Alma Mater Studiorum - Università di Bologna
Scuola di Ingegneria

Tecnologie Web T
A.A. 2022 – 2023

Esercitazione 0
Strumenti per le esercitazioni

Agenda

- Eclipse
  - caratteristiche generali
  - importazione/creazione di un progetto di esempio
  - funzionalità di ausilio alla scrittura del codice
  - compilazione, collaudo ed esecuzione di un'applicazione
  - gestione tramite ANT
- Applicazioni Web
  - avvio e gestione del server Tomcat
  - deploy da filesystem e da Eclipse
  - debug remoto e locale
- Editor HTML, CSS, Javascript
  - FIrefox for Developer, Google Chrome Developer Tools, Visual Studio Code

Eclipse, caratteristiche generali

- Ambiente integrato di sviluppo (IDE)
  - interamente scritto in Java
  - multipiattaforma (Win/Mac/Linux/…)
  - multi-linguaggio (tool anche per programmazione linguaggio C ad es.)
  - open source (controllato dalla Eclipse Foundation)

- Architettura basata su tecnologie core e plug-in
  - fortemente modulare ed espandibile
  - adattabile (e adattato) alle più diverse esigenze attraverso l'installazione di cosiddetti “plug-in”

Configurare il proprio ambiente di lavoro

- Strumenti già presenti in laboratorio, ma a casa...
- Java JDK e JRE (versione 11)
  - https://www.oracle.com/java/technologies/javase-jdk11-downloads.html
- Eclipse IDE (2020-03 for Java Enterprise Edition Developers)
  - https://www.eclipse.org/downloads/packages/
  - https://www.eclipse.org/downloads/packages/release/2020-03/r/eclipse-ide-enterprise-java-developers-includes-incubating-components
  - http://eclipsetutorial.sourceforge.net/
  - http://eclipsetutorial.sourceforge.net/totalbeginner.html
- Troubleshooting
  - http://www.google.com/

Primo impatto

- Avviare Eclipse per la prima volta
  - scelta del direttorio per il **Workspace** (dove verranno salvati i progetti)
  - Welcome... eccetera: → *close*
  - dovesse mai servire di nuovo: *Help → Welcome*
- **Workbench** (area di lavoro) costituita da un insieme di **View** (viste)
  - *Package View* (struttura logica dei progetti)
  - *Navigator View* (struttura dei file su disco)
  - *Java Editor* (scrittura del codice)
  - *Outline View* (struttura del file aperto nell'editor)
  - *Console* (stdout e stderr prodotti dalle attività eseguite)
  - *Problems* (*primo luogo dove guardare quando qualcosa va storto!!!*)

    …e tante altre: *Window → Show view*
- **Perspective** (prospettiva) come associazione di un preciso insieme di viste, in precise posizioni, per affrontare determinate operazioni (codifica, debug, Web, condivisione su SVN...)
  - *Windows → Open perspective*

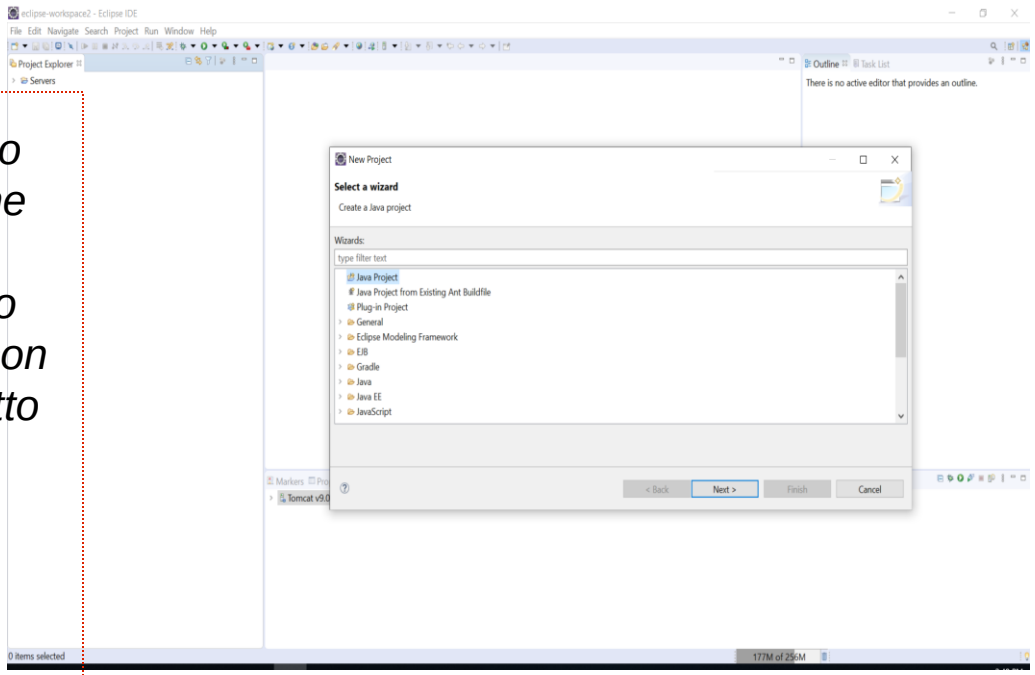
Perché un IDE

- Numerose funzionalità “di comodo” per velocizzare la scrittura del codice e garantire la sua correttezza a tempo di compilazione
  - ***supporto per il refactoring*** (nomi di package, classe, metodi, variabili, ...)
  - ***generazione automatica di codice*** (costruttori, metodi getter/setter, …)
  - ***evidenziazione*** (parole chiave del linguaggio, errori, …)
  - ***messaggi di errore e consigli per risoluzione*** (a volte automatica)
  - ***autocompletamento*** (parentesi, nomi delle variabili, modificatori di tipo, …): si attiva da solo dopo un istante, o su comando: *Ctrl+Space*
  - ...
- Tantissime funzionalità
  - *right-click* dovunque :)
  - menu *Help → Search*
  - sito di Eclipse, tutorial on-line (spesso persino animati)

Gestione dei progetti

- Creazione
  - *File → New → Java Project / Project…*

- Importazione da file zip (esempi del corso)
  - *File → Import → General → **Existing Projects into Workspace** → Next → **Select archive file***



nota bene: nel workspace non possono esistere più progetti con lo stesso nome

Occorre cancellare o rinominare quello già esistente, prima di importarne uno con lo stesso nome: diversamente, il progetto "omonimo" contenuto nel file ZIP non viene visualizzato tra i progetti individuati nell'archivio
ERRORE COMUNE nelle prime esperienze di lab

Importazione di un progetto

- All'interno dell'archivio ZIP dell'esercitazione, nel direttorio ***progetti***
  - il file ***00_TecWeb.zip*** contiene un semplice progetto Java di esempio
  - creato con Eclipse: contiene già tutti i descrittori necessari a essere riconosciuto e configurato correttamente

- Importare il progetto come appena visto, senza esploderne l'archivio su file system (lo farà Eclipse)

- Problemi? (librerie, versioni JRE, versioni del compilatore, ...)
  - vista ***Problems View****:* diagnosi
  - **...a breve li risolveremo**

Struttura del progetto

All'interno della directory radice

- ***src:*** sorgente (file *.java*) dell'applicazione da sviluppare
- ***test:*** sorgente delle routine di test (opzionali) che verificano il corretto funzionamento dell'applicazione
- ***LIBRERIE*** (visualizzazione può variare da versione a versione di Eclipse): codice fornito da terze **parti necessario allo sviluppo**
  - **JRE** le classi base del runtime di Java (es: `java.lang.String`)
  - **API** e loro eventuale **implementazione** riferita dall'applicazione (es: *JUnit* per i test, oggi; *specifiche J2EE* per lo sviluppo di Servlet, in future esercitazioni)
- ***ant:*** strumenti per l'esecuzione automatica di operazioni
  - compilazione, esecuzione dei test, packaging, distribuzione, ...
- ***lib:*** direttorio che fisicamente contiene gli archivi *.jar* delle librerie in uso nel progetto *(nota: alcune versioni di Eclipse "nascondono" le librerie aggiunte al build-path, onde evitare di visualizzare informazioni "doppie")*
- ***resources:*** altre risorse da allegare alla versione distribuibile del progetto (immagini, file multimediali, …)
- ***tmp:*** direttorio per scopi temporanei

Compilazione, collaudo, esecuzione

- Poche semplici classi
  - logica di business
  - routine per il collaudo automatizzato
  - avvio dell'applicazione

- Completamento del progetto:
  - **correzione errori (autocompletamento), importazioni mancanti (organize imports), creazione dei metodi richiesti (autogenerazione sorgente, quickfix), ecc...**

- Esecuzione dei test
  - **analisi della struttura di una suite di test**
    - Ulteriori informazioni su *http://junit.sourceforge.net/#Documentation)*
  - **apertura della classe che realizza la suite *Junit:*** *Esegui come... → JUnit Test*
  - **aggiunta di ulteriori test (es: corretto funzionamento metodi getter/setter)**

- Avvio dell'applicazione
  - **apertura della classe che contiene il metodo** `main()`
  - **scrittura del metodo:** *Esegui come... → Java application*

ANT

- Lo sviluppo di un'applicazione richiede di eseguire tipiche sequenze di operazioni
  - scrittura del codice sorgente, compilazione, collaudo, packaging, distribuzione, ...
- Alcune di queste operazioni sono ripetitive e la loro esecuzione può richiedere azioni diverse in ambienti di sviluppo diversi
  - posizioni e convenzioni dei file su disco e convenzioni di nome
  - posizione e nome di menu e pulsanti nei diversi ambienti di sviluppo (e spesso anche in diverse versioni dello stesso ambiente)
  - ...
- Strumenti di sviluppo come ANT (o Maven, o altri, …), detti *build tool*, permettono invece di
  - definire una volta per tutte le operazioni da compiere
  - eseguire tali operazioni in maniera automatica
  - fare tutto questo in maniera indipendente dall'IDE utilizzato

build.xml

- ANT è a sua volta realizzato in Java e configurato mediante file XML

- Permette di definire in maniera leggibile e facilmente modificabile un insieme di obiettivi (***target***) il cui raggiungimento permette di completare le diverse fasi di sviluppo del progetto
  - inizializzazione, compilazione, collaudo, packaging, ...
  - relazioni di dipendenza
  - definizione di proprietà *(**property**)* mediante variabili di tipo write-once che è possibile riferire all'interno dei diversi obiettivi

- Non esistono obiettivi predefiniti, ma ciascuno è definito attraverso l'indicazione di una o più operazioni (***task***)
  - copia di file, compilazione, creazione di archivi, ...

- ANT rende disponibili una serie di operazioni predefinite (***core task***) e prevede una serie di operazioni opzionali (***optional task***) dipendenti da librerie di terze parti
  - è inoltre possibile definire nuovi “task”, attraverso apposite classi Java

Uso di ANT ai fini delle esercitazioni

- L'insegnamento di ANT ***non*** è tra gli obiettivi del corso, ma il suo uso "ai morsetti" permette di:
  - rendere ciascuno studente in grado di eseguire le operazioni "di contorno" richieste dall'esercitazione, in modo efficiente e indipendente dagli specifici OS e IDE e dai diversi percorsi locali su file system
  - mantenere traccia delle operazioni svolte e di come esse sono realizzate, attraverso il contenuto del ***file build.xml***

- Istruzioni per l'uso:
  - ***ant/build.xml***: definizione degli obiettivi da completare (nonostante sia possibile modificare ed estendere tale file a piacimento, esso è **concepito per poter essere usato senza alcuna modifica**)
  - ***ant/environment.properties***: proprietà richiamate da *build.xml* che differiscono da macchina a macchina e sono quindi tipicamente **DA MODIFICARE**

- Accorgimenti:
  - è possibile lanciare *ant* da riga di comando
    - **cd $PROJECT_HOME/ant**
    - **ant <nome_obiettivo>**
  - è possibile lanciare *ant* dall'interno di Eclipse (in questo caso, se ne eredita la JAVA_HOME):
    - *Windows → Show view → Other.. → Ant → Ant*
    - Trascinare il file *build.xml* nella nuova vista
    - Eseguire un obiettivo tramite *double-click*

Funzionalità avanzate - Debug

- Attraverso l'IDE, è possibile seguire passo-passo il flusso di un programma:
  - specificare opportuni ***breakpoint*** nei quali interrompere e monitorare l'esecuzione
    - ad esempio, nella classe `HelloWorld.java`...
    - *...left-click* oppure *right-click → toggle breakpoint* sulla fascia grigia a sinistra del codice, nell'editor principale
  - eseguire il programma in modalità debug dall'interno dell'IDE stesso
    - *right click → Debug come... → Java application* sulla classe contenente il metodo `main()`
- In caso di successo, la prospettiva corrente dell'IDE si modifica per esporre le tipiche funzionalità da debug
- Possibili operazioni:
  - giocare con i comandi *Play/Pause, Step into, Step over, Step return*
  - controllare il valore run-time delle variabili nella vista *Variables*
  - *eccetera... eccetera...*
    - *cambiare il valore di una variabile,*
    - *ispezionare il risultato di un'espressione,*
    - *...*

Debug di applicazioni “remote”

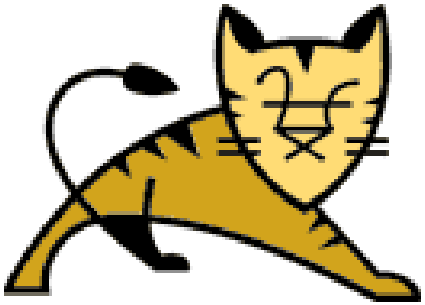
- Tuttavia...
  - le applicazioni “tradizionali” ***non*** eseguono all'interno dell'IDE...
  - in particolare, le applicazioni “Web” eseguono all'interno di opportuni server “contenitori”

- Per poter fare debugging di tali applicazioni è quindi necessario imparare ad eseguire l'applicazione in esame e lo strumento di debug come processi separati, che comunicano attraverso la rete
  - specificare opportuni breakpoint nell'applicazione d'esempio
  - modificare il suo metodo `main()` affinché non termini subito!
    - es: ciclo for + attese
  - lanciare il programma come applicazione Java indipendente con l'opzione...
    - **-Xdebug -Xrunjdwp:transport=dt_socket,address=$PORT,server=y,suspend=n**
    - nel file *build.xml* è già presente uno specifico obiettivo
  - creare (e poi avviare) una apposita “Debug configuration” nell'IDE
    - *Run → Debug configurations... → Remote Java application → right-click → New...*
    - ...e indicare la stessa porta di ascolto $PORT

Apache Tomcat (1)

- Un semplice Web Server, sviluppato e distribuito dalla fondazione Apache, interamente scritto in Java
  - permette di pubblicare siti Web
  - fornisce l'ambiente di esecuzioni per applicazioni Web scritte in accordo alle specifiche Java Servlet e JSP
  - In questo corso utilizzeremo la versione 9.0.22

- Installazione del server
  - download dal sito ufficiale *https://archive.apache.org/dist/tomcat/tomcat-9/v9.0.22/bin/apache-tomcat-9.0.22.zip* oppure
  - dal path *C:\\Applicativi\TecnologieWeb* *del file system dei PC client LAB* oppure
  - dalla intranet universitaria (sito del corso)
  - estrazione del contenuto del file ZIP



Apache Tomcat (2)

- Tomcat è uno dei Web Server più utilizzati sia sul mercato che nel settore dell'education
- Tomcat è un Web Server STAND-ALONE
  - Non ha bisogno di un IDE (e.g., Eclipse) per funzionare
    - anche se, come vedremo a breve, il suo uso integrato con Eclipse può semplificarci la vita ai fini del corso…
  - Il deploy di web application, debug remoto, ed operazioni di tuning possono essere fatte indipendentemente dal tipo di IDE che si utilizza, evitando così il vincolo molto stringente di un IDE specifico
- Per dare una panoramica completa, nel seguito verrà illustrato il suo utilizzo in entrambe le modalità:
  - standard STAND-ALONE e
  - integrata con l'IDE Eclipse

Avvio di Apache Tomcat (STAND-ALONE)

- È necessario impostare la variabile d'ambiente JRE_HOME o JAVA_HOME affinché “indichi” un'installazione JDK con versione >= 1.6

  export JRE_HOME=.... (linux; per verificare: **echo $PROVA**)

  SET JRE_HOME=... (windows; per verificare: **echo %PROVA%**)

- Lanciare il server attraverso il comando

  TOMCAT_HOME/bin/startup.sh (linux)

  TOMCAT_HOME/bin/startup.bat (windows)

- Controllare il corretto avvio
  - nei log del server stesso
    - **tail -f TOMCAT_HOME/logs/catalina.out** (linux)
    - popup *e/o* **notepad.exe TOMCAT_HOME/logs/catalina.out** (windows)
  - accedendo a http://localhost:8080/

Eclipse & Tomcat (1)

- Eclipse dispone di un'apposita *perspective* per facilitare la creazione di applicazioni Web
  - *Windows → Open Perspective → Other... → Web*
- Creazione di un semplice progetto Web
  - *File → New → Other... → Web → Dynamic Web Project*
  - vedremo a lezione la struttura di un'applicazione Web; per ora limitiamoci a realizzare un semplice test con servlet e JSP
- Creazione Servlet
  - tasto destro del mouse sul nome del progetto e poi *New → Other... → Web → Servlet*
  - inserire nel metodo doGet della classe creata la seguente istruzione: **response.getOutputStream().println("ciao da TestServlet.doGet");**
- Creazione JSP
  - tasto destro del mouse sul nome del progetto e poi *New → Other... → Web → JSP File*
  - inserire nel body del file creato la seguente riga: **ciao da TestJsp.jsp <%= new java.util.Date() %>**

Eclipse & Tomcat (2)

- Problemi di compilazione: mancano le librerie per le servlet
  1) prelevare servlet-api.jar da Tomcat (directory **lib**), copiare nel progetto (directory **lib**) ed inserire tale jar nel path del progetto
  2) gestione di un server direttamente da Eclipse (vedremo poi)

- Creare il file WAR che contiene tutte le informazioni necessarie per effettuare il deploy
  - *Export → WAR file*
  - spostare il file creato nella directory **webapps** di Tomcat
  - attendere pochi secondi e controllare la directory **webapps**
  - testare tramite
    - **http://127.0.0.1:8080/nome-progetto/nome-servlet**
    - **http://127.0.0.1:8080/nome-progetto/nome-jsp**

- Metodo di deploy tramite interfaccia grafica
  - http://127.0.0.1:8080/ → Manager App
  - ma bisogna aggiungere un utente di tipo "manager-gui" nel file di configurazione di Tomcat **conf/tomcat_users.xml**, e.g., **<user username="studente" password="tomcat" roles="manager-gui"/>**

Esecuzione in modalità “debug remoto”

- Per poter eseguire *Tomcat* in modalità “debug” è necessario modificarne lo script di avvio
  - in Windows modificare **bin/startup.bat** ed aggiungere
    - set JPDA_ADDRESS=8787
    - set JPDA_TRANSPORT=dt_socket
  - in Linux modificare **bin/startup.sh** ed aggiungere
    - export JPDA_ADDRESS=8787
    - export JPDA_TRANSPORT=dt_socket
  - inoltre cercare la riga
    - call "%EXECUTABLE%" start %CMD_LINE_ARGS% (*startup.bat)*
    - exec "$PRGDIR"/"$EXECUTABLE" start "$@" (*startup.sh*)
  - e modificarla come segue
    - call "%EXECUTABLE%" jpda start %CMD_LINE_ARGS% (*startup.bat)*
    - exec "$PRGDIR"/"$EXECUTABLE" jpda start "$@" (*startup.sh*)

- All'interno dell'IDE occorre poi...
  - creare ed avviare una apposita configurazione di debug remoto
    - *Run → Debug configurations... → Remote Java application → right-click → New...*
    - ...e indicare la stessa porta di ascolto $PORT
  - specificare opportuni breakpoint nei quali interrompere e monitorare l'esecuzione
    - *...left-click oppure right-click → toggle breakpoint sulla fascia grigia a sinistra del codice, nell'IDE*
  - eseguire il deploy dell'applicazione
  - richiedere via browser la risorsa che determina l'esecuzione del codice contenente i breakpoint
  - osservare l'esecuzione passo-passo (pause/play, step in/out/over)

Esecuzione in modalità “debug locale”

- In Eclipse seleziona la view "Servers"
  - elenco dei server su cui è possibile effettuare il deploy di applicazioni Web direttamente da Eclipse
  - tramite Eclipse è possibile: avviare/fermare Tomcat, deploy/undeploy di applicazioni, debug locale
  - inoltre **re-deploy automatico** ad ogni compilazione di servlet e/o JSP
  - per creare un nuovo server: *File → New → Other... → Server → Apache …*

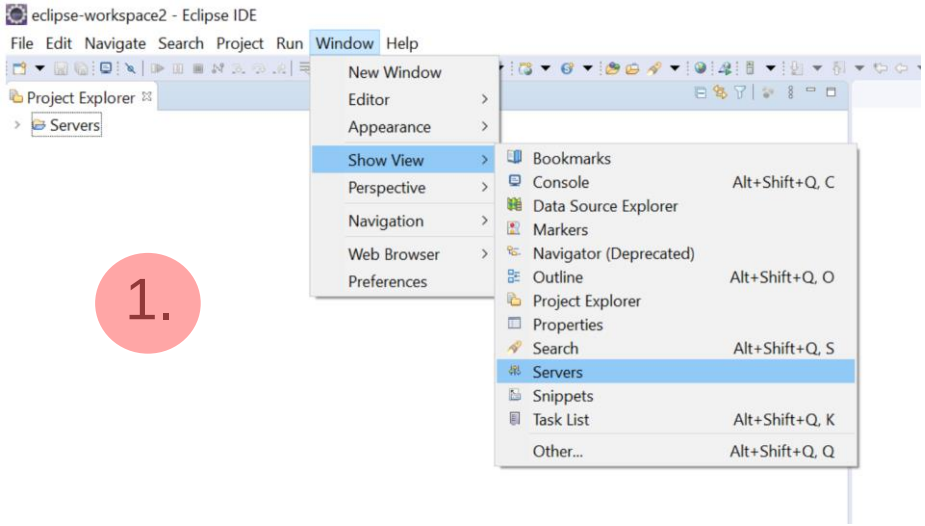
- Esercizio: provare ad avviare Tomcat da Eclipse ed effettuare debug locale

Apache Tomcat integrato con Eclipse

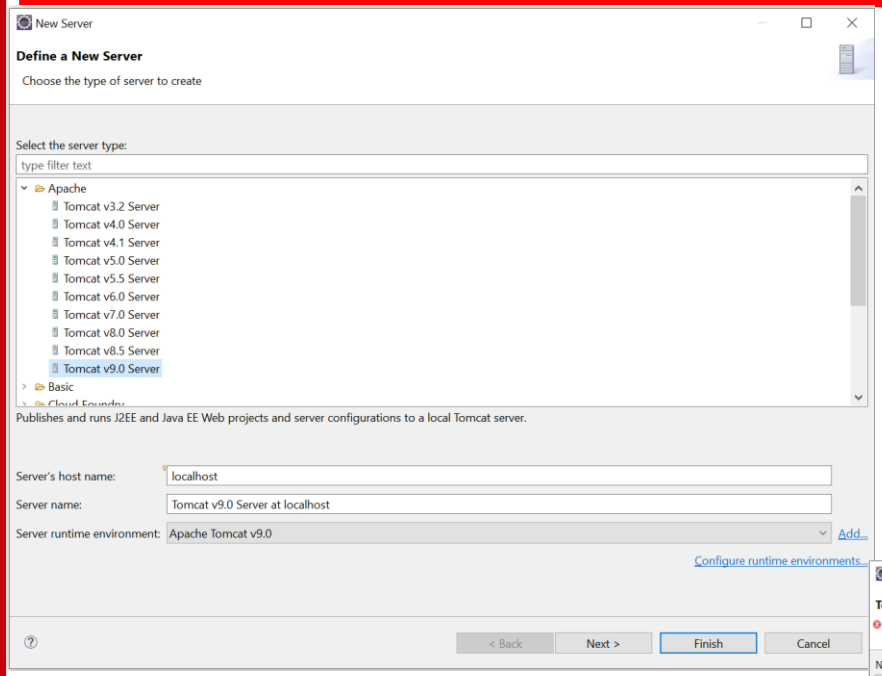
- Il modo più facile e intuitivo per utilizzare Tomcat è integrarlo con l'IDE Eclipse
- L'integrazione Tomcat-Eclipse permette:
  - Start/Stop del Web Server direttamente dall'IDE
  - Modificare facilmente i parametri di configurazione tramite GUI
  - Debug remoto local-like
  - Visualizzare eccezioni/errori/output dei componenti lato Server direttamente nella console dell'IDE
  - Gestire Server multipli
  - ecc.
- …ripartiamo da «abbiamo appena scaricato ed estratto tomcat sul desktop»
  - Potrebbe essere utile utilizzare Tomcat dalla vostra chiavetta USB
    - questa soluzione porta le operazioni di deployment ad essere più time-consuming

Integrazione Tomcat-Eclipse (1)

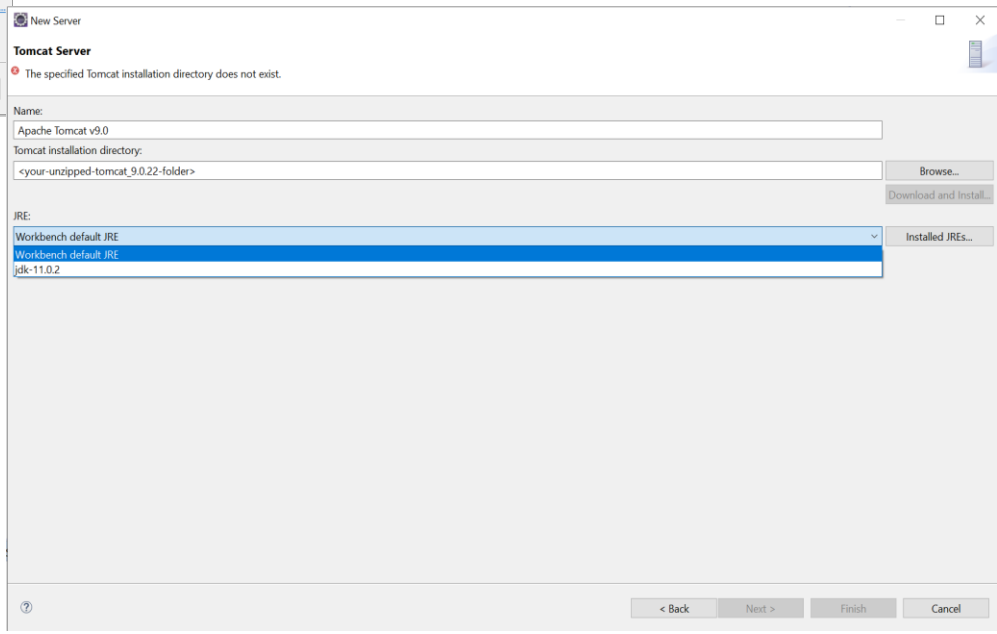
1. Per prima cosa ci serve la view «Servers»
   1. Window > Show View > Servers … nel caso non la trovaste direttamente da Show View, continuare su Other > Server > Servers
2. Procedere su Create New Server
3. Da qui selezionare sotto la cartella Apache la versione di Tomcat che avete scaricato, poi Next
4. Indicare il path del vostro Tomcat
5. e selezionate la giusta JDK
6. Finish



Integrazione Tomcat-Eclipse (2)



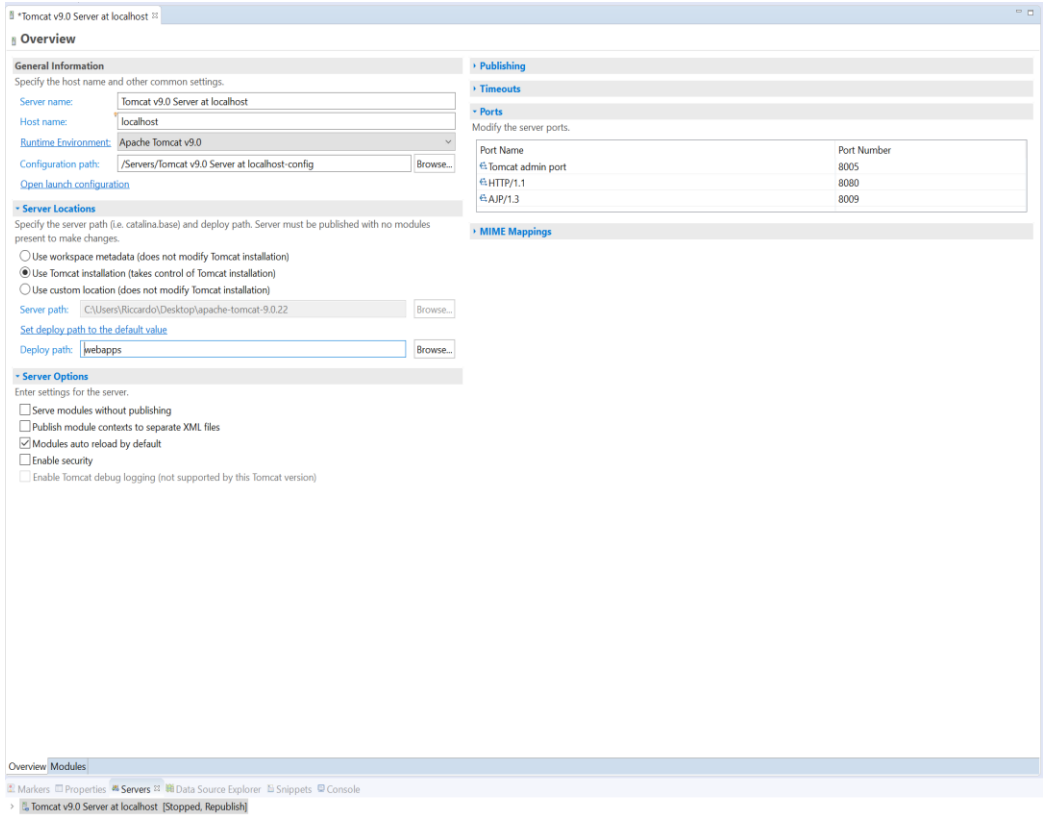
3.



4. – 5.

Integrazione Tomcat-Eclipse (3)

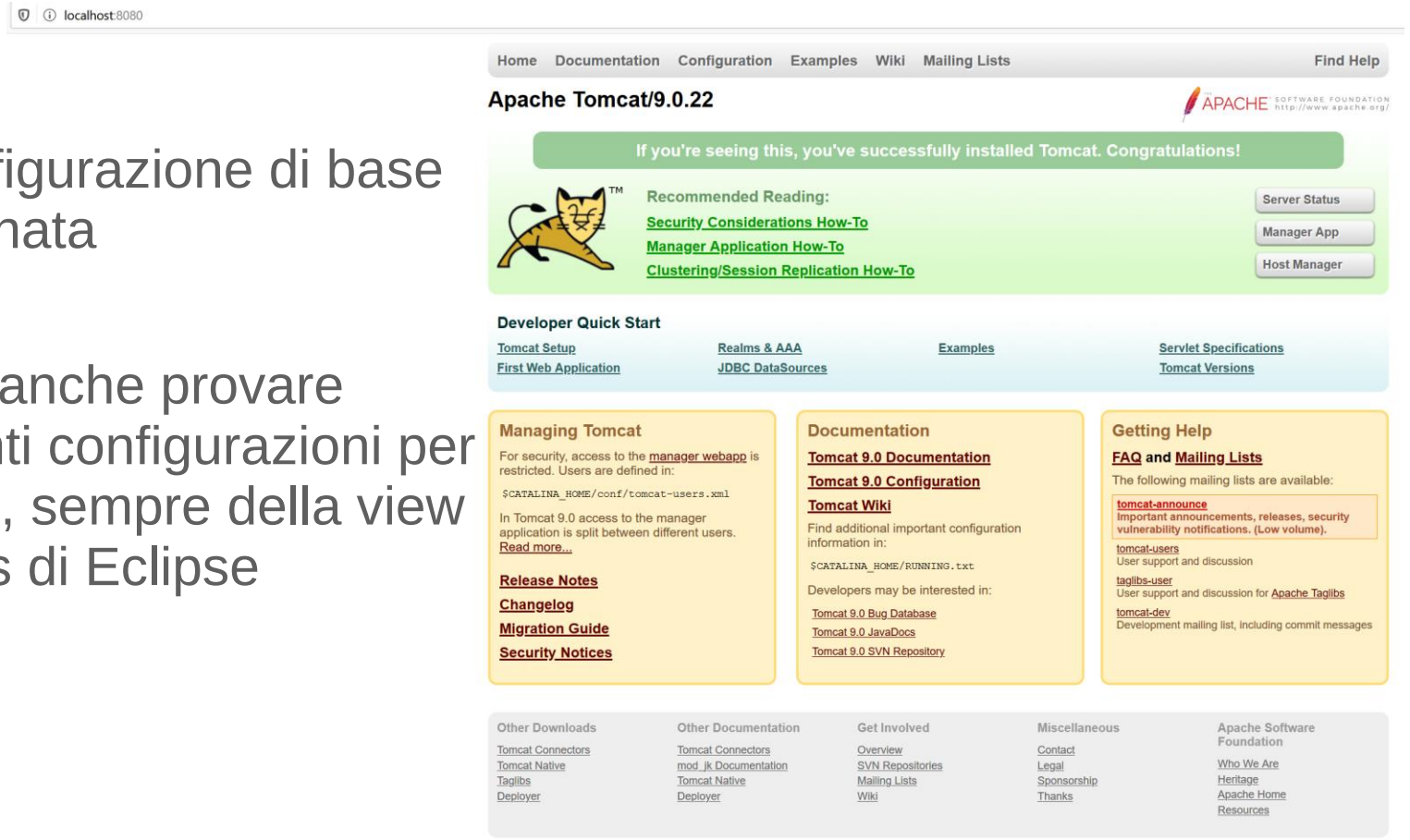
- Ora doppio click sulla voce di Tomcat comparsa nella vista Servers
- Sotto Server Location selezionare:
  - Use Tomcat installation
  - Impostare correttamente il deploy path su: webapps
- Infine salvare
- Provate a lanciare il Server con il pulsante Play/Stop
- Provate a connettervi a http://localhost:8080



Da questa finestra è possibile impostare altri parametri di Tomcat, anche il Debug (lo vedremo fra poco..)

Integrazione Tomcat-Eclipse (4)

- Questa è la pagina che dovrebbe comparire all'indirizzo localhost:8080



- La configurazione di base è terminata
- Potete anche provare differenti configurazioni per Tomcat, sempre della view Servers di Eclipse

Tomcat-Eclipse: «debug locale»

- Come sappiamo, il debugging è una pratica fondamentale
- Vediamo come eseguire il debugging di una Web application server-side
- Andiamo nella vista Servers di Eclipse
  - Doppio click Server Tomcat creato in precedenza
  - Sotto General Information
  - Click su «Open launch configuration»
  - Andiamo sotto il tab «Source»
  - Add > Java Project
  - Selezionare il progetto di cui vogliamo fare il debugging, OK, Applica
- Riavviamo Tomcat in modalità debug
  - cliccando sull'insettino, invece che il solito bottone play
- Possiamo ora sfruttare tutte le feature del debug locale dell'IDE, come breakpoint, step over, step into, ecc., per debugging di applicazioni Web remote

Firefox for Developer (Firebug) (1)

- Cos'è **Firefox for Developer**?

  È una versione del browser **Mozilla Firefox** gratuita e open source pensata per gli sviluppatori Web
- Universalmente riconosciuta come uno strumento indispensabile per lo sviluppo di "*rich internet application*"
- Funzionalità offerte
  - **Navigazione** all'interno del DOM delle pagine web
  - **Analisi** e **modifica** dell'HTML e dei fogli di stile in tempo reale
  - **Monitoraggio**, **debug** e **modifiche** del codice Javascript in tempo reale
  - **Visualizzazione** nel browser, in tempo reale, delle modifiche apportate

Interfaccia Inspect di Firefox

- Consiste di 6 schede principali
  - **Console**: riporta log relativi al codice Javascript richiamato dalla pagina, e permette la scrittura ed esecuzione di codice aggiuntivo
  - **HTML**: riporta il sorgente della pagina in esame evidenziando le aree su cui si sposta il mouse
  - **CSS**: visualizza e permette di modificare gli stili associati ai componenti HTML nella pagina
  - **Script**: supporta il debug del codice JavaScript eseguito dalla pagina
  - **DOM**: riproduce la struttura ad albero del DOM della pagina e ne permette modifiche
  - **Net**: analisi dei tempi di download di ogni risorsa nella pagina e delle connessioni effettuate

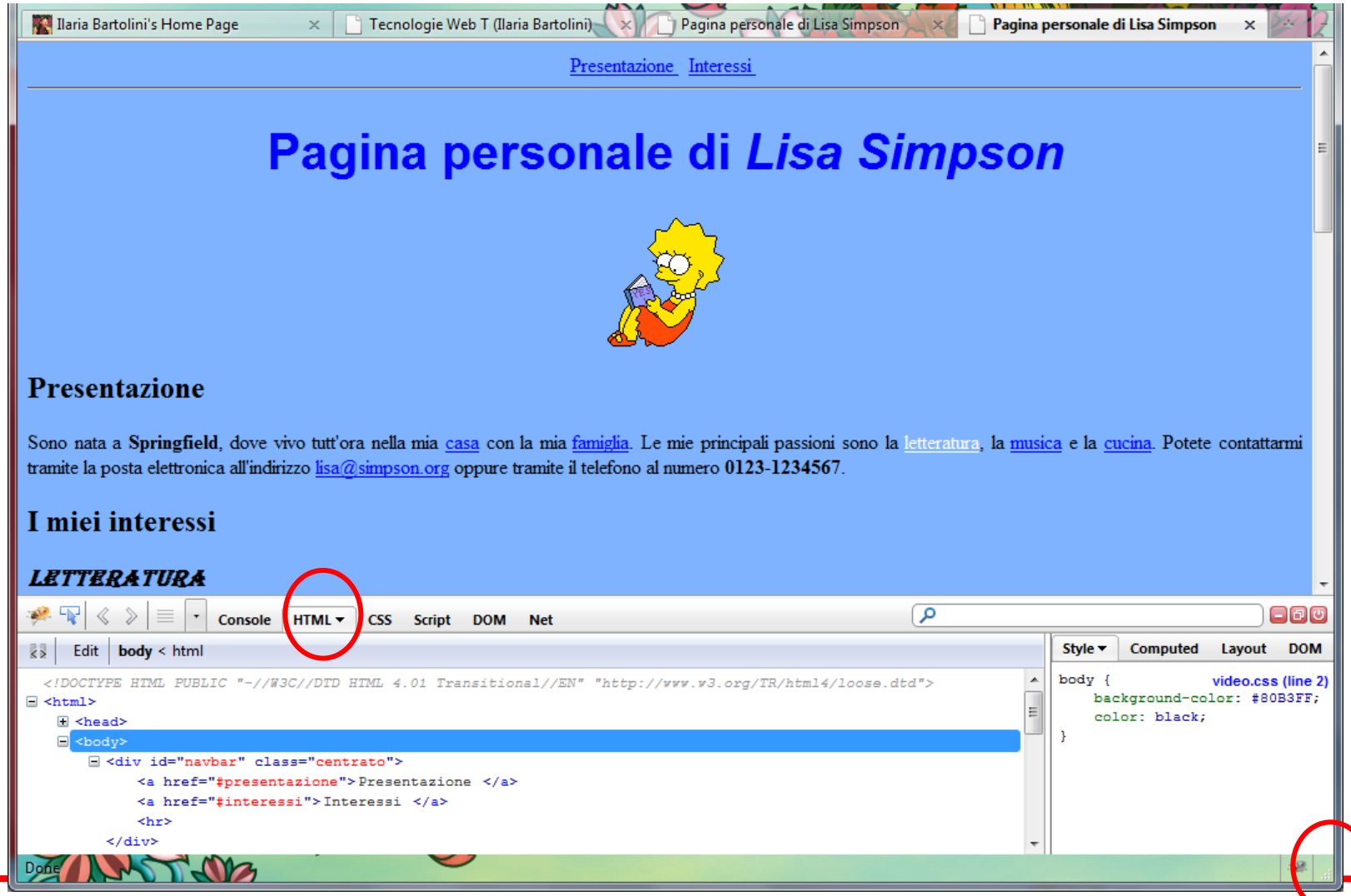
Firefox Vs Firefox for Developer

La feature che ci interessa maggiormente, l'Inspect Element, è presente in entrambe le versioni, ma ecco alcune differenze:

- **Profili Separati**: si mantiene l'ambiente di sviluppo separato da quello di browsing standard, rendendolo specificamente configurabile solo per lo sviluppo web
- **Feature**: nella versione for Developer si ha accesso a funzionalità esclusive per lo sviluppo
- **Supporto**: lead time di 12 settimane sul supporto per le ultime aggiunte agli standard web
- **Configurazione**: preferenze e configurazioni di default per sviluppatori
- **Tema**: tema Dark per sviluppatori

Firefox Inspect Element: un esempio

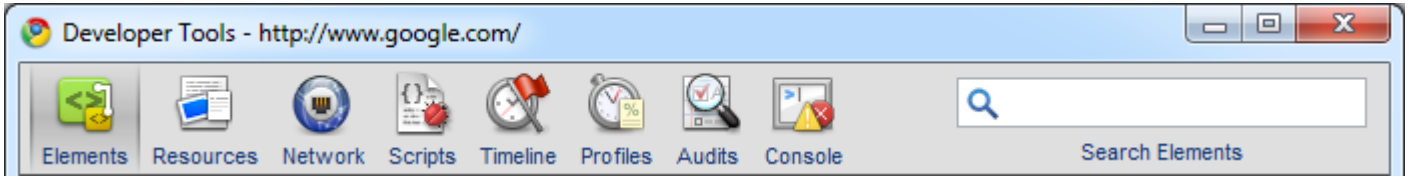
Vediamo l'Inspect Element tool "all'opera" sulla pagina web di Lisa Simpson:



Firefox vs. Google Chrome Developer Tools

- Cos'è **Chrome Developer Tools**?

  È una suite di tool inclusi nel browser **Google Chrome** le cui funzionalità sono del tutto analoghe a quelle offerte dall'Inspect Element di Firefox nella sua versione completa



- Strumento alternativo per lo sviluppo di "*rich internet application*"
- Funzionalità offerte
  - **Navigazione** all'interno del DOM delle pagine web
  - **Analisi** e **modifica** dell'HTML e dei fogli di stile in tempo reale
  - **Monitoraggio**, **debug** e **modifiche** del codice Javascript in tempo reale

Tips & Tricks

- Il filesystem dei lab è usualmente accessibile in sola lettura
  - solo la directory c:\temp è accessibile in scrittura; il suo contenuto viene cancellato sistematicamente
  - ogni utente ha a disposizione 200MB di spazio per il profilo
- Si consiglia vivamente di presentarsi alle esercitazione con una memoria esterna (512MB sono più che sufficienti)
  - tale memoria verrà utilizzata per contenere il workspace di Eclipse e le versioni portable di Tomcat
  - in tal modo sarà possibile salvare il lavoro svolto, ad esempio, i progetti realizzati, la configurazione del workspace ed il settaggio dei server
- Infine, alla pagina "Laboratorio" del sito del corso (*http://lia.disi.unibo.it/Courses/twt2122-info/*) potete trovare:
  - link al software utilizzato nelle esercitazioni e relativa manualistica
  - un progetto Eclipse *ANT-based* pronto all'uso