

STRUMENTI DEL JDK

PROGRAMMAZIONE AD OGGETTI
C.D.L. INGEGNERIA E SCIENZE INFORMATICHE

Danilo Pianini — danilo.pianini@unibo.it

Slide compilate il: 2025-09-28

 [versione stampabile](#)

 [menu principale](#)

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, eccetera)
- **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

STILI E CONVENZIONI PER IL CODICE Sorgente

Stili e Convenzioni

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 sufficiente metrica di qualità
- è importante adottare uno stile e seguirlo
 - ▶ **chiaro** – facilmente comprensibile
 - ▶ **condiviso** – piuttosto che il “proprio stile”
 - ▶ **consistente** – con regole che non si contraddicono

“ 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

Linguaggi diversi, regole di stile diverse!

- Usare convenzioni comuni su altri linguaggi su Java è una pessima idea
 - ▶ esempio tipico: usare lo stile **Allman** (tipico di C#) invece di **1TBS/OTBS** (tipico di Java)
- È vero anche il contrario, ovviamente!
 - ▶ Quando imparerete altri linguaggi, imparate anche le loro convenzioni e non riusate quelle di Java!

Le prassi di riferimento per Java sono disponibili qui:

- <http://bit.ly/java-style-guide>
- <http://bit.ly/java-code-conventions>
- <http://bit.ly/oracle-java-code-conventions>

Ogni azienda / team poi può darsi regole interne (solitamente *in aggiunta*)

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 CHE INTRODURREMO MAN MANO)**

Java Code Conventions (un estratto)

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 sbagliano 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>

Si usi lo stile “One True Brace Style” (1TBS o OTBS)

Le parentesi graffe vanno sempre “all’egiziana” (Egyptian brackets)

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

Naming conventions - **molto importanti!**

```
// mini guida: PascalCase, camelCase, snake_case, kebab-case
```

- I nomi di **package** usano sempre e solo lettere minuscole e numeri, senza underscore (`_`)
- I nomi di variabili, campi, e metodi usano sempre **camelCase**: `myVariable`, `myMethod()`, `myObject.myField`
- I nomi di classe utilizzano invece **PascalCase** (cominciano per maiuscola): `SomeClass`
- I campi **static final** (costanti di classe) usano **SNAKE_CASE**, ma solo con lettere maiuscole

COME SEGUIRE STILI E CONVENZIONI?

Ovviamente può essere difficile fare tutto a mano: esistono strumenti automatici a supporto, che introdurremo nelle prossime lezioni...

COMPILAZIONE ED ESECUZIONE AVANZATA IN JAVA

Nuova opzione per **javac**

- Abbiamo già visto come compilare file sorgenti Java (file **.java**), generando classi sotto forma di file in bytecode con estensione **.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 **sh** e shell derivate 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

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

Come fa la JVM a trovare (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 nei suddetti percorsi (in ordine) la classe che gli serve
 - ▶ Ad esempio: se si danno i due percorsi **/a/b/c** e **../foo** e si chiede di eseguire il programma definito nella classe **Program**, allora la JVM cercherà di caricare la classe da **/a/b/c/Program.class** e, se non la trova, da **../foo/Program.class**
 - ▶ I percorsi possono essere directory, file compressi, o indirizzi di rete
 - ▶ Per approfondire: <http://archive.is/Oziau>

L'INSIEME **ORDINATO** DEI PERCORSI PRENDE IL NOME DI **CLASSPATH**

Il classpath in Java




Default classpath

Se non specificato, il classpath di Java include automaticamente:







- Il Java Runtime Environment
 - ▶ Contengono ad esempio `java.lang.Math`
 - ▶ La directory da cui viene invocato il comando

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 `:` su  
 - ▶ o dal simbolo `;` su 
 - ▶ Per evitare problemi con simboli e percorsi, conviene circondare l'intero classpath con doppi apici (simbolo `"`)
 - ▶ Diversamente, percorsi con spazi o simboli speciali potrebbero non essere interpretati correttamente

Esempi

- Su   `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`: la compilazione avrà successo anche se le classi che stanno venendo compilate usano librerie contenute nelle cartelle precedenti.
 - ▶ Equivalente : `javac -d bin -cp "lib1;lib2;lib3" src/*.java`
- Su   `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 : `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!**
 - ▶ Si tratta, tuttavia, di un aspetto implementativo: noi metteremo nel classpath le directory che abbiamo passato al compilatore con **-d**

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

Supponiamo di avere in mano la seguente classe:

```
package oop.lab02.math;

public class UseComplex {

    public static void main(final String[] args) {
        final ComplexNum c1 = new ComplexNum();
        c1.build(1, -45);
        final ComplexNum c2 = new ComplexNum();
        c2.build(2, 8);

        System.out.println(c1.toStringRep());
        System.out.println(c2.toStringRep());

        c1.add(c2);
        System.out.println("c1 new value is: " + c1.toStringRep() + "\n");
    }
}
```

ed eseguiamo `javac UseComplex.java`. Cosa otteniamo?

Comprensione degli errori

Otteniamo 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?

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`







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  
 - ▶ **;** su 
- Useremo quindi:
 - ▶   **java -cp bin:lib oop.lab02.math.UseComplex**
 - ▶  **java -cp bin;lib oop.lab02.math.UseComplex** (Windows)

Esempio con javac e java

Cartella da cui
javac è invocato

a/b

\$

javac

-d ../dest

-cp ../lib

./C2.java ../C1.java

Cartella da cui
java è invocato

a/b

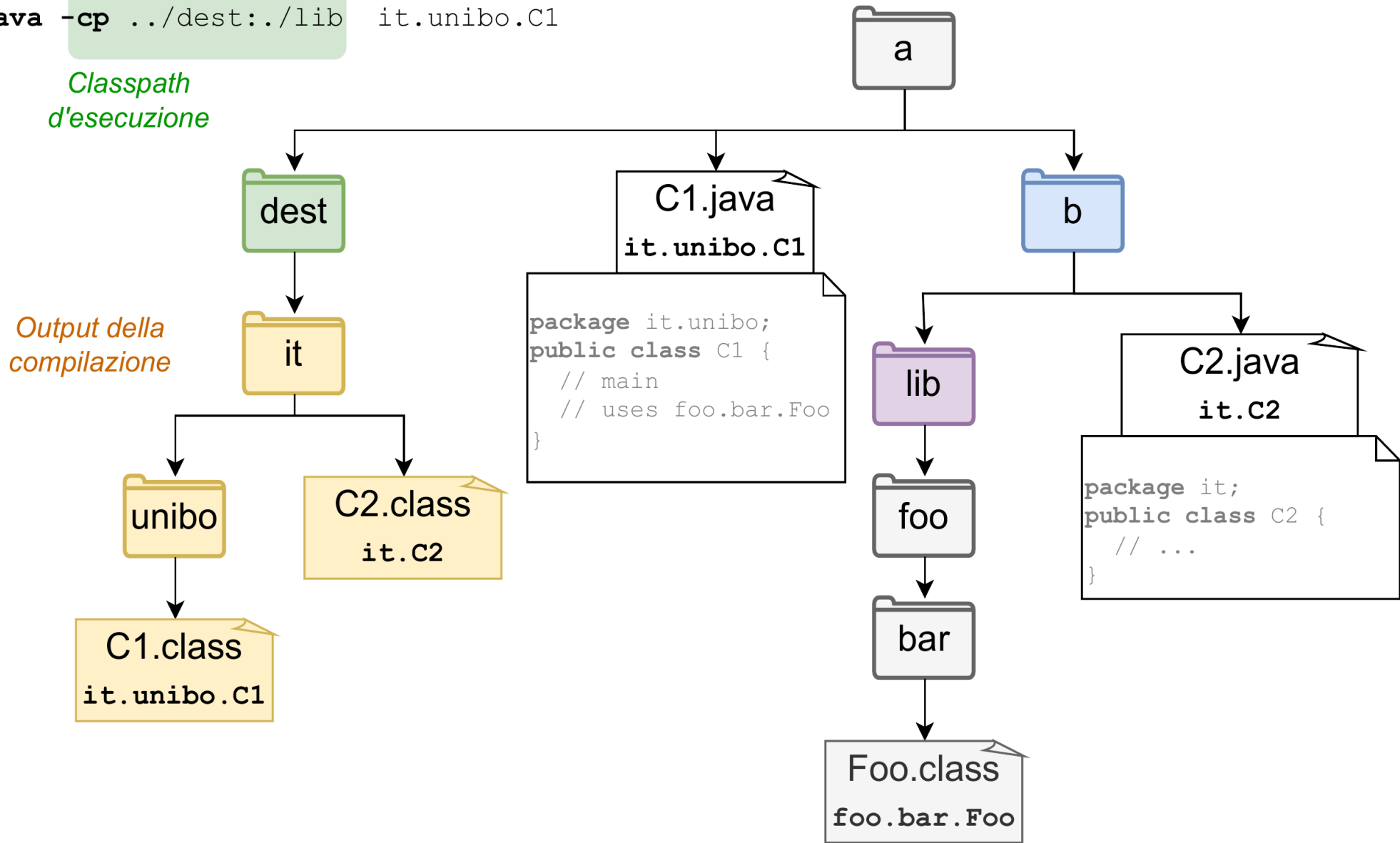
\$

java

-cp ../dest:../lib

it.unibo.C1

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.

ESECUZIONE DI PROGRAMMI JAVA CON ARGOMENTI

Passaggio di argomenti ad un programma Java

La maggior parte dei comandi supporta degli argomenti

Ad esempio, quando eseguite `javac -d bin MyClass.java` gli argomenti sono le seguenti tre stringhe:

1. `-d`
2. `bin`
3. `MyClass.java`

- In C, questi vengono passati al metodo `int 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

La gestione è un po' *più semplice rispetto a *C perché che *gli array si portano dietro la loro dimensione come campo*

In Java la signature del metodo `main()` è una univoca: `public static void main(String [])`, mentre in C sia `int main(void)` che `int main(char **, int)` sono 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.

ESERCIZI DI OGGI

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
- Puntare il terminale alla directory con i sorgenti dell'esercitazione odierna

APPENDICE: RICHIAMI UTILI PER GLI ESERCIZI DEL LAB

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:

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

STRUMENTI DEL JDK

PROGRAMMAZIONE AD OGGETTI
C.D.L. INGEGNERIA E SCIENZE INFORMATICHE

Danilo Pianini — danilo.pianini@unibo.it

Slide compilate il: 2025-09-28

 [versione stampabile](#)

 [menu principale](#)