Università di Torino – Facoltà di Scienze MFN
Corso di Studi in Informatica

Programmazione I - corso B
a.a. 2009-10

prof. Viviana Bono

Blocco 3 – Nozioni preliminari: algoritmi, macchine, programmi, file, tipi di file.

Nozioni preliminari

Algoritmi e macchine, struttura del calcolatore

Algoritmo

- un *algoritmo* è un procedimento (di calcolo) specificato per mezzo di regole precise e non ambigue, in modo da poter essere eseguito dall'uomo "meccanicamente", cioè applicando le regole senza pensare al significato;
- algoritmi esistono da migliaia di anni (es.: quello di Euclide)
- vi sono molti modi per descrivere un algoritmo; ad esempio l'algoritmo per la divisione in colonna imparato alle elementari è un algoritmo perfettamente specificato a parole;
- *algoritmo* non è sinonimo di *diagramma di flusso*: il metodo dei diagrammi di flusso (o flow-chart) è una rappresentazione grafica di un algoritmo, che viene ormai usata solo in casi molto particolari: in generale, per scrivere un programma, non serve fare prima il diagramma di flusso.

Algoritmi e macchine

Un algoritmo, essendo un metodo “meccanico”, può essere eseguito da una macchina.

Si possono costruire macchine capaci di eseguire un particolare algoritmo, o un piccolo numero di algoritmi simili; ad esempio, una macchinetta calcolatrice è in grado di eseguire degli algoritmi corrispondenti alle quattro operazioni.

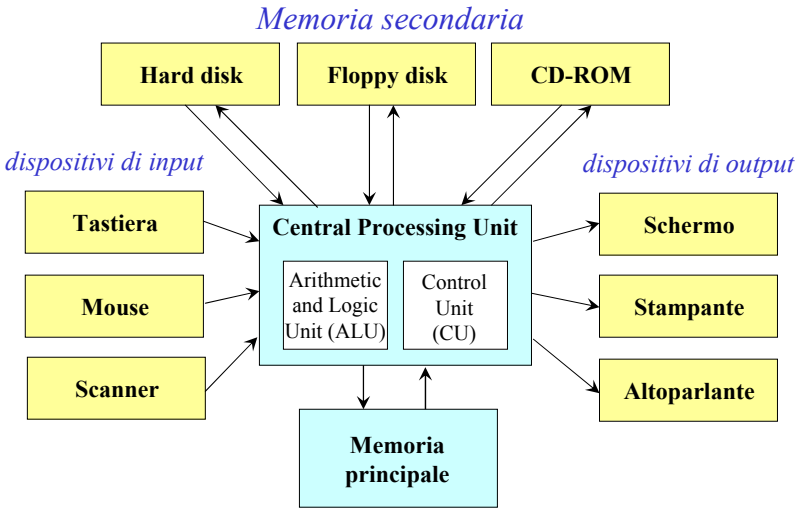
Un computer è una macchina cosiddetta *universale*, cioè capace di eseguire – in linea di principio – qualunque algoritmo concepibile, specificato attraverso un *programma* (scritto in un linguaggio la cui forma dipende dalla “macchina”).

NB Notate “macchina”: da qui in avanti sentire spesso parlare di macchine concrete, ma anche di macchine astratte.

Come è fatto un calcolatore?

- I primi calcolatori potevano essere considerati composti di tre parti, secondo il noto modello di von Neumann, valido - nelle sue linee generali - ancora oggi:
- una memoria (RAM), da concepirsi metaforicamente come un insieme di contenitori o “scatole”;
- una unità centrale di elaborazione, o *cpu*, che è in grado di eseguire dei programmi costituiti da sequenze di ben determinati tipi di *istruzioni-di-macchina*, che in generale modificano lo stato della memoria, cioè i contenuti delle celle di memoria; la cpu a sua volta è costituita da:
  - una ALU (o unità logico-aritmetica), che esegue le operazioni aritmetiche (+,-, ecc.) e logiche (and, or, not, ecc.);
  - una unità di controllo (CU), che legge le istruzioni e le *decodifica*, ossia le "trasforma" in comandi alla ALU
  - una (o piú) unità di comunicazione con l'esterno

Architettura di un calcolatore



Memoria (principale)

- Contiene i programmi in esecuzione e i loro dati.
- È paragonabile ad un insieme di scatole o celle numerate. Ogni cells di memoria ha:
  - un indirizzo, che la identifica univocamete e non può variare;
  - un contenuto o valore, che può essere cambiato;

  entrambi sono rappresentati da numeri binari:

Indirizzo	*Contenuto*
0010011010:	**0000110010101001**

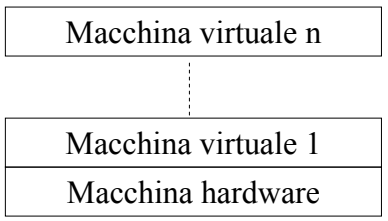
- Si chiama anche RAM (Random Access Memory), cioè memoria ad accesso casuale, perché si può accedere alle sue celle in qualsiasi ordine: cioè il tempo necessario per accedere ad una cella è lo stesso per tutte le celle della memoria.
- È volatile.

Unità di misura della capacità di memoria

Byte	B	8 bits	
Kilobyte	KB	2^{10} bytes = 1,024 bytes	$\sim 10^3$
Megabyte	MB	2^{20} bytes = 1,048,576 bytes	$\sim 10^6$
Gigabyte	GB	2^{30} bytes = 1,073,741,824 bytes	$\sim 10^9$
Terabyte	TB	2^{40} bytes = 1,099,511,627,776 bytes	$\sim 10^{12}$

Macchine virtuali

Gerarchia di macchine virtuali:



Per ogni funzionalita` definita al livello *i*, viene definita la traduzione nel linguaggio del livello sottostante *i-1*

Funzionalita` sempre più astratte

Linguaggio macchina: istruzioni eseguite direttamente dalla macchina hardware

Che cosa vuol dire “programmare”?

Concetto di gerarchia di macchine virtuali ⇒ possibilità di definire nuove funzionalita` astratte, da esprimere nel linguaggio di una macchina virtuale ⇒ potenza e flessibilità dei moderni elaboratori

Programmare = utilizzare un insieme di “comandi” virtuali o di macchina

Software di base:

- **Sistema Operativo** (insieme di programmi per la gestione dell’elaboratore)
- **Interpreti e compilatori** (programmi che *traducono* le funzioni dei linguaggi ad alto livello in funzioni appartenenti ai linguaggi sottostanti… fino al linguaggio macchina!)

La programmazione imperativa

È un modo di specificare gli algoritmi che viene “capito” dalla macchina hardware (concreta):

- Un programma è costituto da una sequenza di istruzioni o comandi (per questo si chiama imperativa).
- La macchina esegue il programma eseguendo un'istruzione dopo l'altra (alcune istruzioni hanno l'effetto di far eseguire un'istruzione successiva piuttosto che un'altra, oppure di far ripetere una sequenza di istruzioni, ecc.).
- La macchina ha uno stato interno, ed ogni istruzione modifica tale stato.

NB Anche Java è imperativo, ma non agisce direttamente sulla macchina concreta…

Istruzioni-macchina

Una istruzione-macchina è costituita a sua volta, di solito, dalla specifica del tipo di operazione aritmetica o logica da eseguire (ad esempio somma, o sottrazione, o confronto, ecc.) e dalle identità degli operandi, cioè dai nomi (o meglio, indirizzi) dei contenitori in cui andare a prendere gli operandi (ad es. gli addendi) e in cui andare a mettere il risultato (ad es. il risultato dell'addizione); oltre a queste vi sono delle istruzioni "di salto" che permettono di "saltare" ad eseguire un'istruzione diversa da quella successiva, in particolare permettono di "saltare" o no ad un'altra parte del programma a seconda del risultato di un certo confronto od operazione (istruzioni di salto condizionato).

Possiamo infine assumere, per semplicità, che vi siano delle istruzioni di input/output per comunicare con l'esterno (naturalmente non è così, la realtà è molto più complessa).

Un programma in un ipotetico linguaggio-macchina

istruz.1:	**inputint 325**	preleva un intero dal dispositivo di input e mettilo nella cella 325
istruz.2:	**inputint 326**	preleva un numero dal dispositivo di input e mettilo nella cella 326
istruz.3:	**mov 326 327**	copia il contenuto della cella 326 nella cella 327
istruz.4:	**add 325 326**	leggi i contenuti delle celle 325 e 326 fanne la somma, e metti il risultato nella cella 326
istruz.5:	**bnz 7** (branch on not zero to 7)	se il risultato dell'ultima operazione non è 0, salta all'istruzione 7
istruz.6:	**imm 100 326**	metti il valore immediato 100 nella cella 326
istruz.7:	**outputint 326**	leggi il contenuto della cella 326 e invialo al dispositivo di output

La sequenza di istruzioni-di-macchina costituente un programma, per poter essere eseguita dal calcolatore, deve anch'essa essere codificata in forma numerica; in particolare devono esserlo i nomi simbolici delle istruzioni, come ADD, MOV, BPOS, ecc.: ad esempio ADD sarà codificato con il numero 214, MOV con il numero 215, BPOS con il numero 216, ecc.; l'istruz.3 sarà allora codificata come 214 325 326, ecc. Un programma completamente codificato in forma numerica e direttamente eseguibile dallo hardware di una macchina viene detto programma in *linguaggio-macchina*; vi sono naturalmente tanti linguaggi-macchina diversi quanti sono i tipi di cpu esistenti.

Come si vede, la strutturazione di questo tipo di programma - cioè il modo in cui le istruzioni debbono o possono venire “disposte” - è molto povera, poiché coincide con la semplice sequenza (tanto che in tal caso si parla di programmi non strutturati).

Anche oggi la forma finale di programma che viene eseguita dallo hardware di una macchina è simile a quella sopra ricordata. Tuttavia raramente i programmi vengono scritti in tale forma; vengono invece scritti, per lo piú, in un cosiddetto *linguaggio di alto livello*, di piú facile concezione e comprensione per l'uomo, con una strutturazione piú ricca ed articolata che la semplice sequenza, e indipendente (almeno in linea di principio) dalla particolare macchina.

In un linguaggio di alto livello il programma precedente verrebbe scritto in un modo simile al seguente (notate i simboli al posto degli indirizzi):

...

```
readint(a);
readint(b);
c = b;
b = a + b;
if(b == 0) b = 100;
printint(b);
```

...

Programmi e dati

Come si è detto, un programma è rappresentato nel calcolatore da una sequenza di numeri (in forma binaria, cioè una sequenza di bits), esattamente come i dati su cui opera ed sta in un *file* (o più file).

Un programma può perciò avere come input, cioè come dato su cui lavorare, un altro programma; un programma può produrre come risultato un altro programma.

Tipi di file: eseguibili, files di testo, ecc.

File di testo e codifica dei caratteri alfanumerici

L'unico metodo di rappresentazione usato dai dispositivi elettronici è quello binario, dove le due cifre 0 e 1 sono rappresentate da due stati elettrici diversi.

Per memorizzare elettronicamente dei testi occorre quindi rappresentare (cioè codificare) ogni carattere con un numero; a tale scopo è stato definito negli anni '60 il codice ASCII.

Naturalmente per poter rappresentare un testo bisogna poter rappresentare anche il carattere "spazio", l'andata-a-capo, i segni di interpunzione, i caratteri speciali come $,@, [, ecc.

Ognuno di tali caratteri deve cioè essere rappresentato da un numero. Ad esempio lo spazio è rappresentato dal numero 32 (= 20 esadecimale).

Naturalmente all'interno di un testo possono comparire dei caratteri numerici, come nell'esempio seguente:

"26 is the hexadecimal representation of the number 38"

Non vogliamo però rappresentare il numero decimale 38 nel suo formato binario, perché la rappresentazione non può tener conto del significato dei caratteri (ad esempio 26 nel testo precedente ha lo stesso significato di 38, ma ovviamente nel testo li vogliamo proprio scritti nei due modi diversi!).

Le dieci cifre da 0 a 9 devono perciò essere rappresentate ciascuna separatamente da un numero.

Per evidenziare il fatto che tali caratteri numerici in un testo sono caratteri come tutti gli altri, il cui significato non interessa, essi sono rappresentati da numeri diversi dai loro rispettivi valori numerici: cioè il carattere 0 non è rappresentato dal numero 0, bensì dal numero 48 (esadecimale 30);

il carattere 1 è rappresentato dal numero 49 (esadecimale 31), ecc., fino al carattere 9 rappresentato dal numero 57 (esadecimale 39).

Originariamente il codice ASCII era a 7 bit, cioè ogni carattere era rappresentato da (= codificato con) un numero compreso fra 0 e 127; in esso non erano rappresentati i caratteri usati in lingue diverse dall'inglese, come le lettere accentate.

Per rimediare a ciò, il codice ASCII venne in seguito esteso a 8 bit (= 1 byte) cioè ogni carattere è rappresentato da un numero compreso fra 0 e 255.

Osservazione linguistica (E. Giovannetti)

La parola codice (in inglese code) originariamente indicava una tabella di corrispondenza, ad esempio il codice ASCII:

A 65

B 66

C 67

ecc.

Per comodità, invece di "la rappresentazione di A nel codice ASCII", ecc., si è poi cominciato a dire e scrivere brevemente

"il codice ASCII di A", ecc.

Oggi si usa abitualmente la parola *codice* sia nel senso originario, sia nel senso di *codifica di qualcosa (ad esempio un carattere) in un certo codice*, come nell'esempio precedente.

La si usa anche, come vedremo, nel senso di *programma*.

Intorno al 1990 è stato definito un codice a 16 bit, chiamato Unicode, che è un'estensione del codice ASCII (cioè Unicode rappresenta ogni carattere rappresentabile in ASCII con lo stesso numero assegnatogli da ASCII, con la sola aggiunta fisica di otto zeri "non significativi" a sinistra; ad esempio la rappresentazione del carattere 9 sarà rappresentata da:

0000 0000 0011 1001 (in notazione esadecimale 0039)

L' Unicode permette di rappresentare $2^{16} = 65536$ caratteri distinti!

In questo modo si sono potuti assegnare dei codici non solo a tutti i caratteri in uso negli alfabeti latini, greco e cirillico, ma anche ai caratteri di tutte le lingue del mondo, inclusi i 21000 ideogrammi cinesi.

File di testo e file eseguibili

Un file di testo contiene una sequenza di numeri che sono codici di caratteri; se "aperto" con un programma "editor" (elaboratore di testo) come Textpad o Vim, esso viene visualizzato come un testo, che l'utente può modificare e poi ri-salvare su memoria secondaria.

Un programma in linguaggio-macchina è invece una sequenza di numeri che codificano una sequenza di istruzioni di macchina. Se dato in input direttamente a un apposito programma del sistema operativo, esso viene eseguito dalla macchina (microprocessore), dopo che il sistema operativo ha effettuato alcuni controlli e ne ha avviato l'esecuzione stessa.

I file di entrambi i tipi sono quindi sequenze di bits; come fa allora il calcolatore a sapere che uno rappresenta un testo e l'altro un programma eseguibile? Lo hardware del calcolatore non lo sa!

Semplicemente, con un file si fanno operazioni diverse che con l'altro, cioè i due file vengono dall'utente (o dal sistema operativo) dati in input a programmi diversi: il file di testo viene dato ad un editor, il programma eseguibile viene dato ad un programma che ne controlla la forma e poi ne lancia l'esecuzione.

Possiamo cercare di far eseguire un file di testo: il sistema operativo risponderà che il contenuto del file non è eseguibile. Possiamo aprire con un editor un file eseguibile: l'editor interpreterà come caratteri i bytes costituenti il programma: essi saranno in generale codici di caratteri "strani" oppure non visualizzabili (lo vedrete…).

In conclusione:

tipo di file = operazioni eseguibili su di esso

I file di qualunque tipo sono costituiti da sequenze di uni e di zeri (bit);

ciò che distingue un tipo dall'altro sono le (oper)azioni che si possono sensatamente eseguire su di essi, cioè i programmi a cui essi possono essere dati in input (senza generare errori).

Tipi di file e nomi con estensioni (in Windows)

Vi sono quindi diversi tipi di file, finora ne abbiamo considerati due: i file di tipo testo, e i file di tipo eseguibile.

Per poter riconoscere il tipo di un file senza dover "provare" a darlo in input a vari programmi, e soprattutto per permettere al sistema operativo Windows di trattare in modo corretto ogni file (ad esempio passandolo al programma opportuno quando l'utente chiede di "aprirlo"), i nomi dei files sono in genere composti di due parti separate da un punto (dot), di cui la seconda, detta estensione, indica il tipo del file.

Ad esempio i file con estensione txt sono file di testo, i file con estensione exe sono (per i sistemi Windows) file eseguibili (executable).

File di testo *formattato*

Un file di testo, sia esso ASCII o Unicode, non contiene l'informazione concernente l'aspetto tipografico di un testo: cioè l'informazione relativa al font, che è il tipo di carattere tipografico, allo stile, cioè se normale, grassetto, corsivo, ecc, alla dimensione del carattere, eventualmente al colore, ecc.

Per poter memorizzare e quindi riprodurre anche questi aspetti vi sono dei programmi come Word che memorizzano tali informazioni per mezzo di codici particolari inframmezzati ai codici dei caratteri. Tali codici particolari non corrispondono a caratteri sensati per gli uomini; pertanto se si visualizza un file prodotto da Word (caratterizzato dall'estensione doc) con un editor di puro testo come Textpad, si ottengono sullo schermo dei simboli strani, in mezzo ai quali vi è anche il testo originale.

Linguaggi di programmazione: file sorgente, compilazione, esecuzione.

Linguaggi di programmazione

- Per esempio Java, C, C++, BASIC, Pascal, …
- Proprietà:
  - Non ambigui
  - Precisi
  - Concisi
  - Espressivi
  - Alto livello (molte astrazioni)

Diversi tipi di linguaggi di programmazione

Diversi tipi di linguaggi (ad alto livello):

- Linguaggi imperativi
  Basati sul concetto di *assegnamento*: le istruzioni sono sostanzialmente operazioni di modifica delle strutture dati
  [es: Fortran, Cobol, Basic (non strutturati); Pascal, C (strutturati)]
- Linguaggi funzionali
  Basati sul concetto di *funzione*: le istruzioni sono funzioni che vengono valutate [es: Lisp]
- Linguaggi logici
  Basati sul concetto di *dimostrazione*: le istruzioni esprimono relazioni logiche tra input e output [es: Prolog]
- Linguaggi ad oggetti
  Basati sul concetto di dato come *oggetto attivo*, in grado di ricevere e inviare *messaggi* [es: Smalltalk, C++, CLOS, Java]

Linguaggio

Linguaggio:

- *alfabeto*: insieme di simboli con cui si possono costruire i termini del linguaggio (*lessico*)
- *sintassi*: definita da una *grammatica* che fornisce le regole di composizione dei termini in *frasi ben formate* del linguaggio
- *semantica*: definisce il significato delle frasi ben formate del linguaggio

Analizzatore sintattico (*parser*):

analizza frasi e decide se sono frasi ben formate del linguaggio o no

Una grammatica (o sintassi)

Regole

Frase → *ParteNominale* *ParteVerbale* .
ParteNominale → *Nome* *Relativa*$_{opt}$
ParteVerbale → *VerboIntransitivo* |
VerboTransitivo *ParteNominale*
Nome → *NomeProprio* | *Articolo* *NomeComune*
Relativa → che *ParteVerbale*
NomeProprio → Mario | Lucia
NomeComune → cane | gatto
Articolo → il | un
VerboIntransitivo → corre | scappa
VerboTransitivo → insegue | raggiunge

Nota: opt = opzionale (cioè che può esserci oppure no)

Esempi di frasi corrette secondo la sintassi precedente

Mario corre.

Il cane scappa.

Il cane insegue il gatto.

Mario insegue Mario.

Lucia insegue Mario che scappa.

Lucia che insegue Mario raggiunge il cane.

Mario insegue il cane che insegue un gatto che insegue Lucia.

Il gatto raggiunge il gatto che raggiunge il gatto che raggiunge il gatto che corre.

Un cane raggiunge un gatto che insegue il cane.

Mario che insegue Mario raggiunge Lucia che insegue Lucia.

...

Esempi di frasi scorrette secondo la stessa sintassi

Il Mario corre.

Mario corre *(manca il punto finale!)*

Cane insegue cane.

Il gatto insegue.

Un cane raggiunge un Mario.

Il gatto insegue il topo.

Corre il cane.

Il cane che scappa.

Che cane.

...

Linguaggio di programmazione

Linguaggio di programmazione:

- *frasi ben formate* = programmi
- programma = insieme di "istruzioni"
- *semantica* = esecuzione del programma

⇒ occorre tradurre il programma (scritto in un linguaggio ad alto livello) in istruzioni in linguaggio macchina

Due tecniche per effettuare questa traduzione:

- compilazione
- interpretazione

Interpretazione e compilazione

Un programma in un linguaggio "ad alto livello" come Pascal, Java o C è un file di puro testo, detto file sorgente, composto seguendo ben precise regole di sintassi. Esso pertanto non può essere eseguito direttamente dallo hardware della macchina. Per *eseguire* il programma sono possibili, a seconda del tipo di linguaggio, diverse strade:

linguaggi interpretati (Javascript): vi è un apposito programma, detto *interprete*, che "legge" il programma dato e lo *esegue*, cioè esegue istruzioni di macchina che realizzano il voluto significato del programma-sorgente;

linguaggi compilati (Pascal, C): vi è un apposito programma, detto *compilatore*, che traduce il programma-sorgente in un programma in linguaggio macchina, che potrà essere eseguito direttamente dallo hardware.

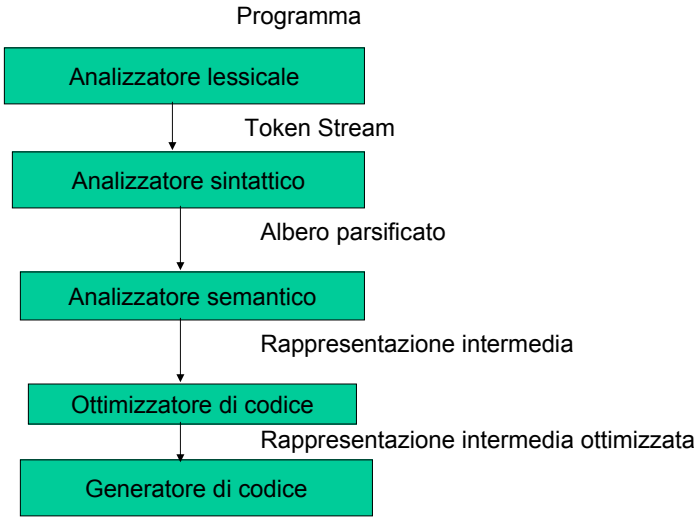
Interpretazione e compilazione

- I linguaggi interpretati sono inefficienti: l'interprete deve "leggere" il programma e tradurlo al volo in istruzioni macchina.
- Un programma interpretato, essendo un file-sorgente (cioè un file di testo), "gira" su qualunque macchina su cui vi sia un interprete del linguaggio. Naturalmente gl'interpreti per macchine di tipo diverso saranno diverse.
- I linguaggi compilati sono efficienti, perché un programma nella forma compilata è (semplificando un po') direttamente eseguibile dalla macchina.
- Un programma nella forma compilata, essendo un programma in linguaggio-macchina per un certo tipo di macchina, non "gira" su macchine di tipo diverso: occorre prima ricompilare il sorgente.

Linguaggi semi-compilati (semi-interpretati): Java, C#

- Il programma sorgente viene prima compilato (cioè tradotto) in un programma in un linguaggio intermedio uguale per tutti i tipi di macchina.
- Tale programma in linguaggio intermedio viene poi interpretato da un programma interprete, diverso per ogni diverso tipo di macchina.
- Il programma compilato può quindi essere distribuito a tutti in un'unica versione: sui diversi tipi di macchine esso sarà poi interpretato da interpreti appositi per quelle macchine.

Anatomia di un compilatore



Codice macchina (**non** per Java: bytecode)

Perché scrivere un programma?

Per risolvere un problema!

1° passo: analizzare il problema ed identificare una procedura di soluzione = ***algoritmo*** = insieme di passi che descrivono i dati e le azioni da eseguire su di essi

2° passo: tradurre l'algoritmo in un insieme di istruzioni per eseguire quelle azioni sui dati = *programma* (programmazione in-the-small).

Se il problema è complesso (es: gestione automatizzata di una segreteria studenti) occorre un *sistema software* (= insieme di programmi, programmazione in-the large, vi accenneremo)

Che cosa vuol dire“programmare”?

1º passo: **analisi** del problema e **progettazione** dell’*architettura* del sistema

2º passo: **realizzazione** delle diverse componenti dell’architettura

Nello sviluppo del software, le fasi di **analisi** e **progettazione** sono importantissime!

Il linguaggio Java: qualche anticipazione

Le sue origini

- **Linguaggio orientato agli oggetti**, nato all'inizio degli anni '90 presso la Sun Microsystems
- Pensato per **Internet** e per la **programmazione Web**
- Estremamente **portabile**
- Possiede moltissime **librerie**

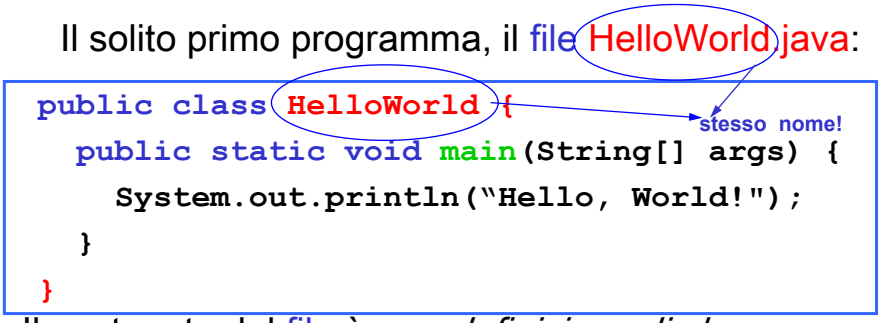
Java

- Formalismo ad alto livello...
  - Permette di descrivere programmi basandosi su concetti primitivi "sofisticati" (file, finestre, tabelle, liste, ecc.)
- ...basato su una notazione testuale familiare
  - Codice sorgente
  - Simile, nella sintassi e nelle istruzioni al linguaggio C

Avvertenza

Alcuni “termini” che compariranno da qui in avanti nei programmi avranno valenza di “parole magiche” per qualche tempo, perché dovremmo scriverle per far funzionare le cose senza capire bene il loro significato, ma piano piano tutto assumerà un senso preciso…

Il solito primo programma, il file HelloWorld.java:



Il contenuto del file è una *definizione di classe,* contenente a sua volta una *definizione di metodo*

`HelloWorld` è il nome della classe che viene definita;
`main` è il nome del metodo definito nella classe `HelloWorld`;
`String` è un *tipo* di dato: è l’insieme delle sequenze di caratteri

Il sistema Java

Il **nucleo** del sistema Java è costituito da:

- il programma compilatore javac
  (in Windows propriamente javac.exe)
- il programma interprete java
  (in Windows propriamente java.exe)

Il file sorgente, che deve avere l'estensione *java*, ad es. HelloWorld.java, viene dato in input a javac, con il comando:

```
javac HelloWorld.java
```

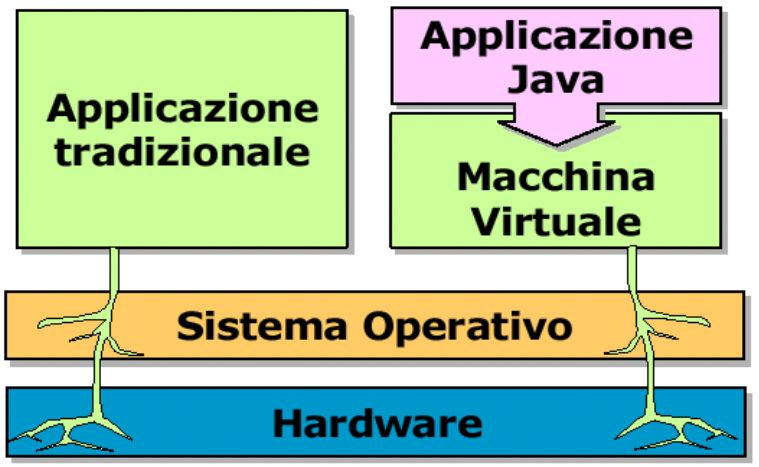
javac lo trasforma in un programma in un formato intermedio detto *bytecode*, contenuto in un file di estensione class, cioè HelloWorld.class

Il file HelloWorld.class deve poi essere dato in input al programma java (la *Virtual Machine*). , che lo interpreta, cioè lo esegue. Il comando è:

```
java HelloWorld
```

L'effetto è la "stampa" a video di: Hello, World!

VM = Macchina Virtuale (I)



Macchina Virtuale (II)

- Astrazione di un elaboratore “generico”
  - Ambiente di esecuzione delle applicazioni Java
- Esempio:
  - *java HelloWorld*
- Responsabilità:
  - Caricamento classi dal disco
  - Verifica consistenza codice
  - Esecuzione applicazione

Compilazione ed esecuzione

Un file HelloWorld.java è un file di testo (che sta sullo hard disk).

Come visto, il compilatore legge tale file e genera il file HelloWorld.class, che è una traduzione del programma in una forma più conveniente per l’esecuzione (il *bytecode*, il linguaggio della VM).

L’esecutore esegue (o interpreta) il file HelloWorld.class

NB il compilatore e l’esecutore non sono macchine fisiche, bensì dei programmi che vengono eseguiti dalla macchina fisica. Durante l’esecuzione del programma HelloWorld la macchina esegue il programma java.exe che esegue il programma HelloWorld.class.

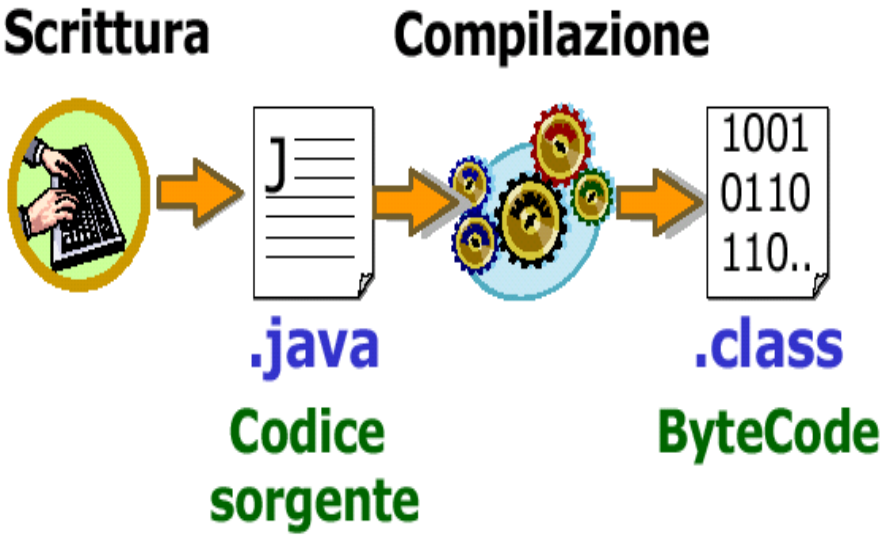
Attenzione!

Diciamo “un programma Java” per intendere un programma-sorgente scritto in Java, oppure il suo compilato in bytecode.

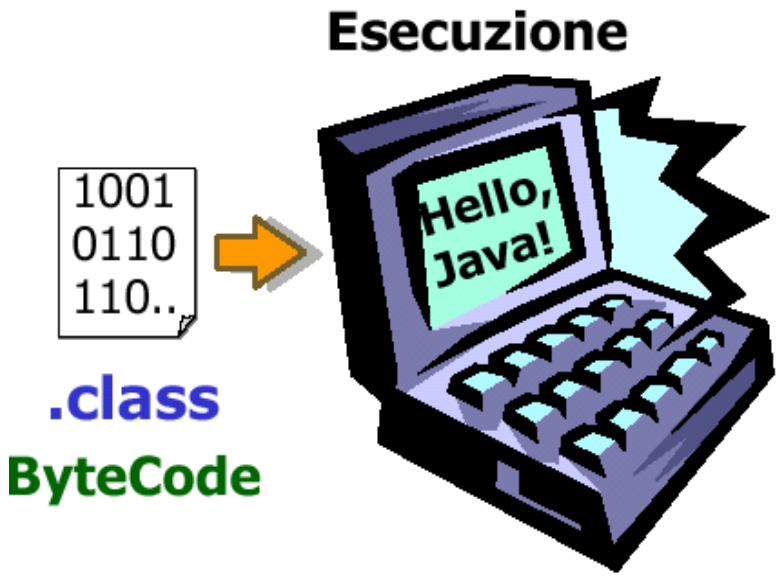
Non confondere un programma Java in tal senso, cioè un programma in Java, con *il* programma di nome *java*, che è l’interprete!

In realtà, eccetto che in queste note introduttive, non si parlerà mai del “programma java” nel secondo senso: si dirà “l’interprete java”, oppure la JVM (Java Virtual Machine).

Passi concettuali (I)



Passi concettuali (II)



Gli strumenti di Java

- La macchina virtuale
- Il compilatore

Ma anche:

- Gli ambienti di sviluppo
- La documentazione
- Il debugger
- …

Struttura dei file per Java (I)

- **<File>.java**
  - Contengono il codice sorgente Java
- **<File>.class**
  - Contengono il risultato della compilazione
  - Espressi in bytecode (formato binario)
  - All'interno sono presenti le istruzioni, la tabella dei simboli, le informazioni ausiliarie necessarie all'esecuzione

Struttura dei file per Java (II)

- **`<File>.jar`**
  - Archivi compressi che consentono di aggregare più file
  - Contengono una o più classi Java, risorse e ulteriori informazioni ausiliarie (documentazione, parametri di configurazione, …)
- **`<File>.properties`**
  - Coppie chiave – valore
  - proprietà del sistema o dell'applicazione
  - Si usano per configurare il sistema

Programmi in Java

- Java è un linguaggio orientato agli oggetti
  - L'unità minima di programmazione è la classe
  - Un programma Java utilizza una o più classi
- All'inizio, consideriamo programmi formati da una sola classe
  - Un solo file sorgente
  - Il nome del file coincide con il nome della classe
  - L'estensione del file è ".java"

Classi

Un programma Java può essere costituito di molti "pezzi", che si chiamano

classi o meglio dichiarazioni (o definizioni) di classi.

Nella programmazione "senza oggetti", una definizione di classe è un insieme di:

- definizioni (o dichiarazioni) di campi statici
- definizioni (o dichiarazioni) di metodi statici

campo statico: un *contenitore* di dato (numero, carattere,ecc.)

metodo statico: un sottoprogramma, dotato di un nome, che può agire sui dati contenuti nei campi statici

Confrontando Java "senza oggetti"
con i linguaggi *imperativi* tradizionali:

- una classe è l'analogo di un programma o di un modulo di un programma
- i campi statici sono simili alle variabili globali di un programma (con alcune differenze)
- i metodi statici sono simili alle procedure e alle funzioni di un programma, cioè ai sottoprogrammi di un programma

Il metodo `main(…)`: la "porta" sul mondo Java


```
public static void main(String[] args){
    /* istruzioni ... */
}
```

- Punto di ingresso di ogni applicazione
  - Invocato automaticamente dalla VM Java

Una delle classi deve contenere un metodo main, da cui comincia l'esecuzione: il metodo main potrà poi invocare altri metodi della stessa classe o di altre classi, che a loro volta potranno invocare altri metodi, e così via.

Ogni definizione di classe può essere messa in un file separato e compilata separatamente; un file non può contenere più di una classe pubblica, cioè dichiarata public.

Il file che contiene la classe in cui è definito il main deve avere lo stesso nome di tale classe, con l'aggiunta dell'estensione .java.

NB

In Java i campi statici, così come i campi non-statici che vedremo in seguito, *possono essere definiti soltanto all'interno di una definizione di classe*. La stessa cosa vale per i metodi statici (così come per quelli non-statici, come vedremo).

Esempio:

```
class MyClass {
  // campi statici:
  static int a, b;
  // definizione di metodo:
  static int myFun(int x, int y) {
    return 3*x - 4*y;
  }
  ...
```

Definizione e chiamata di una procedura (metodo)

Consideriamo la seguente definizione di classe:

```
class MyClass {
  // definizione di metodo:
  static int myFun(int x, int y) {
    int t = 3*x - 4*y;
    return t*t*t;
  }

  public static void main(String[] args) {
    int w = myFun(6,2); //chiamata della proc.
    System.out.println(w); ...
  }
}
```

myFun ha: parametri formali: x,y; variabili locali: t

IMPORTANTE: “Come” si scrive un programma Java

Buone norme stilistiche

- adottare un font a spaziatura fissa (ad es. Courier)
- fissare il tab (cioè la tabulazione) a 2 o 3 spazi e usarlo per l'*indentazione*
- indentare i costrutti sintattici annidati (vedi esempi a lezione) e allineare ogni graffa chiusa con la corrispondente aperta o meglio con l'inizio del corrispondente costrutto sintattico. Ad es., stile di indentazione raccomandato per i metodi:

```
void deposita(double...) {
  istruz;
  ...
}
```

- niente spazi fra nome del metodo e parentesi aperta:

  `void deposita(double...)` NON

  `void deposita (double...)`
- saltare una riga per separare un metodo dall'altro

Convenzioni universali di scrittura di programmi

- i nomi dei *campi*, delle *variabili* e dei *metodi* sono tutti minuscoli, eventualmente con maiuscole in mezzo, per distinguere paroleDiverse nelloStessoNome
- i nomi delle *classi* (e dei file .java che le contengono) iniziano con una maiuscola, seguita da minuscole, eventalmente con maiuscole in mezzo, per distinguere ParoleDiverse NelloStessoNome (es. `Hello` e `Hello.Java`)
- le *costanti* sono tutte maiuscole, eventualmente con carattere di sottolineatura per distinguere PAROLE_DIVERSE NELLO_STESSO_NOME (**NB** Usare _ solo nelle costanti, **non** usare – da nessuna parte)
- ATTENZIONE: Java è *case-sensitive*: "ciao" è diverso da "CIAO", che è diverso da "Ciao", che è diverso da "CIao"… Questo vale anche per i nomi dei file: pippo.java è diverso da Pippo.java
- Convenzioni di scrittura: **`http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html`**