AMBIENTI DI PROGRAMMAZIONE

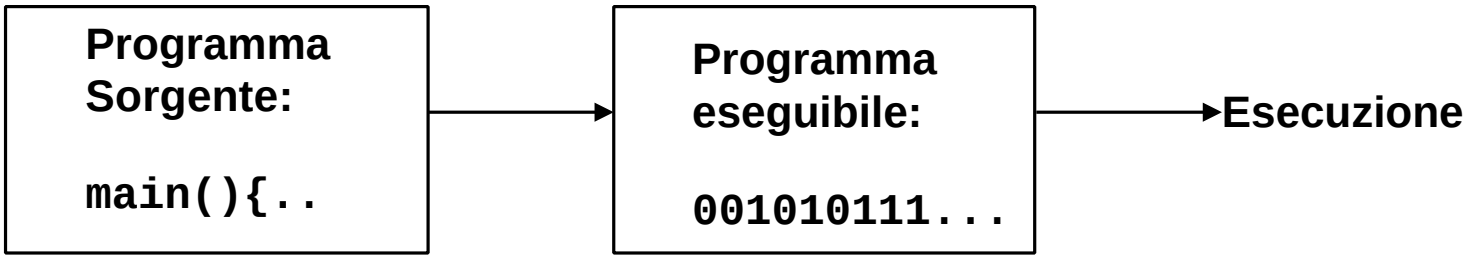
È l'insieme dei programmi che consentono la scrittura, la verifica e l'esecuzione di nuovi programmi (***fasi di sviluppo***)

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***

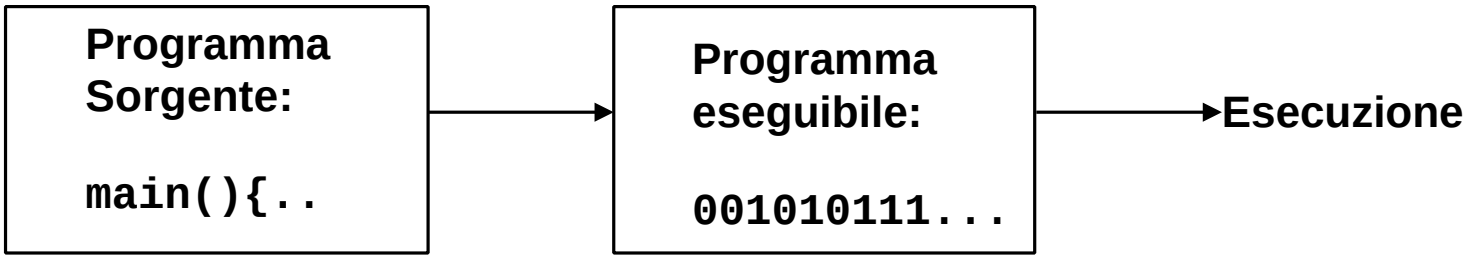
SVILUPPO DI PROGRAMMI



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

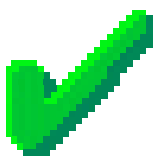
- **Compilatore:** opera la **traduzione di un programma *sorgente*** (scritto in un linguaggio ad alto livello) **in un *programma oggetto*** direttamente eseguibile dal calcolatore
- **Linker:** (*collegatore*) nel caso in cui la costruzione del programma oggetto richieda l'unione di ***più moduli***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

AMBIENTI DI PROGRAMMAZIONE

II° CASO: INTERPRETAZIONE

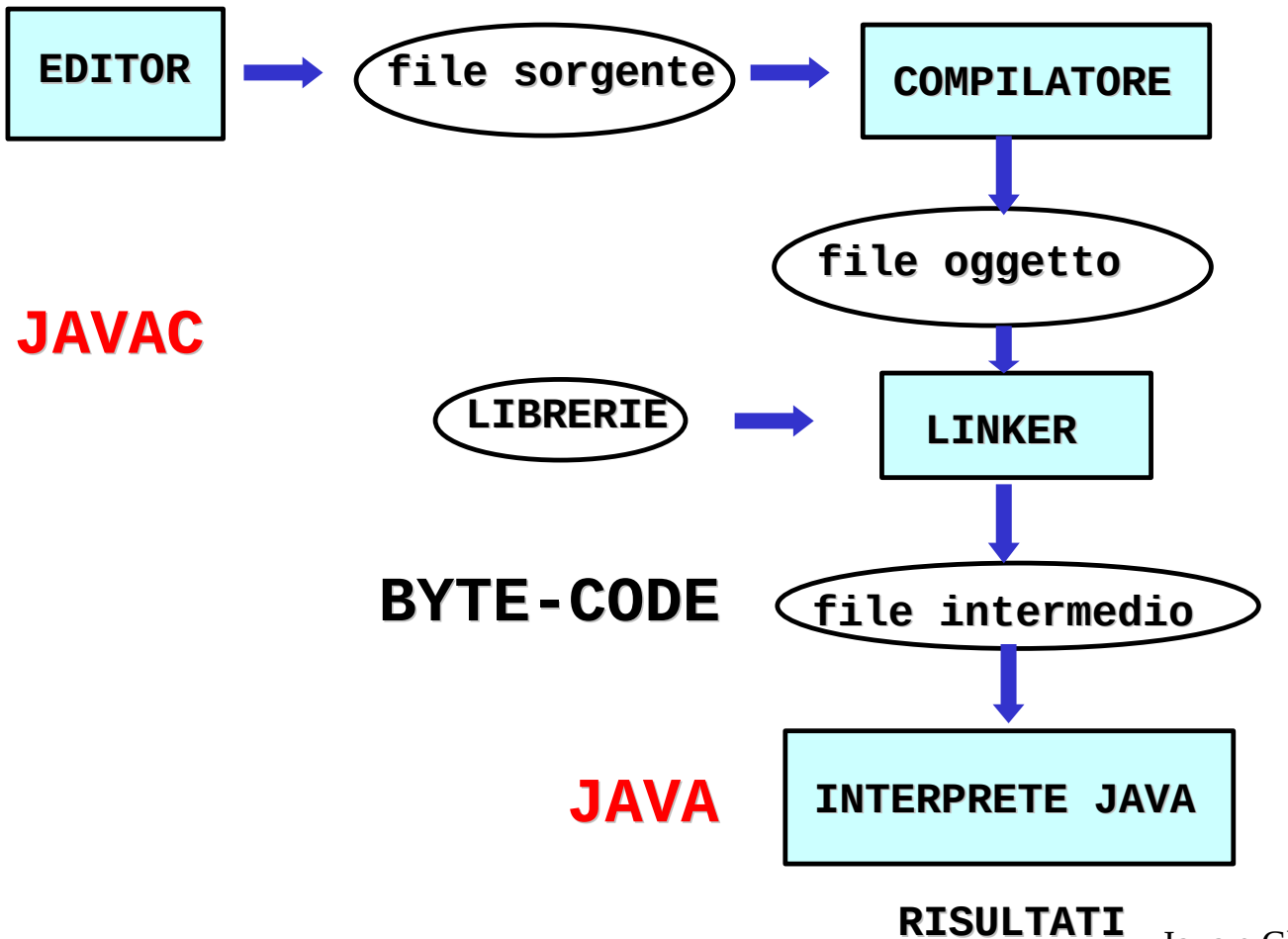
- **Interprete**: ***traduce ed esegue*** direttamente ***ciascuna istruzione*** del *programma sorgente*, **istruzione per istruzione**

  È generalmente in alternativa al compilatore (raramente presenti entrambi)

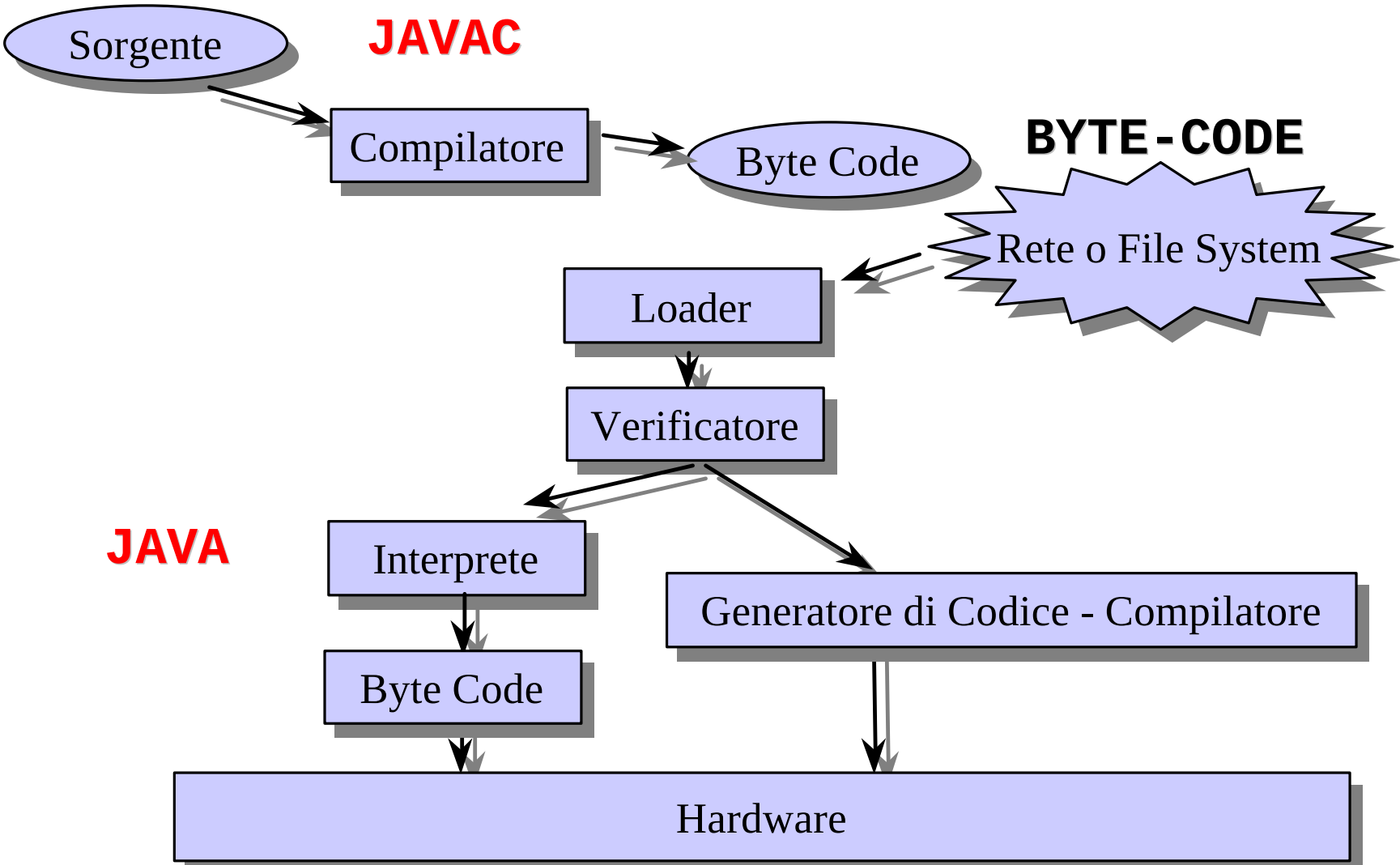


Traduzione ed esecuzione sono *intercalate*, e avvengono *istruzione per istruzione*

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!***

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!!!***