Universita' degli Studi di Bologna
Facolta' di Ingegneria

Anno Accademico 2008-2009

Laboratorio di Tecnologie Web

Servlet

http://www-lia.deis.unibo.it/Courses/TecnologieWeb0809

Introduzione

- Componenti software modulari
  - operano sul lato server
    - all'interno dell'ambiente di esecuzione offerto dal servlet container
      - accesso a richiesta e risposta HTTP, sessione, contesto, ecc…
    - venendo richiamate quando l'utente richiede la URI ad esse associata
    - non hanno un *main()!!!*
  - sono usate per
    - processare le richieste degli utenti
    - generare dinamicamente il contenuto HTML delle risposte
    - eseguire ulteriore logica di business server-side (es: aggiornamento DB)

Esempi di Servlet

- Tomcat fornisce out-of-the-box alcuni esempi di come utilizzare le Servlet API
  - utili come cheat sheet di riferimento
    - parametri della request, accesso ai cookie, alla sessione, ecc…
  - visionabili (funzionamento e excerpt del codice sorgente) a partire da:

    http://localhost:8080/servlets-examples

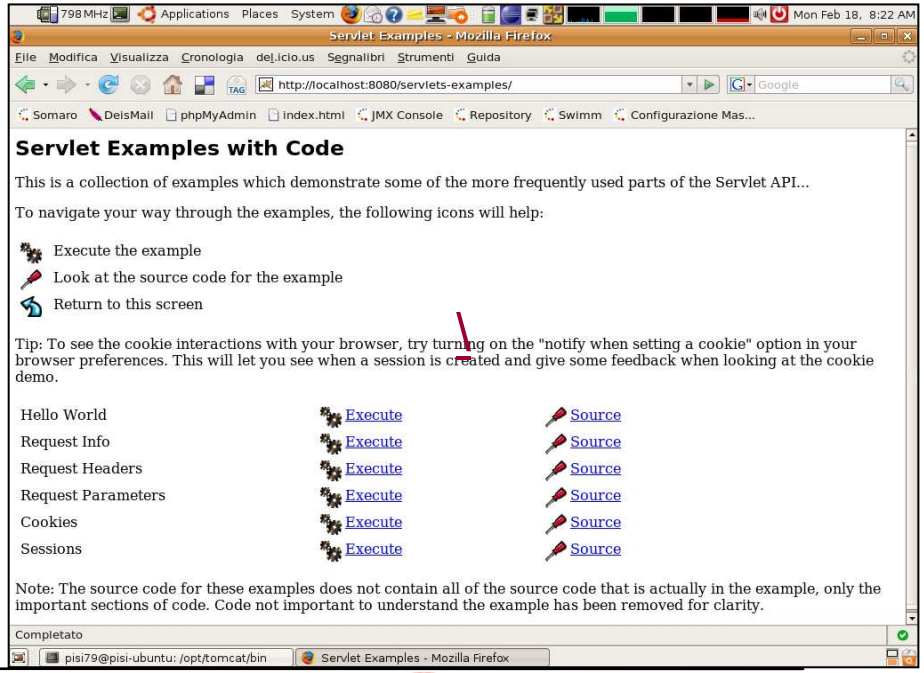
    codice sorgente completo

    disponibile su *file system*,

    nella corrispondente directory

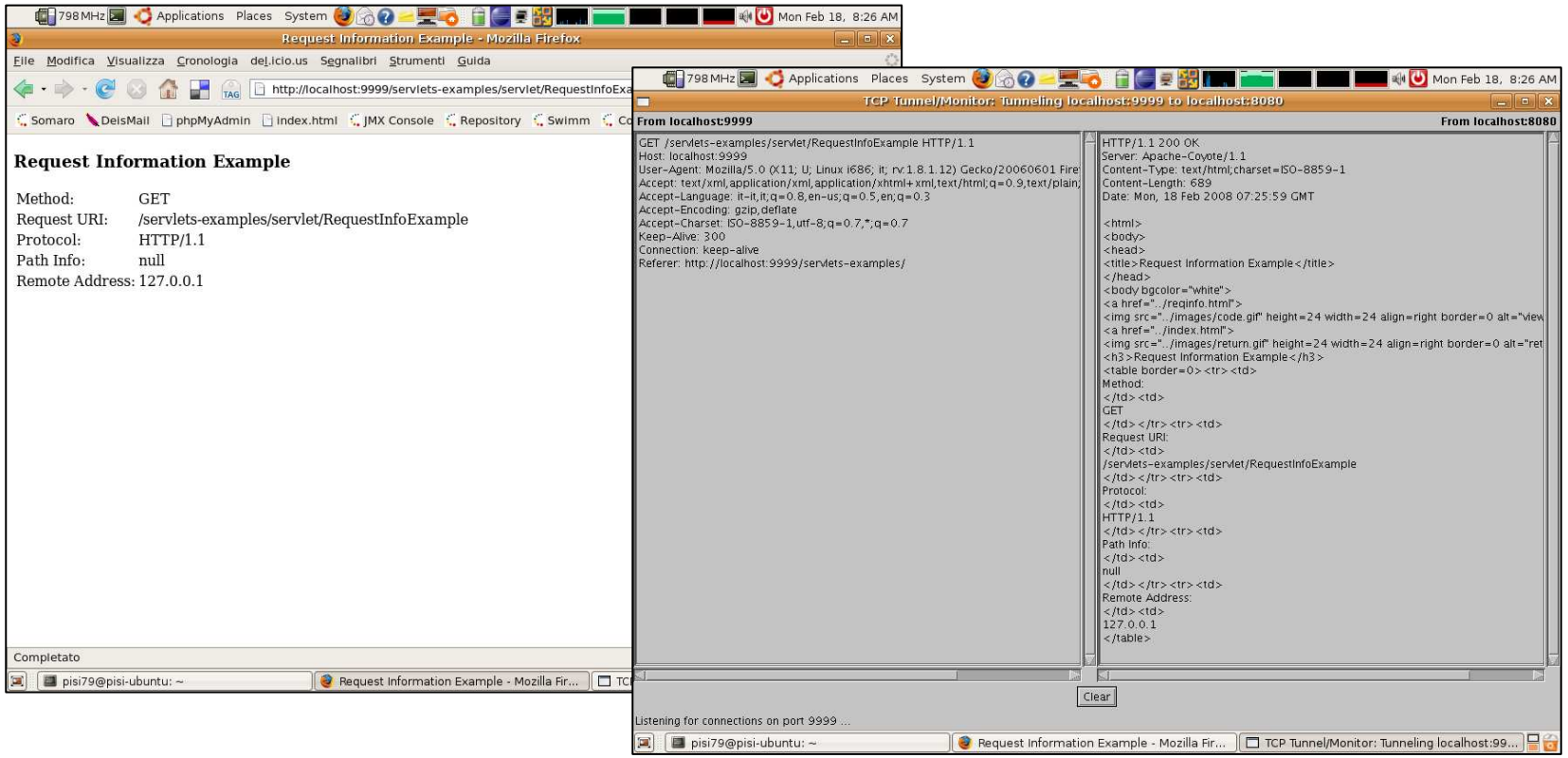
    di deployment

    `$TOMCAT_HOME/webapps/servlet-examples`



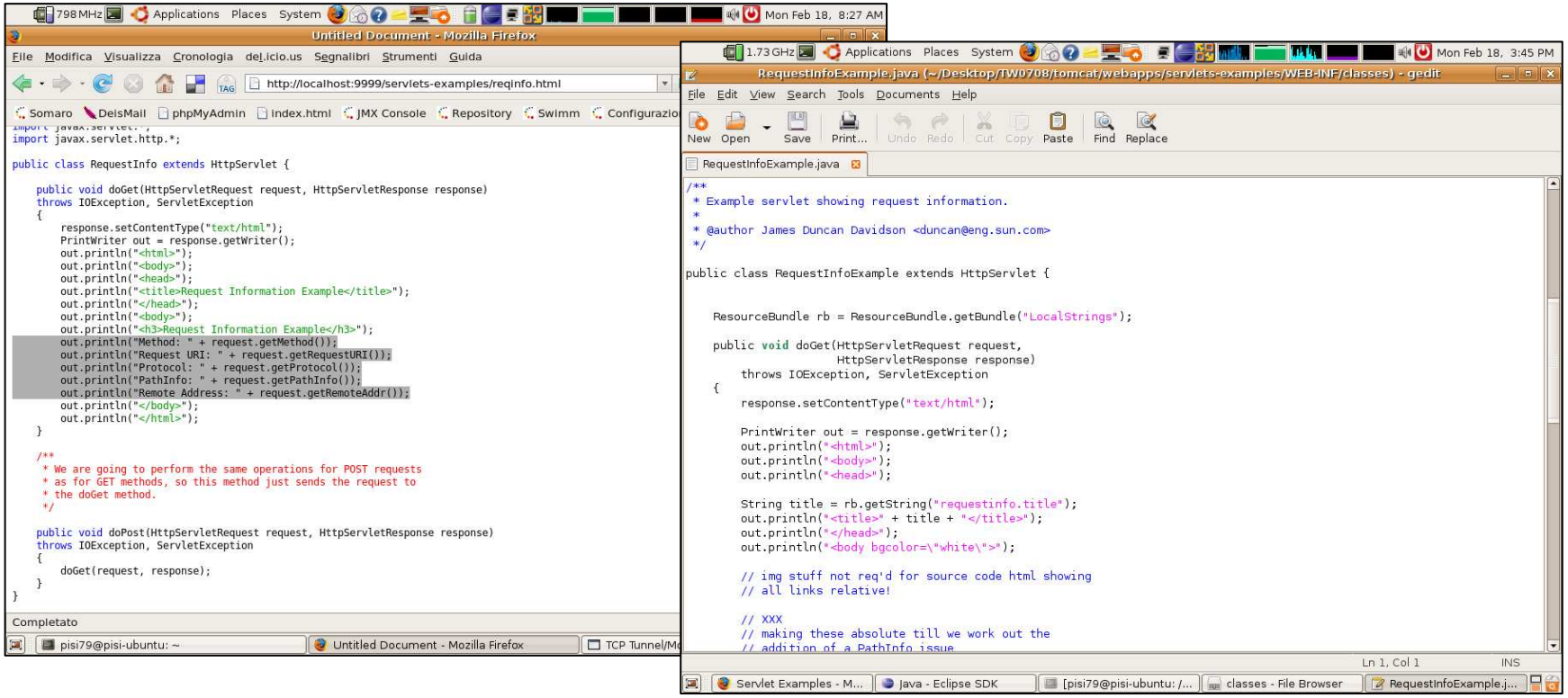
…per esempio: informazioni contenute nella richiesta

- La servlet 'Request Info' accede alle informazioni veicolate dalla richiesta
- Le stesse che potete ritrovare nel TunnelTCP (lanciando il tunnel via ANT e indirizzando la richiesta sull'apposita porta, anziché su quella 8080 di default)



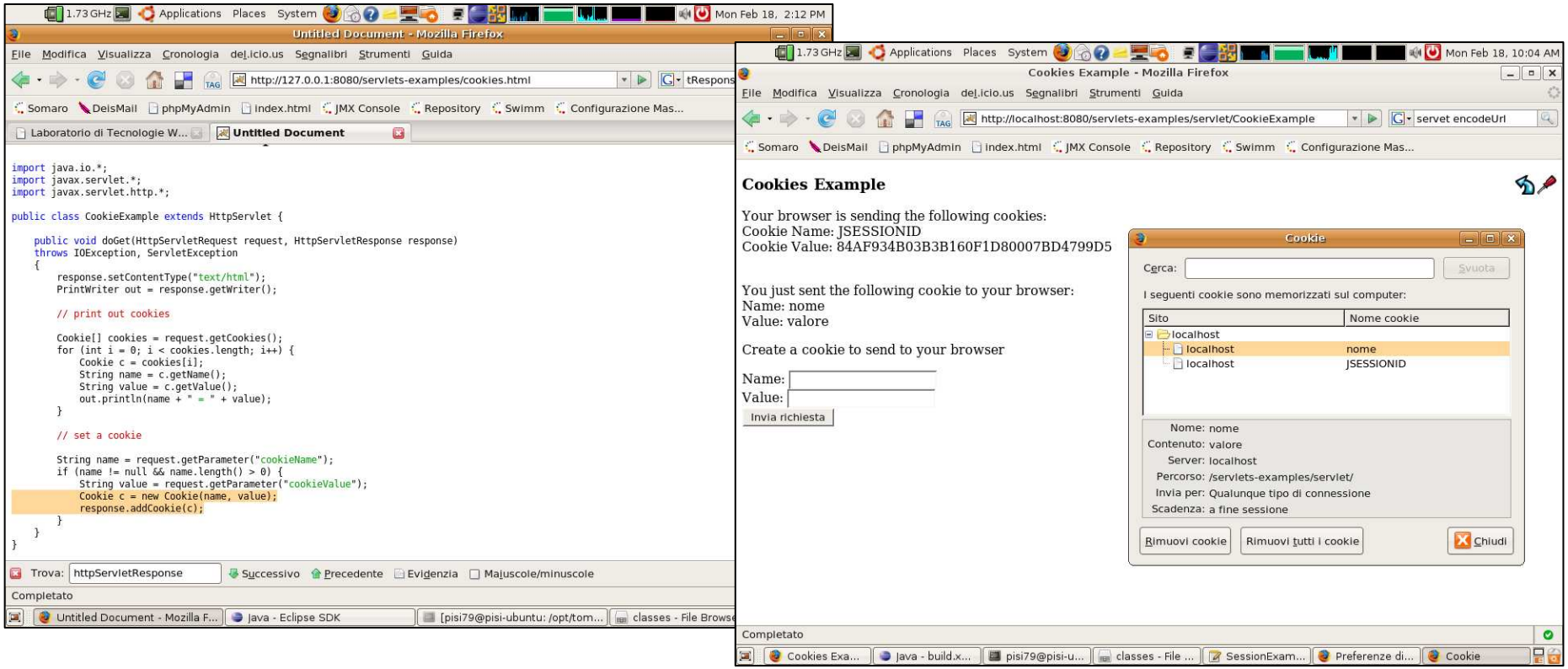
...per esempio: informazioni contenute nella richiesta

- Cliccando sul cacciavite vengono mostrate le parti di codice più significative
- In alternativa, il codice completo è visibile nel file *RequestInfoExample.java* nel direttorio `$TOMCAT_HOME/webapps/servlets-examples/WEB-INF/classes`



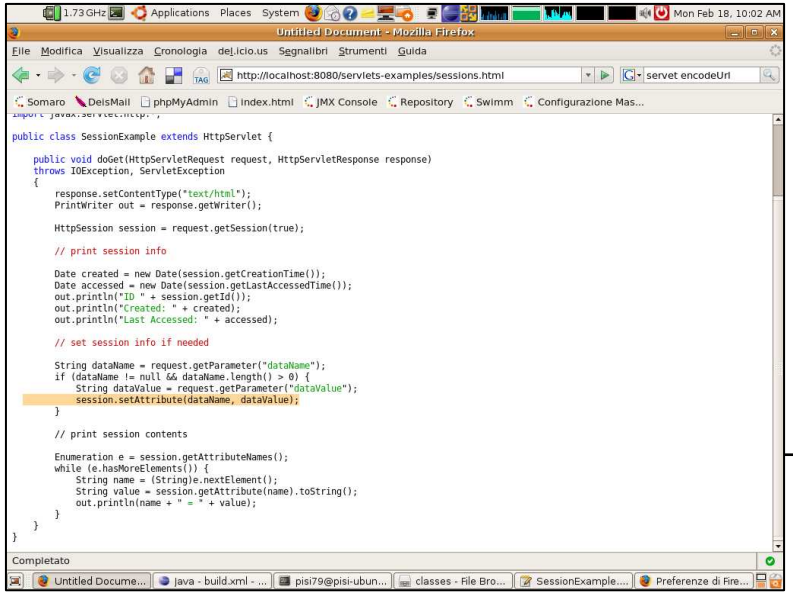
...per esempio: i cookie

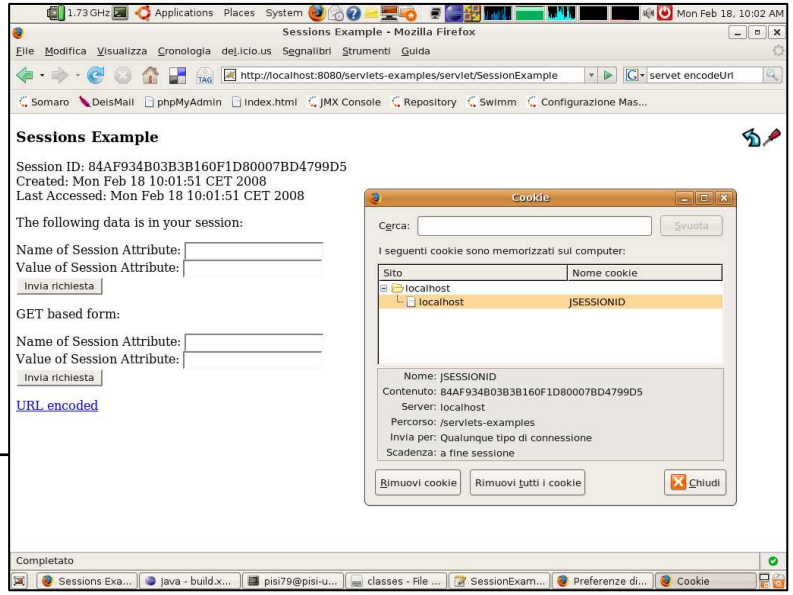
- I cookie sono frammenti di informazioni (coppie nome-valore)
  - che il codice lato-server, agendo sulla risposta, ordina al browser di salvare
  - che il browser salva, associandoli al dominio da cui li ha ricevuti
- La servlet 'Cookie Example' mostra come gestire i cookie
- I browser moderni permettono di visualizzare (e cancellare!) i cookie memorizzati



…per esempio: informazioni in sessione

- Attraverso l'oggetto "sessione", il Servlet Container permette alle Servlet (e non solo a loro!) di salvare informazioni lato server
- L'associazione tra un browser (lato-client) e le sue informazioni di sessione (lato-server) è ottenuta attraverso:
  - **cookie di sessione** (*JSESSIONID*) che il server ordina al browser di salvare e che il browser allega poi a ogni nuova richiesta
  - **tecniche di URL-rewriting**: riscrittura di tutti i link presenti nelle pagine restituite dal server all'utente, aggiungendo il coda alle URI l'id della sessione a lui associata, in modo che le successive richieste lo inviino come come parametro di GET (*http://.../myrequest?mypar=myval&session_id=2323*)





Documentazione

- Numerosissimi tutorial ed esempi online
  - *http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Servlets.html*
  - *http://www.novocode.com/doc/servlet-essentials/*
  - …
- La sezione **Risorse** del sito del corso ospita repliche sia dei JavaDoc che del tutorial J2EE (consultabili anche dal lab4)
- *http://www.lia.deis.unibo.it/Courses/TecnologieWeb0708/materiale/laboratorio/guide/j5eeapi/*
- *http://www.lia.deis.unibo.it/Courses/TecnologieWeb0708/materiale/laboratorio/guide/j2ee14tutorial7/index.html*

- E poi, ovviamente, per singoli problemi…
  - Google e gli altri motori di ricerca
  - i forum, ecc…

Sviluppo di applicazioni web con servlet

- Scrittura del **codice** della servlet (e testing, possibilmente!)
- Modifica del **descrittore** *web.xml* per dichiarare la servlet
- **Compilazione** del codice sorgente della servlet in formato bytecode
- **Packaging** di…
  - risorse per il client (es: pagine HTML, fogli di stile CSS, script javascript, …)
  - descrittore *web.xml*
  - bytecode delle servlet
  - librerie di utilità invocate dalle servlet
  - altre risorse per il server (es: pagine JSP, altri descrittori XML, …)

…all'interno di un archivio zip con estensione *.war* e struttura interna ben definita

- **Deployment** sul web server

Sviluppando con Eclipse potete utilizzare come punto di partenza il progetto ***TemplateServlet*** ed il relativo file di build di ANT (praticamente la stessa cosa del progetto ***TemplateWebApp***, ma finalmente con un po' di codice d'esempio)

Prima di proseguire, solo una precisazione

- I progetti che si scaricano dal siti sono pensati per
  - essere importati in Eclipse (contengono i metafile descrittori *.classpath* e *.project)* al fine di sfruttare tale IDE durante lo sviluppo:
    - parser
    - evidenziazione
    - autocompletamento
    - refactoring
    - …
  - gestiti attraverso ANT (contengono appositi file *.properties* e *build.xml*) per automatizzare le operazioni di:
    - packaging
    - deployment
    - undeployment
    - lancio di eventuali strumenti aggiuntivi (es: TunnelTCP)

Prima di proseguire, solo una precisazione

- Importando ed utilizzando il progetto su macchine diverse (Eclipse 3.1 in lab4 su linux, Eclipse 3.3 in lab4 su Windows, Eclipse 3.4 sui vostri pc personali, ecc…) l'unica cosa da modificare è il file ***environment.properties*** e in particolare:
  - *tomcat5.home* (o, in alternativa: *tomcat6.home*) deve puntare al path di installazione di tomcat sulla macchina che state usando

es: linux

```
/home/USER_ID/Desktop/apache-tomcat-5.5.20
```

es: windows (occhio ai caratteri di escape come '\' o ' ')

```
C:\\Documents\ and\ Settings\\USER_ID\\Desktop\\ecc...
```

es: macosX

```
/Users/USER_ID/ecc...
```

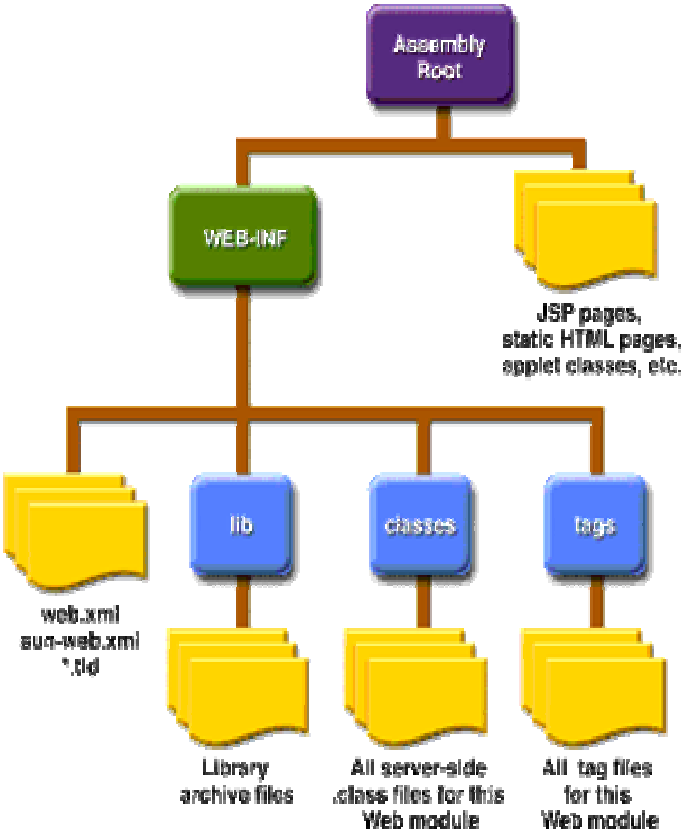
  - *appserver.home*, *appserver.common.lib.path*, *appserver.lib.path* e *deploy,path* devono puntare alla giusta Tomcat version in uso (se usate la 5 vanno già bene)
  - tutte le altre proprietà vanno già bene, perché definite in termini di altre proprietà
  - la proprietà *java.home* è ignorata dentro Eclipse perché impostata dall'IDE

Struttura del progetto di esempio

- All'interno dei progetti di esempio, il direttorio *war* rappresenta l'*assembly root* dell'applicazione web da deployare

- I sorgenti delle servlet sono invece da collocare nel direttorio *src* come al solito

- **I target di ANT si occupano di**
  - compilare il codice sorgente
  - collocare il bytecode ottenuto sotto *war/WEB-INF/classes*
  - comprimere il direttorio *war* così ottenuto per farne un archvio in formato zip con estensione *.war (web archive)*
  - copiare (o esplodere) tale archivio nella directory di deployment di Tomcat



Librerie

- Occorre capire **dove** servono **quali** librerie
- Esempi:
  - il file *servlet-api.jar* contiene le API definite dalle specifiche delle Servlet
    - è da aggiungere al build-path del progetto Eclipse per poter compilare
    - ma è sbagliato deployarlo, perché Tomcat già ne possiede una copia
  - il file *junit4.jar* contiene l'implementazione del framework di test Junit
    - è da aggiungere al build-path del progetto Eclipse per poter eseguire i test
    - ma ovviamente non interessa deployarlo su Tomcat
  - per esigenze di modularità e riuso del codice, i driver per il database (es: *mysql-connector-XXX.jar)* vengono di solito caricati in memoria mediante una istruzione del tipo `Class.forName(DRIVER)`
    - non serve aggiungere tale tipo di librerie al build path di Eclipse
    - ma è necessario lo stesso assicurarsi che siano presenti su Tomcat
  - altre librerie utilizzate di volta in volta dalle Servlet
    - devono essere aggiunte al build-path di Eclipse per poter compilare
    - devono essere deployate su Tomcat per poter eseguire

Convenzioni, accorgimenti

- I progetti di esempio utilizzano questa convenzione:
  - *lib*

    librerie necessarie per compilare su Eclipse (es: *servlet-api.jar*), per eseguire su Eclipse (es: *soap.jar, junit4.jar*, …), ma non per eseguire su Tomcat
  - *war/WEB-INF/lib*

    librerie necessarie per eseguire su Tomcat (es: driver database, librerie di terze parti con funzioni di utilità richiamate dalle servlet, ecc…)

- Occorre poi tenere presente che:
  - le librerie aggiunte al build-path di Eclipse non sono più visibili nelle loro posizioni originali nella vista *Package Explorer* (occorre abilitare la vista *Navigator* per ottenere una rappresentazione del contenuto del file system!)
  - ANT ed Eclipse utilizzano due motori di compilazione diversi:
    - il target *set.classpath* deve riflettere le modifiche al build-path di Eclipse
    - per stare dalla parte dei bottoni, comunque, il classpath di ANT include già a default tutto `lib/**/*.jar` e tutto `WEB-INF/lib/**/*.jar`

Getting familiar with

- Scaricare (dal sito del corso)…
  - …importare (come *'existing project'* in Eclipse)…
    - …configurare (modificando *environment.properties*)…
      - …deployare (lanciando il target di ANT *'deploy.as.XXX'*)…

        - …il progetto di esempio *TemplateServlet.zip*

- Funziona tutto?

Uno sguardo piu' approfondito

- Descrittore *web.xml*
  - dichiara una sola servlet e la mappa su una particolare URI *(/yourservlet)*
  - definisce pagine per gli errori di tipo 404 e *java.lang.Exception*

- Pagina *index.html*
  - Carica l'immagine di attesa *images/wait.js* invocando lo script javascript *scripts/wait.js* in corrispondenza dell'evento *onload*
  - Redirige automaticamente, dopo 4 secondi, alla servlet */yourservlet* mediante un header di tipo *http-equiv* (niente AJAX qui)

- Servlet */yourservlet*
  - il solito *"Hello world"*
  - e un easter egg (il lancio di un'eccezione), se invocata con il parametro "bad"

- Pagine di errore
  - ottenute specificando URI non valide o attivando l'easter egg di cui sopra

Debug su Tomcat

- Anche se non si tratta di applicazioni da lanciare in Eclipse, è possibile debuggare il codice delle servlet…
  - …facendo partire tomcat in modalità debug

```
export JAVA_OPTS="$JAVA_OPTS
    -Xdebug
    -Xrunjdwp:transport=dt_socket,address=8787,server=y,suspend=n"
```

    (prima di lanciare startup.sh)

  - …selezionando *Debug As… → Remote Java Application → New Configuration*

    (e lanciare il Debug da Eclipse specificando la stessa porta su cui Tomcat è stato lasciato in ascolto; in questo caso: *8787*)

- Provate con un breakpoint inserito in *YourServlet:39*

Esercitazione: la servlet di San Valentino (un po' in ritardo)

- Da realizzare sulla base del progetto *TemplateServlet.zip*
  - cambiare il nome del progetto (tasto *F2* in Eclipse)
  - cambiare il nome usato per il deploy (file *project.properties*)
- Utilizza la libreria *MyLibrary.jar* per il calcolo dell'affinità di coppia
  - già presente (sebbene non utilizzata) in *TemplateServlet.zip*
  - già pronta per il deploy su Tomcat (poiché in *war/WEB-INF/lib*)
  - da aggiungere al *build-path* di Eclipse del progetto
  - fornita dal progetto *TemplateLib.zip*
    - anch'esso scaricabile dal sito
    - contiene suite di test basate su JUnit4
    - ci si può curiosare, a casa…

Requisiti

- La classe *it.unibo.tw0708.utils.Valentine* dentro *MyLibrary.jar* presenta un metodo…

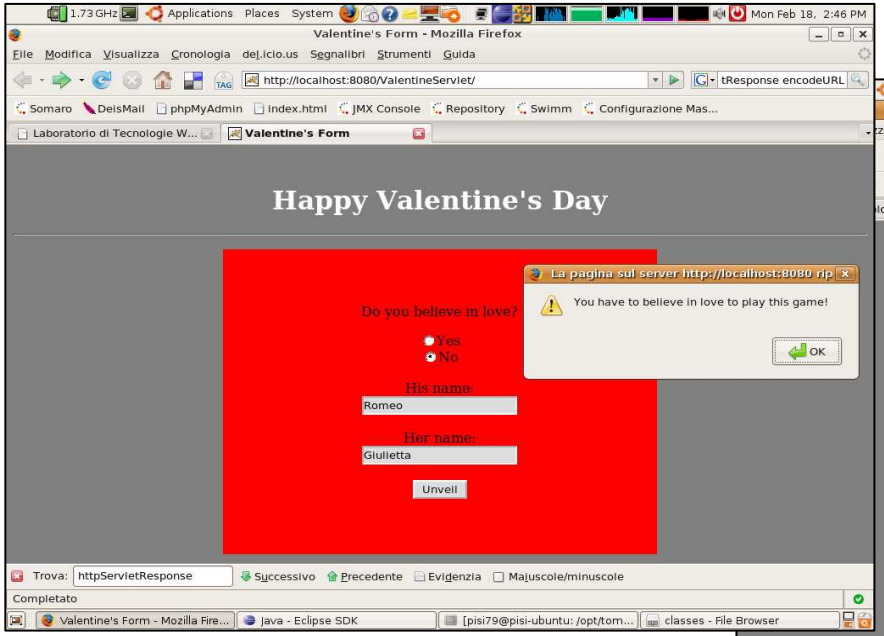
```
public static int heartMatch(String he, String she) throws Exception;
```

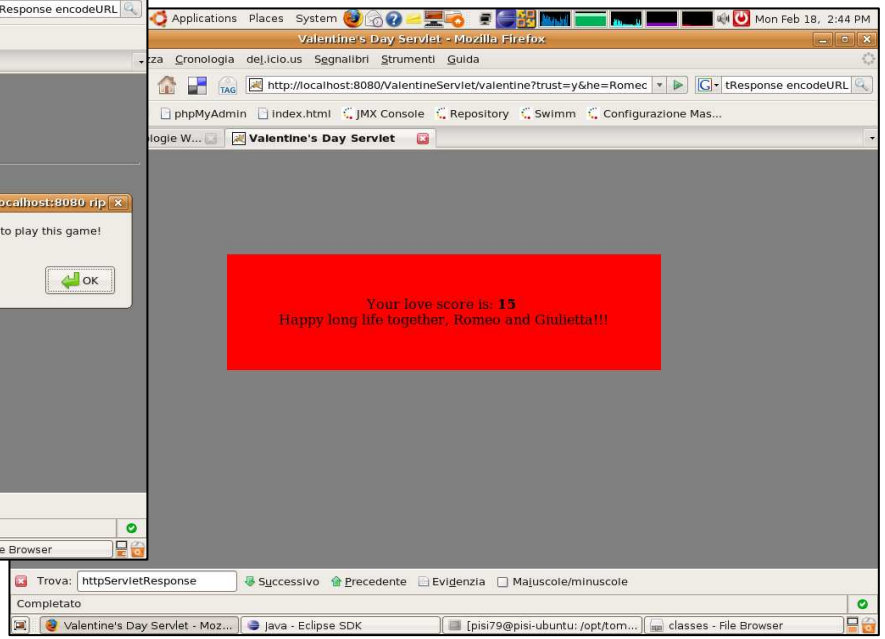
…che dati due nomi di persone restituisce un punteggio di affinità di coppia (romanticamente, tramite la differenza degli *hashCode()* delle stringe)

- Progettate una pagina HTML…
  - …con una form in grado di accettare i nomi di due persone
  - …e la cui action punti alla URI di una servlet che sfrutta tale libreria per produrre una pagina HTML di risposta.
- Validate via Javascript i dati inseriti (es: nomi non nulli, accettazione preliminare delle conseguenze del test, ecc…) prima di scatenare la richiesta
- E, a tempo perso, applicate dei CSS molto romantici

Soluzione

- On line tra pochissimo…
  - …progetto *ValentineServlet.zip*





  - …mostra anche come utilizzare cookie + filtro + dispatcher (per evitare che il gioco generi dipendenza nell'utente dopo aver fatto piu' di 3 test)