Universita' degli Studi di Bologna
Facolta' di Ingegneria

Anno Accademico 2008-2009

Laboratorio di Tecnologie Web
Introduzione ad Eclipse

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

Cosa è Eclipse

- 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
  - Espandibile da chiunque (wizard per creare nuovi plug-in all'interno dello stesso eclipse)

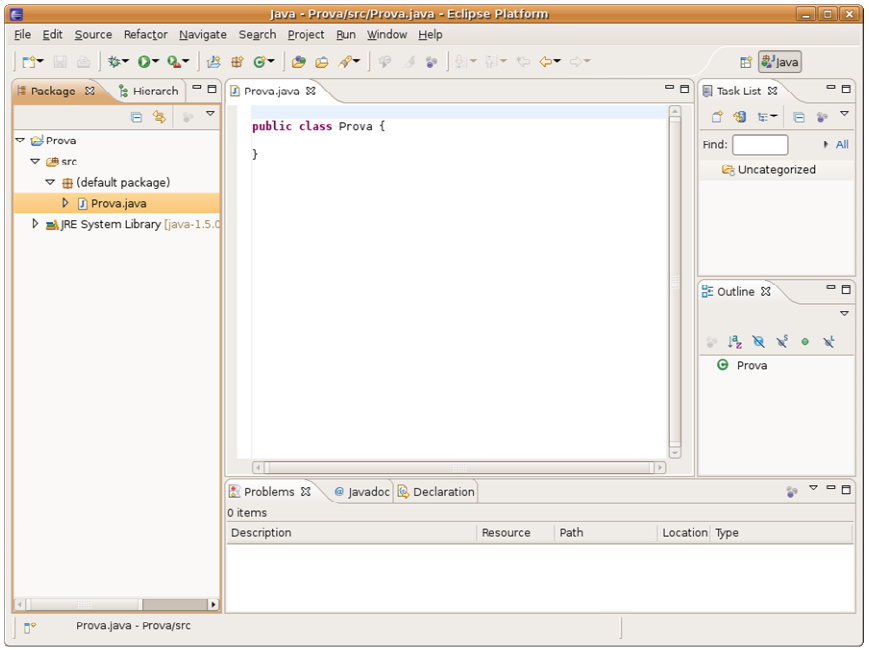
Per iniziare…

- 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/

Il mio primo impatto

- Avviamo Eclipse
  - Scelta del direttorio per il workspace
- Welcome... eccetera → *close*
  - Dovesse servire.. *Help → Welcome*
- **Workbench** (area di lavoro) come insieme di view (viste)
  - *Package* (struttura logica dei progetti)
  - *Navigator* (struttura fisica dei file)
  - *Outline* (struttura dei file aperti)
  - *Java Editor* (dove si codifica)
  - *Console* (stdout)
  - *Problems* (**problemi da risolvere**)

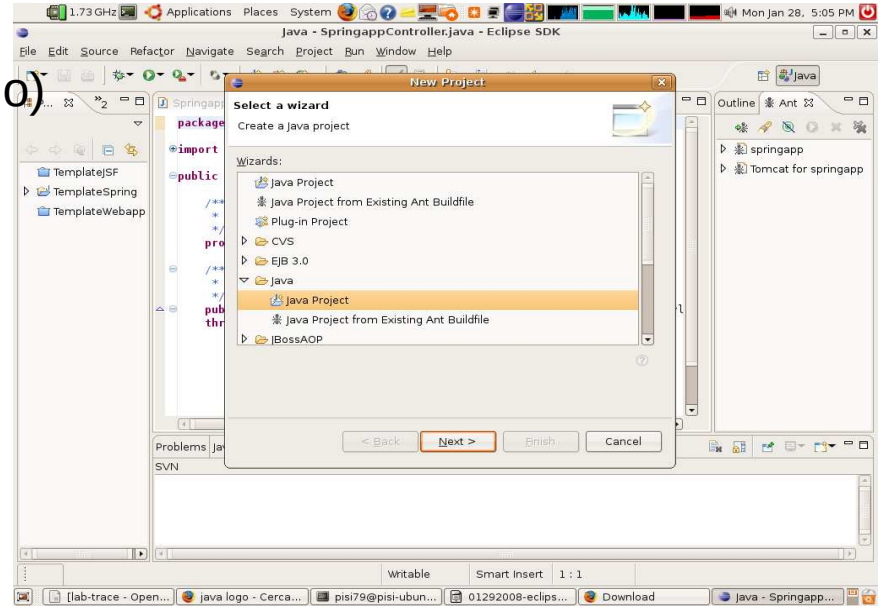
…e altre (*Window → Show view*)



- Perspective (prospettiva) come associazione di un preciso insieme di viste, in precise posizioni, per affrontare tipiche operazioni (*Windows → Open perspective*)

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*
- Nel workspace non possono esistere più progetti con lo stesso nome (rinominare quello esistente per importarne uno omonimo)
- Esploriamo le funzionalità contestuali
  - *right click…*
- Problemi (librerie, versioni JVM, …)?
  - vista *Problems* (diagnosi)
  - menu *Project → Properties* (soluzione)



Perché non un normale editor di testo e la shell?

- Autocompletamento (parentesi, nomi delle variabili, modificatori di tipo, …. si attiva da solo o premendo *Ctrl*+*Space*)
- Generazione automatica di codice (costruttori, metodi getter/setter, …)
- Evidenziazione (parole chiave del linguaggio, errori, …)
- Messaggi di errore e consigli per risoluzione (a volte) automatica
- Supporto per il refactor: *"Ho cambiato idea.. La classe* `MyClass` *diventerà superclasse di di qualcos'altro e cambierà anche package"*

Veramente un sacco di funzionalità!

Esplorare da soli e fare prove:

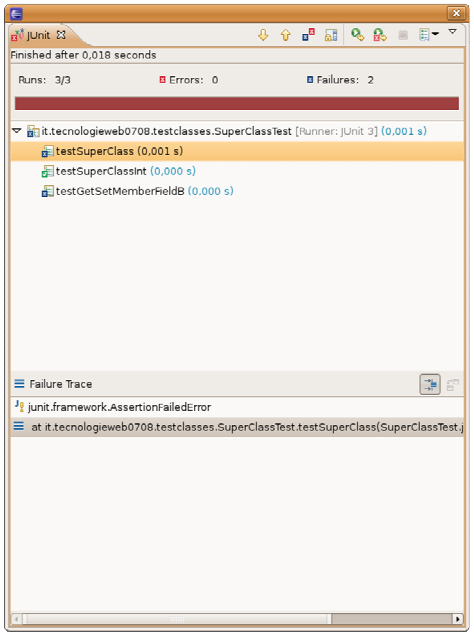
- *right-click* dovunque ☺
- *Help → Search*
- Tutorial online, Google, …

Importare i progetti di esempio

- Sul sito del corso…
  - http://www-lia.deis.unibo.it/Courses/TecnologieWeb0809/laboratorio.html
  - http://www-lia.deis.unibo.it/Courses/TecnologieWeb0809/materiale/laboratorio/.....zip
- Poche semplici classi per provare…
  - …evidenziazione
  - …parsing del codice
  - …autocompletamento
  - …esperimenti vari
- Problemi di configurazione di Eclipse
  - discussione in laboratorio (JVM, compiler version, buildpath, User Libraries…)
  - arricchimento delle FAQ sul sito
- Introduzione al collaudo automatico (JUnit)
- Introduzione all'utilizzo di tool di build (ANT)

Collaudo automatico (JUnit)

- Integrazione con la suite JUnit
  - **librerie** di JUnit già parte della configurazione base di Eclipse
    - JUnit3 da Eclipse 3.1 (basato su ereditarietà)
    - JUnit3 e JUnit4 (basato su annotazioni) da Eclipse 3.3
  - operazioni integrate nei **menu contestuali**
    - *Right-click* sulle classi di test → *Run As* → *JUnit Test*
  - **wizard** per la creazione di nuove classi di test
    - *Right-click* sulla classe da testare → *New* → *JUnit Test Case*
  - **viste** per la reportistica e l'analisi degli errori



- Tutorial on-line
  - http://open.ncsu.edu/se/tutorials/junit/ (Junit 3.8)
  - http://www.instrumentalservices.com/media/articles/java/junit4/JUnit4.pdf

Compilare, eseguire, redistribuire… perché un tool di build (1)

- A default, Eclipse **compila** automaticamente tutti i progetti aperti nel workspace (*Project → Build automatically*)
  - …file *.class* (bytecode) collocati insieme ai *.java* (codice sorgente) o in un direttorio *ProjectName/bin* con struttura analoga a *ProjectName/src*
  - la struttura è completamente configurabile (*Project → Properties*)

- **Esecuzione** via menu contestuale (*Right click* su una classe con metodo `main()` → *Run As → Java Application*)
  - configurabile (*Right click → Run Configurations…*)
    - argomenti, proprietà per la JVM, classpath, variabili di ambiente, …
    - **le applicazioni web tuttavia non hanno un metodo `main()`**: girano all'interno di un container che ne gestisce invocazioni, ciclo di vita, …

Compilare, eseguire, redistribuire… perché un tool di build (2)

- **Packaging** come file *.jar* (archivi zip di codice + descrittori + risorse)
  - distribuzione del *bytecode*
    - librerie per altri progetti
    - **deployment** in un container
  - integrazione in Eclipse
    - Abilitazione del packaging mediante Project properties
    - Wizard per la configurazione del contenuto del file *.jar*

➢ difficile ricordarsi “dove sta cosa”

➢ tedioso fare sempre le stesse sequenze di operazioni

➢ difficile specificare “come fare cosa” per condividere il lavoro

➢ si rimane legati a un particolare IDE

➢ …e come se non bastasse le strutture dei menù cambiano anche fra differenti versioni dello stesso IDE

Il più diffuso tool di build (ANT)

- **Obiettivi (*target*)** relativi al progetto definiti come **insiemi ordinati di attività (*task*)** da mandare in esecuzione
  - Impostazione di proprietà
  - Compilazione dei sorgenti
  - Creazione di archivi per la distribuzione
  - Deployment all'interno di un container
  - Esecuzione di applicazioni
  - Stampa di informazioni a video
  - …molto molto altro
- Attività di tipo
  - *core* (`javac, delete, kdir`, …) predefinite e preimplementate
  - *optional* (`scp, xmlvalidate,` …) predefinite + librerie aggiuntive
  - personalizzate (`taskdef name=`"…") da definire + librerie aggiuntive
- Descrizione dichiarativa, leggibile, manutenibile e portabile
  - http://ant.apache.org/manual/index.html

Il più diffuso tool di build (ANT)

- ANT si basa su
  - file XML (detti file di *build*) → descrizione degli obiettivi
  - file di proprietà → caratteristiche legate al contesto (es: percorsi locali, credenziali, ecc…: le cose che possono assumere valori diversi su macchine diverse!)
  - librerie java (core + opzionali + personalizzate) che implementano le attività utilizzate nell'XML
  - un'applicazione Java (contenuta in *ant.jar*) → analisi dell'XML e attuazione degli obiettivi
- Download ed installazione
  - già integrato in Eclipse (viste, menu contestuali, …)
  - usabile anche da shell (aggiunta di ANT_HOME al path dei comandi) o altri IDE

```
/home/me/myworkspace/MyProject/ant$ ant package
```

- All'interno di Eclipse, ANT **non è influenzato dal buildpath** del progetto!!
  - file *.properties* contenenti path e altre proprietà
  - target "preparatori" per leggere queste proprietà
  - dipendenza dei target "veri e propri" da quelli preparatori

Lanciare un target di ANT

- 3 possibili modi
  - da shell

    posizionandosi nel direttorio che contiene il file di build

    ```
    $ ant nomeDelTarget
    ```

  - dalla vista *Outline*

    Double-click sul file di build (si apre nell'editor principale e nella vista *Outline*

    Right-click su un target nella vista *Outline* → *Run As…* → *Ant build*

    Possibile configurare parametri e proprietà scegliendo → *Ant build…*

  - dalla vista *Ant*

    Window → *Show view* → *Ant (*scegliere *Other* se non è presente e cercarla)

    Trascinare il file di build nella vista *Ant*

    Double-click su un target nella vista *Ant*