07
JAR e Javadoc

Danilo Pianini
Giovanni Ciatto, Angelo Croatti, Mirko Viroli

Ingegneria e Scienze Informatiche
Alma Mater Studiorum—Università di Bologna, Cesena

5 novembre 2017

Outline

Preparazione dell'ambiente di lavoro

1. Clonare il repository degli esercizi
   - `https://bitbucket.org/danysk/courses-2017-oop-lab-07`
   - Si copi la URI con cui effettuare il clone dall'interfaccia web di Bitbucket
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

Modalità di lavoro

1. Nella root del repository c'è un file README.md che descrive le operazioni da svolgere
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 presenti nei sorgenti per il testing dell'esercizio
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 il Javadoc per l'esercizio svolto
7. Effettuare almeno un commit ad esercizio completato
   - Fatene pure quanti ne volete durante l'esercizio stesso
8. **A esercizio ultimato sottoporre la soluzione al docente**
9. Proseguire con l'esercizio seguente

Outline

Outline

Deployment di applicazioni Java

- Finora, abbiamo visto che un'applicazione Java è composta di un insieme di classi: noi vorremmo distribuirla come singolo file!
- Normalmente, le applicazioni Java vengono confezionate in file JAR (Java ARchive)

File JAR

- Un JAR è un archivio (normalmente compresso) che contiene le classi, le risorse (e.g. icone) e un file descrittivo detto "Manifest"
- Il Manifest viene creato sempre in `META-INF/MANIFEST.MF`, e contiene informazioni sull'applicazione, ad esempio su quale classe contenga il main del programma
- È possibile associare (a livello di sistema operativo) l'esecuzione del JAR file al comando Java, in modo che l'applicazione si avvii automaticamente "col doppio click" (avviando automaticamente la classe scritta nel Manifest)
- È possibile utilizzare i file JAR come componenti di altre applicazioni

Il comando `jar`

- Per creare un file JAR, si può utilizzare direttamente il comando `jar`
  - Normalmente, tuttavia, ci si avvale del supporto dell'IDE
  - O meglio ancora, di strumenti di build automation (Gradle, Maven...)
    - Non sono parte del corso...

Opzioni importanti del comando `jar`

`c` — Specifica l'intenzione di creare un JAR file
`f` — Specifica un file di output (se non presente, l'output è rediretto su standard output)
`m` — Specifica l'intenzione di allegare un manifest file personalizzato (se non presente, ne viene creato uno di default, che non specifica alcuna classe da eseguire)

Esempi d'uso del comando jar

```
jar cf jar-file.jar file1 file2 directory1
```

- Crea un nuovo JAR file di nome `jar-file.jar` contenente `file1` `file2` `directory1`. Include un Manifest di default.

```
jar cf jar-file.jar *
```

- Crea un nuovo JAR file di nome `jar-file.jar` contenente tutti i file e le directory nel path corrente. Include un Manifest di default.

Esecuzione di JAR file tramite command line shell

- `java` ha un'opzione che consente l'esecuzione di file jar.
- Tale opzione è `-jar`
- Quando si lancia `java -jar nomefile.jar`, la Java Virtual Machine automaticamente legge il file Manifest, cerca una descrizione della Main Class da eseguire e tenta di eseguirla.

Outline

Generazione di file JAR con Eclipse (1/5)

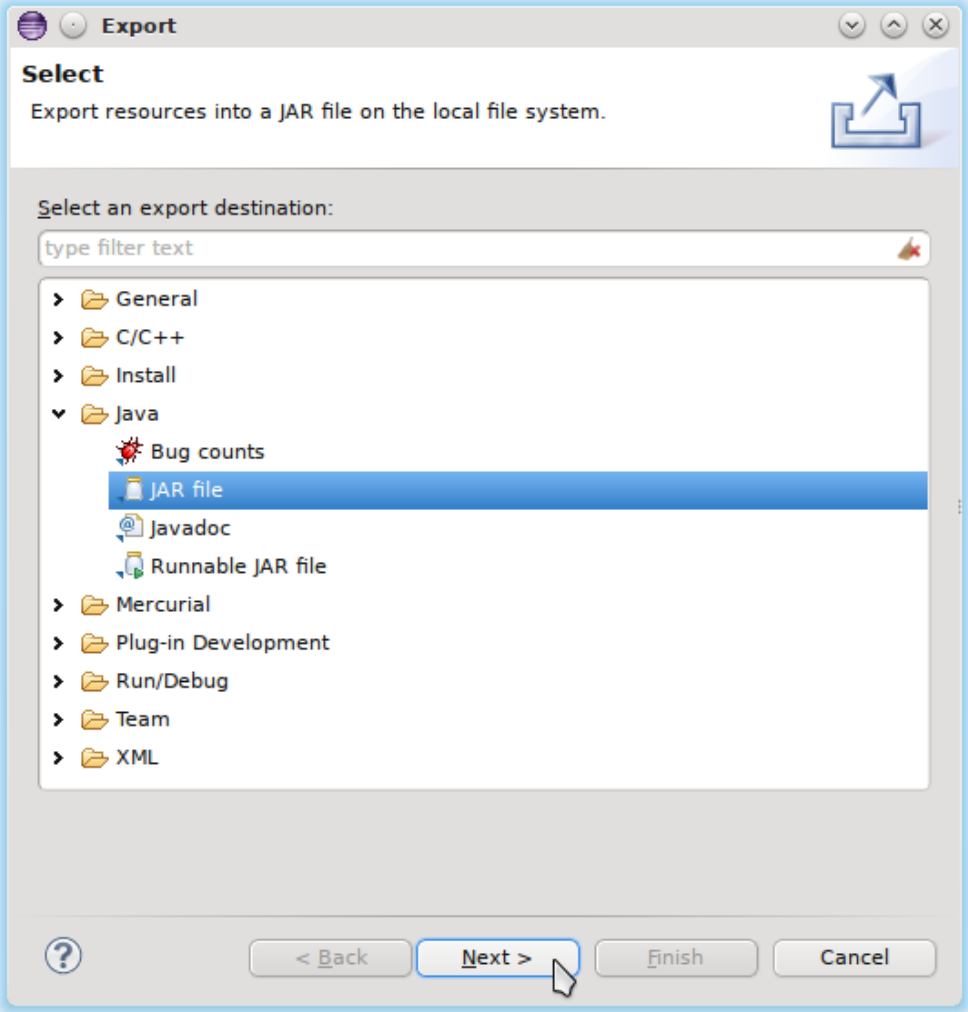
- L'IDE Eclipse supporta efficacemente l'utente nella generazione di file JAR (eseguibili o per la creazione di librerie).

Esercizio

- Si seguano i passi descritti nelle slide successive per generare un file JAR a partire dai tre packages:
  - `it.unibo.oop.jar.packages.pkg1`
  - `it.unibo.oop.jar.packages.pkg2`
  - `it.unibo.oop.jar.packages.pkg3`

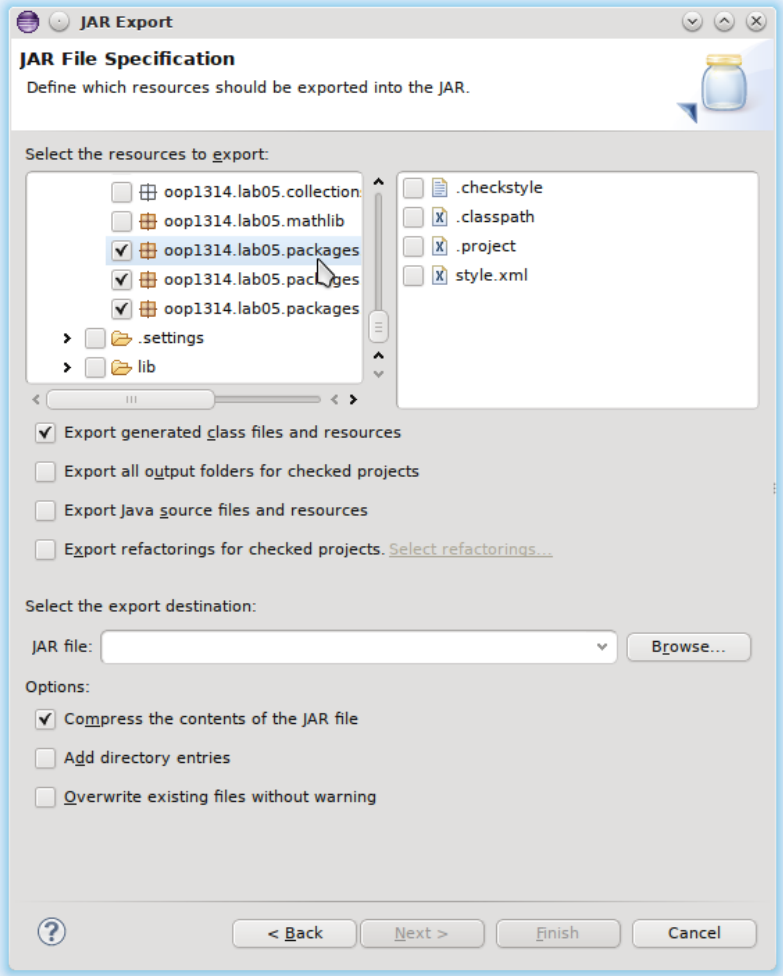
Generazione di file JAR con Eclipse (2/5)

- Click col tasto destro sul progetto > Export...
- Nella schermata di export: java > JAR file > Next



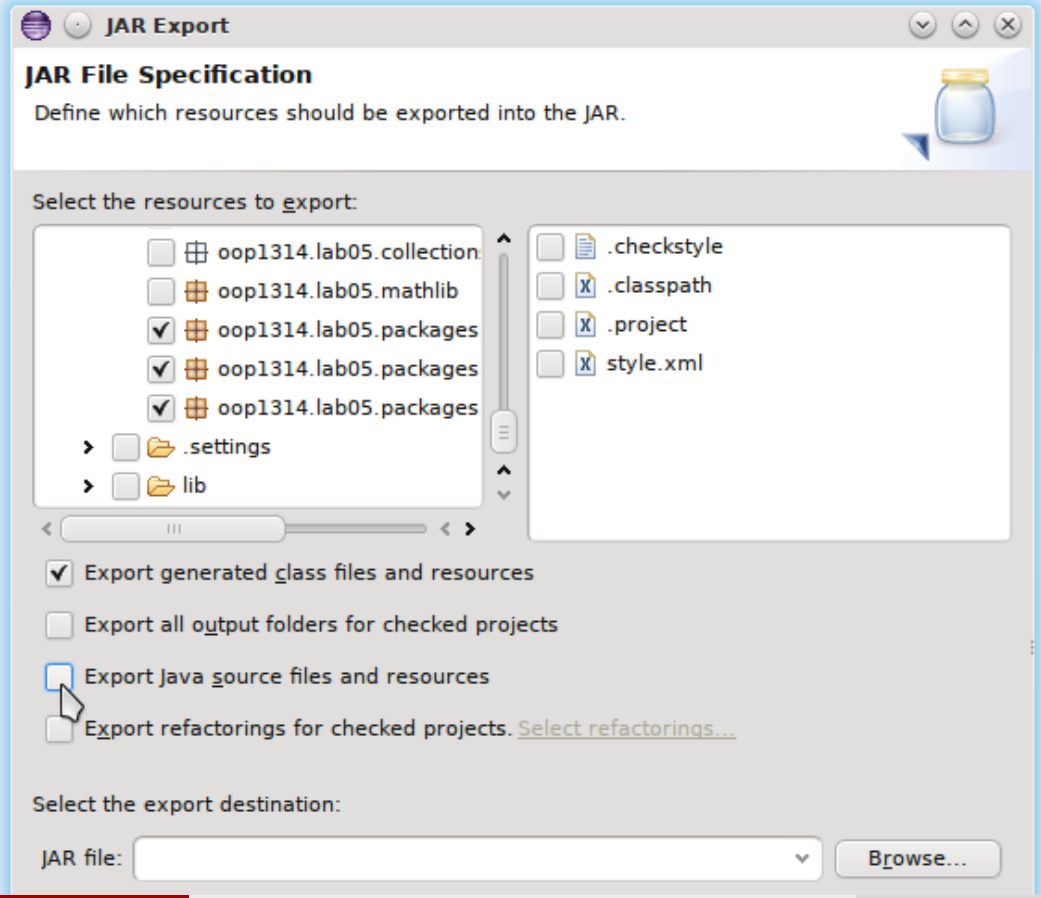
Generazione di file JAR con Eclipse (3/5)

- Espandere l'elenco dei file: verificare **sempre** che siano selezionate **tutte e sole** le risorse d'interesse per il JAR file



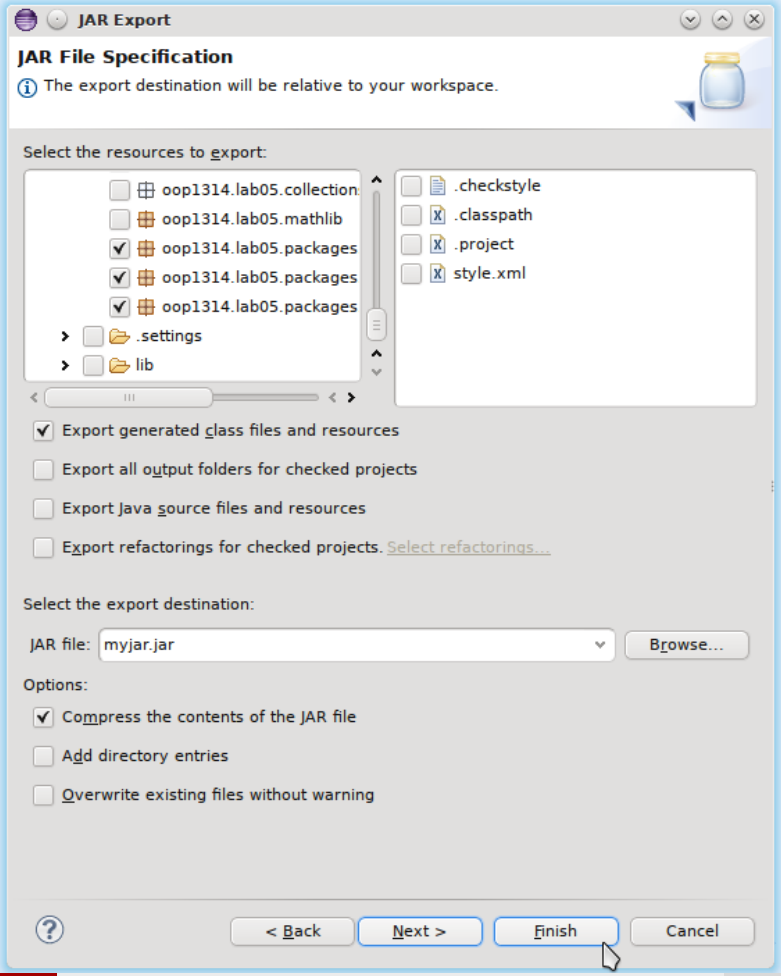
Generazione di file JAR con Eclipse (4/5)

- Risulta possibile esportare anche i sorgenti nel file JAR
  - Utile nel caso si voglia rendere ispezionabile il codice sorgente delle librerie distribuite come file JAR (ad esempio) in fase di debug.
- In questo caso, selezionare: *Export Java Source files and resources*



Generazione di file JAR con Eclipse (5/5)

- Selezionare una directory di destinazione
- Fornire un nome (con estensione .jar) al file che sarà generato
- Click su Finish



Verifica del contenuto di un JAR

- Il tool `jar` distribuito con il JDK consente di verificare il contenuto del file compresso
  - Aprire un terminale
  - Digitare `jar tf myjar.jar` (dove `myjar.jar` è il nome del file JAR da verificare

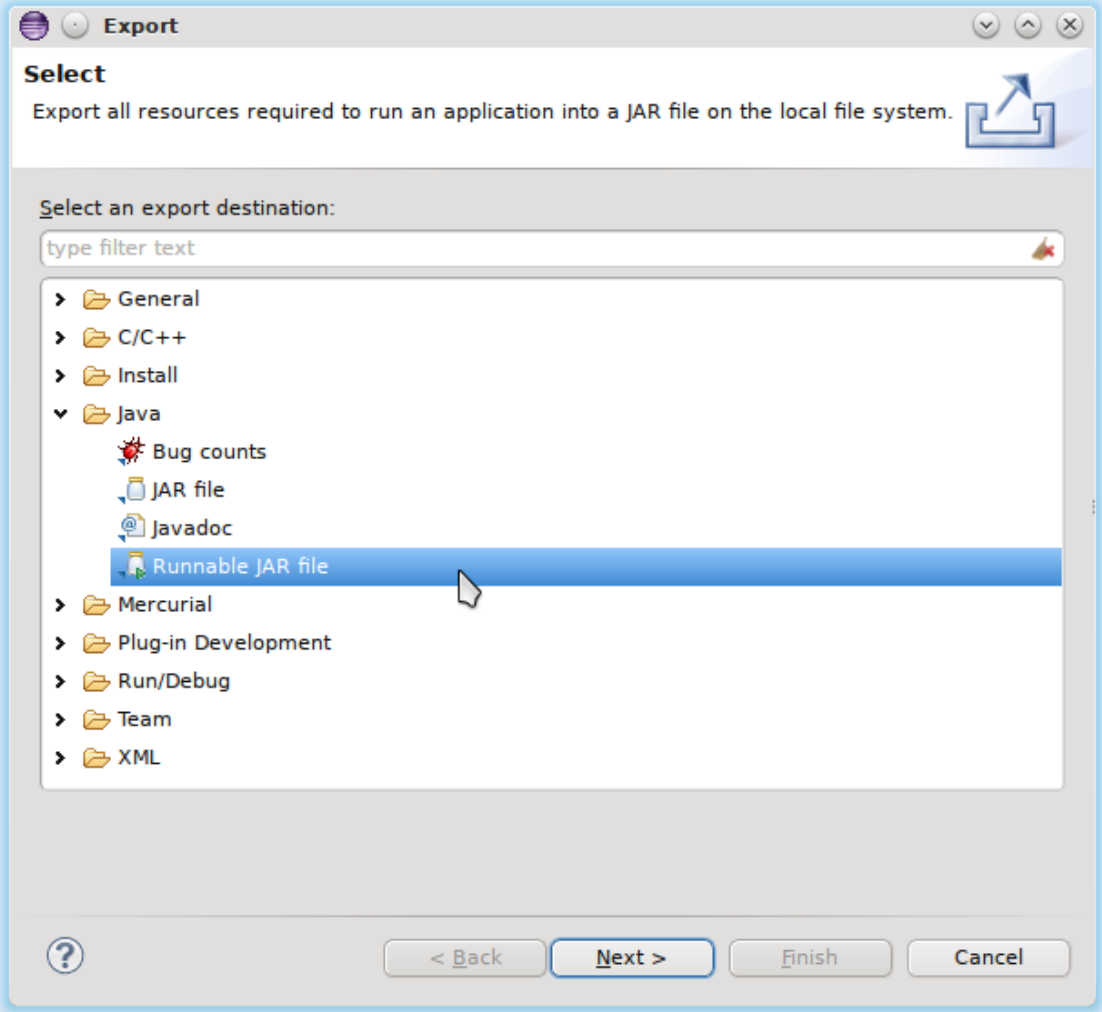
Outline

Generazione di file JAR eseguibili con Eclipse (1/3)

- Eclipse consente anche di generare file JAR con un file Manifest ad hoc per mettere in esecuzione il progetto software contenuto nell'archivio.
- Come prerequisito, è necessario che esista in Eclipse una "Run Configuration" valida per il progetto.
  - Ad esempio se il progetto sia stato eseguito almeno una volta in Eclipse.

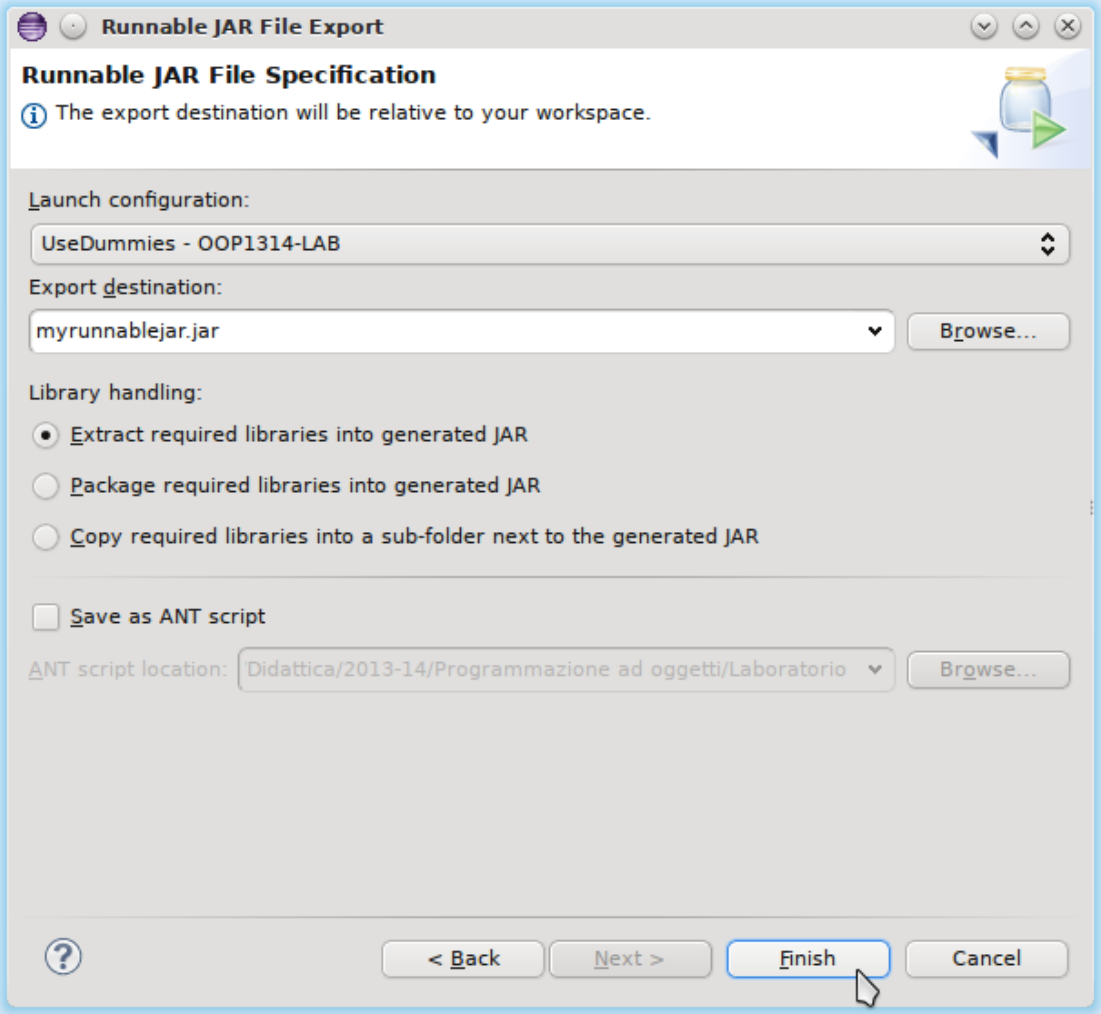
Generazione di file JAR eseguibili con Eclipse (2/3)

- Clik DX su progetto > Export...
- Nella schermata di export: java > Runnable JAR file > Next



Generazione di file JAR eseguibili con Eclipse (3/3)

- Selezionare la "Run configuration" da utilizzare
- Inserire il nome del file (e selezionare il path in cui si desidera crearlo)



Outline

Documentazione per il Codice Sorgente

Il ruolo fondamentale della documentazione

La documentazione di un progetto software è un aspetto fondamentale

- Al fine di garantirne la manutenibilità
  - Siamo così sicuri di ricordarci a distanza di settimane del perché abbiamo adottato una certa architettura, scritto una certa classe, una data estensione di una classe, un certo metodo?
  - Arriva in azienda un nuovo sviluppatore: come fa a costruirsi il background necessario per lavorare su un progetto software esistente?
- Al fine di aumentare la comprensione del codice
  - Cosa farà mai il metodo `doStuff()` sviluppato dal collega?
  - Ci sono dei metodi che sono disponibili solo per ragioni di compatibilità e che non dovrei più utilizzare?

Oggi ci concentreremo sulla Javadoc che, integrata all'uso di UML, è una delle parti fondamentali di una buona documentazione di un progetto SW

Outline

Javadoc – Introduzione

Javadoc: che cos'è?

Tool di supporto per la generazione automatica della documentazione (HTML-based) dei programmi Java, tramite utilizzo di annotazioni *"speciali"* collocate in punti ben precisi dei sorgenti

Come funziona

- Il tool processa tutti i commenti del tipo `/** ... */`
- Commenti che si trovano in testa a dichiarazione di classi, metodi, etc. vengono inclusi per generare automaticamente la documentazione
- Custom tag consentono di classificare diverse tipologie di informazioni, facilitando la scrittura e la generazione della documentazione

Outline

Javadoc – Principali Tag (1/2)

- `@param`
  - Utilizzabile nei commenti relativi a costruttori/metodi o in classi parametriche
  - Descrive un generico parametro di input oppure un tipo generico
- `@return`
  - Utilizzabile nei commenti relativi ai metodi
  - Descrive il valore di ritorno
- `@deprecated`
  - Utilizzabile nei commenti relativi a classi/costruttori/metodi/campi
  - Descrive che quel particolare costruttore/metodo etc. è stato deprecato. E' ancora disponibile per ragioni di compatibilità ma è opportuno non utilizzarlo nello sviluppo di nuove applicazioni

Javadoc – Principali Tag (2/2)

- `@throws`
  - Utilizzabile nei commenti relativi a costruttori/metodi che lanciano una eccezione (le vedremo in seguito)
  - Descrive l'eccezione e il motivo per cui viene lanciata
- `@link`
  - Descrive collegamenti ipertestuali (link) a metodi/classi/costanti della stessa classe o anche di classi esterne
  - Esempi di utilizzo
    - `{@link package.OtherClass#someMethod}`
    - `{@link #someMethodOfSameClass}`
    - `{@link #someFieldOfSameClass}`
- `@author` ed `@version`
  - Utilizzabili nel commento che descrive la classe
  - Descrive l'autore/autori della classe, e la sua versione
  - Non usateli. Abbiamo di DVCS per sapere esattamente chi ha fatto cosa, quando, e qual è la versione della classe.

Javadoc – Linee Guida

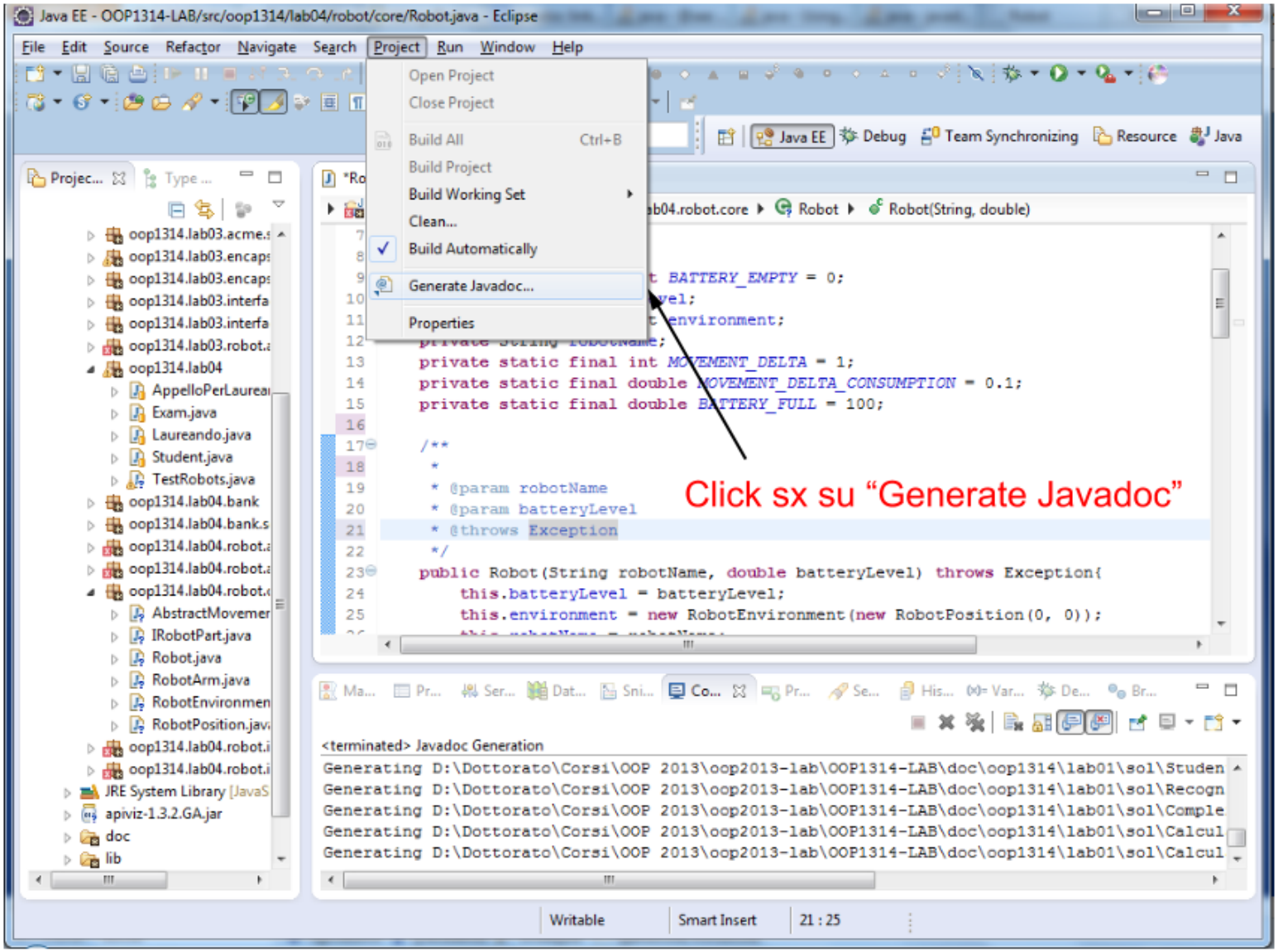
Cosa commentare e quali tag usare?

- Inserire sempre un commento (anche corto) che descrive il ruolo e il funzionamento generale della classe
- Inserire un commento per tutti i costruttori (parametri e return value inclusi), metodi, e campi con livello di accesso `public` e `protected`
- Non è necessario documentare metodi di cui si fa override, a meno che non vi siano peculiarità non rilevate nella documentazione della superclasse. In questo caso, è sufficiente utilizzare `{@inheritDoc}` all'interno del commento, e verrà generata Javadoc prendendola dalla superclasse.

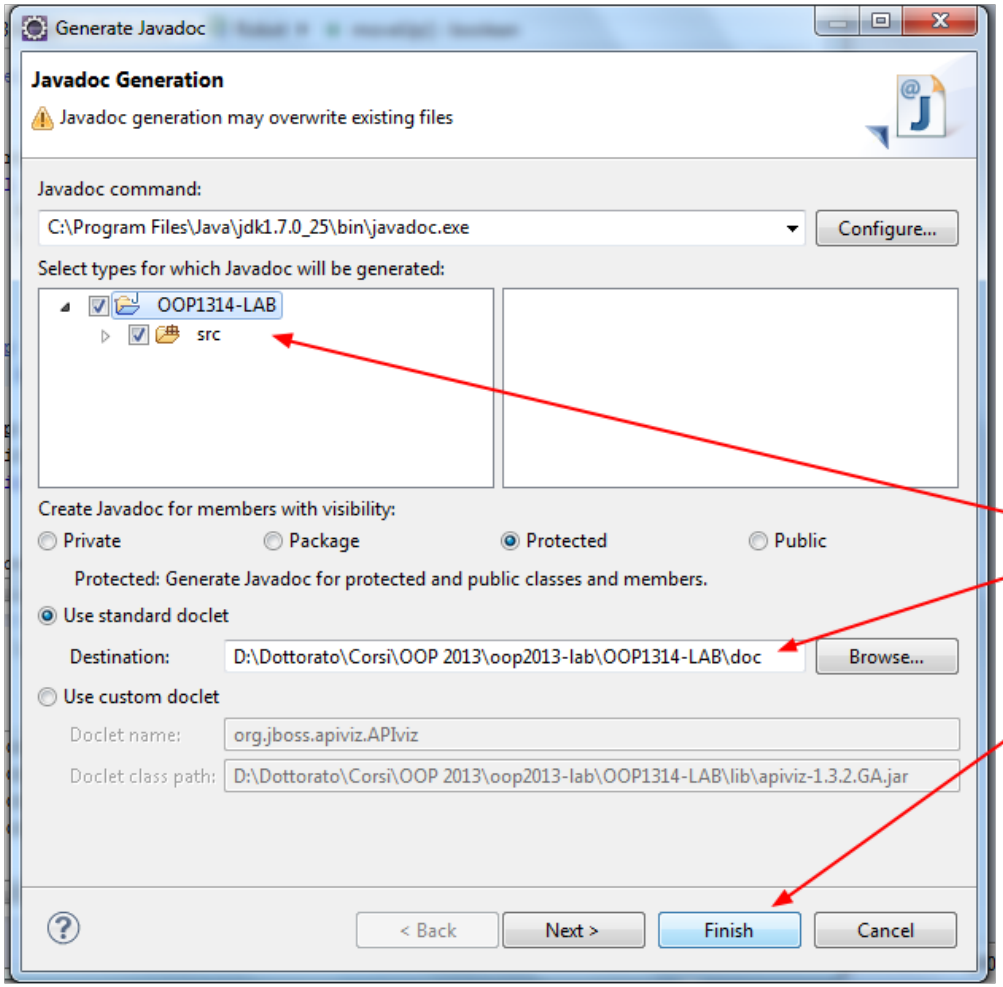
Utilizzare i sorgenti che vi forniamo in lab come linea guida di riferimento

Outline

Generazione Javadoc tramite Eclipse (1/2)



Generazione Javadoc tramite Eclipse (2/2)



1) Selezionare i package
2) Selezionare l'output folder
3) Click sx finish

Javadoc: Esempio di Documentazione Generata