Concetti introduttivi

La programmazione

- Programma: sequenza di operazioni semplici (istruzioni e decisioni) eseguite in successione
  - Un programma indica al computer i passi da compiere per svolgere un compito preciso
  - I programmi danno flessibilità di impiego ai computer

- L'attività di progettazione e implementazione dei programmi è detta *programmazione*.

- I programmi sono scritti utilizzando linguaggi di programmazione

Linguaggi di Programmazione

I linguaggi di programmazione sono in genere classificati in

- Linguaggi macchina
  - Istruzioni macchina codificate con sequenze numeriche
  - Dipendenti dalla macchina

```
30 40 16 100
156
```

- Linguaggi assembly
  - Istruzioni macchina codificate con codici mnemonici
  - Dipendenti dalla macchina

```
LOAD  REG, loc_b
ADD  REG, loc_a
MOV   loc_b, REG
```

- Linguaggi di alto livello (C, Pascal, Java,ecc.)
  - Istruzioni ad un livello concettuale più elevato
  - Indipendenti dalla macchina

```
b = a+b;
```

Linguaggi di alto livello

- I linguaggi di alto livello consentono un maggiore livello di astrazione
  - Permettono di descrivere l'idea che sta dietro l'operazione da compiere
    - Esempio: if x>0 then print("x è positivo")
      else print ("x è negativo")
- Sono più vicini ai linguaggi naturali
- Seguono delle convenzioni rigide per facilitarne la traduzione in codice macchina (compilazione)

Compilazione

- Le istruzioni scritte in un linguaggio di alto livello devono essere tradotte in istruzioni macchina per poter essere “comprese” dalla CPU
  - Il *compilatore* è il programma che si occupa di tradurre il codice

- L’insieme di istruzioni macchina (linguaggio macchina) dipende dalla CPU
  - → Il “back-end” di un compilatore dipende dalla CPU

Linguaggi di alto livello

- I linguaggi di alto livello sono classificati per tipologia
- Di interesse per il corso:
  - Linguaggi procedurali o imperativi
    - C, Pascal, …
  - Linguaggi orientati agli oggetti
    - C++, Java, …
- Altre tipologie di linguaggi:
  - Linguaggi funzionali
    - Lisp, SML, …
  - Linguaggi logici o dichiarativi
    - Prolog, LDL, …

Merge-sort imperativo (pseudo-codice)

```
MERGE-SORT(A, p, r)

1 if p < r
2    then q ← ⌊(p + r)/2⌋
3       MERGE-SORT(A, p, q)
4       MERGE-SORT(A, q + 1, r)
5       MERGE(A, p, q, r)
```

```
MERGE(A, p, q, r)
1   n1 ← q - p + 1,  n2 ← r - q
2   create arrays L[1:n1+1], R[1:n2+1]
3   for i ← 1 to n1  do L[i] ← A[p+i-1]
4   for j ← 1 to n2  do R[j] ← A[q+j]
5   L[n1+1] ← ∞,  R[n2+1] ← ∞
6   i ← 1,   j ← 1
7   for k ← p to r
8      do if L[i] ≤ R[j]
9            then  A[k] ← L[i],   i ← i+1
10           else  A[k] ← R[j],   j ← j+1
```

Merge-sort funzionale (ML)

```
fun split(nil) = (nil,nil)   |  split(a::nil) = ([a],nil)
          |   split (a::b::y) = (a::fst(split(y)),b::snd(split(y)));   (partizione lista)


fun merge(nil, l) = l   |   merge (l, nil) = l
      |    merge(h::t, k::s) =   if h < k then  h :: merge(t,k::s)
                                          else k :: merge(h::t, s);

fun mergeSort(nil) = nil   |   mergeSort(a::nil) = [a]
|   mergeSort(a::l) =
          merge(mergeSort(fst(split(a::l)), mergeSort(snd(split(a::l))));
```

Merge-sort logico (predicati)

```
split([ ], [ ], [ ]).
split([A], [A], [ ]).
split(A|B|X, A|Y, B|Z) :-  split(X, Y, Z).

merge(A, [ ], A).
merge([ ], A, A).
merge(A|X, B|Y, A|Merged) :- A<B AND merge(X, B|Y, Merged).
merge(A|X, B|Y, B|Merged) :- B<=A AND merge(A|X, Y,Merged).

mergesort([ ],[ ]).
mergesort([A],[A]).
mergesort(A|B|X, Sorted)  :-   split(A|B|X, L1, L2)
                               AND mergesort(L1, SortedL1)
                               AND mergesort(L2, SortedL2)
                               AND merge(SortedL1, SortedL2, Sorted)
```

Limite dei linguaggi procedurali

- Costringe a pensare soluzioni che riflettono il modo di operare del computer piuttosto che la struttura stessa del problema.
  - Per problemi non numerici questo spesso è difficile
  - Il riutilizzo delle soluzioni è più complicato e improbabile
  - La produzione e la manutenzione del software sono costose

Linguaggi Orientati agli Oggetti

- I *linguaggi ad oggetti* permettono al programmatore di rappresentare e manipolare non solo dati numerici o stringhe ma anche dati più complessi e aderenti alla realtà (conti bancari, schede personale,…)
  - Progettazione e sviluppo più semplice e veloce
  - Alta modularità
    - Estensibilità e manutenzione più semplici
- Tutto questo si traduce in costi più bassi

Esistono controindicazioni?

- Il paradigma di programmazione orientata agli oggetti paga la sua semplicità e versatilità in termini di efficienza

- Va molto bene per lo sviluppo di applicazioni, ma non è adatto per lo sviluppo di software di base
  - Sistemi operativi
  - Driver
  - Compilatori

Introduzione al linguaggio Java

Un po' di storia: versioni Java

- 21/1/1996 Java 1.0 (prima versione)
- Numerazione fino alla quinta versione: 1.1, 1.2, 1.3, 1.4
- 30/9/2004 Java 5.0:
  - Classi generiche, ciclo for each, auto-boxing/unboxing, enumerazioni, strutture concorrenti (java.util.concurrent)

  ….
- 19/3/2019 Java 12.0

- Storia dettagliata:
  http://en.wikipedia.org/wiki/Java_version_history

JAVA: caratteristiche generali (1)

- E' object-oriented :
  - risponde all'esigenza di realizzare sistemi software facili da modificare e mantenere
  - consente alti livelli di *riutilizzabilità* del codice
- Ha una ricchissima libreria per lo sviluppo di interfacce utente e di applicazioni Internet impiegabili con relativa facilità
- E' robusto
  - Una delle principali cause di *crash* dei programmi scritti in C/C++ è l'uso scorretto dell'aritmetica dei puntatori:
    - non fornisce tipi puntatori, né tanto meno l'aritmetica dei puntatori

JAVA: caratteristiche generali (2)

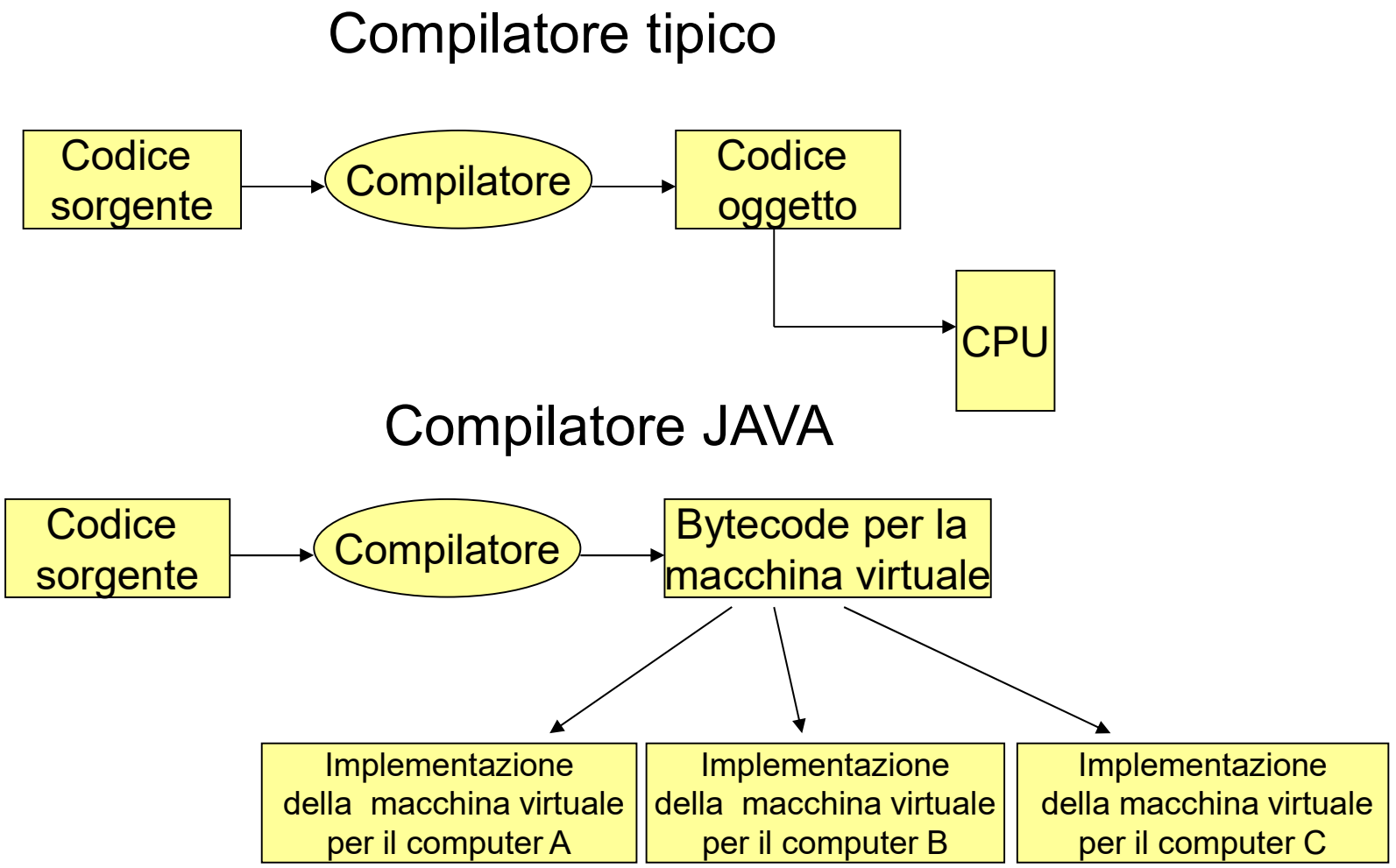
- E' efficiente pur essendo un linguaggio interpretato:
  - i programmi Java sono mediamente meno di 10 volte più lenti dei corrispondenti programmi C++
  - riduzione di efficienza accettabile per tipiche applicazioni Java: programmi interattivi
  - meglio di altri linguaggi interpretati (Basic, PERL, etc.)
- E' sicuro:
  - esecuzione programmi confinata in un "firewall" da cui non è possibile accedere ad altre parti del computer
    - estremamente utile per l'esecuzione di programmi scaricati da internet

JAVA: caratteristiche generali (3)

- E' portabile:
  - programmi scritti in tipici linguaggi compilati come il C/C++ devono essere ricompilati ogni nuova piattaforma
    - Programmi scaricabili da internet: bisogna predisporre l'eseguibile per ogni tipo di CPU o mettere a disposizione il codice sorgente
  - JAVA:

    il compilatore genera un codice (Bytecode) eseguibile per una CPU virtuale, detta macchina virtuale Java (JVM)
    - La macchina virtuale viene poi simulata su una CPU reale

Bytecode e JVM



JAVA: caratteristiche generali (4)

- E’ semplice da usare:
  - se si conosce programmazione ad oggetti
  - caratteristiche complesse di C++ eliminate o realizzate in maniera più semplice
  - non ci sono caratteristiche insolite o sorprendenti
    - esistono sempre pochi modi ben chiari per eseguire un determinato compito
  - poche caratteristiche di base
    - estendibili (se necessario) con le librerie

Un semplice programma Java

```
1  public class HelloPrinter
2  {
3     public static void main(String[] args)
4     {
5        // Display a greeting in the console window
6
7        System.out.println("Hello, World!");
8     }
9  }
```

Esecuzione produce nella console:

```
Hello, World!
```

Struttura di un semplice programma: dichiarazione di una classe

- Le classi sono gli elementi fondamentali di un programma Java:
  - `public class HelloPrinter`

    introduce la definizione di una nuova classe
- Limitazione: ogni file sorgente contiene al più una classe public
- Il nome di questa classe deve coincidere con il nome del file che la contiene (estensione esclusa) :
  - *La classe `HelloPrinter` deve essere contenuta nel file `HelloPrinter.java`*

Struttura di un semplice programma: il metodo `main`

- Ogni programma Java contiene una classe con il metodo main
  - *L'esecuzione del programma parte dal metodo main*

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

definisce il metdo `main`

Struttura di un semplice programma: i commenti

- La prima riga nel metodo main è un commento:

```
// Display a greeting in the console window
```

- Il compilatore ignora ogni carattere che segue `//` fino alla fine della linea
- I commenti servono per documentare il codice e facilitarne la comprensione ad un essere umano

Struttura di un semplice programma: le istruzioni

- Il corpo del metodo main contiene le istruzioni comprese tra le parentesi graffe
- ogni istruzione termina con “;”
- Le istruzioni vengono eseguite una alla volta nell’ordine in cui sono scritte
- Il metodo considerato ha una sola istruzione:

```
System.out.println("Hello, World!");
```

  - Il suo effetto è di scrivere nella console:

```
Hello, World
```

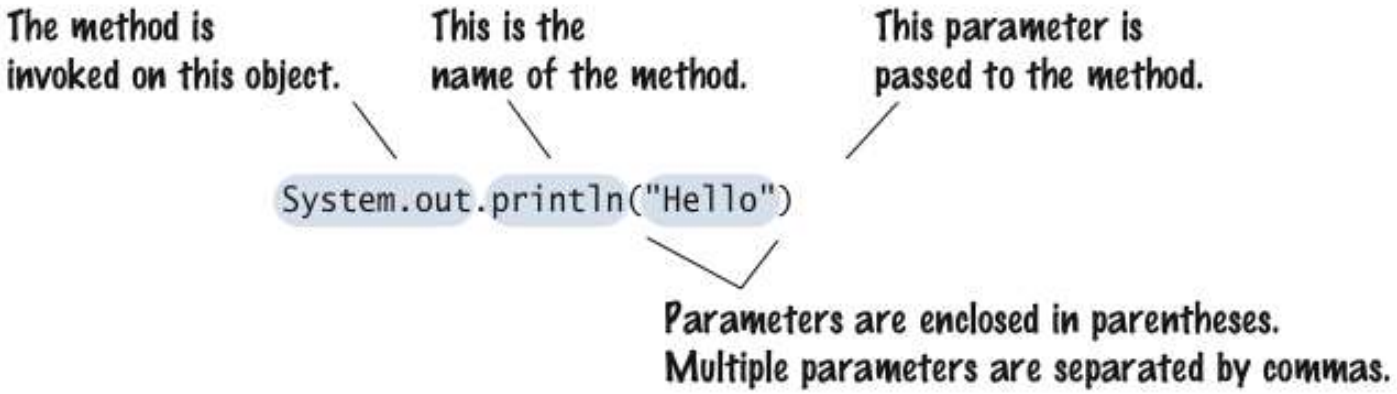
Struttura di un semplice programma: invocazione di un metodo

- `System.out.println("Hello, World!");`
  - invoca il metodo `println`
- L'invocazione di un metodo richiede:
  1. *L'oggetto che vogliamo utilizzare (`System.out`)*
  2. *Il nome del metodo da utilizzare (`println`)*
  3. ***I parametri** racchiusi tra parentesi tonde (un solo parametro la stringa `"Hello, World!"`)*

Sintassi: invocazione di un metodo

Syntax *object.methodName(parameters)*

Example



Struttura di un semplice programma: stringhe

- **Stringa:** una sequenza di caratteri racchiusa tra doppi apici

```
"Hello, World!"
```

Due semplici quesiti.....

- Come modifichereste `HelloPrinter` per scrivere le parole `"Hello,"` e `"World!"` su due linee?
  - **Risposta:**

```
System.out.println("Hello,");
System.out.println("World!");
```

- Il programma funziona ancora se viene omessa la riga con `//`?
  - **Risposta:** Si

Compilare ed eseguire un programma Java

- Il compilatore traduce il codice sorgente in file `.class` (class file) che contengono il bytecode per la Java Virtual Machine (JVM)
- Il compilatore non produce il class file se ha riscontrato un errore
- La Java virtual machine:
  - carica le istruzioni dai class file,
  - inizia l'esecuzione del programma e
  - carica le librerie necessarie quando sono richieste

Schema di compilazione Java

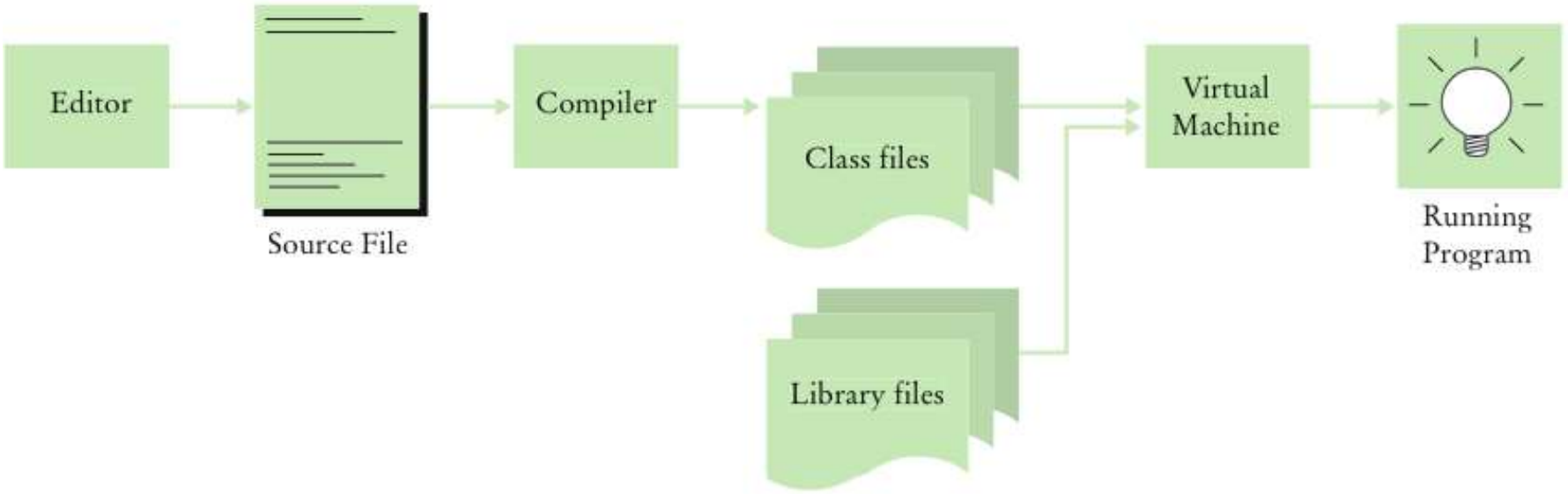


Figure 10 From Source Code to Running Program

Errori

- **Compile-time error:** una violazione delle regole del linguaggio individuate dal compilatore
  - *Es.*: `System.ou.println("Hello, World!);`
  - *Syntax error*

- **Run-time error:** il programma ha un comportamento non voluto dal programmatore
  - *Es:* `System.out.println("Hello, Word!");`

    `System.out.println(1/0);`
  - *errore logico*

Realizzazione di programmi corretti

- Conoscere gli errori comuni e imparare ad evitarli
- Utilizzare strategie di programmazione “difensive” per minimizzare l’impatto degli errori
- Applicare tecniche e metodi per l’individuazione e la correzione degli errori residui (testing, verifica, debugging, etc.)