

Programmazione 2

Marco Ronchetti

Dipartimento di Ingegneria e Scienza dell'Informazione
University of Trento, Italy

marco.ronchetti@unitn.it
<http://latemar.science.unitn.it>

Slides condivise con i proff. Giampietro Picco, Maurizio Marchese

Programming “in the large”

Comune nella pratica industriale, caratterizzata da:

- Suddivisione del lavoro tra persone/ gruppi
(*divide et impera*)
- Manutenibilità
(che succede se voglio cambiare qualcosa tra un mese/ un anno/ ...)
- Robustezza
(che succede se sostituisco una persona?)

Programming "in the large"

Le risposte:

Ingegneria del software
(corso del prossimo anno)

Buone tecniche di programmazione
(es. commenti aggiornati)

Supporto dal linguaggio:
object-oriented programming
(Java, C++, ...)

Obiettivi del corso

- Il corso introduce le **tecniche e i costrutti della programmazione ad oggetti**
 - come evoluzione necessaria per affrontare il problema della crescente complessità degli artefatti software
- Verrà utilizzato il linguaggio Java
 - dopo alcuni richiami di C++, utili anche per confronto
- Il corso è prevalentemente teorico, ma contiene esercitazioni
 - principalmente **introduzione a strumenti per l'uso di Java**

Orario

Lezioni:

- Martedì 10:30 B107
- Mercoledì 15:30 B107

Esercitazioni SOLO QUANDO INDICATO

- Mercoledì 15:30 B107
- Giovedì 13:30 B106

Crediti e impegno

- Il modulo è di 6 crediti
- 1 credito = 25 ore di studio => 150 ore
- di queste, 48 ore sono in aula: quindi...
- ... per ogni ora di lezione in aula ...
- ... occorre studiare autonomamente fuori aula per ~2 ore

Modalità d'esame

- L'esame è costituito da due parti:
 - Prova scritta:
 - ~40 minuti
 - ~8 esercizi di lettura di codice
 - ~10 domande a risposta multipla
 - correzione immediata
 - Prova pratica:
 - sviluppo di codice ~3 - 4 ore

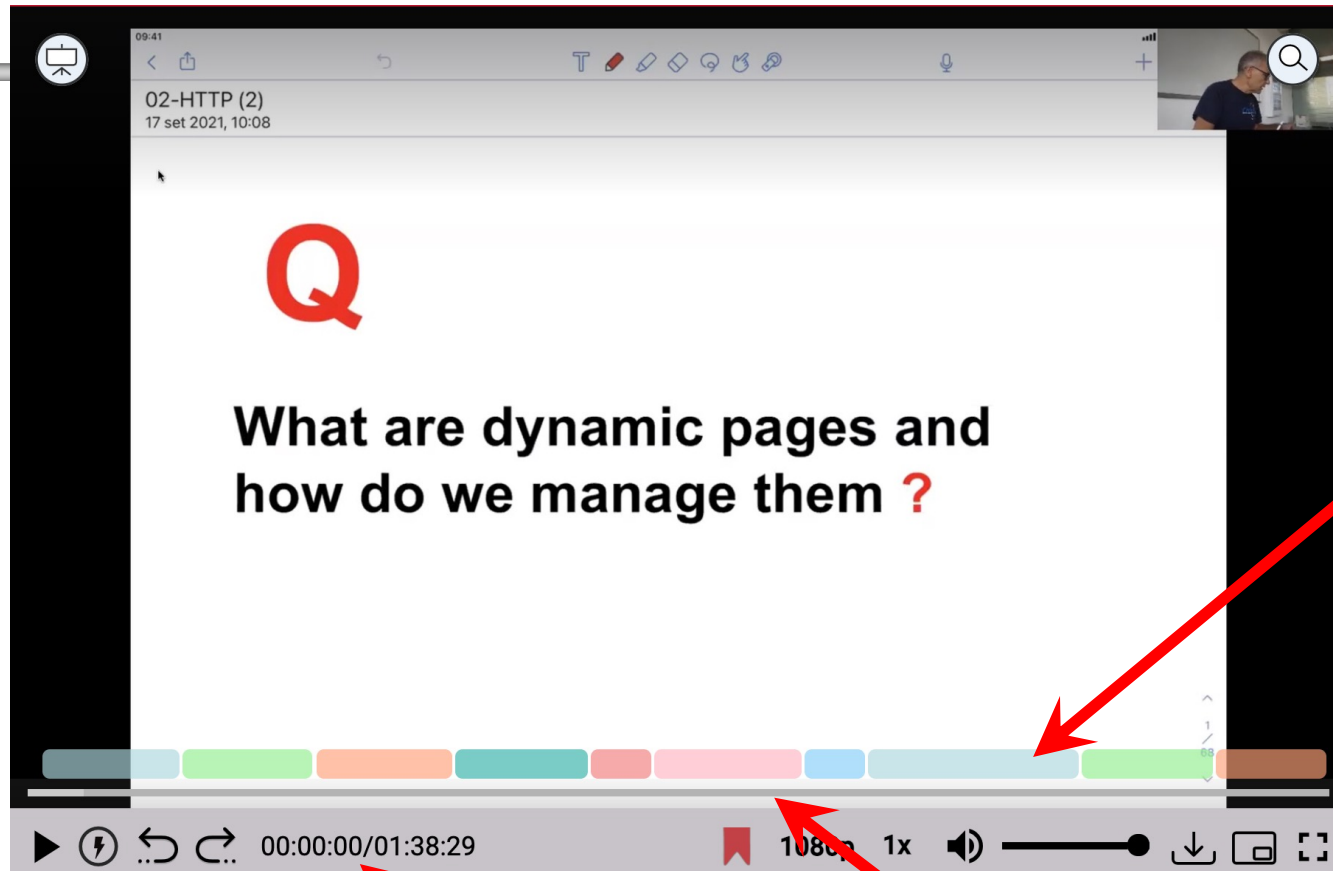
Modalità di superamento del modulo

- Le due prove vengono effettuate una dopo l'altra nella stessa giornata
- Il voto finale è la media aritmetica delle due prove, se sufficienti
 - ... altrimenti va ripetuto l'intero esame

Materiale didattico

- *Sito Web:*
`latemar.science.unitn.it`
- *Moodle -> accesso ai Video (Daddy)*
- *Piazza*

DADdy video quick guide



Navigation - 1

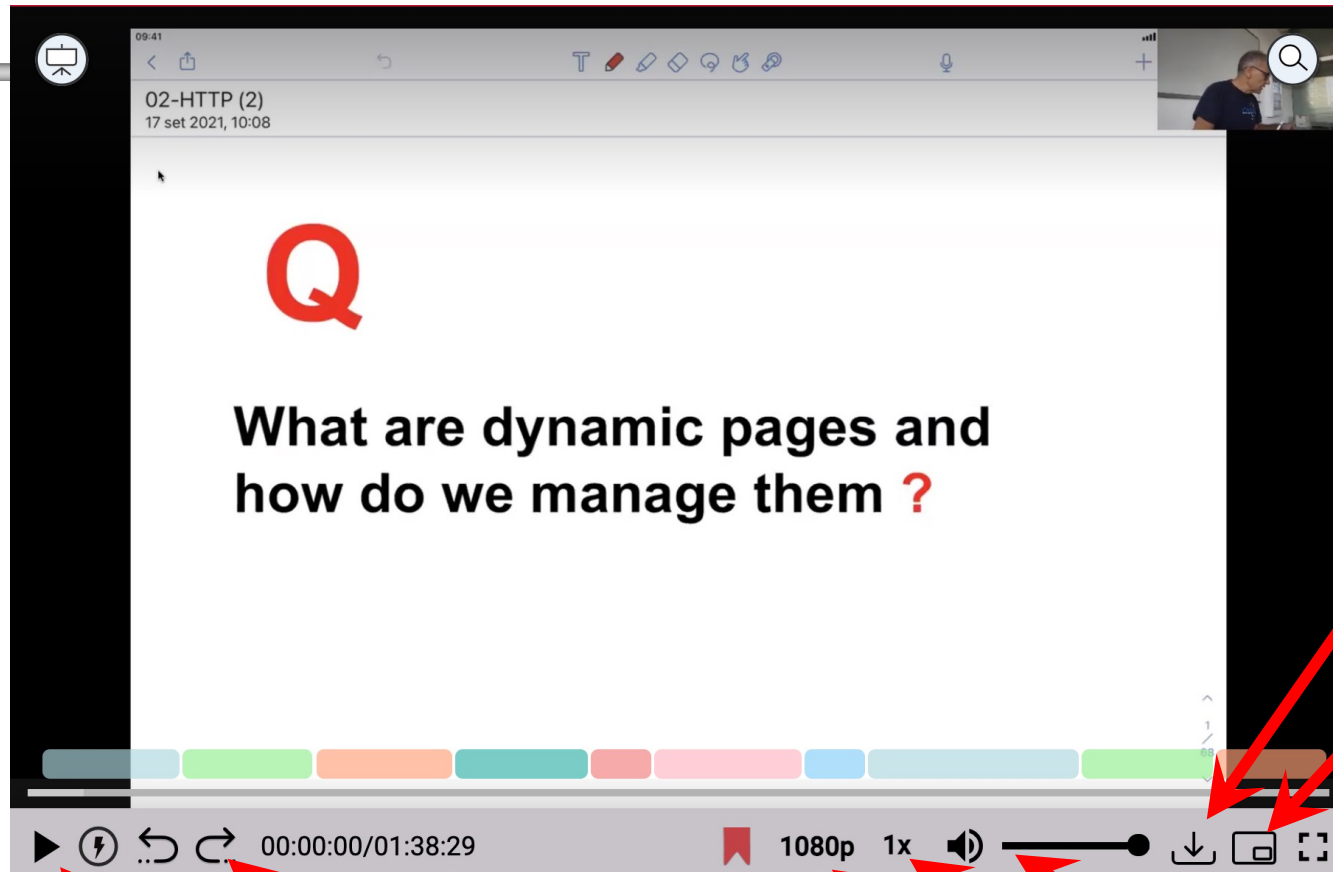
Colored segments correspond to different topics (as guessed by the A.I. engine)

Current time/Total time

Clicks on the time bar allow you to move there

DADdy video quick guide

Navigation -2



Download

Picture in Picture

Full Screen

Change volume

Change speed

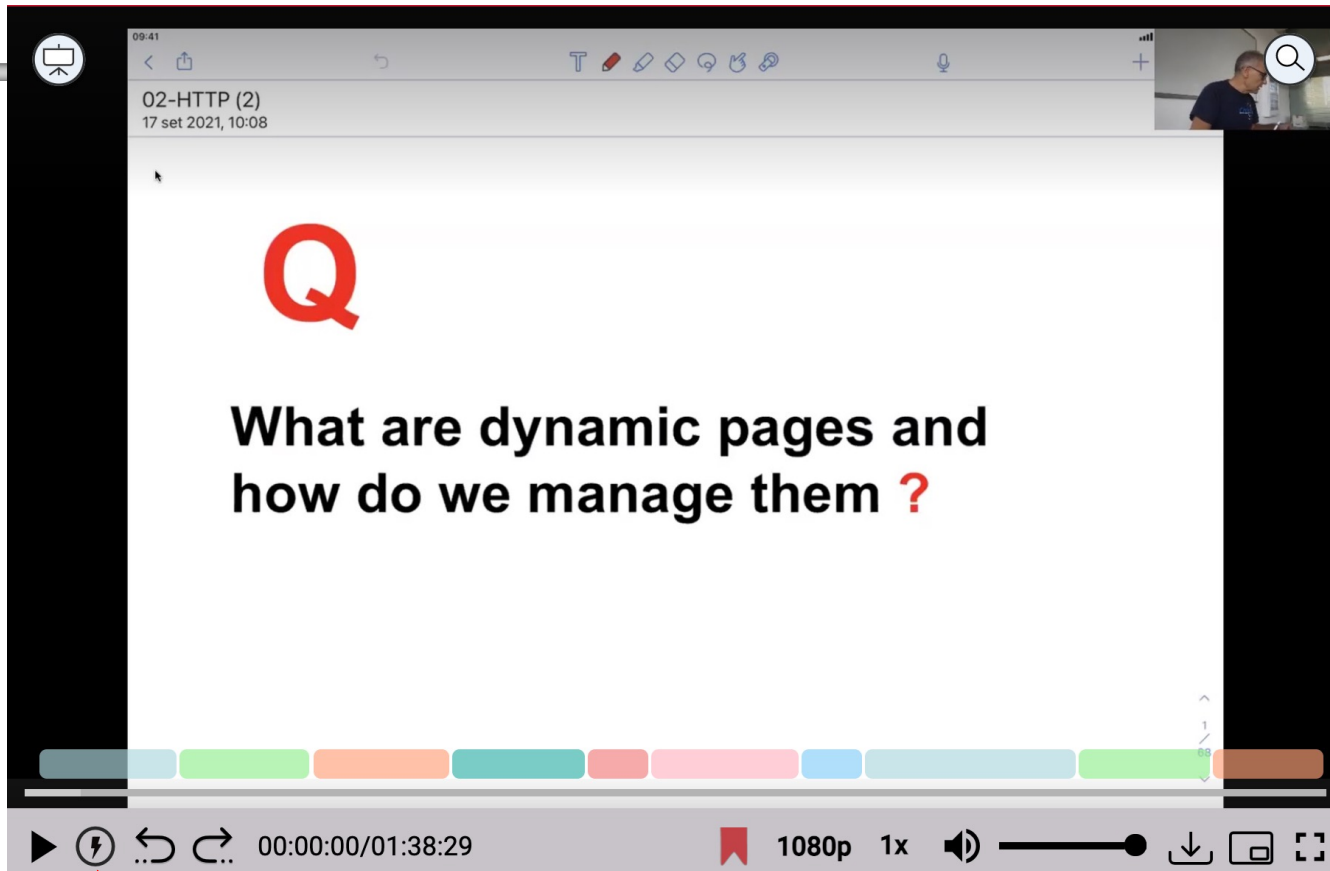
Change video resolution (for slow connections)

Go to beginning of next phrase

Go to beginning of previous phrase

Play/Pause

DADdy video quick guide



Enable/disable Turbo mode

Turbo mode

Turbo mode cuts all the silences in the video and changes speed to 1.25 to reduce the time needed to go through the presentation.

Of course you can then still change the playback speed.

Navigation by slide

Lecture 1: Intro to the course - HTTP

show me the list of slides

This is the list of all slides in the presentation. You can scroll to find what you are looking for.

The screenshot shows a presentation slide titled "Port" with a table of network ports. A red box highlights the table, and a red arrow points to the "show me the list of slides" text. Another red arrow points to the "Go to this slide" text. A third red arrow points to the "Show me this slide" text. The slide content includes a definition of a port, a list of common ports, and a table of ports.

Port
A port is an **endpoint of communication** in an operating system.
A process associates its input or output channels, via an Internet socket, with a transport protocol, a port number, and an IP address.
This process is known as **binding**.

socket: (protocol, local address, local port, remote address, remote port)
Mistranslated into Italian as "Porta" (door)

Common ports:

- HTTP on port 80
- HTTP with SSL (HTTPS) on port 443
- FTP on port 21
- SMTP on port 25
- POP on port 110
- SSH on port 22

PID	PORT	IP	Protocol
84	21	193.205.196.130	FTP
78	80	193.205.196.130	HTTP
321	8080	193.205.196.130	HTTP
341	25	193.205.196.130	SMTP

Mistranslated into Italian as "Porta" (door)

Go to this slide

Show me this slide

DADdy video quick guide

Text search

You can search in the lecture transcripts either by one of the keywords defined by the system, or by the text that you freely enter.

Here you can enter the word you search for in the lecture

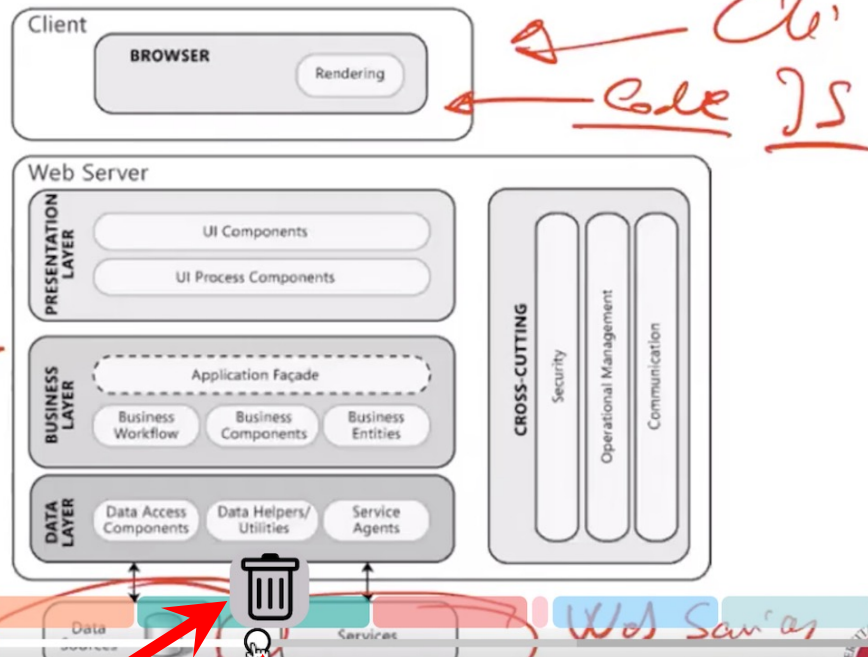
List of keywords defined by the system

Occurrences found in transcript

The screenshot shows a video player interface. The main content area displays a diagram titled "URL structure" with components like scheme, host, path, query, and fragment. Below the diagram, there are examples of URLs: `https://en.wikipedia.org/wiki/URL` and `https://en.wikipedia.org/w/index.php?title=`. A search overlay is visible on the right side of the video player, featuring a search bar and a list of keywords: protocol, url, https, watch, host, scheme, search, issue, and page. Each keyword has a red circle next to it indicating its frequency. The video player controls at the bottom show a progress bar, a play button, and a volume icon. Red arrows point from the text annotations to the search bar, the keyword list, and the video player controls.

DADdy video quick guide

(Traditional) Logical Web Archite



Adding personal bookmarks

00:40:45/01:39:58

720p

1x



Delete personal bookmark

Add personal bookmark

The circle is my personal bookmark.
Click on it to move to it.

Un'occhiata a Java

Java Sneak Peek

Java e C/C++: cosa resta uguale?

Istruzioni e commenti

```
int a=5%3; // commento
```

Strutture di controllo

```
for {int a=2; a<10; a++ {...}
```

```
do {...} while (i<10);
```

```
while (i<10) {...}
```

```
switch (s) { case 1: ...; defeult:... }
```

```
if (i<10){...} elseif (i>200) {...} else {...}
```

Definizione e chiamate di funzioni

```
int f(int k) { return k+2;}
```

```
f(3); // ma niente puntatori e indirizzi!
```

Se volete...

<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html>

Java e C/C++: cosa è diverso?

Varie cose. Ad esempio:

- arrays
- implementazione dei tipi (qual'è il massimo intero in C/C++ ?)
- il main
- assenza di variabili globali
- struct, union
- ...

Java e C/C++: cosa è diverso?

- Java **toglie** al C alcune caratteristiche complesse e potenzialmente “pericolose” (es. puntatori)
- Java **aggiunge** al C le caratteristiche di un linguaggio object-oriented
 - es. classi, ereditarietà, polimorfismo
- Java **introduce** una gerarchia di classi predefinite che in pratica diventano parte del linguaggio:
 - gestione dell'I/O
 - gestione di stringhe
 - gestione delle eccezioni
 - networking
 - concorrenza (**Thread**)
 - strutture dati (es. **Vector**, **Dictionary**, **Hashtable**)

Perché è diverso?

- Molti errori comuni di programmazione sono legati alla gestione della memoria tramite *puntatori*:
 - puntatori che puntano a locazioni illecite (non allocate)
 - puntatori che puntano a locazioni lecite ma sbagliate
 - memoria allocata e non più rilasciata (*memory leaks*)
- Soluzioni in Java:
 - *Abolizione dei puntatori*: di conseguenza, **non** fornisce gli operatori di de-referenziazione e addressing *****, **->**, **&**
 - *Garbage collection*: gli oggetti vengono allocati e deallocati automaticamente in memoria: **sizeof**, **malloc**, **free** non sono più necessari

Hello World in Java

```
class HelloWorld {  
    /* Hello World, my first  
       Java application */  
    public static void main (String args[]) {  
        System.out.println("Hello World!");  
        // qui va il resto  
        // del programma principale  
    }  
}
```

versione minimale...

Uso di Java (da terminale)

Compilazione:

>**javac HelloWorld.java**

produce **HelloWorld.class**

(in realtà: un file **.class** per ogni classe nel sorgente)

obbligatorio
specificare
l'estensione!

Esecuzione...

>**java HelloWorld**

la classe indicata deve contenere il **main**

obbligatorio
omettere
l'estensione!

La struttura di un programma Java

- Un programma Java è organizzato come un insieme di **classi**
- Classi diverse possono essere raggruppate all'interno della stessa "*compilation unit*"
 - Es. un file **.java**
- Il programma principale è rappresentato da una funzione speciale (**main**) della classe il cui nome coincide con il nome del programma

Qualche anticipazione...

```
class HelloWorld {  
    /* Hello World, my first Java application */  
    public static void main (String args[]) {  
        int i;  
        for (i=1;i<10;i++)  
            if (i%2 == 0)  
                System.out.println(i);  
        System.out.println("finito!");  
    }  
}
```

Class **String**

java.lang

Class **String**

[java.lang.Object](#)

|

+-java.lang.String

All Implemented Interfaces:

[CharSequence](#), [Comparable](#), [Serializable](#)

public final class **String**

extends [Object](#)

implements [Serializable](#), [Comparable](#), [CharSequence](#)

The `String` class represents character strings. All string literals in Java programs, such as `"abc"`, are implemented as instances of this class.

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:

```
String str = "abc";
```

is equivalent to:

```
char data[] = {'a', 'b', 'c'};  
String str = new String(data);
```

molto più semplice e intuitivo
di **char[]** e **char***!

Lettura di stringhe con GUI

```
import javax.swing.JOptionPane;  
class Applicazione {  
...  
String input =  
    JOptionPane.showInputDialog("How are you?");  
System.out.println(input);  
...  
    System.exit(1);  
...  
}
```

inclusione: **import** non ha bisogno di pre-compilatori, **#ifdef**, etc.

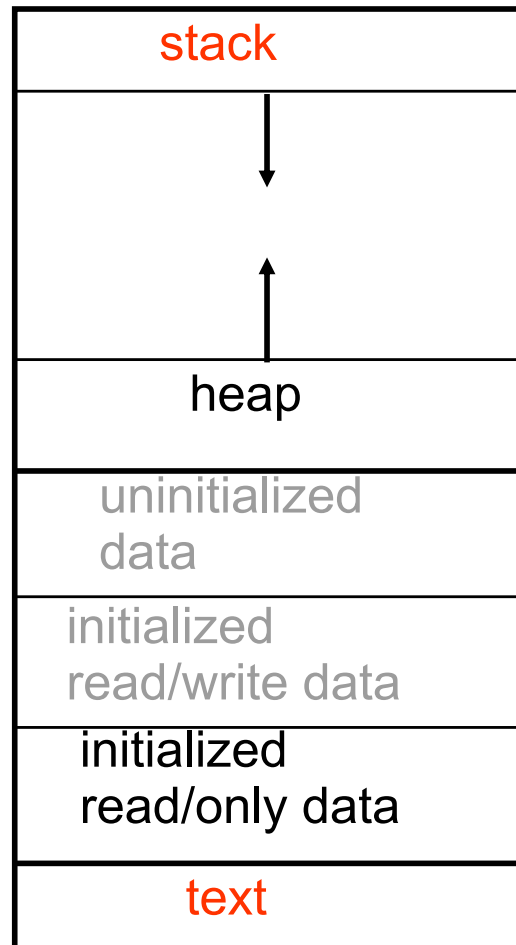
Essenziale! Altrimenti il *thread* che gestisce la GUI rimane vivo, e il processo non termina...



Richiami di C++ di base

Richiami di C++ di base Parte 1

Il modello di memoria



*memoria allocata dalle funzioni
(Variabili automatiche)*

*memoria allocata dinamicamente
dal programmatore*

Variabili globali e statiche

<- questo é supportato solo da alcuni hardware

Codice eseguibile

Modularizzazione: Funzioni

Esempio

```
int somma(int a, int b) {
    int res;
    res=a+b;
    return res;
}
int prodotto(int b, int a) {
    int res=0;
    for (int k=0; k<b; k++)
        res=somma(res,a);
    return res;
}
main() {
    int a,b,res;
    cout << "dammi due numeri \n";
    cin >> a >> b;
    res=prodotto(a,b);
    cout << a << " * " << b << " =
" << res << "\n";
}
```

stack

a	2
b	3
res	?

main

0
4
8
12
16
20
24
28
32
36
40
44
...

heap

Modularizzazione: Funzioni

Esempio

```
int somma(int a, int b) {
    int res;
    res=a+b;
    return res;
}

int prodotto(int b, int a) {
    int res=0;
    for (int k=0; k<b; k++)
        res=somma(res,a);
    return res;
}

main() {
    int a,b,res;
    cout << "dammi due numeri \n";
    cin >> a >> b;
    res=prodotto(a,b);
    cout << a << " * " << b << " =
" << res << "\n";
}
```

stack

a	2	0	main
b	3	4	
res	?	8	
a	3	12	prodotto
b	2	16	
res	0	20	
k	0	24	
		28	
		32	
		36	
		40	
		44	
		...	
heap			

Modularizzazione: Funzioni

Esempio

```
int somma(int a, int b) {
    int res;
    res=a+b;
    return res;
}
int prodotto(int b, int a) { stack
    int res=0;
    for (int k=0; k<b; k++)
        res=somma(res,a);
    return res;
}
main() {
    int a,b,res;
    cout << "dammi due numeri \n";
    cin >> a >> b;
    res=prodotto(a,b);
    cout << a << " * " << b << " =
" << res << "\n";
}
```

a	2	0	main
b	3	4	
res	?	8	
a	3	12	prodotto
b	2	16	
res	0	20	
k	0	24	
a	0	28	somma
b	3	32	
res	?	36	
		40	
		44	
heap		...	

Modularizzazione: Funzioni

Esempio

```
int somma(int a, int b) {
    int res;
    res=a+b;
    return res;
}
int prodotto(int b, int a) { stack
    int res=0;
    for (int k=0; k<b; k++)
        res=somma(res,a);
    return res;
}
main() {
    int a,b,res;
    cout << "dammi due numeri \n";
    cin >> a >> b;
    res=prodotto(a,b);
    cout << a << " * " << b << " =
" << res << "\n";
}
```

a	2	0	main
b	3	4	
res	?	8	
a	3	12	prodotto
b	2	16	
res	0	20	
k	0	24	
a	0	28	somma
b	3	32	
res	3	36	
		40	
		44	
heap		...	

Modularizzazione: Funzioni

Esempio

```
int somma(int a, int b) {
    int res;
    res=a+b;
    return res;
}
int prodotto(int b, int a) {
    int res=0;
    for (int k=0; k<b; k++)
        res=somma(res,a);
    return res;
}
main() {
    int a,b,res;
    cout << "dammi due numeri \n";
    cin >> a >> b;
    res=prodotto(a,b);
    cout << a << " * " << b << " =
" << res << "\n";
}
```

stack

a	2	0	main
b	3	4	
res	?	8	
a	3	12	prodotto
b	2	16	
res	3	20	
k	1	24	
a	0	28	
b	3	32	
res	3	36	
		40	
		44	
		...	
heap			

Modularizzazione: Funzioni

Esempio

```

int somma(int a, int b) {
    int res;
    res=a+b;
    return res;
}
int prodotto(int b, int a) { stack
    int res=0;
    for (int k=0; k<b; k++)
        res=somma(res,a);
    return res;
}
main() {
    int a,b,res;
    cout << "dammi due numeri \n";
    cin >> a >> b;
    res=prodotto(a,b);
    cout << a << " * " << b << " =
" << res << "\n";
}

```

a	2	0	main
b	3	4	
res	?	8	
a	3	12	prodotto
b	2	16	
res	3	20	
k	1	24	
a	3	28	somma
b	3	32	
res	6	36	
		40	
		44	
heap		...	

Modularizzazione: Funzioni

Esempio

```
int somma(int a, int b) {
    int res;
    res=a+b;
    return res;
}

int prodotto(int b, int a) {
    int res=0;
    for (int k=0; k<b; k++)
        res=somma(res,a);
    return res;
}

main() {
    int a,b,res;
    cout << "dammi due numeri \n";
    cin >> a >> b;
    res=prodotto(a,b);
    cout << a << " * " << b << " =
" << res << "\n";
}
```

stack

a	2	0	main
b	3	4	
res	?	8	
a	3	12	prodotto
b	2	16	
res	6	20	
k	1	24	
a	3	28	
b	3	32	
res	6	36	
		40	
		44	
heap		...	

Modularizzazione: Funzioni

Esempio

```
int somma(int a, int b) {
    int res;
    res=a+b;
    return res;
}
int prodotto(int b, int a) {
    int res=0;
    for (int k=0; k<b; k++)
        res=somma(res,a);
    return res;
}
main() {
    int a,b,res;
    cout << "dammi due numeri \n";
    cin >> a >> b;
    res=prodotto(a,b);
    cout << a << " * " << b << " =
" << res << "\n";
}
```

stack

a	2	0
b	3	4
res	6	8
a	3	12
b	2	16
res	6	20
k	1	24
a	3	28
b	3	32
res	6	36
		40
		44
		...

main

heap

Funzioni ricorsive

Una funzione può richiamare se stessa.

```
int fact(int n) {  
    if (n==0) return 1;  
    else return n*fact(n-1);  
}  
main(void) {  
    int n;  
    cout<<"dammi un numero\n";  
    cin >> n;  
    cout << "Il suo fattoriale vale "<<fact(n)<<"\n";  
}
```

Cosa avviene nello stack?

Elementi di C++ di base

Arrays (vettori)

Array

Gli array sono collezioni di elementi omogenei

```
int valori[10];  
char v[200], coll[4000];
```

Un array di k elementi di tipo T in è un blocco di memoria contiguo di grandezza

```
(k*sizeof(T))
```


Array - 2

Ogni singolo elemento di un array può essere utilizzato esattamente come una variabile con la notazione:

`valori[indice]`

dove indice stabilisce quale posizione considerare all'interno dell'array

Limitazioni

Gli indici spaziano sempre tra 0 e $k-1$

Il numero di elementi è fisso (deciso a livello di compilazione - *compile time*): non può variare durante l'esecuzione (a *run time*)

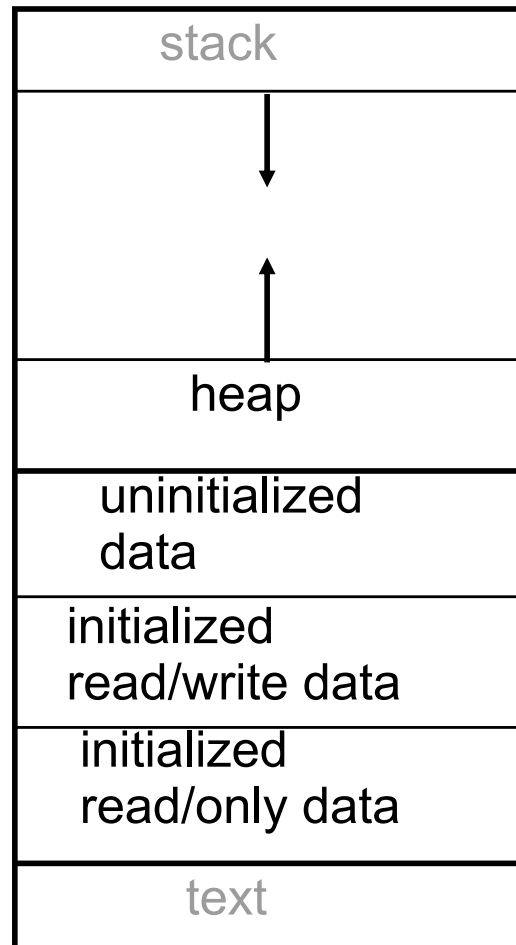
Non c'è nessun controllo sugli indici durante l'esecuzione

Catastrofe (potenziale)

```
...  
int a[10];  
a[256]=40;  
a[-12]=50;  
...
```

NOTA: le stringhe sono array di char

Il modello di memoria



*memoria allocata dalle funzioni
(Variabili automatiche)*

*memoria allocata dinamicamente
dal programmatore*

Variabili globali e statiche

<- questo é supportato solo da alcuni hardware

Codice eseguibile

Scope delle variabili: le globali

Nel seguente esempio a è una variabile globale.

Il suo valore è visibile a tutte le funzioni.

Le variabili globali vanno EVITATE a causa dei side-effects.

```
int a=5;
void f() {
    a=a+1;
    cout << "a in f: " << a << " - ";
    return;
}
main() {
    cout << "a in main:" << a << " - ";
    f();
    cout << "a in main: " << a << endl);
}
```

Output:

```
a in main: 5 - a in f: 6 - a in main: 6
```

Scope delle variabili: le automatiche

Nel seguente esempio a e' una variabile automatica per la funzione f.
Il suo valore è locale ad f.

```
int a=5;
void f() {
    int a=2, b=4;
    printf("(a,b) in f: (%d,%d) -",a,b);
    return;
}
main() {
    int b=6;
    printf("(a,b) in main: (%d,%d) -",a,b);
    f();
    printf("(a,b) in main: (%d,%d)\n",a,b);
}
```

Output:

(a,b) in main: (5,6) - (a,b) in f: (2,4) - (a,b) in main: (5,6)

ATTENZIONE! Le variabili automatiche SCHERMANO le variabili globali.

Quanto vale s?

```
void modifica(int s) {  
    s++;  
}  
main(void) {  
    int s=1;  
    modifica(s);  
    cout << "s=" << s << "\n";  
}
```

← A) "locale"

```
int s;  
void modifica() {  
    s++;  
}  
main(void) {  
    s=1;  
    modifica();  
    cout << "s=" << s << "\n";  
}
```

"globale" (B→)

Variabili globali

Le variabili globali sono "cattive"
(almeno quanto il GOTO)!

perchè violano il principio della località della informazione
(Principio di "**Information hiding**")

E' impossibile gestire correttamente progetti "grossi" nei quali
si faccia uso di variabili globali.

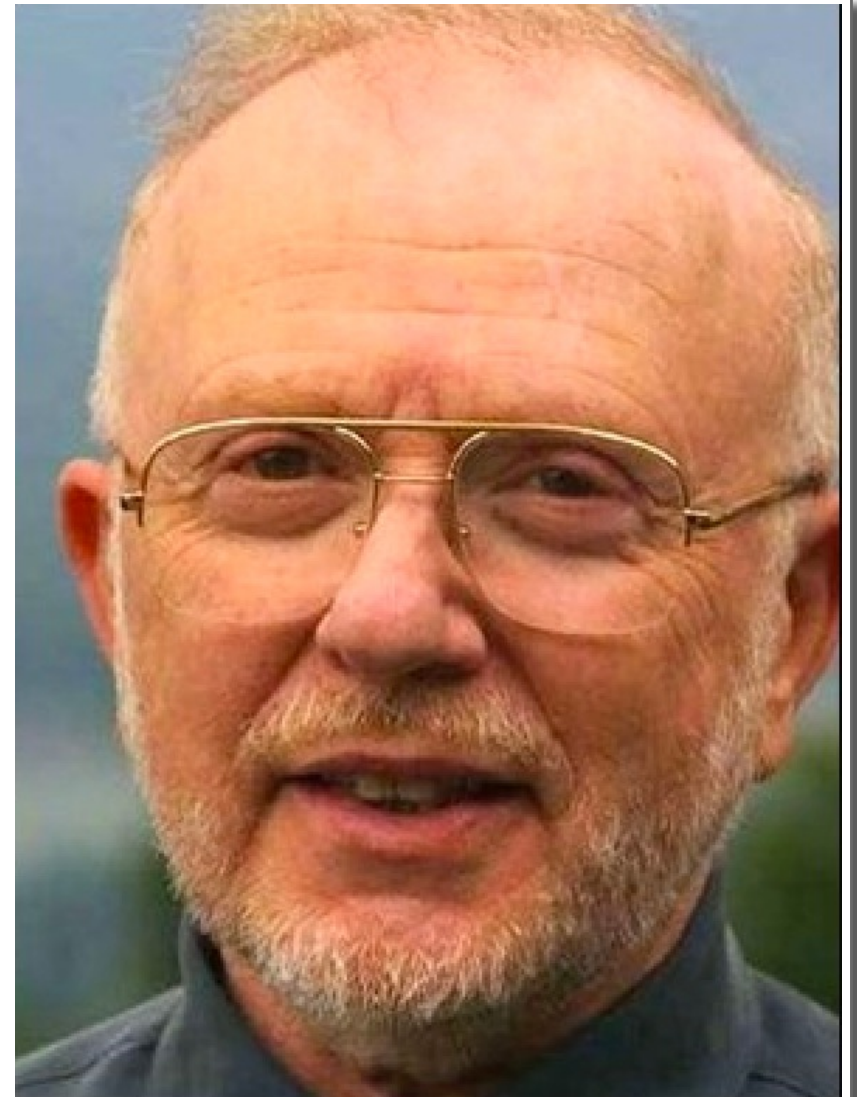
Principio del **NEED TO KNOW**:

Ciascuno deve avere **TUTTE** e **SOLO** le informazioni che
servono a svolgere il compito affidato

Principi di Parnas

Il committente di una funzione
deve dare all'implementatore
tutte le informazioni necessarie
a realizzare la funzione,
e NULLA PIÙ

L'implementatore di una
funzione deve dare all'utente
tutte le informazioni
necessarie ad usare la funzione,
e NULLA PIÙ



David Parnas

Funzioni: problema #1

```
void incrementa(int x) {  
    x=x+1;  
}  
main(void) {  
    int a=1;  
    incrementa(a);  
    cout << "a=" a << "\n";  
}
```

Come faccio a scrivere una funzione che modifichi le variabili del chiamante?

Quanto vale a quando viene stampata?
I parametri sono passati per valore (copia)!

Funzioni: problema #2

Come faccio a farmi restituire
più di un valore da una funzione?

Puntatori

Operatore indirizzo: &

&a fornisce l'indirizzo della variabile a

Operat. di dereferenziazione: *

***p** interpreta la variabile p come un puntatore (indirizzo) e fornisce il valore contenuto nella cella di memoria puntata

```
main() {
    int a,b,c,d;
    int * pa, * pb;
    pa=&a; pb=&b;
    a=1; b=2;
    c=a+b;
    d=*pa + *pb;
    cout << a<<" "<<b<<" "<< c <<endl;
    cout << a <<" "<< *pb <<" "<< d <<endl;
}
```

stack

a	1	0
b	2	4
c	?	8
d	?	12
pa	0	16
pb	4	20

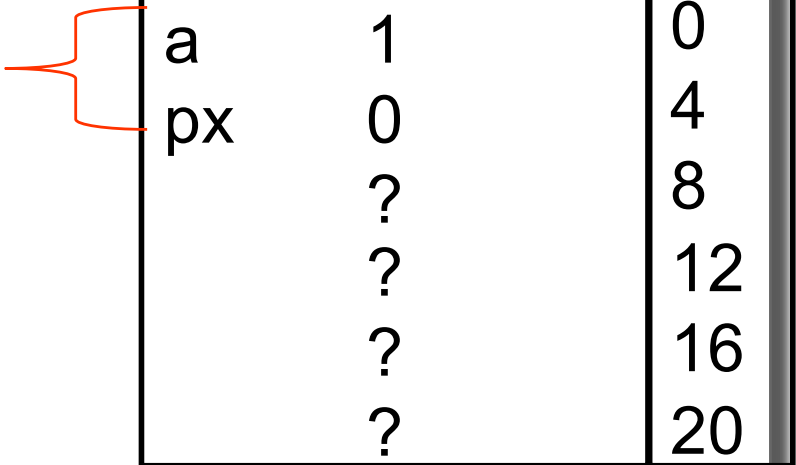
...

Funzioni e puntatori

TRUCCO: per passare un parametro *per indirizzo*,
passiamo per valore un puntatore ad esso!

```
void incrementa(int *px) {
    *px=*px+1;
}
main(void) {
    int a=1;
    incrementa(&a);
    cout<<a<<endl;
}
```

stack

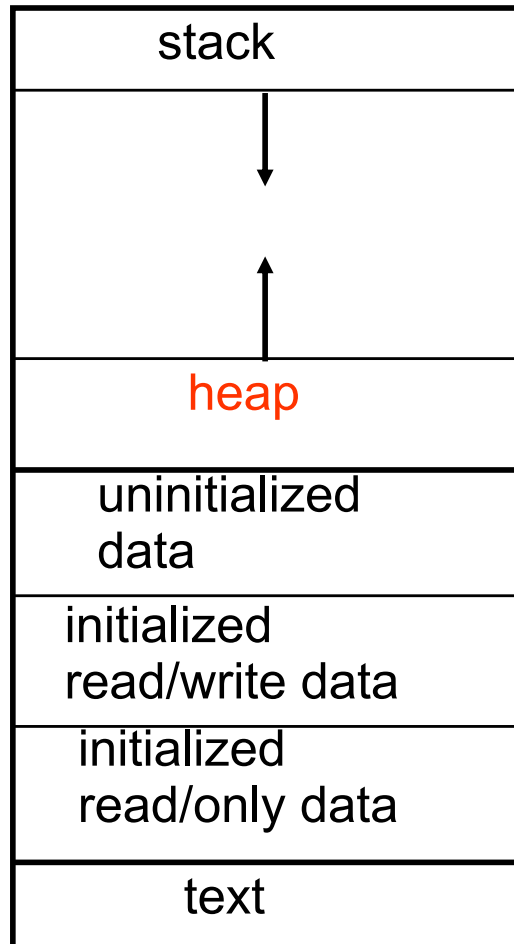


a	1	0
px	0	4
	?	8
	?	12
	?	16
	?	20

OUTPUT: 2

...

Il modello di memoria



*memoria allocata dalle funzioni
(Variabili automatiche)*

*memoria allocata dinamicamente
dal programmatore*

Variabili globali e statiche

<- questo é supportato solo da alcuni hardware

Codice eseguibile

Operatori *new* e *delete*

new type alloca **sizeof (type)** bytes in memoria (heap) e restituisce un puntatore alla base della memoria allocata. (esiste una funzione simile usata in C e chiamata **malloc**)

delete (* p) dealloca la memoria puntata dal puntatore p. (Funziona solo con memoria dinamica allocata tramite new. Esiste un'analogia funzione in C chiamata **free**).

Il mancato uso della **delete** provoca un insidioso tipo di errore: il **memory leak**.

Allocazione della memoria

Allocazione statica
di memoria
(at compile time)

```
main() {  
    int a;  
    cout<<a<<endl; //NO!  
    a=3;  
    cout<<a<<endl;  
}
```

OUTPUT: 1
3

Allocazione
dinamica
di memoria
(at run time)

```
main() {  
    int *pa;  
    pa=new int;  
    cout<<*pa<<endl; //NO!  
    *pa=3;  
    cout<<*pa<<endl;  
    delete (pa);  
    cout<<*pa<<endl; //NO!  
}
```

OUTPUT: 4322472
3
8126664

Vettori rivistati

Dichiarare un vettore è in un certo senso come dichiarare un puntatore.

`v[0]` è equivalente a `*v`

Attenzione però alla differenza!

`int v[100];` è "equivalente" a:
`int *v; v=new int[100];`

ATTENZIONE!

la prima versione alloca spazio STATICAMENTE (Stack)

la seconda versione alloca spazio DINAMICAMENTE (Heap)

Java: breve storia

- Inizio anni '90: Java nasce nei laboratori di ricerca Sun Microsystems con il nome "Oak"
 - Obiettivo: linguaggio per *intelligent consumer electronics*, poi cambiato in *set-top box*
- 1994: linguaggio per il Web
 - In particolare, Java consente di muovere codice (non solo dati) fra client e server: nascono le *applet*, rese popolari dal browser HotJava
- 1996: linguaggio general-purpose, fortemente connotato verso *distributed, network, Internet computing* (non solo Web)
- 2010: Oracle acquisisce Sun e di conseguenza anche Java

Superamento dei problemi

Memory management (memory leaks)

Aritmetica dei puntatori

Arrays senza controllo

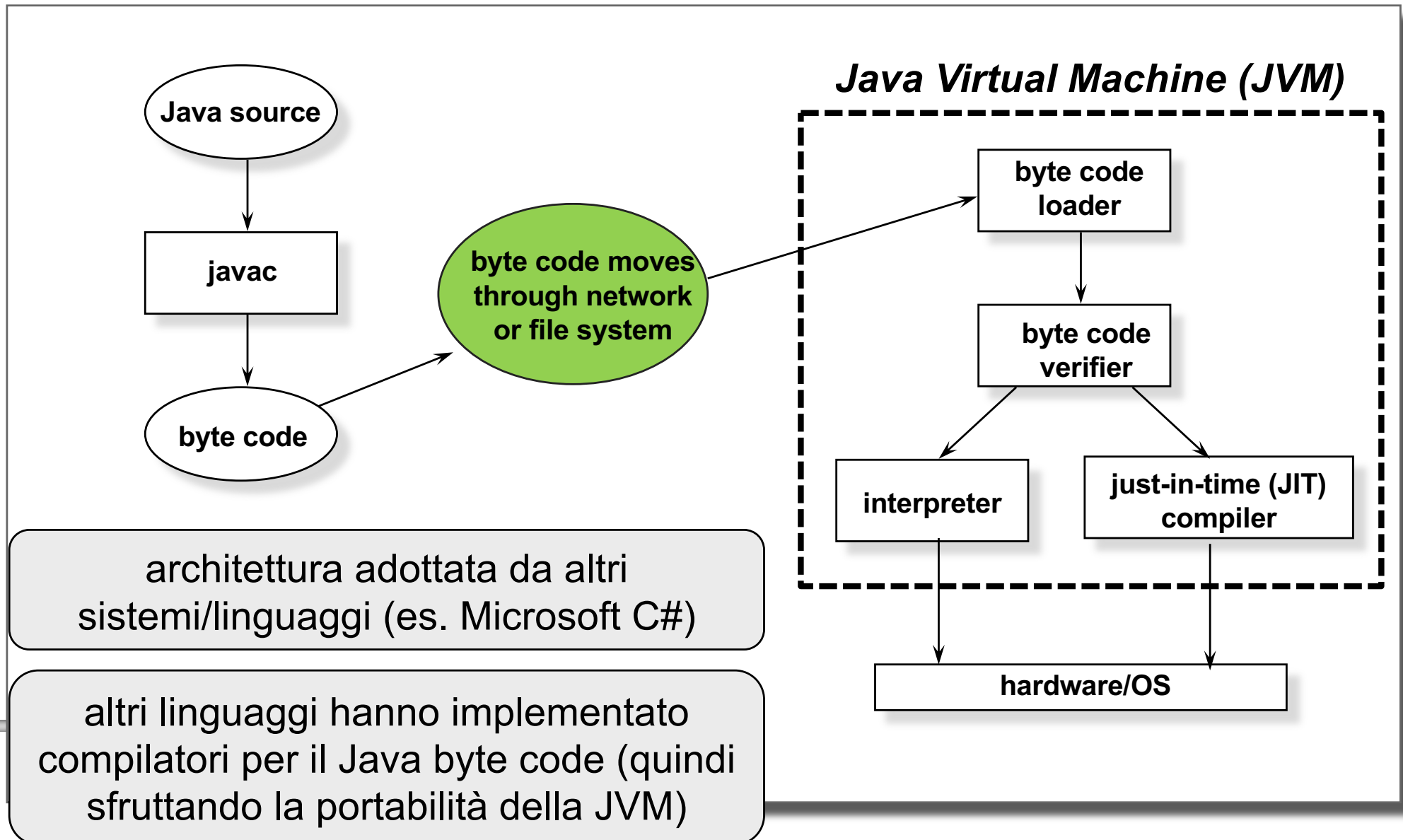
Stringhe promosse a tipo di dato

Miglior gestione di progetti “grossi”:

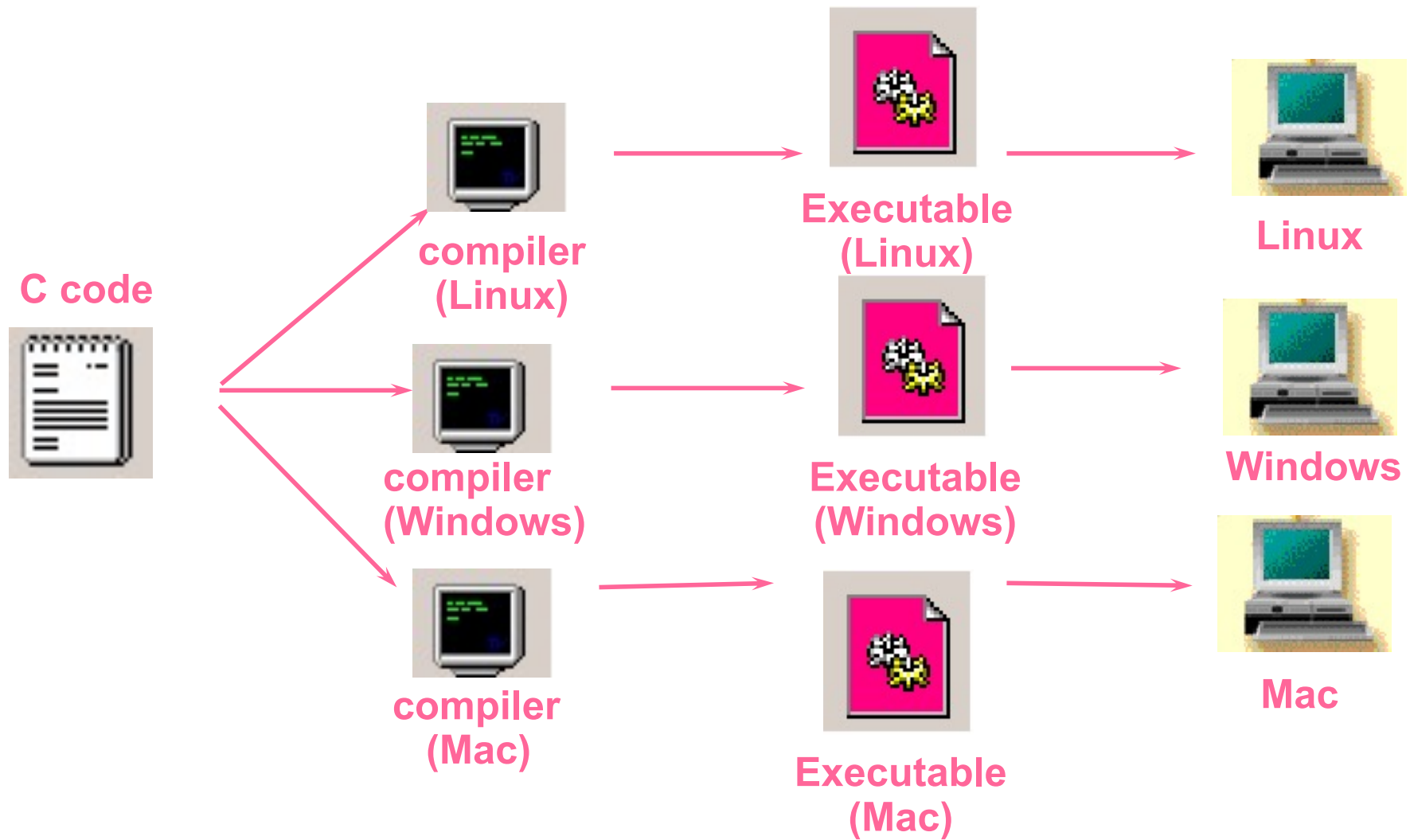
adozione (quasi) completa di

Object Oriented paradigm

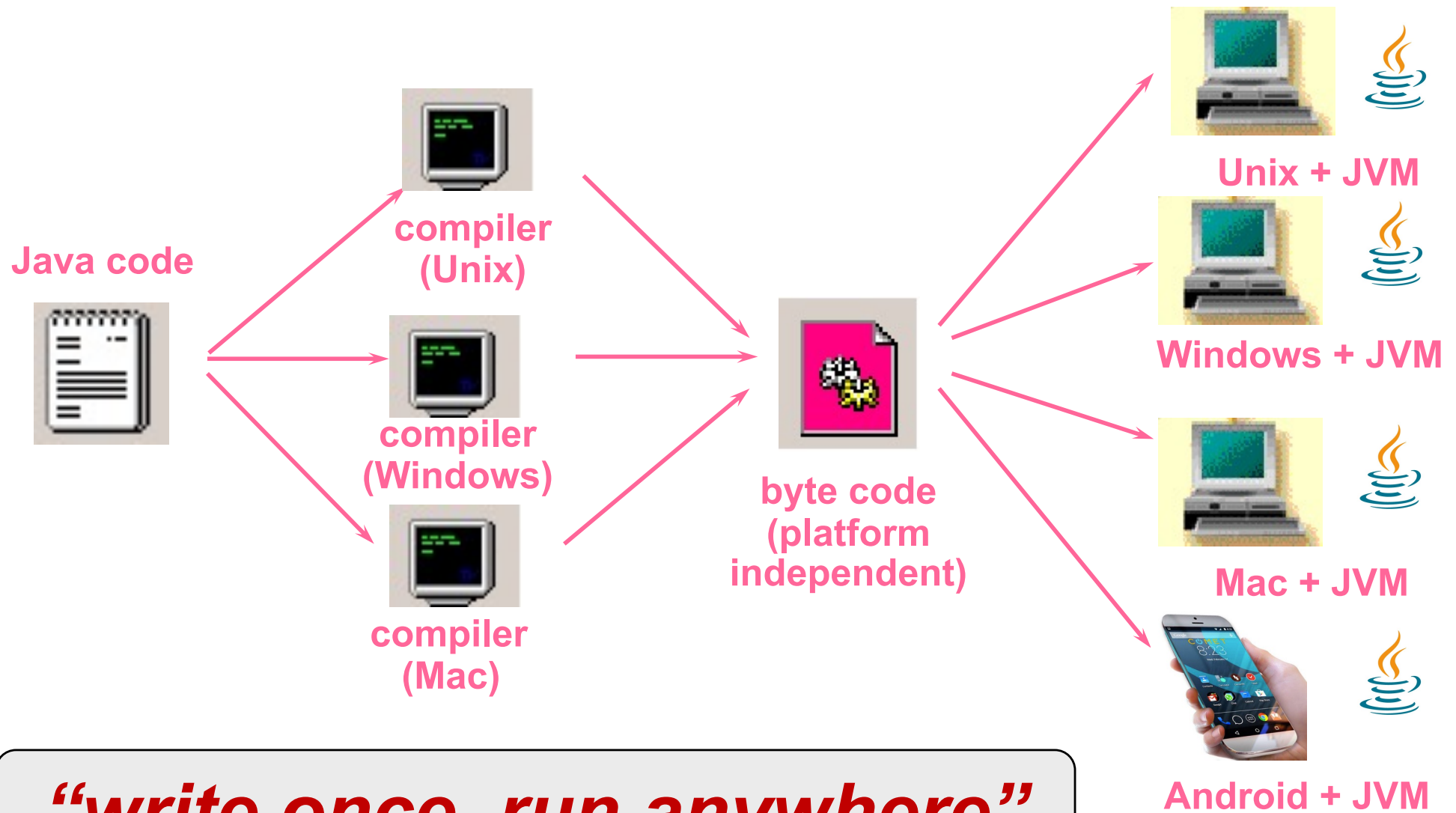
L'architettura di Java



Piattaforme e “portabilità”



Piattaforme e “portabilità”



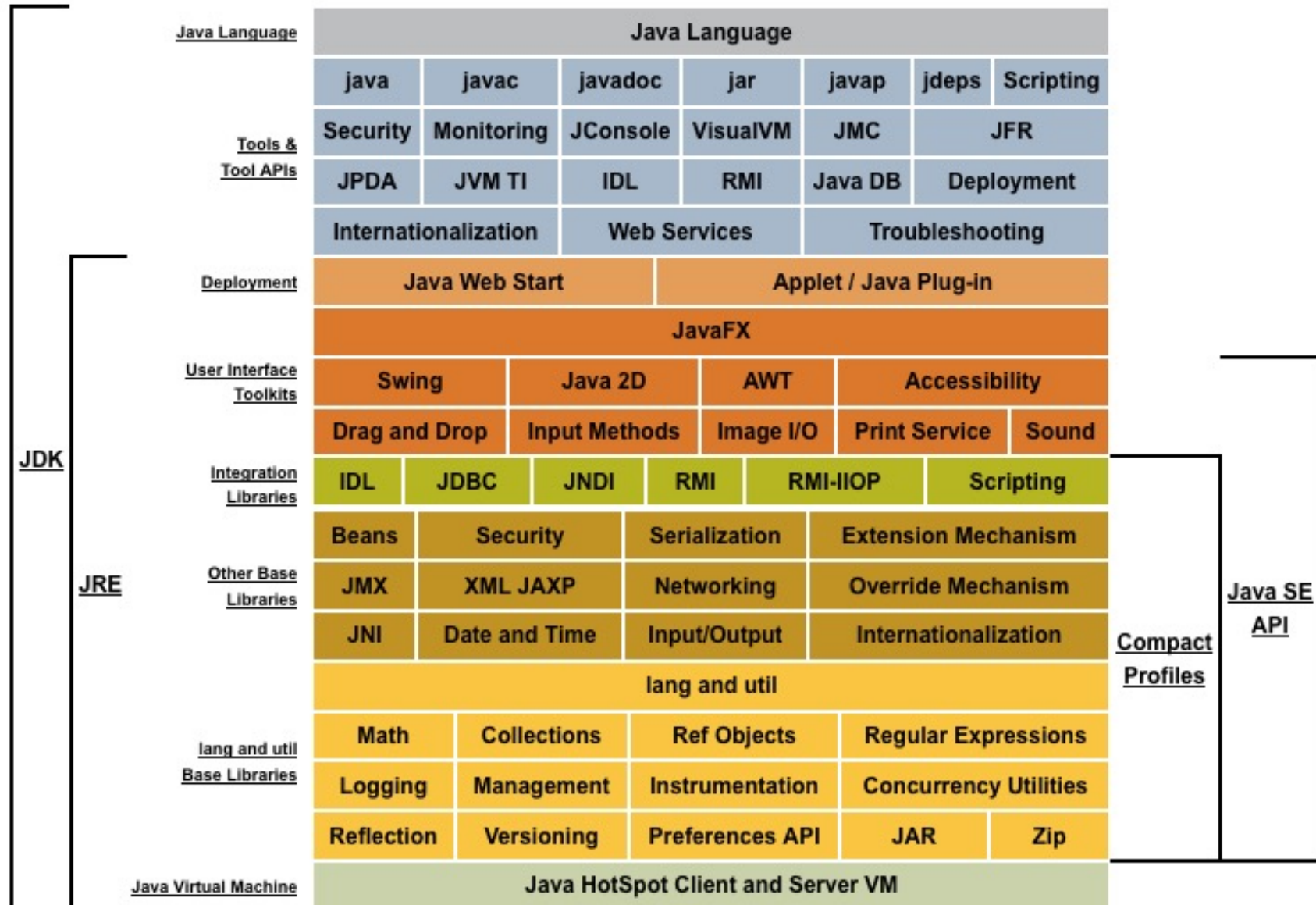
“write once, run anywhere”

Java: oggi

- Si stima che Java sia installato su oltre 15 miliardi di dispositivi...
- Android ha avuto un ruolo chiave nella diffusione recente di questo linguaggio
 - Anche se sfrutta una sua JVM incompatibile con quella standard, e si basa su librerie specifiche...
- Java è usato ampiamente in ambito Web e cloud
 - Web services, servlets, JSP, Struts, EJB, XML, ...
- Numerosi IDE (*Integrated Development Environments*) sono scritti in Java
 - Eclipse, IntelliJ, NetBeans, ...



The Java Platform



Tutorial ed esempi

<https://docs.oracle.com/javase/tutorial>



The Java™ Tutorials

The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases.

The Java Tutorials are practical guides for programmers who want to use the Java programming language to create applications. They include hundreds of complete, working examples, and dozens of lessons. Groups of related lessons are organized into "trails".

Trails Covering the Basics

These trails are available in book form as *The Java Tutorial, Sixth Edition*. To buy this book, refer to the box to the right.

- » [Getting Started](#) — An introduction to Java technology and lessons on installing Java development software and using it to create a simple program.
- » [Learning the Java Language](#) — Lessons describing the essential concepts and features of the Java Programming Language.
- » [Essential Java Classes](#) — Lessons on exceptions, basic input/output, concurrency, regular expressions, and the platform environment.
- » [Collections](#) — Lessons on using and extending the Java Collections Framework.
- » [Date-Time APIs](#) — How to use the `java.time` packages to write date and time code.
- » [Deployment](#) — How to package applications and applets using JAR files, and deploy them using Java Web Start and Java Plug-in.
- » [Preparation for Java Programming Language Certification](#) — List of available training and tutorial resources.



Not sure where to start?
See [Learning Paths](#)

Tutorial Contents

[Really Big Index](#)

Tutorial Resources

- » Last Updated 7/19/2016
- » [The Java Tutorials' Blog](#) has news and updates about the Java SE tutorials.
- » [Download the latest Java Tutorials bundle.](#)

Un buon libro...

Thinking in Java, Bruce Eckel

- È disponibile online:

[https://people.inf.elte.hu/delsaai/java/6Eckel%20-%20Thinking%20in%20Java%20\(4th%202006\)%20p1079.pdf](https://people.inf.elte.hu/delsaai/java/6Eckel%20-%20Thinking%20in%20Java%20(4th%202006)%20p1079.pdf)

- ... non è l'edizione più recente, ma è gratis
- Se invece desiderate un'edizione cartacea e/o in italiano, la trovate (con lo stesso titolo/autore) edita da Apogeo – ma è sconsigliata perché la traduzione ha introdotto errori.