

1 LA SINTASSI JAVA E L'AMBIENTE DI SVILUPPO

UNITÀ DI APPRENDIMENTO

L1 L'ambiente
di programmazione

L2 La struttura
del codice

L3 Le variabili
e i tipi primitivi

L4 Le operazioni sui dati

L5 Le stringhe

L6 Le strutture di controllo

L7 Le strutture di controllo
derivate

OBIETTIVI

- Riconoscere il ruolo degli ambienti IDE
- Conoscere e utilizzare le librerie Java (JDK) e la JVM
- Gestire progetti con BlueJ
- Saper riconoscere le classi e gli operatori
- Conoscere la sintassi Java per le istruzioni di selezione e iterazione



Info

Nella cartella **AU01** del CD-ROM allegato al volume sono presenti i progetti e i file sorgenti utili per questa unità di apprendimento.

ATTIVITÀ

- Utilizzare l'editor e il debugger BlueJ
- Applicare i tipi e le conversioni offerti dal linguaggio Java
- Utilizzare le istruzioni per sequenza, selezione e iterazione in Java

LEZIONE 1

L'AMBIENTE DI PROGRAMMAZIONE

IN QUESTA LEZIONE IMPAREREMO...

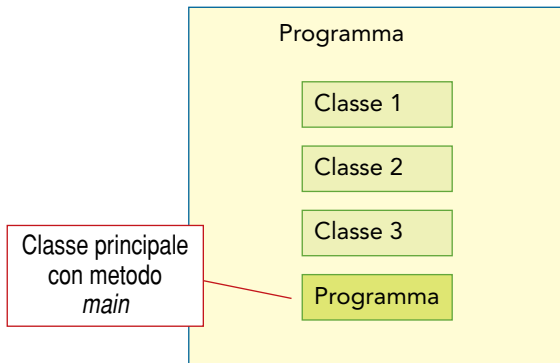
- a installare e utilizzare le librerie di Java (JDK) e la JVM
- a utilizzare gli strumenti di debugging di BlueJ
- a utilizzare i breakpoints e le variabili locali

I programmi in Java

Prima di poter scrivere un programma mediante il linguaggio ◀ Java ▶ vogliamo introdurre il concetto di **ambiente di programmazione** di questo linguaggio. Innanzi tutto è necessario chiarire il concetto fondamentale di **classe**.



◀ Java Un programma Java si compone di una o più **classi**. Ogni classe risiede in un **file** che possiede lo stesso nome della classe con estensione **.java**, e inoltre deve esistere almeno una classe con il nome del programma e con un metodo speciale chiamato **main**. ▶



Tuttavia per i primi esempi utilizzeremo una sola classe che fungerà anche da programma includendo al suo interno il metodo **main**.


■ L'ambiente di programmazione

Per eseguire una qualunque applicazione dobbiamo svolgere tre attività:

- 1 **editing**: mediante questa fase rendiamo il programma accessibile al computer;
- 2 **compilazione**: mediante questa fase traduciamo il programma in un formato eseguibile dal computer;
- 3 **esecuzione**: mediante questa fase facciamo eseguire il programma al computer.

Editing

Per rendere accessibile al calcolatore una **classe** o un **programma** dobbiamo memorizzare la definizione di ciascuna classe all'interno di un file di testo con estensione **.java**. La definizione della classe prende il nome di **codice sorgente** e può essere scritto mediante un programma chiamato **Editor**. La figura seguente mostra l'editing di una semplice applicazione in Java:



Ciascuna classe Java è contenuta all'interno di un singolo file di testo, il cui nome riflette quello della classe. Non è possibile memorizzare più classi pubbliche all'interno di uno stesso file di testo; in questo caso il compilatore segnala un errore di sintassi.

La compilazione e l'esecuzione in Java

Un **compilatore** è un programma che in grado di tradurre programmi scritti in un linguaggio di programmazione nel **linguaggio macchina** del computer.



◀ **Linguaggio macchina** È un linguaggio di programmazione molto più elementare e primitivo di Java, ed è specifico di un computer. ▶

La fase di compilazione traduce da un linguaggio di alto livello a un linguaggio macchina: ovvero dal codice sorgente scritto in linguaggio di alto livello al codice eseguibile scritto in linguaggio macchina.

Tuttavia la compilazione e l'esecuzione sono dipendenti dall'ambiente hardware e software e un compilatore in generale è in grado di tradurre uno specifico linguaggio di programmazione (come per esempio il linguaggio C++) in uno specifico linguaggio macchina relativo a uno specifico processore (per esempio Intel MMX) e inoltre relativamente a uno specifico sistema operativo (per esempio Windows 8). In tal modo il codice eseguibile generato da un compilatore potrà essere eseguito soltanto dai computer corredati di uno specifico ambiente **hardware/software**.

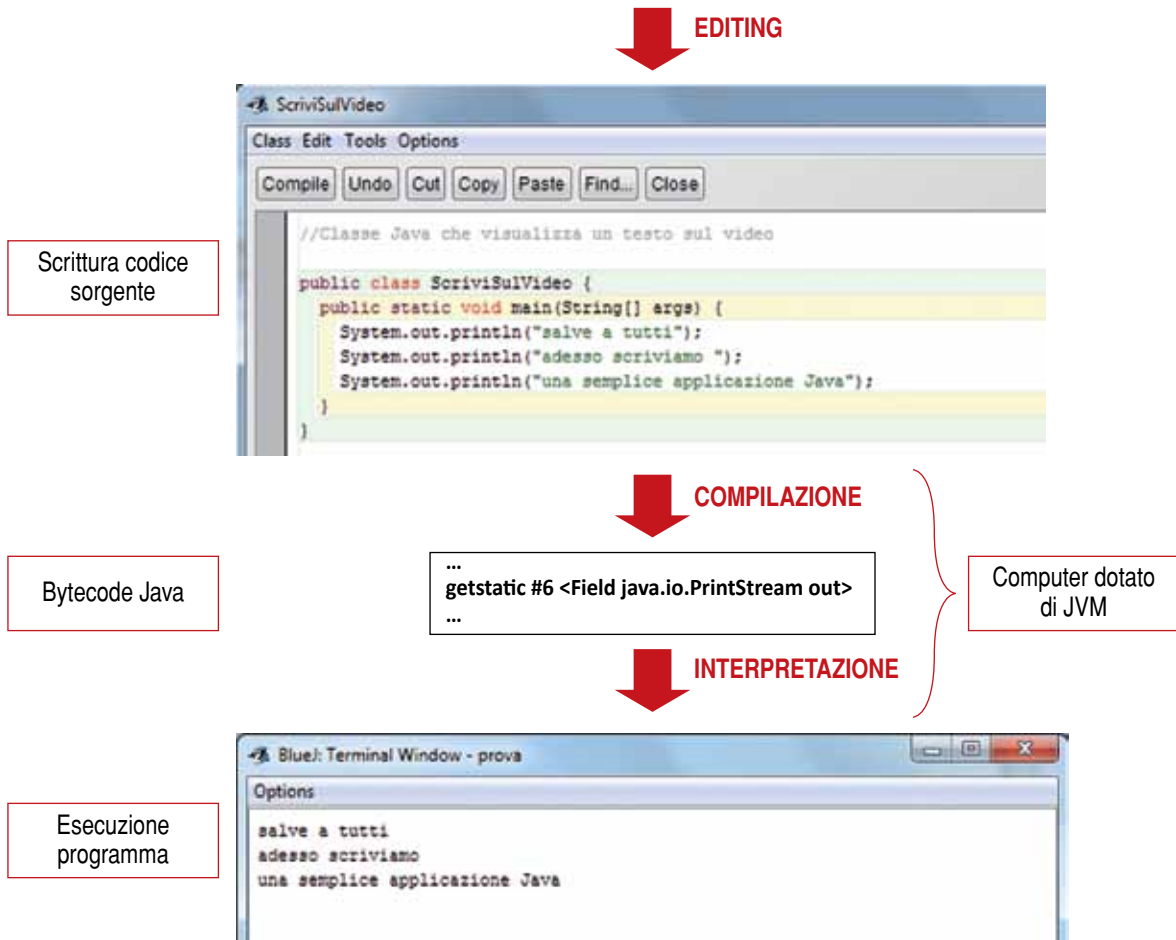
Java utilizza un approccio molto particolare per la compilazione dei programmi, infatti possiamo compilare il codice sorgente scritto in Java ottenendo un codice misto chiamato **bytecode Java**, una sorta di linguaggio assembly di un calcolatore virtuale. Inoltre il programma nella forma di bytecode Java potrà essere eseguito da un interprete chiamato **JVM** (Java Virtual Machine).



◀ **JVM** È un'applicazione che sa eseguire il **bytecode Java** e inoltre rende il computer una **macchina virtuale** in grado di eseguire programmi in bytecode Java. ▶

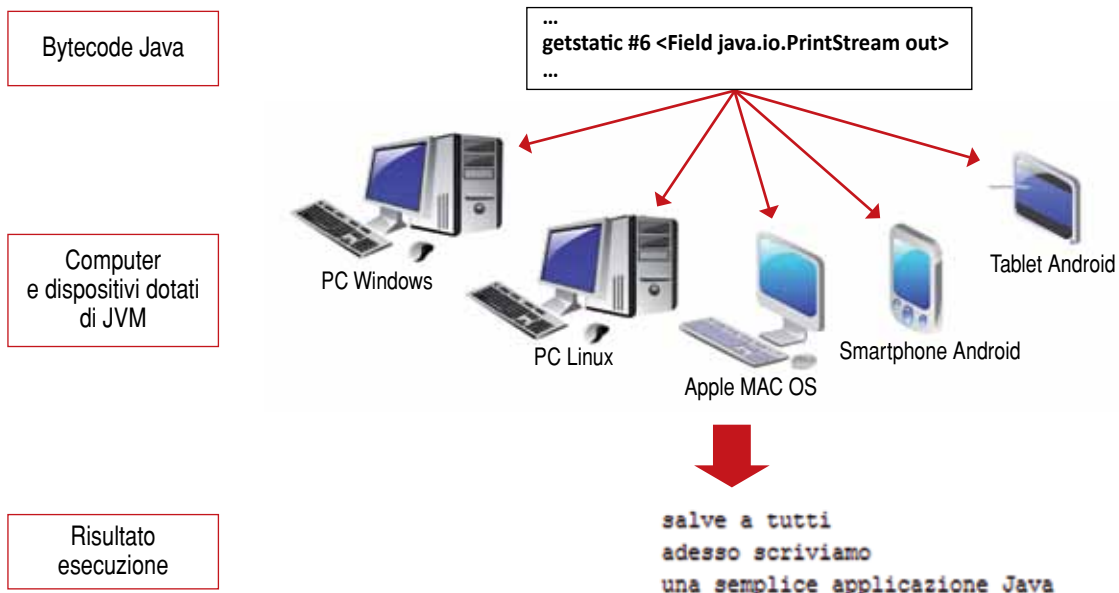
Abbiamo un diverso compilatore Java e una diversa macchina virtuale JVM per ciascun ambiente hardware/software. Il bytecode Java prodotto in uno specifico ambiente hardware/software potrà essere infatti eseguito in qualsiasi altro ambiente hardware/software, purché dotato di una macchina virtuale Java apposita.

In sintesi le fasi che consentono di compilare ed eseguire un programma scritto in Java sono così riassunte:



Una caratteristica Java è proprio quella di essere un codice sorgente scrivibile su una piattaforma qualsiasi ed eseguibile su altrettante diverse piattaforme secondo il motto **“Write once, run everywhere”**.

Come possiamo notare nella figura seguente, una volta compilato il sorgente e generato il bytecode esso può essere eseguito su piattaforme tra loro diverse sia come caratteristiche hardware che software:



■ Programmare in Java

Per poter programmare in Java dobbiamo procurarci:

- ▶ un **compilatore**;
- ▶ un **ambiente di sviluppo**.

Il primo si chiama **JSESDK** (**Java Standard Edition Software Development Kit** – spesso abbreviata in **JDK**) e consente di compilare i programmi realizzati in Java.

Lo si può scaricare gratuitamente dal sito della Oracle (www.oracle.com/technetwork/java/javase/downloads)

Il secondo programma di cui abbiamo bisogno è l'**ambiente di sviluppo** che funziona appoggiandosi al **JDK** scaricato prima. Esistono numerosi ambienti di sviluppo di tipo **◀ IDE ▶** tra cui citiamo i più noti:

- ▶ **BlueJ**: per uso didattico, freeware;
- ▶ **Eclipse**: per uso professionale, freeware;
- ▶ **netBeans**: fornito insieme alle librerie di JDK, freeware;
- ▶ **JBuilder**: fornito dalla Borland, a pagamento.



◀ **IDE (Integrated Development Environment)**. Indica un software che consente di sviluppare programmi mediante un ambiente amichevole che aiuta il programmatore consentendo a volte l'immissione del codice in modo più agevole. ▶

L'ambiente di sviluppo può essere **semplice** o **integrato**. L'ambiente integrato offre tutte le funzionalità di sviluppo all'interno di una unica applicazione, con il vantaggio di fornire passaggi più agevoli alle varie fasi; in alternativa possiamo usare strumenti singoli per eseguire le varie fasi (editing, compilazione ecc.).

JSESDK

Java Standard Edition Software Development Kit è un ambiente di sviluppo per la programmazione in Java realizzato dalla [Sun Microsystems](#) per diverse piattaforme. Fornisce un certo numero di funzionalità sotto forma di comandi da eseguire in una shell dei comandi. Inoltre comprende i seguenti strumenti di programmazione:

- ▶ un compilatore Java – `javac`;
- ▶ una macchina virtuale Java – `java`;
- ▶ alcune librerie **API** (Application Programming Interface) di Java;
- ▶ un visualizzatore di Applet – `appletviewer`;
- ▶ un debugger – `jdb`;
- ▶ e un generatore automatico di documentazione – `javadoc`.

Le attività che dobbiamo svolgere mediante JDK sono le seguenti:

- ▶ **editing** (mediante l'uso di un editor oppure direttamente nell'ambiente di sviluppo prescelto);
- ▶ **compilazione** (mediante l'uso del **compilatore** Java presente nel JSESDK – **«comando javac»** da linea di comando oppure mediante un comando presente nell'ambiente di sviluppo);
- ▶ **esecuzione** (mediante l'uso di una macchina virtuale Java presente nel JSESDK – comando `java` da linea di comando oppure mediante un comando presente nell'ambiente di sviluppo).



◀ **Comando javac** Se questo comando viene eseguito senza parametri, si ottiene una schermata che ne riassume l'uso:

```
C:\>javac
Usage: javac <options> <source files>
where possible options include:
  -g               Generate all debug info
  -g:none          Generate no debug info
  -g:<lines,vars,source> Generate only some debug info
  -nowarn          Generate no warnings
  -verbose         Output messages about each source file
  -deprecation     Output source location estimates
```

■ Installazione di JSESDK per Windows

Per prima cosa dobbiamo scaricare l'ultima versione della SDK per Java dal sito della Oracle (www.oracle.com) www.oracle.com/technetwork/java/javase/downloads. La procedura che segue illustra come scaricare, installare e mandare in esecuzione la SDK.

- 1 Una volta aperto il sito all'indirizzo indicato sopra, dobbiamo focalizzare la nostra attenzione sulla sezione che ci interessa, cioè quella relativa al **download**. In questa sezione dobbiamo selezionare il pulsante che indica la piattaforma Java (**Java Platform (JDK)**), così infatti si chiama l'intero kit di sviluppo per Java: ▶
- 2 A questo punto, dopo aver selezionato **Accept License Agreement** per indicare di accettare i termini della licenza d'uso, dobbiamo selezionare la versione di JDK che intendiamo scaricare, in accordo con il sistema in uso. In questo caso decidiamo di scaricare la versione a 64 bit per Windows: ▶



- 3 Il file che otteniamo è di questo tipo (dipenderà ovviamente dalla versione, quella della figura a fianco è valida al momento della scrittura del testo ma varierà nel tempo con versioni più aggiornate): ►



- 4 Dopo aver fatto doppio click sul file appare la seguente finestra di installazione nella quale si deve selezionare **Next** per iniziare la fase di installazione: ►



- 5 Adesso dobbiamo selezionare i componenti da installare, tuttavia è consigliabile installare tutti quelli proposti. Mediante il pulsante **Change** si può modificare il percorso di installazione. Facendo click su **Next** proseguiamo con l'installazione: ►



- 6 A questo punto viene eseguita l'installazione di tutti i componenti necessari mediante una barra di progressione: ►



- 7 Una volta completata la procedura viene chiesto di fare click su **Continue** per uscire dall'installazione: ►

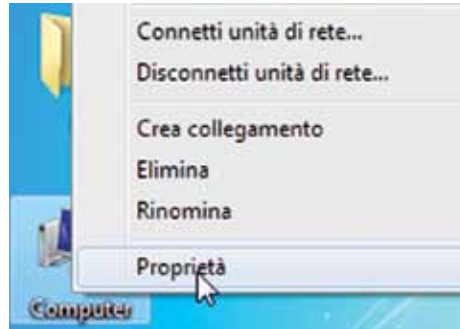


Adesso la JVM è in esecuzione e tutti i componenti per la compilazione sono stati installati.

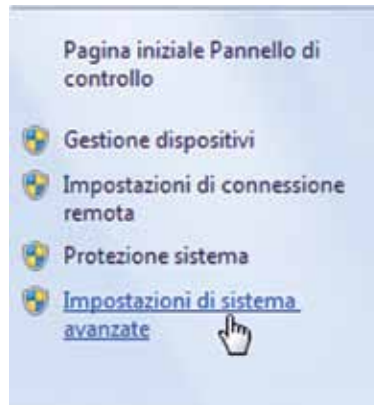
■ Il settaggio delle variabili d'ambiente

Affinchè la JDK possa funzionare è necessario settare le **variabili d'ambiente** del sistema operativo. In questo caso avendo deciso di effettuare una installazione per Windows a 64 bit procediamo come segue.

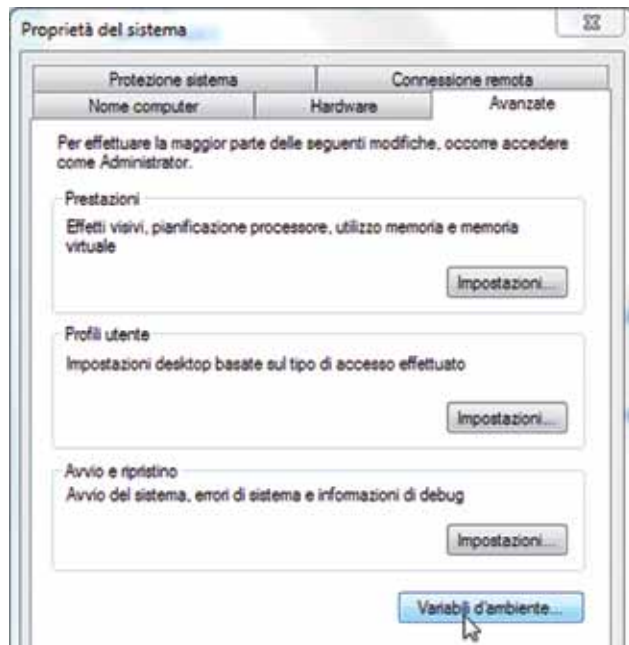
- 1 Facciamo click con il tasto destro del mouse sull'icona **Computer** e selezioniamo la voce **Proprietà**: ►



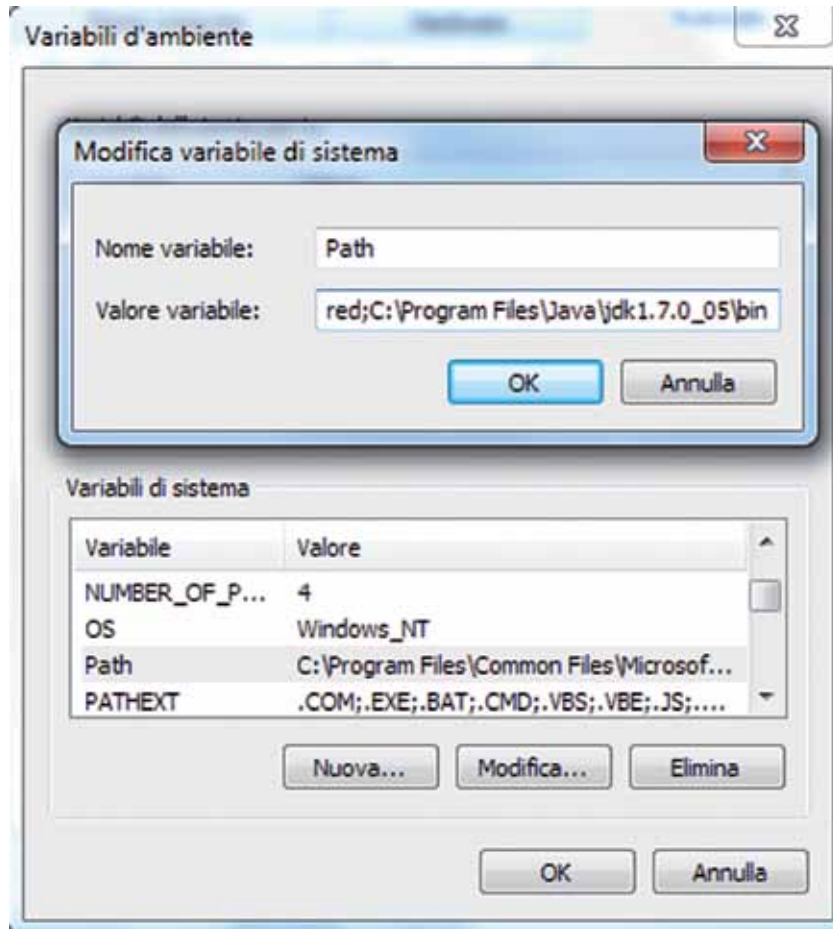
- 2 A questo punto attiviamo la voce **Impostazioni di sistema avanzate**: ►



- 3 Appare una finestra nella quale dobbiamo selezionare la scheda **Avanzate** e quindi fare click sul pulsante **Variabili d'ambiente**: ►



- 4 A questo punto dobbiamo selezionare la **variabile d'ambiente** chiamata **Path** che contiene i percorsi che il sistema è in grado di riconoscere da solo. Il percorso della JDK è importante per poter utilizzare il **compilatore** e la **macchina virtuale** per l'esecuzione delle nostre applicazioni. Per fare questo dobbiamo selezionare la riga della variabile d'ambiente **Path** e aggiungere alla fine della riga chiamata **valore variabile** il percorso della JDK installata sul computer. Nel nostro caso esso è: **C:\Program Files\Java\jdk1.7.0_05\bin** ▼



- 5 A questo punto possiamo provare il nostro primo programma. Per fare questo editiamo il codice seguente all'interno del file **ScriviSulVideo.java**, usando un editor qualunque: ▼

```
//Classe Java che visualizza un testo sul video

public class ScriviSulVideo {
    public static void main(String[] args) {
        System.out.println("salve a tutti");
        System.out.println("adesso scriviamo ");
        System.out.println("una semplice applicazione Java");
    }
}
```

- 6 Salviamo il file in una cartella qualunque, in questo caso di nome **prova** direttamente nella **root** del disco fisso **C:** ►



ScriviSulVideo.java

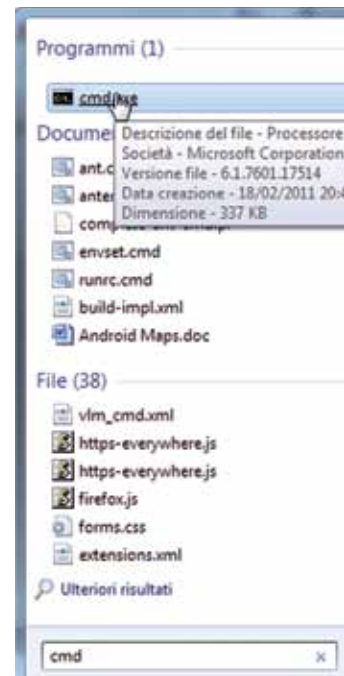
- 7 Adesso per verificare l'avvenuta installazione della JDK dobbiamo provare il compilatore usando l'ambiente a **linea di comando**. Per fare questo attiviamo l'ambiente prompt dei comandi attraverso il comando **cmd.exe**: ►

- 8 Ora ci posizioniamo nella cartella che contiene il file. La cartella **prova** è stata creata proprio sotto alla **root** del disco fisso per agevolare gli spostamenti nell'ambiente a linea di comando. Digitiamo il comando **cd \prova** seguito dal tasto **Invio** per posizionarci all'interno della nostra directory che contiene il programma da compilare: ▼

```
C:\Windows\system32\cmd.exe
C:\>cd prova
C:\prova>
```

- 9 Adesso possiamo compilare il codice sorgente rappresentato dal file **ScriviSulVideo.java** attraverso il comando **javac** seguito dal nome del file da compilare: ▼

```
C:\Windows\system32\cmd.exe
C:\>cd prova
C:\prova>javac ScriviSulVideo.java
```



Il risultato della compilazione viene memorizzato in un file avente il medesimo nome di quello della classe contenuta nel file java, ma con estensione **.class**. Si trova quindi nella stessa directory di lavoro utilizzata nel comando precedente, un file chiamato **ScriviSulVideo.class** che contiene il bytecode della classe compilata. Il bytecode non è direttamente eseguibile dalla macchina, ma viene interpretato dall'ambiente di esecuzione, il **Java Runtime Environment** (JRE).



◀ **Java Runtime Environment** Si tratta del software che consente l'esecuzione di programmi compilati per la piattaforma Java. All'interno di questo componente troviamo la Virtual Machine per la piattaforma Java che si occupa di eseguire il bytecode Java risultato della compilazione, le librerie di base che contengono le funzionalità di base della piattaforma, come per esempio la classe String e i file di supporto come per esempio i messaggi localizzati nelle diverse lingue supportate da Java, le icone e altro. ►

- 10 Se non ci sono errori il sistema non segnala nulla. Per verificare la creazione del bytecode possiamo digitare il comando **dir** che mostra tutti i file presenti nella directory. Come si può notare appare anche un nuovo file chiamato **ScriviSulVideo.class** che rappresenta il codice bytecode: ▼

```
C:\prova>javac scrivisulvideo.java
C:\prova>dir
Il volume nell'unità C è OS
Numero di serie del volume: 7270-8F2A

Directory di C:\prova
17/07/2012  22:31    <DIR>          .
17/07/2012  22:31    <DIR>          ..
17/07/2012  22:31                518 ScriviSulVideo.class
16/07/2012  21:58                274 ScriviSulVideo.java
                2 File             792 byte
                2 Directory 157.873.946.624 byte disponibili
C:\prova>
```

- 11 Adesso possiamo **eseguire** il bytecode attraverso una fase di interpretazione dello stesso. Per fare questo digitiamo il comando **java** seguito dal nome del file, in questo caso senza l'estensione: ►

```
C:\prova>java ScriviSulVideo
salve a tutti
adesso scriviamo
una semplice applicazione Java
C:\prova>
```

Come possiamo notare il file viene eseguito e appare sullo schermo il messaggio contenuto nel codice. Nell'esempio a fianco si nota che il file viene compilato nella cartella prova, quindi il controllo passa a un'altra directory che non contiene più il file **.class**. In questo caso siamo obbligati a specificare il percorso che contiene la classe da eseguire mediante l'opzione **-cp** (classpath) che indica il percorso dove trovare i file delle classi. Se la directory dove sono contenute le classi è diversa da quella attuale, va specificato un percorso completo assoluto o relativo. ►

```
C:\prova>javac CiaoMondo.java
C:\prova>cd..
C:\>java -cp prova CiaoMondo
Ciao Mondo!
```

Non siamo obbligati a indicare l'estensione del file (**.class**).

Come abbiamo visto per eseguire un programma Java non è possibile digitarne semplicemente il nome da linea di comando; per eseguire **bytecode** Java è infatti necessario lanciare l'ambiente di runtime, specificando il nome della classe da cui partire per l'esecuzione (e in cui deve essere definito il metodo **main()**).

■ Installare l'ambiente di sviluppo BlueJ

BlueJ è un ambiente di sviluppo gratuito, a carattere didattico, ed è stato progettato proprio per imparare a programmare con Java. Per poter funzionare dobbiamo prima di tutto aver installato la JDK, come abbiamo visto sopra, in quanto BlueJ è un ambiente che mette a disposizione alcuni strumenti grafici ma non contiene il compilatore. Lo scopo principale di BlueJ è quello di fornire al programmatore un'interfaccia semplice per la programmazione.

Il programma è disponibile per vari sistemi operativi:

- Windows
- MAC OS
- Linux

Nel nostro caso installeremo la versione di BlueJ per Windows a 64 bit.

La procedura che segue illustra i vari passaggi necessari all'installazione di BlueJ in ambiente Windows.

Ricordati sempre di installare la JDK prima di iniziare l'installazione di BlueJ!

- 1 Collegati al sito www.bluej.org e cerca la sezione **download**: ►



- 2 Una volta entrato nella sezione **download** devi selezionare la versione di BlueJ adatta al tuo sistema operativo. In questo caso scaricheremo la versione per Windows: ▼

Step 1: Get Java

Check the system requirements (right side of this page). Make sure you have an appropriate version of Java:

Step 2: Download the installation package

BlueJ version 3.0.7

for Windows 2000, XP, Vista or newer (6.8 Mb)	bluej-307.msi
for MacOS X (5.7 Mb)	BlueJ-307.zip
for Debian, Ubuntu and other Debian-based systems (5.5 Mb)	bluej-307.deb
all other systems (executable jar file) 5.7 Mb)	bluej-307.jar

- 3 Dopo aver fatto click sul collegamento ipertestuale prescelto viene scaricato il file. ►

Il nome del file di installazione è **bluej-xxx.msi**, dove **xxx** rappresenta il numero della versione del programma. In questo caso la versione è la **3.07**, tuttavia le versioni cambiano molto spesso, quindi scarica sempre la versione più aggiornata.



- 4 Adesso fai doppio click sul file per iniziare l'installazione, apparirà la seguente finestra in cui selezionare **Next**: ►



- 5 La seguente finestra indica che è stata trovata una versione della JDK. In questa videata potrebbe verificarsi un errore qualora la JDK non venisse localizzata. Fai click su **Next** per proseguire. ►

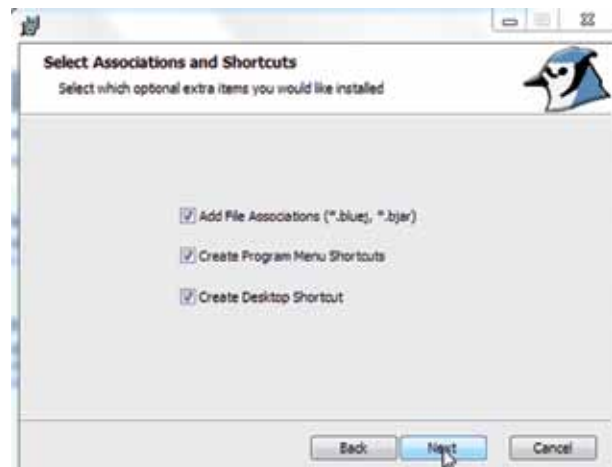


- 6 La finestra che segue chiede se l'installazione deve essere effettuata per l'utente attivo in quel momento oppure per tutti gli utenti. In questo caso decidiamo di installare il programma per tutti gli utenti selezionando **Install for all users of this machine**, quindi fai click su **Next** per proseguire come di consueto: ►

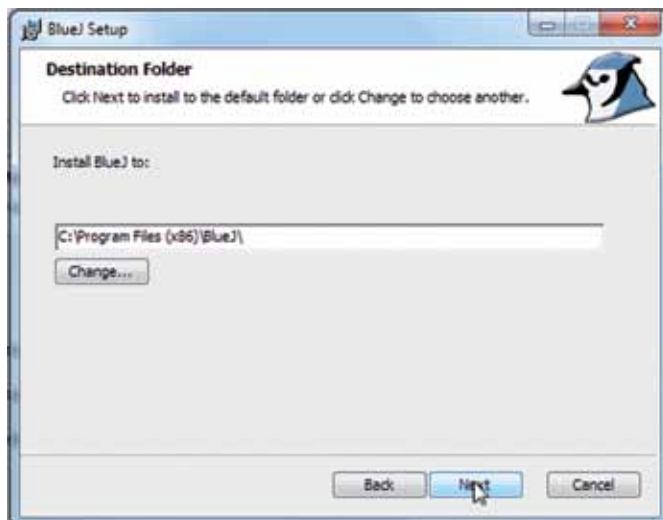


In alcuni sistemi operativi meno recenti è necessario effettuare una ricerca per localizzare la directory della JDK.

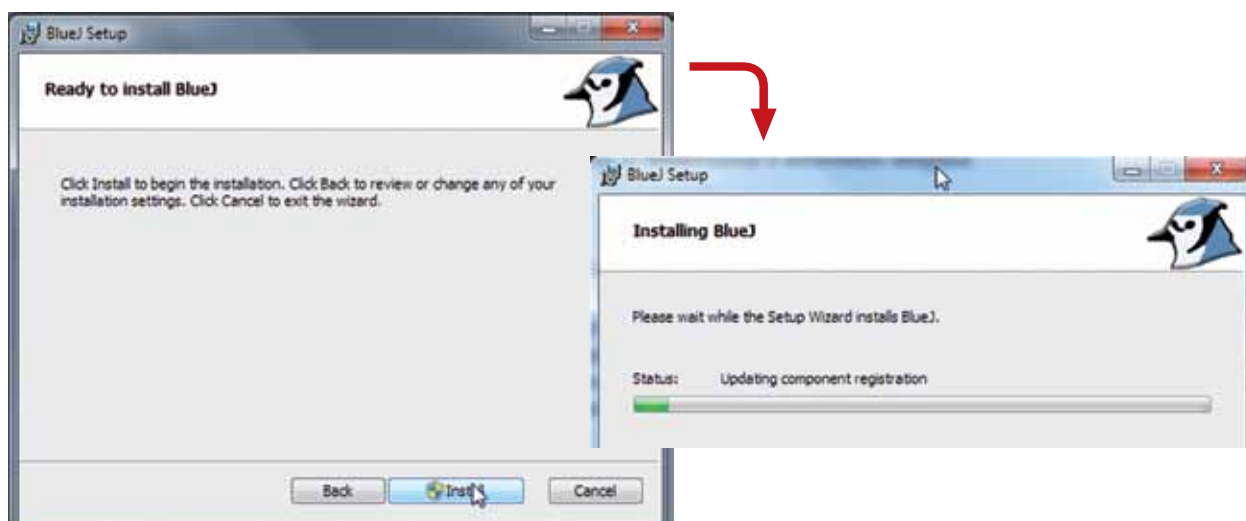
- 7 La finestra che segue ci mostra semplicemente se associare un collegamento a BlueJ anche sul desktop e se associare i file con estensione **.bluej** e **.bjar** a questo programma. Il consiglio è di lasciare le caselle di testo spuntate come indicato e quindi premere **Next**: ►



- 8 In questa finestra viene chiesto in quale directory installare il programma. Anche in questo caso il consiglio è quello di lasciare inalterata la cartella indicata. Fai click sul pulsante **Next** per proseguire: ►

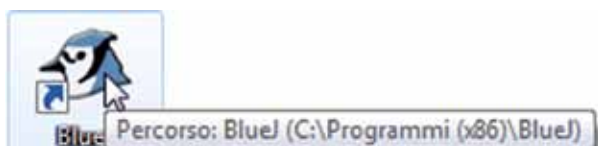


- 9 Facendo click sul pulsante **Install** inizierà l'installazione che copierà i file sul disco fisso: ▼



- 10 Fai click infine sul pulsante **Finish** per terminare l'operazione. ►

- 11 Sul desktop apparirà l'icona seguente che consente di aprire il programma BlueJ: ▼



■ Utilizzare l'ambiente di sviluppo BlueJ

Ora analizzeremo come scrivere e compilare un programma in Java mediante BlueJ, anche se i concetti base di programmazione verranno introdotti in seguito.

La seguente procedura illustra come creare un nuovo ◀ **progetto BlueJ** ▶.



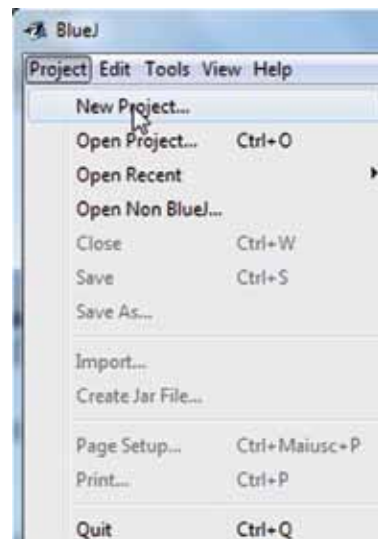
◀ **Progetto BlueJ** I progetti BlueJ sono molto simili ai **package** standard di Java, e non sono altro che semplici **cartelle** contenenti i file inclusi nel progetto. Pertanto a ogni progetto viene associata una directory. ▶

Prima di tutto è necessario affermare che per scrivere un programma in Java mediante BlueJ dobbiamo necessariamente includerlo all'interno di un progetto BlueJ, anche se si tratta solo di una classe.

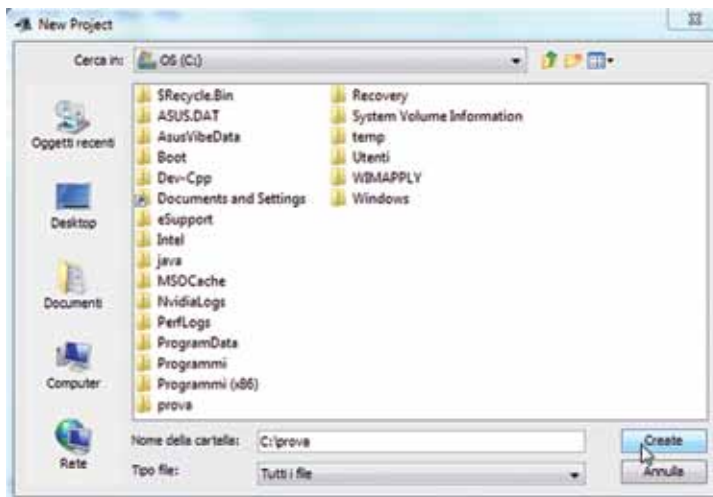
- 1 Fai doppio click sull'icona di **BlueJ** per aprire l'ambiente di sviluppo: appare la finestra seguente: ▶



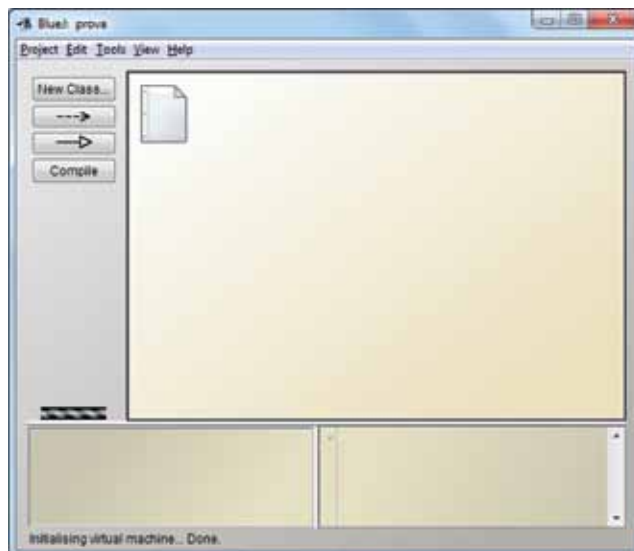
- 2 Per creare un nuovo progetto fai click sul menu **Project**, quindi sulla voce **New Project...**: ▶



- 3 A questo punto viene richiesto di **salvare il progetto**. In realtà dobbiamo solo decidere il nome e il percorso della directory che rappresenta il nostro progetto BlueJ. In questo caso si chiamerà **prova** e sarà posizionata nella **root** del disco fisso **C:**. ▼



- 4 A questo punto appare un foglio stilizzato che rappresenta il progetto, esso contiene solo informazioni che ne documentano il contenuto, è il programmatore che deve compilarlo. ►

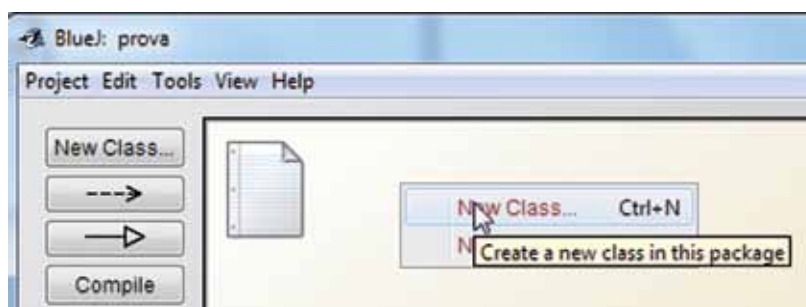


Se vuoi aggiungere informazioni riguardo al progetto fai doppio click su di esso e scrivi per esempio il titolo, lo scopo, la versione, come eseguirlo, gli autori, la sintesi del codice, i dati in ingresso e in uscita ecc.

```
-----
This is the project README file. Here, you should describe your project.
Tell the reader (someone who does not know anything about this project)
all he/she needs to know. The comments should usually include at least:
-----

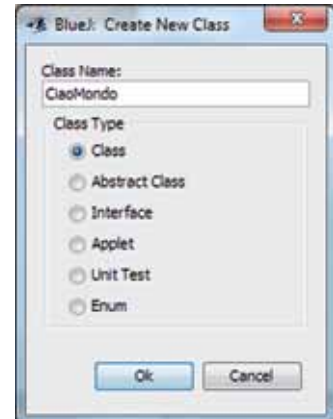
PROJECT TITLE:
PURPOSE OF PROJECT:
VERSION or DATE:
HOW TO START THIS PROJECT:
AUTHORS:
USER INSTRUCTIONS:
```

- 5 Possiamo adesso all'**editing**, cioè alla scrittura del codice Java del programma. Per creare un programma dobbiamo necessariamente creare una nuova **classe**. Per creare una classe fai click con il tasto destro all'interno del riquadro principale della finestra di BlueJ e seleziona la voce **New Class...** dal menu contestuale che appare: ▼



- 6 La finestra a fianco chiede il nome della classe da creare. In questo caso la chiameremo **CiaoMondo**: ►

Puoi quindi scegliere tra quattro tipi di classi: classe standard (**Class**), astratta (**Abstract**), interfaccia (**Interface**) oppure **Applet**. Questa scelta determina quale struttura (**skeleton**) verrà inizialmente creata per la tua classe. Puoi cambiare il tipo di classe successivamente, modificando il codice sorgente.



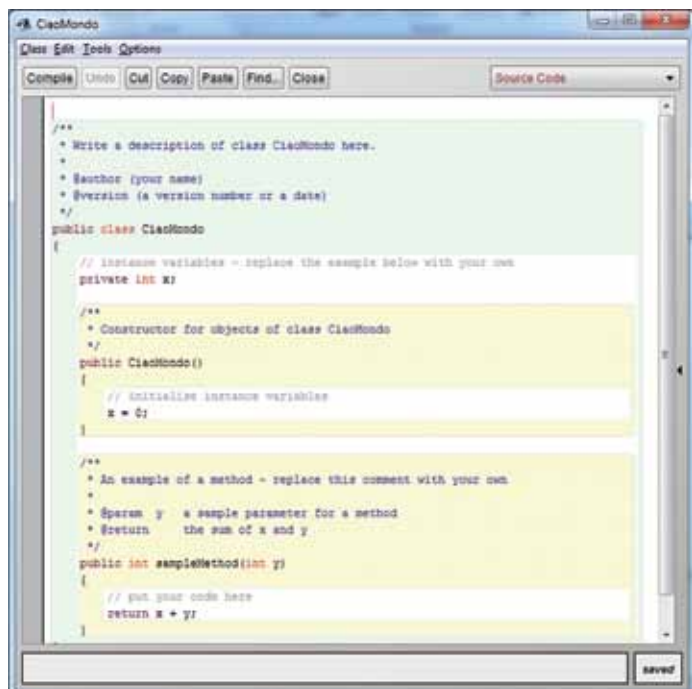
- 7 Dopo aver fatto click su **OK** appare una nuova classe, rappresentata da un'icona nel diagramma. Se non è una classe standard, il tipo (**interface**, **abstract** o **applet**) è indicato nell'icona della classe. ►



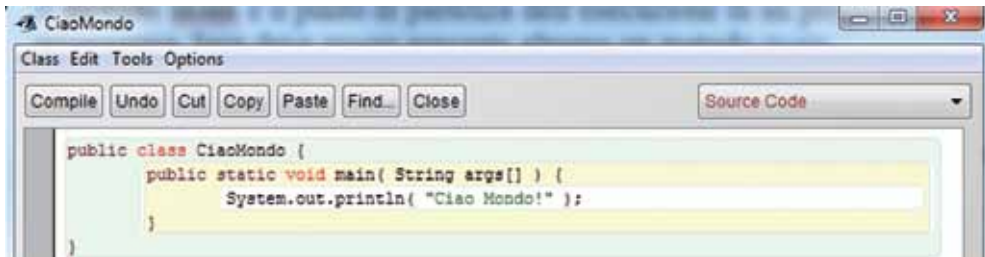
- 8 A questo punto passiamo alla scrittura del codice contenuto nella classe. Per fare questo fai click con il tasto destro del mouse e seleziona **Open Editor**: ►



- 9 Come puoi notare si apre la finestra dell'editor che reca il listato di un programma di esempio: ►

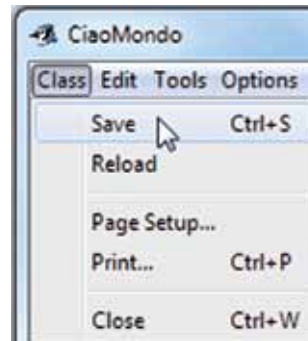


- 10 Adesso devi cancellare questo codice di esempio per scrivere al suo posto il nostro programma di esempio che mostra un messaggio in output: ▼

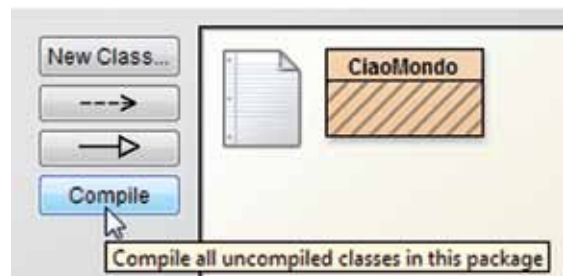


Osserva che quando il cursore si trova subito a destra di una parentesi graffa chiusa BlueJ evidenzia in grigio qual è la parentesi graffa aperta corrispondente. Questa funzione è molto utile perché uno degli errori di stesura più frequenti riguarda le parentesi graffe.

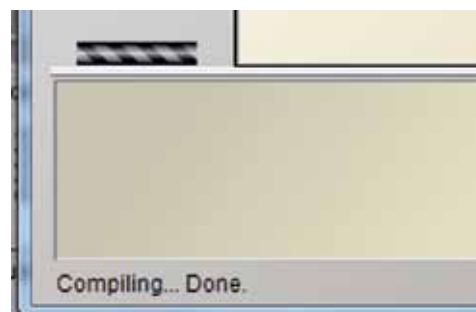
- 11 Dopo aver scritto il codice apri il menu **Class** e seleziona la voce **Save** per salvare il codice sorgente: ►



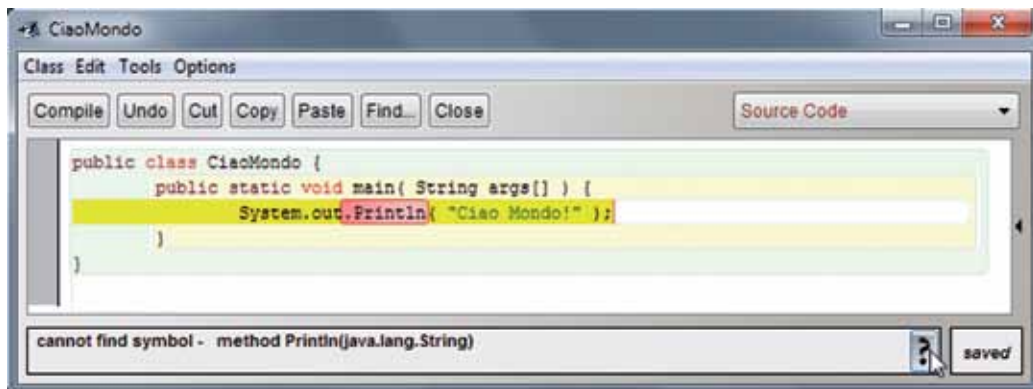
- 12 Adesso fai click su **Close** posto in alto a destra, in tal modo ritornerai al menù principale. Abbiamo così terminato la stesura del programma e siamo pronti per compilarlo, per fare questo fai click sul pulsante **Compile**: ►



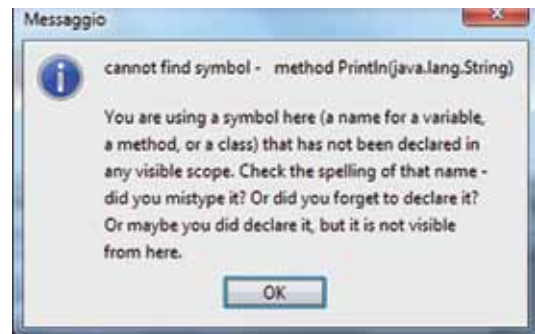
- 13 Se non ci sono errori BlueJ mostrerà nella barra di stato il messaggio **Compiling... Done**: ►



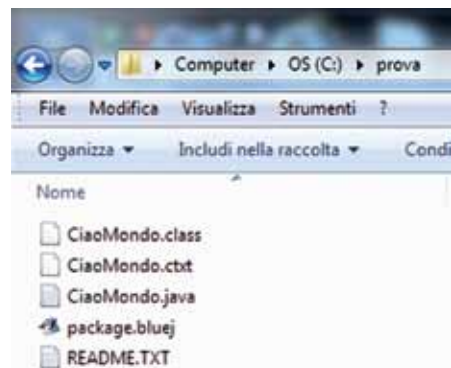
Quando il compilatore trova degli errori all'interno del codice, si apre automaticamente la finestra dell'editor evidenziando sia la riga in cui si è verificato l'errore che il relativo messaggio di spiegazione. ▼



Se clicchiamo sul punto interrogativo posto a lato del messaggio, ci viene mostrata la spiegazione del messaggio d'errore prodotto dal compilatore. ►



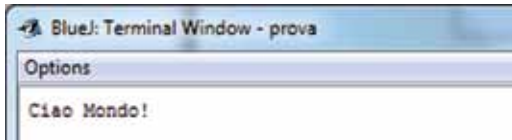
- 14 Quando un file viene compilato viene generato un file con estensione `.class` che ne rappresenta il **bytecode**. Posizionati nella cartella **prova** che contiene il progetto per verificare se il file è stato effettivamente creato: ►



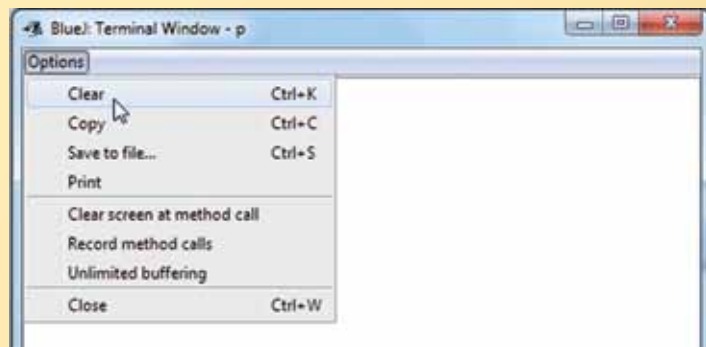
- 15 Adesso possiamo testare il nostro programma. Posizionati col puntatore del mouse sopra alla classe dove si trova il metodo `main`, quindi sopra alla classe **CiaoMondo**, cliccando col tasto destro apparirà un menu nel quale devi selezionare la voce `void main(String[] args)`: ►



- 16 Appare la finestra seguente nella quale potresti inserire dei parametri da passare al metodo `main`. In questo caso basterà fare click su **OK** per proseguire: ►
- 17 Come puoi notare si apre la finestra di **output** del programma nella quale appare il testo visualizzato: ▼



La finestra di output di BlueJ purtroppo non cancella il contenuto a ogni esecuzione. Per cancellare lo schermo dobbiamo agire sul menu **Options**, quindi selezionare la voce **Clear**:



■ Il debugging con BlueJ

Il programmatore, durante l'attività di scrittura di un programma deve tener conto dei possibili errori che si producono. Tali errori possono essere raggruppati in tre categorie:

- **errori di compilazione** causati dall'utilizzo di parole che non appartengono al linguaggio oppure dalla costruzione non corretta di istruzioni del codice;
- **errori in fase di esecuzione**, chiamati anche errori di **run time**, segnalati durante l'esecuzione del programma;
- **errori logici** che generano risultati diversi da quelli attesi.

Per evitare di commettere errori, il programma deve essere progettato tenendo conto di tutti i possibili valori che l'utente potrà immettere durante l'esecuzione.

L'ambiente **BlueJ** mette a disposizione uno strumento che consente di individuare i diversi tipi di errore e di apportare al codice le opportune correzioni, chiamato ◀ **debugger** ▶. L'attività di individuazione e correzione degli errori del codice sorgente viene comunemente chiamata fase di **debugging**.

Le principali funzionalità del debugger presente in questo ambiente di sviluppo sono:

- impostare i **breakpoints**;
- eseguire il codice istruzione dopo istruzione (**step by step**);
- ispezionare il **contenuto** delle variabili.



◀ **Debugger** Questo termine ha una origine molto lontana nel tempo, venivano infatti chiamati così i lavoratori che avevano il compito di ripulire le valvole dei primi computer dai nidi di alcuni tipi di coccinelle (dall'inglese **bug** che significa appunto coccinella). Più recentemente il termine ha preso via via un significato diverso, attualmente indica un software che ripulisce il programma dagli errori. ►

Impostare i breakpoints

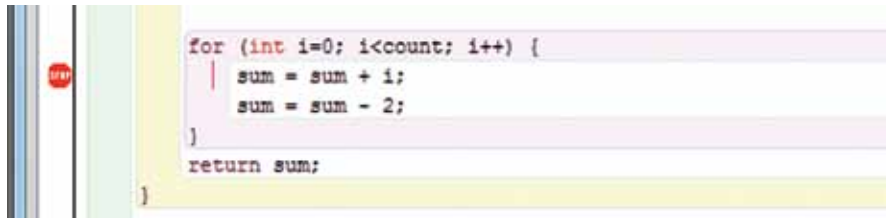
Per fissare i ◀ **punti di interruzione** ▶ si deve procedere facendo click nella colonna alla sinistra del codice sulla riga interessata.



◀ **Punti di interruzione** I punti di interruzione (in inglese **breakpoints**) servono per far eseguire il programma fino a un punto prefissato utile per verificare quale valore assumono alcune variabili in quel particolare istante. Il programma si interrompe prima della riga che contiene il punto di arresto. ▶

Per provare le operazioni di debug utilizziamo un progetto già creato; lo scopo è quello di verificare il funzionamento delle funzioni di debug.

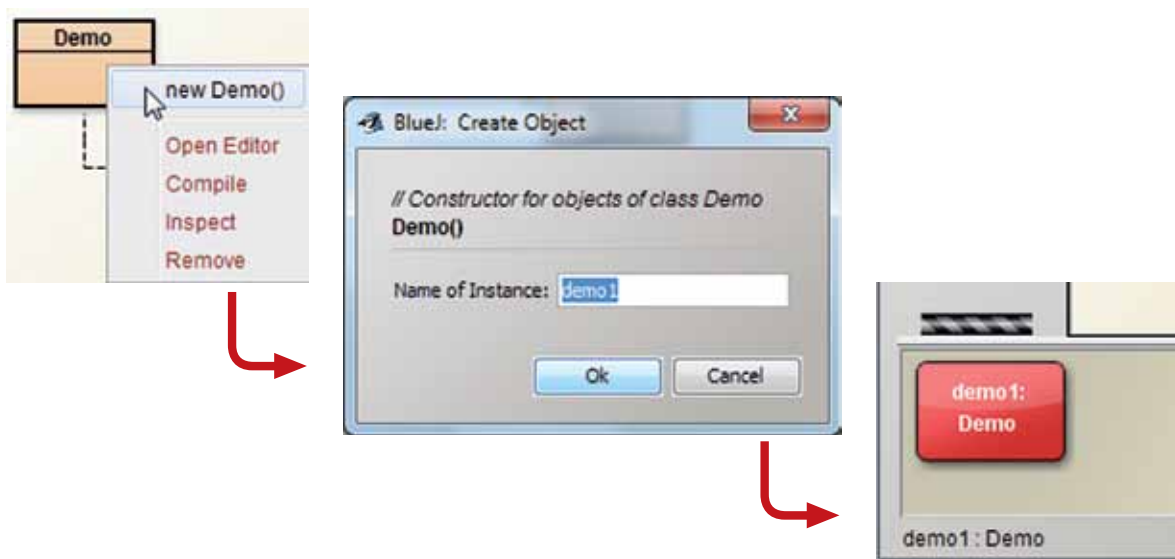
Prima di tutto devi **compilare** le due classi presenti: **Demo** e **Car**. A questo punto apri l'editor sulla classe **Demo**, cerca il metodo **loop** e imposta il **breakpoint** in qualsiasi punto del ciclo **for**. Apparirà un simbolo di stop accanto alla riga in cui è presente il breakpoint, in questo caso la riga **sum = sum + i;**:



Quando la riga del codice, che ha il breakpoint impostato, viene raggiunta, l'esecuzione del programma si interrompe.

Per provare quanto indicato esegui la procedura che segue.

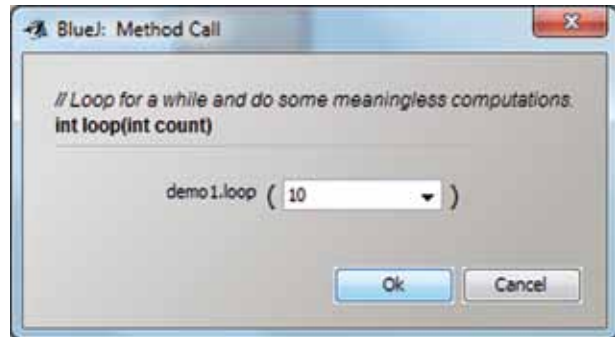
- 1 Apri il progetto **Esempio1**.
- 2 Crea un oggetto di classe **Demo** facendo click con il tasto destro del mouse sulla classe e selezionando la voce **new Demo()**. Quindi conferma con **OK**. A questo punto viene creato un oggetto di classe Demo chiamato **demo_1**: ▼



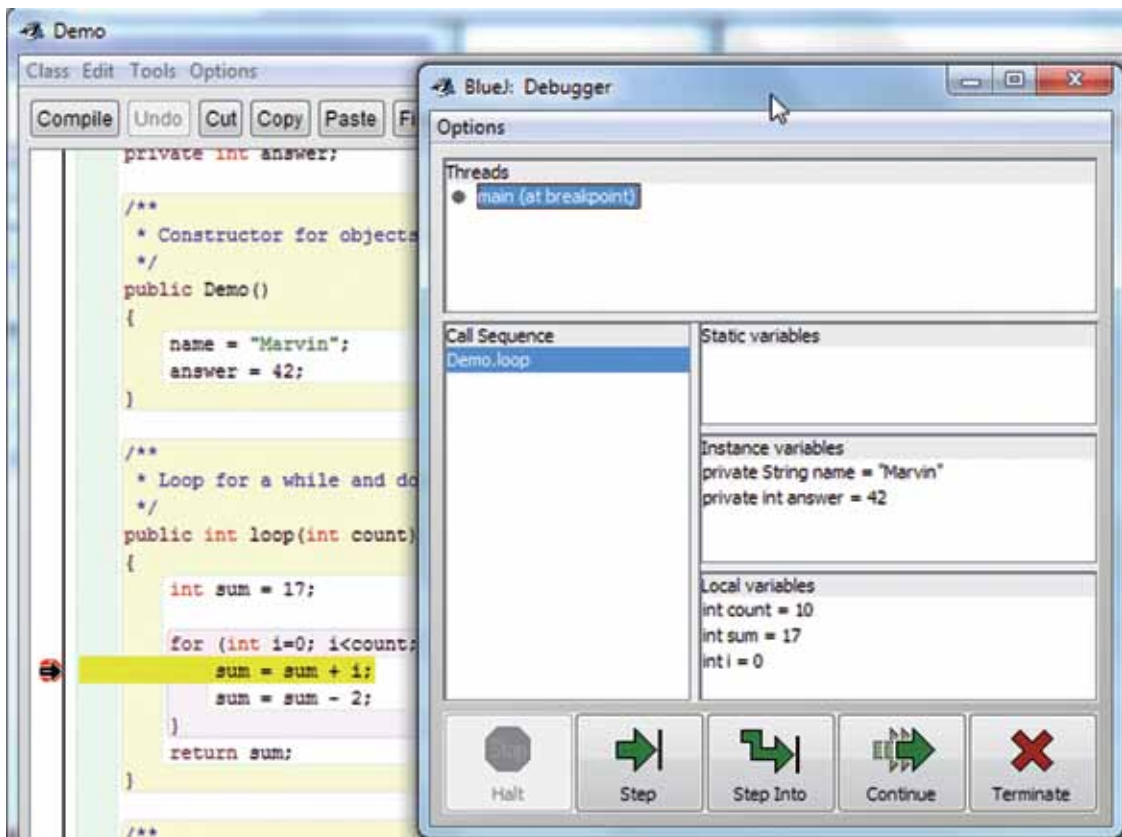
- 3 Adesso chiama il metodo `loop` passando ad esso un **parametro**, per esempio 10. Per fare questo fai click con il tasto destro sopra all'oggetto `demo_1` e seleziona la voce `int loop(int count)`: ➤



- 4 Adesso inserisci il parametro (in questo caso 10): ➤



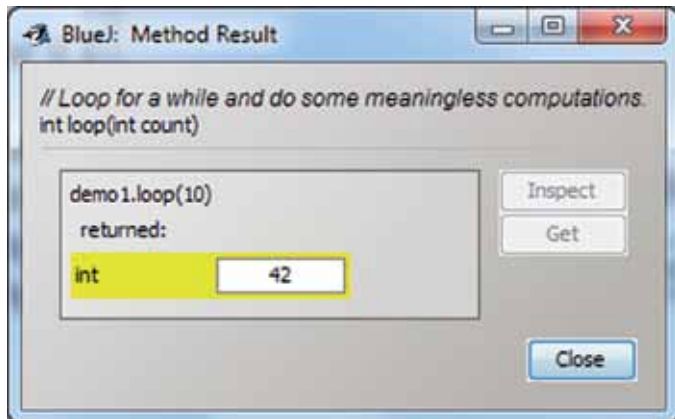
- 5 In questo modo il programma viene eseguito e terminerà esattamente dove indicato nel **breakpoint**. Non appena il breakpoint viene raggiunto, si apre la finestra dell'editor che visualizza la riga corrente del codice insieme alla finestra di **debugger**: ▼



L'esecuzione si ferma prima che la riga che contiene il breakpoint venga eseguita.

- 6 Adesso che l'esecuzione è interrotta momentaneamente possiamo usare il tasto **Step** per eseguire una istruzione per volta dal punto di arresto in poi. Ogni volta che clicchiamo sul pulsante **Step** viene eseguita una singola riga del codice e l'esecuzione interrotta. Il contenuto delle variabili in uso nel programma vengono mostrati nella finestra del debugger chiamata **Local variables**. Continua a fare click sul pulsante **Step** e nota il contenuto delle variabili fino al termine del metodo **loop**. A quel punto appare una **finestra** che mostra il valore restituito dal metodo: ►

Il pulsante **Continue** nella finestra di debugger serve per riavviare l'esecuzione e far eseguire normalmente il programma.



L'esecuzione passo passo

Per eseguire il programma passo passo (**step by step**) dobbiamo utilizzare i pulsanti **Step** e **Step Into** presenti nella finestra di debugger.



Step e **Step Into** si comportano allo stesso modo se la riga di codice non contiene una chiamata a un metodo.

Quando l'esecuzione del programma raggiunge il breakpoint ed è pertanto in fase di Stop, possiamo eseguire una istruzione per volta per verificare il corretto svolgimento del programma in esecuzione. Per fare questo dobbiamo ripetere più volte il click sul pulsante **Step**. L'evidenziatore di riga del codice sorgente in corso di esecuzione si sposterà via via secondo l'ordine prefissato dal programma. A ogni click su **Step** ogni singola riga del codice viene eseguita e l'esecuzione viene fermata di nuovo. I valori che le variabili assumono variano in tempo reale. Verifica per esempio il valore della variabile **sum**.

L'ispezione del contenuto delle variabili

Quando utilizziamo il debugger risulta assai importante, per la verifica del funzionamento del codice, verificare il contenuto delle variabili durante l'esecuzione del codice. La finestra di debugger suddivide le variabili in due categorie:

- **variabili locali**;
- ◀ **variabili di istanza** ►.

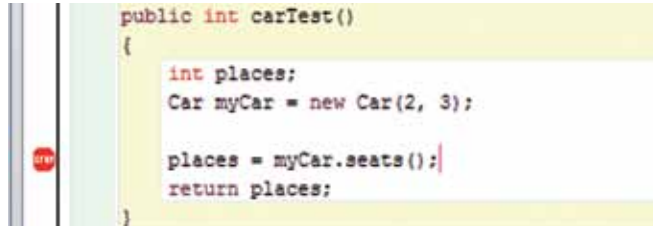


◀ **Variabili di istanza** Le variabili d'istanza vengono anche chiamate attributi o proprietà dell'oggetto corrente. ►

Le variabili locali sono le variabili utilizzate all'interno del metodo corrente e ne viene sempre visualizzato il contenuto aggiornato.

È possibile anche selezionare i metodi nella sequenza delle chiamate per esaminare le variabili di altri oggetti e metodi ancora attivi. La seguente procedura illustra come.

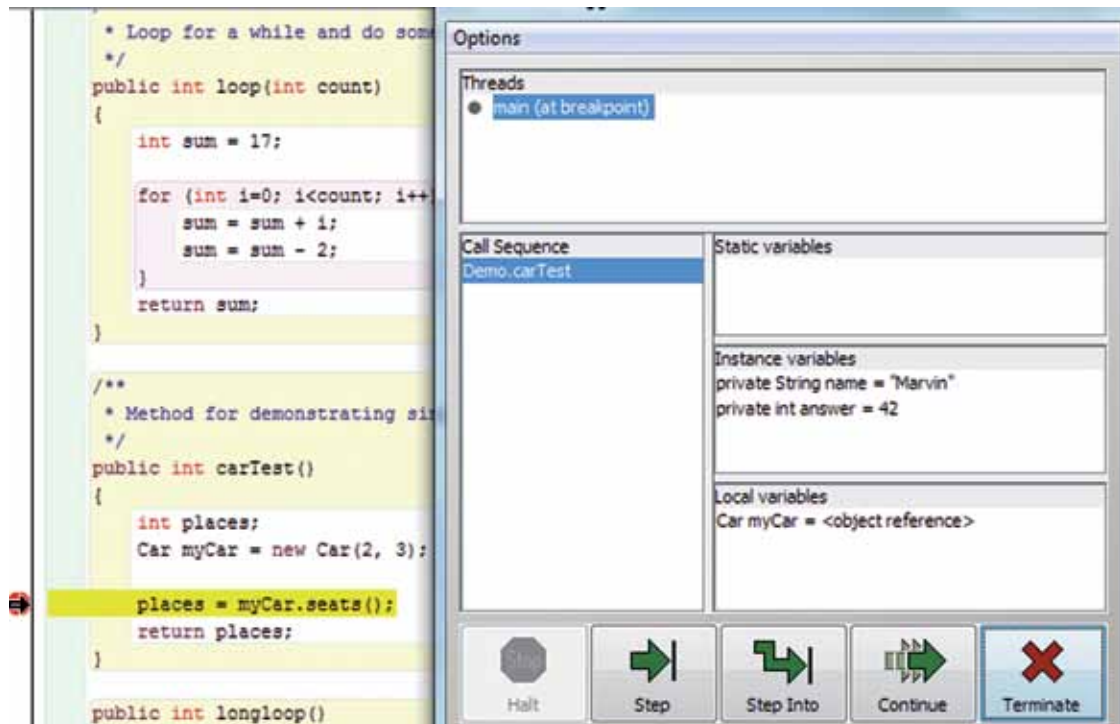
- 1 Apri il progetto [Esempio1](#).
- 2 Aggiungi un breakpoint all'interno del metodo `carTest()` ►



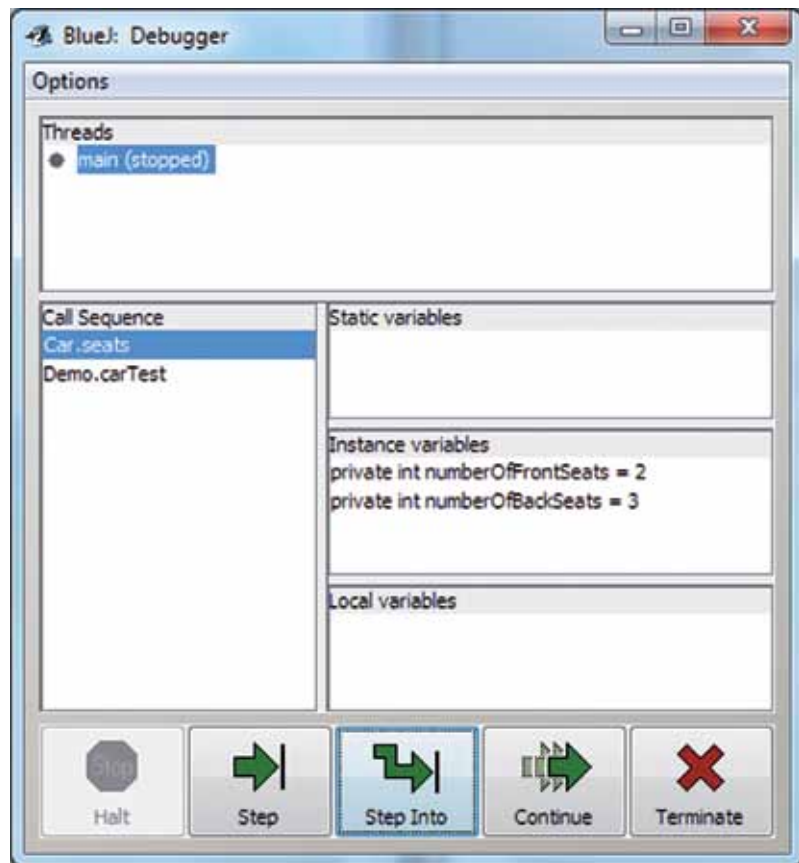
- 3 Chiama il metodo `carTest()` dall'oggetto `demo_1`: ►



- 4 Quando il breakpoint viene raggiunto, appare la finestra di debugger che ci informa sulla situazione delle variabili. ►



- 5 Facendo click su **Step** viene eseguita la riga successiva che richiama il metodo `seats()` della classe `Car`.
- 6 Facendo invece click su **Step Into** viene indicato anche il metodo `Car.seats()`. Questo accade perché il metodo `Car.seats` è stato chiamato da `Demo.carTest`. Se selezioni `Demo.carTest` dalla lista chiamata **Call sequence** puoi ispezionare il codice sorgente e i valori delle variabili correnti di questo metodo. ►
- 7 Procedendo alla riga successiva, che contiene l'istruzione `new Car(...)`, puoi osservare che il valore della variabile locale `myCar` è visualizzata come `<object reference>`.



Tutti i valori di tipo oggetto, a eccezione delle stringhe (tipo `String`) vengono visualizzati in questo modo. Per ispezionare il contenuto di questa variabile fai doppio click su di essa, si aprirà una finestra di ispezione dell'oggetto uguale a quella descritta in precedenza.



Prova adesso!



APRI IL PROGETTO Esemplio1

- 1 Riprova ancora con un altro metodo. Imposta un breakpoint nella classe `Demo`, nel metodo `carTest()`, quindi chiama il metodo.
- 2 Crea un oggetto di classe `Car` con 2 posti davanti e 3 posti dietro.
- 3 Aggiungi un secondo breakpoint al metodo `seats()` della classe `Car`.
- 4 Chiama il metodo `seats()` e verifica il contenuto delle variabili locali e attributi.

- Uso del Debugger
- Creazione breakpoints
- Ispezione contenuto variabili
- Esecuzione passo passo

Verifichiamo le conoscenze

>> Esercizi a scelta multipla

1 Metti in ordine logico le fasi di sviluppo di un programma in Java scritte a sinistra con il relativo software posto a destra:

- | | |
|-----------------------------|--------------|
| a) esecuzione | Editor |
| b) stesura del programma | JDK |
| c) compilazione in bytecode | JRE |
| d) interpretazione | |

2 Le classi in Java sono contenute:

- a) ciascuna classe in un metodo
- b) ciascuna classe in un singolo file
- c) ciascuna classe pubblica in un singolo file
- d) raggruppate in gruppi logici e memorizzate in pochi file sorgenti

3 Il compilatore a linea di comando viene eseguito con il comando:

- | | |
|---------------------------|--------------------------|
| a) >javac Nomeclasse | c) >java Nomeclasse |
| b) >javac Nomeclasse.java | d) >java Nomeclasse.java |

4 Il compilatore produce file con estensione:

- | | |
|----------|-----------|
| a) .java | c) .class |
| b) .clas | d) .exe |

5 Quale comando da linea tra i seguenti consente di eseguire un file bytecode?

- | | |
|--------------------------|--------------------------------|
| a) java -cp . Nomeclasse | d) java Nomeclasse |
| b) jre Nomeclasse.class | e) java -cp . Nomeclasse.class |
| c) jre Nomeclasse.java | |

>> Esercizi di completamento

1 Indica almeno tre ambienti di sviluppo per Java:

- a)
- b)
- c)

2 Il compilatore Java traduce il in

3 Il risultato della compilazione in Java è il

4 La sigla JRE indica

5 Il JRE è composto da due elementi principali: la che si occupa dell'esecuzione del codice mentre la contiene le funzionalità di base.