

# Linguaggi di Programmazione

Marco Ronchetti

Dipartimento di Ingegneria e Scienza dell'Informazione  
University of Trento, Italy  
[marco.ronchetti@unitn.it](mailto:marco.ronchetti@unitn.it)  
<http://latemar.science.unitn.it>

Slides condivise con il prof. Giampietro Picco

# 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 e organizzazione

Il corso è svolto in coordinamento con  
Linguaggi di Programmazione offerto nella  
LT Inf. Org. e LT Matematica  
(prof. Picco)

Il materiale è sostanzialmente lo stesso  
Gli esami sono congiunti

# 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 (NetBeans su Linux)
    - ~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

Entro un anno vanno superati entrambi i moduli  
(Ronchetti + Kuper) - registrazione comune (12 crediti)

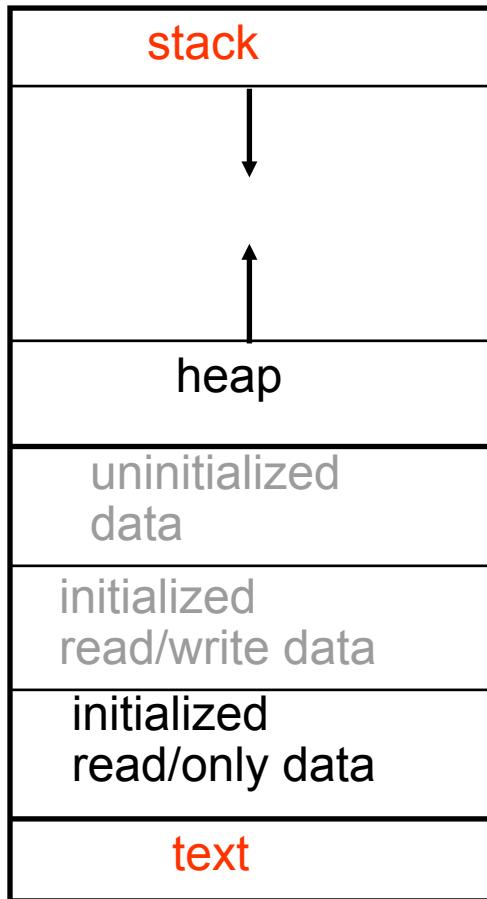
# Materiale didattico

- *Sito Web:*  
`latemar.science.unitn.it`

# 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 è 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	0
b	3	4
res	?	8
		12
		16
		20
		24
		28
		32
		36
		40
		44
		...

heap

main

# 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	
b	2	16	prodotto
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	
			heap
		40	
		44	
		...	

# 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
		...
		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

$$(k * \text{sizeof}(T))$$

# 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

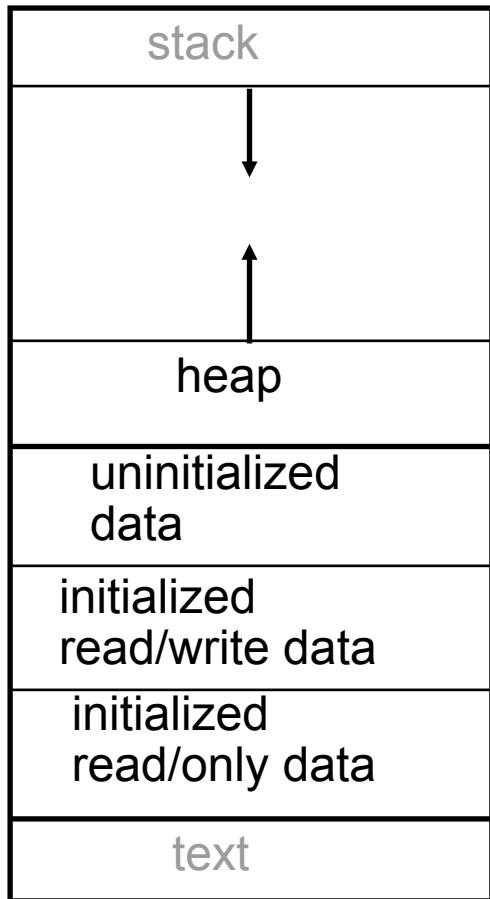
# Catastrofe (potenziale)

```
    . . .
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 è 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 Parna

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

# 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!

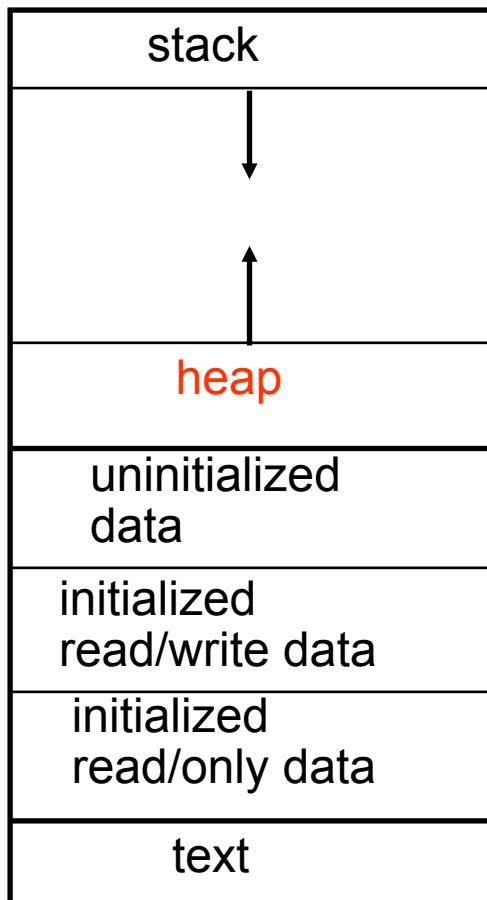
```
void incrementa(int *px) {  
    *px = *px + 1;  
}  
  
main(void) {  
    int a = 1;  
    incrementa(&a);  
    cout << a << endl;  
}
```

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 è 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 probemi

Memory management (memory leaks)

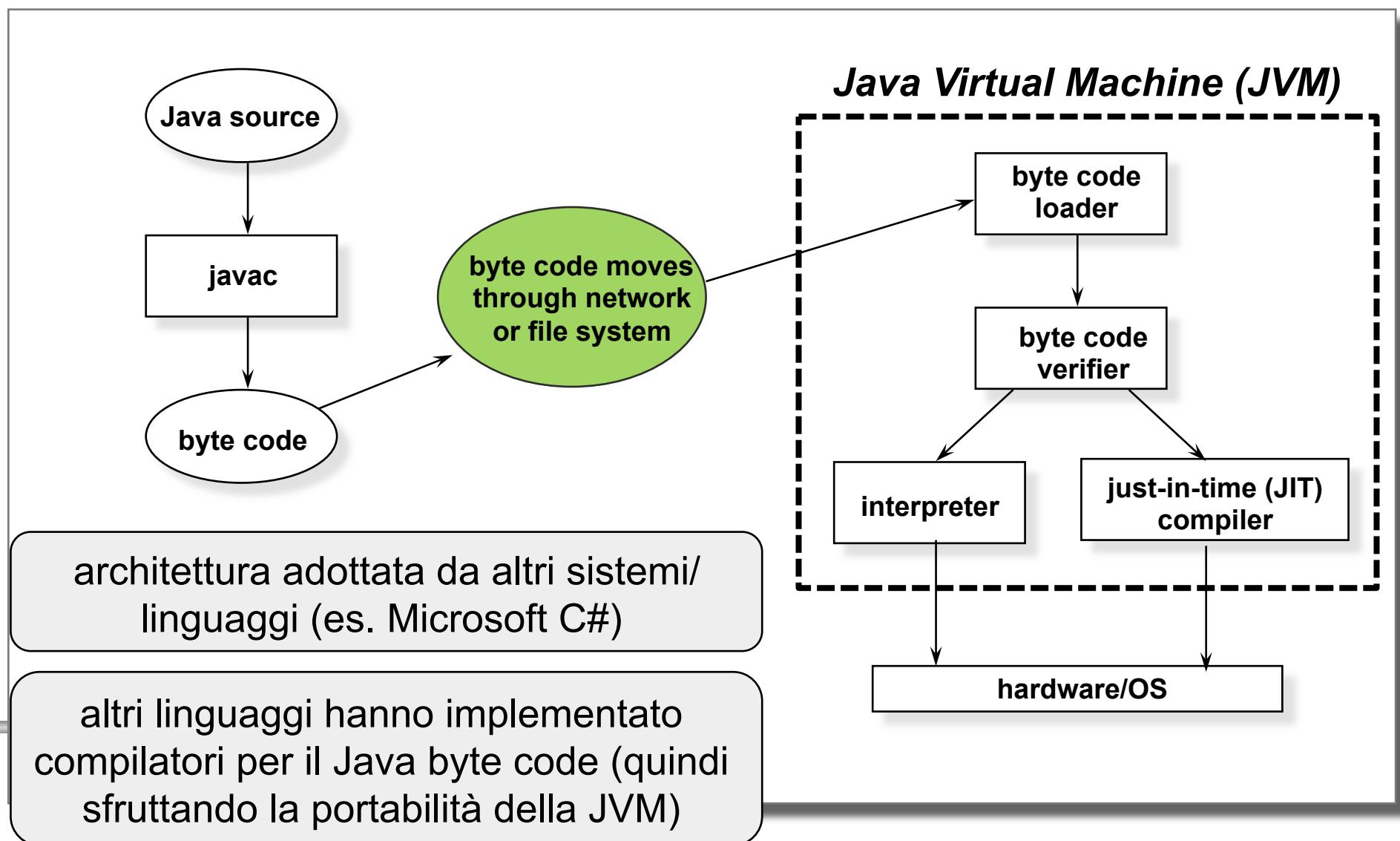
Aritmetica dei puntatori

Arrays senza controllo

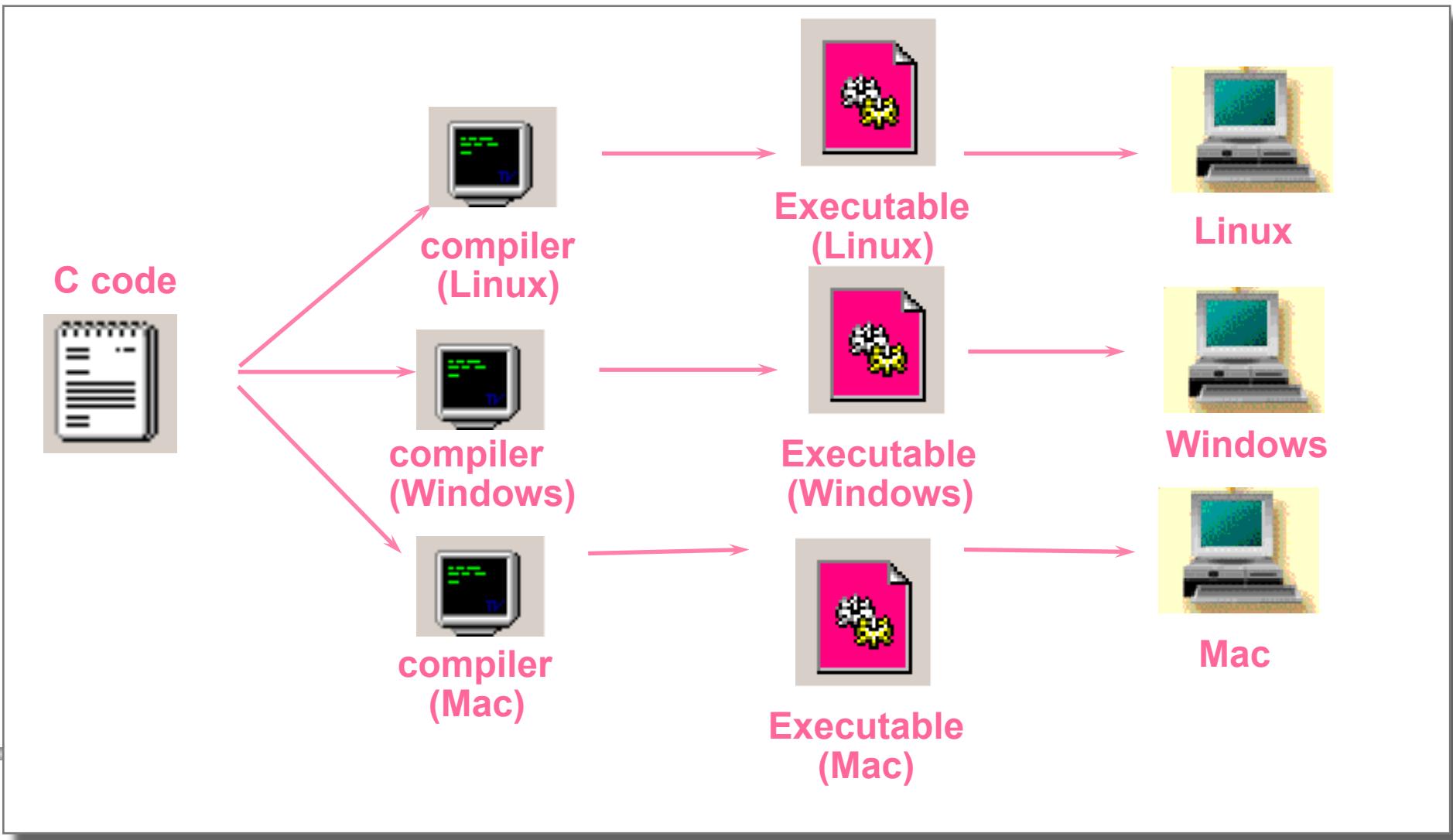
Stringhe promosse a tipo di dato

Miglior gestione di progetti “grossi”:  
adozione (quasi) completa di  
Object Oriented paradigm

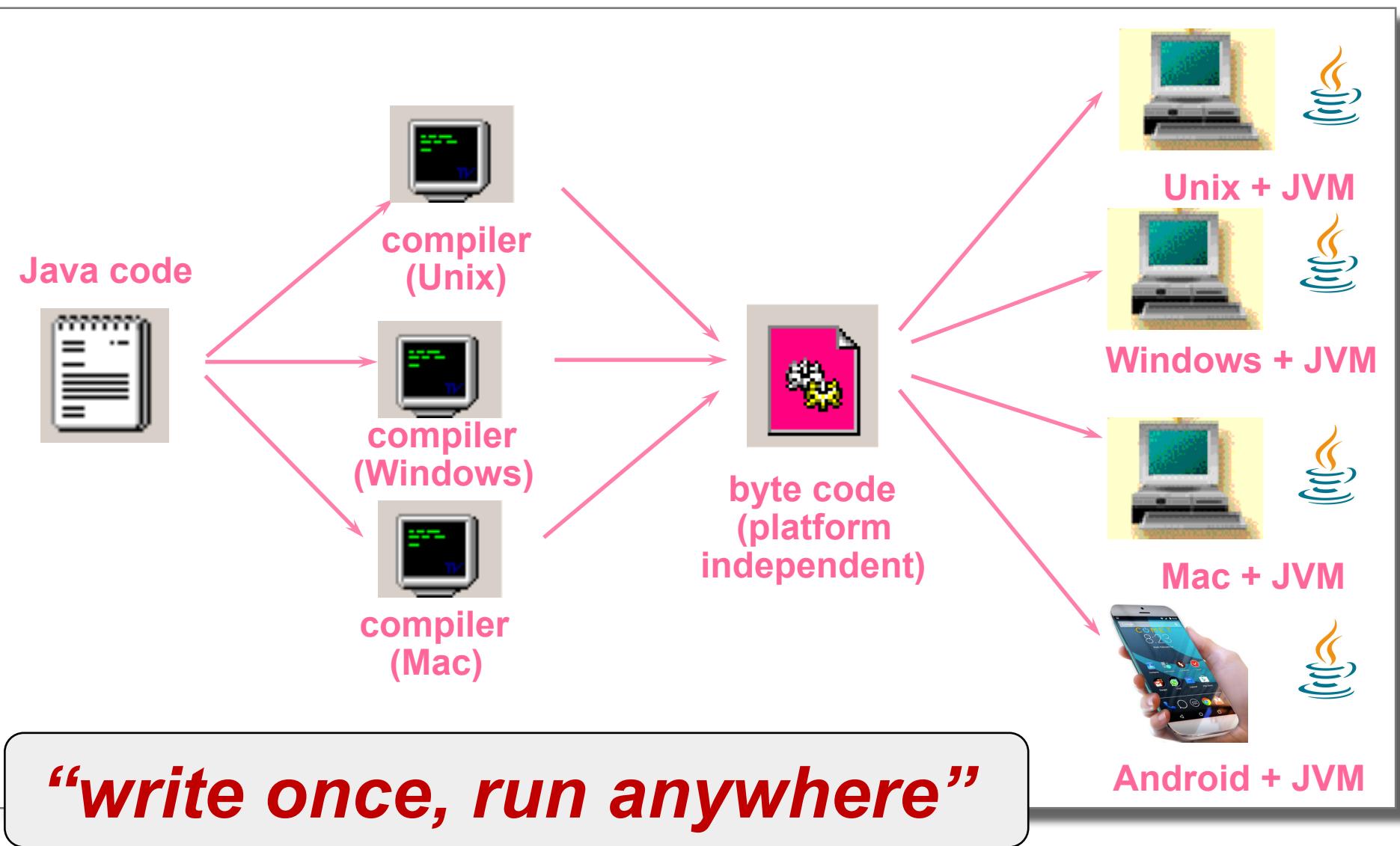
# L'architettura di Java



# Piattaforme e “portabilità”



# Piattaforme e “portabilità”

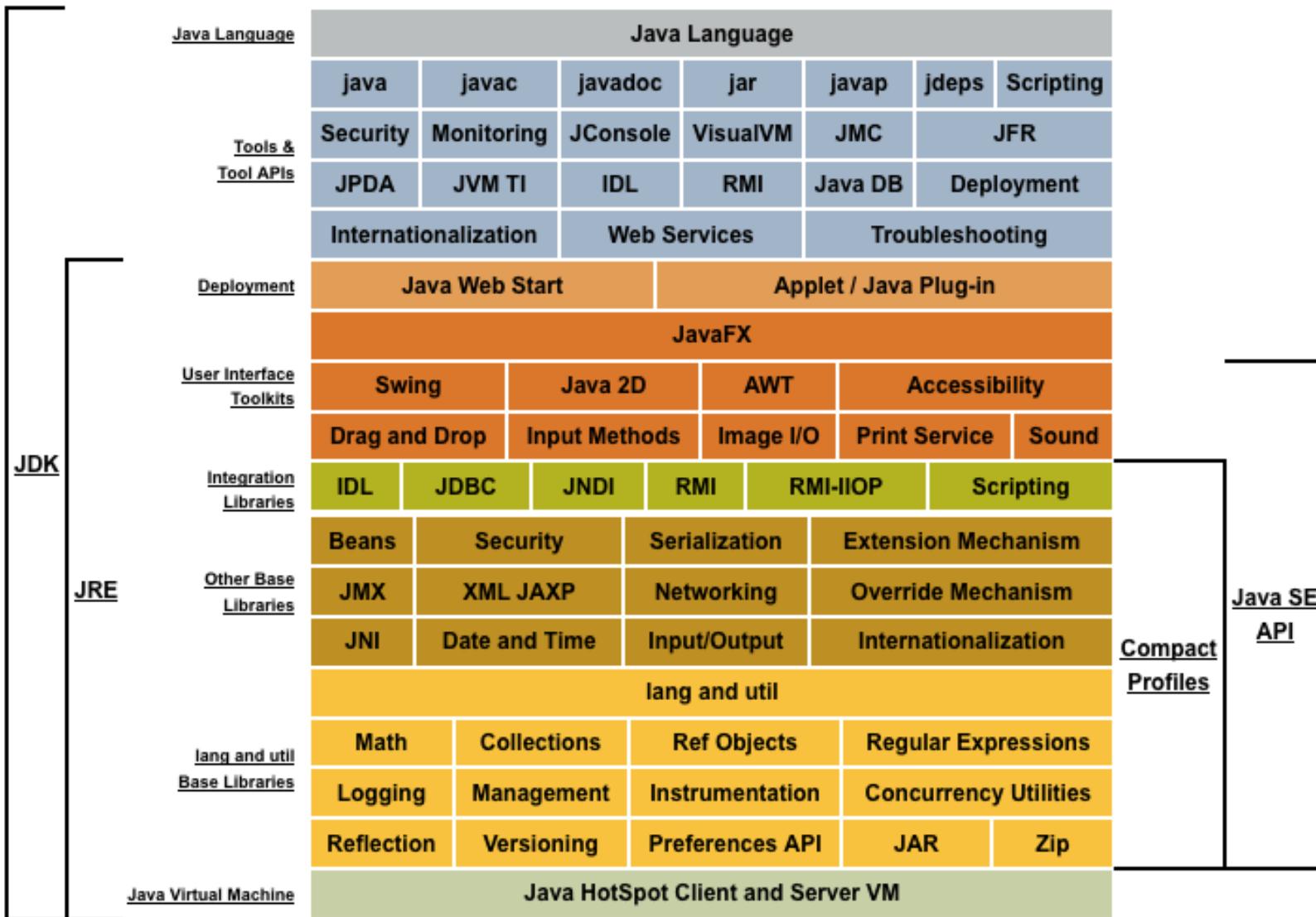


# Java: oggi

- 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





# Dove trovo Java?

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

The screenshot shows the Oracle Java SE Downloads page. On the left, there's a sidebar with links like Java SE, Java EE, Java ME, etc. In the center, there are two main download sections: 'Java Platform (JDK) 9' and 'NetBeans with JDK 8'. Below these, there's a detailed section for 'Java Platform, Standard Edition' (Java SE 9.0.4). To the right, a sidebar lists 'Java SDKs and Tools' and 'Java Resources'. A large red arrow points from the 'Java SE' link in the sidebar to the 'Java Resources' section.

ciò che serve a noi  
è il JDK  
(Java Development Kit)

già incluso nel  
NetBeans IDE  
(versione Java SE)

versione di riferimento:  
Java 8  
(usata da NetBeans  
e dai tutorial)

# 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` package 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: **<http://www.mindview.net/Books/TIJ>**
- ... 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.

# 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 JDK (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 un metodo speciale (**main**) della classe il cui nome coincide con il nome del programma

# La struttura di un programma Java

- Package
  - Ogni package contiene una o più compilation unit
  - Il package introduce un nuovo ambito di visibilità dei nomi
- Compilation unit
  - Ogni compilation unit contiene una o più classi o interfacce delle quali una sola pubblica
- Classi e interfacce
- Relazioni con il file system
  - package  $\Leftrightarrow$  directory
  - compilation unit  $\Leftrightarrow$  file

# Il linguaggio Java in breve

## Programmazione “in the small”

Tipi primitivi

Dichiarazione di variabili

Strutture di controllo: selezione condizionale, cicli  
identiche a quelle del C: if, switch, while, for...

Array

## Programmazione “in the large”

Classi

Interfacce

Package

# Piccolo esempio

```
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!");  
    }  
}
```

# Struttura dell'applicazione

```
class MyApp{  
    public static void main (String args[]) {  
        MyApp p = new MyApp();  
        p.doSomething();  
    }  
    public MyApp() {  
        // inizializza l'oggetto applicazione  
    }  
    public doSomething() {  
        // comincia il programma vero e proprio  
    }  
}
```

The diagram illustrates the execution flow of the Java code. It starts with the line `MyApp p = new MyApp();`. A callout points to `new MyApp()` with the text "crea un ‘oggetto’ a partire dalla classe". This is followed by `p.doSomething();`, with a callout pointing to it stating "e ne invoca una funzionalità". The next line is `public MyApp() {`, which has two callouts pointing to it: one to the opening brace with "costruttore" and one to the body with "metodo". Finally, the line `public doSomething() {` has a callout pointing to it with "... tutto è un oggetto ...".

crea un “oggetto” a partire dalla classe

e ne invoca una funzionalità

“costruttore”

“metodo”

... tutto è un oggetto ...

# Java vs. C: in breve

- 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
  - concorrenza (**Thread**)
  - networking
  - gestione delle eccezioni
  - strutture dati (es. **Vector**, **Dictionary**, **Hashtable**)

# Robustezza e puntatori

- 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

# 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);
...
}
```

package: viene incluso con l'istruzione **import** senza bisogno di pre-compilatori, **#ifdef**, etc.

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



## Altre differenze ...

---

... esistono: alcune sono profonde e avremo modo di discuterne nel resto del corso!