01
Introduzione al laboratorio e strumenti di base

Danilo Pianini
Angelo Croatti, Giovanni Ciatto, Mirko Viroli

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

29 settembre 2017

Il Laboratorio di OOP

Titolare del corso: Prof. Mirko Viroli

- e-mail — `mirko.viroli@unibo.it`
- homepage — `http://mirkoviroli.apice.unibo.it`

Titolare del corso di laboratorio: Ing. Danilo Pianini

- e-mail — `danilo.pianini@unibo.it`
- homepage — `http://danilopianini.apice.unibo.it`
- orario di ricevimento — Martedì, dalle ore 17:00

Tutors: Ing. Angelo Croatti & Ing. Giovanni Ciatto

- e-mail — `a.croatti@unibo.it` & `giovanni.ciatto@unibo.it`

Obiettivi

1. Apprendere ulteriori aspetti metodologici e pratici, oltre a quanto visto nel corso delle lezioni in aula.

2. Svolgere esercizi necessari per la comprensione e la sperimentazione dei contenuti del corso.

3. Confrontarsi con il docente e i tutor per delucidazioni in merito allo svolgimento degli esercizi proposti.

Organizzazione

Orario A.A. 2017–2018

- Primo Turno : Lunedì, 09:00 – 13:00, Lab. VeLa
- Secondo Turno : Martedì, 13:00 – 17:00, Lab. VeLa

Organizzazione

- Introduzione di nuovi metodi e strumenti da parte del docente
- Esercizi di difficoltà crescente
- Esercizi "aggiuntivi" (che **non** sono opzionali)

Come svolgere gli esercizi...

1. Leggere attentamente la consegna e provare in autonomia a risolvere ciascun esercizio.
   - Se dopo alcuni minuti non avete nessuna idea su come procedere per ottenere la soluzione, chiamate il docente o il tutor.
2. Fare riferimento alle slide del corso (da tenere a portata di mano, per una facile consultazione)
   - In modo particolare nei primi laboratori – ma in generale vale per tutti i laboratori – **evitate** di cercare porzioni di codice online.
   - Assicuratevi di avere sempre chiaro cosa stiate facendo e perché.
3. Concluso ciascun esercizio, chiamare il docente per la correzione e la discussione.

NOTA — Riuscire a completare anche gli esercizi "aggiuntivi" nelle quattro ore è indice di buona preparazione. Non riuscirci non è preoccupante, ma si raccomanda caldamente di terminarli a casa **prima** del laboratorio successivo.

Il ruolo del Forum Studenti

- Accessibile dal sito del corso (link **Forum Studenti** in homepage)

```
https://elearning-cds.unibo.it/mod/forum/view.php?id=71223
```

- È il punto di coordinamento per il laboratorio, e per tutto il corso di OOP.
- Stimola il confronto e la discussione tra gli studenti.
  - Per ciascuna questione posta sul forum, oltre ai docenti, anche gli studenti possono rispondere confrontandosi su soluzioni e problematiche nell'ambito del corso.
- È preferibile chiedere supporto tramite il forum piuttosto che avvalersi del ricevimento
  - La riposta data ad uno studente può essere utile anche per gli altri.

Outline

Goal della lezione

- Java Overview (linguaggio + piattaforma)
- Compilazione ed esecuzione di programmi Java a riga di comando
- Ingegnerizzazione e test di semplici classi

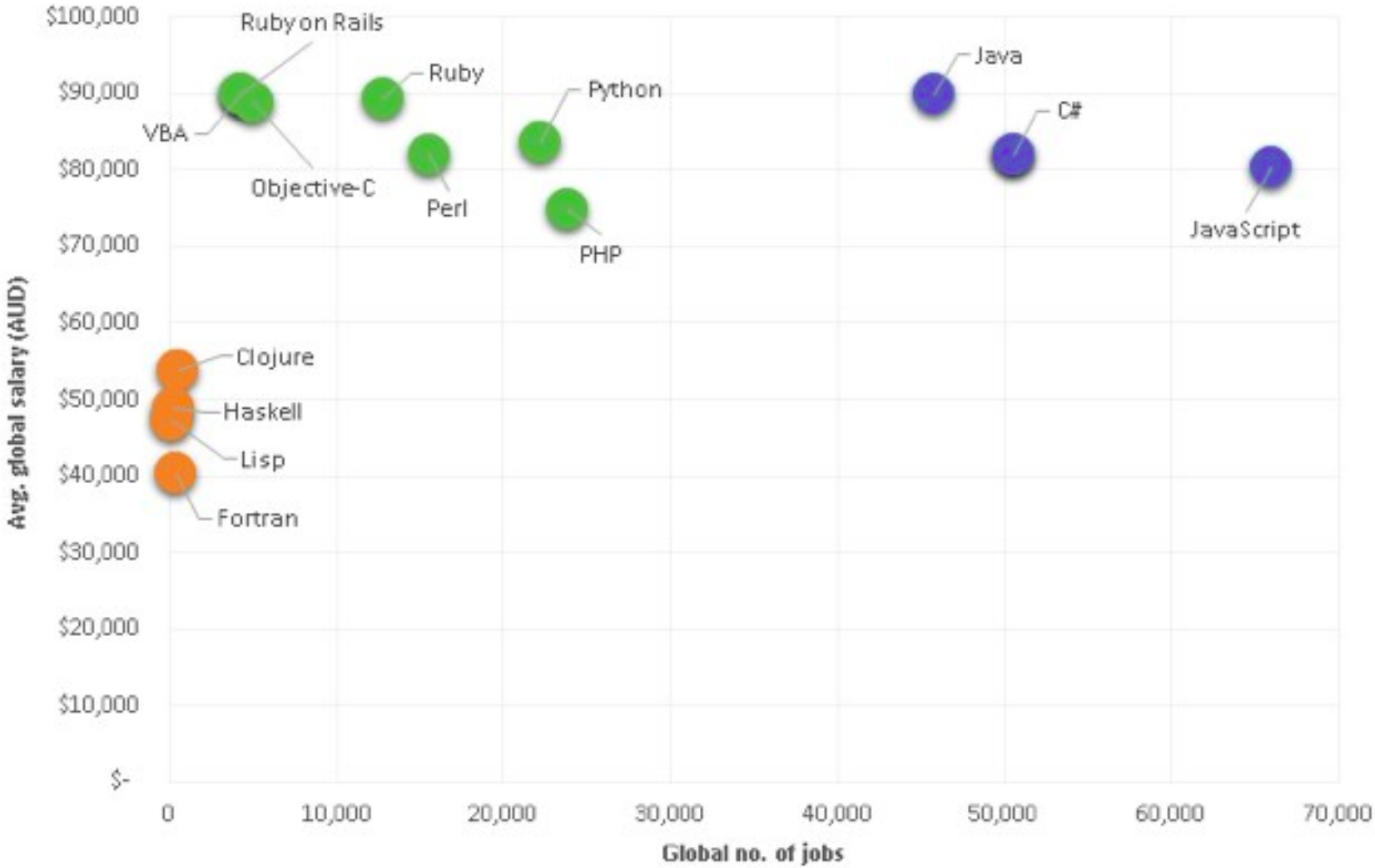
Outline

1. Java Overview
2. Richiamo: il file system
3. Richiamo: Interprete Comandi
4. Preparazione dell'ambiente di lavoro
5. Compilazione ed Esecuzione da Riga di Comando
6. Sviluppo e Test di Semplici Classi
   - Esercizi da Svolgere in Autonomia
7. Esercizi Aggiuntivi

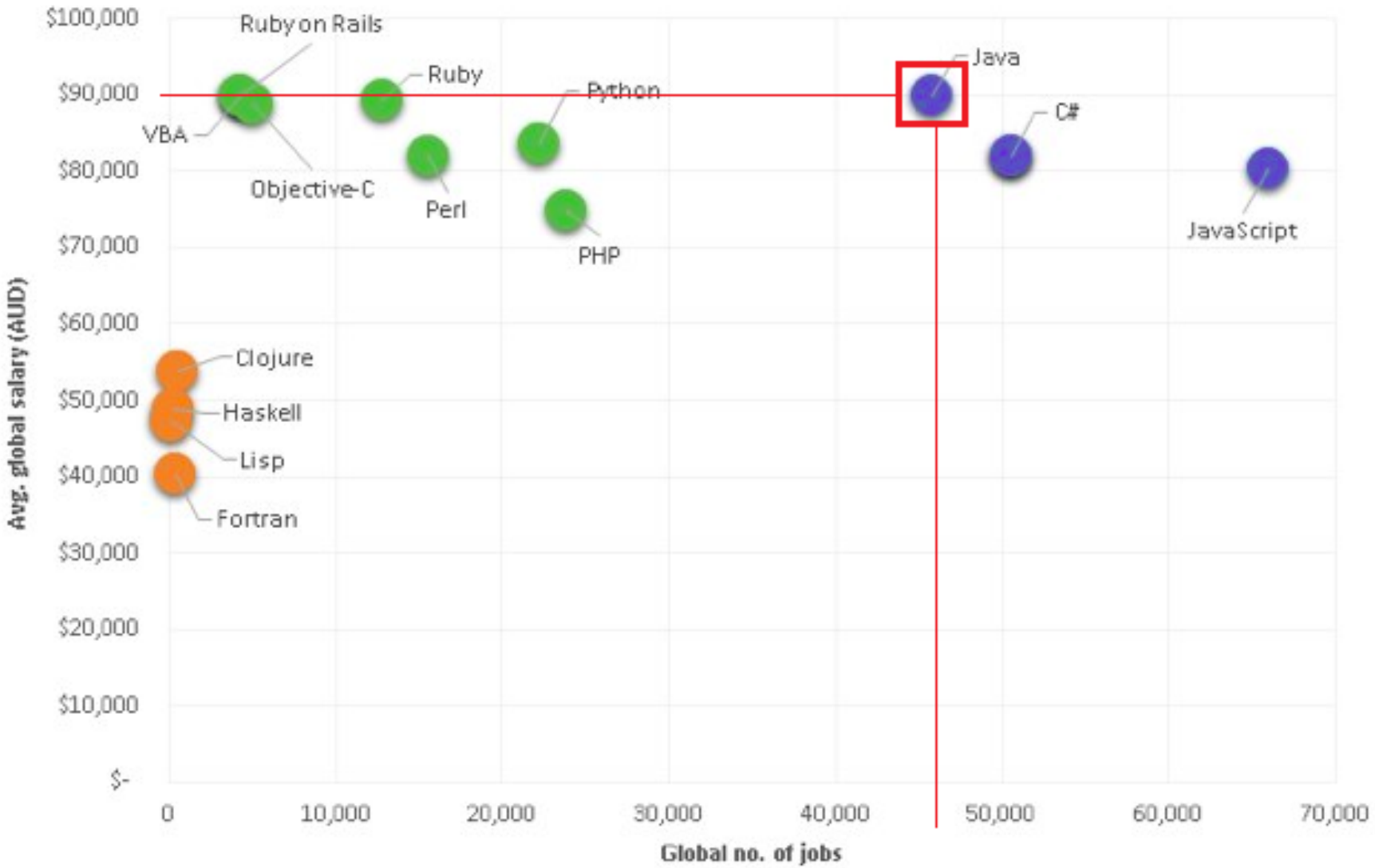
Outline

1. Java Overview
2. Richiamo: il file system
3. Richiamo: Interprete Comandi
4. Preparazione dell'ambiente di lavoro
5. Compilazione ed Esecuzione da Riga di Comando
6. Sviluppo e Test di Semplici Classi
   - Esercizi da Svolgere in Autonomia
7. Esercizi Aggiuntivi

Java nel mondo "reale" I



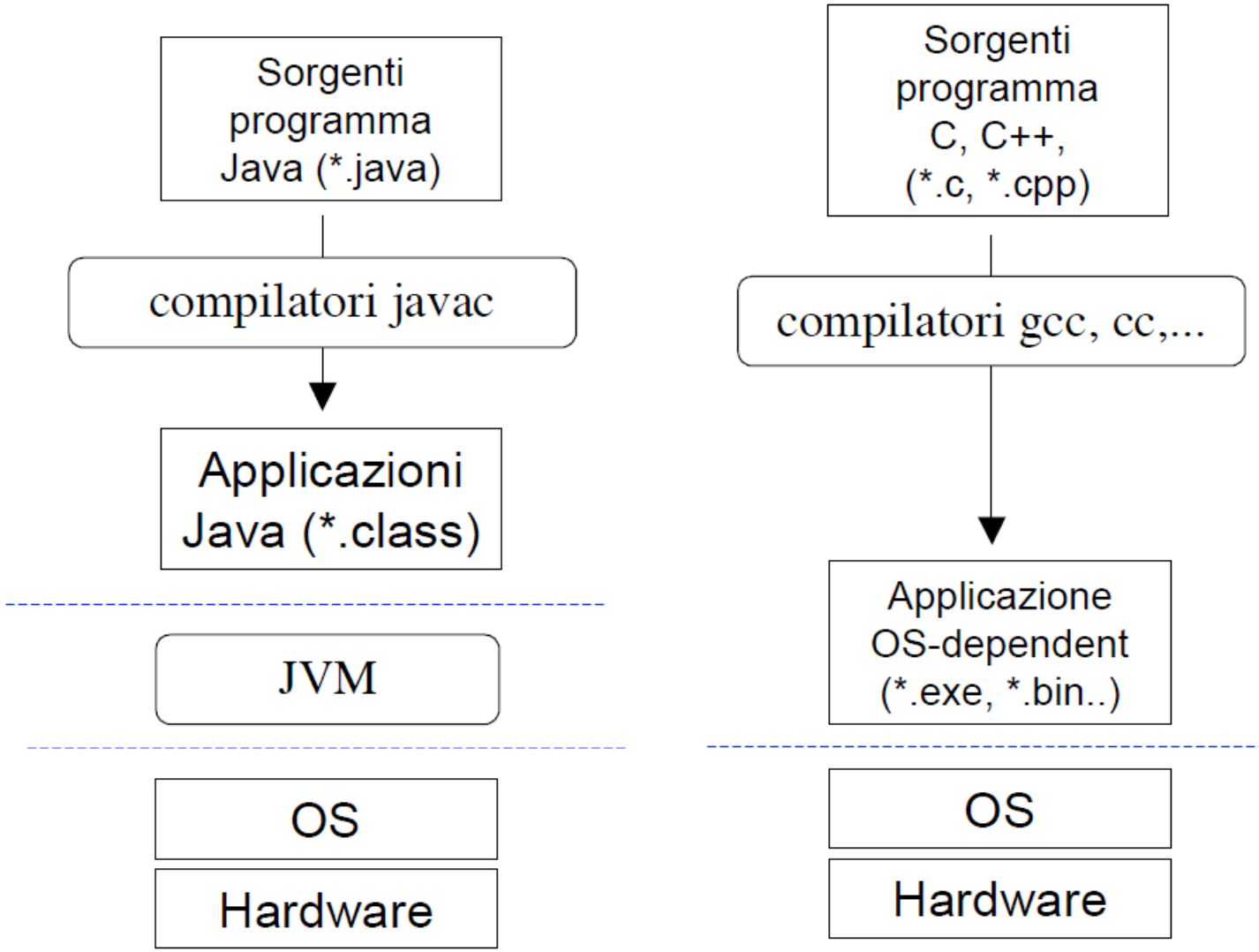
Java nel mondo "reale" II



Java nel mondo "reale" III

Sep 2016	Sep 2015	Change	Programming Language	Ratings	Change
1	1		Java	18.236%	-1.33%
2	2		C	10.955%	-4.67%
3	3		C++	6.657%	-0.13%
4	4		C#	5.493%	+0.58%
5	5		Python	4.302%	+0.64%
6	7	↑	JavaScript	2.929%	+0.59%
7	6	↓	PHP	2.847%	+0.32%
8	11	↑	Assembly language	2.417%	+0.61%
9	8	↓	Visual Basic .NET	2.343%	+0.28%
10	9	↓	Perl	2.333%	+0.43%
11	13	↑	Delphi/Object Pascal	2.169%	+0.42%
12	12		Ruby	1.965%	+0.18%
13	16	↑	Swift	1.930%	+0.74%
14	10	⇊	Objective-C	1.849%	+0.03%
15	17	↑	MATLAB	1.826%	+0.65%
16	34	⇈	Groovy	1.818%	+1.31%
17	14	↓	Visual Basic	1.761%	+0.23%
18	19	↑	R	1.684%	+0.64%
19	44	⇈	Go	1.625%	+1.37%
20	18	↓	PL/SQL	1.443%	+0.36%

Architettura a Runtime



Java Development Kit (JDK)

Esistono diverse distribuzioni di Java

JRE – Java Runtime Environment: esecuzione di programmi

JDK – JRE + tool per lo sviluppo di programmi (e.g. compilatore)

J2EE – Java Enterprise Edition: JDK + set esteso di librerie

Noi faremo riferimento al JDK

Include il necessario per eseguire applicazioni Java (JRE), i tool (e librerie) per sviluppare applicazioni e la relativa documentazione

I Principali Tool

javac – Compilatore Java

java – Java virtual machine, per eseguire applicazioni Java

javadoc – Per creare automaticamente documentazione in HTML

jar – Creazione di archivi compressi (anche eseguibili) contenenti bytecode e risorse

Outline

1. Java Overview
2. Richiamo: il file system
3. Richiamo: Interprete Comandi
4. Preparazione dell'ambiente di lavoro
5. Compilazione ed Esecuzione da Riga di Comando
6. Sviluppo e Test di Semplici Classi
   - Esercizi da Svolgere in Autonomia
7. Esercizi Aggiuntivi

Elementi base del file system

- I sistemi operativi odierni consentono di memorizzare permanentemente le informazioni su supporti di memorizzazione di massa (dischi magnetici, dispositivi a stato solido), unità ottiche (CD, DVD, Blu-Ray), memory stick, ecc...
- Le informazioni su questi supporti sono organizzate in file e cartelle:
  - i file contengono le informazioni
  - le cartelle sono contenitori, all'interno contengono i file ed altre cartelle
- La cartella più esterna, che contiene tutte le altre, è detta root. Essa rappresenta il livello gerarchico più alto del file system
  - In *nix (Linux, MacOS, BSD, Solaris...), vi è una unica radice, ossia `/`
  - In Windows, ciascun file system ha come root una lettera di unità (e.g. `C:`, `D:`)
- La stringa che descrive un intero percorso dalla root fino ad un elemento del file system prende il nome di *percorso* (e.g. `C:\Windows\win32.dll`, `/home/user/frameworkFS.jar`)

Manipolare il file system

L'utente può osservare e manipolare il file system:

- sapere quali files e cartelle contiene una cartella
- creare nuovi files e cartelle
- spostare file e cartelle dentro altre cartelle
- rinominare files e cartelle
- eliminare files e cartelle

Il software che consente di osservare e manipolare il file system prende il nome di file manager.

- Su Windows, esso è "Esplora risorse" (`explorer.exe`)
- Su MacOS, il principale è "Finder"
- Su Linux (e Android) ne esistono diversi (Nautilus, Dolphin, Thunar, Astro...)

Outline

1. Java Overview
2. Richiamo: il file system
3. Richiamo: Interprete Comandi
4. Preparazione dell'ambiente di lavoro
5. Compilazione ed Esecuzione da Riga di Comando
6. Sviluppo e Test di Semplici Classi
   - Esercizi da Svolgere in Autonomia
7. Esercizi Aggiuntivi

Interprete Comandi

Programma che permette di interagire con il S.O. mediante comandi impartiti in modalità testuale (non grafica), via linea di comando

- Nell'antichità (in termini informatici) le interfacce grafiche erano sostanzialmente inesistenti, e l'interazione con i calcolatori avveniva di norma tramite interfaccia testuale
- Tutt'oggi, le interfacce testuali sono utilizzate:
  - per automatizzare le operazioni
  - per velocizzare le operazioni (scrivere un comando è spesso molto più veloce di andare a fare click col mouse in giro per lo schermo)
  - per fare operazioni complesse con pochi semplici comandi
  - non tutti i software sono dotati di interfaccia grafica
  - alcune opzioni di configurazione del sistema operativo restano accessibili solo via linea di comando
    - (anche su Windows: ad esempio i comandi per associare le estensioni ad un eseguibile)

Lo vedrete in maniera esaustiva nel corso di Sistemi Operativi...

Sistemi *nix (Linux, MacOS X, FreeBSD, Minix...)

Nei sistemi UNIX esistono vari tipi di interpreti, chiamati shell

Alcuni esempi

- Bourne shell (sh)
  - Prima shell sviluppata per UNIX (1977)
- C-Shell (csh)
  - Sviluppata da Bill Joy per BSD
- Bourne Again Shell (bash)
  - Parte del progetto GNU, è un super set di Bourne shell
- ...

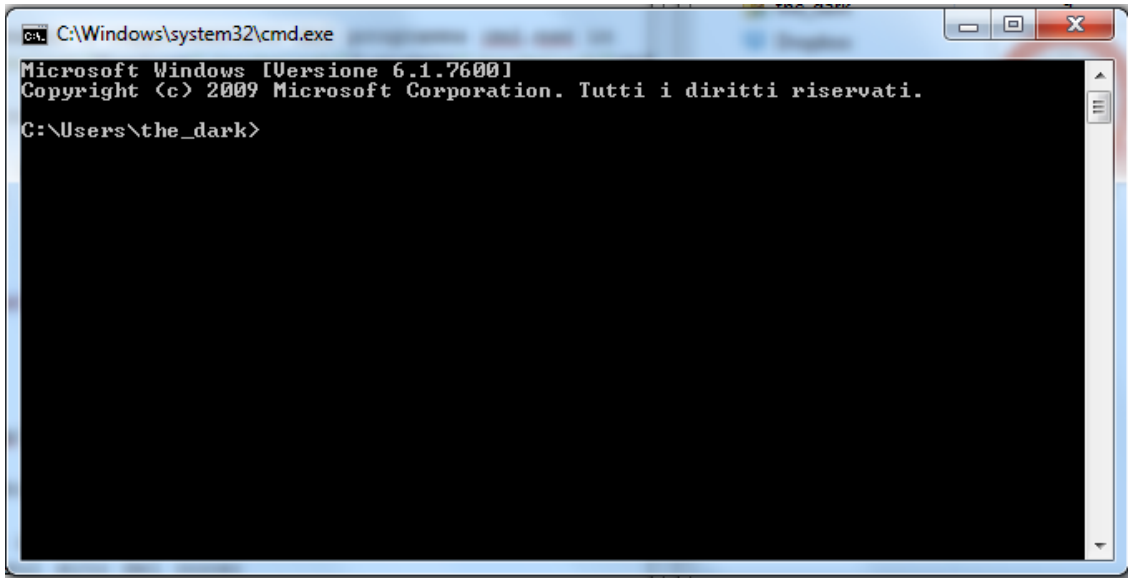
Per una panoramica completa delle differenze

`http://www.faqs.org/faqs/unix-faq/shell/shell-differences/`

Sistemi Windows

L'interprete comandi è rappresentato dal programma `cmd.exe` in `C:\Windows\System32\cmd.exe`

- Eredita in realtà sintassi e funzionalità della maggior parte dei comandi del vecchio MSDOS
- Versioni recenti hanno introdotto PowerShell, basato su .NET e C#
- Windows 10 ha introdotto il supporto a bash tramite Linux Subsystem for Linux



Aprire un terminale in laboratorio

- In laboratorio, troverete il terminale (prompt dei comandi) clickando su Start ⇒ Programmi ⇒ Accessori ⇒ Prompt dei comandi
- Metodo più rapido: Start ⇒ Nella barra di ricerca, digitare `cmd` ⇒ clickare su `cmd.exe`

File system e terminale: cheat sheet

Operazione	Comando Unix	Comando Win
Visualizzare la directory corrente	`pwd`	`echo %cd%`
Eliminare il file `f` (non va con le cartelle!)	`rm f`	`del f`
Eliminare la directory `nd`	`rm -r nd`	`rd nd`
Contenuto della directory corrente	`ls -alh`	`dir`
Cambiare unità disco (passare a D:)	–	`D:`
Passare alla directory `nd`	`cd nd`	`cd nd`
Passare alla directory di livello superiore	`cd ..`	`cd..`
Spostare (rinominare) un file `f1` in `f2`	`mv f1 f2`	`move f1 f2`
Copiare il file `f` in `fc`	`cp f fc`	`copy f fc`
Creare la directory `d`	`mkdir d`	`md d`

Eseguire delle prove ed esser certi di aver compreso come utilizzare ogni comando. Per *cominciare* l'esame, in particolare, dovrete usare il comando `cd`: siate certi di aver capito cosa fa!

Uso intelligente del terminale

Autocompletamento

Sia *nix che Windows offrono la possibilità di effettuare autocompletamento, ossia chiedere al sistema di provare a completare un comando. Per farlo si utilizza il tasto "tab" (quello con due frecce orientate in maniera opposta, sopra il lucchetto).

Memoria dei comandi precendenti

Sia *nix che Windows offrono la possibilità di richiamare rapidamente i comandi inviati precedentemente premendo il tasto "freccia su". I sistemi *nix supportano anche il lancio di comandi eseguiti in sessioni precedenti (non perde memoria col riavvio del terminale).

Interruzione di un programma

È possibile interrompere forzatamente un programma (ad esempio perché inloopato). Per farlo, sia su Windows che in *nix, si prema ctrl+c.

Ricerca nella storia dei comandi precedenti

Premendo ctrl+r seguito da un testo da cercare, i sistemi *nix supportano la ricerca all'interno dei comandi lanciati recentemente, anche in sessioni utente precedenti. Non disponibile su Windows.

Outline

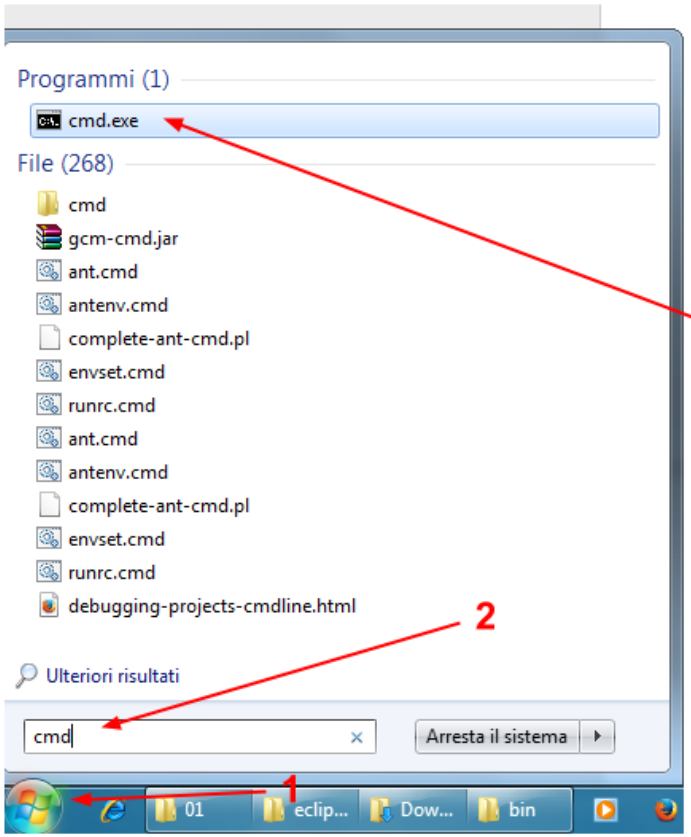
1. Java Overview
2. Richiamo: il file system
3. Richiamo: Interprete Comandi
4. Preparazione dell'ambiente di lavoro
5. Compilazione ed Esecuzione da Riga di Comando
6. Sviluppo e Test di Semplici Classi
   - Esercizi da Svolgere in Autonomia
7. Esercizi Aggiuntivi

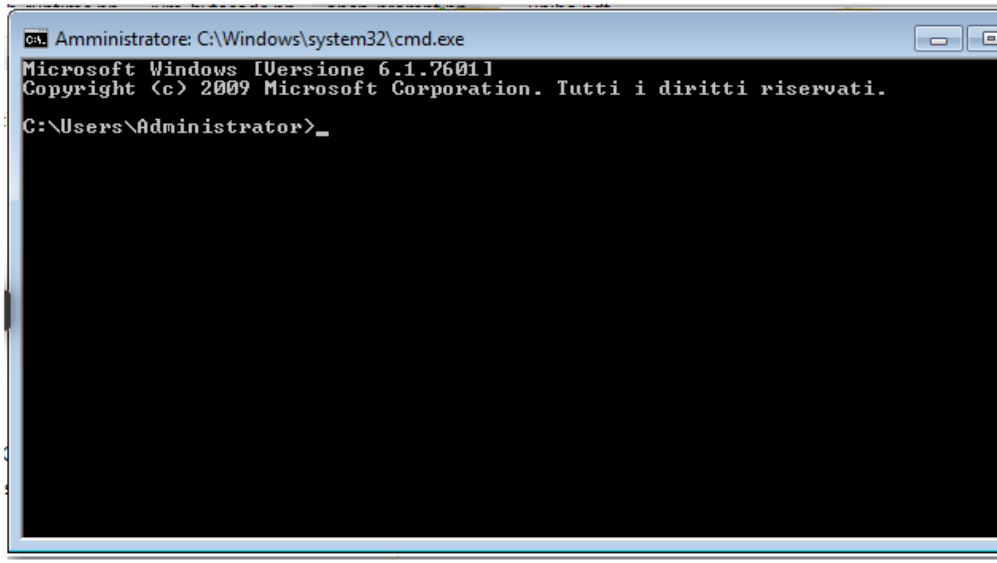
Preparazione Ambiente di Lavoro 1/3

- Accendere il PC
- Loggarsi sul sito del corso
  - https://elearning-cds.unibo.it/course/view.php?id=11689
- Scaricare dalla sezione dedicata a questa settimana il materiale dell'esercitazione odierna
- Spostare il file scaricato sul Desktop
- Decomprimere il file usando 7zip (o un programma analogo) sul Desktop

Preparazione Ambiente di Lavoro 2/3

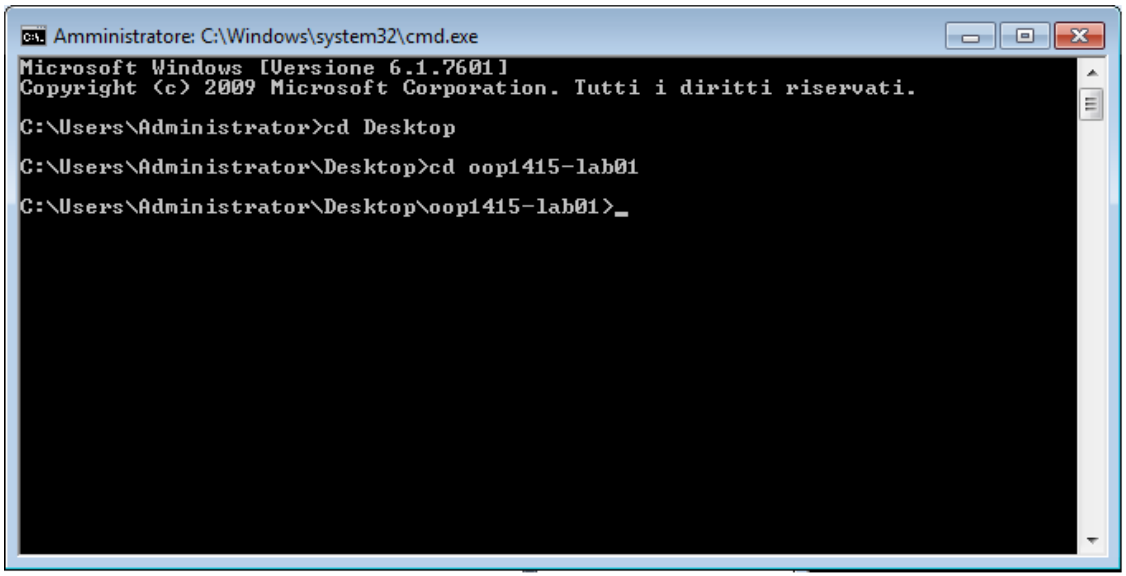
- Aprire il prompt dei comandi





Preparazione Ambiente di Lavoro 3/3

- Posizionarsi all'interno della cartella scompattata con l'ausilio del comando `cd` (change directory)
  1. `cd Desktop` e premere invio, dopodiché
  2. `cd lab01` e premere invio



Outline

1. Java Overview
2. Richiamo: il file system
3. Richiamo: Interprete Comandi
4. Preparazione dell'ambiente di lavoro
5. Compilazione ed Esecuzione da Riga di Comando
6. Sviluppo e Test di Semplici Classi
   - Esercizi da Svolgere in Autonomia
7. Esercizi Aggiuntivi

Compilazione ed Esecuzione (base) 1/2

- Aprire il programma JEdit
  - Si tratta di un editor di testo pensato per chi scrive codice
  - Supporta vari linguaggi tramite plug-in
  - Sviluppato in Java
  - Multipiattaforma
  - Sarà il nostro editor per questa prima lezione
  - In futuro useremo un Integrated Development Environment (IDE)
- Da JEdit aprire la classe `HelloWorld.java`
- Studiare brevemente il comportamento della classe prima di procedere con i passi successivi

Compilazione ed Esecuzione (base) 2/2

Compilazione di una classe (comando javac)

- `javac NomeSorgente.java`
- `javac *.java` compila tutti i sorgenti nella directory corrente
- Compiliamo `HelloWorld.java` da riga di comando
  - `javac HelloWorld.java`

Esecuzione di un programma Java (comando java)

- `java NomeSorgente`
- Eseguiamo `HelloWorld` da riga di comando
  - `java HelloWorld`

Warm-up: Prime Modifiche

Modificare la classe HelloWorld seguendo i passi riportati qui sotto

1. Sostituire la stampa a video con un messaggio a piacere
2. Ricompilare ed eseguire nuovamente il programma per verificarne il corretto comportamento
3. Aggiungere alla fine del messaggio modificato il risultato della computazione 50*50
4. Ricompilare ed eseguire nuovamente il programma per verificarne il corretto comportamento

In caso di problemi/dubbi fare riferimento alle slide del modulo `02-Oggetti e classi`

Outline

1. Java Overview
2. Richiamo: il file system
3. Richiamo: Interprete Comandi
4. Preparazione dell'ambiente di lavoro
5. Compilazione ed Esecuzione da Riga di Comando
6. Sviluppo e Test di Semplici Classi
   - Esercizi da Svolgere in Autonomia
7. Esercizi Aggiuntivi

Outline

1. Java Overview
2. Richiamo: il file system
3. Richiamo: Interprete Comandi
4. Preparazione dell'ambiente di lavoro
5. Compilazione ed Esecuzione da Riga di Comando
6. Sviluppo e Test di Semplici Classi
   - Esercizi da Svolgere in Autonomia
7. Esercizi Aggiuntivi

Modalità di Lavoro

1. Leggere la consegna
2. Risolvere l'esercizio autonomamente, cercando di risolvere da soli eventuali piccoli problemi che possono verificarsi durante lo svolgimento degli esercizi
3. Contattare i docenti nel caso vi troviate a lungo bloccati nella risoluzione di uno specifico esercizio
4. A esercizio ultimato contattare i docenti per un rapido controllo della soluzione realizzata
5. Proseguire con l'esercizio seguente

Test Scope delle Variabili

1. Analizzare il sorgente `TestScopes.java`
2. Prima di compilare e lanciare il programma riflettere sul comportamento dei metodi della classe e provare a prevedere l'output di ogni singola stampa
3. Compilare ed eseguire il programma. L'output ottenuto corrisponde a quanto preventivato?
4. In caso di risultato diverso da quanto aspettato (in questo caso e in tutti i successivi)
   - 4.1 Provare a ragionare e a trovare in autonomia una spiegazione
   - 4.2 Se non è possibile risolvere i dubbi in maniera autonoma, contattare il docente

Costruzione di Semplici Classi 1/5

Completare la classe `Student` caratterizzata da

- Campi
  - `String name`
  - `String surname`
  - `int id` (rappresenta la matricola)
  - `int matriculationYear`
- Metodi da implementare
  - `build` (inizializza `Student` con i parametri forniti in input)
    - Input: `String name, String surname, int id, int matriculationYear`
  - `printStudentInfo` (stampa in standard output le informazioni relative allo studente)

Infine, testare la classe completando il `main` secondo le linee guida riportate nei commenti del sorgente

Costruzione di Semplici Classi 2/5

Completare la classe `ComplexNum` caratterizzata da

- Campi
  - `double re` (parte reale), `double im` (parte immaginaria)
- Metodi (oltre al main)
  - `build` (inizializza `ComplexNum` con i paramtri forniti in input)
    - Input: `double re, double im`
  - equal (determina se il numero complesso è = a quello fornito in input)
    - Input: `ComplexNum num`
    - Returns: `boolean`
  - add (aggiunge il `ComplexNum` fornito in input)
    - Input: `ComplexNum num`
  - `toStringRep` (restituisce una rappresentazione testuale del numero)
    - Returns: `String`

Completare e testare la classe seguendo le linee guida riportate nei commenti del sorgente

Costruzione di Semplici Classi 3/5 (A)

Implementare da zero la classe `Calculator` che realizzi il comportamento di una semplice calcolatrice in grado di lavorare su numeri `double`

Caratteristiche della classe `Calculator`:

- Campi: - (nessuno) (il che la rende una classe "degenere" dal punto di vista dell'OOP)
- Metodi (oltre al `main`)
  - `add/sub/mul/div`
    - Input: `double n1, double n2`
    - Returns: `double` (risultato dell'operazione)

Testing della classe

Infine, testare la classe completando il `main` secondo le linee guida riportate nella slide che segue

Costruzione di Semplici Classi 3/5 (B)

Test della classe `Calculator`

```
1 /*
2      * Testare la classe come segue:
3      *
4      * 1) Effettuare la somma 1+2  (metodo add);
5      * 2) Effettuare la sottrazione -1 - (+2) (metodo sub)
6      * 3) Effettuare la moltiplicazione 6*3 (metodo mul);
7      * 4) Effettuare la divisione 8 / 4 (medodo div)
8  */
```

Costruzione di Semplici Classi 4/5

Modificare la classe `Calculator` costruita in precedenza

- Aggiungere due campi
  - `int nOpDone` (n. di operazioni compiute dalla calcolatrice)
  - `double lastRes` (ultimo risultato computato)
- Inizializzare i campi a zero nel metodo `build`
  - Il metodo dovrà avere zero parametri di input
- Modificare i metodi `add`/`sub`/`mul`/`div` tenendo conto dei due nuovi campi aggiunti alla classe

Verificare il corretto funzionamento delle modifiche utilizzando la calcolatrice come in precedenza e stampando di tanto in tanto in standard output il valore dei campi `nOpDone` e `lastRes`

Costruzione di Semplici Classi 5/5 (A)

Implementare una classe Java che modelli il concetto di treno

Caratteristiche della classe `Train` - Campi

- `int nTotSeats` (n. posti totali del treno)
- `int nFirstClassSeats` n. posti in prima classe
- `int nSecondClassSeats` n. posti in seconda classe
- `int nFirstClassReservedSeats` n. di posti prenotati in prima classe
- `int nSecondClassClassReservedSeats` n. di posti prenotati in seconda classe

Costruzione di Semplici Classi 5/5 (B)

Caratteristiche della classe `Train` - Metodi

- `build` per l'inizializzazione opportuna dei campi. Nota: valutare quale sia l'input minimo necessario per il metodo build.)
- `reserveFirstClassSeats`
  - Input: `int nSeats`
- `reserveSecondClassSeats`
  - Input: `int nSeats`
- `getTotOccupancyRatio`
  - Returns: `double` (percentuale globale di posti occupati)
- `getFirstClassOccupancyRatio`
  - Returns: `double` (percentuale posti occupati in prima classe)
- `getSecondClassOccupancyRatio`
  - Returns: `double` (percentuale posti occupati in seconda classe)
- `deleteAllReservations` (azzera tutti i posti prenotati)

Costruzioni di Semplici Classi 5/5 (C)

Testing

- Da realizzare all'interno della classe `UseTrain`, fornita tra il materiale di questa prima esercitazione
- Seguire le linee guida fornite nei commenti del `main`
- Utilizzare sempre valori numerici consistenti con i vincoli scelti (vedere commenti)

Nota

Gestire la conversione da `int` a `double` come siete abituati a fare in C per quanto riguarda il calcolo delle percentuali di posti occupati

Outline

1. Java Overview
2. Richiamo: il file system
3. Richiamo: Interprete Comandi
4. Preparazione dell'ambiente di lavoro
5. Compilazione ed Esecuzione da Riga di Comando
6. Sviluppo e Test di Semplici Classi
   - Esercizi da Svolgere in Autonomia
7. Esercizi Aggiuntivi

Una Calcolatrice per Numeri Complessi

Completare e testare una classe Java che realizzi il comportamento di una semplice calcolatrice in grado di lavorare su numeri complessi rappresentati dalla classe `ComplexNum`

Caratteristiche della classe `ComplexNumCalculator`:

- Campi `int nOpDone, ComplexNum lastRes`
- Metodi (oltre ai soliti `main` e `build`)
  - `sum/sub/mul`
    - Input: `ComplexNum n1, ComplexNum n2`
    - Returns: `ComplexNum` (risultato dell'operazione)

Un Riconoscitore della Sequenza Ripetuta 1234

Classe `Recognizer`

Completare e testare una classe Java che sia in grado di riconoscere la sequenza 1234 ripetuta (potenzialmente all'infinito)

Traccia Risolutiva

- Implementare un metodo diverso per gestire il riconoscimento di ciascun carattere (`check1`, `check2`, `check3`,..)
- Tenere traccia all'interno della classe delle informazioni di stato relative al prossimo carattere da riconoscere
- Implementare il metodo `nextExpectedInt` che restituisce il prossimo intero atteso nella sequenza
- Implementare il metodo `reset` che consente di resettare lo stato corrente del riconoscitore

Bibliography I