Qualità del codice, librerie, programmazione multipiattaforma, profiling

Programmazione ad Oggetti – Lab08

Docenti: Roberto **Casadei**, Danilo **Pianini**
Tutor: Luca **Deluigi**

C.D.S. Ingegneria e Scienze Informatiche
Alma Mater Studiorum—Università di Bologna, Campus di Cesena

4 aprile 2023

Outline

1. Programmazione multipiattaforma
   - Accesso al file system
   - Accesso ai dettagli del sistema
   - GUI scalabili e internazionalizzazione
2. Costruzione di un ambiente di sviluppo portabile
   - Corretta configurazione di un progetto Eclipse, inclusione di librerie
   - Font ed encoding
   - Caricamento di risorse dal classpath
   - Installazione delle impostazioni per-utente
3. Ricerca/Utilizzo di librerie
4. Qualità del codice
   - Strumenti per il controllo di qualità
   - Coding style: configurazione IDE Eclipse
5. Ispezione e Profiling con VisualVM
6. Preparazione al laboratorio

“Write once, run anywhere...”

Lo slogan coniato originariamente da Sun Microsystems per illustrare i benefici del linguaggio Java vale a patto che:

1. **“write” sia fatta in modo corretto e robusto**;
   - ovvero, sia adottato un approccio di programmazione adeguato.
2. **sia possibile distribuire ciascuna applicazione per qualunque piattaforma**.
   - ovvero, sia predisposto un packaging efficace.

Outline

1. Programmazione multipiattaforma
   - Accesso al file system
   - Accesso ai dettagli del sistema
   - GUI scalabili e internazionalizzazione
2. Costruzione di un ambiente di sviluppo portabile
   - Corretta configurazione di un progetto Eclipse, inclusione di librerie
   - Font ed encoding
   - Caricamento di risorse dal classpath
   - Installazione delle impostazioni per-utente
3. Ricerca/Utilizzo di librerie
4. Qualità del codice
   - Strumenti per il controllo di qualità
   - Coding style: configurazione IDE Eclipse
5. Ispezione e Profiling con VisualVM
6. Preparazione al laboratorio

Path e Separatori

Inserire dei path assoluti nel proprio sorgente è **sempre** fonte di problemi quando si scrive software multipiattaforma:

- `C:\Users\UserName\file` — Non funzionerà su piattaforma *nix, e non funzionerà se l'utente Windows non è "UserName".
- `C:\MyProgram\file` — Non funzionerà su piattaforma *nix, e non funzionerà se l'installazione di Windows è sana e il software non è avviato con diritti di amministratore.
- `/home/username/file` — Non funzionerà su piattaforma Windows, e non funzionerà se l'utente non è `username`.

Problemi

- I separatori per i path cambiano a seconda dell'OS
- La struttura del file system cambia con l'OS
- I diritti di lettura e scrittura cambiano con la configurazione

Proprietà di sistema

Java fornisce nella classe `System` un metodo

```
String getProperty(String p)
```

che consente di accedere a proprietà di sistema

Proprietà relative al file system

- `file.separator` — Restituisce \ per Windows e / per Unix
- `java.home` — La directory di installazione di Java
- `user.dir` — La directory da cui il comando `java` è stato invocato
- `user.home` — Restituisce la home directory dell'utente che ha lanciato `java`
- `user.name` — Restituisce il nome utente

Outline

1. Programmazione multipiattaforma
   - Accesso al file system
   - Accesso ai dettagli del sistema
   - GUI scalabili e internazionalizzazione
2. Costruzione di un ambiente di sviluppo portabile
   - Corretta configurazione di un progetto Eclipse, inclusione di librerie
   - Font ed encoding
   - Caricamento di risorse dal classpath
   - Installazione delle impostazioni per-utente
3. Ricerca/Utilizzo di librerie
4. Qualità del codice
   - Strumenti per il controllo di qualità
   - Coding style: configurazione IDE Eclipse
5. Ispezione e Profiling con VisualVM
6. Preparazione al laboratorio

Funzionalità OS-specifiche

- Talvolta è possibile che in una applicazione si debbano utilizzare librerie non disponibili o non licenziate per alcune piattaforme.
- A supporto di ciò, Java fornisce delle proprietà che consentono di identificare OS, versione e JVM corrente.

Proprietà relative al sistema

- `java.version` — La versione di `java` in uso. Si potrebbe decidere di non usare una funzionalità che si sa non esistere o essere buggata.
- `os.arch` — L'architettura della CPU come rilevata dall'OS (es. x86, i386, amd64, x86_64, IA64N, arm,...)
- `os.name` — Il nome del sistema operativo (es. Linux, MacOS X, MacOS, Windows 8.1, Windows 10, Solaris, FreeBSD, ...)
- `os.version` — Restituisce per Windows il numero di versione effettivo (es. Windows 10 restituisce 10.0), per MacOS il numero di versione (es. 10.3.4), per Linux la versione del kernel (es. 4.17)

Outline

1. Programmazione multipiattaforma
   - Accesso al file system
   - Accesso ai dettagli del sistema
   - GUI scalabili e internazionalizzazione
2. Costruzione di un ambiente di sviluppo portabile
   - Corretta configurazione di un progetto Eclipse, inclusione di librerie
   - Font ed encoding
   - Caricamento di risorse dal classpath
   - Installazione delle impostazioni per-utente
3. Ricerca/Utilizzo di librerie
4. Qualità del codice
   - Strumenti per il controllo di qualità
   - Coding style: configurazione IDE Eclipse
5. Ispezione e Profiling con VisualVM
6. Preparazione al laboratorio

GUI scalabili – Motivazioni

Flessibilità

Diversamente dagli anni 90, i dispositivi oggi hanno una densità di pixel per area **estremamente** variabile. Si va da 120 PPI a 640 PPI, su schermi di dimensione estremamente variabile (da 3 a 200 pollici).

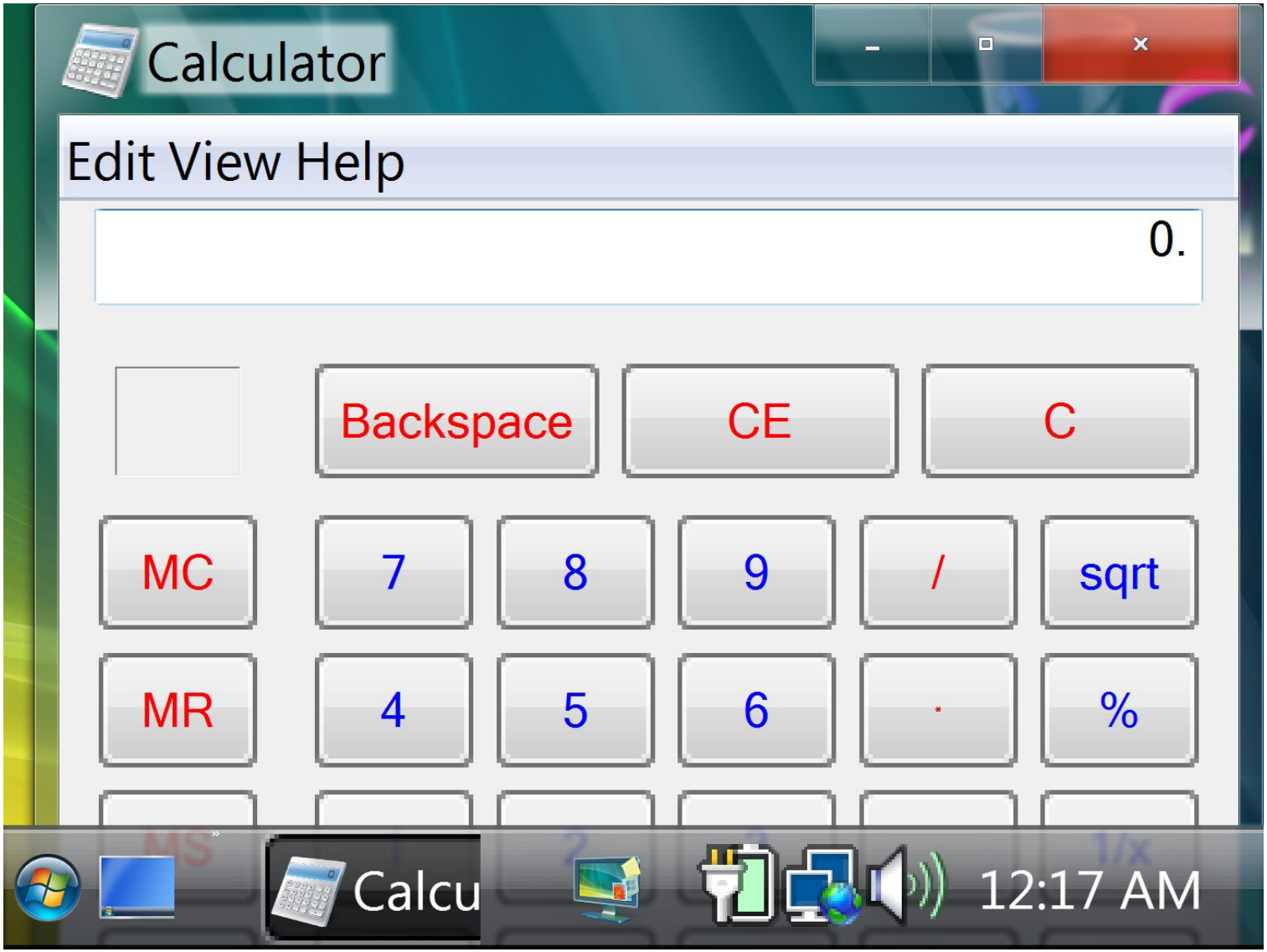
Multipiattaforma

Piattaforme diverse, anche a parità di schermo, possono adottare diverse convenzioni:

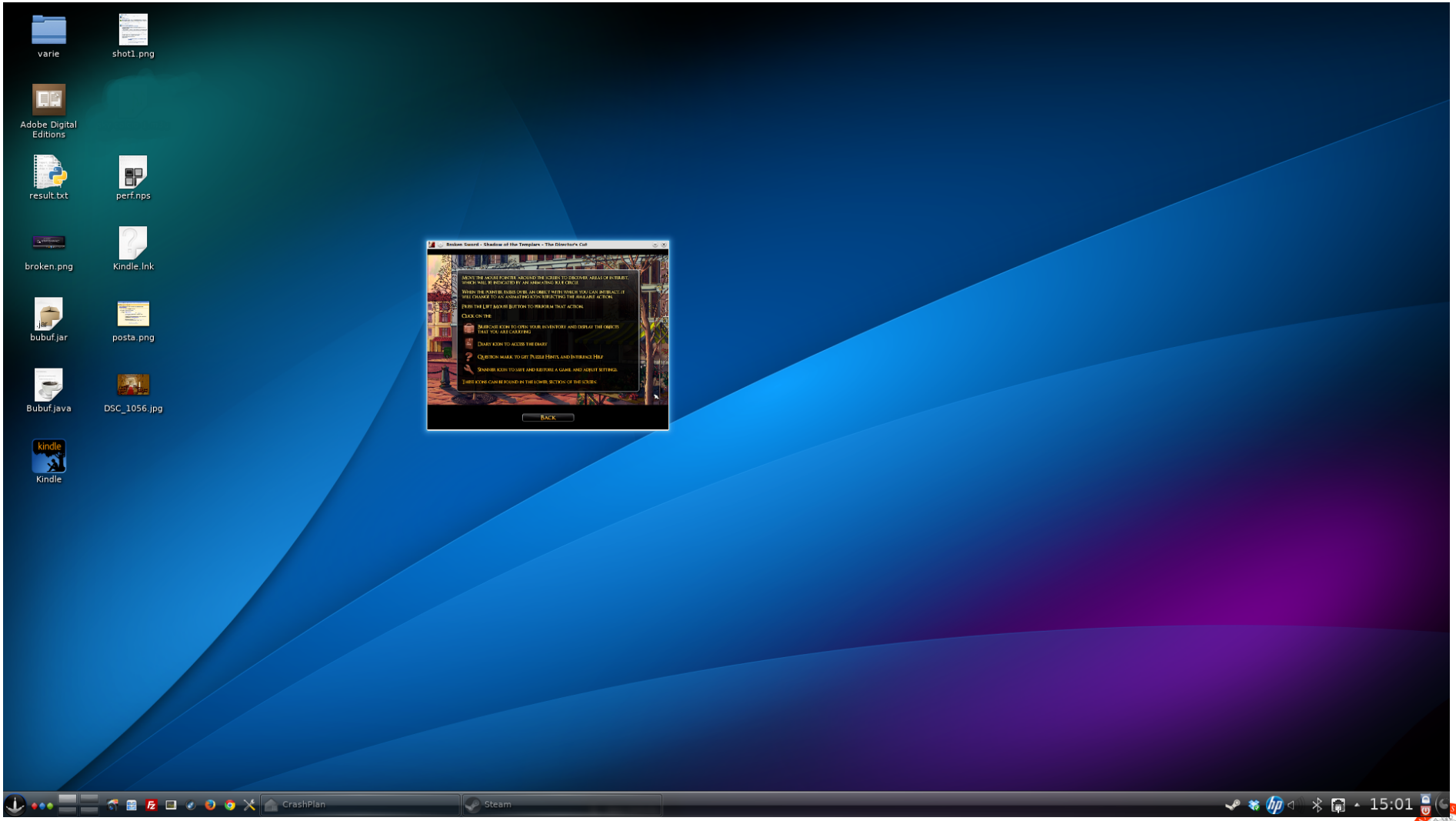
- Diversa grandezza di bordi
- Diversa spaziatura, dimensione e tipo di font
- Diverso sistema di decorazioni

Questi elementi sono stabiliti dal *compositor* e non dallo sviluppatore dell'applicazione. Come indicazione generale vale che **un'applicazione ben sviluppata eredita il "look and feel" dal sistema su cui sta girando**.

Esempio: Finestra non ridimensionabile e bassa risoluzione



Esempio: Finestra non ridimensionabile e alta risoluzione



Best-practices per la costruzione di GUI

- La dimensione di default della finestra va calcolata in base alla dimensione dello schermo.
- È opportuno non specificare dimensioni assolute in pixel per i componenti della GUI, ma dimensionarli in termini relativi rispetto al container.
- Anche per i layout è opportuno non utilizzare dimensioni fisse in pixel.
- I font possono essere allegati all'applicazione
- La dimensione dei font può essere resa relativa alla dimensione corrente della view.
- L'ultente deve essere libero di ridimensionare l'interfaccia a piacimento per adattarla alla propria configurazione di schermi

Supporto multilingua per le UI

- Sarebbe opportuno definire la UI una sola volta e cambiare dinamicamente le parti scritte (il testo) a seconda dell'impostazione della lingua di sistema (o della nostra applicazione).
  - In realtà, non solo per la lingua ma anche per il formato dei numeri, la valuta, le convenzioni sulla data,...

Java Resource Bundles

Java fornisce una architettura per l'internazionalizzazione, che fa uso di "**ResourceBundle**" e di una serie di file di supporto (properties files).
Per approfondimenti (per implementare il supporto multilingua):

- `https://docs.oracle.com/javase/tutorial/i18n/resbundle/index.html`
- `http://tutorials.jenkov.com/java-internationalization/resourcebundle.html`

Outline

1. Programmazione multipiattaforma
   - Accesso al file system
   - Accesso ai dettagli del sistema
   - GUI scalabili e internazionalizzazione
2. Costruzione di un ambiente di sviluppo portabile
   - Corretta configurazione di un progetto Eclipse, inclusione di librerie
   - Font ed encoding
   - Caricamento di risorse dal classpath
   - Installazione delle impostazioni per-utente
3. Ricerca/Utilizzo di librerie
4. Qualità del codice
   - Strumenti per il controllo di qualità
   - Coding style: configurazione IDE Eclipse
5. Ispezione e Profiling con VisualVM
6. Preparazione al laboratorio

Outline

1. Programmazione multipiattaforma
   - Accesso al file system
   - Accesso ai dettagli del sistema
   - GUI scalabili e internazionalizzazione
2. Costruzione di un ambiente di sviluppo portabile
   - Corretta configurazione di un progetto Eclipse, inclusione di librerie
   - Font ed encoding
   - Caricamento di risorse dal classpath
   - Installazione delle impostazioni per-utente
3. Ricerca/Utilizzo di librerie
4. Qualità del codice
   - Strumenti per il controllo di qualità
   - Coding style: configurazione IDE Eclipse
5. Ispezione e Profiling con VisualVM
6. Preparazione al laboratorio

Corretta struttura di un progetto

Project main folder

- src -- Sorgenti
- res -- Risorse (icone, file di configurazione...)
- lib -- Librerie esterne (e.g. file .jar, .so, .dll...)
- bin -- Binario (con copia delle risorse)
- doc -- Documentazione
- README.md
- LICENSE

Corretta struttura di un progetto: Dettagli I

`src`

- Contiene **esclusivamente** il codice sorgente dell'applicazione organizzato in package. Non deve contenere altro!
- Eventualmente possono esistere più cartelle sorgente.
  - Ad esempio perché parte del codice viene generato da un sistema automatico, oppure perché il progetto è sviluppato usando più di un linguaggio.
  - Nel caso di generatori, è buona norma dividere il codice generato da quello generante, ad esempio creando una cartella `src-gen`.
  - Nel caso si utilizzino più linguaggi, è bene distinguerli usando source folder separate, es. `src-java` e `src-prolog`.
- In Eclipse è possibile marcare una cartella come "cartella sorgente": Proprietà del progetto → Java Build Path → Source → Add Folder → Selezione della cartella → OK → OK
  - Tutti i sorgenti di tutte le cartelle sorgente verranno compilati ed inseriti nella cartella di output da Eclipse

Corretta struttura di un progetto: Dettagli II

`res`

- Contiene le risorse del progetto opportunamente organizzate
  - Per risorse si intendono icone, file di testo, video, immagini, modelli 3D e qualunque cosa sia necessaria al corretto funzionamento del programma ma non sia una libreria o un file sorgente.
- In Eclipse, per far sì che le risorse vengano inserite nel classpath (da cui sarà più facile caricarle), è necessario rendere `res` una cartella sorgente con la procedura descritta nella slide precedente.

`lib`

- Contiene tutte le librerie che si sono usate, che siano in file jar o che siano librerie native in formato .so (o .dll)

Corretta struttura di un progetto: Dettagli III

doc

- Contiene la documentazione del progetto.
- Se si decide di inserire sia documentazione di tipo non rigenerabile (e.g. una relazione in pdf) sia quella di tipo rigenerabile (e.g. la javadoc), è bene tenerle separate.
- **La documentazione rigenerabile non va tracciata col DVCS**

bin

- Contiene la versione compilata del software.
- Contiene una copia delle risorse.
- Viene interamente manutenuta dall'IDE.
- **Non va tracciata col DVCS**

Corretta struttura di un progetto: Dettagli IV

README.md

- File con la descrizione del progetto: autori, breve guida d'uso, link a risorse.
  - GitHub è in grado di fare il parse del file e di integrarlo nella pagina del progetto, in modo da dargli una descrizione.

LICENSE

- File con informazioni circa la licenza.
  - Necessario affinché il progetto sia open source.
  - Progetti il cui codice è disponibile ma non hanno alcuna licenza applicata sono automaticamente coperti da copyright.
  - Per software open source, si raccomanda l'uso di Apache License 2.0, MIT license, o BSD license.
  - Qualunque licenza GPL-compatibile è ritenuta idonea per il progetto del corso.

Utilizzo di path relativi in Eclipse

- Quando si lavora ad un progetto, **è bene che tutti i path siano relativi alla radice del progetto stesso**.

- In caso contrario:
  - Gli altri membri del team dovranno continuamente cambiare le impostazioni per adattarle al loro file system.
  - Il progetto non sarà sviluppabile su piattaforme diverse.
  - Si avranno continui merge conflicts se le configurazioni dell'IDE sono in tracking
  - Le librerie non saranno più accessibili automaticamente

Importazione di librerie Jar in Eclipse I

Procedura **NON** corretta

Il modo **sbagliato** di cui importare una libreria in Eclipse è: Proprietà del progetto → Java Build Path → Libraries → Add External JARs → Selezione dei JAR → OK → OK

- È possibile scegliere JAR da qualunque punto del file system.
- Vengono generati percorsi assoluti.

Importazione di librerie Jar in Eclipse II

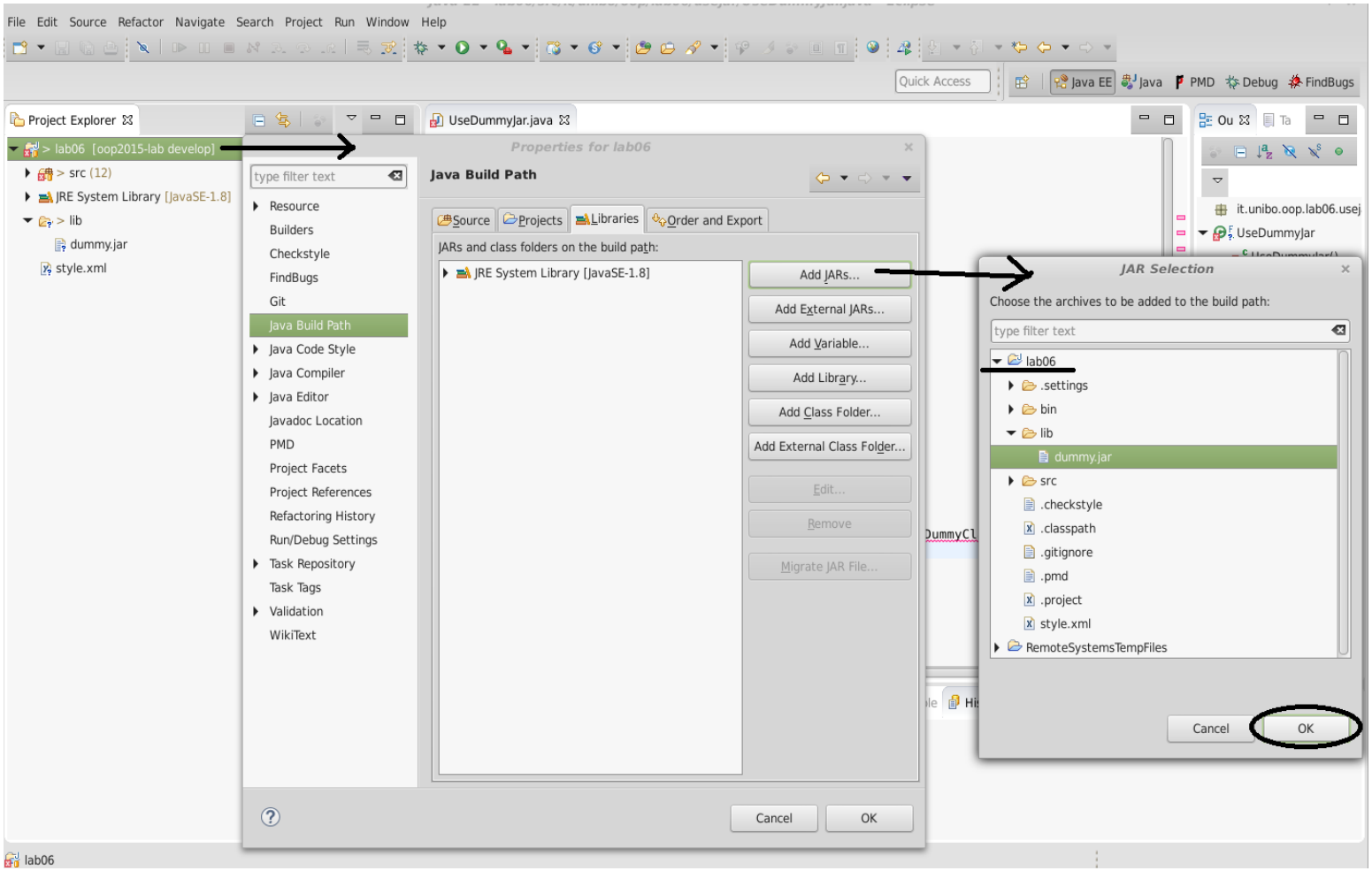
Procedura **corretta**

Il modo **corretto** di cui importare una libreria in Eclipse è: Proprietà del progetto → Java Build Path → Libraries → Add JARs → Selezione dei JAR → OK → OK

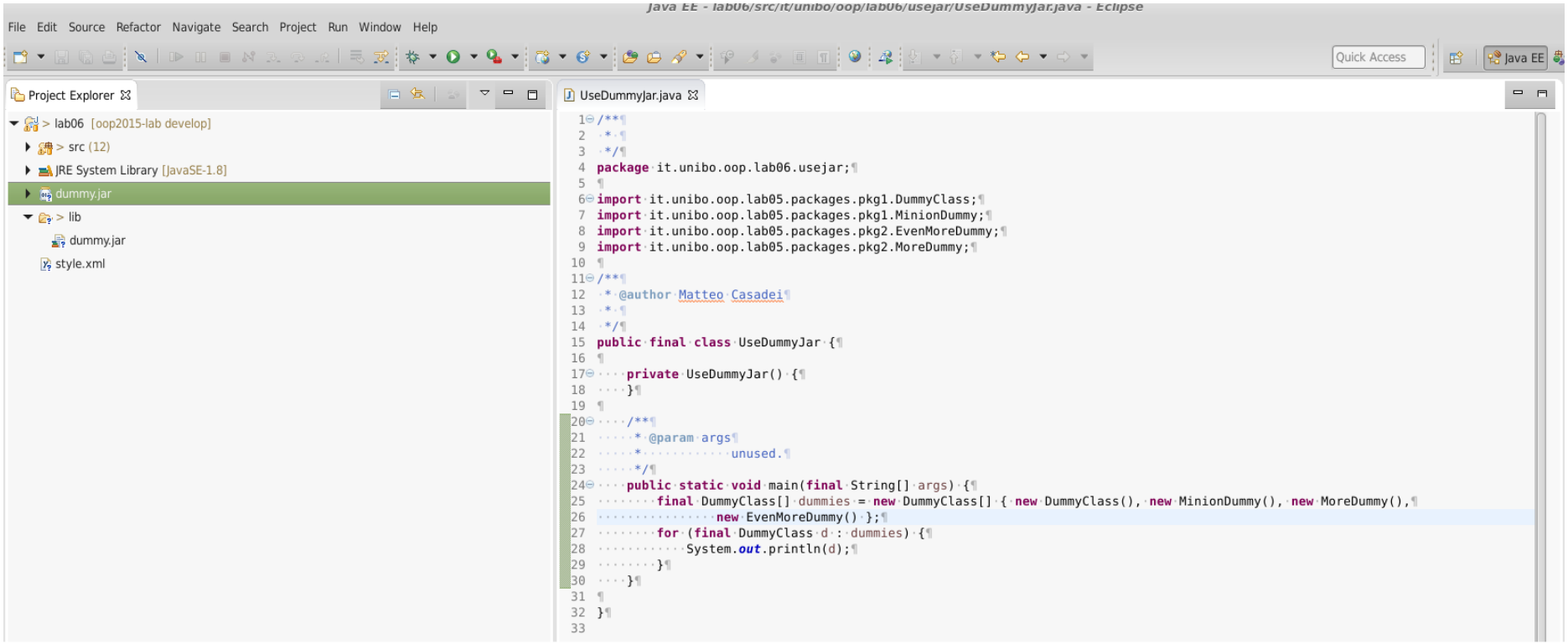
- È possibile scegliere solo JAR file che esistono in una delle cartelle del progetto (e, se il progetto è ben configurato, sono inserite in `lib`).
- Vengono generati percorsi relativi alla root del progetto.

Aggiunta di un jar al classpath di un progetto Eclipse I

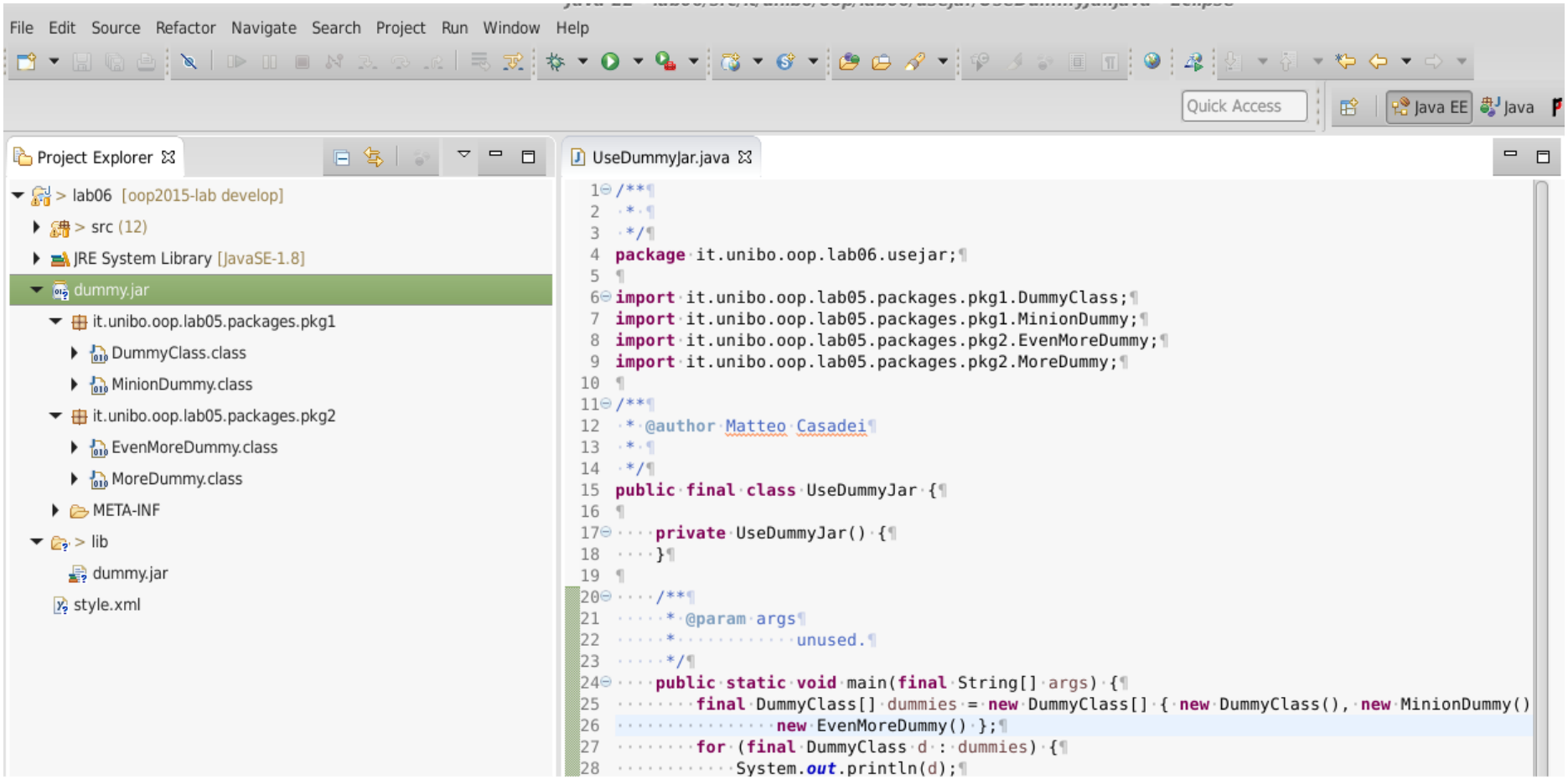
Progetto → Properties → Java build path → Libraries → Add Jar



Aggiunta di un jar al classpath di un progetto Eclipse II



Aggiunta di un jar al classpath di un progetto Eclipse III



Outline

1. Programmazione multipiattaforma
   - Accesso al file system
   - Accesso ai dettagli del sistema
   - GUI scalabili e internazionalizzazione
2. Costruzione di un ambiente di sviluppo portabile
   - Corretta configurazione di un progetto Eclipse, inclusione di librerie
   - Font ed encoding
   - Caricamento di risorse dal classpath
   - Installazione delle impostazioni per-utente
3. Ricerca/Utilizzo di librerie
4. Qualità del codice
   - Strumenti per il controllo di qualità
   - Coding style: configurazione IDE Eclipse
5. Ispezione e Profiling con VisualVM
6. Preparazione al laboratorio

Font ed encoding

Le più note piattaforme utilizzano di default encoding diversi:

- **UTF-8** — default su Linux, può essere considerato lo standard de-facto.
- **MacRoman** — default su MacOS, raramente causa artefatti se riconvertito ad altri formati.
- **ISO-8859-1** — default su Windows, può causare artefatti su quasi tutti i caratteri non ASCII se convertito a UTF-8.

Encoding per il codice sorgente

Solitamente, il codice sorgente si sviluppa utilizzando la codifica UTF-8

- Essenziale se si utilizzano caratteri non inclusi nella tabella ASCII (caratteri accentati, ad esempio).

Carattere di new line I

Unix e Windows differiscono anche per il carattere di newline

- `\n` — Su Linux e MacOS X (e su tutti i sistemi non Windows).
- `\r` — Su MacOS 9 e precedenti
- `\r\n` — Windows

Carattere di new line II

Pillola di storia

- Il "carriage return" è un'eredità dovuta alla compatibilità con MS-DOS
- Che a sua ha ereditato la cosa dalla compatibilità con CP/M-80
- A quel tempo, le stampanti erano macchine da scrivere elettromeccaniche
- `\r` riportava il rullo con la carta al margine sinistro
- Anche se queste stampanti non sono più in uso, alcuni produttori (come HP) mantengono la compatibilità su alcuni prodotti
- Ad oggi, si usa `\r` solo per realizzare animazioni su terminale (far sì che la nuova linea cancelli la precedente)

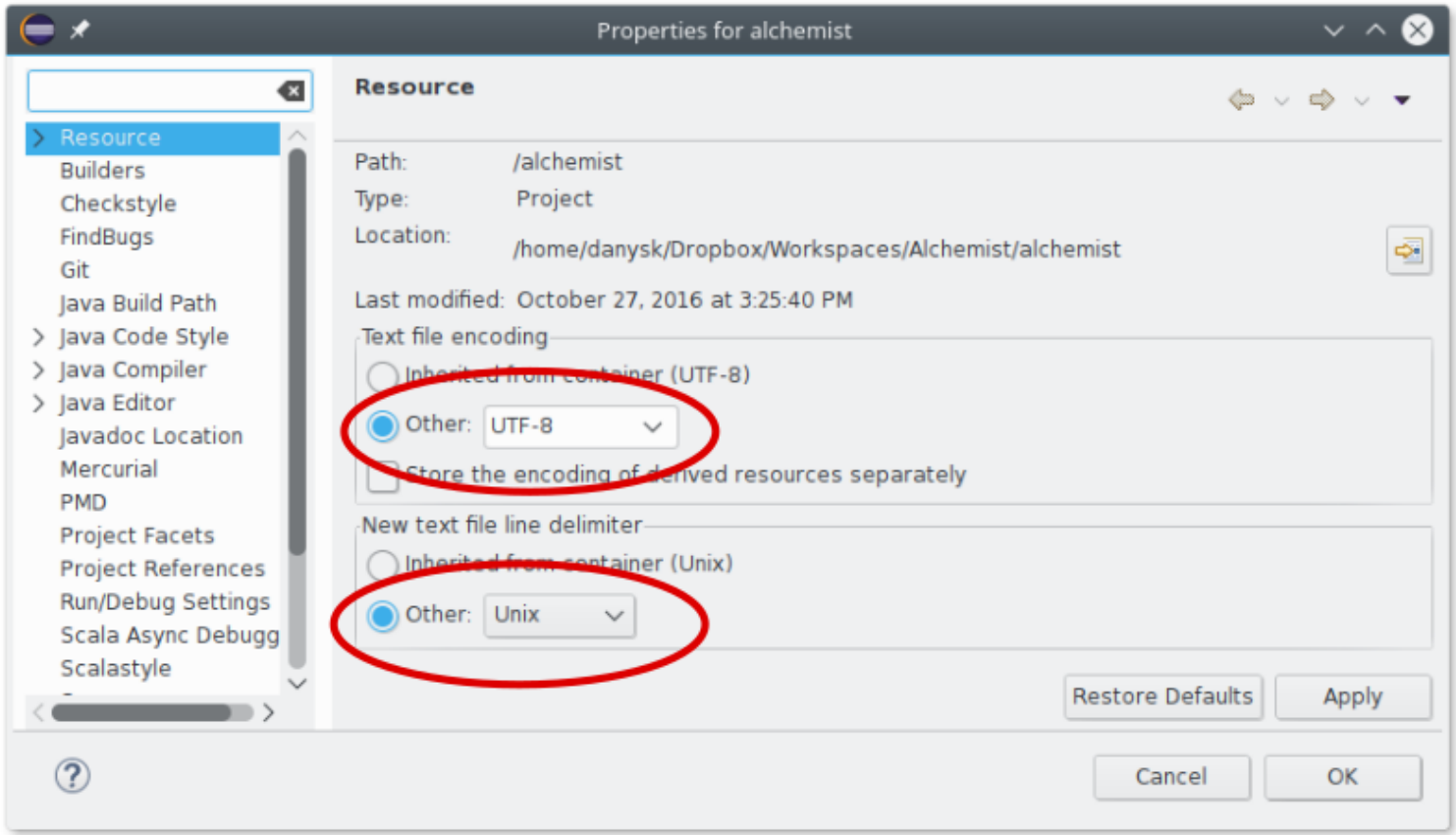
Carattere di new line III

Standardizzazione

- Entrambi i sistemi "digeriscono" entrambi i newline
- Ma se due membri del team usano impostazioni diverse, il DVCS considererà ogni file modificato come integralmente cambiato ad ogni salvataggio
  - Diff incomprensibili
  - Storia del progetto compromessa
  - Difficoltà di produrre patch e ripristinare il lavoro precedente
- Facciamo una scelta, e utilizziamo un solo formato
- La nostra scelta sarà `\n` (Unix newline)

Configurazione di encoding e newline in Eclipse

- Selezione del progetto → Click destro → Preferences → Resource
- Text file encoding → Other → UTF-8
- New text file line delimiter → Other → Unix
- Apply → OK



Outline

1. Programmazione multipiattaforma
   - Accesso al file system
   - Accesso ai dettagli del sistema
   - GUI scalabili e internazionalizzazione
2. Costruzione di un ambiente di sviluppo portabile
   - Corretta configurazione di un progetto Eclipse, inclusione di librerie
   - Font ed encoding
   - Caricamento di risorse dal classpath
   - Installazione delle impostazioni per-utente
3. Ricerca/Utilizzo di librerie
4. Qualità del codice
   - Strumenti per il controllo di qualità
   - Coding style: configurazione IDE Eclipse
5. Ispezione e Profiling con VisualVM
6. Preparazione al laboratorio

Risorse caricate dal classpath

- Abbiamo visto finora il classpath come l'insieme dei percorsi dove la virtual machine va a cercare le classi da caricare
- Esso include anche i contenuti di tutte le cartelle sorgente, che Eclipse copierà dentro la cartella `bin`, e che includerà nel JAR eseguibile assieme ai file compilati
- Come possiamo accedere a queste risorse in modo trasparente?
  - Ossia caricarle sia che si trovino sul file system, sia che si trovino nel JAR eseguibile, sia che vengano incluse in un JAR di risorse separato.

`ClassLoader.getSystemResource()`

- Java fornisce un'utilità per caricare risorse dal **classpath**
- Come abbiamo visto usando l'opzione `-cp` di `java` e `javac`, il classpath può contenere indifferentemente dei path o dei JAR (o anche degli zip)

Risorse caricate dal classpath – Esempi

Caricamento di File

```
1 final InputStream in = ClassLoader.getSystemResourceAsStream
      ("/settings/settings");
2 final BufferedReader br = new BufferedReader(new
      InputStreamReader(in));
3 final String line = br.readLine();
4 in.close();
```

Caricamento di Immagini

```
1 final URL imgURL = ClassLoader.getSystemResource("/images/
      gandalf.jpg");
2 final ImageIcon icon = new ImageIcon(imgURL);
3 final JLabel lab1 = new JLabel(icon);
```

Outline

1. Programmazione multipiattaforma
   - Accesso al file system
   - Accesso ai dettagli del sistema
   - GUI scalabili e internazionalizzazione
2. Costruzione di un ambiente di sviluppo portabile
   - Corretta configurazione di un progetto Eclipse, inclusione di librerie
   - Font ed encoding
   - Caricamento di risorse dal classpath
   - Installazione delle impostazioni per-utente
3. Ricerca/Utilizzo di librerie
4. Qualità del codice
   - Strumenti per il controllo di qualità
   - Coding style: configurazione IDE Eclipse
5. Ispezione e Profiling con VisualVM
6. Preparazione al laboratorio

Installazione delle impostazioni

Motivazione

Spesso un software ha necessità di caricare al primo avvio delle impostazioni di default, quindi lasciare l'utente libero di modificarle e, se avviato successivamente caricare quelle scelte dall'utente. In caso di sistema multiutente, le impostazioni saranno diverse per ciascuno.

Strategia

- Si sceglie una cartella nella home folder dell'utente dove salvare le impostazioni.
  - È norma consolidata creare una cartella `.nomeprogramma`.
- Al primo avvio, si verifica se tale cartella esista e se contenga i file di configurazione previsti.
  - Se non è presente, o se non sono presenti e leggibili alcuni i file, si procede a caricare nella cartella di destinazione i file di default dal jar usando `getResource()`.

Outline

1. Programmazione multipiattaforma
   - Accesso al file system
   - Accesso ai dettagli del sistema
   - GUI scalabili e internazionalizzazione
2. Costruzione di un ambiente di sviluppo portabile
   - Corretta configurazione di un progetto Eclipse, inclusione di librerie
   - Font ed encoding
   - Caricamento di risorse dal classpath
   - Installazione delle impostazioni per-utente
3. Ricerca/Utilizzo di librerie
4. Qualità del codice
   - Strumenti per il controllo di qualità
   - Coding style: configurazione IDE Eclipse
5. Ispezione e Profiling con VisualVM
6. Preparazione al laboratorio

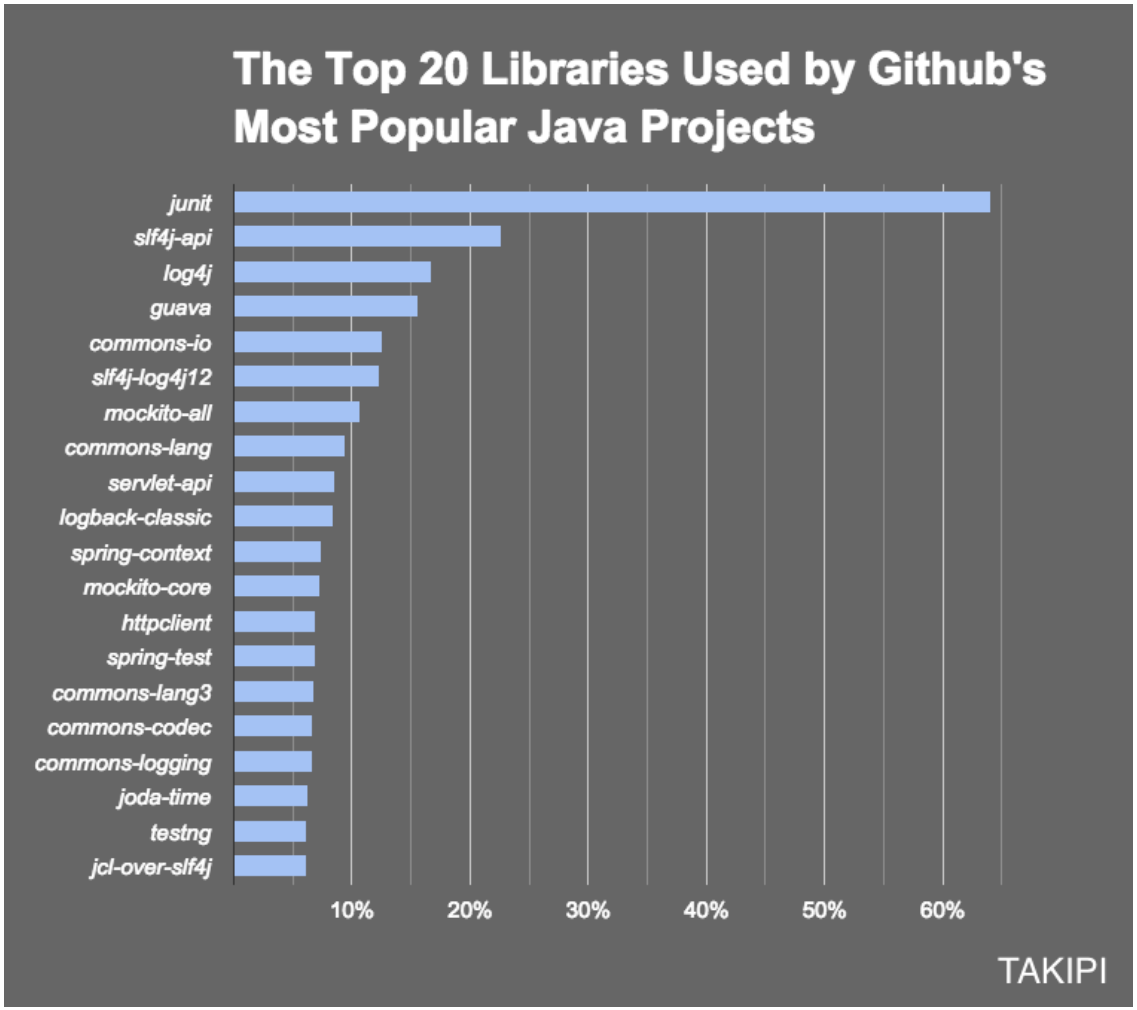
Uso di Librerie

- Durante lo sviluppo di un software, spesso emerge la necessità di realizzare componenti a supporto della propria applicazione, utili per incapsulare un determinato comportamento/funzionalità ma non cruciali per la logica applicativa.
- Nella maggior parte dei casi una tale libreria/componente è già stato sviluppato da qualcun altro!
- Risulta opportuno avvalersi di librerie già disponibili e testate...
  - ...dopo aver letto attentamente la documentazione e averne compreso a fondo il comportamento!

Librerie particolarmente utili

- **Google Guava** (`https://github.com/google/guava`)
  - Il progetto Guava di Google raccoglie diverse librerie core sviluppate da Google che possono essere utilizzate per lo sviluppo di applicazioni.
  - Ad esempio, sono disponibili librerie per Collections Management, Concurrency, I/O, String Processing,...
- **Apache Commons** (`https://commons.apache.org`)
  - Estensioni al linguaggio (Commons Lang3)
  - Libreria matematica estesa (Commons Math3)
  - Accesso semplificato all'I/O (Commons IO)
  - Costruzione semi automatica di una command line (Commons CLI)
  - Encoding e crittazione (Commons Codec), compressione (Commons Compress)
- **Static Logger Facade for Java (SLF4J)** (`http://www.slf4j.org`)
  - Backend-independent logging (addio `println`)

Top 20 Java Library



blog.takipi.com/we-analyzed-60678-libraries-on-github-here-are-the-top-100

Awesome Java

Esiste una lista, costantemente manutenuta, che elenca le più comuni, diffuse e stabili librerie per una pletora di usi:

`https://bit.ly/awesome-java`

U-SA-TE-LE

Usare librerie e non reinventare la ruota è IMPORTANTE e valutato positivamente.

Attenzione però a scegliere le librerie dopo aver fatto il **modello del dominio** dell'applicazione: **PRIMA** si studia il problema, **DOPO** si implementa una soluzione: siete aspiranti ingegneri, cercate di lavorare sempre top-down quando possibile, non partite dalla libreria per costruirci sopra un software, ma partite dai requisiti e - se utile - sfruttate le librerie per soddisfarli.

The Maven Central Repository

- Costituisce una delle più ampie collezioni di librerie e componenti Java open-source.
  - È il repository di default per Apache Maven (noto tool per il project management).
  - Bintray JCenter è un superset di Maven Central.
- Rappresenta uno dei modi più rapidi per accedere a librerie sviluppate da altri sviluppatori e distribuire le proprie.
- Consente di ricercare e scaricare pressoché qualunque libreria a supporto utile nelle proprie applicazioni open-source.
- `https://search.maven.org/`

Outline

1. Programmazione multipiattaforma
   - Accesso al file system
   - Accesso ai dettagli del sistema
   - GUI scalabili e internazionalizzazione
2. Costruzione di un ambiente di sviluppo portabile
   - Corretta configurazione di un progetto Eclipse, inclusione di librerie
   - Font ed encoding
   - Caricamento di risorse dal classpath
   - Installazione delle impostazioni per-utente
3. Ricerca/Utilizzo di librerie
4. Qualità del codice
   - Strumenti per il controllo di qualità
   - Coding style: configurazione IDE Eclipse
5. Ispezione e Profiling con VisualVM
6. Preparazione al laboratorio

Outline

1. Programmazione multipiattaforma
   - Accesso al file system
   - Accesso ai dettagli del sistema
   - GUI scalabili e internazionalizzazione
2. Costruzione di un ambiente di sviluppo portabile
   - Corretta configurazione di un progetto Eclipse, inclusione di librerie
   - Font ed encoding
   - Caricamento di risorse dal classpath
   - Installazione delle impostazioni per-utente
3. Ricerca/Utilizzo di librerie
4. Qualità del codice
   - Strumenti per il controllo di qualità
   - Coding style: configurazione IDE Eclipse
5. Ispezione e Profiling con VisualVM
6. Preparazione al laboratorio

Analisi di qualità del codice sorgente

Definizione

Software in grado di analizzare il codice sorgente per individuare:

- Potenziali bug, magari dovuti a distrazione
- Possibili miglioramenti, ottimizzazioni o pratiche difformi da quelle consigliate
- Codice duplicato, segnale di una progettazione discutibile
- Stile non conforme

Uso

L'analisi automatica del proprio codice garantisce sempre un elevata qualità del codice, aiuta ad apprendere il modo corretto di scrivere, riduce il costo di manutenzione e garantisce uniformità fra le parti sviluppate da persone diverse.

Code checking I

I software che vedremo sono eseguibili in due modalità [1]:

- Stand-alone: il software viene eseguito e genera un report
- Come plug-in: il software viene integrato con l'IDE (nel nostro caso Eclipse), e segnala i problemi sotto forma di warning

Noi ci concentreremo nell'imparare la seconda modalità .

[1] In realtà cominciano a prender piede anche servizi web come Codacy, ma non li tratteremo in questo corso...

Code checkers in Eclipse I

Installazione

Per l'installazione, si seguano le istruzioni fornite sulla pagina dei installazione del software del corso.

Configurazione globale

La configurazione globale viene applicata a **tutti** i progetti del workspace, a meno che non sia sovrascritta localmente (non sempre è possibile).

- Veloce da configurare: si configura una volta, e i settaggi sono applicati su tutti i progetti
- Poco portabile: se inviamo il progetto ad un collaboratore, questi settaggi **non** saranno inviati: c'è il rischio di inconsistenze
- Ottimo da usare quando si hanno molti progetti nel workspace con la stessa configurazione, e si sviluppa da soli.

Code checkers in Eclipse II

Configurazione locale

La configurazione locale va messa a punto singolarmente per ciascun progetto. Bisogna verificare che quella globale non la stia sovrascrivendo.

- Alta flessibilità: si possono specificare regole diverse per progetti diversi
- Alta portabilità: se si invia il progetto ad un collaboratore, queste impostazioni vengono mantenute
- Ottimo da usare quando si hanno pochi progetti, o ai progetti serve una configurazione diversa, oppure quando si lavora in modo collaborativo

Noi faremo uso della configurazione **locale**
Voi farete uso della configurazione locale nell'**elaborato finale**

SpotBugs (ex FindBugs)

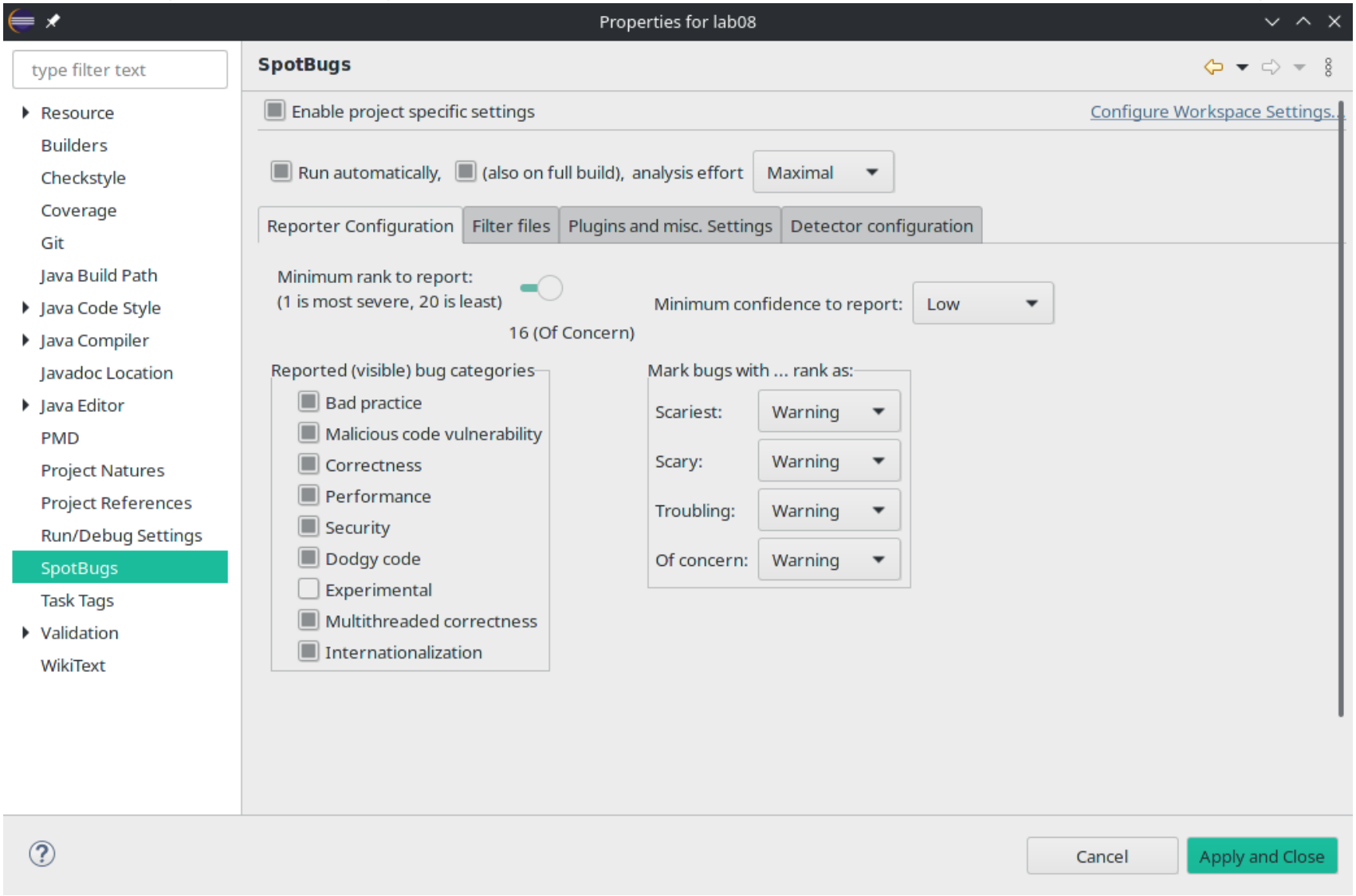
Cos'è

SpotBugs scansiona il bytecode generato dal compilatore, e dalla sua analisi cerca di scoprire potenziali bug nel sorgente, ad esempio:

- Uguaglianza esatta fra `float` o `double`
- Utilizzo di == invece di `equals()`
- Mancata annotazione di annotazioni usate a runtime via reflection
- Uso errato di meccanismi di sincronizzazione
- Vulnerabilità di sicurezza
- Tanti altri! Si veda: `http://findbugs.sourceforge.net/bugDescriptions.html`

SpotBugs — Configurazione del progetto

In SpotBugs, la configurazione locale **è più prioritaria** di quella globale.



PMD e CPD

Cos'è

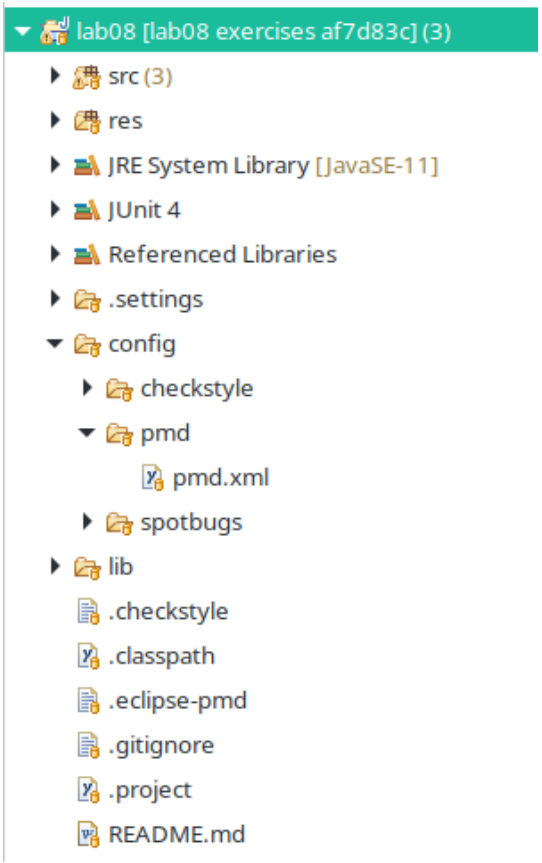
PMD si occupa di trovare imperfezioni nel codice:

- Campi pubblici, protetti o default
- Mancato uso di final
- Singular fields
- Usa CPD per verificare se vi siano blocchi di codice copincollati
- Tanti altri! Si veda:
  `https://pmd.github.io/latest/pmd_rules_java.html`

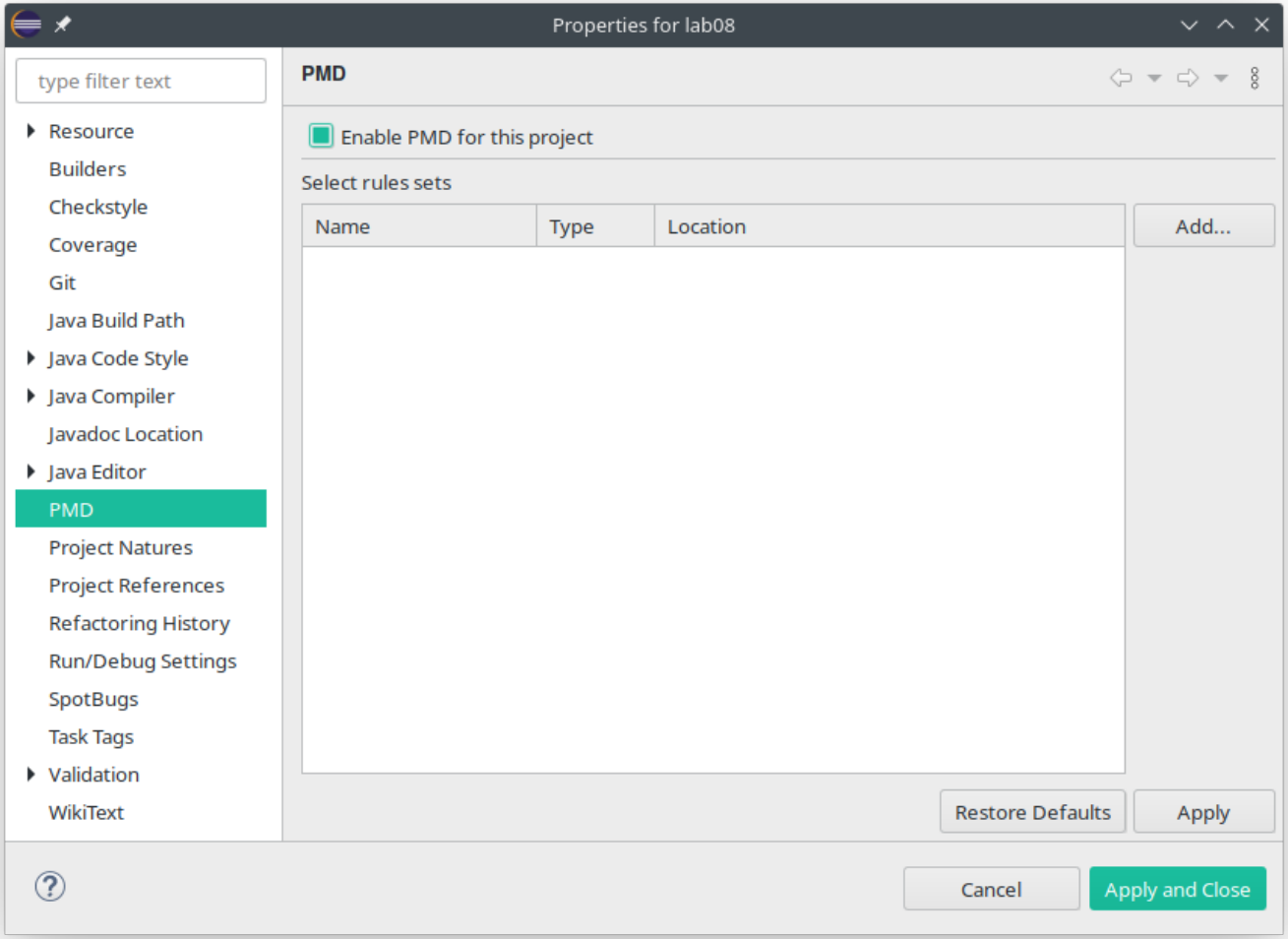
PMD — Configurazione del progetto I

ATTENZIONE

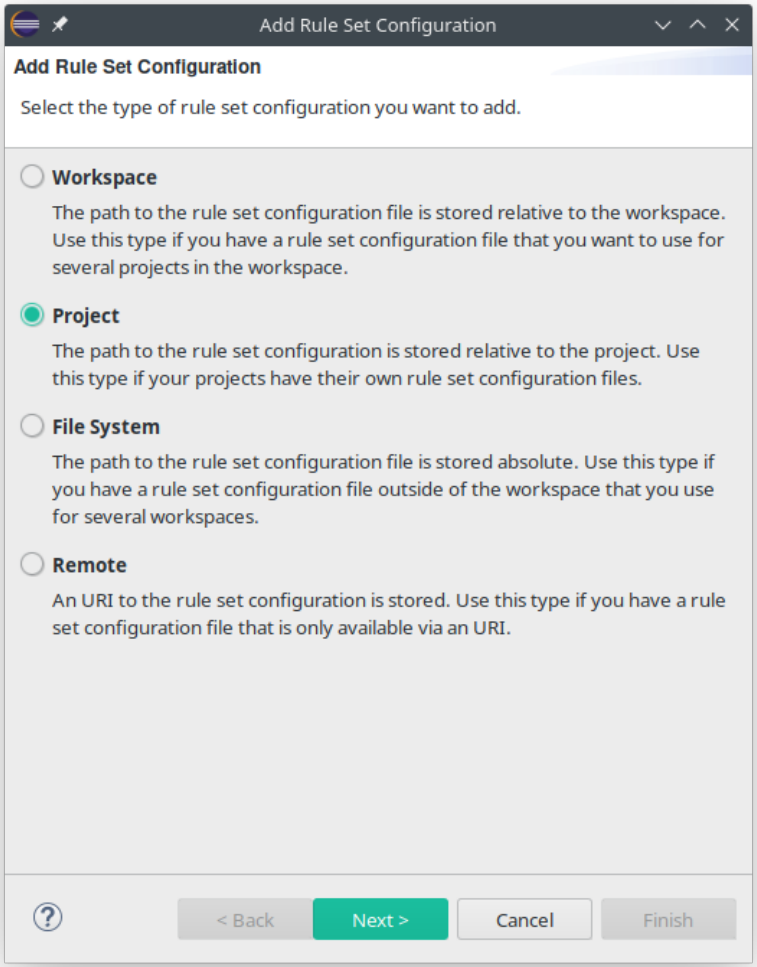
- Prima di attivare la configurazione locale, **assicurarsi** che il file `pmd.xml` sia stato copiato all'interno della cartella del progetto!



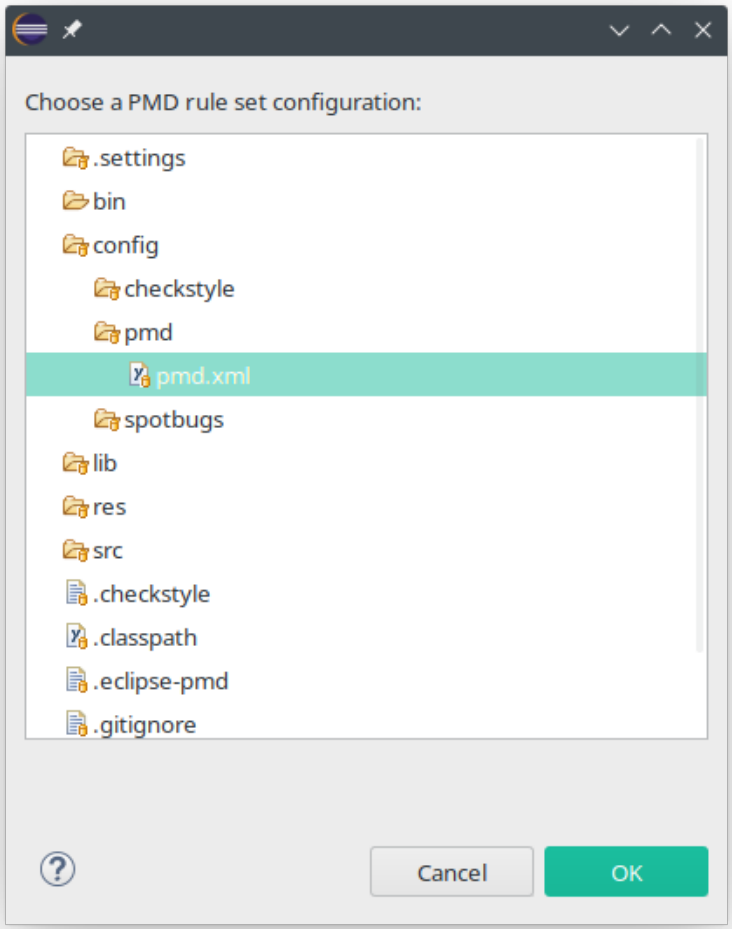
PMD — Configurazione del progetto II



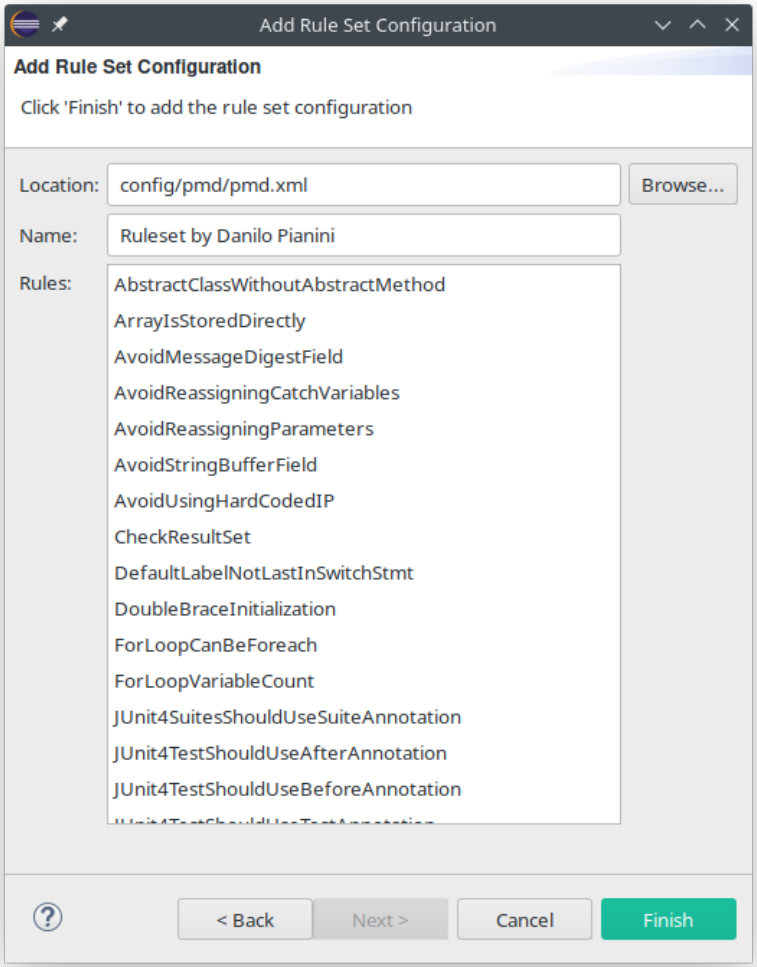
PMD — Configurazione del progetto III



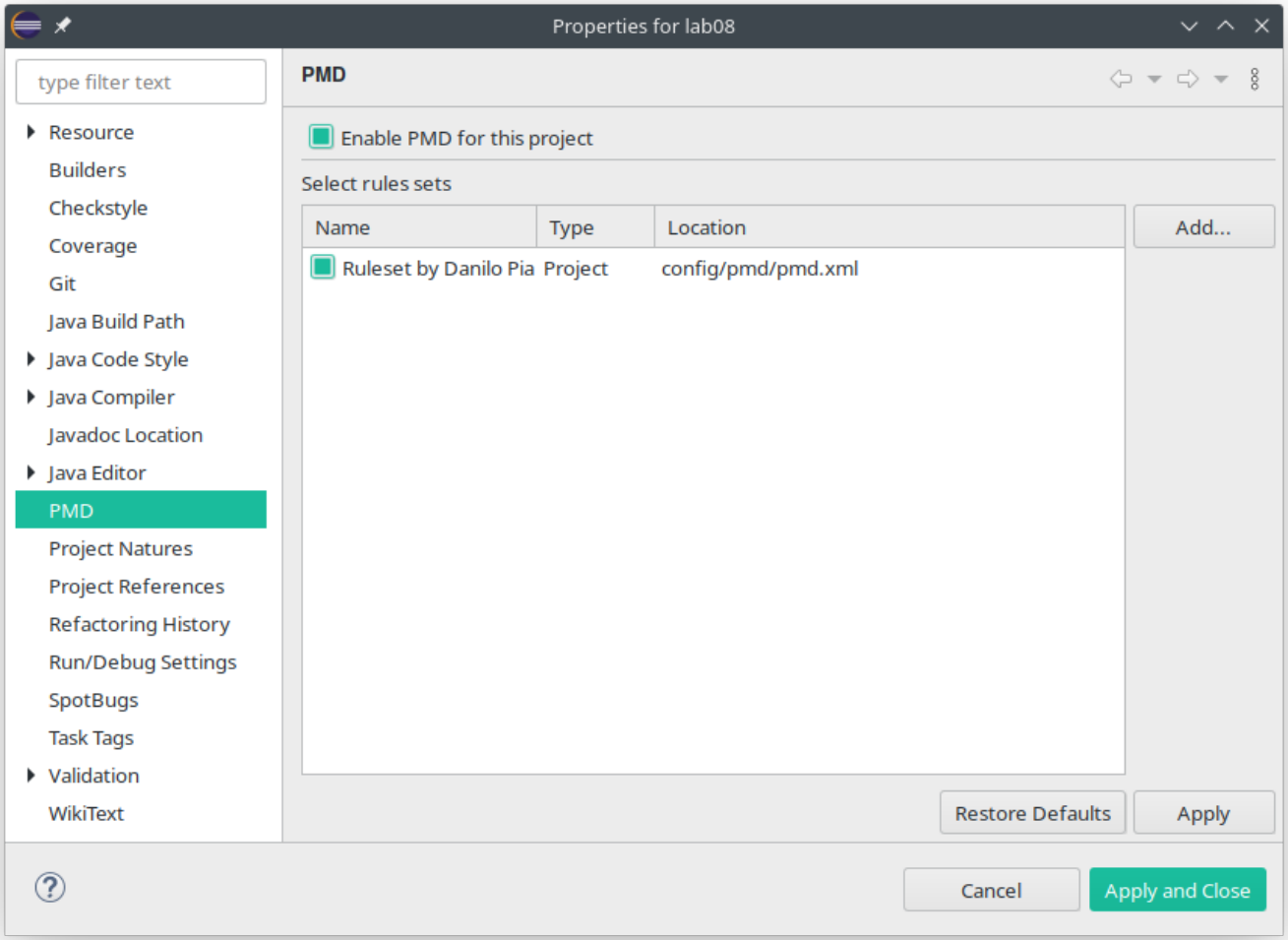
PMD — Configurazione del progetto IV



PMD — Configurazione del progetto V



PMD — Configurazione del progetto VI



Checkstyle

Cos'è

Checkstyle si occupa di trovare errori di stile:

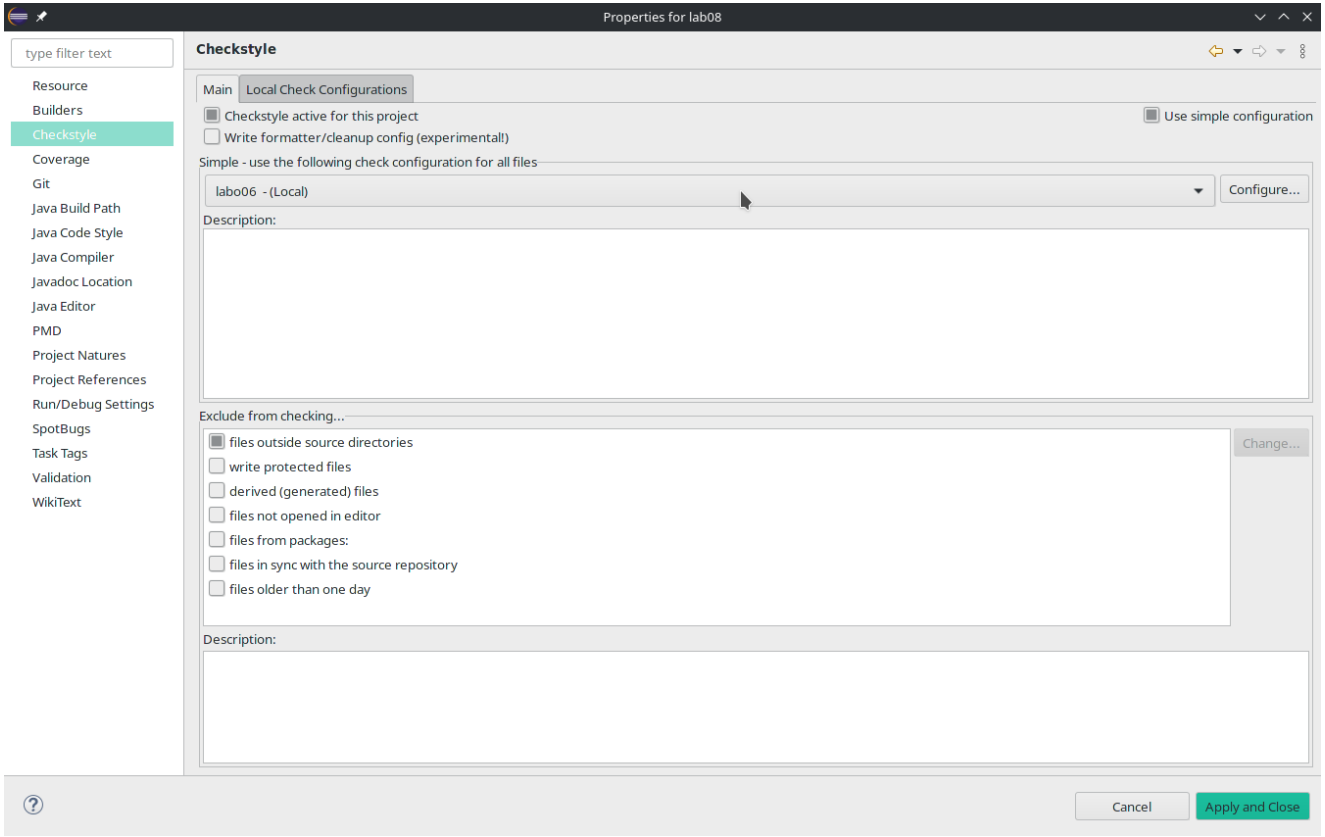
- Mancanza di commento Javadoc
- Spaziature non corrette
- Parentesi assenti
- Magic numbers
- Altro: `http://checkstyle.sourceforge.net/checks.html`

Checkstyle — Configurazione

La configurazione di default di Checkstyle è eccessivamente pedante. Vi forniremo noi un file di configurazione idoneo:

1. Dalle proprietà di progetto, si clicki sul menu Checkstyle
2. Si selezioni "Local Check Configurations"
3. Si scelga "New"
4. Si scelga "Project Relative Configuration" (mai usare path assoluti!)
5. Utilizzando "Browse..." si punti al file di configurazione, che deve esser copiato nel progetto
6. Si dia un nome e si prema "OK"
7. Si vada su "Main"
8. Si selezioni dal menu la nuova Check configuration
9. Si attivi "Checkstyle active for this project" e "Use simple configuration"
10. "OK"

Checkstyle — Configurazione per-progetto



Outline

1. Programmazione multipiattaforma
   - Accesso al file system
   - Accesso ai dettagli del sistema
   - GUI scalabili e internazionalizzazione
2. Costruzione di un ambiente di sviluppo portabile
   - Corretta configurazione di un progetto Eclipse, inclusione di librerie
   - Font ed encoding
   - Caricamento di risorse dal classpath
   - Installazione delle impostazioni per-utente
3. Ricerca/Utilizzo di librerie
4. Qualità del codice
   - Strumenti per il controllo di qualità
   - Coding style: configurazione IDE Eclipse
5. Ispezione e Profiling con VisualVM
6. Preparazione al laboratorio

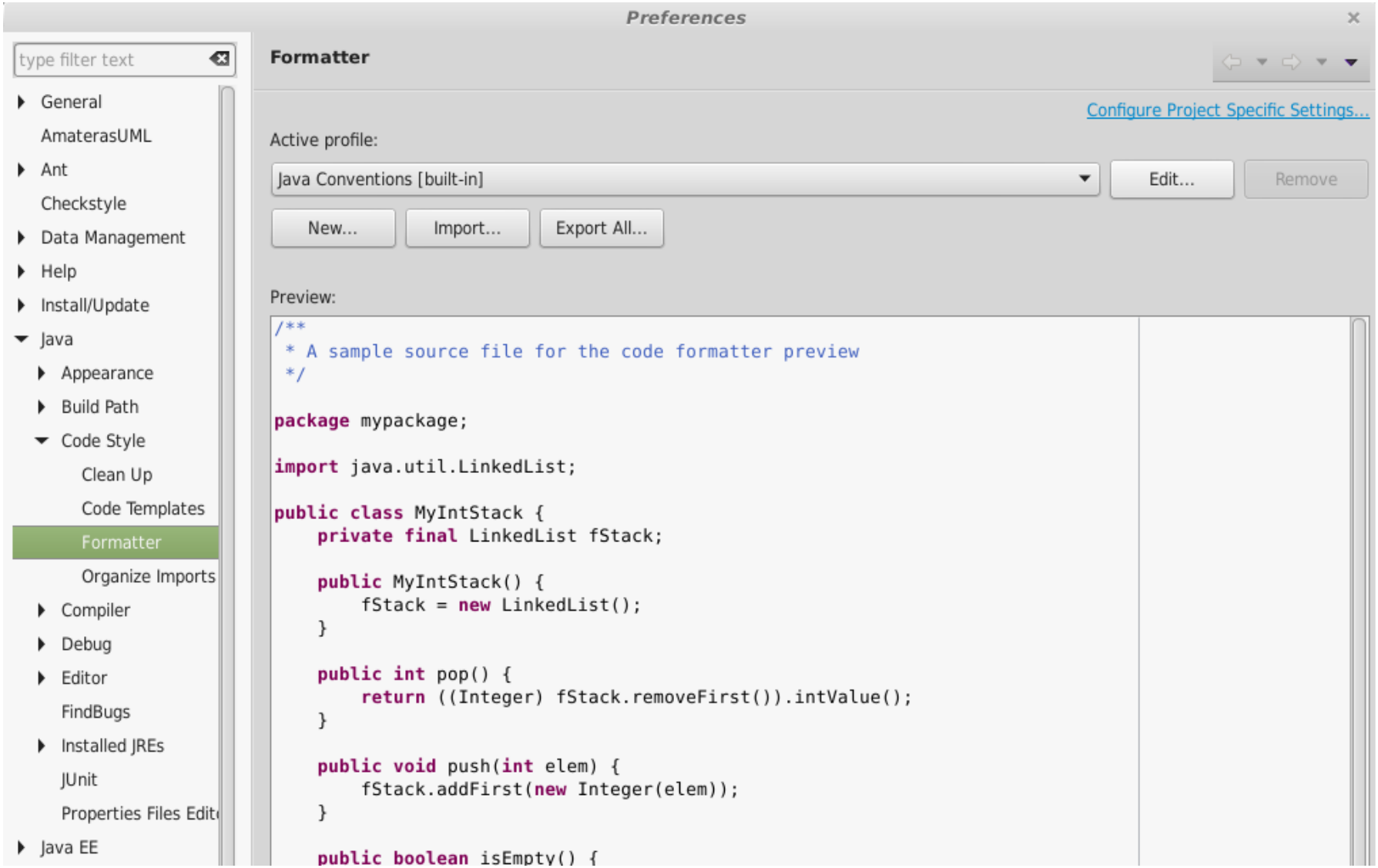
Configurare l'IDE per una corretta formattazione del codice

Checkstyle è in grado di supportarci nel verificare l'utilizzo di un corretto stile di codifica. Adattando le "Java Code Convention":

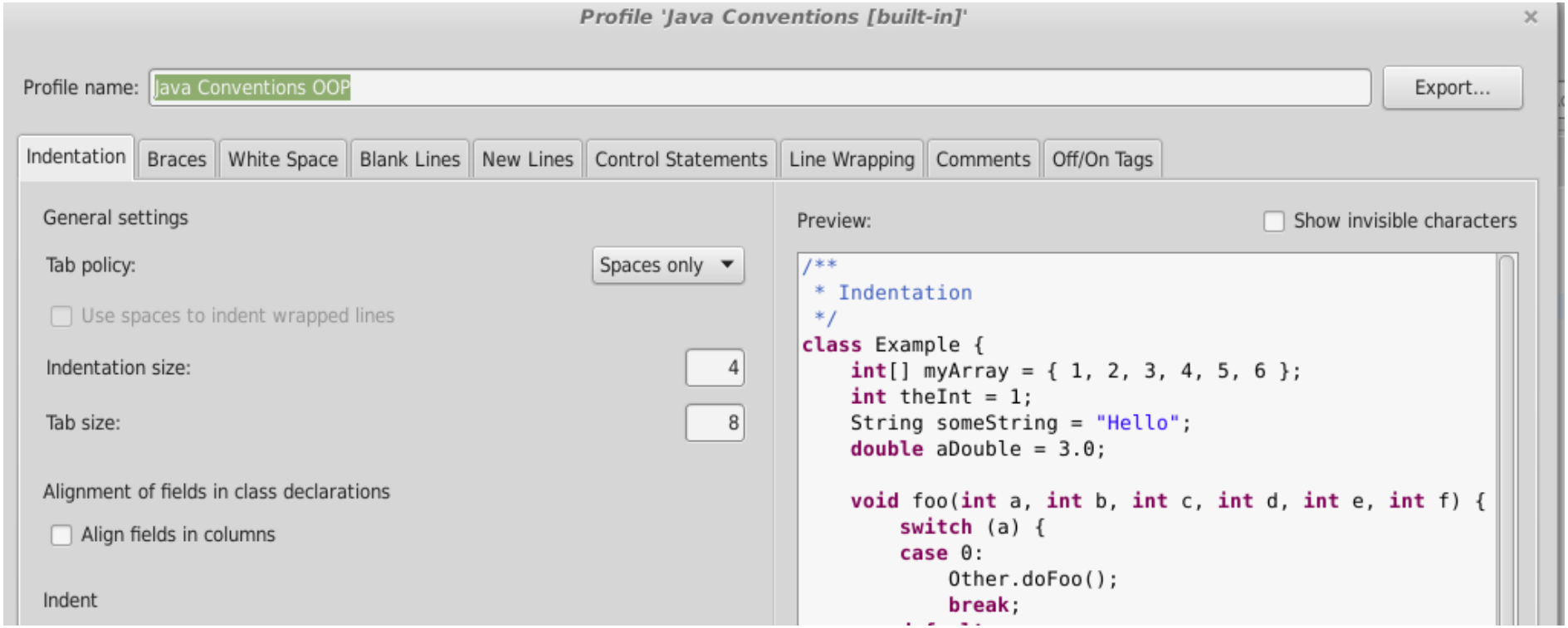
- 4 spazi per l'indentazione
  - Nell'originale è consentito scegliere fra fra i due ed i quattro
- Nessun carattere tab
  - Nell'originale è consentito usarli
  - Ma noi vogliamo indentazione consistente, quindi o spazi, oppure tab
  - Essendo in JCC un tab equivalente ad 8 spazi, diventa troppo largo per una buona indentazione
- Linee lunghe al massimo 120 o 130 caratteri
  - Nell'originale sono 80 per motivi storici (le schede perforate IBM avevano 80 colonne)
  - Noi abbiamo i monitor, e anche i nostri colleghi

Ora, però, dobbiamo configurare l'editor perché ci supporti nella scrittura di codice con lo stile corretto: come fare?

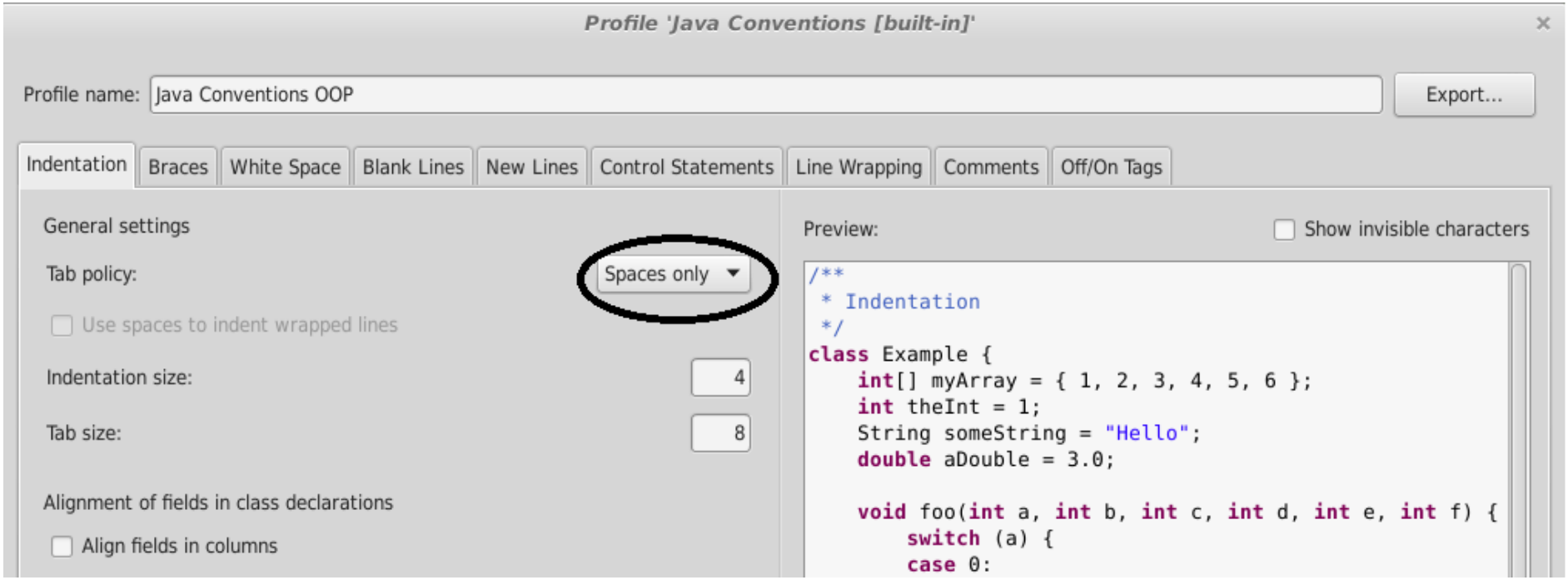
Configurazione del formattatore di Eclipse I



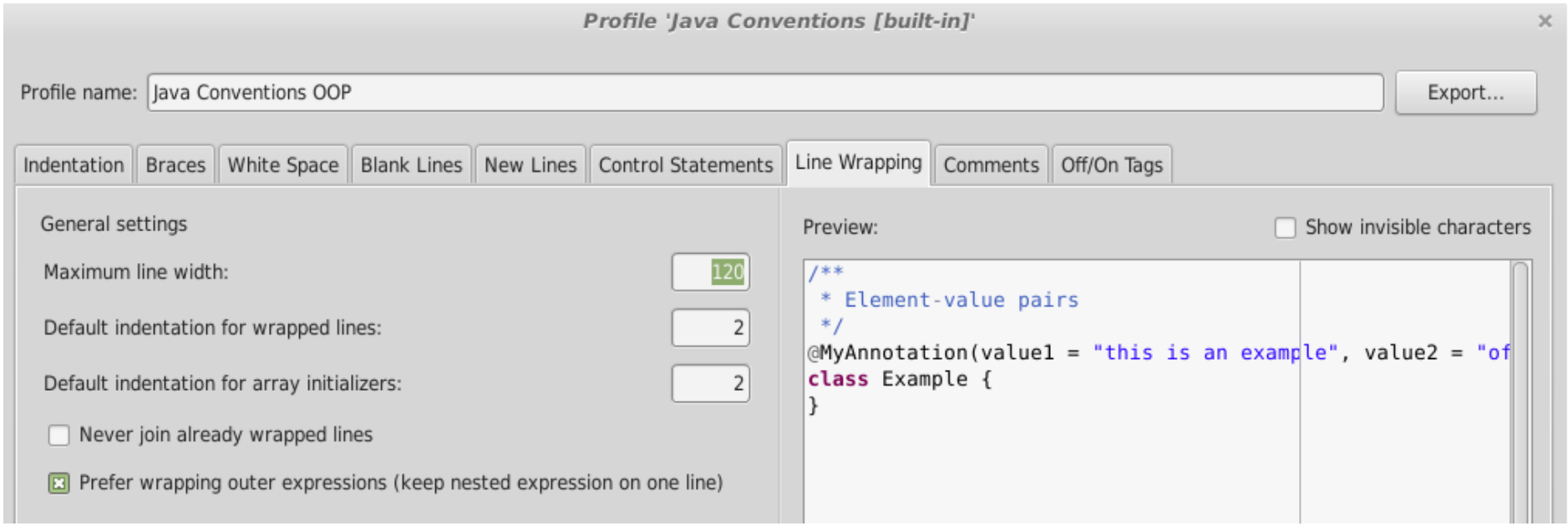
Configurazione del formattatore di Eclipse II



Configurazione del formattatore di Eclipse III



Configurazione del formattatore di Eclipse IV



Outline

1. Programmazione multipiattaforma
   - Accesso al file system
   - Accesso ai dettagli del sistema
   - GUI scalabili e internazionalizzazione
2. Costruzione di un ambiente di sviluppo portabile
   - Corretta configurazione di un progetto Eclipse, inclusione di librerie
   - Font ed encoding
   - Caricamento di risorse dal classpath
   - Installazione delle impostazioni per-utente
3. Ricerca/Utilizzo di librerie
4. Qualità del codice
   - Strumenti per il controllo di qualità
   - Coding style: configurazione IDE Eclipse
5. Ispezione e Profiling con VisualVM
6. Preparazione al laboratorio

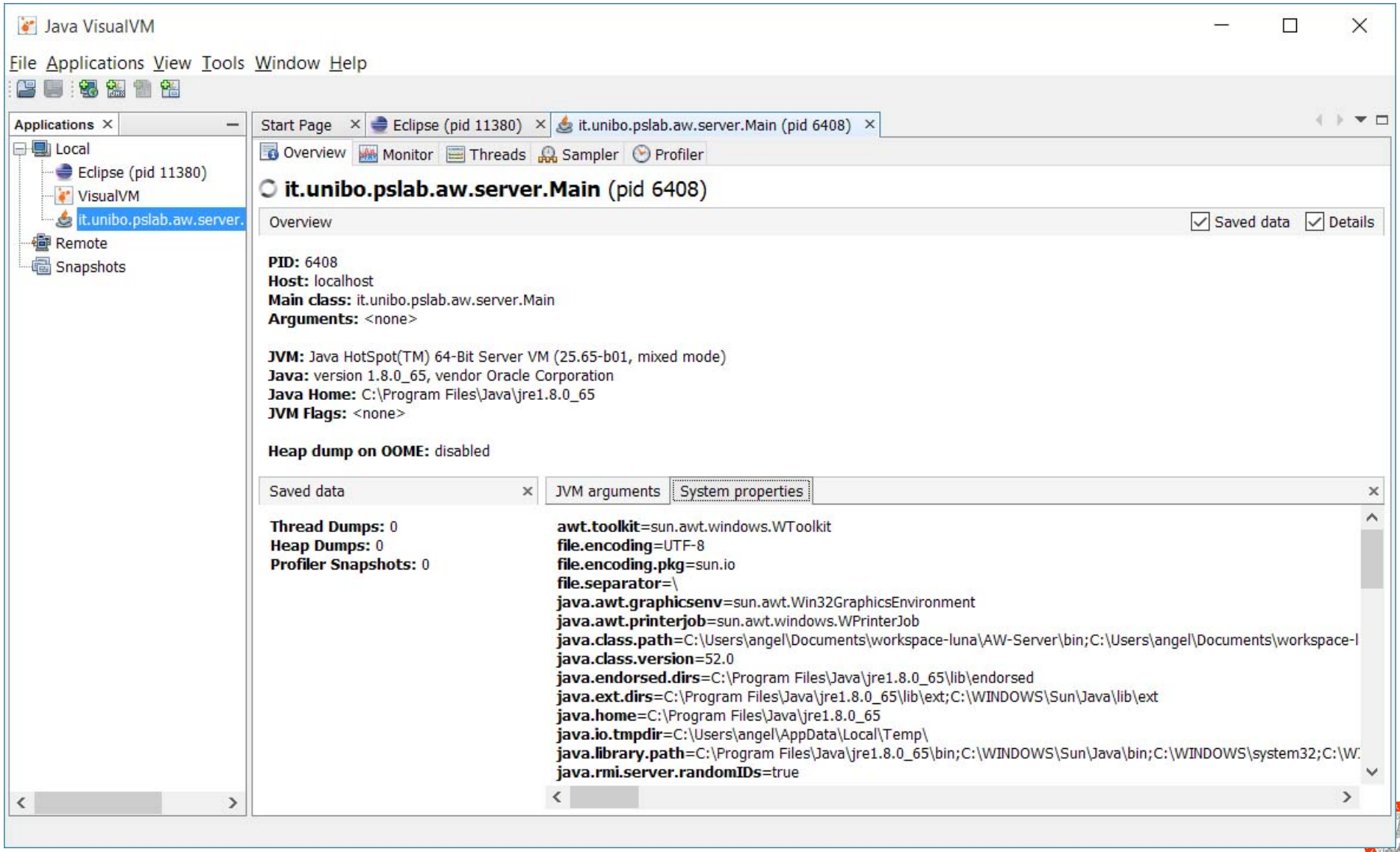
Profiling di applicazioni Java

- Quando si sviluppa un'applicazione complessa, soprattutto se basata su meccanismi di concorrenza, è opportuno analizzarne il comportamento e monitorarne le performance.
  - Spesso il monitoraggio delle performance è essenziale per identificare eventuali problematiche o capire l'origine di problemi che possono essere sorti.
- Tra gli strumenti che possono essere utilizzati per monitorare l'esecuzione di applicazioni che sono eseguite sulla JVM, due sono distribuiti unitamente al Java Development Kit (JDK).
  - **JConsole**, lo storico (e scarno) tool per il profiling di applicazioni Java.
  - **JVisualVM**, il più recente ed evoluto tool utilizzabile per monitorare l'evoluzione e le performance di applicazioni in esecuzione sulla JVM.

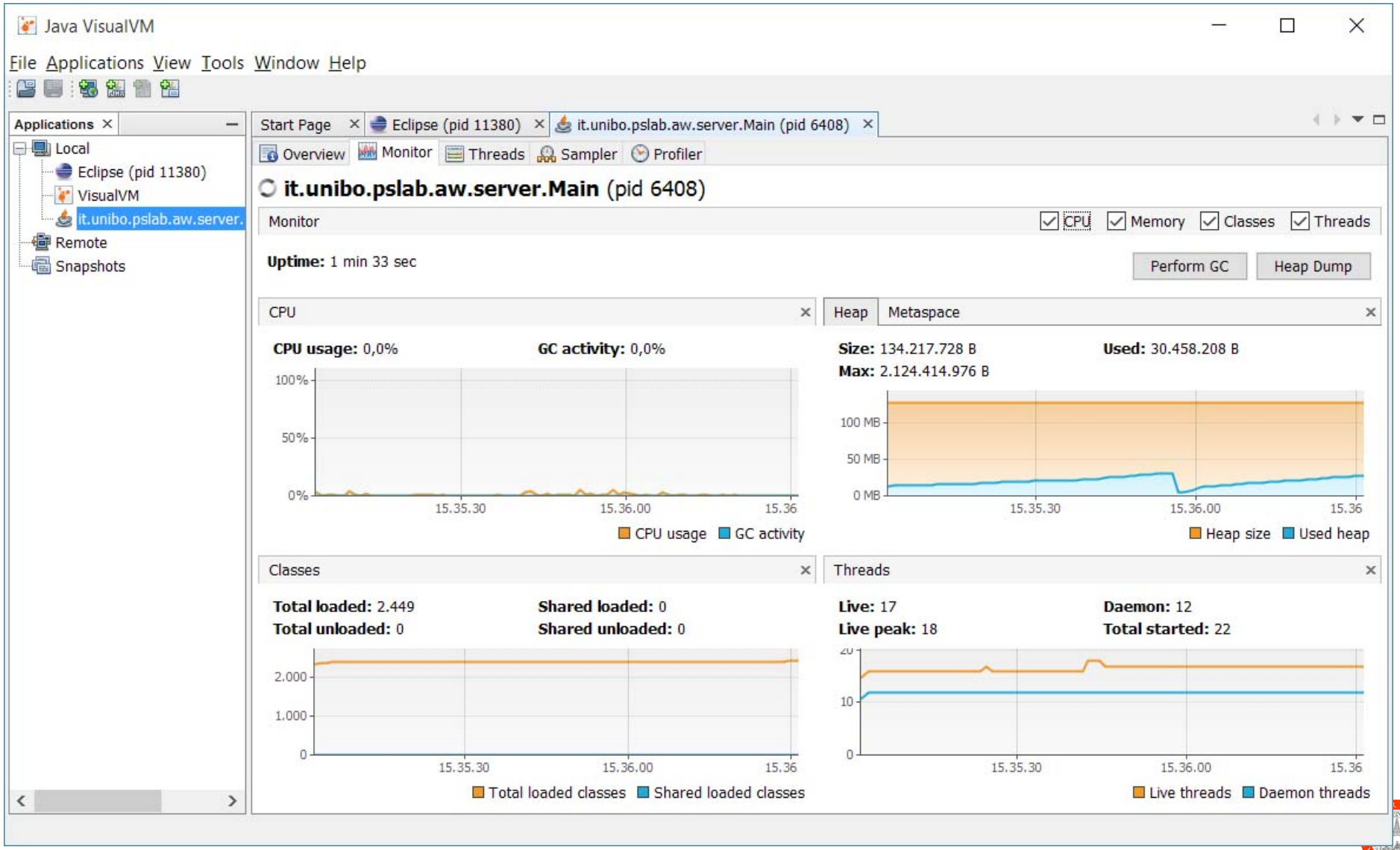
VisualVM

- Si tratta di un profiler per applicazioni Java che consente di misurarne ed analizzarne le performance.
- Interagisce con la JVM per fornire informazioni circa le performance e il consumo di risorse di applicazioni in esecuzione.
- Distribuito internamente al JDK dalla versione 5.0
- Consente di monitorare:
  - la percentuale di CPU utilizzata da singoli metodi, thread, ...
  - per quanto tempo un thread si trova nello stato di running oppure in stato di blocco o di idle.
  - ...
- Richiamabile attraverso il comando `jvisualvm` o `visualvm` (dipendentemente dalla distribuzione Java).

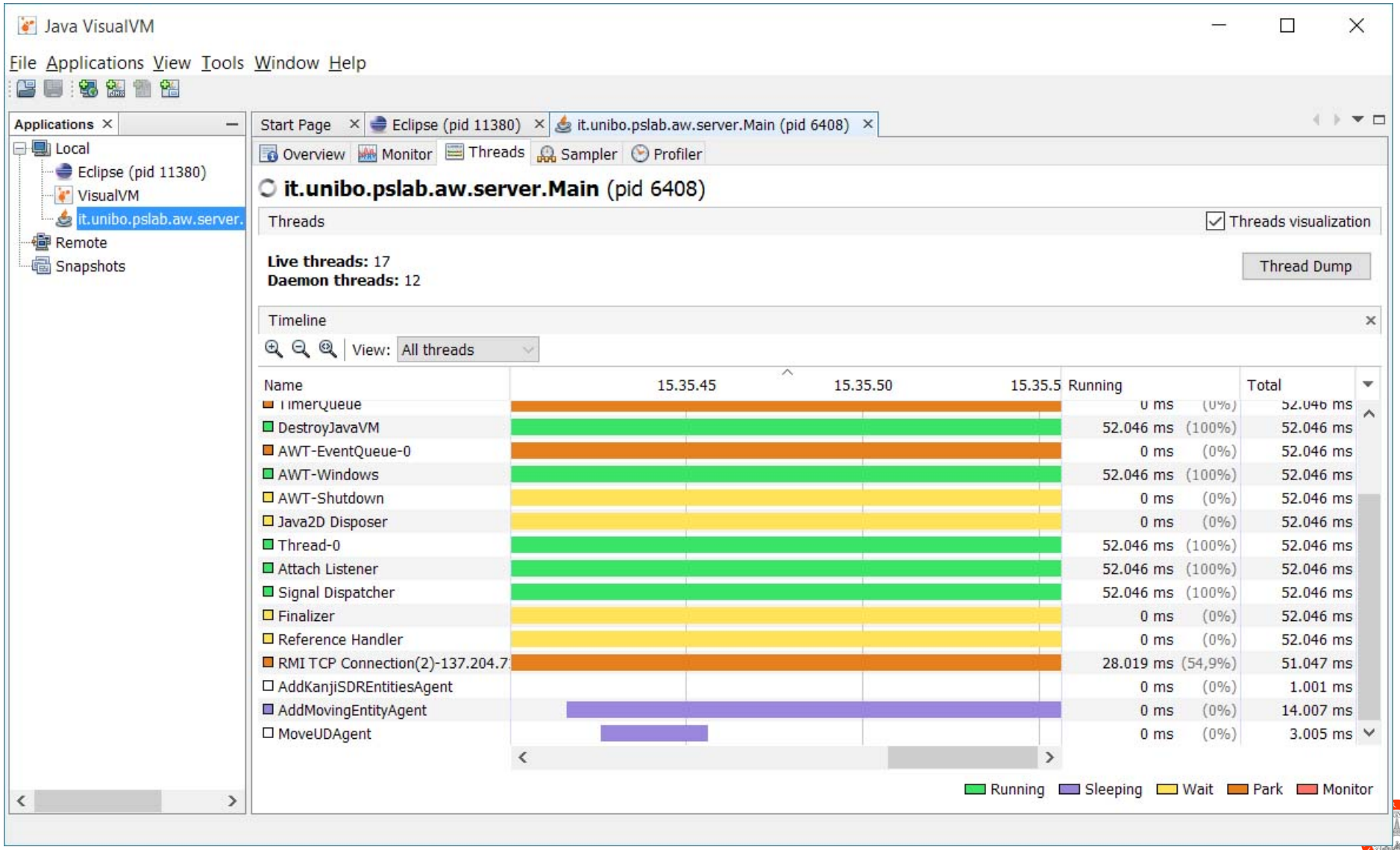
VisualVM - Overview



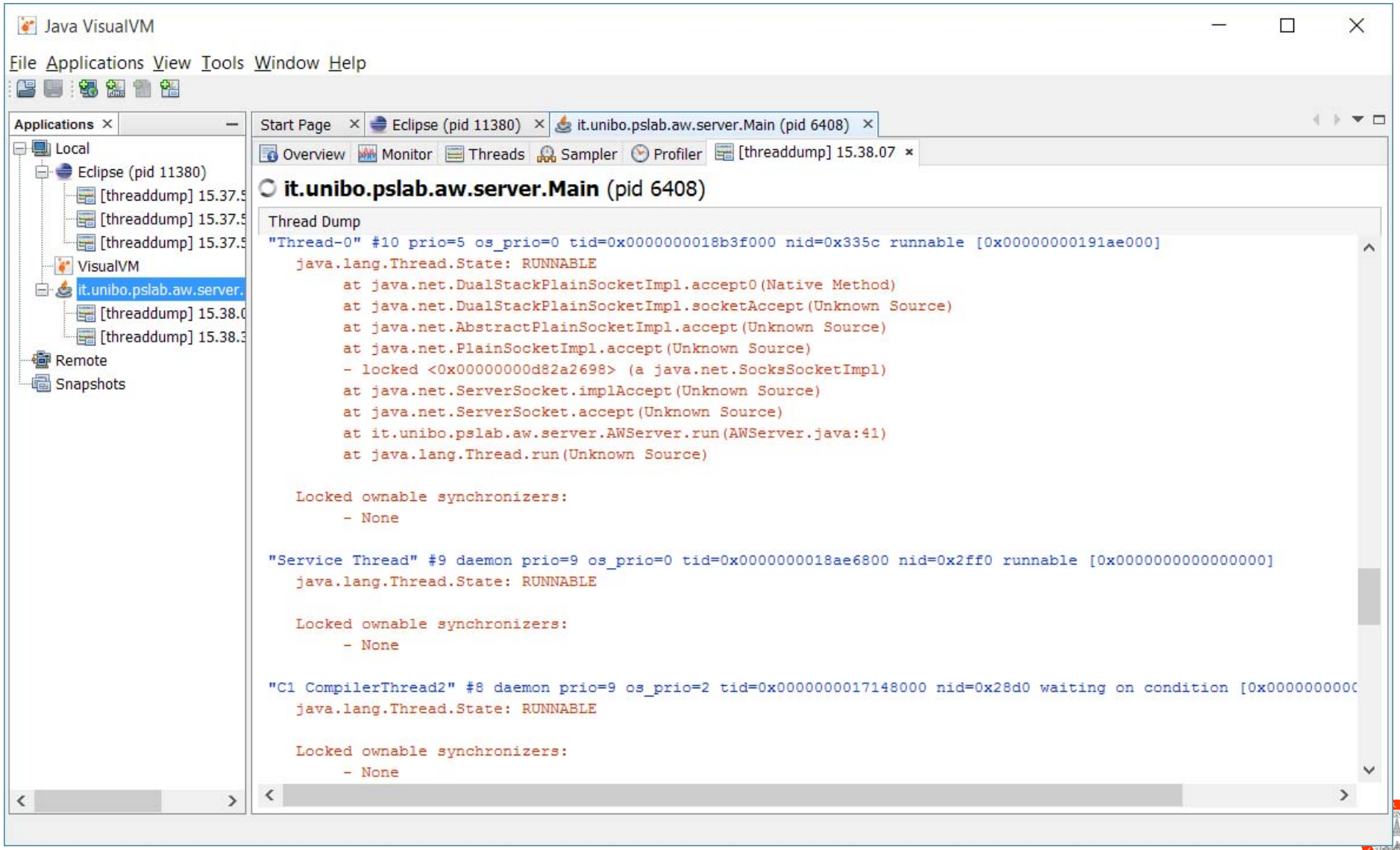
VisualVM - Monitor



VisualVM - Threads Status



VisualVM - Thread Dump Example



Outline

1. Programmazione multipiattaforma
   - Accesso al file system
   - Accesso ai dettagli del sistema
   - GUI scalabili e internazionalizzazione
2. Costruzione di un ambiente di sviluppo portabile
   - Corretta configurazione di un progetto Eclipse, inclusione di librerie
   - Font ed encoding
   - Caricamento di risorse dal classpath
   - Installazione delle impostazioni per-utente
3. Ricerca/Utilizzo di librerie
4. Qualità del codice
   - Strumenti per il controllo di qualità
   - Coding style: configurazione IDE Eclipse
5. Ispezione e Profiling con VisualVM
6. Preparazione al laboratorio

Preparazione dell'ambiente di lavoro

1. Clonare il repository degli esercizi
   - Si raccomanda di creare prima una propria fork
2. Importare il repository in Eclipse come progetto Java
   2.1 File
   2.2 Import
   2.3 General
   2.4 Existing project into workspace
   2.5 Si selezioni la cartella del progetto
   2.6 Si confermi l'import
3. Configurare correttamente Eclipse, abilitare e configurare i plugin per il controllo di qualità del codice

Modalità di lavoro

1. Seguire le istruzioni del file README.md nella root del repository
2. Tentare di capire l'esercizio in autonomia
   - Contattare il docente se qualcosa non è chiaro
3. Risolvere l'esercizio in autonomia
   - Contattare il docente se si rimane bloccati
4. Utilizzare le funzioni di test per verificare la soluzione realizzata
5. Cercare di risolvere autonomamente eventuali piccoli problemi che possono verificarsi durante lo svolgimento degli esercizi
   - Contattare il docente se, anche **dopo aver usato il debugger**, non si è riusciti a risalire all'origine del problema
6. Scrivere la Javadoc per l'esercizio svolto
7. Assicurarsi che non ci siano warning nel proprio codice
8. Effettuare *almeno* un commit ad esercizio completato
9. **A esercizio ultimato sottoporre la soluzione al docente**
10. Proseguire con l'esercizio seguente