Stile del Codice, Compilazione/Esecuzione avanzata di programmi Java, Programmi con Argomenti, VSCode e Debugging

Programmazione ad Oggetti – Lab02

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

Pre-requisiti

- Rudimenti di programmazione e codifica
- Nozioni di base dei **filesystem**
  - **percorsi assoluti e relativi**
- Utilizzo del terminale
  - interazione con il file system attraverso terminale (navigazione; concetto di working directory; etc.)
- **Compilazione ed esecuzione di base** di programmi Java
  - uso basilare dei comandi `javac` e `java`
  - distinzione tra file sorgenti (`.java`) e file di classi compilate (`.class`)
  - concetto di **programma/applicazione** in Java
- Il concetto di **package** in Java
  - contenitore (organizzato gerarchicamente) di tipi (ad es. classi) che funge da namespace e permette controllo degli accessi ai tipi contenuti

Outline

Stili e Convenzioni I

- Il codice sorgente che un programmatore scrive, generalmente è **condiviso** con altre persone (del proprio team, ma anche persone esterne al team o la community)
  - è importante scrivere software **immediatamente comprensibile**
  - il fatto che un software "giri" (rispetti i requisiti e/o produca i risultati attesi) non è una sufficente metrica di qualità
- è importante, fondamentale, adottare uno stile e seguirlo
  - **chiaro** – facilmente comprensibile
  - **condiviso** – piuttoto che il "proprio stile"
  - **consistente** – con regole che non si contraddicono (a vari livelli)

Stili e Convenzioni II

> *Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live. Code for readability.*
>
> — *John Woods [disputed]*

Ogni linguaggio ha le sue prassi...

- Quelle Java di riferimento sono disponibili qui:
  - `http://bit.ly/java-style-guide`
  - `http://bit.ly/java-code-conventions`
  - `http://bit.ly/oracle-java-code-conventions`

Stili e Convenzioni III

Ogni azienda poi è libera di darsi altre regole interne

- Ad esempio:
  - Google: `http://archive.is/a0Jhz`
  - Twitter: `http://archive.is/aa1tE`
  - Mozilla: `http://archive.is/rs3Ns`
- Notare che sono sempre **consistenti**!
  - E che sono tipicamente restrizioni delle convenzioni, non modifiche!

Nel corso faremo riferimento alle Java Code Conventions (con qualche vincolo in più)

Java Code Conventions (un estratto) I

Usare **sempre** le parentesi (graffe) per if, else, for, while, anche se segue una sola istruzione

- Aumentano la manutenibilità del codice
- È facile che nella fretta si modifichi il codice in modo sbagliato
- È facile che alcuni tool automatici si sbaglino quando "uniscono" pezzi di codice scritti da diverse persone
- Apple iOS soffrì di un grave bug a SSL/TLS causato da questa cattiva pratica `http://archive.is/KQp8E`

Le parentesi graffe vanno sempre "all'egiziana" (Egyptian brackets)

- La graffa che apre va in linea con lo statement di apertura
- La graffa che chiude va in a capo, nella stessa colonna dello statement di apertura

Java Code Conventions (un estratto) II

Naming conventions - **molto importanti!**

- I nomi di `package` usano sempre e solo lettere minuscole
- Usare sempre `camelCase`, evitare gli underscore (_)
- I nomi di classe cominciano sempre per maiuscola: `SomeClass`
- I nomi di campi, metodi e variabili locali iniziano sempre per minuscola: `o.someField`, `o.someMethod()`
- I campi `static final` (costanti di classe) sono interamente maiuscoli e possono usare underscore

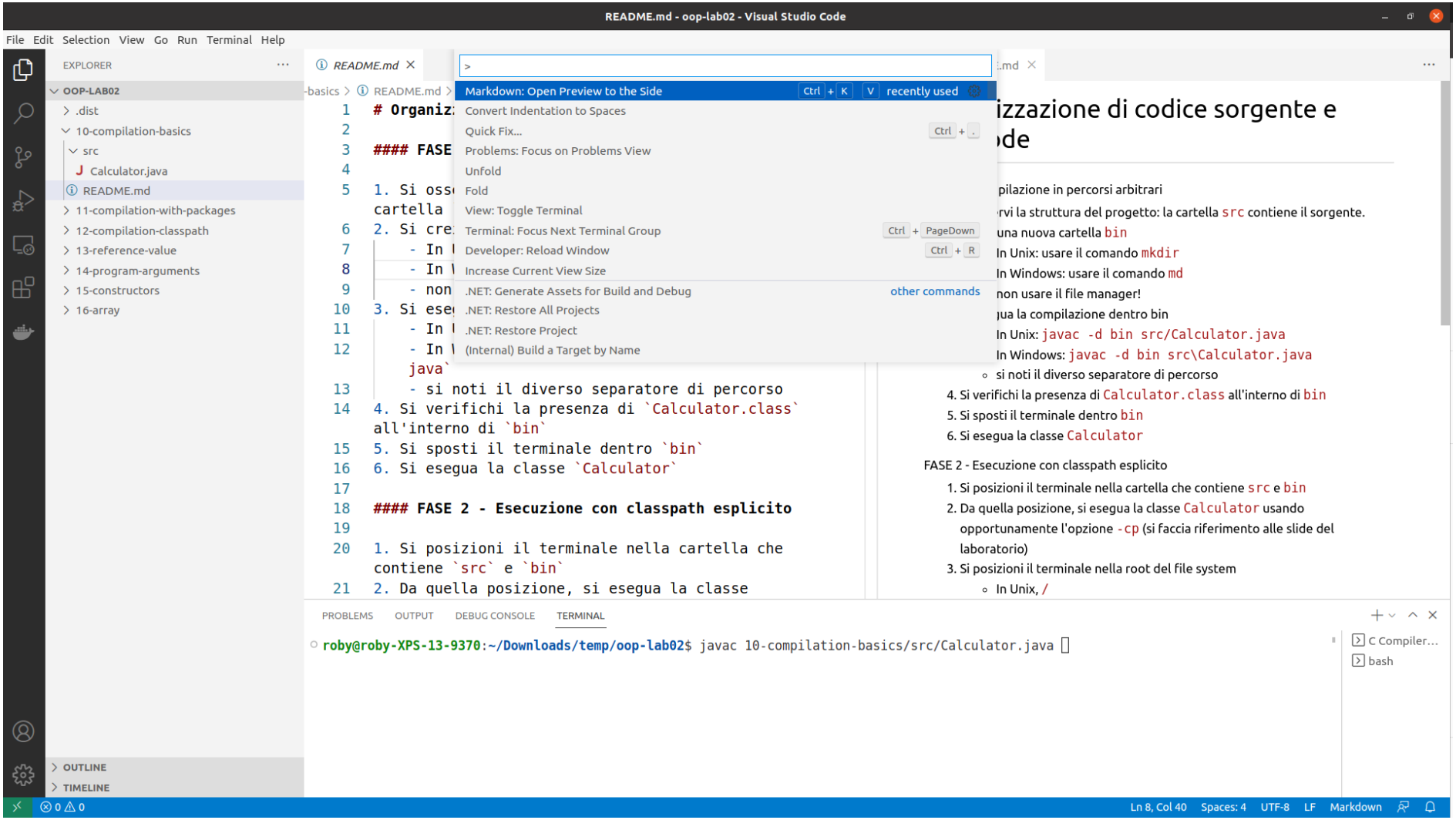
Come seguire stili e convenzioni? Tutto nelle mani del programmatore?

- Esistono strumenti automatici a supporto, introdotti nelle prossime lezioni...

Outline

Visual Studio Code (VSCode)

- **Visual Studio Code (VSCode)** è un editor di codice sorgente leggero, versatile, e multipiattaforma
- Estendibile attraverso un ecosistema di **estensioni** per vari linguaggi di programmazione e strumenti
  - Ne vedremo alcune nei prossimi lab


README.md - oop-lab02 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
OOP-LAB02
.dist
10-compilation-basics
src
Calculator.java
README.md
11-compilation-with-packages
12-compilation-classpath
13-reference-value
14-program-arguments
15-constructors
16-array
OUTLINE
TIMELINE
README.md
Markdown: Open Preview to the Side
Convert Indentation to Spaces
Quick Fix...
Problems: Focus on Problems View
Unfold
Fold
View: Toggle Terminal
Terminal: Focus Next Terminal Group
Developer: Reload Window
Increase Current View Size
.NET: Generate Assets for Build and Debug
.NET: Restore All Projects
.NET: Restore Project
(Internal) Build a Target by Name
other commands
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
roby@roby-XPS-13-9370:~/Downloads/temp/oop-lab02$ javac 10-compilation-basics/src/Calculator.java
C Compiler...
bash
Ln 8, Col 40
Spaces: 4
UTF-8
LF
Markdown

VSCode: il concetto di **workspace**

- Il **workspace** è
  1. una collezione di *una o più* **cartelle** aperte in una finestra (istanza di VSCode)
     - Tali cartelle sono visualizzate nella vista **Explorer** sulla sinistra
  2. più un insieme di **preferenze**, **configurazioni**, **stato**, ed **estensioni** attive memorizzate in una cartella **.vscode/**
- Creazione di un workspace
  - *File* → *Open Folder*
- Aggiunta di cartella top-level a un workspace
  - *File* → *Add Folder to Workspace..*
- Salvataggio ed apertura di un workspace
  - *Save* → *Workspace as...* → *name*.**code-workspace**
  - *File* → *Open Workspace from File*

Utilizzo di VSCode: alcune note

- Varie scorciatoie da tastiera
  - *CTRL + SHIFT + P*: **command palette** (a.k.a. "l'unico shortcut che veramente vi serve ricordare")
  - *CTRL + SPACE*: intellisense
  - *CTRL + S*: salvataggio file corrente
  - *CTRL + PAGE UP/DOWN*: tab sorgente precedente/successivo
- **Apertura di un terminale:** *Terminal* $\rightarrow$ *New Terminal*
  - Sarà MOLTO UTILE
- **Visualizzare/installare estensioni:** *File* $\rightarrow$ *Preferences* $\rightarrow$ *Extensions* oppure *CTRL + SHIFT + X*

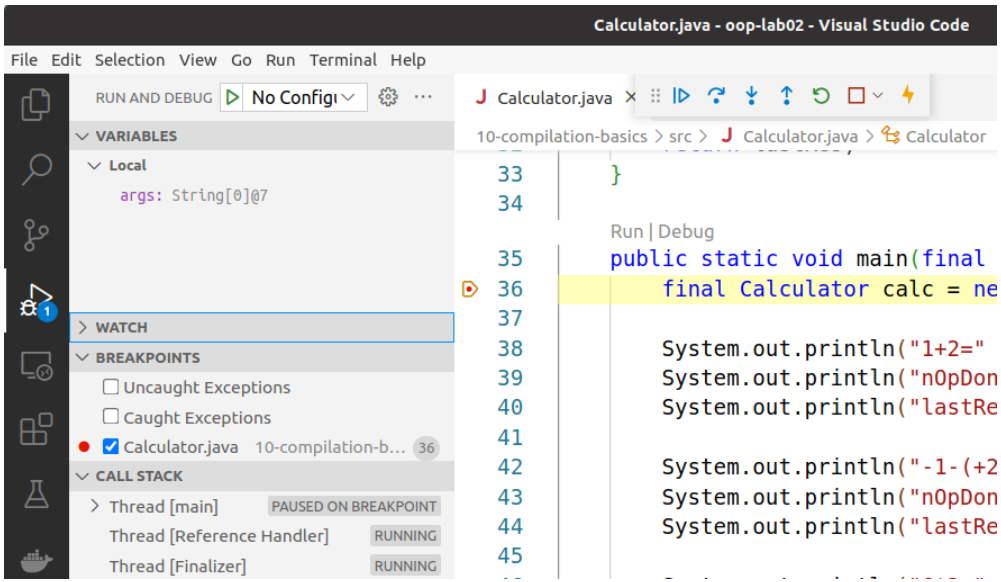
VSCode: debugging di applicazioni Java I

Nota: richiede l'estensione **Debugger for Java**

1. Creazione di **breakpoint** (punti di rottura del flusso di controllo)
   - Click a sinistra del numero di linea di una riga di codice di interesse

```
34
        Run | Debug
35      public static void main(final String[] args) {
36          final Calculator calc = new Calculator();
37
38          System.out.println("1+2=" + calc.add(n1: 1, n2: 2));
```

2. Esecuzione di un'applicazione Java in **"modalità debug"**
   - Click destro su file java → *Debug Java*



VSCode: debugging di applicazioni Java II

3. Controllo e ispezione del comportamento runtime
   - *Variables*: vista che mostra le variabili in scope e il loro valore
   - *Watch*: permette di valutare espressioni rispetto al contesto d'esecuzione corrente
   - **Step Over** (F10) o anche *Run* $\rightarrow$ *Step Over*
   - **Step Into** (F11)
   - **Step Out** (SHIFT + F11)
   - **Continue** (F5)

Outline

Nuova opzione per `javac`

- Abbiamo già visto come compilare file sorgenti Java (file `.java`), generando classi in bytecode, che prendono la forma di file `.class` nella medesima directory
- Tuttavia è uso comune e **buona pratica** nella gestione di progetti articolati, **separare le classi sorgenti dal bytecode**, ad esempio:
  - cartella `src`, per i file sorgenti (`.java`)
  - cartella `bin`, contenente le classi compilate (`.class`)
- Come si fa?

Nuova opzione del comando `javac`

- `-d`: consente di specificare la cartella destinazione in cui compilare i file `.java`
- Si tratta di un'opzione che dovete obbligatoriamente saper usare

Sarà oggetto di valutazione in sede di prova pratica!

Compilazione di più file da qualunque directory verso una qualunque directory

Compilazione in directory arbitrarie

`javac -d <CARTELLA DESTINAZIONE> <FILE JAVA>`

- **OVVIAMENTE** vanno sostituite le variabili fra parentesi angolari con le directory che andranno usate.

Compilazione di più file in una singola passata

`javac -d <CARTELLA DESTINAZIONE> <ELENCO DI FILE JAVA>`

- **OVVIAMENTE** vanno sostituite le variabili fra parentesi angolari con le directory che andranno usate.

È possibile anche utilizzare la wildcard (*) invece di elencare tutti i file!

- Su Unix si possono usare wildcard in più punti del path, ad esempio `progetti/*/src/*.java` elenca tutti i file con estensione java dentro ciascuna cartella `src` di ciascuna cartella dentro `progetti`

Il classpath in Java I

- Il risultato della compilazione di sorgenti Java sono una o più **classi**
  - A partire dalla cartella di destinazione (opzione `-d` di `javac`), ogni compilato `.class` sarà creato in un **sottopercorso di cartelle che corrisponde al percorso del package dichiarato per la classe corrispondente**
  - Ovvero, indipendentemente da dove si trovi un sorgente `C.java` definente una classe `foo.bar.C`, con `javac -d <DEST> path/to/C.java` il compilato sarà creato in `<DEST>/foo/bar/C.class`
- Quando si va ad eseguire (comando `java`), si eseguono **classi**, non files
  - Infatti la virtual machine si aspetta il nome completo di una classe, il **Fully-Qualified Class Name (FQCN)**, in input
    - NON il percorso al file dov'è scritta
    - NON il percorso al file dov'è compilata

Il classpath in Java II

Come fa la JVM a risolvere le classi?

- Possiede un elenco di percorsi **a partire dai quali** i file compilati possono essere trovati
  - All'interno di questi percorsi, i file devono essere opportunamente organizzati: **la struttura delle cartelle deve replicare quella dei package**
- Cerca (in ordine) nei suddetti percorsi la classe che gli serve
- I percorsi possono essere directory, file compressi, o indirizzi di rete
- Ad esempio: se si danno i due percorsi `/a/b/c` e `../foo` e si chiede di eseguire il programma definito in `it.unibo.Program`, allora la JVM cercherà di caricare la classe da `/a/b/c/it/unibo/Program.class` o `../foo/it/unibo/Program.class`
- Per approfondire: `http://archive.is/0ziau`

L'insieme *ordinato* dei percorsi prende il nome di **classpath**

Il classpath in Java III

Default classpath

Se non specificato, il classpath di Java include automaticamente:

- I file jar del Java Runtime Environment
  - `rt.jar` ed altri file importanti
  - Contengono ad esempio `java.lang.Math`
- La directory corrente

Aggiungere directory al classpath

Possono essere aggiunte directory al classpath

- Si usa l'opzione `-cp` (o, equivalentemente, `-classpath`), seguita da un elenco di percorsi
  - separati dal simbolo : (Unix)
  - o dal simbolo ; (Windows)
  - Per evitare problemi con simboli e percorsi, conviene circondare l'intero classpath con doppi apici (simbolo ")

Più cartelle nel classpath

Il classpath non è composto da una sola cartella, ma può contenere più cartelle, oltre a file compressi contenenti bytecode e risorse, come zip o jar (li vedremo in futuro).

Si possono specificare più cartelle utilizzando come separatore il simbolo ":" (per sistemi Unix) oppure ";" (per sistemi Windows)

- `javac -d bin -cp "lib1:lib2:lib3" src/*.java`
  - Compila tutti i file con estensione java che si trovano nella cartella `src`, mettendo i compilati dentro `bin`. In compilazione, potrà linkare tutte le classi che si trovano nelle cartelle `lib1`, `lib2` e `lib3`: nel caso in cui alcuni sorgenti in src stiano usando delle classi definite dentro queste cartelle, la compilazione avrà successo.
  - Equivalente Windows: `javac -d bin -cp "lib1;lib2;lib3" src/*.java`
- `java -cp "bin:lib1:lib2:lib3" MyClass`
  - Esegue il main della classe `MyClass`. Cercherà questa classe e tutte quelle collegate all'interno delle cartelle `bin`, `lib1`, `lib2` e `lib3`.
  - Equivalente Windows: `java -cp "bin;lib1;lib2;lib3" MyClass`

Organizzazione dei sorgenti in presenza di package

È buona norma organizzare i sorgenti in modo da rappresentare su filesystem la struttura dei package. Si noti però che (dato che il compilatore lavora su file) questa scelta **non è teoricamente obbligatoria**!

- Lo è di fatto in questo corso, perché le cose van fatte bene
- Lo sarà nel mondo del lavoro, perché è prassi assolutamente comune

Risultato della compilazione

Quando ad essere compilata è una classe dichiarata in un package, il compilatore **riproduce la struttura dei package usando delle directory**

- Dato che l'interprete non lavora con file ma con classi, il loro layout sul file system **non può essere modificato!**

Esecuzione

L'esecuzione è identica al caso precedente, si faccia solo attenzione ad usare l'intero nome della classe, che in Java include anche il nome del package!

Uso del classpath in fase di compilazione I

Supponiamo di avere in mano la seguente classe:

```
1  package oop.lab02.math;
2
3  public class UseComplex {
4
5      public static void main(final String[] args) {
6          final ComplexNum c1 = new ComplexNum();
7          c1.build(1, -45);
8          final ComplexNum c2 = new ComplexNum();
9          c2.build(2, 8);
10
11         System.out.println(c1.toStringRep());
12         System.out.println(c2.toStringRep());
13
14         c1.add(c2);
15         System.out.println("c1 new value is: " + c1.toStringRep() + "\n");
16     }
17 }
```

Uso del classpath in fase di compilazione II

Comprensione degli errori

Se provassimo a compilarla da sola, potremmo ottenere degli errori

```
src\oop\lab2\math\UseComplex.java:6: error: cannot find symbol
    ComplexNum c1 = new ComplexNum();
    ^
  symbol:   class ComplexNum
  location: class UseComplex
src\oop\lab2\math\UseComplex.java:6: error: cannot find symbol
    ComplexNum c1 = new ComplexNum();
                        ^
  symbol:   class ComplexNum
  location: class UseComplex
src\oop\lab2\math\UseComplex.java:8: error: cannot find symbol
    ComplexNum c2 = new ComplexNum();
    ^
...
```

- Il compilatore ha bisogno di conoscere la classe `ComplexNum` per poterla linkare e per poter compilare una classe che la riferisce
- Il compilatore cerca nel classpath il bytecode della classe `ComplexNum`
- Come risolviamo?

Uso del classpath in fase di compilazione III

Utilizzo di -cp in fase di compilazione

- Supponiamo di avere solo la versione compilata di `ComplexNum` (ovvero non il sorgente)
  - Notate che questa è la *norma* quando si usano delle librerie: vengono fornite già compilate!
- Basterà mettere il percorso a partire dal quale `oop/lab02/math/ComplexNum.class` può essere individuata nel classpath di `javac`!
- Supponiamo di avere `UseComplex.java` nel percorso `src/oop/lab02/math/`
- Supponiamo di aver compilato `ComplexNum` con destinazione (di partenza) `lib/`
- Possiamo usare:

```
javac -d bin -cp lib src/oop/lab02/math/UseComplex.java
```

Uso del classpath in fase di compilazione IV

Spiegazione del comando

`javac -d bin -cp lib src/oop/lab02/math/UseComplex.java`

- `javac` ⇒ Invocazione del compilatore
- `-d bin` ⇒ `-d` determina la **destinazione**. Vogliamo compilare dentro la cartella `bin`
- `-cp lib` ⇒ `-cp` consente di aggiungere percorsi al **classpath**. Noi vogliamo cercare le classi che ci servono, oltre che nella posizione corrente e nelle librerie java, anche dentro `lib`
- `src/oop/lab02/math/UseComplex.java` ⇒ Il *file* che vogliamo compilare

Passare più percorsi al classpath

Avendo come riferimento l'esempio precedente, proviamo ad eseguire.

- Per eseguire correttamente `UseComplex` dobbiamo dire alla JVM, tramite **-cp**, dove trovare:
  - `ComplexNum`
  - `UseComplex`
- Si trovano in *due percorsi diversi*!
- Dobbiamo specificare come argomento di `-cp` due percorsi, usando il **separatore**:
  - `:` su sistemi UNIX (Linux, Mac OSX, BSD)
  - `;` su sistemi Windows
- Useremo quindi:
  - `java -cp bin:lib oop.lab02.math.UseComplex` (Unix)
  - `java -cp bin;lib oop.lab02.math.UseComplex` (Windows)

Esempio con sommario di funzionamento di `javac` e `java`

Cartella da cui `javac` è invocato

Destinazione

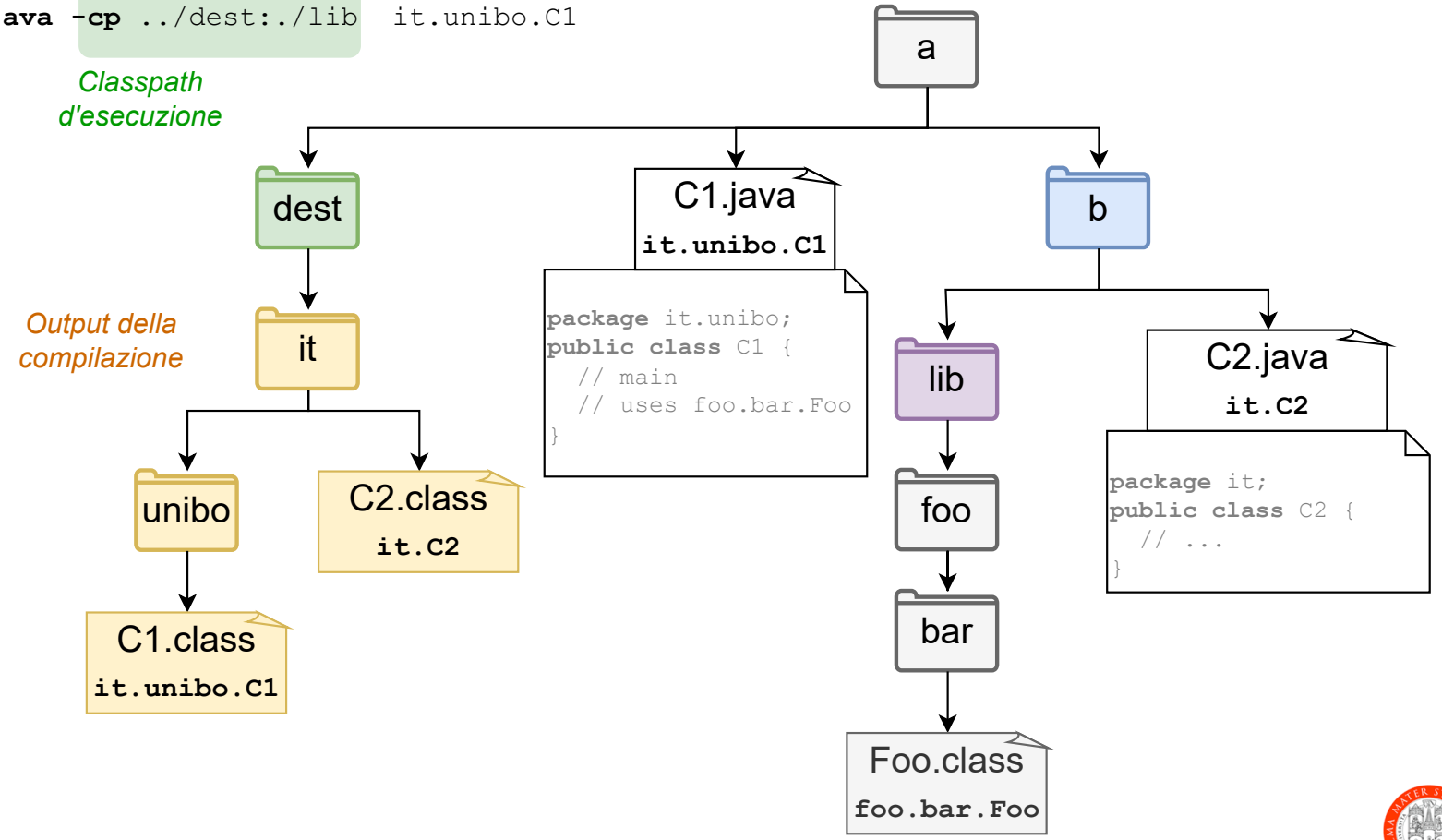
Classpath di compilazione

```
a/b $ javac -d ../dest -cp ./lib ./C2.java ../C1.java
```

```
a/b $ java -cp ../dest:./lib it.unibo.C1
```

Cartella da cui `java` è invocato

Classpath d'esecuzione



Consiglio finale

Visto che all'esame il loro utilizzo è richiesto, è necessario imparare **a memoria** le opzioni di `java` e `javac`?

NO

Entrambi i comandi (e praticamente tutti i comandi Unix) hanno con loro un'opzione che consente di stampare a video un help. Provate

- java -help
- javac -help

Gli help stampano abbondante testo con le relative istruzioni e a me serve una riga, davvero devo imparare a leggere e capire un help?

SÌ

È molto facile dimenticarsi la sintassi delle opzioni di comandi che non si usano spesso. È molto più facile imparare a destreggiarsi in un help che andare a tentativi o ricordare cose a memoria.

Outline

Passaggio di argomenti ad un programma Java I

- La maggior parte dei comandi supporta degli argomenti
  - Ad esempio, quando eseguite `javac -d bin MyClass.java` gli argomenti sono:
    1. `-d`
    2. `bin`
    3. `MyClass.java`
- In C, questi vengono passati al metodo `main()` come coppia di `char **` e `int`, rappresentanti rispettivamente un riferimento all'area di memoria dove sono salvati i parametri ed il numero dei suddetti.
- Anche in Java ovviamente è possibile passare degli argomenti ad un programma

Passaggio di argomenti ad un programma Java II

- La gestione è un po' più semplice che in C, grazie al fatto che gli array si portano dietro l'informazione circa la loro dimensione
- E grazie al fatto che la signature del metodo `main()` è una sola in Java
  - `public static void main(String [])` è l'unica signature valida
  - Mentre in C sia `int main(void)` che `int main(char **, int)` sono ugualmente accettabili
- Gli argomenti con cui un programma Java viene invocato vengono passati come parametri attraverso l'array (`String[] args`) che il metodo `main()` prende in ingresso
  - Nonostante sia un parametro del *metodo principale* di qualunque programma Java, si tratta di un comune array senza alcuna particolarità.

Outline

Preparazione ambiente di lavoro

- Accedere al PC di laboratorio con le proprie credenziali istituzionali
- Accedere al sito del corso
- Scaricare il materiale dell'esercitazione odierna
- Spostare il file scaricato sul Desktop
- Decomprimere il file e aprire con Visual Studio Code la directory ottenuta (File $\rightarrow$ Open Folder...)
- Puntare il terminale alla directory con i sorgenti dell'esercitazione odierna

Outline

A1 – Varianza

Formula per il calcolo della varianza

Sia n il numero di elementi dell'array ed x_i l'elemento all'indice i dell'array, e μ la media dei valori del suddetto array. La varianza σ^2 può essere calcolata come:

$$\sigma^2 = \frac{\sum_{i=0}^{n-1}(x_i - \mu)^2}{n}$$