AMBIENTI DI PROGRAMMAZIONE

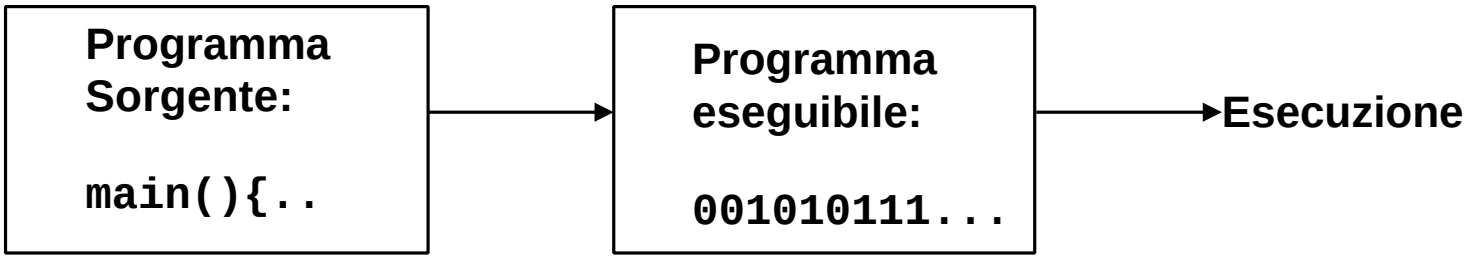
È l'insieme dei programmi che consentono la scrittura, la verifica e l'esecuzione di nuovi programmi (***fasi di sviluppo***)

Sviluppo di un programma

- Affinché un programma scritto in un qualsiasi linguaggio di programmazione sia comprensibile (e quindi eseguibile) da un calcolatore, *occorre* ***tradurlo*** dal linguaggio originario al linguaggio della macchina

Questa operazione viene normalmente svolta da speciali strumenti, detti ***traduttori***

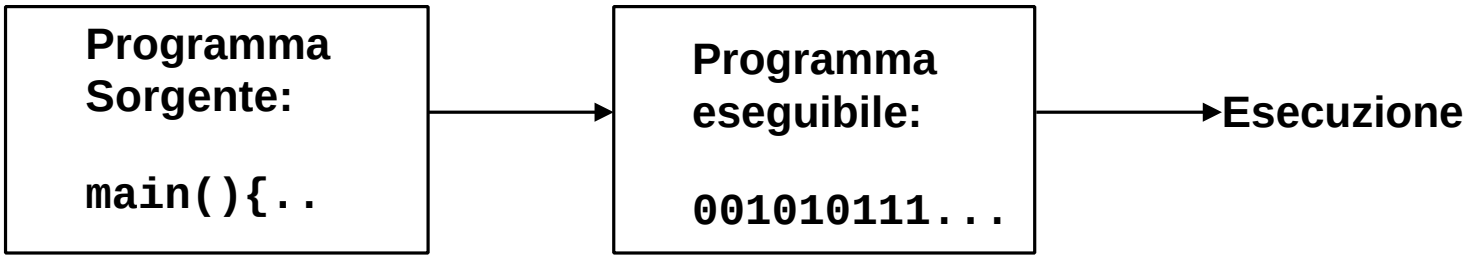
SVILUPPO DI PROGRAMMI



Due categorie di traduttori:

- i ***Compilatori*** traducono l'intero programma e producono il programma in linguaggio macchina
- gli ***Interpreti*** traducono ed eseguono immediatamente **ogni singola istruzione** del *programma sorgente*

SVILUPPO DI PROGRAMMI (segue)



Quindi:

- **nel caso del compilatore**, lo schema precedente viene percorso ***una volta sola*** prima dell'esecuzione
- **nel caso dell'interprete**, lo schema viene invece attraversato ***tante volte quante sono le istruzioni*** che compongono il programma

COMPILATORI E INTERPRETI

- I **compilatori** traducono automaticamente un programma dal linguaggio di alto livello a quello macchina (per un determinato elaboratore)

- Gli **interpreti** sono programmi capaci di eseguire direttamente un programma nel linguaggio scelto, istruzione per istruzione

- I programmi compilati sono in generale più efficienti di quelli interpretati

AMBIENTI DI PROGRAMMAZIONE

I° CASO: COMPILAZIONE

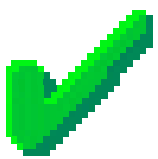
- **Compilatore:** opera la **traduzione di un programma *sorgente*** (scritto in un linguaggio ad alto livello) **in un *programma oggetto*** direttamente eseguibile dal calcolatore
- **Linker:** (*collegatore*) nel caso in cui la costruzione del programma oggetto richieda l'unione di ***più moduli*** (compilati separatamente), il linker provvede a **collegarli** formando un unico *programma eseguibile*

AMBIENTI DI PROGRAMMAZIONE

II° CASO: INTERPRETAZIONE

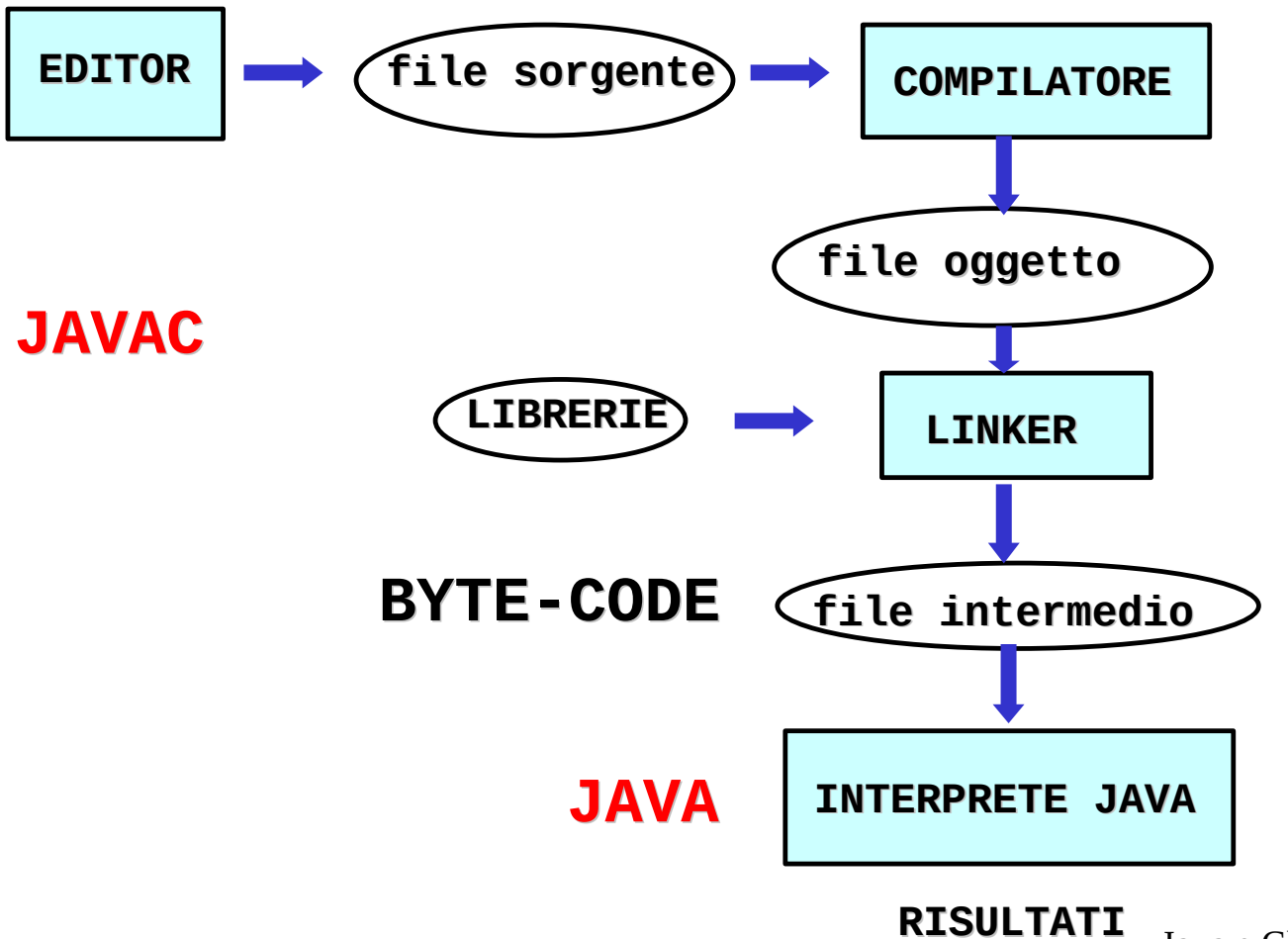
- **Interprete**: ***traduce ed esegue*** direttamente ***ciascuna istruzione*** del *programma sorgente*, **istruzione per istruzione**

  È generalmente in alternativa al compilatore (raramente presenti entrambi)

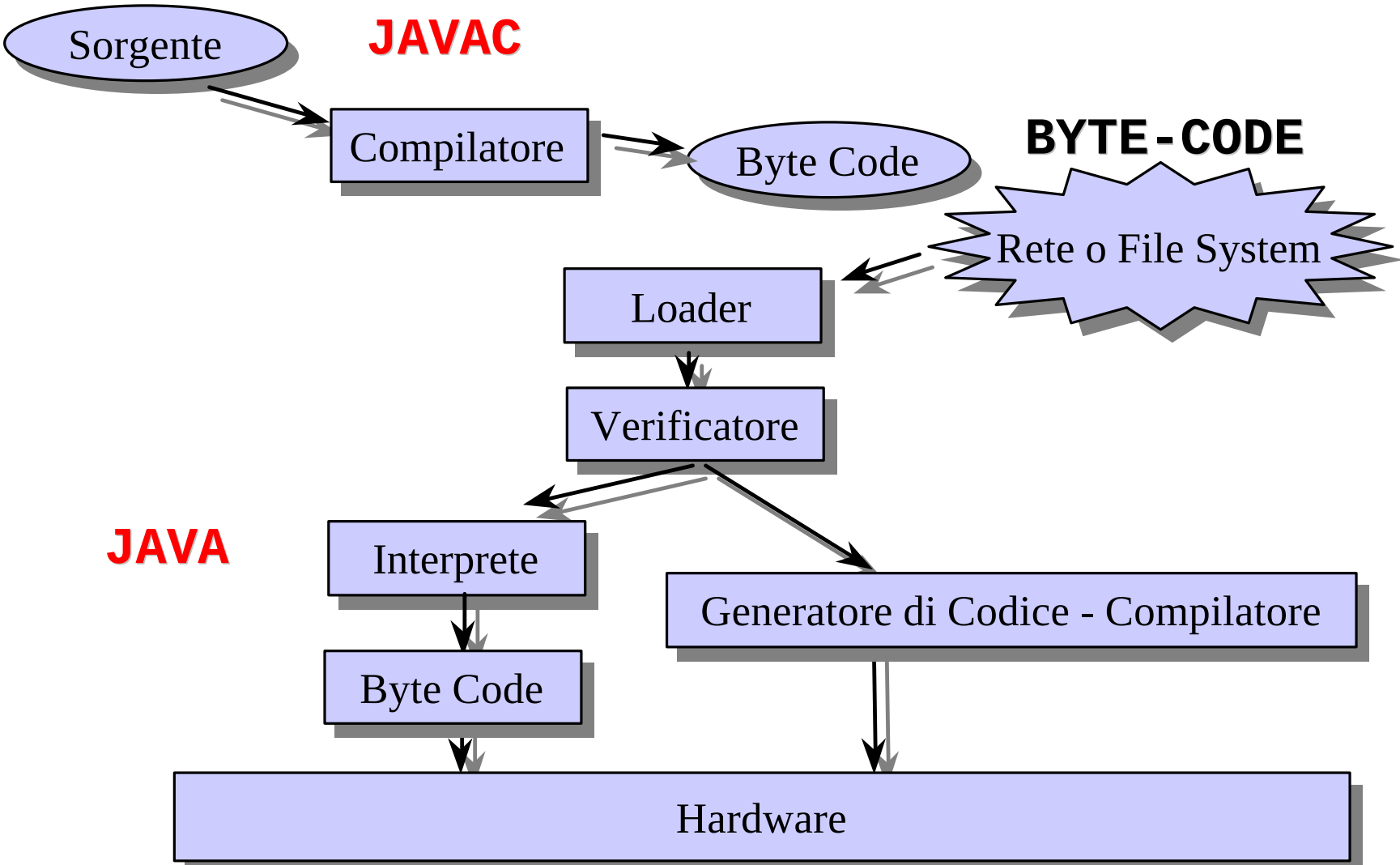


Traduzione ed esecuzione sono *intercalate*, e avvengono *istruzione per istruzione*

APPROCCIO MISTO



APPROCCIO JAVA



IL Java Development Kit (JDK)

Il JDK della Sun Microsystems è l'insieme di strumenti di sviluppo che funge da "*rife-rimento ufficiale*" del linguaggio Java

- ***non è un ambiente grafico integrato:***
  è solo un insieme di strumenti da usare dalla linea di comando
- ***non è particolarmente veloce ed efficiente***
  (non sostituisce strumenti commerciali)
- **però funziona, è gratuito ed esiste per tutte le piattaforme** (Win32, Linux, Solaris, Mac..)

... E OLTRE

Esistono molti strumenti tesi a migliorare il JDK, e/o a renderne più semplice l'uso

- ***editor con "syntax highlightling"***
  - TextTool, WinEdt, JPad, e tanti altri
- ***ambienti integrati freeware*** **che, pur usando "sotto" il JDK, ne consentono l'uso in modo interattivo e in ambiente grafico**
  - FreeBuilder, Forte, Jasupremo, etc...
- ***ambienti integrati commerciali,*** **dotati di compilatori propri e debugger**
  - Jbuilder, Codewarrior, VisualAge for Java, ...

COMPILAZIONE ED ESECUZIONE

Usando il JDK della Sun:

- ***Compilazione:***

  `javac Esempio0.java`

  (produce **`Esempio0.class`**)

- ***Esecuzione:***

  `java Esempio0`

Non esiste una fase di link esplicita:
Java adotta il *collegamento dinamico*

IL LINGUAGGIO JAVA

- **È un linguaggio *totalmente a oggetti:* tranne i tipi primitivi di base (`int`, `float`, ...), *esistono solo classi e oggetti***
- **È fortemente ispirato al C++, ma riprogettato *senza il requisito della piena compatibilità col C* (a cui però assomiglia…)**
- **Un programma è un insieme *di classi***
  - **non esistono funzioni definite (come in C) a livello esterno, né variabili globali esterne**
  - ***anche il main è definito dentro a una classe!***

TIPI DI DATO PRIMITIVI IN JAVA

- **caratteri**
  - **`char` (2 byte)** **codifica UNICODE**
  - **coincide con ASCII sui primi 127 caratteri**
  - **e con ANSI / ASCII sui primi 255 caratteri**
  - ***costanti char anche in forma `'\u2122'`***
- **interi (con segno)**
  - **`byte` (1 byte)** **-128 ... +127**
  - **`short`(2 byte)** **-32768 ... +32767**
  - **`int` (4 byte)** **-2.147.483.648 ... 2.147.483.647**
  - **`long` (8 byte)** **$-9\ 10^{18}$ ... $+9\ 10^{18}$**

NB: le costanti long terminano con la lettera L

TIPI DI DATO PRIMITIVI IN JAVA

- **reali *(IEEE-754)***
  - **`float` (4 byte)** **-10^{45} ... $+10^{38}$**
    (6-7 cifre significative)
  - **`double` (8 byte)** **-10^{328} ... $+10^{308}$**
    (14-15 cifre significative)
- **boolean**
  - **`boolean` (1 bit)** **`false` e `true`**
  - **tipo autonomo *totalmente disaccoppiato dagli interi:* non si convertono boolean in interi e viceversa, *neanche con un cast***
  - **tutte le espressioni relazionali e logiche danno come risultato un `boolean`, non più un `int`!**

OPERATORI CONDIZIONALI IN JAVA

- **Sintatticamente vengono ereditate le stesse regole presenti nel c e c++:**
- **blocchi condizionali if{..}else if{…}else{…}**
- **blocchi switch(exp){ case:…}**
- **cicli for(int i=0;i<y;i++){}, while(exp){} e do{...}while(exp);**

OPERATORI IN JAVA

- **Sintatticamente vengono ereditate le stesse regole presenti nel c e c++:**
- **assegnamento: a=1;**
- **ass.to con operazione: a+=b, a|=b, a*=b,...;**
- **condizionali: (a && b), (a || b),…;**
- **condizionali ternari: (a<b)?…:…;**

CLASSI E FILE

- **In Java esiste una *ben precisa corrispondenza* fra**
  - ***nome di una classe pubblica***
  - ***nome del file* *in cui essa dev'essere definita***
- **Una classe *pubblica* deve essere definita in un file *con lo stesso nome della classe* ed estensione `.java`**
- **Esempi**
  classe `EsempioBase` ® file `EsempioBase.java`
  classe `Esempio0` ® file `Esempio0.java`

COLLEGAMENTO STATICO...

Nei linguaggi "classici":

- si compila ogni file sorgente
- ***si collegano i file oggetto così ottenuti***

In questo schema:

- ogni file sorgente **dichiara** tutto ciò che usa
- il compilatore ne accetta l'uso "condizionato"
- il linker **verifica la presenza delle definizioni** risolvendo i *riferimenti incrociati* fra i file
- ***l'eseguibile è "autocontenuto"*** (**non** contiene **più** riferimenti a entità esterne)

.. E COLLEGAMENTO DINAMICO

In Java

- **non esistono dichiarazioni!**
- si compila ogni file sorgente, ***e si esegue la classe pubblica che contiene il `main`***

In questo schema:

- il compilatore accetta l'uso di altre classi perché ***può verificarne esistenza e interfaccia*** in quanto ***sa dove trovarle nel file system***
- **le classi usate vengono *caricate dall'esecutore solo al momento dell'uso***

ESECUZIONE E PORTABILITÀ

In Java,

- **ogni classe è compilata in un file `.class`**
- **il formato dei file `.class` ("bytecode") non è direttamente eseguibile: *è un formato portabile, inter-piattaforma***
- **per eseguirlo occorre un *interprete Java***
  - **è l'unico strato *dipendente dalla piattaforma***
- **in questo modo si ottiene *vera portabilità:* un file `.class` compilato su una piattaforma *può funzionare su qualunque altra!!!***

Esempio 1: Hello world!

- Scriviamo il “mitico” programma Hello World in Java

```
public class HelloWorld
{
  public static void main(String args[])
  {
   System.out.println("Hello world!");
  }
}
```

Esempio 2: implementazione

```
public class Counter
{
  private int val;
  public void reset() { val = 0; }
  public void inc(){ val++; }
  public int getValue() { return val;}
}

public class Esempio
{
   public static void main(String[] args)
   {
    int n;
    Counter c1;
    c1 = new Counter();
    c1.inc();
    n = c1.getValue();
    System.out.println(n);
   }
 }
```

Riferimenti e oggetti - Copia

- ```
  public class Counter
  {
    private int val;
    public void reset() { val = 0; }
    public void inc(){ val++; }
    public int getValue() { return val;}
    public void copy(Counter c2) { val = c2.val;}
  }
  ```
- Poi si procede così:

```
Counter c1, c2; /* Dichiariamo due variabili */
c1 = new Counter(); /* creiamo due istanze */
c2 = new Counter();
c2.copy(c1); /* copiamo lo stato di c1 in c2 */
```
```

Riferimenti e Uguaglianza –fra oggetti

```
public class Counter
{
  private int val;
  public void reset() { val = 0; }
  public void inc(){ val++; }
  public int getValue() { return val; }
  public void copy(Counter c2) { val = c2.val };
  public boolean equals(Counter c2){
     return val == c2.val;
  }
}
```

- Poi si procede così:

```
Counter c1, c2;
boolean b1,b2;
c1 = new Counter(); c1.inc();
c2 = new Counter();
b1 = c1.equals(c2); /* b1 vale false */
c1.copy(c2);
b2 = c1.equals(c2); /* b2 vale true */
```

Caratteristiche dei costruttori in Java

```
public class Counter
{
  private int val;
  public Counter() { val = 0; }
  public void reset() { val = 0; }
  public void inc(){ val++; }
  public int getValue() { return
val;}
}
```

Costruttori multipli - 1

```
public class Counter
{
  private int val;
  public Counter() { val = 0; }
  public Counter(int n) { val = n; }
  public void reset() { val = 0; }
  public void inc(){ val++; }
  public int getValue() { return val;}
}
```

- In questo caso abbiamo la possibilità di creare un contatore con un valore iniziale prefissato

Overloading - 2

```
public class Counter
{
  private int val;
  public Counter() { val = 0; }
  public void reset() { val = 0; }
  public void inc(){ val++; }
  public void inc(int n){ val = val +
n; }
  public int getValue() { return val;}
}
Counter c1;
c1 = new Counter();
c1.inc(); /*Viene invocata la I ver.*/
c2.inc(3);/*Viene invocata la II ver.*/
```

Esercizio

- Si vuole realizzare una calcolatrice in grado di eseguire le quattro operazioni;
- deve potere mantenere una memoria temporanea ed eseguire su questa le operazioni di somma e di sottrazione;
- deve potere stampare il risultato delle operazioni eseguite e del contenuto della memoria