {
  if (window.XMLHttpRequest) {
    return new XMLHttpRequest();
  }
```

```
10    else if (window.ActiveXObject){
11      return new ActiveXObject("Microsoft.XMLHTTP");
12    }
13  }

15  function doCompletion() {
16    var tipField=document.getElementById("tip");
17    var url = "http://localhost:8080/ajax/completare?tip="+
18      escape(tipField.value);
19    var req = initRequest();
20    if(req!=null){
21      req.open("GET", url, true);
22      req.onreadystatechange = function() {
23              if (req.readyState == 4) {
24                  if (req.status == 200) {
25                     parseMessages(req.responseXML);
26                  } else {
27                      alert(req.status+" : "+req.statusText);
28                  }
29              }
30      };
31      req.send(null);
32    }
33  }

35  function parseMessages(responseXML) {
36    var optionale = responseXML.getElementsByTagName("optionale")[0];
37    var select=document.getElementById("alegere");
38    for (loop = 0; loop < optionale.childNodes.length; loop++){
39      var disciplina = optionale.childNodes[loop];
40      var denumire = disciplina.getElementsByTagName("denumire")[0];
41      var den=denumire.childNodes[0].nodeValue;
42      select.options[loop]=new Option(den,den,false,false);
43    }
44  }
45  -->
46  </script>

48  <title>
49    Auto-Completion using Asynchronous JavaScript and XML (AJAX)
50  </title>
51  </head>
52  <body onload="doCompletion()">

54   <h1>Auto-Completion using Asynchronous JavaScript and XML (AJAX)</h1>

56   <form name="autofillform"
57     action="/ajax/alegerea" method="get">


60     <b>Disciplina optional : </b>

62        <select name="alegere" id="alegere" >
63        </select>

65      <p>
66        <input type="Submit" value="Transmite">
67        <input type="reset" value="Abandon" >
68        <input type="hidden" id="tip" value="xml" >
```

```
69    </form>
70  </body>
71 </html>
```

respectiv (*indexJSONAlegere.html*)

```
1  <html>
2  <head>

4  <script language="javascript">
5  <!--
6  function initRequest() {
7    if (window.XMLHttpRequest) {
8      return new XMLHttpRequest();
9    }
10   else if (window.ActiveXObject){
11     return new ActiveXObject("Microsoft.XMLHTTP");
12   }
13 }

15 function doCompletion() {
16   var tipField=document.getElementById("tip");
17   var url = "http://localhost:8080/ajax/completare?tip="+
18     escape(tipField.value);
19   var req = initRequest();
20   if(req!=null){
21     req.open("GET", url, true);
22     req.onreadystatechange = function() {
23             if (req.readyState == 4) {
24                 if (req.status == 200) {
25                     parseMessages(req.responseText);
26                 }else {
27                     alert(req.status+" : "+req.statusText);
28                 }
29             }
30     };
31     req.send(null);
32   }
33 }

35 function parseMessages(responseText){
36   var s=eval(responseText);
37   var select=document.getElementById("alegere");
38   for(var i=0;i<s.length;i++){
39     select.options[i]=new Option(s[i].nume,s[i].nume,false,false);
40   }
41 }
42 -->
43 </script>

45 <title>
46   Auto-Completion using Asynchronous JavaScript and JSON (AJAX)
47 </title>
48 </head>
49  <body onload="doCompletion()">

51  <h1>Auto-Completion using Asynchronous JavaScript and XML (AJAX)</h1>

53  <form name="autofillform"
```

```
54     action="/ajax/alegerea" method="get">


57     <b>Disciplina optional :</b>

59        <select name="alegere" id="alegere" >
60        </select>

62      <p>
63        <input type="Submit" value="Transmite">
64        <input type="reset" value="Abandon" >
65        <input type="hidden" id="tip" value="json" >
66   </form>
67  </body>
68 </html>
```

Exemplul 9.1.2 *Calcul celui mai mare divizor comun a două numere naturale cu client AJAX.*

Programul servlet este

```
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;

8  public class AJAXCmmdcServlet extends HttpServlet{

10   public long cmmdc(long m, long n){. . .}

12   public void doGet(HttpServletRequest req,HttpServletResponse res)
13       throws ServletException,IOException{
14     String sm=req.getParameter("m"),sn=req.getParameter("n");
15     long m=Long.parseLong(sm),n=Long.parseLong(sn);
16     String tip=req.getParameter("tip");
17     long x=cmmdc(m,n);
18     PrintWriter out=res.getWriter();
19     res.setHeader("Cache-Control","no-cache");
20     if(tip.equals("xml")){
21       res.setContentType("text/xml");
22       out.print("<?xml version=\"1.0\" ?>");
23       out.print("<rezultat>");
24       out.print(Long.valueOf(x).toString());
25       out.print("</rezultat>");
26     }
27     else{
28       res.setContentType("application/json");
29       out.println(Long.valueOf(x).toString());
30     }
31     out.close();
32   }

34   public void doPost(HttpServletRequest req,HttpServletResponse res)
35     throws ServletException,IOException{
36     doGet(req,res);
```

```
37    }
38 }
```

Se observă diferenţa faţa de soluţia non-AJAX doar în răspunsul formulat care este un document `xml` şi nu `html`.

Clientul în format XML

```
1  <html>
2  <head>

4  <script type="text/javascript" >
5  <!--

7    function initRequest() {
8      if (window.XMLHttpRequest) {
9          return new XMLHttpRequest();
10     } else if (window.ActiveXObject){
11         return new ActiveXObject("Microsoft.XMLHTTP");
12     }
13   }

15   function compute() {
16     var mField=document.getElementById("m");
17     var nField=document.getElementById("n");
18     var tipField=document.getElementById("tip");
19     var url = "http://localhost:8080/ajax/cmmdc?m=" +
20        escape(mField.value)+"&n="+escape(nField.value) +
21        "&tip=" + escape(tipField.value);
22     var req = initRequest();
23     req.onreadystatechange = function() {
24         if (req.readyState == 4) {
25             if (req.status == 200) {
26                 parseMessages(req.responseXML);
27             } else {
28                 alert(req.status+" : "+req.statusText);
29             }
30         }
31     };
32     req.open("get", url, true);
33     req.send(null);
34   }

36   function parseMessages(responseXML) {
37     var r = responseXML.getElementsByTagName("rezultat")[0];
38     var cmmdc=r.childNodes[0].nodeValue;
39     document.getElementById("rezultat").innerHTML="Cmmdc = "+cmmdc;
40   }
41 -->
42 </script>

44 <title> Cmmdc AJAX</title>
45 </head>
46  <body>
47          <h1>Cmmdc with AJAX</h1>
48          <p>
49          Primul numar :
50          <input type="text" id="m" value="1" size="15" >
51          <p>
```

```
52          Al doilea numar :
53          <input type="text"  id="n" value="1" size="15" >
54          <input type="hidden" id="tip" value="xml" >
55          <p>
56          <input type="button" value="Calculeaza" onClick="compute()" >
57          <p>
58          Cel mai mare divizor comun a celor doua numere este
59          <p>
60          <div id="rezultat" />
61  </body>
62 </html>
```

În varianta JSON funcţia javascript de prelucrare a răspunsului este

```
function parseMessages(responseText) {
  var cmmdc=responseText;
  document.getElementById("rezultat").innerHTML="Cmmdc = "+cmmdc;
}
```

Funcţiile javascript pot fi salvate într-un fişier iar referinţa la ele se dă prin

```
<script language="javascript" src="fisier_functii.js"
</script>
```

9.2 JSON with Padding

În marcajul <script> se definesc două funcţii Javascript:

1. Funcţie Javascript pentru apelarea servlet-ului utilizând metoda GET;
2. Funcţie Javascript pentru prelucrarea răspunsului.

Exemplul 9.2.1

```
1 <!doctype html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <link rel="stylesheet" href="mycss.css">
6 </head>
7 <body>
8   <script>
9     function sender(){
10      var mField=document.getElementById("m");
11      var nField=document.getElementById("n");
12      s = document.createElement("script");
13      s.src = "/jsonp/cmmdc?m="+escape(mField.value)+
14          "&n="+escape(nField.value);
15      document.body.appendChild(s);
16    }
```

```
18      function myFct(myObj) {
19        document.getElementById("result").innerHTML=
20          "Cmmdc : "+myObj.Cmmdc;
21      }
22    </script>

24    <center>
25    <h1> Pagina de apelare CmmdcServlet </h1>
26        <table>
27          <tr>
28            <td><label> Primul numar </label></td>
29            <td>
30               <input type="number" id="m" size="5"
31               placeholder="introduceti"
32               required min="1">
33            </td>
34          </tr>
35          <tr>
36            <td><label> Al doilea numar </label></td>
37            <td>
38               <input type="number" id="n" size="5"
39               placeholder="introduceti"
40               required min="1"">
41            </td>
42          </tr>
43          <tr>
44            <td>
45               <p><button onclick="sender()">Calculeaza</button>
46            </td>
47          </tr>
48        </table>
49        <p>
50        <div id="result"></div>
51    </center>
52  </body>
53  </html>
```

Servlet-ul returnează un `String` de apelare a funcţiei Javascript de prelucrare a răspunsului. Argumentul funcţiei este o expresie JSON cu datele răspunsului.

```
1   package cmmdc;
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;
9   import java.io.PrintWriter;

11  @WebServlet(urlPatterns = "/cmmdc")

13  public class CmmdcServlet extends HttpServlet{
14    public long cmmdc(long m, long n){. . .}

16    public void doGet(HttpServletRequest req,HttpServletResponse res)
17        throws ServletException,IOException{
18      String sm=req.getParameter("m"),sn=req.getParameter("n");
```

```
19      long m=Long.parseLong(sm),n=Long.parseLong(sn);
20      long x=cmmdc(m,n);
21      PrintWriter out=res.getWriter();
22      res.setContentType("text/plain");
23      String r="{\"Cmmdc\":"+Long.valueOf(x).toString()+"}";
24      out.println("myFct("+r+")");
25      out.close();
26    }

28    public void doPost(HttpServletRequest req,HttpServletResponse res)
29        throws ServletException,IOException{
30      doGet(req,res);
31    }
32  }
```

Capitolul 10

Java Server Page – JSP

10.1 Tehnologia JSP

Tipul tehnologiei JSP este denumit *procesare de şabloane (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`

- referinţă 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 înţelege calea de la catalogul `webapps` până la catalogul ce conţine fişierul jsp.

Vom depozita fişierele 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 fişierului *Hello.html*

```
<html>
  <body>
     Hello
  </body>
</html>
```

în *Hello.jsp* şi plasându-l în catalogul *jsp* se obţine acelaşi efect, dar prelucrarea paginilor / documentelor este diferită. Fişierul html este prelucrat doar de programul navigator şi poate fi deschis ca fişier, în timp ce fişierul JSP este prelucrat de serverul Web cu afişarea prin intermediul navigatorului. Prelucrarea efectuată de serverul Web constă din transformarea paginii / documentului JSP într-un servlet, care este compilat şi lansat în execuţie. Din aceastră cauză prima invocare a paginii / documentului JSP durează mai mult decât apelările ulterioare.

Fişierul 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.

  Fişierul are extensia `jsp` şi se numeşte *pagină JSP.*

- prin elemente xml aparţinând spaţiului de nume

  `http://java.sun.com/JSP/Page`

  Fişierul poate avea extensia `jsp` sau `jspx` şi se numeşte *document JSP.*

Comentariile JSP se scriu de forma

```
<%-- Comentariu --%>
```

Considerăm următorul exemplu introductiv:

Exemplul 10.1.1 *Putem afişa valoarea unei variabile Java (de exemplu data calendaristică) prin*

- Varianta paginii JSP

```
<html>
  <body>
    <p>
    Data calendaristica 1:
    <%= new java.util.Date() %>

    <p>
    <% java.util.Date data1=new java.util.Date(); %>
    Data calendaristica 2:
    <%= data1 %>

    <p>
    <% java.util.Date data2=new java.util.Date(); %>
    Data calendaristica 3:
    <% out.print(data2); %>
  </body>
</html>
```

Dacă se utilizează operatorul de afişare = atunci după expresia de afişat nu se pune ;.

Variabila predefinită `out` este de tip `javax.servlet.jsp.JspWriter`.

- Varianta documentului JSP

```
<?xml version="1.0" encoding="UTF-8" ?>
<html xmlns:jsp="http://java.sun.com/JSP/Page">
<body>
  <jsp:scriptlet>
    java.util.Date d=new java.util.Date();
  </jsp:scriptlet>
  <jsp:expression> d </jsp:expression>
</body>
</html>
```

Codul Java înglobat într-un text html se numeşte *scriptlet.* Sintaxa utilizată este

- Varianta paginii JSP

<% cod Java %>

- Varianta documentului JSP

`<jsp:scriptlet> codJava </jsp:scriptlet>`

Domeniul de valabilitate. Domeniul de valabilitate defineşte intervalul de timp, de existenţă al unui obiect, fiind definit prin valorile:

Valoare	Domeniu de valabilitate
page	pagina curentă
request	în pagina curentă, în paginile incluse şi în paginile către care se face o redirectare
session	în sesiunea curentă
application	pe durata rulării aplicaţiei

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 10.1.2 *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 şi arhivarea servlet-ului o vom realiza prin intermediul lui *apache-ant.* Î̂n acest scop se crează structura:

```
jsphello
|     |--> src
|     |--> web
|     |     |      |--> jsp
|     |     |      |      hello.jsp
|     |     |--> WEB-INF
|     |     |      |--> classes
|     |     |      |--> lib
|     |     |      |      web.xml
|     |     |      index.html
```

Fişierul *build.xml* este simplu

```
1  <project basedir="." default="generate.war">
2    <property name="dist.name" value="JSPApp" />
3    <property name="dist.dir" value="dist" />

5    <path id="myclasspath">
6      <fileset dir="web/WEB-INF/lib">
7        <include name="*.jar"/>
8      </fileset>
9    </path>

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>
```

10.1.1 Declaraţii JSP

Printr-o *declaraţie JSP*, putem defini câmpuri(variabile) şi metode Java ce pot fi apoi folosite, respectiv apelate în documentul respectiv. O declaraţie JSP se defineşte printr-un marcaj

<%! . . . %>

sau, în format XML

<jsp:declaration> . . . </jsp:declaration>

Exemplul 10.1.3 *Calculul celui mai mare divizor comun a două numere naturale cu metoda de calcul este definită într-o declaraţie JSP.*

Pagina JSP a aplicaţiei *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*

```
<!doctype html>
<head>
  <meta charset="utf-8">
</head>
<body bgcolor="#bbccbb">
  <center>
  <h1> Pagina de apelare CmmdcServlet </h1>
  <form method="get"
      action="jsp/cmmdc.jsp">
      <table>
        <tr>
          <td><label> Primul numar </label></td>
          <td>
             <input type="number" name="m" size="5"
             required  min="1">
          </td>
        </tr>
        <tr>
          <td><label> Al doilea numar </label></td>
          <td>
             <input type="number" name="n" size="5"
             required  min="1">
          </td>
        </tr>
        <tr>
          <td>
             <p><input type="submit" value="Calculeaza">
          </td>
        </tr>
      </table>
  </form>
  <center>
</body>
</html>
```

Există două metode `jspInit()` şi `jspDestroy()` care dacă sunt declarate de programator atunci sunt executate la începutul şi la sfârşitul ciclului de viaţă a paginii / documentului JSP.

Exemplul 10.1.4 *Metoda* ***`jspInit`*** *iniţializează un număr cu 0 iar metoda* ***`jspDestroy`*** *afişează numărul pe calculatorul serverului. Un client introduce un număr care este adunat la cel reţinut de pagina JSP.*

Efectul metodei `jspDestroy` are loc în urma opririi aplicaţiei JSP.
Codul paginii JSP este

```
<html>
  <head>
    <title> Init </title>
  </head>
  <body>
    <%!
      int numar;
      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>
```

10.1.2 Directive JSP

Directivele JSP fixează informaţii 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`. Menţionăm atributele

`import=`	*"listă de pachete separate prin ,"*
`info=`	*"text"* Informaţia se poate regăsi apelând metoda getServletInfo()
`errorPage=`	*"adresa url a paginii ce tratează excepţia"*
`isErrorPage=`	`"true | false"`

- Directiva `include` permite includerea unor fişiere .html sau .jsp în document

<%@ `include file=`*"fişier html, jsp"* %>

Includerea are loc în locul în care apare directiva.

Referinţa la fişierul `html` sau `jsp` se face relativ la catalogul paginii JSP iniţiale (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"*

10.1.3 Marcaje JSP predefinite

Un marcaj JSP defineşte o acţiune care se execută în timpul procesării paginii `jsp`. Sintaxa marcajelor JSP seamănă cu cea a marcajelor `html` sau `xml`

<prefix : marcaj *atribute* />

Dintre marcajele JSP predefinite – adică cu prefixul JSP – amintim:

- <jsp : include page=*"numeFişier jsp sau html"* />
- <jsp : forward page=*"numeFişier jsp sau html"* />

  Prelucrarea care urmează va fi cea din fişierul menţional. Diferenţa dintre cele două elemente constă în faptul că `include` prevede revenirea în pagina JSP iniţială iar `forward` nu.

  Referinţa la fişierul `html` sau `jsp` se face relativ la catalogul paginii JSP iniţiale.

  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.set*NumeProp(valoare)*.

  <jsp : setProperty name=*"numeComponentăJava"* property="*" /> vizează toate proprietăţile 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 şi afişează valoarea câmpului *numeComponentăJava.numeProp*.

10.1.4 Pagini JSP cu componente Java

Clasa componentei Java care se va utiliza într-o pagină JSP trebuie inclusă într-un pachet.

Reluăm exemplul 10.1.2 cu o componentă Java corespunzătoare numelui din formularul *index.html*.

Exemplul 10.1.5

```
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*)

```
<jsp:useBean id="obj" class="jsp.HelloBean" scope="request"/>
<jsp:setProperty name="obj" property="*"/>
<html>
  <head>
    <title> jsphello </title>
  </head>
  <body>
    <h1> Pagina de r&#259;spuns </h1>
    <center>
    <%
      out.println("Hi "+obj.getName()+" !");
    %>
    </center>
  </body>
</html>
```

Mai mult, se poate include formularul în pagina JSP, bineînţeles ştergându-l din fişierul `html`:

```
<jsp:useBean id="obj" class="jsp.HelloBean" scope="request"/>
<jsp:setProperty name="obj" property="*"/>
<html>
  <head>
    <title> jsphello </title>
  </head>
  <body>
    <center>
    <h1> Pagina JSP - aplica&#355;ia Hello </h1>
    <form method="post">
      <p> <h3> Introduceti numele: </h3>
      <input type="text" name="name" size=20>
      <p>
      <input type="submit">
    </form>
    <p>
    <%
      out.println("Hi "+obj.getName()+" !");
    %>
    </center>
  </body>
</html>
```

Exemplul 10.1.6 *Pagină JSP pentru calculul celui mai mare divizor comun cu metoda de calcul definită într-o componentă Java.*

Utilizând documentului `html` din Exemplul 10.1.3 se defineşte componenta Java

```
package cmmdc;
public class CmmdcBean{
  private String m="";
  private String n="";
  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 Long.valueOf(cmmdc(a,b)).toString();
24   }

26   long cmmdc(long m,long n){. . .}

28 }
```

Instanţiem o componenta Java şi îi fixăm proprietăţile (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 10.1.7 *Generarea unei excepţii (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 excepţiei (*errorpage.jsp*)

```
<!--
Aceste comentarii sunt foarte importante in cazul utilizarii
navigatorului IE si a lui apache-tomcat-5.*.*./6.*.*. In lipsa
lor nu se genereaza saltul la exceptie prin pagina jsp.
Rolul comentariilor este marirea lungimii fisierului de fata.

O alternativa este ca din IE6 . . . Options sa se dezactiveze
optiunea "Show friendly HTTP error message"

Cu navigatorul Firefox nu exista aceasta problema.
-->

<%@ page isErrorPage="true" %>
<html>
  <body>
    <div align="center">
      <%= exception.getMessage() %>
    </div>

  </body>
</html>
```

apelate prin

```
<html>
  <body>
    <form method=post
      action="jsp/errhandler.jsp">
      Care este materia preferata din anii de studiu universitar ?
      <p>
      Algoritmica si programare
      <input type="radio" name="materia" value="AP" checked>
      <p>
      Analiza numerica
      <input type="radio" name="materia" value="AN">
      <p>
      Inteligenta artificiala
      <input type="radio" name="materia" value="IA">
      <p>
      <input type=submit>
    </form>
  </body>
</html>
```

10.2 JSP Standard Tag Library JSTL

JSTL este o familie de biblioteci de marcaje ce oferă o serie de facilităţi activităţii de realizare a paginilor Web. JSTL ajută la separarea activităţii de programare de proiectarea (design) paginii Web.

JSTL este alcătuită din 5 biblioteci:

URI	Descriere
http://java.sun.com/jsp/jstl/core	Biblioteca de bază
http://java.sun.com/jsp/jstl/xml	Biblioteca de prelucrare a documentelor xml
http://java.sun.com/jsp/jstl/fmt	Biblioteca de formatare a datelor
http://java.sun.com/jsp/jstl/sql	Biblioteca de lucru cu baze de date
http://java.sun.com/jsp/jstl/functions	Biblioteca de funcţii ajutătoare

Instalarea bibliotecilor. Bibliotecile sunt livrate de

- *apache-tomcat-** în catalogul `apache-tomcat-*\webapps\examples\WEB-INF\lib` prin fişierele
  - `taglibs-standard-spec-*.jar`
  - `taglibs-standard-impl-*.jar`
- *glassfish-** în catalogul `glassfish\modules` prin fişierele
  - `javax.servlet.jsp.jstl.jar`
  - `javax.servlet.jsp.jstl-api.jar`

Utilizarea bibliotecilor. În vederea utilizării, cele două fişiere trebuie copiate în catalogul `lib` al aplicaţiei care utilizează bibliotecile.

În pagina / documentul JSP, o bibliotecă utilizată trebuie declarată printr-o directivă `taglib`.

10.2.1 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
`var`	obligatoriu	Numele variabilei ce va stoca valoarea expresiei.
`value`	opţional	Expresia care va fi evaluată şi atribuită variabilei
`scope`	opţional	Domeniul de valabilitate al variabilei. Unul din valorile: `page, request, session, application.`

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 acelaşi 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 obţine `false` sau `true` după cum variabila are sau nu atribuită o valoare.

- `c:remove` Şterge o variabilă.

  Atribute ale marcajului:

Atribut	Fel	Descriere
`var`	obligatoriu	Numele variabilei ce se şterge.
`scope`	opţional	Domeniul de valabilitate al variabilei. Unul din valorile: `page, request, session, application.`

- `c:out` Afişează o valoare.

  Atribute ale marcajului:

Atribut	Fel	Descriere
`value`	obligatoriu	Valoarea ce se evaluează şi se afişează.
`default`	opţional	Cea ce se afişează în cazul în care expresia nu poate fi evaluată.
`escapeXml`	opţional	`true / false.` Valoarea implicită este `true`. Pe `false` interpretează caracterele din `value` ca şi cod html.

- `c:if` Test, verificarea unei condiţii.

  Atribute ale marcajului:

Atribut	Fel	Descriere
`test`	obligatoriu	Condiţia de test.
`var`	opţional	Numele variabilei ce va stoca valoarea testului.
`scope`	opţional	Domeniul de valabilitate al variabilei definită anterior.

  În cazul în care condiţia are valoarea `true` se prelucrează corpul marcajului, în caz contrar, acesta este ignorat.

Exemplul 10.2.1 *Preluarea datelor unui formular cu câmpurile de intrare nume, prenume şi 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="${empty param.nume}" var="testNume" >
6        <c:out value="Numele lipseste !" />
7      </c:if>
8      <c:if test="${not testNume}" >
9        Nume:<c:out value="${param.nume}" />
10     </c:if>
11     <p>
12     <c:if test="${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 selecţie poate conţine oricâte marcaje `c:when` şi cel mult un marcaj `c:otherwise`. Fiecare marcaj `c:when` conţine obligatoriu atributul `test`. Dacă într-un marcaj `c:when` condiţia 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 atribute).

- `c:forEach` Ciclu.

  Atribute ale marcajului:

Atribut	Fel	Descriere
`items`	opţional	Colecţia care se parcurge.
`var`	opţional	Numele variabilei în care se stochează valoarea elementului curent.
`begin`	opţional	Valoarea iniţială a variabilei `var`.
`end`	opţional	Valoarea finală a variabilei `var`.
`step`	opţional	Valoarea pasului de iterare. Implicit este 1.
`varStatus`	opţional	Informaţii despre elementul curent.

  Variabila `varStatus` are câmpurile:

  - `index` valoarea curentă a elementului după care se realizează ciclarea;
  - `count` numărul iteraţiei curente;
  - `first` are valoarea `true` dacă este primul element al ciclului;
  - `last` are valoarea `true` dacă este ultimul element al ciclului;

Exemplul 10.2.2 *Lista parametrilor formularului de apelare a exemplului anterior se afişează 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 10.2.3 *Lista parametrilor unui header se afişează 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 10.2.4 *Evidenţierea 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 aceaşi funcţionalitate ca şi clasei `java.util.StringTokenizer`.

  Atribute ale marcajului:

Atribut	Fel	Descriere
`value`	obligatoriu	Valoarea ce se evaluează şi se afişează. expresiei.
`default`	opţional	Cea ce se afişează în cazul în care expresia nu poate fi evaluată.
`escapeXml`	opţional	`true / false.` Valoarea implicită este `true`. Pe `false` interpretează caracterele din `value` ca şi cod html.

- `c:import` Permite includerea altor pagini JSP în pagina curentă.

  Atribute ale marcajului:

Atribut	Fel	Descriere
`url`	obligatoriu	Adresa documentului importat.
`context`	opţional	Context-ul paginii / documentului importat. Simbolul /, urmat de numele unei aplicaţii de pe acelaşi server.
`var`	opţional	Numele variabilei în care va fi stocat documentul importat.
`scope`	opţional	Domeniul de valabilitate al variabilei `var`. Unul din valorile: `page, request, session, application.`

Cu marcajul `c:param` se pot fixa parametri pentru pagina importată. Acest marcaj are două atribute `name` şi `value`. Acesti parametri se transmit cu metoda `get`.

- `c:redirect` Redirectarea activitatea către o altă pagină.

  Atribute ale marcajului:

Atribut	Fel	Descriere
`url`	obligatoriu	Adresa paginii către care se face redirectarea.
`context`	opţional	Context-ul paginii către care se face redirectarea. Simbolul /, urmat de numele unei aplicaţii de pe acelaşi server.

  Prin redirectare, parametrii nu sunt retransmişi automat mai departe.

- `c:url` Reţine adrese URL.

  Atribute ale marcajului:

Atribut	Fel	Descriere
`value`	obligatoriu	Adresa documentului de reţinut.
`context`	opţional	Context-ul documentului. Simbolul /, urmat de numele unei aplicaţii de pe acelaşi server.
`var`	opţional	Numele variabilei în care va fi stocată adresa documentului.
`scope`	opţional	Domeniul de valabilitate al variabilei `var`. Unul din valorile: `page, request, session, application.`

10.2.2 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 referinţa la baza de date.

  Atribute ale marcajului:

Atribut	Fel	Descriere
`dataSource`	opţional	Referinţa la baza de date
`driver`	opţional	Driver-ul bazei de date
`url`	opţional	url-ul bazei de date
`username`	opţional	nume utilizatorului bazei de date
`password`	opţional	parola de acces la baza de date
`var`	opţional	variabila cu referinţa la baza de date
`scope`	opţional	Domeniul de valabilitate al variabilei `var`.

- `sql:query` O interogare a bazei de date.

  Atribute ale marcajului:

Atribut	Fel	Descriere
`sql`	obligatoriu	Fraza sql
`dataSource`	opţional	Referinţa la baza de date
`startRow`	opţional	Linia de la care se începe interogarea
`maxRows`	opţional	Numărul maxim de rezultate acceptate
`var`	obligatoriu	Variabila cu rezultatele interogării bazei de date
`scope`	opţional	Domeniul de valabilitate al variabilei `var`.

- `sql:update` Actualizarea bazei de date.

  Atribute ale marcajului:

Atribut	Fel	Descriere
`sql`	obligatoriu	Fraza sql
`dataSource`	opţional	Referinţa la baza de date

Exemplul 10.2.5 *Să se afişeze 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="rezult"
13       sql="select * from adrese" />
```

```
14      <c:if test="${rezult.rowCount gt 0}" >
15        <table>
16         <tr>
17            <c:forEach items="${rezult.columnNames}" var="col">
18              <th>
19                <c:out value="${col}" />
20              </th>
21            </c:forEach>
22          </tr>
23          <c:forEach items="${rezult.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>
```

10.3 Marcaje JSP personale

Programatorul poate crea marcaje JSP proprii care se grupează în colecţii numite biblioteci de marcaje. O bibliotecă de marcaje este reprezentată de un identificator, pe care-l vom denumi identificatorul bibliotecii de marcaje.

10.3.1 Marcaje fără atribute şi fără corp.

Pentru a crea unui asemenea marcaj JSP propriu este necesară definirea următorelor componente:

1. O clasă de definiţie a comportamentului marcajului JSP *(tag handler class).*

2. Descriptorul bibliotecii de marcaje JSP, care leagă clasa de definiţie a marcajului cu identificatorul bibliotecii de marcaje. Acest descriptor este un fişier cu extensia `tld`. Serverul Web va depista descriptorul bibliotecii de marcaje în catalogul aplicaţiei.

3. Fişierul JSP ce utilizează marcajul JSP (clientul).

Exemplificăm acestă tehnologie prin

Exemplul 10.3.1 *Să se realizeze un marcaj dateTag, a cărui efect să fie afişarea datei calendaristice.*

1. Clasa de definiţie a comportamentului marcajului. Programul constă din:

   (a) Importul pachetelor

   ```
   import javax.servlet.jsp.*;
   import javax.servlet.jsp.tagext.*;
   ```

   Aceste pachete se găsesc în fişierul `jsp-api.jar`.

   (b) Un marcaj fără atribute şi fără corp trebuie să extindă clasa `TagSupport` şi să suprascrie metoda `doStartTag`, care defineşte 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 ieşire se face cu un obiect `JspWriter`, care se obţine cu `pageContext.getOut()`. Metoda `print` a clasei `JspWriter` poate genera o excepţie `IOException`.

   Textul sursă al clasei de definiţie 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 fişier trebuie să aibă extensia `tld` (Taglib Language Definition).

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- a tag library descriptor -->
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd"
    version="2.0">

  <tlib-version> 1.0 </tlib-version>
  <jsp-version> 2.0 </jsp-version>
  <short-name> mytagslibrary </short-name>
  <uri>mytags</uri>
  <description> Librarie de marcaje </description>

  <tag>
     <name> dateTag </name>
     <tag-class> jsp.DateTag </tag-class>
     <body-content> empty </body-content>
     <description> furnizeaza data curenta </description>
  </tag>
</taglib>
```

Elementul *uri* conţine identificatorul bibliotecii de marcaje, *mytags.*

Pentru fiecare marcaj propriu se completează un marcaj `tag`. Pentru un marcaj propriu fără atribute elementele acestui marcaj sunt

(a) `name` Numele simbolic al marcajului.

(b) `tag-class` Referinţa la fişierul class al clasei de definiţie a comportamentului marcajului propriu. Referinţa 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

$$<prefix : NumeMarcaj/>$$

Referinţa la identificatorul bibliotecii de marcaje şi prefixul se fixează în directiva `taglib`.

Un fişier *jsp* care utilizează marcajul realizat este *(dateTag.jsp)*:

```
<html>
  <head>
     <title>
        Tag pentru data calendaristica curenta
     </title>
  </head>
  <body>
     <%@ taglib uri="mytags"
       prefix="mk" %>
     <p>
     Data curenta este:
     <mk:dateTag/>
  </body>
</html>
```

apelat din

```
<html>
  <body>
    <form method="get"
      action="jsp/dateTag.jsp">

      Data calendaristic&#259;:
      <p><input type="submit" value="Afiseaza">
    </form>
  </body>
</html>
```

10.3.2 Marcaje cu atribute şi fără corp.

Realizăm un marcaj *ziuaTag* cu un atribut *ziua* care va fi afişat în momentul prelucrării marcajului.

1. Pentru fiecare atribut clasa ce defineşte acţiunea marcajului trebuie să conţină o metodă

   `public void set`*NumeAtribut*`(String value){...}`

   care preia valoarea atributului dată de parametrul *value.*

   Exemplul 10.3.2

   Pentru exemplul enunţat această clasă 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 ZiuaTag extends TagSupport{
7    String ziua;

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 defineşte un marcaj <`attribute`>...< /`attribute`> având incluse marcajele

Nume marcaj	Semnificaţie	Fel
`name`	numele atributului	obligatoriu
`required`	`true | false` după cum atributul e obligatoriu sau nu	obligatoriu
`rtexprvalue`	`true | false` după cum atributul se poate utiliza într-o expresie < %= *numeAtribut* % >	opţional

   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

```
<html>
  <body>
    <form method="get"
      action="jsp/ziuaTag.jsp">
      <p>Astazi, este
      <select name="ziua">
         <option value="luni"> Luni
         <option value="marti"> Marti
         <option value="miercuri"> Miercuri
         <option value="joi"> Joi
         <option value="vineri"> Vineri
         <option value="simbata"> Simbata
         <option value="duminica"> Duminica
      </select>
      <p><input type="submit" value="Afiseaza">
    </form>
  </body>
</html>
```

unde *ziuaTag.jsp* este

```
<html>
  <head>
    <title> Tag cu marcaj </title>
  </head>
  <body>
    <%@ taglib uri="mytags"
      prefix="mk" %>
    <p>
    <%
      String zi=request.getParameter("ziua");
    %>
    Ziua este:
    <mk:ziuaTag ziua="<%= zi %>" />
  </body>
</html>
```

10.3.3 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 doreşte ca marcajul să execute acţiuni după interpretarea corpului, atunci acele activităţi sunt definite în metoda `doEndTag`. Această metodă returnează valoarea `EVAL_PAGE` sau `SKIP_PAGE` după cum se doreşte sau nu continuarea procesării paginii jsp.

Exemplul 10.3.3 *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 marcaje 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

```
<html>
  <body>
    <form method="get"
      action="jsp/modtextTag.jsp">
      Introduce o fraz&#259;
      <input type="text" name="text" size="40" >
      <p>
      Se transform&#259; &#238;n litere
      <select name="trans">
        <option value="upperCase"> mari
        <option value="lowerCase"> mici
      </select>
      <p><input type="submit" value="Afiseaza">
    </form>
  </body>
</html>
```

împreună cu *modtextTag.jsp*

```
<html>
  <body>
    <%@ taglib uri="mytags" prefix="mk" %>
    <%
      String text=request.getParameter("text");
      String trans=request.getParameter("trans");
      String t;
      if(trans.equals("upperCase"))
        t="true";
      else
        t="false";
    %>
    <p>
    <mk:modTag text="<%= text %>" trans="<%=t%>">
        <mk:dateTag/>
    </mk:modTag>
  </body>
</html>
```

10.4 *Apache-tiles*

Apache-tiles (*tiles*) este un cadru de lucru pentru realizarea de interfeţe grafice pentru aplicaţii Web, prin *cărămizi*. O *cărămidă* corespunde unui dreptunghi în fereastra atribuită aplicaţiei Web din navigator. *Tiles* a fost dezvoltat iniţial pentru *Struts,* dar poate fi utilizat şi în cazul unui servlet sau JSP.

Ideea cadrului de lucru *tiles* este crearea şi utilizarea de elemente reutilizabile în cadrul unei aplicaţii sau în aplicaţii distincte.

O *cărămidă* este umplută de reprezentarea dată de o pagina HTML sau JSP, numită în continuare componentă.

Punctul de plecare este dat de un şablon - un deptunghi umplut / acoperit de dreptunghiuri. Acele dreptunghiuri corespund *cărămizilor.*

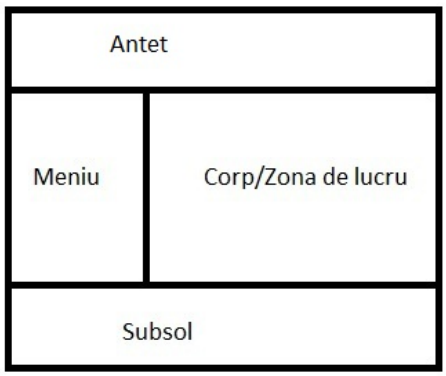
Astfel şablonul şi componentele JSP sunt elementele reutilizabile.

Instalarea înseamnă dezarhivarea resursei descărcate din Internet. Se utilizează toate fişierele `jar` ale distribuţiei.

10.4.1 *Tiles* în servlet şi JSP

Definirea unui şablon

Pentru interfaţa grafică *clasică*



codul şablonul poate fi (*template.jsp*)

```
1 <%@ taglib uri="http://tiles.apache.org/tags-tiles" prefix="tiles" %>
2 <html>
3   <body style="width:100%;height:100%">
4     <table border="1" cellspacing="0" cellpadding="0"
5         style="width:100%;height:100%">
6       <tr>
7         <td colspan="2">
```

```
8           <tiles:insertAttribute name="antet" />
9         </td>
10      </tr>
11      <tr>
12        <td>
13          <tiles:insertAttribute name="meniu" />
14        </td>
15        <td>
16          <tiles:insertAttribute name="corp" />
17        </td>
18      </tr>
19    <tr>
20      <td colspan="2">
21        <tiles:insertAttribute name="subsol" />
22      </td>
23    </tr>
24    </table>
25  </body>
26 </html>
```

Umplerea *cărămizilor* definite în şablon - în cazul exemplului de mai sus *antet, meniu, corp, subsol* - cu componentele JSP se specifică într-un fişier de configurare `tiles.xml`:

```
1 <?xml version="1.0" encoding="ISO-8859-1" ?>
2 <!DOCTYPE tiles-definitions PUBLIC
3         "-//Apache Software Foundation//DTD Tiles Configuration 2.0//EN"
4         "http://tiles.apache.org/dtds/tiles-config_2_0.dtd">
5 <tiles-definitions>
6   <definition name="emptyPage" template="/template.jsp">
7     <put-attribute name="antet" value="/myHeader.jsp" />
8     <put-attribute name="meniu" value="/myMenu.jsp" />
9     <put-attribute name="corp" value="/empty.jsp" />
10    <put-attribute name="subsol" value="/myFooter.jsp" />
11  </definition>
12  . . .
13 </tiles-definitions>
```

Exemplul 10.4.1 *Aplicaţia HelloServlet şi hello.jsp apelate dintr-o interfaţă grafică tiles bazată pe şablonul definit anterior.*

Apelarea aplicaţiei Web, prin *index.jsp,*

```
1 <%@ taglib uri="http://tiles.apache.org/tags-tiles" prefix="tiles" %>
2 <tiles:insertDefinition name="emptyPage" />
```

încarcă şablonul cu componentele:

- *myHeader.jsp*

```
1 <div>
2 This is the default header
3 <h1> HelloServlet and hello.jsp through Tiles </h1>
4 </div>
```

- *myMenu.jsp*

```
<div>
<ul>
  <li><a href="/mytiles/servlet.jsp">HelloServlet</a></li>
  <li><a href="/mytiles/jsp.jsp">Hello jsp</a></li>
</ul>
</div>
```

- *empty.jsp*

```
<div></div>
```

- *defaulFooter.jsp*

```
<div>This is the default footer...</div>
```

Prin intermediul fişierelor *servlet.jsp* şi *jsp.jsp* se comandă reutilizarea şablonului încărcat iniţial cu ansamplul dat de *emptyPage.*

- *servlet.jsp*

```
<%@ taglib uri="http://tiles.apache.org/tags-tiles" prefix="tiles" %>
<tiles:insertDefinition name="servletPage" />
```

- *jsp.jsp*

```
<%@ taglib uri="http://tiles.apache.org/tags-tiles" prefix="tiles" %>
<tiles:insertDefinition name="jspPage" />
```

servletPage şi *jspPage* apar în *tiles.xml.* Codul complet este

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE tiles-definitions PUBLIC
       "-//Apache Software Foundation//DTD Tiles Configuration 2.0//EN"
       "http://tiles.apache.org/dtds/tiles-config_2_0.dtd">
<tiles-definitions>
  <definition name="emptyPage" template="/template.jsp">
    <put-attribute name="antet" value="/defaultHeader.jsp" />
    <put-attribute name="meniu" value="/defaultMenu.jsp" />
    <put-attribute name="corp" value="/empty.jsp" />
    <put-attribute name="subsol" value="/defaultFooter.jsp" />
  </definition>

  <definition name="servletPage" template="/template.jsp">
    <put-attribute name="antet" value="/defaultHeader.jsp" />
    <put-attribute name="meniu" value="/defaultMenu.jsp" />
    <put-attribute name="corp" value="/form.html" />
    <put-attribute name="subsol" value="/defaultFooter.jsp" />
  </definition>

  <definition name="jspPage" template="/template.jsp">
    <put-attribute name="antet" value="/defaultHeader.jsp" />
```

```
22      <put-attribute name="meniu" value="/defaultMenu.jsp" />
23      <put-attribute name="corp" value="/hello.jsp" />
24      <put-attribute name="subsol" value="/defaultFooter.jsp" />
25    </definition>
26  </tiles-definitions>
```

sau mai elegant, cu precizarea doar a diferenţelor faţă de o componentă de *bază*

```
1  <?xml version="1.0" encoding="ISO-8859-1" ?>
2  <!DOCTYPE tiles-definitions PUBLIC
3          "-//Apache Software Foundation//DTD Tiles Configuration 2.0//EN"
4          "http://tiles.apache.org/dtds/tiles-config_2_0.dtd">
5  <tiles-definitions>
6    <definition name="emptyPage" template="/template.jsp">
7      <put-attribute name="antet" value="/defaultHeader.jsp" />
8      <put-attribute name="meniu" value="/defaultMenu.jsp" />
9      <put-attribute name="corp" value="/empty.jsp" />
10     <put-attribute name="subsol" value="/defaultFooter.jsp" />
11   </definition>

13   <definition name="servletPage" extends="emptyPage">
14     <put-attribute name="corp" value="/form.html" />
15   </definition>

17   <definition name="jspPage" extends="emptyPage">
18     <put-attribute name="corp" value="/hello.jsp" />
19   </definition>
20 </tiles-definitions>
```

Valoarea atributului `value` indică elementul care umple câmpul specificat de atributul `name`, definit în şablon.

Acţiunea are loc în urma unui clic pe ancorele din meniu. *Cărămida corp* se încarcă cu fişierele de apelare a aplicaţiilor concrete, respectiv *form.html* şi *hello.jsp.*

- *form.html*

```
1  <html>
2    <head>
3      <title> Servlet-ul Hello </title>
4    </head>
5    <body>
6      <center>
7      <h1> Pagina de apelare a servletului HelloServlet </h1>
8      <form method="post"
9            action="hello">
10       <p>Introduceti numele:
11       <input type="text" name="name" size=20>
12       <p>
13       <input type="submit" value="Calculeaza">
14       <input type="hidden" name="tip" value="text/html" >
15     </form>
16     </center>
17   </body>
18 </html>
```

- *hello.jsp*

```
<html>
  <head>
    <title> jsphello </title>
  </head>
  <body>
    <center>
    <form method="post">
      <p> <h3> Introduceti numele: </h3>
      <input type="text" name="name" size=20>
      <p>
      <input type="submit">
    </form>
    <p>
    <%
       String nume=request.getParameter("name");
       out.println("Hi "+nume+" !");
    %>
    </center>
  </body>
</html>
```

Utilizarea lui *tiles* de către serverul Web este declarată în `web.xml`.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                       http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
   version="3.1">

  <listener>
    <listener-class>
       org.apache.tiles.extras.complete.CompleteAutoloadTilesListener
    </listener-class>
  </listener>

  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

Desfăşurarea aplicaţiei este

```
mytiles
|--> WEB-INF
|    |--> classes
|    |    |     HelloServlet.class
|    |--> lib
|    |    |     *.jar
|    |    tiles.xml
|    |    web.xml
|    myFooter.jsp
|    myHeader.jsp
|    myMenu.jsp
|    empty.jsp
|    form.html
```

```
|    hello.jsp
|    index.jsp
|    jsp.jsp
|    servlet.jsp
|    template.jsp
```

10.5 Autentificare şi autorizare cu *apache-shiro*

Apache-shiro este un produs care permite autentificarea şi autorizarea unei aplicaţii informatice ca o entitate independentă de aplicaţia în cauză. Datele de autentificare sunt înregistrate într-un fişier `shiro.ini`.

Un utilizator este definit de perechea (*nume_utilizator, parola*), îi sunt atribuite unul sau mai multe *roluri* iar unui rol i se atribuie unul sau mai multe *acţiuni* -(*permission* - în terminologia *apache-share*).

Apache-shiro este dezvoltat ca un filtru din tehnologia servlet.

În momentul de faţă nu este posibilă actualizarea dinamică a fişierului `shiro.ini`.

Termeni

- *Credential* informaţie 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 reţine datele de identificare a unui utilizator;
- *Subject* termen utilizat pentru un utilizator (om sau program).

Fişierul de configurare `shiro.ini` pentru aplicaţie Web conţine:

1. `[main]` shiro.loginUrl = /login.jsp

   Se indică pagina de autentificare.

2. `[users]`

   Declararea utilizatorilor împreună cu parola de autentificare şi *rolul / rolurile.* Un rol fixează activităţile (*permissions*) de care dispune utilizatorul.

   Sintaxa utilizată este
   numeUtilizator = parola, rol_1,rol_2,...

3. `[roles]`

   Se declară activităţile 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 referinţele din serverul Web la care se asigură accesul doar în urma autentificării, care sunt *filtrate* de *apache-shiro.*

   Sintaxa utilizată este
   /fişier.jsp sau *catalog/*** = `authc`

   Deconectarea se indică prin
   /logout = `logout`

Exemplul 10.5.1

```
1  [main]
2  shiro.loginUrl = /login.jsp

4  [users]
5  # format: username = password, role1, role2, ..., roleN
6  admin = admin,admin
7  guest = guest,guest
8  cmmdc=cmmdc,rolCmmdc
9  hello=hello,rolHello

11 [roles]
12 # format: roleName = permission1, permission2, ..., permissionN
13 admin = all
14 rolCmmdc=cmmdc
15 rolHello=hello

17 [urls]
18 /login.jsp = authc
19 /logout = logout
20 /accesAutorizat/** = authc
```

Structura unei aplicaţii Web cu indicarea resurselor pentru autentificare şi autorizare este

```
catalogul_aplicatiei
|--> accesAutorizat
|      |      alegeActiune.jsp
|      |      fisiere html/jsp de apelare ale aplicatiilor
|--> WEB-INF
|      |--> lib
```

```
|      |      |--> *.jar
|      |      shiro.ini
|      |      web.xml
|      index.html
|      home.jsp
|      login.jsp
```

Resursele `jar` necesare sunt:

`commons-beanutils-*.jar`	`shiro-web-*.jar`
`jcl-over-slf4j-*.jar`	`jstl.jar`
`standard.jar`	`slf4j-api-*.jar`
`slf4j-simple-*.jar`	`shiro-care-*.jar`

Fişierul `web.xml` este

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app 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/web-app_2_5.xsd"
  version="2.5">
    <listener>
        <listener-class>
          org.apache.shiro.web.env.EnvironmentLoaderListener
        </listener-class>
    </listener>

    <filter>
        <filter-name>ShiroFilter</filter-name>
        <filter-class>
          org.apache.shiro.web.servlet.ShiroFilter
       </filter-class>
    </filter>

    <filter-mapping>
        <filter-name>ShiroFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
</web-app>
```

iar fişierul `index.html` are codul

```
<html>
  <head>
    <META HTTP-EQUIV="Refresh" CONTENT="0;URL=home.jsp">
  </head>
  <body>
    <p>Apelarea aplicatiei...</p>
  </body>
</html>
```

Valoarea atributului `URL`, în cazul de faţă *home.jsp* defineşte pagina de deschidere a aplicaţiei.

În acest fişier se utilizează marcajele bibliotecii `prefix=shiro, url="http://shiro.apache.org/tags"`. Activităţile cuprinse sunt:

- solicitarea autentificării;
- în cazul autentificarii se indică prin ancore (link) posibilităţile de navigare (spre aplicaţia propriu-zisă sau deconectare (logout))
- opţional, se vor putea determina rolurile şi acţiunile permise utilizatorului autentificat, adică autorizarea.

Biblioteca de marcaje `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=`*"acţiune"*`>`

  Execută marcajul interior dacă rolului îi este atribuit activitatea *acţiune.*

- `<shiro:hasPermission name=`*"acţiune"*`>`

  Execută marcajul interior dacă rolului nu îi este atribuit activitatea *acţiune.*

Exemplul 10.5.2 *Aplicaţie Web utilizând fişierul* `shiro.ini` *cu acces către aplicaţiile cmmdc1pagina.jsp şi hello1pagina.jsp.*

Se definesc trei clienţi admin, cmmdc şi hello. Administratorul va avea acces la ambele aplicaţii dar clientul cmmdc / hello va avea acces doar la aplicaţia cmmdc1pagina / hello1pagina.

Structura aplicaţiei şi fişierul *shiro.ini* sunt cele date mai sus.
Fişierul `home.jsp` este

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

<html>
  <body>
    <h1>Autentificare (Apache Shiro) </h1>

    <p>Salut
    <shiro:guest>Guest</shiro:guest>
    <shiro:user><shiro:principal/></shiro:user>!

    <shiro:user>
      <table>
        <tr>
          <td>
            <a href="<c:url
                value="/accesAutorizat/alegeActiune.jsp"/>">
                Ac&#355;iuni</a>
          </td>
        </tr>
        <tr>
          <td>
            <a href="<c:url value="/logout"/>">Log out</a>
          </td>
        </tr>
      </table>
    </shiro:user>

    <shiro:guest>
      <a href="<c:url value="/login.jsp"/>">Log in</a>
    </shiro:guest>

    <p/>
    <shiro:hasRole name="admin">admin<br/>
      <%
        session.setAttribute("rol","admin");
      %>
    </shiro:hasRole>
    <shiro:hasRole name="rolCmmdc">rolCmmdc<br/>
      <%
        session.setAttribute("rol","rolCmmdc");
      %>
    </shiro:hasRole>
    <shiro:hasRole name="rolHello">rolHello<br/>
      <%
        session.setAttribute("rol","rolHello");
      %>
    </shiro:hasRole>

    <p/>
    <h3>Roluri pe care nu le ave&#355;i</h3>
    <p/>
      <shiro:lacksRole name="admin">admin<br/></shiro:lacksRole>
      <shiro:lacksRole name="rolCmmdc">rolCmmdc<br/></shiro:lacksRole>
      <shiro:lacksRole name="rolHello">rolHello<br/></shiro:lacksRole>
```

```
57      <p/>
58      <h3>Activit&#259;&#355;ile d-voastr&#259;</h3>
59      <p>
60        <shiro:hasPermission name="all">all<br/>
61          <%
62            session.setAttribute("act","all");
63          %>
64        </shiro:hasPermission>
65        <shiro:hasPermission name="cmmdc">cmmdc<br/>
66           <%
67            session.setAttribute("act","cmmdc");
68          %>
69        </shiro:hasPermission>
70        <shiro:hasPermission name="hello">hello<br/>
71          <%
72            session.setAttribute("act","hello");
73          %>
74        </shiro:hasPermission>

76      <p/>
77      <h3>Activit&#259;&#355;i de care nu dispune&#355;i</h3>
78      <p/>
79        <shiro:lacksPermission name="cmmdc">cmmdc<br/></shiro:lacksPermission>
80        <shiro:lacksPermission name="hello">hello<br/></shiro:lacksPermission>
81    </body>
82  </html>
```

Fişierul `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>
```

Fişierul `alegeActiune.jsp` este

```
1  <html>
```

```
2  <body bgcolor="#aaeeaa">
3    <table>
4      <%
5        String act=(String)session.getAttribute("act");
6        if(act=="all"){
7      %>
8      <tr>
9        <td>
10         <a href="alege.html">Actiuni administrator</a>
11       </td>
12     </tr>
13     <%
14       }
15       if(act=="cmmdc"){
16     %>
17       <tr>
18         <td>
19           <a href="cmmdc.html">Calcul cmmdc</a>
20         </td>
21     </tr>
22     <%
23       }
24       if(act=="hello"){
25     %>
26       <tr>
27         <td>
28           <a href="hello.html">Aplicatia Hello Name</a>
29         </td>
30     </tr>
31     <%
32       }
33     %>
34    </table>
35 </body
36 </html>
```

Aplicaţia propriu-zisă (*cmmdc1pagina.jsp*) se completează la sfârşit cu

```
<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 şi pentru *hello1pagina.jsp.*

10.6 Aplicaţie 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 aplicaţiei JSP constă din

1. Generarea cadrului:

```
set GroupID=jsp
set ArtifactID=cmmdcjsp
set Version=1.0
mvn -B archetype:generate
  -DgroupId=%GroupID%
  -DartifactId=%ArtifactID%
  -Dversion=%Version%
  -DarchetypeArtifactId=maven-archetype-webapp
```

2. Se completează aplicaţia cu fişierele *CmmdcBean.java, cmmdc.jsp* şi *index.html* din cursul Programare distribuită, desfăşurarea fiind

```
cmmdcjsp
|--> src
|    |--> main
|    |    |--> java
|    |    |    |     CmmdcBean.java
|    |    |--> resources
|    |    |--> webapp
|    |    |    |--> jsp
|    |    |    |    |   cmmdc.jsp
|    |    |    |--> WEB-INF
|    |    |    |    |   web.xml
|    |    |    index.html
|    pom.xml
```

3. Fişierul `pom.xml` se completează cu referinţa 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`.

Întrebări recapitulative

1. Care este tipul tehnologiei JSP ?

2. Precizaţi diferenţa dintre o pagină JSP şi un document JSP.

3. Unde se instalează o pagina JSP?

4. Care este rolul unei declaraţii JSP ?

5. Care este rolul unei directive JSP ?

6. Cum apelează o pagină JSP ?

7. Cum prelucrează un server Web o pagina / document JSP ?
8. Care sunt trăsăturile unei componente Java (bean) ?
9. Care este rolul unui element <jsp:useBean> ?

Capitolul 11

Microservicii Java

MicroProfile defineşte o arhitectură de aplicaţie - microserviciu - care simplifică desfăşurarea în nor. *MicroProfile* este coordonat de Eclipse şi defineşte un cadru standardizat pentru microservicii. Bazat pe Jaca SE 8 într-un microserviciu se integrază tehnologiile:

- *Context and Dependency Injection for Java* (CDI);
- *Java API for RESTful Web Services* (JAX-RS);
- *Java API for JSON Processing* (JSON-P);
- *Common Annotations for the Java Platform.*

Un microserviciu este o aplicaţie mică, componentă a unei aplicaţii mai mari, componente care comunică prin intermediul unei interfeţe comune.

O aplicaţie monolitică se vrea descompusă în componente - microserviciile - independente, cu propriul ciclu de evoluţie şi care comunică prin intermediul unei interfeţe comune.

Microserviciul simplifică modul de folosire a unei aplicaţii / serviciu Web.

Un microserviciu se caracterizează prin:

- Execuţia nu presupune desfăşurarea într-un server de aplicaţie sau Web (*container-less, out of the box*). Forma finală a aplicaţiei este o arhivă. Pentru o arhivă `jar` execuţia revine la

  ```
  java -jar microserviciu.jar . . .
  ```

- Arhiva conţine toate resursele necesare aplicaţiei (*self-contained*). Astfel arhiva poate avea o dimensiune mare (*fat jar/war deployment*)[1].

[1] Arhivarea întregii aplicaţii se poate face cu `jar`. Pentru o asemenea arhivă se foloseşte terminologia *über-jar*.

- Potrivit practicii este nevoie de JEE, caz în care se preferă utilizarea unui container - de exemplu *Docker* - care să asigure cadrul de execuţie al microserviciului (*in-container*).

11.1 Payara Micro

Microserviciul este dat de `payara-micro-*.jar` şi are la bază `payara / glassfish`. Tehnologiile JEE suportate sunt servlet, Java Server Pages (JSP), websocket, Java Server Faces (JSF), JAX-RS (jersey), JAX-WS (jaxws-ri).

Aplicaţia Web se dezvoltă şi se apelează în mod obişnuit, nefiind integrată în microserviciul *payara,* care-l desfăşoară dinamic în timpul execuţiei.

Lansarea unei aplicaţii în execuţie:

- în linie de comandă (*Command Line Interface* - CLI)

```
java -jar payara-micro-*.jar --deploy cale/app.war

java -jar payara-micro-*.jar --deploy cale1/app1.war --deploy cale2/app2.war . . .

java -jar payara-micro-*.jar --deploymentDir cale
```

- programat

```
1  import fish.payara.micro.BootstrapException;
2  import fish.payara.micro.PayaraMicro;

4  public class EmbeddedPayara {
5    public static void main(String[] args) throws BootstrapException{
6        String fileName="";
7        if(args.length==0){
8          System.out.println("Usage: java EmbeddedPayara fileName\n");
9        }
10       else{
11         fileName=args[0];
12         System.out.println(fileName);
13         PayaraMicro.getInstance()
14           .addDeployment(fileName)
15           .bootStrap();
16       }
17     }
18 }
```

Probleme

- Utilizare *tomee embedded, wildfly-swarm, spring boot, dropwizard, bootique.*

- *kumuluzee*: Pathparam, bean.
- *kumuluzee*: Apelarea unui serviciu REST în linie de comandă.

Capitolul 12

Desfăşurarea în *nor*

Dezvoltarea Internetului, nevoia de a reduce costurile legate de realizarea şi întreţinerea infrastructurii care oferă servicii pe Internet, concomitent cu nevoia de creştere a calităţii serviciilor a condus la *servicii în nor* (*Cloud Computing*).

Avantajele oferite de serviciile *serviciile în nor* sunt:

- Reducerea costurilor
- Agilitate (*Agility*)

  Reducerea duratei:

  - de aşteptare în cazul apariţiei unei disfuncţionalităţi din partea furnizorului *serviciului în nor;*
  - de actualizare şi întreţinere din partea realizatorului *serviciului în nor.*

- Elasticitate (*Elasticity*)

  Posibilitatea de creştere / descreştere a resurselor (în principal hard) alocate pentru a satisface cerinţele clienţilor într-un interval de timp.

  Se face distincţie de *scalabilitate,* termen care desemnează nevoia de creşere / descreştere a resurselor alocate legată de dezvoltarea aplicaţiilor care compun serviciul.

Tipuri de *servicii în nor:*

- Aplicaţii 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)
- Plaformă ca serviciu (*Platform as a Service - PaaS*)
  PaaS poate fi
  - Ne-portablă : aplicaţia va avea o structură predefinită.
    Google AppEngine (GAE), *Microsoft Azure*, *OpenShift*
  - Portabilă
    Heroku

În cele ce urmează ne interesează doar platformele PaaS care acceptă desfăşurarea de aplicaţii Java, în mod gratuit, sau oferă un simulator local.

12.1 Servlet şi JSP în *Google App Engine*

Google AppEngine permite:

- încărcarea unei aplicaţii Web pe un simulator local al platformei de Cloud Computing;
- încărcarea unei aplicaţii Web pe platforma Google de Cloud Computing.

În prezent, pe platforma GAE se pot încărca aplicaţii realizate în Java, Python, PHP şi Go, alături de care care pot apărea fişiere *http, css, js.* Există câte o distribuţie 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 fişierului *appengine-java-sdk-*.*

Utilizarea. În vederea încărcării unui servlet pe simulatorul local se crează structura de cataloage şi fişiere

```
war
|--> WEB-INF
|    |-->  classes
|    |     |  *.class
|    |-->  lib
|    |     |  *.jar
|    |     web.xml
|    |     appengine-web.xml
|    index.html
```

Singurul fişier specific GAE este *appengine-web.xml.* Codul acestui fişer este

```
<appengine-web-app xmlns="http://appengine.google.com/ns/1.0">
  <!-- Replace this with your application id from
       http://appengine.google.com -->
  <application></application>

  <version>1</version>

</appengine-web-app>
```

Datele fişierului *web.xml* corespund servlet-ului, iar prin fişierul *index.html* se apelează aplicaţia Web. Parametrul `action` al elementului `form` are forma simplificată `action=/`*numeApel,* unde *numeApel* coincide cu valoarea atributului `urlPattern.`

Dacă aplicaţia se încarcă pe platforma *Google* de *Cloud Computing* atunci trebuie completat elementul <`application`>.

Lansarea simulatorului şi încărcarea se poate face prin comenzile

```
set GAE\_HOME=. . .\appengine-java-sdk-*
%GAE\_HOME%\bin\dev_appserver war
```

lansate într-o fereastă DOS, în catalogul care-l conţine pe *war.* Aplicaţia se apelează prin `http://localhost:8080.` Dacă în loc de *index.html* se utilizează alt nume, atunci apelarea aplicaţiei este
`http://localhost:8080/fişier.html.`

O aplicaţie JSP se tratează asemănător.

Distribuţia GAE pentru Java conţine şablonul unei aplicaţii împreună cu un fişier `build.xml` (`appengine-java-sdk-*\demos\new_project_template`) prin intermediul căruia, cu ajutorul lui *ant,* se construieşte catalogul *war* descris anterior.

Exemplul 12.1.1 *Servlet-ul CmmdcServlet instalat în platforma Google App Engine de Cloud Computing.*

Şablonul se copiază în zona de lucru sub numele *appcmmdc* şi se completează cu fişierele 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 *appcmmdc.* Deoarece numele fişierului html diferă de *index*, aplicaţia se apelează prin
`http://localhost:8080/cmmdc.html`.

Alternativ, aplicaţia se poate construi cu *Eclipse* folosind o componentă (*plug-in*) specifică.

GAE conţine în plus un serviciu de autentificare şi autorizare *UserService,* un sistem de persistenţă a datelor *Datastore*, *Task-Queue.*

Desfăşurarea în GAE

Acest pas necesită din partea dezvoltatorului cont Google.
Desfăşurarea gratuită (cel mult 10 aplicaţii) în GAE presupune:

1. Înregistrarea aplicaţiei:
   (a) Se accesează pagina Web https://appengine.google.com/.
   (b) Înregistrarea propriu-zisă:
      i. Application Identifier
      Se fixează *identificatorul aplicaţiei* – *app_id,* care se trece şi în elementul <application> din `appengine-web.xml;`
      ii. Check Availibility
      Verificarea disponibilităţii identificatorului;
      iii. Application title
      Fixarea titlului aplicaţiei;
      iv. Create Application
2. Încărcarea în nor (upload)

```
%GAE_HOME%\bin\appcfg.cmd --oauth2 update locatia_aplicatiei_web
```

   Prin *locatia_aplicatiei_web* se înţelege catalogul `war` construit de GAE.

Aplicaţia va fi disponibilă prin
http://app_id.appspot.com/ sau
http://app_id.appspot.com/fisier.html

12.2 *Heroku*

Heroku este o PaaS care oferă suport pentru mai aplicaţii dezvoltate în mai multe limbaje de programare, printre care şi 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 Toolbeit.* Cu acest prilej se instalează *Ruby* şi *Git* în `c:\Program Files (x86).`[1] *Ruby* se instalează în *Heroku.* Variabila de sistem `PATH` se actualizează cu căile la *Heroku* şi *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 aplicaţiilor se realizează la https://dashboard.heroku.com/apps.

Dezvoltarea aplicaţiilor se bazează pe *maven* iar desfăşurarea pe *Git.*

12.2.1 JSP în *Heroku*

1. Generarea aplicaţiei:

   ```
   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 fişierului `pom.xml`:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
```

[1] 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 aplicaţiei cu fişierele `jsp`.

4. Opţional aplicaţia se poate arhiva şi verifica.

   `mvn clean package`

5. Completarea cu fişierele

   - `Procfile`

```
web: java $JAVA_OPTS -jar target/dependency/jetty-runner.jar
--port $PORT target/*.war
```

- `system.properties`

```
java.runtime.version=1.7
```

- `.gitignore`

```
target
```

Structura aplicaţiei 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 şi desfăşurarea aplicaţiei

```
heroku create
git push heroku master
```

Heroku atribuie un nume aplicatiei. Dintr-un navigator aplicaţia 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 aplicaţiei la 1 *dynos*. Un număr mai mare de resurse presupune un cost.

9. Lansarea aplicaţiei în linia de comandă din catalogul aplicaţiei

```
heroku open
```

10. Opţional se poate consulta fişierul de jurnalizare

```
heroku logs
```

12.2.2 Servlet în *Heroku*

Aplicaţia servlet se bazează pe interfaţa 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.SESSIONS);
23       context.setContextPath("/");
24       server.setHandler(context);
25       context.addServlet(new ServletHolder(new HelloWorld()),"/*");
26       server.start();
27       server.join();
28   }
29 }
```

Fişierele `html, css` care asigură interfaţa grafică a aplicaţiei servlet nu fac parte din ce se încarcă în nor.

Servletul se apelează prin

`http://`*numeDatDeHeroku*`.herokuapp.com/`*numeApel*

Desfăşurarea şi apelarea constă din

1. Generarea aplicaţiei.
2. Completarea fişierului `pom.xml`:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>myGroupId</groupId>
  <artifactId>myArtifactId</artifactId>
  <packaging>war</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>myArtifactId Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>org.eclipse.jetty</groupId>
      <artifactId>jetty-servlet</artifactId>
      <version>9.1.0.v20131115</version>
    </dependency>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>servlet-api</artifactId>
      <version>2.5</version>
    </dependency>
  </dependencies>
  <build>
    <finalName>myArtifactId</finalName>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.1</version>
        <configuration>
          <source>1.7</source>
          <target>1.7</target>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-dependency-plugin</artifactId>
        <version>2.4</version>
        <executions>
          <execution>
            <id>copy-dependencies</id>
            <phase>package</phase>
            <goals><goal>copy-dependencies</goal></goals>
          </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 aplicaţiei cu fişierele `java` şi web.xml.

4. Opţional aplicaţia se poate arhiva şi verifica.
   Verificarea se face prin

```
set PORT=5000
java -cp myArtifactId\target\classes;"myArtifactId\target\dependency\*"
  clasaServlet
```

5. Completarea cu fişierele

   - `Procfile`

     ```
     web: java $JAVA_OPTS -cp target/classes:target/dependency/*
       clasaServlet
     ```

   - `system.properties`
   - `.gitignore`

   Structura aplicaţiei 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 şi desfăşurarea aplicaţiei

   ```
   heroku create
   git push heroku master
   ```

   Heroku atribuie un nume aplicatiei. Dintr-un navigator aplicaţia se va apela prin acest nume.

8. Calibrare

   ```
   heroku ps:scale web=1
   ```

   cu verificarea calibrării

   ```
   heroku ps
   ```

   Această setare fixează resursele atribuite aplicaţiei la 1 *dynos.* Un număr mai mare de resurse presupune un cost.

9. Dacă se transmit date aplicaţiei printr-un fişier `html` atunci lansarea aplicaţiei se face din navigator completând atributul `action` cu adresa furnizata de heroku iar în caz contrar din linie de comandă se apelează

   ```
   heroku open
   ```

10. Opţional se poate consulta fişierul de jurnalizare

    ```
    heroku logs
    ```

12.3 *OpenShift*

OpenShift este produs de *RedHat* cu suport pentru aplicaţii distribuite în Java dar şi pentru (python, ruby, javascript, php).

Comunicaţiile unei aplicaţii se fac exclusiv potrivit protocoalelor şi porturilor din tabelul următor:

Protocolul	Portul
HTTP	80
HTTPS	443
SSH-*SecureSHell*	22
WebSocket HTTP	8000
WebSocket HTTPS	8443

O aplicaţie aparţine unui cartuş - **cartridge** - dar care poate conţine şi alte componente ale aplicaţiei (de exemplu un SGBD). Unui cartuş i se asociază un angrenaj - **gear** - ce trebuie privit ca un container al aplicaţiei. Angrenajul poate fi mic (512 MB), mediu (1 GB) sau mare (2 GB).

Cartuşul defineşte natura aplicaţiei (servlet sau JSP în tomcat 6 sau 7, aplicaţie JEE, etc).

Un utilizator trebuie să se înregistreze (gratuit) la `www.openshift.com`, moment în care se fixează codul de identificare - o adresă de e-mail - şi o parolă.

Unui dezvoltator i se asociază un *domeniu* - *namespace* - indicat printr-un cuvânt. Apelarea unei aplicaţii se face după schema

`http://`*numeApp-domeniu*`.rhcloud.com`

Gratuit un dezvoltator poate instala până la 3 aplicaţii într-un angrenaj mic.

O aplicaţie 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`

   Locaţia este `RUBY_HOME\include`.

4. Instalarea clientului *OpenShift*

   `rhc setup`

Dezvoltarea unei aplicaţii

1. `rhc app create` *numeApp cartuş*

Cartuşe pentru aplicaţii Java (gratuite)

Denumire cartuş	Conţinut
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 fisierele sursă ale aplicaţiei Web.

Incărcarea în nor

```
git add .
git commit -m "text"
git push
```

Operaţii rhc

- `rhc --help`
- `rhc --help` *comandă_rhc*
- `rhc cartridge list`

  Lista cartuşelor disponibile.
- `rhc app delete` *numeApp*

  Ştergerea aplicaţiei.
- `rhc apps`

  Lista aplicaţiilor.

Întrebări recapitulative

1. Precizati tipurile principale de servicii în nor.
2. Enumerati servicii în nor de tip PaaS cu suport pentru Java.

Capitolul 13

Java Web Start

13.1 Java Web Start

O aplicaţie Java destinată a fi utilizată pe un calculator poate fi valorificată şi într-o reţea, pe baza protocolului *Java Network Launching Protocol - JNLP.* Tehnologia poartă numele de *Java Web Start.*

Aplicaţia Java trebuie să satisfacă o serie de restricţii:

- Aplicaţia trebuie arhivată cu `jar`. În fişierul MANIFEST.MF trebuie să apară atributul

  ```
  Permissions: all-permissions
  ```

- Arhiva `jar` trebuie certificată. Certificarea se realizează cu utilitarele `keytool.exe` şi `jarsigner.exe` din distribuţia jdk;

- Eventualele date necesare aplicaţiei se introduc prin intermediul unei interfeţe grafice (*JavaFX, swing, SWT, apache-pivot*);

- Acces limitat la proprietăţile şi resursele sistemului client.

Aplicaţiei i se ataşează un fişier `xml`, dar cu extensia `jnlp`, care se va apela dintr-un navigator. Pentru acest fişier folosim terminologia de fişier *jnlp.*

Fişierul *jnlp* fixează:

- referinţa URL a aplicaţie (prin atributul `codebase`);

- arhivele `jar` utilizate (prin atributul `href`);

- clasa cu metoda `main` (prin atributul `main-class`).

Ansamblul de resurse formează o aplicaţie Web care poate fi desfăşurată într-un server Web (container de servlet) sau poate fi incorporat într-un servlet.

Arătăm activităţile 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    }

6    private long cmmdc(long m,long n){. . .}

8    private void initComponents() {
9      // cod generat de Netbeans
10   }

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=Long.valueOf(c).toString();
20       cmmdcTextField.setText(s);
21     }
22     catch(Exception e){
23       System.err.println("Exception : "+e.getMessage());
24     }
25   }

27   private void exitForm(java.awt.event.WindowEvent evt){
28     System.exit(0);
29   }

31   public static void main(String args[]) {
32     new VisualCmmdc().setVisible(true);
33   }

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 }
```

Interfaţa grafică a programului este ilustrată în Fig. 13.1

Instalare într-un server Web

Pentru instalarea aplicaţiei într-un server Web, se parcurg paşii:

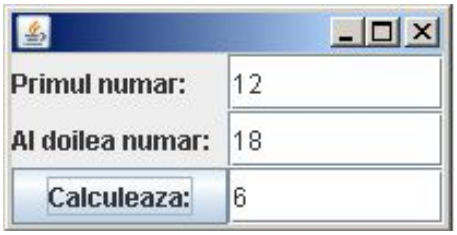


Figure 13.1: Interfaţa grafică a aplicaţiei.

1. Se arhivează aplicaţia

   `jar cmfv myManifest.mf cmmdc0.jar *.class`

   unde fişierul *myManifest.mf* conţine doar linia

   `Permissions: all-permissions`

2. Certificarea se obţine prin

   (a)
   ```
   keytool -genkey -keystore myKeystore -alias myself
       -dname "cn=XYZ, ou=cs, o=unitbv, l=brasov, c=RO"
       -keypass 123abc -storepass 123abc
   ```

   (b)
   ```
   keytool -selfcert -alias myself -keystore myKeystore
       -keypass 123abc -storepass 123abc
   ```

   (c)
   ```
   jarsigner -keystore myKeystore -signedjar cmmdc.jar
      -keypass 123abc -storepass 123abc cmmdc0.jar myself
   jarsigner -verify cmmdc.jar
   ```

   Primele două acţiuni au ca rezultat obţinerea certificatului *myKeystore* iar ultima acţiune reprezintă înglobarea certificării în arhiva `resursa.jar` - în cazul exemplului *cmmdc.jar*.

3. Se editează fişierul *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 înlocuieşte cu numele calculatorului care găzduieşte serverul Web.

4. Se crează fişier de apelare a aplicaţiei (*cmmdc.html*)

```
<html>
  <body bgcolor="#AAEEAA">
    <center>
      <h3>Visual Cmmdc Application </h3>
      <a href="http://host:8080/cmmdc/launch.jnlp">
         Launch the application </a>
    </center>
  </body>
</html>
```

5. Ansamblul

```
cmmdc
|   cmmdc.jar
|   cmmdc.html
|   launch.jnlp
```

se copiază în serverul Web.

Dintr-un navigator, apelarea aplicaţiei este *http://host:port/cmmdc/ cmmdc. html.*

Fişierul *jnlp* se descarcă pe calculatorul clientului şi se procesează cu `bin\javaws.exe` din Java.

Observaţie 13.1.1 *Deoarece testarea se face utilizând auto-certificarea, execuţia clasei este posibilă prin exceptarea site-ului http://localhost:8080 în*

`Java Control Panel` → `WebSettings` → `Exception Site List`

În Linux accesul la `Java Control Panel` este asigurat de funcţia `jcontrol` din JDK.

Aplicaţia Java ca servlet

Pentru includerea aplicaţie Java într-un servlet, primii trei paşi prezentaţi mai sus coincid. Singura diferenţă constă în conţinutul 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 şi fişiere:

```
app
   |     cmmdc.jar
   |     launch.jnlp
WEB-INF
   |--> lib
   |     |    jnlp-servlet.jar
   |     web.xml
cmmdc.html
```

În *launch.jnlp* atributul `codebase` devine

```
<jnlp codebase="http://host:8080/cmmdc/app">
```

Fişierul `jnlp-servlet.jar` se preia din distribuţia JDK, din catalogul *sample/jnlp/servlet.*

Fişierul `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. Conţinutul catalogului de lucru se arhivează într-un fişier `war`.

   ```
   jar cfv cmmdc.war app/* WEB-INF/* cmmdc.html
   ```

6. Utilizând *apache-tomcat* sau *jetty,* de exemplu, fişierul `war` se copiază în catalogul `webapps` al serverului web.

   Dintr-un navigator, apelarea aplicaţiei se obţine prin *http://host:8080/cmmdc/cmmdc.html.*

- Dacă aplicaţia Java utilizează imagini grafice acestea trebuie arhivate împreună cu aplicaţia, de exemplu într-un catalog *images.*

  Încărcarea şi afişarea 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 aplicaţia Java utilizează alte resurse date prin fişiere `jar` atunci toate fişierele `jar` trebuie să poarte aceaşi certificare.
  2. Fişierele `jar` certificate se depun în catalogul *app\lib.*
  3. În fişierul *jnlp,* în elementul `resources,` pentru fiecare fişier `jar` utilizat se introduce declaraţia

     <jar href="lib/resource.jar"/>

- Dacă arhiva `jar` a aplicaţiei Java este executabilă atunci, în fişierul *jnlp,* elementul `application-desc` poate lipsi.

  Într-o arhivă `jar` executabilă, fişierul `MANIFEST.MF` indică clasa cu metoda `main` şi eventualele resurse externe utilizate (fişiere `jar`) prin proprietăţile

```
Main-class: numeClasa
Class-path: lib/resursa1.jar lib/resursa2.jar . . .
```

  Pentru a obţine fişierul `MANIFEST.MF` cu aceste proprietăţi se editează un fişier text cu conţinutul de mai sus, denumit de exemplu *myManifest.mf,* şi se arhivează prin

```
jar cfvm numeArhiva.jar myManifest.mf *.class lib
```

  În mod asemănător se procedează şi pentru introducerea proprietăţii `Permissions: all-permissions.`

Întrebări recapitulative

1. Ce posibilitate oferă tehnologia Java Web Start ?
2. O aplicaţie Java urmează să fie valorificat pe Internet prin Java Web Start. Dacă aplicaţia necesită date ale clientului ce restricţie există şi cum se satisface restricţia ?
3. Ce înseamnă abrevierea JNLP?
4. În ce constă utilizarea protocolului JNLP?
5. Care este programul executabil care precesează un fişier `jnlp`?

Capitolul 14

Enterprise Java Beans

În prezent se evidenţiază două abordări privind realizarea aplicaţiilor de complexitate mai mare (*aplicaţiilor de întreprindere*):

- *Eclipse Enterprise for Java - EE4J / Java Enterprise Edition - JEE* este o extensie a interfeţei de programare (*API*) Java, pentru care există mai multe implementări. Principalul promotor este al modelului este *Oracle* , dar este susţinut şi de *RedHat - JBoss.*;
- *Spring* a cărei dezvoltare ţine de firma *VMware. Spring* este alcătuit din mai multe cadre de lucru. Acest model de dezvoltare este susţinut de *Google.*

Eclipse Enterprise for Java/Java Enterprise Edition este un cadru de lucru care înglobează pe o serie de implementări ale interfeţelor de programare (JDBC, JMS, JNDI, JSF, RMI-IIOP, JPA, JTA, JAAS, etc), resurse care pot utiliza componente *Enterprise Java Bean*- EJB.

O componentă EJB este o clasă Java - de obicei de tip POJO - care face parte din aplicaţia server, conţine metodele care rezolvă o problemă (*business logic*) şi este conţinut într-un server de aplicaţii. Serverul de aplicaţii asigură o serie de funcţionalităţi ca instanţierea componentelor EJB, injectarea dependinţelor, conexiunea cu bazele de date, gestiunea tranzacţiilor, etc. Serverul de aplicaţii poate fi interpretat şi ca un container de componente EJB.

Se spune că serverul de aplicaţie JEE asigură accesul la contextul în care rulează aplicaţia, adică la resursele serverului, şi la injectarea dependinţelor (*Context and Dependency Injection -CDI*).

Servere de aplicaţii gratuite:

- *glassfish* - Oracle.

- *WildFly* - RedHat - jboss.
- *apache-tomee*
- *WebLogic* - Oracle.

În acelaşi timp, aceste servere Web sunt containere EJB, de servlet şi 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*

14.1 Session EJB

Starea unui obiect Java este dată de valorile câmpurilor sale. Din punctul de vedere al reţinerii stării, există următoarele tipuri de componente sesiune EJB:

- *stateless* - fără reţinerea stării;
- *stateful* - cu reţinerea stării pe parcursul sesiunii serverului;
- *singleton* - există o singură instanţă a componentei EJB. Durata de viaţă a componentei coincide cu intervalul de timp în care componenta EJB este activă în serverul de aplicaţii.

Aplicaţiile cu componentă EJB constau din:

- componenta EJB - desfăşurată în serverul de aplicaţii. Această componentă poate fi totodată şi un serviciu Web de tip JAX-WS;
- aplicaţie 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 şi clientul RMI-IIOP formează câte un modul. Componenta EJB şi clientul Web formează o aplicaţie JEE care se instalează în serverul de aplicaţii JEE.

Modulele EJB şi clientul RMI-IIOP se arhivează cu extensia `jar` iar modulul Web se arhivează cu extensia `war`. Aplicaţia 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.

14.1.1 Componentă EJB sesiune stateless

Şablonul pentru crearea unei componente EJB de tip *stateless session* este

```
import javax.ejb.Stateless;

@Stateless
public class Componenta{
  public tip metoda(. . .){. . .}
  . . .
}
```

Exemplul 14.1.1 *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 }
```

Aplicaţia client va fi un servlet găzduit de acelaşi server de aplicaţie. Accesul la componenta EJB se poate programa:

- prin injectare, utilizând adnotări;
- prin JNDI.

În ambele cazuri serverul Web este responsabil de instanţierea componentei EJB.

Injectare cu adnotare

Serverul de aplicaţie *injectează* în servlet o instanţă a componentei EJB sau altfel explicat

```
@EJB                    |   @EJB(name="xyz")
Tip var;                |   Tip var;
```

injectează (realizează o referinţă) pentru *var* către un obiect de tipul *Tip* specificat. Actiunea este declanşată 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("<HTMI><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></HTMI>");

33     out.close();
34   }

36   public void doPost(HttpServletRequest req,HttpServletResponse res)
37       throws ServletException,IOException{
38     doGet(req,res);
39   }
40 }
```

Desfăşurarea aplicaţiei

```
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

Referinţa 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 `ear`.
java:app	Componenta se poate utiliza în aplicaţie Coincide cu numele arhivei `jar` sau `war`.
java:module	Componental 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  }
```

14.1.2 Componentă cu metode asincrone

În cazul apelării unei metode a unei componente EJB, clientul este blocat până la furnizarea rezultatului de către serverul de aplicaţie. Programând metode asincrone, programul client va deţine controlul imediat după apelare sau altfel explicat apelarea nu mai este blocantă.

Definirea unei metode asincrone

Se îndeplinesc două conditii:

1. Metoda asincronă este adnotată *@Asynchronous* (`javax.ejb.Asynchronous`);
2. Rezultatul metodei va fi de tip `java.util.concurrent.Future<V>`.

Exemplul 14.1.2

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=m%n;
15       m=n;
16       n=r;
17     }while(r!=0);
18     Long result=Long.valueOf(c);
19     return new AsyncResult<Long>(result);
20   }
21 }
```

Clasa `AsyncResult` implementează interfaţa `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();
}
```

14.1.3 Aplicaţie JEE cu module EJB, Web şi client RMI-IIOP

Modificăm arhitectura aplicaţiei 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
```

Desfăşurarea aplicaţiei va fi

```
|--> META-INF
|    |    application.xml
|    modul-ejb.jar
|    modul-Web.war
```

Instalarea aplicaţie, adică a ansamblului alcătuit din cele 2 module se face prin intermediul administratorului.

Fişierul `application.xml` precizează structura aplicaţiei

```
<?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 secţiunea 14.1.1 în sensul definirii unei interfeţe cu adnotarea `Remote`. Componenta EJB implementează interfaţa şi în plus, dacă interfată are numele *Xyz* atunci componenta EJB va avea numele *XyzBean*.

Această construcţie este necesară numai în cazul în existenţei unui client RMI-IIPO.

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 }
```

şi

```
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 secţiunea 14.1.1, cu diferenţa că referinţa la componenta EJB se face prin interfaţă.

În cazul exemplului tratat, desfăşurarea 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;

6  public class CmmdcClient{
7    @EJB
8    private static Cmmdc cb;

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 referinţa 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

- Sistemul de operare Windows

```
set GLASSFISH_HOME=. . .\glassfish*
%GLASSFISH_HOME%\glassfish\bin\appclient -client
    dist\cmmdc-iiop.jar -targetserver localhost:3700
```

- Sistemul de operare Linux

```
#!/bin/bash
GLASSFISH_HOME=. . . /glassfish*
$GLASSFISH_HOME/glassfish/bin/appclient -client
    dist/cmmdc-iiop.jar -targetserver localhost:3700
```

Apelarea modulului Web se face în mod obişnuit, prin:

`http://host:8080/`*context*

unde *context* a fost definit în fişierul `application.xml`.

14.1.4 Componentă EJB sesiune singleton

Serverul de aplicaţie crează o singură instanţiază a componentei EJB care este utilizată de toţi clienţii. Diferenţa constă în utilizarea adnotării `@Singleton` în loc de `@Stateless`.

Utilizăm modelul din secţiunea 14.1.3. Considerăm doar varianta bazată pe adnotări.

Exemplul 14.1.3 *Numărarea solicitărilor de utilizare a componentei EJB.*

Codul componentei EJB singleton

```
package single.ejb;
import javax.ejb.Remote;

@Remote
public interface Single{
  public int getIndex();
}
```

```
package single.ejb;
import javax.ejb.Singleton;

@Singleton
public class SingleBean implements Single{
  private int index=0;
  public int getIndex(){
     return ++index;
  }
}
```

Servlet-ul aplicaţiei client are codul

```
package single.web;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import single.ejb.Single;
import javax.ejb.EJB;
import javax.servlet.annotation.WebServlet;

@WebServlet(urlPatterns = "/single")
public class SingleServlet extends HttpServlet{
  @EJB
  Single sb;

  public void doGet(HttpServletRequest req,HttpServletResponse res)
      throws ServletException,IOException{
    int x=sb.getIndex();
    PrintWriter out=res.getWriter();
    res.setContentType("text/html");
    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 fişierul *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 aplicaţiei definit în `META-INF\application.xml` este *ejbsingle.*

Aplicaţia client are codul

```
1  package single.client;
2  import javax.ejb.EJB;
3  import single.ejb.Single;
4  import java.util.Scanner;

6  public class SingleClient{
7    @EJB
8    private static Single nb;

10   public static void main(String[] args)throws Exception{
11     Scanner scanner=new Scanner(System.in);
12     String ch="Y";
13     int index=0;

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 }
```

14.1.5 Componentă EJB sesiune stateful

Diferenţa formală faţă de modelele *Stateless* şi *Singleton* constă în utilizarea adnotării `@Stateful`.

Considerăm un exemplu ce derivă din cel precedent.

Exemplul 14.1.4 *Numărarea solicitărilor de utilizare a componentei EJB.*

Componenta EJB are interfaţa

```
1  package numara.ejb;
2  import javax.ejb.Remote;

4  @Remote
5  public interface Numara{
6    public int getIndex();
7    public void remove();
8  }
```

şi 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 ştergerea instanţei componentei EJB din serverul de aplicaţii. Din acest motiv, apelarea acestei metode într-un modul Web, are ca efect compromiterea servlet-ului.

Injectarea componentei EJB are loc în momentul activării servlet-ului. Astfel fiecare client Web va utiliza aceaşi componentă EJB.

Modulul IIOP este mult mai util. La fiecare apelare a modulului IIOP are loc injectarea unei componente EJB. Serverul de aplicaţii păstrează starea componentei 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 aplicaţiei Web este *ejbnumara.*

Codul modulului 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 aplicaţiilor corespunzătoare adnotărilor `Stateful` şi `Singleton` observaţi diferenţa rezultatelor furnizate.

14.2 Componentă EJB *MessageDriven*

O componentă EJB *MessageDriven* (MDB) preia mesaje care sunt trimise către un obiect destinaţie predefinit de tip *Queue* sau *Topic.* Componenta EJB este adnotată cu

`@MessageDriven(mappedName=`*nume_Destinaţie*`)`

Prelucrarea mesajelor primite se programează în metoda `onMessage` a interfeţei `MessageListener.`

Afişarea unor mesaje se obţine prin intermediul unei jurnalizări `java.util.logging.Logger.` Textele apar în fişierul `server.log` aflat în catalogul `log` al *domeniului.*

Obiectele administrator - fabrica de conexiuni, obiectul destinaţie - se crează înaintea desfăşurării 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 14.2.1 *O aplicaţie 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 }
```

Desfăşurarea componentei MDB va fi

```
simple-mdb
|--> META-INF
|      |     application.xml
|      simple-mdb.jar
```

Fişierul simple-mdb.jar nu conţine decât clasa *mdb.MessageBean,* reprodusă mai sus.

Codul clientului cu regăsirea obiectilor 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 }
```

şi 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`.

14.3 Componentă EJB *Entity*

O componenta de tip *entity* simplifică accesul la o bază de date într-o aplicaţie JEE.

Crearea bazei de date

Glassfish conţine SGBD *JavaDB* (*derby*). Vom lansa serverul *glassfish* şi 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* şi 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 ataşat cuvei
2. `Resource Type` Se selecteaza *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* şi se completează coloana `Value` cu *jdbc:derby:\. . . calea către locaţia bazei de date. . .\AgendaEMail*
8. `Finish`

Creare JDBC Resources

1. `JNDI Name` *jdbc/Adrese,* un nume simbolic ataşat resursei
2. `Pool Name` Se selectează *adresePool,* numele cuvei de conexiuni
3. `OK`

În cazul SGBD *mysql* driver-ul `mysql-connector-java-*-bin.jar` se copiază în catalogul `modules` din *Glassfish.*

Creare JDBC Connection Pool

1. `Pool Name` *mysqlPool* un nume simbolic ataşat cuvei
2. `Resource Type` Se selecteaza *java.sql.Driver*
3. `Database Driver Vendor` Se completează *Oracle*
4. `Introspect` Se bifează *Enabled*

5. `Next`

6. `Driver Classname` Se completează *com.mysql.jdbcDriver*

7. `Ping` Se bifează *Enabled*

8. `Additional Properties` Se bifează *URL* şi se completează coloana `Value` cu *jdbc:mysql:\. . . calea către locaţia bazei de date:3306\AgendaEMail?user=root*

9. `Finish`

Creare JDBC Resources

1. `JNDI Name` *jdbc/mysql,* un nume simbolic ataşat resursei

2. `Pool Name` Se selectează *mysqlPool,* numele cuvei de conexiuni

3. `OK`

Aplicaţie cu componentă Entity

O componenta *Entity* va fi adnotată `@Entity` şi corespunde unui tabel al bazei de date relaţionale.

Exemplul 14.3.1 *Aplicaţie de consultarea a bazei de date AgendaEMail.*

Desfăşurarea aplicaţiei 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* şi corespunde tabelei *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 interfeţei *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  }
```

Fişierul 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 corelaţii

- Valoarea atributului `name` a elementului `persistence-unit` (*agendae_persistence_ctx*) este utilizat în clasa *AgendaEMailBean* în adnotarea `@PersistenceContext`
- Elementul `jta-data-surse` conţine numele JNDI a bazei de date definit în *glassfish.*

Aplicaţia client

Desfăşurarea aplicaţiei client este

```
agendae-client
|--> ejb
|      |     AgendaEMail.class
|--> entity
|      |     Adrese.class
|--> META-INF
|      |     MANIFEST.MF
|      Client.class
```

Fişierul `MANIFEST.MF` este completat cu

```
Main-Class: Client
Class-Path: agendae-ejb.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 posibilităţile de apelare / utilizare al unei componente EJB?
3. Care este tehnologia de comunicaţie utilizată pentru apelerea unei componente EJB de un client program?
4. Cum se apelează un client - program al unei componente EJB?
5. Care sunt tiputile de componente sesiune EJB din JEE?

Capitolul 15

Aplicaţie pe nivele

Ne propunem să arătăm locul unde instrumentele de programare dezvoltate în cadrul cursului apar în activitatea uzuală de programare pentru o întreprindere economică.

Pe lângă faptul ca trebuie să asigure o funcţionalitate specifică, definirea / implementarea unui nivel presupune alegerea unei tehnologii informatice sau a unui produs informatic.

Una din aplicaţiile uzuale este întreţinerea unei baze de date. Considerăm cazul unei baze de date gestionată de un sistem de gestiune a bazelor de date (SGBD) relaţional. Problema întreţinerii constă în asigurarea următoarelor operaţii (denumite generic CRUD):

- adăugarea unei înregistrări noi (Create);
- consultarea bazei de date, extragerea de date (Read);
- modificarea unei înregistrări (Update);
- ştergerea unei înregistrări (Delete).

În cele ce urmează luăm în considerare platforma Java.

Punem în evidenţă nivele (tiers) (se mai utilizează şi terminologia staturi - layers) ca obiective ale unei metode pentru rezolvarea problemei enunţată anterior.

15.1 Nivelele unei variante de rezolvare

Nivelul bazei de date

În acest nivel se defineşte baza de date şi eventual se populează cu date de iniţializare. În anexa H (Programare distribuită 1), în acest scop s-au utilizat scripturi *sql* executate prin intermediul unui utilitar pus la dispoziţie de fiecare SGBD. Un script *sql* conţine comenzi specifice SGBD.

Nivelul de acces la SGBD

Interacţiunea dintre Java şi SGBD se programează:

- prin intermediul unui produs de mapare (Object Relational Mapping -ORM)
  - care implementează interfaţa Java Persistent API (JPA), parte din Java Enterprise Edition (JEE) prin `javax.persistence`. Amintim implementătile JPA:
    - Hibernate
    - OpenJPA
    - eclipselink
  - apache-empire-db
  - apache-cayenne
  - iBATIS
- prin acces nemijlocit (varianta utilizată în anexa H - Programare distribuită 1).
- prin tehnologia dezvoltată în cadrul de lucru *spring.*

Utilizând interfaţa de programare JPA sarcinile sunt:

- Fiecărui tabel al bazei de date supusă prelucrării i se asociază o clasă Java (componentă Java, clasă POJO atribuind fiecărei coloane a tabelului un câmp cu metodele *set* şi *get.* Denumim o asemenea clasă prin clasa *entity* asociată tabelului.

  Uzual numele clasei coincide cu numele tabelului.
- Declararea mapării prin fişierul `persistence.xml` (şablon minimal)

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1"
  xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
  http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">

  <persistence-unit name="numele_unitatii_de_persistenta"
          transaction-type="RESOURCE_LOCAL">
    <class>clasa_entity_a_tabelului </class>
    <properties>
      <property name="javax.persistence.jdbc.driver"
           value="driver-ul_bazei_de_date"/>
      <property name="javax.persistence.jdbc.url"
           value="adresa_url_a bazei_de_date"/>
      <property name="javax.persistence.jdbc.user"
           value="user_name"/>
      <property name="javax.persistence.jdbc.password"
           value="password"/>
    </properties>
  </persistence-unit>
</persistence>
```

În *glassfish* acest fişier este

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1"
  xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
  http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">

  <persistence-unit name="numele_unitatii_de_persistenta">
    <provider>
       org.eclipse.persistence.jpa.PersistenceProvider
    </provider>
    <jta-data-source>numele_jndi_al_bazei_de_date</jta-data-source>
    <properties>
      <property name="eclipse.jdbc.driver"
         value="driver-ul_bazei_de_date"/>
    </properties>
  </persistence-unit>
</persistence>
```

Maparea înseamnă stabilirea corespondenţei dintre câmpurile unui obiect de tip entity cu valorile corespunzătoare coloanelor ale unei înregistrări din tabel.

Folosind chiar accesul nemijlocit la baza de date, definirea clasei / claselor entity este utilă.

În acest nivel programatorul nu are probleme de comunicaţii.

Nivelul prelucrării unui tabel al bazei de date

Modelul prezentat în acest nivel este *Data Access Object* (DAO). O variantă de programare a acestui nivel constă din:

- O interfaţa cu operaţiile asupra datelor din tabel;
- Implementarea interfeţei;
- O clasă care mijloceşte apelarea operaţiile interfeţei apărând ca un serviciu. Rolul acestei clase este simplificarea aplicaţiei client.

Această arhitectură permite utilizarea ei pentru diferite tipuri de client sau moduri de acces la SGBD. Funcţie de modul de acces la SGBD utilizat sau de tipul de client detaliile de programare pot diferi.

Nivelul aplicaţiei client

Utilizatorul interacţionează cu aplicaţia de întreţinere a bazei de date prin intermediul unei aplicaţii client:

- de sine stătătoare (desktop) cu
  - interfaţă grafică (Swing, JavaFX)
  - text (într-o fereastră DOS)
- aplicaţie Web
  - servlet sau JSP;
  - aplicaţie bazată pe un cadru de lucru de tip Struts, JSF, GWT, etc;
  - serviciu Web (JAX-WS, JAX-RS);
  - websocket.

  Trebuie făcută distincţie între aplicaţia client de nivel şi clientul aplicaţiei Web care poate fi *html, JSP* sau chiar o aplicaţie desktop.

În toate cazurile aplicaţia client de nivel va utiliza serviciul realizat în nivelul anterior.

În acest nivel apar problemele de comunicaţii tratate în cadrul cursului de *Programare distribuită.*

Nivelul de autentificare şi autorizare

Aplicaţia client poate fi completată cu o componentă care asigură autentificare şi autorizare. În acest scop se poate utiliza *apache-shiro.*

Exemple

Exemplul 15.1.1 *Întreţinerea bazei de date Agenda de adrese email.*

- – Nivelul SGBD : *
  - Nivelul de acces la SGBD : **nemijlocit**
  - Nivelul aplicaţiei client : desktop, servlet, rest (jersey), websocket
  - Nivelul autentificare şi autorizare : desktop, servlet, rest (jersey, inclusiv client jersey)

- – Nivelul SGBD : Derby (glassfish 4.1)
  - Nivelul de acces la SGBD : **eclipselink (glassfish 4.1)**
  - Nivelul aplicaţiei client : servlet + desktop (RMI-IIOP), rest (jersey)
  - Nivelul autentificare şi autorizare : servlet (fără client RMI-IIOP)

- – Nivelul SGBD : Derby (cu username,passwd), Mysql
  - Nivelul de acces la SGBD : **hibernate**
  - Nivelul aplicaţiei client : desktop, servlet, rest (jersey), websocket

- – Nivelul SGBD : Derby (cu username,passwd), Mysql
  - Nivelul de acces la SGBD : **openjpa**
  - Nivelul aplicaţiei client : desktop

- – Nivelul SGBD : Derby (cu username,passwd), Mysql
  - Nivelul de acces la SGBD : **spring**
  - Nivelul aplicaţiei client : desktop, spring-mvc

Probleme

1. glassfish cu Mysql
2. glassfish cu client websocket (cu injectii)
3. openjpa cu client servlet, restful, websocket
4. empire sau alt produs ORM

Partea III

ANEXE

Anexa A

Unelte de dezvoltare

Începem această secţiune 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 părţi 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 şi semantica şi 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 declaraţia 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 sfârşit. Caracterele mari şi 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 sfârşit corespunde ultimului marcaj de început.

Un comentariu se indică prin

```
<!--
  Text comentariu
-->
```

Exemplul A.1.1 *Fişier* `XML` *- denumirile elementelor şi conţinutul lor permit înţelegerea 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 inchidere;
- marcajele încuibărite nu se amestecă.

Un document XML este *valid* dacă

- este *bine formatat;*
- în cazul că există o referinţă către XML Schema atunci documentul XML este conform schemei.

Un produs informatic accesibil gratuit, care permite verificarea bine formatării şi a validităţii unui document `xml`, este *XML Copy Editor.*

Pentru validare, dacă fişierul XML Schema se află în alt catalog decât cel al fişerului `xml` atunci acestuia i se asociază XML Schema prin *XML* → *Associate* → *XML Schema...*

Prelucrarea unui document XML în Java se poate face pe baza interfeţelor 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* şi *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 aplicaţii cooperante între sisteme de calcul simbolic - CAS (*Computer Algebra System*).

Formatul XML se utilizează şi pentru serializarea unui obiect, adică reprezentarea câmpurilor împreună cu valorile aferente sub formă de text, în cadrul transmisiilor de date. Interfaţa de programare este *Java Architecture for XML Binding* - JAXB.

A.2 *apache-ant*

Utilitarul *apache-ant* asigură executarea unui şir de comenzi de operare. Aceste comenzi se înregistrează într-un fişier de tip `xml`, cu denumirea `build.xml`. Astfel, *apache-ant* se substituie unui fişier de comenzi `bat` în Windows sau unui script shell din Linux/Unix. Avantajul obţinut constă în independenţa faţă de platforma de calcul (Windows, Linux).

Instalarea constă în dezarhivarea fişierului descărcat din Internet.

Lansarea în execuţie necesită fixarea variabilei de mediu `JAVA_HOME`, ce conţine calea la distribuţia Java. Lansarea se poate face prin următorul fişier de comenzi

- Sistemul de operare Windows

```
set JAVA_HOME=. . .
set ANT_HOME=. . .
%ANT_HOME%\bin\ant.bat %1
```

- Sistemul de operare Linux

```
#!/bin/bash
ANT_HOME=. . .
$ANT_HOME/bin/ant $1
```

Parametrul %1 acestui fişier de comenzi reprezintă obiectivul care se doreşte a fi atins. Dacă se modifică denumirea sau locaţia fişierului `build.xml` atunci fişierul de comenzi se invocă cu opţiunea `-buildfile`.

Un fişier `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* conţine o familie predefinită de sarcini. Programatorul are datoria fixării atributelor sarcinilor. Manualul din documentaţia produsului conţine descrierea atributelor cât şi exemple. În general, o sarcină reprezintă o operaţie executată uzual în linia de comandă.

Atributele se dau, respectând sintaxa `XML`

$$\text{numeAtribut} = \text{"valoareAtribut"}$$

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* lipseşte parametrul opţional 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}.

Sarcina de compilare <`javac...`> se face conform versiunii de Java declarată prin `JAVA_HOME`. Prin atributele `source` şi `target` se pot fixa modul Java utilizat pentru interpretarea textului sursă şi respectiv, pentru versiunea codului compilat.
Dacă este nevoie doar de *jdk1.7.0_** atunci sarcina <`javac...`> se foloseşte cu <`javac source="1.7" target="1.7" ...`>

Exemplul A.2.1 *Fişierul build pentru execuţia programelor din §1.2 pentru aplicaţia server.*

- Varianta modulară

```
1  <project name="Socket" default="Server" basedir=".">
2    <description>  Socluri TCP  </description>

4    <!-- set global properties for this build -->
5    <property name="build" location="mods"/>

7    <target name="init">
8      <!-- Create the time stamp -->
9      <tstamp/>
10     <!-- Create the build directory structure used by compile -->
11     <delete dir="${build}"/>
12     <mkdir dir="${build}"/>
13   </target>

15   <target name="Compile" depends="init"
16           description="compile the source ">
17     <javac srcdir="src" destdir="${build}" includeantruntime="false"/>
18   </target>
```

```
20    <target name="Server" depends="Compile">
21      <java classname="cmmdc.socket.server.MyMServer"   fork="true"
22       modulepath="mods" module="server"/>
23    </target>
24  </project>
```

- Varianta nemodulară

```
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"/>

7    <target name="init">
8      <!-- Create the time stamp -->
9      <tstamp/>
10     <!-- Create the build directory structure used by compile -->
11     <delete dir="${build}"/>
12     <mkdir dir="${build}"/>
13   </target>

15   <target name="Compile" depends="init"
16       description="compile the source " >
17     <javac srcdir="src" destdir="${build}" includeantruntime="false"/>
18   </target>

20   <target name="Server" depends="Compile">
21     <java classname="cmmdc.socket.server.MyMServer"
22         classpath="${build}" fork="true"/>
23   </target>
24 </project>
```

Utilizarea lui *apache-ant* presupune că toate resursele utilizate, cuprinse uzual în fişiere cu extensia `jar` (*j*ava *ar*hive) sunt disponibile pe calculatorul local. Dacă calculatorul este conectat la Internet, atunci resursele publice pot fi descărcate împreună cu toate dependinţele şi utilizate prin *apache-ant,* folosind suplimentar *apache-ivy* [1].

In vederea utilizării lui *apache-ivy* se copiază fişierul `ivy-*.jar` din distribuţia *apache-ivy* în `ANT_HOME\lib`.

Fişierul *build.xml* conţine în plus

```
<project name="Proiect" basedir="." default="obiectiv_final"
   xmlns:ivy="antlib:org.apache.ivy.ant">

  . . .
  <!-- ================================
          target: resolve
```

[1] Asemănător cu *maven,* un alt produs de dezvoltare.

```
        ================================= -->
  <target name="resolve" depends="init"
     description="--> retreive 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 fişier *ivy.xml*, cu dependinţele necesare aplicaţiei 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[2] este un alt cadru de dezvoltare şi gestiune a proiectelor (Project management framework). Calculatorul pe care se dezvoltă proiectul / aplicaţia trebuie să fie conectat la internet. Resursele necesare îndeplinitii diferitelor sarcini (*maven artifacts*) sunt preluate din internet şi depuse într-un depozit local *maven* (*local repository*). În prezent sunt întreţinute depozite publice de resurse soft necesare dezvoltării de aplicaţii cu *maven*[3] iar dezvoltatorii de instrumente soft au posibilitatea de a-şi promova produsele prin depunerea într-un depozit *maven*. Dintr-un asemenea depozit public resursele necesare sunt descărcate în depozitul local.

[2]Maven – acumulator de cunoştiinţe (Idiş).

[3]De exemplu repo1.maven.org/maven2.

Gestiunea proiectelor cu *apache-maven*

Instalarea produsului constă în dezarhivarea fişierului 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 distribuţia `jdk` folosită.

În mod obişnuit depozitul local *maven* este

- Sistemul de operare Windows

  `C:\Documents and Settings\`*client*`\.m2\repository`

- Sistemul de operare Linux

  `home/`*client*`/.m2/repository`

  Catalogul devine vizibil cu `Ctrl+H`.

Locaţia depozitului local se poate modifica, introducând elementul

```
<localRepository>volum:/cale/catalog_depozit</localRepository>
```

în fişierul `MAVEN_HOME\conf\settings.xml`.

Potrivit principiului separării preocupărilor (Separation of Conserns), un proiect *maven* produce o singură ieşire.

Declararea unui proiect se face printr-un fişier `pom.xml` (Project Object Model). Este sarcina programatorului să completeze fişierul `pom.xml`, creat la generarea structurii de cataloage ale proiectului, cu specificarea resurselor suplimentare sau a condiţionărilor în efectuarea unor operaţii (de exemplu, prezenţa adnotărilor necesită utilizarea versiunii Java ≥ 1.5).

Dezvoltarea unei aplicaţii / proiect prin *maven* presupune generarea unei structuri de cataloage (Standard directory layout for projects). Această structură de cataloage este specifică tipului / şablonului de aplicaţie (*archetype,* în limbajul *maven*).

Şabloane uzuale de aplicaţii:

Nume şablon	Semnificaţia
`maven-archetype-quickstart`	aplicaţie simplă (şablonul implicit)
`maven-archetype-webapp`	aplicaţie Web

Şablonul se specifică în parametrul `-DarchetypeArtifactId` al comenzii `mvn archetype:generate`.

Îndeplinirea diferitelor obiective (generarea unui proiect, compilare, arhivare, testare, etc) se obţin prin comenzi *maven*. În cazul unei erori, detalii suplimentare se obţin dacă este prezentă opţiunea `-e`.

Comenzile *maven* sunt de două tipuri:

- Comenzi pentru gestiunea ciclului de viaţă al unui proiect (lifecycle commands):

Comanda *maven*	Semnificaţia
`mvn -version`	afişează versiunea *maven* (utilă pentru verificarea funcţionării lui *maven*)
`mvn clean`	şterge fişierele *maven* generate
`mvn compile`	compilarea sursele Java
`mvn test-compile`	compilează sursele Java care realizează testele `junit`
`mvn test`	execută testul `junit`
`mvn package`	crează o arhivă jar sau war
`mvn install`	depune arhiva jar sau war în depozitul local

- Comenzi de operare inserate (plugin commands):

Comanda *maven*	Semnificaţia
`mvn -B archetype:generate`	generează structura de cataloage a proiectului. Opţiunea -B are ca efect generarea neinteractivă. `mvn -B archetype:generate` \ `-DgroupId=`*numelePachetuluiAplicaţiei* \ `-DartifactId=`*numeleProiectului* \ `-DarchetypeArtifactId=`*numeŞablon*\ `-Dversion=`*versiuneaProiectului*
`mvn archetype:generate`	generează structura de cataloage ai proiectului. Şablonul se alege dintr-o listă
`mvn clean:clean`	şterge fişierele generate în urma compilării
`mvn compiler:compile`	compilarea sursele Java
`mvn surefire:test`	execută testul `junit`
`mvn jar:jar`	crează o arhivă jar
`mvn install:install-file`	depune o arhivă jar în depozitul local `mvn install:install-file` \ `-Dfile=`*numeFişier* \ `-DgroupId=`*numePachet* \ `-DartifactId=`*numeProiect* \ `-Dversion=`*versiunea* \ `-Dpackaging=`*tipArhivă*
`mvn exec:java`	execută metoda `main` a unei clase `mvn exec:java -Dexec.mainClass=` *"clasaMetodeiMain"* `-Dexec.args=`*"listă argumente"*
`mvn dependency:copy-dependencies`	descarcă în catalogul `target` resursele declarate în `dependencies`.

Astfel comanda

- Sistemul de operare Windows

```
set GroupID=simple.app.helloworld
set ArtifactID=helloworld
set Version=1.0
mvn -B archetype:generate -DgroupId=%GroupID%
  -DartifactId=%ArtifactID%
  -Dversion=%Version%
  -DarchetypeArtifactId=maven-archetype-quickstart
  -DinteractiveMode=false
```

- Sistemul de operare Linux

```
GroupID=simple.app.helloworld
ArtifactID=helloworld
Version=1.0
mvn  archetype:generate -DgroupId=$GroupID
  -DartifactId=$ArtifactID
  -Dversion=$Version
  -DarchetypeArtifactId=maven-archetype-quickstart
  -DinteractiveMode=false
```

generează arborescenţa

```
hello
|--> src
|    |--> main
|    |    |--> java
|    |    |    |--> unitbv
|    |    |    |    |--> cs
|    |    |    |    |    |--> calcul
|    |    |    |    |    |    |  App.java
|    |--> test
|    |    |--> java
|    |    |    |--> unitbv
|    |    |    |    |--> cs
|    |    |    |    |    |--> calcul
|    |    |    |    |    |    |  AppTest.java
|    pom.xml
```

Proprietatea `-DinteractiveMode=false` se poate înlocui cu opţiunea `-B`.

Descrierea proiectului este dată în fişierul `pom.xml` generat

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>simple.app.helloworld</groupId>
  <artifactId>helloworld</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>helloworld</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

App.java este programul Java *HelloWorld* iar *AppTest.java* este un program de verificare bazat pe `junit`.

Pentru testarea aplicaţiei, din catalogul *hello* se execută comenzile

```
mvn compile
mvn exec:java -Dexec.mainClass="simple.app.helloworld.App"
mvn test
```

Execuţia programului se poate lansa prin intermediul unui profil (*profile*)

```
mvn exec:java -PnumeProfil
```

În prealabil în fişierul `pom.xml` se introduce secvenţa

```
<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>
        <version>1.6.0</version>
        <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 programatorul este acela de a înlocui aceste programe cu cele care rezolvă sarcinile proiectului. Pentru orice prelucrare toate dependinţele trebuie să se găsească în depozitul local *maven*. Dacă o dependinţă (resursă `jar`) nu se găseşte în depozitul local atunci resursa este căutată într-un depozit global şi este descărcată în depozitul global. Este sarcina programatorului să declare toate dependinţele necesare unei aplicaţii. 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 aplicaţiei 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(14l,rez);
18    }

20    public static void main(String args[]){
21      org.junit.runner.JUnitCore.main("proiect.TestProiect");
22    }
23  }
```

Completarea fişierului `pom.xml`:

1. Aplicaţia în Java 9. Această cerinţa se declară prin secvenţa de cod

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.7.0</version>
      <configuration>
        <source>1.9</source>
        <target>1.9</target>
      </configuration>
    </plugin>
  </plugins>
</build>
```

2. Întrucât dorim să folosim cea mai recentă versiune a produsului `junit` modificăm numărul versiunii.

Fişierul `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>

7    <groupId>simple.app.cmmdc</groupId>
8    <artifactId>cmmdc</artifactId>
9    <version>1.0-SNAPSHOT</version>
10   <packaging>jar</packaging>

12   <name>cmmdc</name>
13   <url>http://maven.apache.org</url>

15   <properties>
16     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
17   </properties>
```

```
19   <dependencies>
20     <dependency>
21       <groupId>junit</groupId>
22       <artifactId>junit</artifactId>
23       <version>4.12</version>
24       <scope>test</scope>
25     </dependency>
26   </dependencies>

28   <build>
29     <plugins>
30       <plugin>
31         <groupId>org.apache.maven.plugins</groupId>
32         <artifactId>maven-compiler-plugin</artifactId>
33         <version>3.7.0</version>
34         <configuration>
35           <!-- or whatever version you use -->
36           <source>1.9</source>
37           <target>1.9</target>
38         </configuration>
39       </plugin>
40     </plugins>
41   </build>
42 </project>
```

După asamblarea aplicaţiei, din catalogul aplicaţiei, operarea poate fi

```
mvn clean compile test
mvn exec:java -Dexec.mainClass="simple.app.cmmdc.App"
```

Utilizând jdk1.8.0_* dar cu cerinţa ca sursa şi rezultatul compilării să fie conforme cu Java 7 se va introduce în fişierul `pom.xml` secvenţa

```
<properties>  <!-- App Engine Standard currently requires Java 7 -->
  <maven.compiler.target>1.7</maven.compiler.target>
  <maven.compiler.source>1.7</maven.compiler.source>
</properties>
```

maven cu *ant*

În *maven* se pot integra sarcini *apache-ant* pentru orice etapă al evoluţiei unei aplicaţii. Utilizarea constă în completarea fişierului `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 evoluţia proiectului *maven* - compile, package, install, test, etc.

Mai precis, elementele <*echo*> se înlocuiesc cu sarcinile `ant` care se doresc efectuate.

Proprietăţile se definesc în elementul

```
<properties>
   <nume.proprietate>valoare</nume.proprietate>
   . . .
</properties>
```

A.4 *Gradle* pentru Java

Gradle este un instrument de dezvoltare oferind:

- funcţionalităţi pentru mai multe limbaje de programare (Java, Scala, Python, C/C++, Android, etc);
- posibilităţ de integrare în medii integrate de dezvoltare (IDE) Eclipse, Android Studio, IntelliJ;
- funcţionalităţi care combină pe cele oferite de *ant, maven* şi *ivy.*

Instalarea constă în dezarhivarea fişierului descărcat.

Gradle pentru aplicaţii Java desktop

În vederea utilizarea lui Gradle sunt necesare:

- fixarea variabilei `JAVA_HOME;`
- fixarea variabilei `GRADLE_HOME;`
- includerea în variabila de sistem PATH a căii `GRADLE_HOME/bin.`

```
set JAVA_HOME=. . .
set GRADLE_HOME=. . .\gradle-*
set PATH=%GRADLE_HOME%\bin;%PATH%
```

Un proiect dezvoltat prin Gradle cere o structură fixă a aplicaţiei. Generarea constă din

1. Se crează catalogul aplicaţiei.
2. Din catalogul creat se generează structura aplicaţiei

   ```
   gradle init --type java-library
   ```

   Se obţine

   ```
   Catalogul_aplicatiei
   |--> .gradle
   |--> gradle
   |--> src
   |    |--> main
   |    |    |--> java
   |    |    |     |     Library.java
   |    |--> test
   |    |    |--> java
   |    |    |     |     LibraryTest.java
   |    build.gradle
   |    gradlew
   |    gradlew.bar
   |    settings.gradle
   ```

   Dezvoltarea se face înlocuind fişierele *Library.java, LibraryTest.java* cu ale aplicaţiei.

3. Se înlocuiesc fişierele *Library.java, LibraryTest.java* cu cele ale aplicaţiei.

Testarea este bazată pe *junit*. Dacă proiectul nu conţine componenta de testare catalogul `test` se şterge.

Fişierul `build.gradle` precizează prelucrarile solicitate.

Linia `apply plugin: 'java'` prezentă în fişierul `build.gradle` asigură compilarea. Din catalogul_aplicaţiei Gradle pentru compilare şi arhivare se va lansa

```
gradle clean assemble
```

În catalogul_aplicaţiei va rezulta subcatalogul `build` cu conţinutul

```
build
|--> classes
|--> libs
|    |    catalogul_aplicatiei.jar
|--> tmp
```

Catalogul `libs` va lipsi dacă sarcina de prelucrare cerută este `compileJava`.

Pentru lansarea în execuţie a programului, fişierul `build.gradle` trebuie să conţină

```
apply plugin: 'java-library'
apply plugin: 'application'

mainClassName='clasa cu metoda main'
```

iar lansarea va fi

```
gradle clean assemble run
```

Tabelul A.1 conţine sarcini Gradle.

Sarcină	Acţiune
clean	Şterge catalogul `build`
compileJava	Compilare
compileTest	Compilare claselor test
assemble	Compilare şi arhivare
run	Execută aplicaţia
test	Execută testele

Table A.1: Sarcini Gradle

Apache-ant prin *Gradle*

Un fişier *build.xml* pentru *apache-ant* se execută prin Gradle cu fişierul *build.gradle*

```
ant.importBuild 'build.xml'
```

prin

```
gradle obiectiv_Ant_din_build.xml
```

Detalii de configurare

- **Introducerea de date** se poate face doar prin intermediul unei interfeţe grafice, cu excepţia celor introduse ca argumente la apelarea execuţiei (*gradle run*).
- **Comentariile** în `build.gradle` se indică prin
  - // linie de comentariu
  - /* text comentariu */
- **Preluarea argumentelor din linia de comandă** se indică în `build.gradle` prin

1. Varianta 1

```
apply plugin: 'java-library'
apply plugin: 'application'

mainClassName='clasa cu metoda main'

run{
  args System.getProperty("exec.args").split()
}
```

cu

```
gradle.bat clean assemble run -Dexec.args="arg_1 arg_2 . . ."
```

2. Varianta 2

```
apply plugin: 'java'
apply plugin: 'application'

mainClassName='clasa cu metoda main'

run{
  if(project.hasProperty('args')){
    args project.args.split('\\s')
  }
}
```

cu

```
gradle.bat clean assemble run -Pargs="arg_1 arg_2 . . ."
```

- **Arhiva `jar` va fi executabilă** dacă în fişierul `build.gradle` se include secvenţa

```
jar {
    manifest {
        attributes 'Main-class': 'clasa cu metoda main'
    }
}
```

  Fişierul `jar` se află în catalogul `build/distributions` în arhiva aplicaţiei.

- **Proprietăţile de sistem** se declară în fişierul `build.gradle`

```
run {
    classpath = sourceSets.main.runtimeClasspath
    //classpath.each { println it }
    /* Can pass all the properties: */
    //systemProperties System.getProperties()

    /* Or just each by name: */
    systemProperty "java.rmi.server.codebase", System.getProperty("java.rmi.server.codebase")

}
```

  şi valorile proprietăţilor sunt definite în

```
gradle.bat clean run -DnumeProprietate=valoare . . .
```

- **Resursele suplimentare** - fişiere `jar` - depuse în catalogul *lib* sunt considerate dacă în `build.gradle` se includ secvenţele

```
repositories{
  flatDir{
    dirs 'lib'
  }
}

dependencies{
  compile 'junit:junit:4.12'
}
```

  În secţiunea `dependencies` sintaxa utilizată este
  nume_pachet : nume_arhiva_jar : versiune.

Gradle pentru aplicaţii Web

Servlet

Se generează în mod obişnuit o aplicaţie Java. Structura aplicaţiei se completează cu catalogul `webapp` care conţine `WEB-INF`.

Fişierul `build.gradle` va conţine cu

```
apply plugin: 'war'

dependencies {
    providedCompile 'javax.servlet:javax.servlet-api:3.1.0'
}
```

Se va executa comanda

`gradle clean assemble`

În catalogul `buils/libs` se găseşte arhiva `war` care trebuie desfăşurată într-un server Web.

Aplicaţie din mai multe proiecte Gradle

Pentru simplitate considerăm aplicaţia de calcul a celui mai mare divizor comun a două numere întregi cu socluri TCP.

În locul clientului DOS se dezvoltă doi clienţi cu interfaţă grafică bazate pe *Swing* (VisualCmmdcClient.java) şi *JavaFX* (ClientFXCmmdc.java).

Restricţia de a preciza în `build.gradle` clasa cu metoda `main` impune ca fiecare aplicaţie client să fie inclusă într-un proiect Gradle.

Gradle oferă posibilitatea de a lansa compilarea şi arhivarea o singură dată pentru toate proiectele Gradle incluse. Astfel vom crea structura

```
cmmdc
|--> server
|--> clientswing
|--> clientfx
|     settings.gradle
|     build.xml
|     build.gradle
```

unde

- *settings.gradle*

```
1 include 'server','clientfx','clientswing'
```

  Se precizează proiectele Gradle ale aplicaţiei.

- *build.xml*

```
<project>
  <path id="myclasspath">
     <pathelement path="server/build/classes/main"/>
     <pathelement path="clientfx/build/classes/main"/>
     <pathelement path="clientswing/build/classes/main"/>
  </path>

  <target name="starter">
    <parallel>
      <java classname="server.AppServer"
        classpathref="myclasspath"  fork="true"/>
      <java classname="client.ClientFXCmmdc"
        classpathref="myclasspath" fork="true"/>
      <java classname="client.VisualCmmdcClient"
        classpathref="myclasspath" fork="true"/>
    </parallel>
  </target>
</project>
```

Fişier *apache-ant* pentru prelucrarea fiecărui proiect Gradle.

- *build.gradle*

```
ant.importBuild 'build.xml'
```

Gradle lansează prin *apache-ant* prelucrarea proiectelor.

Proiectele componente se dezvoltă în mod obişnuit, după care se lansează comanda `gradle clean assemble`

Lansarea în execuţie a fiecărei proiect se obţine din catalogul proiectului prin `gradle run`

Anexa B

Lambda expresii

Prin λ-expresii se introduc funţii. Reamintim că o metodă se utilizează doar în urma instanţierii clasei care încapsulează metoda respectivă.

Sintaxa unei λ-expresii este alcătuită din trei părţi:

Lista argumentelor	Simbol	Corpul funcţiei
$(x_1, \ldots, x_n)$	$->$	expresia funcţiei

O λ-expresie se introduce prin intermediul unei interfeţe funcţionale.

B.1 Interfeţe funcţionale

Interfeţe funcţionale predefinite

java.util.function.Function$< T, R >$

Se defineşte o funcţie $f : \{T\} \to \{R\}$, unde prin $\{X\}$ s-a notat mulţimea obiectelor de tip X.

Metode

- `R apply(T` t`)`

```
@FunctionalInterface
interface Function<T,R> {
   R apply(T t);
}
```

Exemplul B.1.1 *Transformarea gradelor Celsius în şi din grade Fahrenheit.*

```
import java.util.function.Function;

  static double cf(int trans,double g){
    Function<Double,Double> c2f=x->Double.valueOf(1.8*x+32);
    Function<Double,Double> f2c=x->Double.valueOf((x-32)/1.8);
    if(trans==1)
      return c2f.apply(g);
    else
      return f2c.apply(g);
  }
```

java.util.function.BiFunction$< T1, T2, R >$

Se defineşte o funcţie $f : \{T1\} \times \{T2\} \to \{R\}$.
Metode

- `R apply(T1` *t1,* `T2` *t2)*

java.util.function.Consumer$< T >$

Se defineşte o funcţie $f : \{T\} \to \emptyset$. Funcţia se manifestă prin efectele pe care le produce, fără să returneze o valoare.
Metode

- `void accept(T` t`)`

B.2 Fir de execuţie prin λ-expresie

Java introduce firele de execuţie prin interfeţele

- `java.lang.Runnable`

  Interfaţa declară metoda `public void run()`.

  Clasa `java.lang.Thread` implementează interfaţa `Runnable`.

- `java.util.concurrent.Callable<V>`

  Interfaţa declară metoda `public V call()`.

Şablon pentru definirea unui fir de execuţie 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.2.1

```
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 execuţie:

- ```
  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();
    }
  ```

O varianta mai evoluată, dar mai simplă constă în utilizarea interfeţelor `java.util.function.Function<T,R>`, `java.util.function.Consumer<T>`.

Corespunzător celor două cazuri de mai sus codurile sunt

```
1 import java.util.function.Consumer;
2 public class AplicR83{

4   static Consumer<String> f=(txt)->{
5     new Thread(()->{
6       System.out.println(txt);
7     }).start();
8   };
```

```
10    public static void main(String args[]){
11      f.accept("Primul fir de executie");
12      f.accept("Al doilea fir de executie");
13      f.accept("Al treilea fir de executie");
14    }
15  }
```

şi respectiv

```
1  import java.util.concurrent.ExecutorService;
2  import java.util.concurrent.Executors;
3  import java.util.function.Function;

5  public class AplicR84{
6    static final int NTHREADS=100;
7    static ExecutorService exec=Executors.newFixedThreadPool(NTHREADS);

9    static Function<String,Thread> f=txt->{
10     return new Thread(()->{
11       System.out.println(txt);
12     });
13   };

15   public static void main(String args[]){
16     exec.execute(f.apply("Primul fir de executie"));
17     exec.execute(f.apply("Al doilea fir de executie"));
18     exec.execute(f.apply("Al treilea fir de executie"));
19     exec.shutdown();
20   }
21 }
```

Şablon pentru definirea unui fir de execuţie 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.2.2

```
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 execuţie:

- ```
  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());
     }
   }
  ```

Versiunile bazate pe interfaţa `import java.util.function.Function<T,R>` sunt

```
import java.util.concurrent.Callable;
import java.util.function.Function;

public class AplicC80{
  static Function<Integer,Callable<Integer>> f=index->{
    Callable<Integer> c=()->{
      System.out.println("I am "+index);
      return index;
    };
    try{
      c.call();
    }
    catch(Exception e){
      e.printStackTrace();
    }
    return c;
  };

  public static void main(String args[]){
    int numarFire=5;
    try{
      for(int i=0;i<numarFire;i++){
        f.apply(i);
      }
    }
    catch(Exception e){
      System.out.println("Exception : "+e.getMessage());
    }
  }
}
```

şi

```
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;
import java.util.function.Function;

public class AplicC830{
  static final int NTHREADS=3;
  static ExecutorService exec=Executors.newFixedThreadPool(NTHREADS);

  static Function<Integer,Callable<Integer>> f=index->{
    Callable<Integer> c=()->{
      System.out.println("I am "+index);
      return index;
    };
    return c;
  };

  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.apply(i));
      }
```

```
27          List<Future<Integer>> results=exec.invokeAll(tasks);
28          for(int i=0;i<NTHREADS;i++){
29            Future<Integer> r=results.get(i);
30            if(r.isDone()) System.out.println("Returned : "+r.get());
31          }
32          exec.shutdown();
33        }
34        catch(Exception e){
35          System.out.println("Exception : "+e.getMessage());
36        }
37      }
38  }
```

Anexa C

Verificare automată

C.1 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 funcţionarea unui program de rezolvare, pentru depistarea unor greşeli.

S-a dezvoltat şi 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 acelaşi scop de verificare a rezultatelor este *TestNG.*

junit permite verificarea automată a rezultatelor furnizate de un program, pentru o mulţime de date de test.

Instalarea produsului constă din dezarhivarea fişierului descărcat într-un catalog `JUNIT_HOME`. Pentru compilare şi execuţie, variabila de sistem `classpath` trebuie să conţină referinţa `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 operaţii necesare înainte sau după efectuarea testelor se precizează respectiv, în câte o metodă care a fost declarată cu adnotarea `@org.junit.Before` şi 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 *aşteptat*, Tip *actual*)
     unde Tip poate fi `double, int, long, Object`.
   - `static void assertEquals(double` *aşteptat*, `double` *actual*, `double` *delta*)
     Testul reuşeşte dacă $|aşteptat\text{-}actual| < delta$.
   - `static void assertArrayEquals`(Tip[ ] *aşteptat*, Tip[ ] *actual*)
     unde Tip poate fi `byte, char, int, long, short, Object`.
   - `static void assertTrue(boolean` *condiţie*)
   - `static void assertFalse(boolean` *condiţie*)
   - `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 obţine

```
JUnit version 4.5
.
Time: 0.03

OK (1 test)
```

Exemplul C.1.1 *Testarea clasei App (2.2.1).*

```
1  package server;
2  import org.junit.*;
3  import static org.junit.Assert.*;

5  public class TestApp{

7    @Test
8    public void test(){
9      assertEquals(8,App.cmmdc(56,24));
10   }


13   public static void main(String[] args){
14     org.junit.runner.JUnitCore.main("server.TestApp");
15   }
16 }
```

Exemplul C.1.2 *Testarea clasei MyMServer (2.2.1).*

Primul test se referă la metoda *getServerSocket.* Se verifică afirmaţiile:

1. Metoda returnează un obiect $\neq$ `null`.
2. Obiectul returnat este de tip `ServerSocket`.

Al doilea test verifică metoda *myAction.* În acest scop se defineşte într-o clasă un fir de execuţie care în metoda `run()` apelează metoda *myAction.* Acţiunile metodei de testare sunt:

1. Se obţine un obiect `ServerSocket`.
2. Se lansează firul de execuţie 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 initializare(){
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 comunication 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  }
```

C.2 Testare cu *selenium*

Selenium este un produs care permite testarea funcţionării unor aplicaţii de tip servlet, JSP, websocket. *Selenium* interacţionează cu aplicaţia Web prin intermediul fişierului `html` care apelează aplicaţia - clientul Web - şi a răspunsului `html`.

Utilizarea produsului presupune accesul prin variabila de sistem `classpath` la toate resursele `jar` ale distribuţiei *selenium* pentru Java. În plus este nevoie de o resursă care asigură conexiunea dintre *selenium* şi navigatorul utilizat. În cazul lui *Chrome* acestă resursă pentru Windows este `chromedriver.exe`, iar în cazul lui *Mozilla / Firefox* sub Linux este `geckodriver`.

Verificarea presupune că aplicaţia Web este activă.

Şablonul de programare este ilustrat în exemplul următor

Exemplul C.2.1 *Verificarea funcţionării servlet-elor CmmdcServlet, HelloServlet; a aplicaţiei JSP cmmdc1pagina.jsp şi a aplicaţiei CmmdcWebSocketParam.*

În aplicaţia Web butonului *submit* trebuie completat cu atributul `name="btn"` şi răspunsul să conţină stringul *Cmmdc.*

```
1  import org.openqa.selenium.By;
2  import org.openqa.selenium.WebDriver;
3  import org.openqa.selenium.chrome.ChromeDriver;
4  import org.openqa.selenium.support.ui.ExpectedCondition;
5  import org.openqa.selenium.support.ui.Wait;
6  import org.openqa.selenium.support.ui.WebDriverWait;

8  public class TestSelenium {
9    static WebDriver driver;
10   static Wait<WebDriver> wait;

12   public static void main(String[] args) {
13      String browser=args[0];
14     if(browser.equals("chrome")){
15       // Pentru Google Chrome
16       driver = new ChromeDriver();
17     }
18     else{
19       // Pentru Mozilla Firefox
20       driver = new FirefoxDriver();
21     }
22     wait = new WebDriverWait(driver, 30);
23     driver.get("http://localhost:8080/apphelloP/");
24     //driver.get("http://localhost:8080/myservlet/cmmdc.html");
25     //driver.get("http://localhost:8080/cmmdc1/cmmdc1pagina.jsp");
26     //driver.get("http://localhost:8080/CmmdcWebSocketParam/");

28     boolean result;
29     try {
30       result=helloNameServlet();
31       //result=cmmdcApp();
```

```
32      }catch(Exception e) {
33        e.printStackTrace();
34        result = false;
35      } finally {
36          driver.close();
37      }

39      System.out.println("Test " + (result? "passed." : "failed."));
40      if (!result) {
41        System.exit(1);
42      }
43    }

45    private static boolean helloNameServlet() {
46      // Completarea formularului
47      driver.findElement(By.name("name")).sendKeys("xyz");
48      // Clic buton
49      driver.findElement(By.name("btn")).click();
50      // Asteptam rezultatul
51      wait.until(new ExpectedCondition<Boolean>() {
52        public Boolean apply(WebDriver webDriver) {
53          System.out.println("Searching ...");
54          return webDriver.findElement(By.tagName("body"))
55            .getText()
56            .contains("Hi")!= false;
57        }
58      });
59      return driver.findElement(By.tagName("body"))
60        .getText()
61        .contains("Hi");
62    }

64    private static boolean cmmdcApp() {
65       // Completarea formularului
66      driver.findElement(By.name("m")).sendKeys("56");
67      driver.findElement(By.name("n")).sendKeys("42");
68      // Clic buton
69      driver.findElement(By.name("btn")).click();
70     // Asteptam rezultatul
71      wait.until(new ExpectedCondition<Boolean>() {
72        public Boolean apply(WebDriver webDriver) {
73          System.out.println("Searching ...");
74          return webDriver.findElement(By.tagName("body"))
75            .getText()
76            .contains("Cmmdc")!= false;
77        }
78      });
79      return driver.findElement(By.tagName("body"))
80        .getText()
81        .contains("Cmmdc");
82    }
83  }
```

Anexa D

Jurnalizare

Jurnalizarea adică afişarea / reţinerea rezultatelor sau evenimentelor într-un fişier. Deseori prezintă interes evoluţia 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`, Quality of Open Software).
- *logback* (`logback.qos.ch`).

Jurnalizare prin `java.util.logging`

Şablonul de programare cu afişarea mesajelor pe ecranul monitorului este

```
1  import java.util.logging.Logger;

3  public class Exemplu{
4    static Logger logger = Logger.getLogger(Exemplu.class.getName());

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 afişează

```
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 fişier, 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 reţinerea rezultatelor în fişier

```
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 fişierul 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>
```

```
    </encoder>
  </appender>

  <appender name="FILE" class="ch.qos.logback.core.FileAppender">
    <file>results.log</file>
    <encoder>
      <pattern>
        %date %level [%thread] %logger{10} [%file:%line] %msg%n
      </pattern>
    </encoder>
  </appender>

  <root level="trace">
    <appender-ref ref="STDOUT" />
          <appender-ref ref="FILE" />
  </root>
</configuration>
```

Jurnalizare prin `apache-log4j-2`

Jurnalizare cu reţinerea rezultatelor în fişier

```
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class Exemplu{
  static Logger logger = LogManager.getLogger(Exemplu.class);

  public static void main(String args[]) {
    logger.warn("WARN : Hello");
    logger.debug("DEBUG : Hello");
    logger.info("INFO : Hello");
    logger.fatal("FATAL : Hello");
  }
}
```

cu fişierul de configurare

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration status="OFF">
  <properties>
    <property name="filename">results.log</property>
  </properties>
  <appenders>
    <Console name="Console" target="SYSTEM_OUT">
      <PatternLayout
        pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n" />
    </Console>
    <File name="File" fileName="${filename}">
       <PatternLayout
        pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n" />
    </File>
  </appenders>
  <loggers>
    <root level="trace">
      <appender-ref ref="Console" />
      <appender-ref ref="File" />
```

```
20      </root>
21    </loggers>
22  </configuration>
```

Anexa E

Componentă Java (Java *Bean*)

O componentă Java este o clasă care poate interacţiona cu alte componente Java, cu un document `jsp`, etc.

O componenta Java conţine cel puţin

- Un constructor fără nici un argument;
- O mulţime de câmpuri declarate `private;`
- Pentru fiecare asemenea câmp

```
private Tip xyz;
```

trebuie definite cel puţin una din metodele

```
public void setXyz(Tip xyz){
  this.xyz=xyz;
}

public Tip getXyz(){
  return xyz;
}
```

Generarea automata a metodelor *set/get*

Produsul `lombok` permite generarea facilă a metodelor *set / get*. Produsul este reprezentat de o arhivă `jar`.

Următorul exemplu simplu pune în evidenţă modul de utilizare:

Definirea componentei Java

```
1  import lombok.Getter;
2  import lombok.Setter;
3
4  public class CmmdcBean{
5    @Getter @Setter private long m=1;
6
7    @Getter @Setter private long n=1;
8
9    @Override public String toString() {
10     return String.format("%s %d,%s %d", "m = ", m, "n = ",n);
11   }
12 }
```

Utilizarea componentei Java

```
1  public class Utilizare{
2    public static void main(String[] args){
3      CmmdcBean obj=new CmmdcBean();
4      obj.setM(56);
5      obj.setN(42);
6      System.out.println(obj.toString());
7    }
8  }
```

Anexa F

Serializare fără XML

Scopul serializării unor date are ca scop obţinerea unei reprezentări text în vederea transmiterii prin reţea. Orice soluţie trebuie să ofere şi posibilitatea refacerii datelor din forma serializată Serializarea se poate obţine 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*[1]

  – *JavaScript Object Notation - JSON*

Se consideră că JSON este o restricţie de YAML.

[1] Acronimul YAML are şi altă semnificaţie *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(){}

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 instanţiază 3 obiecte care vor fi convertite în reprezentări YAML - stringuri ce sunt reţinute într-un obiect de tip `java.util.Map.` În final acest obiect este reprezentat YAML, care este salvat într-un fişier.

```
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;

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 }
```

Fişierul creat anterior este preluat, conţinutul 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 şi un client.

Pentru reprezentarea datelor în JSON se utilizează structurile de date:

- colecţie de atribute, adică perechi (nume, valoare). O colecţie de atribute este denumit obiect JSON.
- şir de valori.

Aceste structuri de date sunt prezente în toate limbajele de programare de uz general.

O colecţie de atribute se reprezintă prin

```
{numeAtribut:valAtribut,numeAtribut:valAtribut,...}
```

Un şir de valori se reprezintă prin

```
[valoare,valoare,...]
```

valAtribut, valoare poate fi un string, număr, true, false, null, o colecţie sau un şir.

JSON în JavaScript

Utilizarea entităţilor JSON în javascript este exemplificat în aplicaţia următoare.

Exemplul F.2.1

```
<HTML>
  <HEAD>
    <TITLE>Primul exemplu JavaScript</TITLE>
    <SCRIPT LANGUAGE="JavaScript">
    <!--
      var myJSONObj=[{"disciplina":"Analiza Numerica"},
      {"disciplina":"Programare distribuita"},
      {"disciplina":"Soft matematic"}];
      for(var i=0;i<myJSONObj.length;i++){
        document.writeln("<br>");
        document.writeln(myJSONObj[i].disciplina);
      }
      document.writeln("<br>");
      var myObj=eval(myJSONObj);
      document.writeln(myObj.toString());
      for(var i=0;i<myObj.length;i++){
        document.writeln("<br>");
        document.writeln(myObj[i].disciplina);
      }
    //-->
    </SCRIPT>
  </HEAD>
  <BODY>
  </BODY>
</HTML>
```

JSON în Java

Există mai multe pachete pentru transformarea unei componente Java în / din fişier Json:

- *google-gson*
- *javax.json*
- *apache-jonhzon*

google-gson

Analogul aplicaţiei javascript de mai sus, poate fi

Exemplul F.2.2

```
import com.google.gson.Gson;
import com.google.gson.reflect.TypeToken;
import java.lang.reflect.Type;
import java.util.Collection;
import java.util.Iterator;
```

```
7  class Disciplina{
8    private String nume;
9    Disciplina(){}

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

Interfaţa `javax.json.JsonValue` introduce obiectele nemodificabile (*immutable*) `JsonArray, JsonObject, JsonString, JsonNumber, JsonValue.TRUE, JsonValue.FALSE, JsonValue.NULL`.

Obiectele se instanţiază prin intermediul unor metode statice ale clasei `javax.json.Json`.

Există structura de interfeţe

`JsonValue`	`JsonStructure`	`JsonArray`
		`JsonObject`
	`JsonString`	
	`JsonNumber`	acoperă tipurile de date Java numerice `BigDecimal, BigInteger, int, long, double`

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*`)`

Interfaţa `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()`

Şablon de utilizare

```
JsonObject jsonObject=Json.createObjectBuilder()
    .add("name", value)
    . . .
    .build();
```

Interfaţa `javax.json.JsonArrayBuilder`

Metode

- `JsonArrayBuilder add(`*TipJson value)*

  TipJson∈{`BigDecimal, BigInteger, int, long, double, boolean, JsonObjectBuilder, JsonArrayBuilder, JsonValue, String`}.
- `JsonArrayBuilder addNull()`
- `JsonArray build()`

Şablon de utilizare

```
JsonArray jsonArray=Json.createArrayBuilder()
    .add(value)
    . . .
    .build();
```

Interfaţa `javax.json.WriterBuilder`

Metode

- `void writeArray(JsonArray` *array)*
- `void writeObject(JsonObject` *object)*
- `void close()`

Şablon de utilizare

```
PrintWriter printWriter=new PrintWriter(System.out)
JsonWriter jsonWriter=Json.createWriter(printWriter);
jsonWriter.writeArray(jsonArray);
jsonWriter.close();
```

Interfaţa `javax.json.ReaderBuilder`

Metode

- `void readArray()`
- `void readObject()`
- `void close()`

Şablon de utilizare

```
String string=. . .
JsonReader jsonReader = Json.createReader(new StringReader(string));
JsonArray array = jsonReader.readArray();
jsonReader.close();
```

Exemplul F.2.3 *Crearea unui fişier 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 fs=System.getProperty("file.separator");
18     JsonArray array=null;
19     try{
20       String path = new java.io.File( "." ).getCanonicalPath();
21       JsonReader jsonReader =
22         Json.createReader(new FileReader(path+fs+fileName));
23       array = jsonReader.readArray();
24       jsonReader.close();
25     }
26     catch(IOException e){
```

```
27        System.out.println("Ex : "+e.getMessage());
28      }
29      analyse(array);
30    }

32    private static void analyse(JsonArray v){
33      Iterator<JsonValue> iterator=v.iterator();
34      while(iterator.hasNext()){
35        JsonValue value=iterator.next();
36        if(value instanceof JsonArray){
37          JsonArray array=(JsonArray)value;
38          analyse(array);
39        }
40        if(value instanceof JsonObject){
41          JsonObject obj=(JsonObject)value;
42          analyseJsonObject(obj);
43        }
44        if(value instanceof JsonString){
45          JsonString string=(JsonString)value;
46          String s=string.getString();
47          System.out.println(s);
48        }
49        if(value instanceof JsonNumber){
50          JsonNumber number=(JsonNumber)value;
51          double d=number.doubleValue();
52          System.out.println(d);
53        }
54      }
55    }

57    private static void analyseJsonObject(JsonObject obj){
58      Map<String,JsonValue> object=(Map<String,JsonValue>)obj;
59      Set<String> keys=object.keySet();
60      Iterator<String> iter=keys.iterator();
61      while(iter.hasNext()){
62        String name=iter.next();
63        System.out.println();
64        System.out.println("JsonObject name : "+name);
65        JsonValue vv=object.get(name);
66        if(vv instanceof JsonArray){
67          JsonArray array=(JsonArray)vv;
68          analyse(array);
69        }
70        if(vv instanceof JsonObject){
71          JsonObject o=(JsonObject)vv;
72          analyseJsonObject(o);
73        }
74        if(vv instanceof JsonString){
75          JsonString string=(JsonString)vv;
76          String s=string.getString();
77          System.out.println(s);
78        }
79        if(vv instanceof JsonNumber){
80          JsonNumber number=(JsonNumber)vv;
81          double d=number.doubleValue();
82          System.out.println(d);
83        }
84      }
85    }
```

```
86 }
```

Anexa G

Adnotări

O *adnotare* este o completare, o notă sau o însemnare care explică sau întregeşte 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 entităţi din codul Java).

O adnotare poate să-şi facă efectul:

- asupra codului sursă, înaintea compilării;
- asupra codului obiect, după compilare, dar înaintea executării;
- în timpul execuţiei 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.*;
```

modificator `@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 fişier 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, interfaţă, metodă, câmp.
Înaintea declarării elementului asupra căruia acţionează, adnotarea se indică prin

@NumeAdnotare(numeElement_1=valoare,numeElement_2=valoare,. . .)

G.3 Procesarea unei adnotări

În Java sunt predefinite adnotările

Override	Target
Deprecated	Retention
SuppressWarnings	Documented
SafeVarargs	Inherited
Repeatable	FunctionalInterface

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    }

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 afişarea 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 dispariţia mesajelor de avertisment.

Adnotarea `SuppressWarnings`

Adnotarea `SuppressWarnings` inhibă afişarea mesajelor de avertisment. Reluăm exemplul anterior schimbând clasa *Test.* În urma compilării nu mai apar mesajele de avertisment menţionate mai sus.

```
1 @SuppressWarnings("deprecation")
2 public class TestSuppressWarnings{
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 acţionează:
`ElementType.TYPE`
`ElementType.FIELD`
`ElementType.METHOD`
`ElementType.PARAMETER`
`ElementType.CONSTRUCTOR`
`ElementType.LOCAL_VARIABLE`
`ElementType.ANNOTATION_TYPE`

Adnotarea `Retention`

Adnotarea `Retention` precizează momentul acţiunii adnotării:
`RetentionPolicy.SOURCE`
`RetentionPolicy.CLASS`
`RetentionPolicy.RUNTIME`

Adnotarea `Documented`

Adnotarea `Documented` are ca efect menţionarea adnotării în documentul obţinut prin `javadoc`.

Fie clasele

```
1 import java.lang.annotation.Documented;
2
3 @Documented
4 public @interface MyDocumented{}
```

şi

```
1  public class TestDocumented{
2    public static void main(String[] args){
3      new TestDocumented().doDocumented();
4    }
5
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
```

Adnotarea `SafeVarargs` (jdk 7)

Adnotarea `SafeVarargs` inhibă unele mesaje de atenţionare ale compilatorului.

```
1  public class TestSafeVarargs{
2    @SafeVarargs
3    static <T> T[] asArray(T... args) {
4      return args;
5    }
6
7    static <T> T[] arrayOfTwo(T a, T b) {
8      return asArray(a, b);
9    }
10
11   public static void main(String[] args) {
12     String[] bar = arrayOfTwo("hello", "world");
13     System.out.println(bar[0]+" "+bar[1]);
14   }
15 }
```

Execuţia generează o excepţie `ClassCastException`.

```
1  public class TestSafeVarargs1{
2    @SafeVarargs
3    // static <T> T[] asArray(T... args) {
```

```
4     static <T> String[] asArray(T... args) {
5       String[] sargs=new String[args.length];
6       int i=-1;
7       for(int j=0;j<args.length;j++){
8         if(args[j] instanceof String){
9           i++;
10          sargs[i]=(String)args[j];
11        }
12      }
13    return sargs;
14  }

16  //static <T> T[] arrayOfTwo(T a, T b) {
17  static <T> String[] arrayOfTwo(T a, T b) {
18    return asArray(a, b);
19  }

21  public static void main(String[] args) {
22    String[] bar = arrayOfTwo("hello", "world");
23    System.out.println(bar[0]+" "+bar[1]);
24  }
25 }
```

Procesarea unei adnotări se referă în primul rând la regăsirea în momentul execuţiei a valorilor elementelor adnotării. În funcţie de valorile regăsite se pot implementa acţiuni specifice.

Procesarea unei adnotări se bazează pe metodele interfeţei `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;

3  public class TestAnnotation{
4    public static void main(String[] args){
5      TestAnnotation obj=new TestAnnotation();
6      obj.verif(obj);
7    }

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 execuţiei. Metoda
`public Method getMethod(String` *name,* `Class<?>...` *parameterTypes*`) throws NoSuchMethodException, SecurityException`
furnizează o reprezentare (reflectare) a unei metode în timpul execuţiei unui program.

Probleme

Exemple pentru adnotările `Repeatable, FunctionalInterface` (jdk8).

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 şi utilizare pentru crearea şi exploatarea unei baze de date corespunzătoare unei agende de adrese e-mail.

H.1 *Derby / Javadb*

Instalarea produsului constă în dezarhivarea fişierului descărcat.

Utilizarea produsului. Va fi utilizată varianta de *reţea* bazată pe un server al SGBD care utilizează implicit portul 1527.

Se întreprind următoarele operaţii:

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
   ```

   Prezenţa opţiunii `-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 distribuţia 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 fişierul *CreateAgendaE.sql* este

```
connect 'jdbc:derby:AgendaEMail;create=true';
create table adrese(
  id int generated always as identity(start with 1, increment by 1)
    primary key,
  nume char(20) not null,
  email char(30) not null
);
```

şi încărcarea cu date

```
run 'ValuesAgendaE.sql';
```

cu fişierul *ValuesAgendaE.sql*

```
insert into adrese(nume,email) values('aaa','aaa@yahoo.com'),
  ('bbb','bbb@gmail.com'),('ccc','ccc@unitbv.ro'),
  ('aaa','xyz@unitbv.ro');
```

H.2 *mysql*

Instalarea produsului. Pentru instalare s-a descărcat varianta fără instalare automată *mysql-*.*.*-win*.zip*. Acest fişier se dezarhivează într-un catalog `MYSQL_HOME`.

Dezarhivarea este urmată de iniţializare

- 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 parole iniţială şi 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 fişierele bazelor de date vor fi găzduite în catalogul `MYSQL_HOME\data`.

Pentru utilizarea în aplicaţii Java trebuie descărcat un conector *mysql-connector-java-*.*.*.tar.gz,* conţinând fişierul `mysql-connector-java-*.*.*-bin.jar`.

Utilizarea produsului.

Se întreprind următoarele operaţii:

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 fişierului 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 opreşte 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 şi 3 trebuie să conţină opţiunea `-p`. Prin această opţiune se cere autentificarea prin introducerea parolei.

H.3 Şablonul de utilizare a unei baze de date într-un program Java

Interacţiunea cu baze de date relaţionale implică:

1. Stabilirea corespondenţei dintre date aflate în obiecte şi atribute / tabele (*object to relational database mapping*).
2. Apelarea acţiunilor CRUD (*Create, Read, Update, Delete*).

Limbajul SQL (*Structured Query Language*) este dependent de SGBD utilizat. Dezvoltarea interacţiunii dintre un program Java (client) cu o bază de date dintr-un SGBD s-a născut din dorinţa de a asigura independenţa *stratului* Java de SGBD. Soluţia găsită constă în introducerea unui strat suplimentar, între Java şi SGBD care asigură corespondenţa între obiectele Java cu tabelele unei baze de date şi permite o configurare simplă la schimbarea SGBD. Astfel se foloseşte terminologia de aplicaţie multistrat.

Materializarea acestor idei (*Object Relational Mapping* - ORM) se află în

- interfaţa de programare (API) *Java Persistence API* (JPA);

  Există mai multe implementări JPA.
- interfaţa de programare (API) *Java Data Object* (JDO);
- produsul *Hibernate.*

  Hibernate conţine şi 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 abstracţie de modelele menţionate anterior şi vom trata în modul cel mai simplu realizarea unei aplicaţii care interacţionează 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 cunoaşterea:

- driver-ului de acces la sistemul de gestiune a bazei de date (SGBD)

Tip SGBD	Driver	Fişierul 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 (distribuţia derby)
postgresql	org.postgresql.Driver	postgresql-*.*-*.jdbc4.jar
hypeqsql	org.hsqldb.jdbcDriver	hsqldb.jar
H2	org.h2.Driver	h2-*.jar
oraclexe	oracle.jdbc.driver.OracleDriver	ojdbc14.jar

  Dacă într-un servlet se realizează o conexiune la o bază de date atunci fişierul driver-ului trebuie copiat în catalogul `lib` al aplicaţiei.

- adresa URL a bazei de date (String *URLBazaDate*), sub forma

Tip SGBD	Referinţă 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:hsql://host/numeBazaDate
H2	jdbc:h2:tcp://host/numeBazaDate
oracle	jdbc:oracle:thin:@host:1521:XE

Şablonul 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 fixaţi parametrii *username* şi *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 obţine 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=DriverManager.getConnection(jdbcURL,username,password);

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;

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=instructiune.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=instructiune.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 }
```

```
1 module simpledb{
2   requires java.sql;
3 }
```

Fraza *select* şi interogarea se mai putea programa prin

```
String sql="select * from adrese where nume =?";
PreparedStatement prepStmt=con.prepareStatement(sql);
prepStmt.setString(1,nume);
```

```
RezultSet rs=prepStmt.executeQuery();
. . .
prepStmt.close();
```

În acest caz, valoarea variabilei `nume` este fără apostroafe.

Compilarea şi execuţia programului necesită declararea în variabila `classpath` a fişierelor

- Varianta *Derby*

  `derbyclient.jar` din catalogul `%DERBY_INSTALL%\lib`.

- Varianta *mysql*

  `%MYSQL_CONNECTOR_JAVA_HOME%\mysql-connector-java-*.*.*-bin.jar`

Execuţia programului presupune serverul SGBD activ.

Anexa I

Injectarea dependinţelor

Injectarea dependinţelor (*Dependency Injection - DI*) costă în oferirea spre utilizare a unor obiecte instanţiate de mediul de lucru (server Web, server de aplicaţii, container specializat) de către o clasă.

Injectarea dependinţelor este o tehnică uzuală în JEE, dar poate fi programată şi utilizată şi înafara unui cadru JEE. Produse informatice ce asigură această facilitate sunt:

- *Weld* realizat de *Jboss - RedHat* şi utilizat de Glassfish.
- *Guice* realizat de *Google.*

I.1 Weld

Exemplificăm pe aplicaţia simplă de calcul a celui mai mare divizor comun. Obiectul ce se va injecta este instanţă a clasei

```
1 package cmmdc;
2
3 public class Cmmdc{
4   public long cmmdc(long m,long n){. . .}
5 }
```

Aplicaţie de sine stătătoare

Structura aplicaţiei este

```
catalogul_applicatiei
|--> 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 }
```

Fişierul `beans.xml` este

```
1  <beans></beans>
```

Servlet

Structura aplicaţiei 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*

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://java.sun.com/xml/ns/javaee"
   xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
   http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
   id="WebApp_ID" version="3.0">

   <display-name>CDI Web Application</display-name>
    <listener>
      <listener-class>
         org.jboss.weld.environment.servlet.Listener
      </listener-class>
   </listener>
</web-app>
```

- *beans.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<beans 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/beans_1_0.xsd">
</beans>
```

Partea IV

TEME DE LABORATOR

Anexa J

Teme de aplicaţii

J.1 Probleme propuse

1. Să se realizeze conversia temperaturii exprimată în grade Celsius în grade Fahrenheit şi invers. Formula de transformare este

$$t_F = 1.8t_C + 32^oF$$

2. Să se realizeze conversia a unei sume de bani între USD, EURO şi RON. x Rata zilnică de schimb preia dinamic sub forma unui fişier `xml`, apelând *http://www.bnr.ro/nbrfxrates.xml.*

3. Să se realizeze conversia unui număr natural, din cifre arabe în cifre romane şi 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 subunitar dintr-o bază în alta.

   Codul Java

   ```
   Integer.toString(Integer.parseInt(numar, bazavVeche, bazaNoua);
   ```

   realizează conversia unui număr natural dintr-o bază în alta.

5. Un server adună la un număr (iniţializat cu 0) o valoare trimisă de un client returnând rezultatul. Există o singură instanţă a numărului, acelaşi 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 fişier text se introduc într-o colecţie Java. Rezolvarea cererii unui client se face utilizând facilităţile de programare oferite de interfaţa `java.util.Collection` din `jdk`.

7. Se consideră baza de date *UNITAŢI_DE_MĂSURĂ* formată din tabelul

CONVERSIE
UM_SI
UM_SIMBOL
DENUMIRE (ROM)
VAL

   Exemplu:

1	M	INCH	Ţol	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 aplicaţie pentru conversia unităţilor de măsură extinzând conţinutul 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 întreţinere şi interogare a bazei de date.

Sistemului informatic se cere să furnizeze următoarele date:

- Cine predă materia X?
- Ce materii predă cadrul didactic Y?

Baza de date:

CADRU_DIDACTIC

1	*aaa*
2	*bbb*
3	*ccc*
4	*ddd*

MATERIA

1	POO 1
2	Baze de date
3	Inteligenţa artificială
4	Probabilităţi şi statistică

CURSURI

1	1	1
2	1	2
3	2	4
4	3	1
5	4	3

9. Se consideră baza de date *APROVIZIONARE* formată din tabelele

RESURSE	FURNIZORI	CONTRACTE
COD-RESURSA	COD-FURNIZOR	COD-CONTRACT
DENUMIRE	NUME	COD-FURNIZOR
		COD-RESURSA
		CANTITATE

Să se elaboreze programe de întreţinere şi interogare a bazei de date.

Scenariu: Un centru comercial vinde lapte ($prod_1$), smântână ($prod_2$), spaghete ($prod_3$), bere ($prod_4$), etc. care sunt achiziţionate de la Fabrica de produse lactate A (fur_1), Fabrica de bere B (fur_2), Ferma agricolă C (fur_3), Fabrica de paste făinoase D (fur_4), etc.

Între centrul comercial şi furnizori se încheie contracte privind livrarea unei cantităti, preţul de cumpărare / vânzare, etc.

Sistemului informatic se cere să furnizeze următoarele date:

- Care sunt produsele cumpărate de la furnizorul X?
- Care sunt furnizorii de la care se cumpără produsul Y?

Baza de date:

RESURSE

1	$prod_1$
2	$prod_2$
3	$prod_3$
4	$prod_4$

FURNIZORI

1	fur_1
2	fur_2
3	fur_3
4	fur_4

CONTRACTE

1	1	1	50
2	1	2	20
3	2	4	25
4	3	1	10
5	4	3	30

10. Se consideră baza de date *DESFACERE* formată din tabelele

PRODUSE	BENEFICIARI	CONTRACTE
COD-PRODUS	COD-BENEFICIAR	COD-CONTRACT
DENUMIRE	NUME	COD-BENEFICIAR
		COD-PRODUS
		CANTITATE

Să se elaboreze programe de întreţinere şi interogare a bazei de date.

Scenariu: O întreprindere / fabrică de unelte de mână distribuie / produce set şurubelniţe ($prod_1$), set chei ($prod_2$), cleşte ($prod_3$), ciocan ($prod_4$), etc. care sunt cumpărate de utilizatori Uzina de autoturisme A (ben_1), Uzina de reparaţii material rulant B (ben_2), Baza de desfacere en gros C (ben_3), etc.

Între producător şi cumpărător se încheie contracte privind livrarea unei cantităti, preţul de vânzare / cumpărare, etc.

Sistemului informatic se cere să furnizeze următoarele date:

- Care sunt produsele cumpărate de firma X?
- Care sunt firmele care cumpără produsul Y?

Baza de date:

PRODUSE

1	$prod_1$
2	$prod_2$
3	$prod_3$
4	$prod_4$

BENEFICIARI

1	ben_1
2	ben_2
3	ben_3

CONTRACTE

1	1	2	50
2	1	4	20
3	2	1	25
4	3	3	10
5	2	1	30

11. Validarea codului numeric personal.

Codul Numeric Personal (CNP) constituie numărul de ordine atribuit de Evidenţa populaţiei unei persoane la naştere, care se înscrie în actele şi certificatele de stare civilă şi se preia în celelate acte cu caracter oficial.

Structura unui CMP este

```
S AA LL ZZ JJ NNN 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 s}i 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 in Romania
9   persoane straine
```

Codul judeţului (JJ)

Cod	Judeţ	Cod	Judeţ	Cod	Judeţ
01	Alba	16	Dolj	32	Sibiu
02	Arad	17	Galaţi	33	Suceava
03	Argeş	18	Gorj	34	Teleorman
04	Bacău	19	Harghita	35	Timiş
05	Bihor	20	Hunedoara	36	Tulcea
06	Bistriţa-Năsăud	21	Ialomiţa	37	Vaslui
07	Botoşani	22	Iaşi	38	Vâlcea
08	Braşov	23	Ilfov	39	Vrancea
09	Brăila	24	Maramureş	40	Bucureşti
10	Buzău	25	Mehedinţi	41	Bucureşti-S1
11	Caraş-Severin	26	Mureş	42	Bucureşti-S2
12	Cluj	27	Neamţ	43	Bucureşti-S3
13	Constanţa	28	Olt	44	Bucureşti-S4
14	Covasna	29	Prahova	45	Bucureşti-S5
15	Dâmboviţa	30	Satu Mare	46	Bucureşti-S6
		31	Sălaj	51	Călăraşi
				52	Giurgiu

$NNN \in \{001 - 999\}$ Numerele din acest interval se împart pe judeţe, Birourilor de Evidenţa Populaţiei, astfel încât un anumit număr să nu fie alocat decât unei persoane născute în ziua respectivă.

Verificarea cifrei de control: Se foloseşte cheia de testare 279146358279. Primele 12 cifre ale CNP se înmulţesc 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 împărţirii este mai mic decât 10, atunci acesta va fi cifra de control. Dacă restul împărţirii este 10 atunci cifra de control este 1.

Să se elaboreze un program pentru verificarea CNP punând în evidenţă toate elementele.

12. Informaţiile unui aeroport civil privind decolările(plecări) şi 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 aplicaţie de furnizare a datelor către clienţi (interogarea bazei de date).

Sistemului informatic se cere să furnizeze următoarele date:

- Lista aterizărilor.
- Lista decolărilor.

Baza de date:

ACTIVITATE

1	Aterizare	Luni	10/20	RyanAir	Vienna
2	Aterizare	Marti	8/30	Tarom	Londra
3	Decolare	Luni	3/00	WizzAir	Sofia
4	Decolare	Miercuri	9/10	WizzAir	LaValetta
5	Decolare	Joi	14	RyanAir	Vienna
6	Decolare	Joi	16	Tarom	Londra
7	Aterizare	Vineri	11/00	WizzAir	Sofia
8	Aerizare	Vineri	14/10	WizzAir	LaValetta

13. Se dau n tipuri de monede $M_{x_1}, M_{x_2}, \ldots, M_{x_n}$, x_i reprezentând valoarea monedei, $i = 1, 2, \ldots, n$. 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 $k_1, k_2, \ldots, k_n (k_1 x_1 + \ldots + k_n x_n \geq S)$.

14. Globul pământesc este împărţit în 24 de fuse orare de câte $15^o (24 * 15 = 360)$ pornind de la meridianul 0 spre stânga şi dreapta.

    - Dându-se longitudinile a 2 puncte de pe glob să se calculeze diferenţa 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 coeficienţii 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 şi $n \in \mathbb{N}, 0 \leq n \leq 2^p - 1$ cu reprezentarea binară $n = \overline{a_{p-1}a_{p-2}\ldots a_1a_0}_2$. Să se determine numărul m care are reprezentarea binară $m = \overline{a_0a_1\ldots a_{p-2}a_{p-1}}_2$.

De exemplu, pentru $p = 6, n = 17 = \overline{010001}_2$ numărul căutat este $m = \overline{100010}_2 = 34$.

19. Să se determine numărul de apariţii al fiecărui cuvânt dintr-un text dat.

20. Să se calculeze B_n (Numărul lui Bernoulli) definit prin

$$B_n = \begin{cases} 1 & \text{dacă} \quad n = 0 \\ -\dfrac{\sum_{k=0}^{n-1} \binom{n+1}{k} B_k}{n+1} & \text{dacă} \quad n > 0 \end{cases}$$

Calculele se fac în $\mathbb{Q}$.

21. **Problema** $3x + 1$**.** Fie şirul de numere naturale $(a_n)_{n\in\mathbb{N}}$ definit prin formula de recurenţă

$$a_{n+1} = \begin{cases} 3a_n + 1 & \text{dacă} \quad a_n \text{ impar} \\ \frac{a_n}{2} & \text{dacă} \quad a_n \text{ par} \end{cases}$$

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 şi a_{n+p} determină un pas dacă sunt numere impare şi $a_{n+1}, \ldots, a_{n+p-1}$ sunt numere pare).

Pentru un a_0 dat să se determine numărul de paşi până la atingerea lui 1.

Exemplu. Pentru $a_0 = 7$ şirul este

$$7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1$$

iar numărul de paşi este 5.

22. Se definesc tablourile (T_n^d) şi (P_n^d) pentru $n \in \mathbb{N}^*,\ d \in \mathbb{N}$ prin formulele de recurenţă:

$$\begin{aligned} P_1^d &= 1, \quad d \in \mathbb{N} \\ P_{n+1}^d &= T_n^d, \quad d \in \mathbb{N},\ n \in \mathbb{N}^* \\ T_n^d &= \sum_{j=0}^{d} P_n^j \quad d \in \mathbb{N},\ n \in \mathbb{N}^* \end{aligned}$$

Să se calculeze T_n^d.

Exemplu.

$P_n^d =$

n \ d	0	1	2	3	4
1	1	1	1	1	1
2	1	2	3	4	5
3	1	3	6	10	15

$T_n^d =$

n \ d	0	1	2	3	4
1	1	2	3	4	5
2	1	3	6	10	15
3	1	4	10	20	35

23. Pentru $a, b \in \mathbb{N}$ să se calculeze $u, v \in \mathbb{Z}$ astfel încât $ua + vb = (a, b)$.

Indicaţie. Se poate utiliza algoritmul lui Euclid pentru calcul celui mai mare divizor comun a două numere naturale (Ion D.I., Niţă C., 1978, *Elemente de aritmetică cu aplicaţii în tehnici de calcul.* Ed. Tehnică, Bucureşti, 103-107).

24. Dacă $a \in \mathbb{Z}_n$ să se calculeze, dacă există, a^{-1}.

25. Să se rezolve ecuaţia $ax = b$ în $\mathbb{Z}_n$.

Exemplu. În $\mathbb{Z}_8$

(a) $3x = 5 \quad \Rightarrow \quad x = 7$

(b) $4x = 2 \quad \Rightarrow \quad x \in \emptyset$

(c) $2x = 4 \quad \Rightarrow \quad x \in \{2, 6\}$

26. Să se calculeze numerele lui Bell, B_n, definite prin

$$\begin{aligned} B_{n+1} &= \sum_{k=0}^{n} \binom{n}{k} B_k \\ B_0 &= 1. \end{aligned}$$

27. Fie $n, k \in \mathbb{N}^*,\ k \leq n$. Să se calculeze toate soluţiile formate din numere naturale ale sistemului

$$\begin{cases} x_1 + x_2 + \ldots + x_{n-k+1} &= k \\ 1x_1 + 2x_2 + \ldots + (n-k+1)x_{n-k+1} &= n \end{cases}$$

Exemplu. Pentru $n = 6$ şi $k = 3$ sistemul este

$$\begin{cases} x_1 + x_2 + x_3 + x_4 &= 3 \\ 1x_1 + 2x_2 + 3x_3 + 4x_4 &= 6 \end{cases}$$

şi are soluţiile $(2, 0, 0, 1), (1, 1, 1, 0), (0, 3, 0, 0)$.

28. Fie $0 < b < a$. Şirurile $(a_n)_{n\in\mathbb{N}}$ şi $(b_n)_{n\in\mathbb{N}}$ definite prin formulele de recurenţă

$$\begin{array}{ll} a_{n+1} = \frac{a_n + b_n}{2} & b_{n+1} = \sqrt{a_n b_n} \\ a_0 = a & b_0 = b \end{array}$$

converg către acelaşi limită, numită media aritmetică-geometrică a numerelor a şi b, notată prin $M(a, b)$.

Dându-se $a, b, \varepsilon > 0$ să se calculeze $M(a, b)$ cu precizia ε, abstracţie făcând de erorile de rotunjire (x aproximează pe a cu precizia ε dacă $|x - a| < \varepsilon$).

Bibliografie

[1] ALBOAIE L., BURAGA S., 2006, *Servicii Web.* Ed. Polirom, Iaşi.

[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 aplicaţii Web.* Ed. Albastră, Cluj-Napoca.

[4] BOIAN F.M., 2011, *Servicii Web; Modele, Platforme, Aplicaţii.* Ed. Albastră, Cluj-Napoca.

[5] BURAGA S.C., 2001, *Tehnologii Web.* Ed. Matrix Rom,Bucureşti.

[6] BURAGA S. (ed), 2007, *Programarea în Web 2.0.*, Ed. Polirom, Iaşi.

[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 reţelelor de calculatoare.* Ed. de Vest, Timişoara.

[10] LEŢIA S.T., 2002, *Programare avansată în Java.* Ed. Albastră, Cluj-Napoca.

[11] TANASĂ Ş., ANDREI Ş., OLARU C., 2011, *Java de la 0 la extert.* Ed. Polirom, Iaşi.

[12] TANASĂ Ş., OLARU C., 2005, *Dezvoltarea aplicaţiilor Web folosind Java.* Ed. Polirom, Iaşi.