Osservazioni

- Nell'esempio della precedente lezione del contatore di accessi "persistente" il nome del file in cui memorizzare il numero di "hit" ed il valore iniziale del contatore sono codificati direttamente nel codice della servlet.
- Se vogliamo cambiare il nome del file o il valore iniziale del contatore o il separatore, dobbiamo *modificare* il sorgente, *ricompilare* e *riavviare* la relativa applicazione web.

Variante

- Per evitare la ricompilazione della servlet, si può memorizzare il nome del file e/o il valore iniziale del contatore e/o il carattere separatore nel file di configurazione **`web.xml`**.
- Tale parametro può poi essere recuperato dalla servlet.
- Infatti il server (Tomcat) passa automaticamente al metodo **`init()`** un oggetto di tipo **`ServletConfig`** che include i parametri di inizializzazione accessibili tramite i metodi **`getInitParameter()`** e **`getInitParameterNames()`**.
- Sia **`GenericServlet`** che **`HttpServlet`** implementano l'interfaccia **`ServletConfig`**: è sufficiente quindi richiamare **`super.init()`** per avere a disposizione tali metodi.

Specificare i parametri in `web.xml`

```
<servlet>
    <servlet-name>Contatore3</servlet-name>
    <servlet-class>ContatorePersistente2</servlet-class>
    <init-param>
        <param-name>nomeFile</param-name>
        <param-value>contatore.txt</param-value>
    </init-param>
    <init-param>
        <param-name>valoreContatore</param-name>
        <param-value>0</param-value>
    </init-param>
    <init-param>
        <param-name>separatore</param-name>
        <param-value>;</param-value>
    </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>Contatore3</servlet-name>
    <url-pattern>/servlet/ConPersParam</url-pattern>
</servlet-mapping>
```

Accedere ai parametri memorizzati in **web.xml**

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.Date;
import java.text.*;

public class ContatorePersistente2 extends HttpServlet {
  private int i;

  public void init(ServletConfig conf) throws ServletException {
    super.init(conf);

    try {
      String path=getServletContext().getRealPath("")+File.separator+
                  getInitParameter("nomeFile");
```

```
// ... continua dal lucido precedente
    BufferedReader input=new BufferedReader(new FileReader(path));
    String lineaCorrente=null;
    String ultimaLinea=null;
    while((lineaCorrente=input.readLine()) != null) {
      ultimaLinea=lineaCorrente; }
    i=Integer.parseInt((ultimaLinea.
        split(getInitParameter("separatore")))[1]);
    input.close();
  } catch (IOException e) {
    i=Integer.parseInt(getInitParameter("valoreContatore"));
  }
}
```

Il resto del codice (I)

```
// ... continua dal lucido precedente
   public void doGet(HttpServletRequest req,
                     HttpServletResponse res)
                     throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("<HTML>");
        out.println("<HEAD><TITLE>Contatore di "+
                    "accessi</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("<STRONG>Accessi registrati "+
                    "finora: "+(++i)+"</STRONG>");
        out.println("</BODY></HTML>");
   }
```

Il resto del codice (II)

```
// ... continua dal lucido precedente
  public void destroy() {
    try {
      String path=getServletContext().getRealPath("")+
                  File.separator+getInitParameter("nomeFile");
      BufferedWriter output=new BufferedWriter(new
                                  FileWriter(path,true));
      DateFormat formatoData=new SimpleDateFormat("dd/MM/yyyy");
      output.write(formatoData.format(new
   Date())+getInitParameter("separatore")+i+"\n");
      output.close();
    } catch (IOException e) { }
  }
}
```

Quindi il resto del codice rimane pressoché invariato.

Modifiche al file `index.html`

```
<HTML>
<HEAD>
<TITLE>Servlet di prova</TITLE>
</HEAD>

<BODY>
  <TABLE>
    . . .
    <TR>
      <TD>
        <A HREF="servlet/ConPersParam">Un contatore di accessi
          persistente con parametri memorizzati in web.xml</A>
      </TD>
    </TR>
  </TABLE>
</BODY>

</HTML>
```

Note

- Memorizzando i parametri fondamentali di una servlet nel file che descrive il dispiegamento (**`web.xml`**), si evita di ricompilare il codice in caso di cambiamenti.
- Tuttavia è **sempre necessario riavviare** (Stop + Start) l'applicazione web tramite il Tomcat Manager.

File Upload

- Vogliamo realizzare una servlet che ci consenta di caricare (upload) un file sul server.
- Operazioni preliminari:
  - rendere la directory delle proprie servlet scrivibile da tomcat:

    `cd; chmod g+w servlets`
  - scaricare e copiare in **`$TOMCAT_HOME/common/lib`** il package Commons FileUpload da

    `http://jakarta.apache.org/commons/fileupload/`

    ed il package Commons IO da

    `http://jakarta.apache.org/commons/io/`

    (su **`latoserver.dimi.uniud.it`** sono già presenti).

Il form (`file.html`)

```
<HTML>
<HEAD>
<TITLE>File Upload Form</TITLE>
</HEAD>

<BODY>
  <FORM METHOD="post" ACTION="servlet/FileUpload" ENCTYPE="multipart/form-data">
    Scegli il file da caricare:
    <INPUT TYPE="file" NAME="file">
    <P>
    Breve descrizione:
    <INPUT TYPE="text" NAME="desc" SIZE="50">
    <P>
    <INPUT TYPE="submit" VALUE="Invia &gt;&gt;">
  </FORM>
</BODY>

</HTML>
```

ENCTYPE="multipart/form-data"

- Si immagini di cliccare sul tasto “Sfoglia…” e di selezionare il file di testo prova.txt contenente la stringa **`ciao, mondo!`**, scrivendo “Prova di upload” nel campo descrizione; ecco come apparirebbe il corpo della richiesta HTTP generata al momento della pressione del pulsante di sottomissione del form:

```
-----------------------------7d6222212903d4
Content-Disposition: form-data; name="file"; filename="C:\Web\Test\prova.txt"
Content-Type: text/plain

ciao, mondo!
-----------------------------7d6222212903d4
Content-Disposition: form-data; name="desc"

Prova di upload
-----------------------------7d6222212903d4
Content-Disposition: form-data; name="invia"

Invia >>
-----------------------------7d6222212903d4--
```

Il codice Java (I)

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.List;
import java.util.Iterator;
import org.apache.commons.fileupload.*;
import org.apache.commons.fileupload.servlet.*;
import org.apache.commons.fileupload.disk.*;

public class FileUpload extends HttpServlet {
  public void doPost(HttpServletRequest req, HttpServletResponse res)
              throws ServletException, IOException {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.println("<HTML>");
    out.println("<HEAD><TITLE>File Upload Servlet</TITLE></HEAD>");
    out.println("<BODY style=\"font-size: 20\">");
    FileItemFactory factory=new DiskFileItemFactory();
```

Nuovi package

Inizializzazione

Il codice Java (II)

```
// ... continua dal lucido precedente

    try {
      ServletFileUpload upload=new ServletFileUpload(factory);
      List fileItems=upload.parseRequest(req);
      Iterator it=fileItems.iterator();

      while(it.hasNext()) { // Scansione di tutti i campi del form
        FileItem f=(FileItem)it.next();
        if(f.isFormField()) { // Se il campo è di tipo testuale...
          out.println("Valore del campo testuale "+f.getFieldName()+
                      ": " +f.getString()+"<BR>");
```

Il codice Java (III)

```
    } else {// Se il campo contiene un file...
      char carattereSeparatore=File.separatorChar;
      String separatore="";

      if(carattereSeparatore=='\\') separatore="\\\\";
      else separatore=File.separator;

      String[] partiPercorso=f.getName().split(separatore);
      File file=new File(getServletContext().getRealPath("/"),
                         partiPercorso[partiPercorso.length-1]);
      f.write(file);
      out.println("File "+f.getName()+" salvato in "+
                  file.getAbsolutePath()+"<BR>");
     }
   }
} catch(Exception e) { out.println("Errore: "+e.toString()+"<BR>");}
out.println("</BODY></HTML>");
}}
```

Operazione necessaria (altrimenti split non opererebbe correttamente nei sistemi Windows)

Consideriamo l'ultima parte del percorso, ovvero, il nome del file

Compilazione

- In fase di compilazione bisogna tener conto di includere nel classpath il nuovo jar:
  - in ambiente Windows:

```
javac -classpath "c:\Programmi\Apache Software Foundation\Tomcat
5.5\common\lib\servlet-api.jar;c:\Programmi\Apache Software
Foundation\Tomcat 5.5\common\lib\commons-fileupload-1.1.jar"
FileUpload.java
```

  - in ambiente Linux (latoserver.dimi.uniud.it):

```
javac FileUpload.java
```

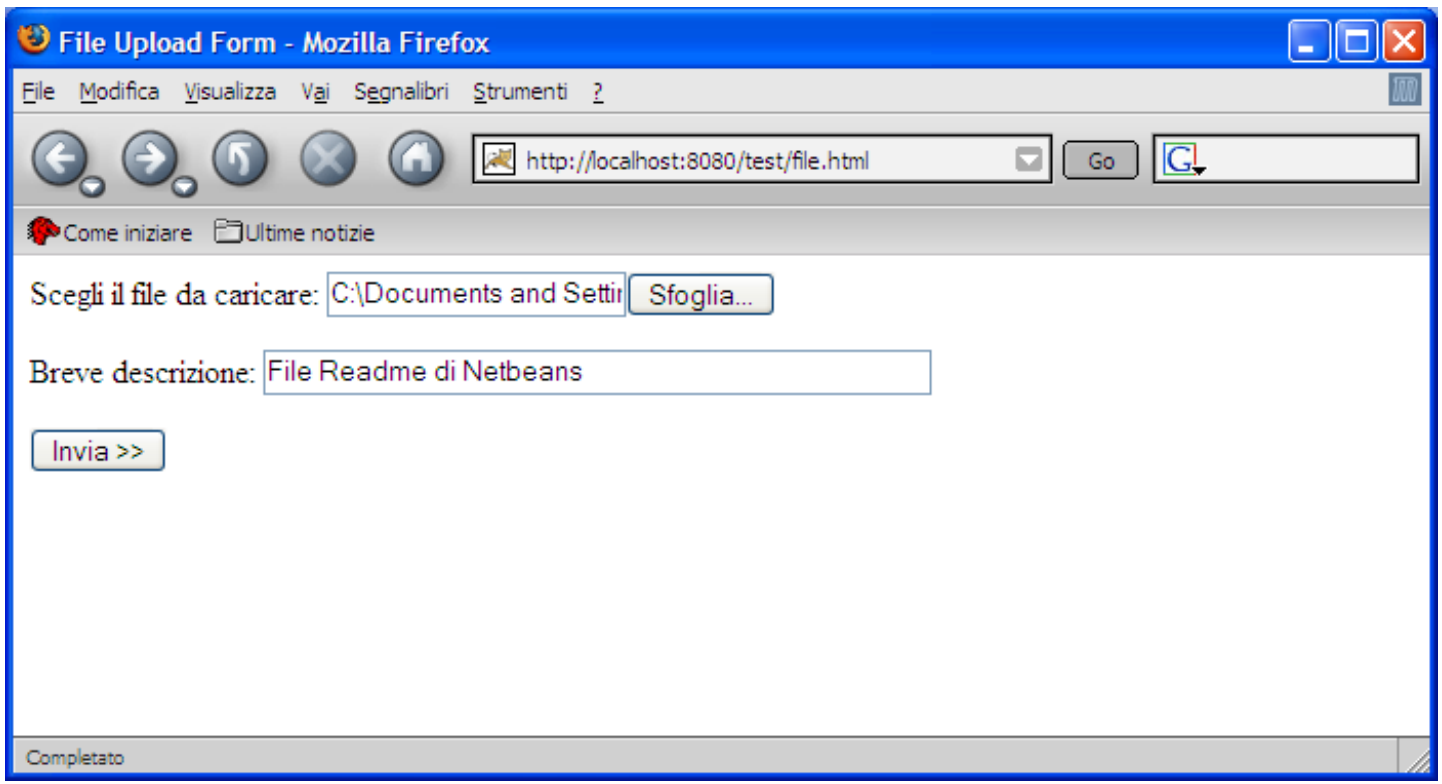
Modifiche al file **web.xml**

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!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>
  . . .
    <servlet>
        <servlet-name>FileUpload</servlet-name>
        <servlet-class>FileUpload</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>FileUpload</servlet-name>
        <url-pattern>/servlet/FileUpload</url-pattern>
    </servlet-mapping>
</web-app>
```

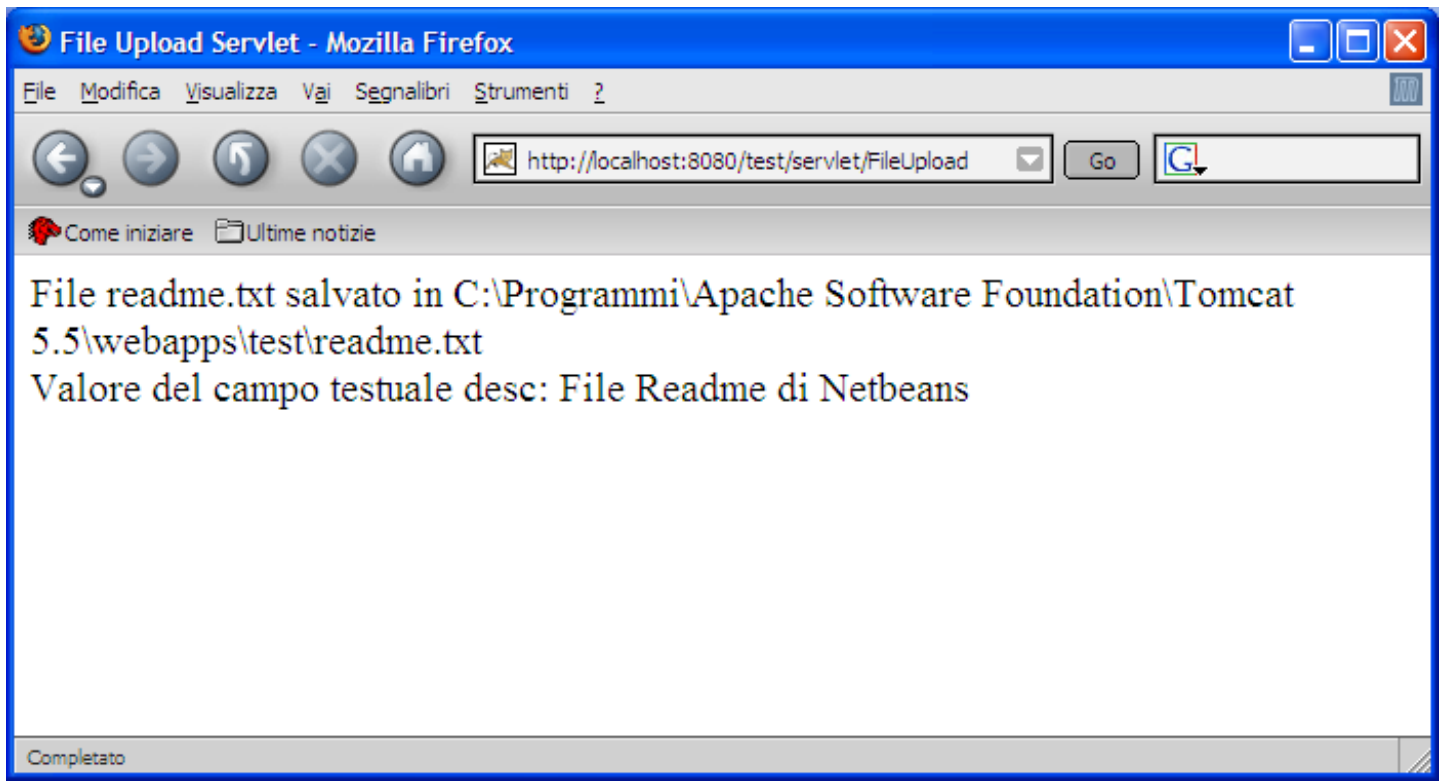
Modifiche al file `index.html`

```
<HTML>
<HEAD>
<TITLE>Servlet di prova</TITLE>
</HEAD>

<BODY>
  <TABLE>
     . . .
    <TR>
      <TD>
        <A HREF="file.html">Form per l'upload di un file sul
   server.</A>
      </TD>
    </TR>
  </TABLE>
</BODY>

</HTML>
```

File Upload (Form)



File Upload (servlet)



Thread safety

- Tipicamente ad ogni dato istante soltanto una copia di ogni servlet è presente in memoria centrale.
- Tuttavia ogni servlet può trovarsi a dover soddisfare più richieste concorrenti.
- Siccome ogni richiesta genera un thread distinto, le servlet dovrebbero essere thread-safe.
- Se i metodi di una servlet invocati da Tomcat per soddisfare una richiesta non accedono a variabili visibili al di fuori dei metodi stessi, la servlet è automaticamente thread-safe.

Thread safety

- Tuttavia una servlet che mantiene delle risorse persistenti, deve assicurare che queste ultime siano aggiornate in modo consistente.
- Un tipico esempio è costituito dall'utilizzo di una servlet per gestire un conto bancario (un membro di tipo `int` della classe mantiene l'ammontare corrente del conto).

Un tipico scenario

- Un primo utente accede alla servlet per prelevare 100 Euro.
- La servlet controlla l'ammontare del conto, verificando la presenza di 120 Euro e consentendo l'operazione.
- Un secondo utente accede alla servlet per prelevare 50 Euro.
- La servlet controlla l'ammontare del conto, verificando la presenza di 120 Euro e consentendo l'operazione.
- La servlet addebita 100 Euro per la richiesta del primo utente e immediatamente dopo 50 Euro per la richiesta del secondo utente.
- Risultato: il bilancio complessivo del conto è in rosso di 30 Euro.
- Il programmatore viene licenziato in tronco.

Come evitare il licenziamento

- E' necessario fare in modo che la servlet soddisfi una richiesta per volta, utilizzando una delle due soluzioni seguenti:
  - inglobando il codice critico in blocchi **`synchronized`**;
  - implementando l'interfaccia **`SingleThreadModel`** (viene creato un insieme di istanze di servlet, invece di una sola; ogni copia può servire una singola richiesta per volta).
- La seconda soluzione è più onerosa dal punto di vista della gestione delle risorse del server.

Il codice Java (I)

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import java.io.*;
public class Bilancio extends HttpServlet {
  class ContoCorrente {
    public int bilancio;
  }
  ContoCorrente conto;
  public void init(ServletConfig conf) throws ServletException {
    super.init(conf);
    conto = new ContoCorrente();
    try {
      String path=getServletContext().getRealPath("")+File.separator+
                  getInitParameter("nomeFile");
      DataInputStream f=new DataInputStream(new FileInputStream(path));
      conto.bilancio=f.readInt();
      f.close();
    } catch (IOException e) { conto.bilancio =
      Integer.parseInt(getInitParameter("valoreConto")); }
    }
```

Oggetto su cui useremo il blocco synchronized.

Il codice Java (II)

```
public void doGet(HttpServletRequest req, HttpServletResponse res)
   throws ServletException, IOException {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.println("<HTML><BODY>");
    out.println("<H2>Java Bank</H2>");
    out.println("Bilancio corrente: <B>" + conto.bilancio +
                "</B><BR>");
    out.println("<FORM METHOD=\"post\" ACTION=\"Bilancio\">");
    out.println("Ammontare: <INPUT TYPE=\"text\" NAME=\"Ammontare\""+
                " SIZE=\"3\"><BR>");
    out.println("<INPUT TYPE=\"submit\" NAME=\"Deposito\""+
                " VALUE=\"Deposito\">");
    out.println("<INPUT TYPE=\"submit\" NAME=\"Prelievo\""+
                " VALUE=\"Prelievo\">");
    out.println("</FORM>");
    out.println("</BODY></HTML>");
}
```

Come la servlet rimanda il post a se stessa (notare l'assenza di "servlet/")

Il codice Java (III)

```
public void doPost(HttpServletRequest req, HttpServletResponse res)
  throws ServletException, IOException {
  int ammontare=0;
  try {
     ammontare = Integer.parseInt(req.getParameter("Ammontare"));
  } catch (NullPointerException e) {
     // Ammontare dell'operazione non specificato
  } catch (NumberFormatException e) {
    // Ammontare dell'operazione non valido
  }
  synchronized(conto) { // inizio del blocco sincronizzato
     if(req.getParameter("Prelievo") != null &&
        (ammontare <= conto.bilancio))
        conto.bilancio = conto.bilancio - ammontare;
     if(req.getParameter("Deposito") != null && (ammontare > 0))
        conto.bilancio = conto.bilancio + ammontare;
  } // fine del blocco sincronizzato
  doGet(req, res); // ripresenta il form
}
```

Il codice Java (IV)

```
  public void destroy() {
    try {
      String path=getServletContext().getRealPath("")+File.separator+
                  getInitParameter("nomeFile");
      DataOutputStream f=new DataOutputStream
                                (new FileOutputStream(path));
      f.writeInt(conto.bilancio);
      f.close();
    } catch (IOException e) { }
  }
}
```

Il metodo **`destroy()`** permette di salvare il bilancio del conto corrente nel file specificato in **`web.xml`** (**`bilancio.txt`**).

Modifiche al file **web.xml**

```
<servlet>
    <servlet-name>Bilancio</servlet-name>
    <servlet-class>Bilancio</servlet-class>
    <init-param>
        <param-name>nomeFile</param-name>
        <param-value>bilancio.txt</param-value>
    </init-param>
    <init-param>
        <param-name>valoreConto</param-name>
        <param-value>0</param-value>
    </init-param>
</servlet>

<servlet-mapping>
    <servlet-name>Bilancio</servlet-name>
    <url-pattern>/servlet/Bilancio</url-pattern>
</servlet-mapping>
```

Modifiche al file `index.html`

```
<HTML>
<HEAD>
<TITLE>Servlet di prova</TITLE>
</HEAD>

<BODY>
  <TABLE>
    . . .
    <TR>
      <TD>
        <A HREF="servlet/Bilancio">Bilancio di un conto corrente</A>
      </TD>
    </TR>
  </TABLE>
</BODY>

</HTML>
```

Esercizi

- Modificare la servlet “Bilancio” in modo che visualizzi un messaggio di avvertimento nel caso in cui l’utente tenti di prelevare una somma maggiore di quella depositata.
- Aggiungere la funzionalità di autenticazione alla servlet “Bilancio” in modo che l’utente debba inserire un nome utente ed una password prima di effettuare il prelievo.