




# Curs practic de Java

## *Curs 12*

Cristian Frăsinaru

`acf@infoiasi.ro`

Facultatea de Informatică  
**Universitatea "Al. I. Cuza" Iași**



# Java Web Start

# Cuprins

---

- Introducere
- Crearea unei aplicații
- Considerații generale
- JNLP API

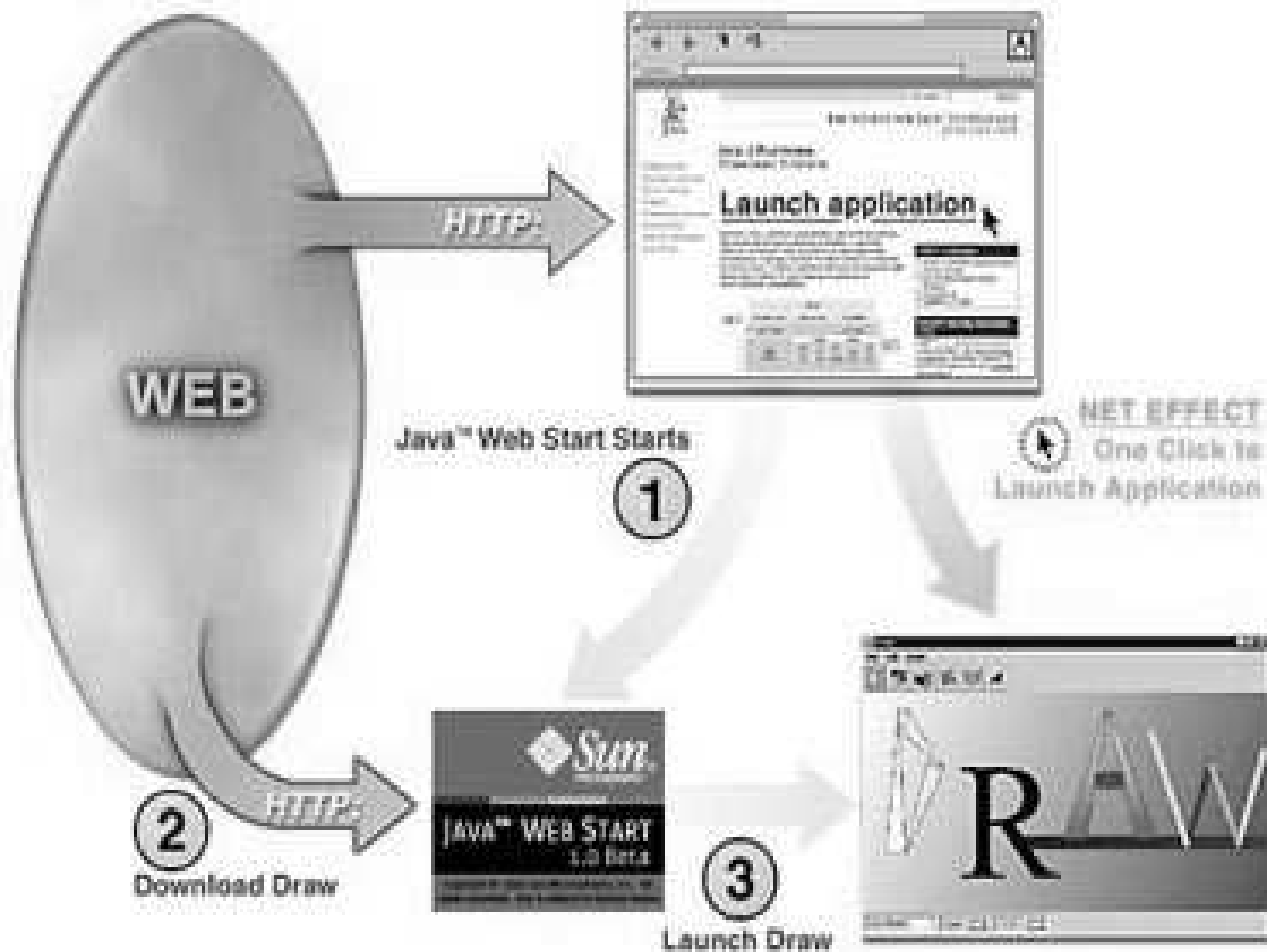
# Introducere

# Ce este Java Web Start ?

JWS reprezintă o tehnologie pentru **instalarea, lansarea, actualizarea aplicațiilor Java direct de pe Web.**

- Ce aplicații Java pot fi instalate astfel ?
- **Orice aplicație J2SE.**
- Ce trebuie să facă utilizatorul pentru a instala o aplicație Java astfel?
- **Un singur click** pe pagina Web unde este oferită aplicația de către producător (dezvoltator).

# Imaginea generală



# Cum funcționează JWS ? (1)

## Producătorul

1. **Creează** o aplicație Java **oarecare**.
2. **Arhivează** programul împreună cu toate resursele sale. Dacă aplicația solicită acces la resursele mașinii client, **semnează** jar-ul.
3. Creează un **fișier de configurare (.jnlp)** care descrie aplicația .
4. Creează o **pagină Web** cu o legătură către fișierul de configurare.
5. Plasează arhiva, fișierul jnlp și pagina Web pe un server Web.

# Cum funcționează JWS ? (2)

## Utilizatorul

1. Cu **un click** accesează legătura către aplicație de pe pagina Web a producătorului.
2. Aplicația va fi **instalată local automat** (copiată într-o zonă cache).
3. Dacă este necesară existența unui **anumit JRE**, acesta va fi **instalat automat**.
4. Aplicația va fi **lansată în execuție**.
5. La fiecare execuție ulterioară, se va verifica dacă nu există o **versiune nouă** pe pagina producătorului.



# Caracteristici

- **Parte integrantă a JRE.**
- Creată **exclusiv pentru platforma J2SE.**
- Suportă **versiuni multiple** de platforme standard
- Aplicațiile pot fi lansate din **browser sau local**
- Beneficiază de sistemul de **securitate** Java
- Utilizează **protocolul JNLP** (Java Network Launching Protocol), fiind implementarea de referință a acestuia.



# Crearea și configurarea unei aplicații



# Crearea aplicației

1. Crearea unei aplicații care să poată fi instalată cu JWS respectă aceleași etape ca orice aplicație standard Java, ținând însă cont de eventualele restricții la care poate fi supusă aceasta.
2. Arhivarea claselor și a resurselor necesare.
3. Eventual, semnarea arhivei.

# Instalarea pe server a aplicației

## 1. Configurarea serverului Web să utilizeze tipul MIME specific JWS.

```
<!-- Apache: fisierul .mime.types -->
<!-- Tomcat: fisierul web.xml -->
<mime-mapping>
  <extension>jnlp</extension>
  <mime-type>application/x-java-jnlp-file</mime-type>
</mime-mapping>
```

## 2. Crearea fișierului JNLP

## 4. Crearea paginii Web

```
<html> <body>
  <a href=http://www.infoiasi.ro/~acf/jws/aplicatie.jnlp>
    Lanseaza aplicatia!
  </a>
</body> </html>
```

# Sintaxa fișierului JNLP

Fișierul de configurare jnlp este în **format XML** și oferă informații legate de aplicația ce va fi instalată folosind JWS. Structura sa este:

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.5+" href="aplicatie.jnlp"
      codebase="http://www.infoiasi.ro/~acf/jws">
  <information> ... </information>
  <security>
    <all-permissions/>
  </security>
  <resources>
    <j2se version="1.4.2+"/>
    <jar href="lib/aplicatie.jar"/>
  </resources>
  <application-desc main-class="Main"/>
</jnlp>
```

# Informații

```
<information>
  <title>Aplicatie demonstrativa</title>
  <vendor>Sun Microsystems, Inc.</vendor>
  <homepage href="docs/help.html"/>
  <description>Aplicatie demonstrativa</description>
  <icon href="images/logo.jpg"/>
  <icon kind="splash" href="images/splash.gif"/>
  <offline-allowed/>
  <shortcut online="false">
    <desktop/>
    <menu submenu="Aplicatie "/>
  </shortcut>
</information>
```



# Considerații legate de dezvoltarea aplicațiilor



# Condiții generale

- Aplicația trebuie să fie distribuită sub forma unor arhive jar.
- Resursele aplicației trebuie incluse în arhive.
- Aplicația poate avea restricții severe pe mașina utilizatorului:
  - nu este permis accesul la disc
  - nu pot fi lansate procese
  - nu pot fi accesate librării native, etc.
- Aplicația are nevoie de acces nerestricționat - arhivele jar trebuie semnate.



# Folosirea resurselor

Resursele aplicației (imagini, sunete, etc) trebuie arhivate (eventual chiar împreună cu aplicația). Referirea lor nu va putea fi făcută în mod uzual.

```
// Obținem class-loaderul
ClassLoader cl = this.getClass().getClassLoader();

// Obținem resursele
Icon openIcon  = new ImageIcon(cl.getResource("images/open.gif"));
Icon saveIcon  = new ImageIcon(cl.getResource("images/save.gif"));
...
```

# Securitate



## Scopul

Protejarea utilizatorilor (companiilor) împotriva unor aplicații de rețea care ar putea afecta integritatea sistemelor lor sau ar putea sustrage date confidențiale.

## Mecanisme

- Rularea programelor de rețea într-un mediu sigur (*sandbox*), întocmai ca appleturile
- Semnarea arhivelor din care este formată o aplicație ce necesită acces la sistemul local al utilizatorului, iar acesta să poată hotărâ dacă acordă sau nu permisiunile cerute.



# Solicitarea permisiunilor

Solicitarea permisiunilor va fi făcută în cadrul tagului **security** din fișierul de configurare jnlp.

```
<security>  
  <all-permissions/>  
</security>
```

# Semnarea arhivelor jar

## 1. Crearea unei chei

```
keytool -genkey -keystore cheie -alias georgel
```

## 2. Crearea unui certificat

```
keytool -selfcert -alias georgel -keystore cheie
```

## 3. Semnarea arhivei jar

```
jarsigner -keystore cheie aplicatie.jar georgel
```

# JNLP API

# Ce reprezintă JNLP API ?

JNLP API oferă soluții de utilizare a mecanismelor specifice JWS, nedisponibile în J2SE API. Acestea se prezintă sub forma unor **servicii**, disponibile aplicației indiferent de nivelul de restricții impus acesteia.

Clasele și interfețele din JNL API se găsesc în arhiva **jnlp.jar**, inclusă în kitul standard JDK 1.5.0 (în directorul `sample/jnlp`).

# Tipuri de servicii

- BasicService
- ClipboardService
- DownloadService
- FileOpenService
- FileSaveService
- PrintService
- PersistentService
- SingleInstanceService
- ExtendedService

# Folosirea serviciilor



```
// 1. Importul pachetului javax.jnlp
import javax.jnlp.*;

try {
    // 2. Identificarea serviciului
    XService s = (XService)ServiceManager.lookup("javax.jnlp.XService");

    // 3. Folosirea serviciului

} catch(UnavailableServiceException ue) {
    // Serviciul nu exista
}
```

XService poate fi oricare din serviciile disponibile în pachetul javax.jnlp.





# BasicService



Oferă un set de metode pentru interacțiunea cu mediul de execuție al aplicației. Este similară cu clasa `AppletContext` pentru appleturi.

## Metode

```
getCodeBase  
isOffline  
isWebBrowserSupported  
showUrl
```



# ClipboardService

Permite accesarea clipboard-ului sistemului, chiar în condițiile în care aplicația rulează în *sandbox*

```
ClipboardService cs;
...
// Setarea clipboard-ului
StringSelection ss = new StringSelection("Java Web Start!");
cs.setContents(ss);

// Obținerea conținutului clipboard-ului
Transferable tr = cs.getContents();
if (tr.isDataFlavorSupported(DataFlavor.stringFlavor)) {
try {
    String s = (String)tr.getTransferData(DataFlavor.stringFlavor);
    System.out.println("Conținut clipboard: " + s);
} catch (Exception e) {
    e.printStackTrace();
}
```

# DownloadService

Permite aplicațiilor să controleze modul în care resursele lor sunt stocate local (în cache), să forțeze actualizarea sau eliminarea acestora.

```
DownloadService ds;
...
// Determina daca o anumita resursa este in cache
URL url = new URL("http://www.infoiasi.ro/~acf/jws/lib/aplicatie.jar");
boolean cached = ds.isResourceCached(url, "1.0");

// Elimina o resursa din cache
if (cached) {
ds.removeResource(url, "1.0");
}

// Reincarca resursa in cache
DownloadServiceListener dsl = ds.getDefaultProgressWindow();
ds.loadResource(url, "1.0", dsl);
```

# FileOpenService

Permite selectarea unui fișier din sistemul local de fișiere, chiar pentru aplicații care rulează în *sandbox*.

```
FileOpenService fos;  
...
```

```
String path = null;  
String[] ext = null;
```

```
// Selectarea unui singur fisier  
FileContents fc = fos.openFileDialog(path, ext);
```

```
// Selectare multipla de fisiere  
FileContents[] fcs = fos.openMultiFileDialog(path, ext);
```

# FileContents

Incapsulează numele și conținutul unui fișier, fiind clasa utilizată de serviciile JNLP ce lucrează cu fișiere, cum ar fi `FileOpenService`, `FileSaveService`, și `PersistenceService`.

```
FileOpenService fos;
...
FileContents fc = fos.openFileDialog(null, null);
// Copiem conținutul unui fișier într-un buffer
byte [] buf = new byte[fc.getLength()];
InputStream is = fc.getInputStream();
int pos = 0;
while ((pos = is.read(buf, pos, buf.length - pos)) > 0) { ... }
is.close();
// Scriem conținutul înapoi în fișier
if (fc.canWrite()) {
    OutputStream os = fc.getOutputStream(false);
    os.write(buf);
}
```

# FileSaveService

Permite utilizatorilor să salveze un fișier în sistemul local de fișiere, chiar pentru aplicații care rulează în *sandbox*.

```
FileSaveService fss;  
FileOpenService fos;  
...  
// Selectam un fisier oarecare  
FileContents fc = fos.openFileDialog(null, null);  
  
// Salvam continutul fisierului selectat in alt fisier  
FileContents newfc = fss.saveFileDialog(null, null,  
    fc.getInputStream(), "newFileName.txt");  
  
// O alta modalitate de a face aceeasi operatiune  
FileContents newfc2 = fss.saveAsFileDialog(null, null, fc);
```

# PrintService

Oferă utilizatorilor metode pentru accesarea facilităților locale de tipărire, chiar pentru aplicații care rulează în *sandbox*. Cererea de tipărire va fi afișată și, în cazul în care este acceptată, va fi efectuată.

```
PrintService ps;  
...  
PageFormat pf = ps.getDefaultPage();  
PageFormat newPf = ps.showPageFormatDialog(pf);  
ps.print(new DocToPrint());  
...  
  
class DocToPrint implements Printable {  
    public int print(Graphics g, PageFormat pageformat, int pageIndex) {  
        ...  
    }  
}
```

# PersistentService

Acest serviciu oferă metode pentru memorarea locală a unor date pe sistemul utilizatorului, chiar pentru aplicații care rulează în *sandbox*. Este similar cu utilizarea cookie-urilor, datele memorate folosind acest mecanism fiind de fapt o copie locală a unor informații aflate pe server. Fiecare intrare în tabela de persistență este are două câmpuri:

- un **URL**, ce referă informații necesare aplicației.
- un **tag**, ce furnizează informații despre starea datelor: `CACHED`, `TEMPORARY`, `DIRTY`

O aplicație are posibilitatea de a crea, actualiza, șterge informații în tabela de persistență.



# Folosirea persistenței

```
PersistenceService ps;  
BasicService bs;  
...  
// Obținerea tuturor articolelor din tabela  
URL url = bs.getCodeBase();  
String [] muffins = ps.getNames(url);  
  
// Actualizam articolele care sunt 'dirty'  
int [] tags = new int[muffins.length];  
URL [] muffinURLs = new URL[muffins.length];  
for (int i = 0; i < muffins.length; i++) {  
    muffinURLs[i] = new URL(codebase.toString() + muffins[i]);  
    tags[i] = ps.getTag(muffinURLs[i]);  
    if (tags[i] == PersistenceService.DIRTY) {  
        ps.setTag(muffinURLs[i], PersistenceService.CACHED);  
    }  
}
```

# JNLPRandomAccessFile



Permite scrierea, citirea datelor în fișiere.

```
FileOpenService fos;  
...  
// Deschidem un fisier  
FileContents fc = fos.openFileDialog(null, null);  
  
//Scriem o informatie la o pozitie specificata  
JNLPRandomAccessFile raf = fc.getRandomAccessFile("rw");  
raf.seek(raf.length() - 1);  
raf.writeUTF("Java Web Start!");  
raf.close();
```



# SingleInstanceService

Furnizează o modalitate de a înregistra aplicația ca un singleton și de a specifica eventual argumente la instanțiere aplicației. Reapelarea aplicației înainte de a o înregistra ca singleton va determina lansarea unei alte JVM.

```
SingleInstanceService sis;
...
// La inceputul aplicatiei
SISListener sisL = new SISListener();
sis.addSingleInstanceListener(sisL);
...
// La sfarsitul aplicatiei
sis.removeSingleInstanceListener(sisL);
System.exit(0);

// Implementam interfata SingleInstanceListener
class SISListener implements SingleInstanceListener {
    public void newActivation(String[] params) { ... }
}
```

# ExtendedService

Oferă suport adițional mecanismului de lucru cu fișiere, permițând aplicațiilor să deschidă un fișiere anume aflat pe sistemul utilizatorului.

```
ExtendedService es;
...
// Deschidem un fisier anume
File a = new File("c:\\unFisier.txt");
// Java Web Start va deschide un dialog pentru a cere permisiunea
// de scriere/ citire a fisierului specificat

FileContents fc_a = es.openFile(a);
// Deschidem mai multe fisiere
File[2] fArray = new File[2];
fArray[0] = a;
fArray[1] = new File("c:\\altFisier.txt");
FileContents[] fc_Array = es.OpenFiles(fArray);
```

# javaws la linia de comandă

**javaws [ optiuni ] [ URL ]**