Università degli Studi di Bologna
Facoltà di Ingegneria

Tecnologie Web L-A
A.A. 2009 – 2010

Esercitazione 2
Eclipse, ANT

Tutor:
Ing. Pasini Samuele *samuele.pasini@unibo.it*

Agenda

- Eclipse
  - Caratteristiche generali
  - Importazione di un progetto di esempio
  - Funzionalità di ausilio alla scrittura del codice
  - Compilazione, collaudo ed esecuzione di un'applicazione
- ANT
  - Concetti fondamentali
  - Esempi di utilizzo
- Funzionalità avanzate
  - Debug di un'applicazione

Eclipse, caratteristiche generali

- Ambiente integrato di sviluppo (IDE)
  - interamente scritto in Java
  - multipiattaforma (Win/Mac/Linux/…)
  - multilinguaggio (tool anche per il C)
  - 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 SDK (versioni 5.0 o successive)
  - http://java.sun.com/javase/downloads/index.jsp
- Eclipse IDE
  - http://www.eclipse.org/downloads/
  - http://eclipsetutorial.sourceforge.net/
  - http://eclipsetutorial.sourceforge.net/totalbeginner.html
- Troubleshooting
  - http://www-lia.deis.unibo.it/Courses/TecnologieWeb0809/faq.html
  - 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 V**iew** (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* (*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, 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
  - 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*
  - generazione automatica di codice (costruttori, metodi getter/setter, …)
  - supporto per il refactor (nomi di package, classe, metodi, variabili, ...)
  - ...

- Veramente un sacco di funzionalità
  - *right-click* dovunque :)
  - menu *Help → Search*
  - sito di eclipse, tutorial on-line (spesso persino animati)
  - Ricerche specifiche su Google

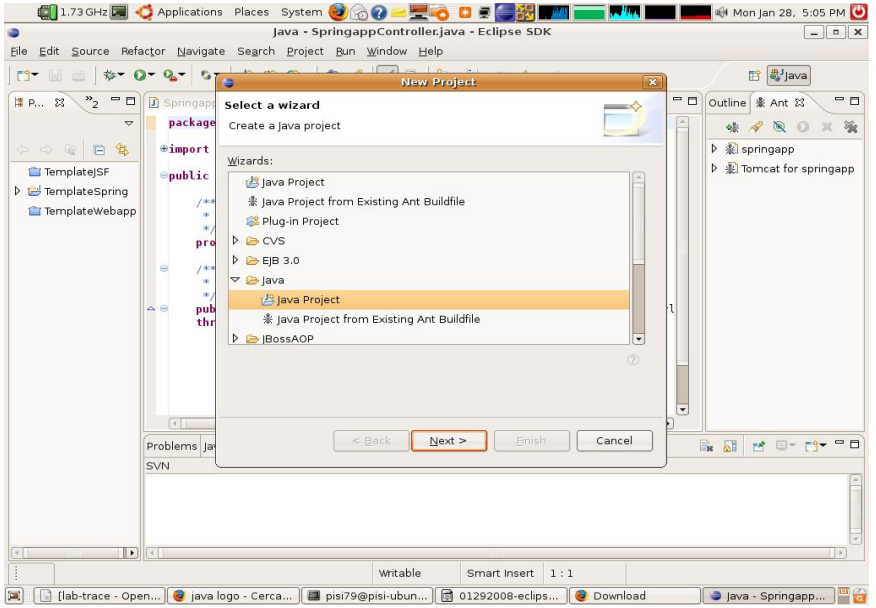
Gestione dei progetti

- 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 "omononimo" contenuto nel file ZIP non viene neanche visualizzato tra i progetti individuati nell'archivio



Importazione di un progetto

- All'interno dell'archivio ZIP dell'esercitazione, nel direttorio ***progetti***
  - il file ***02_TemplateApp.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 del direttorio radice
  - ***src:*** sorgente (file *.java*) dell'applicazione da sviluppare
  - ***test:*** sorgente delle routine di test (opzionali) che verificano il corretto funzionamento dell'applicazione
  - ***LIBRERIE*** *(la 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 **implementezioni**. riferite 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 un ulteriore 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, ...
- Tali 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)
  - ...
- Gli strumenti di sviluppo come ANT, 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 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
  - *http://www-lia.deis.unibo.it/Courses/TecnologieWeb0708/materiale/laboratorio/guide/ant/manual/index.html*

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 ed indipendente dagli specifici OS e IDE e dai diversi percorsi locali su file system
  - di 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 DA MODIFICARE per poter completare l'esercitazione

- Accorgimenti:
  - E' possibile lanciare *ant* da riga di comando
    - **cd $PROJECT_HOME/ant**
    - **ant <nome_obiettivo>**
  - E' 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
  - ...ed 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...
  - ...e, in particolare, le applicazioni “web” eseguono all'interno di opportuni server “contenitori”

- Per poter “debuggare” 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
  - lancianre 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