Introduzione a Java

Programmazione 2, a.a. 2003-2004

Novella Brugnolli,
email: novella.brugnolli@unitn.it

Configurazione dell'ambiente di lavoro

Configurazione della variabile di ambiente PATH
Linux, che shell state usando?
per saperlo lanciate il comando:
> echo $SHELL

shell **tcsh:** nel file .tcsrc
set path=(/usr/local/JBuilderx/bin $path)

shell **bash:** nel file .bashrc
PATH=/usr/local/JBuilderx/bin:$path
export PATH

Ambiente di sviluppo: JBuilder X

JBuilder X Foundation Edition
scaricabile gratuitamente da:
http://www.borland.com

Directory di installazione: /usr/local/JBuilderx

Per avviare il programma:
> /usr/local/JBuilderx/bin/jbuilder &

se la path è settata correttamente:
> jbuilder

KeyStrokes

F1 se posizionato sul nome di una classe ne visualizza la documentazione

Ctrl+H provides methods and members of active scope
Ctrl+J provides code template
Ctrl+Alt+Space (**Ctrl+Alt+H**) inserts a class name into a writable file

Ctrl+/ comment line
Ctrl+L select line
Ctrl+W select world
Ctrl+A select all

Trovate tutte le definizioni:

Tools -> Preferences ->Keymapping

Schema minimo di un'applicazione

```
public class Test {
    public static void main(String args[]){
        Test test1 = new Test();
    }
    Test(){
        // in questo blocco vanno scritte le istruzioni
    }
}
```

Commenti

3 tipi di commenti:

```
/*      questo è un commento
        su più righe      */

// questo è un commento su una riga sola

/**
  *  anche questo è un commento su più righe
  */
```

Tipi di dati primitivi

	dimensione	valori	esempio
boolean	-	true / false	false
char	16 bit		‘c’
byte	8 bit	[-128, +127]	3
short	16 bit	[-2^15, +2^15-1]	8
int	32 bit	[-2^31, +2^31-1]	187
long	64 bit	[-2^63, +2^63-1]	8834L
float	32 bit		87.363F
double	64 bit		26.77e3

Nota: Le variabili locali (interne ad un metodo) devono essere inizializzate prima di essere richiamate. Le variabili membro di una classe, se non esplicitamente inizializzate, vengono inizializzate a dei valori di default.

Operatori

operatori binari + - * / %

operatori unari - +

operatori di incremento e di decremento

prefisso ++a --a, postfisso a++ a--

operatori relazionali < > <= => == !=

operatori booleani && ||

Array

```
int[ ] a = new int[10];
double a[ ] = {1, 5, 23.9};
char gatto[ ] = new char[ ] {'g', 'a', 't', 't', 'o'};

int lunghezzaArray = gatto.length;

int matrice[ ][ ] = new int [3][24];
```

Definizione di un metodo

```
tipo_restituito nomeMetodo(elenco parametri){
  // elenco istruzioni
}
```

un metodo è univocamente identificato dal **nome** e dal **tipo e numero degli argomenti**

Costrutti iterativi: while e do…while

```
while (espressione condizionale) {
    // istruzioni
}
```

```
do {
    // istruzioni
} while (espressione condizionale)
```

Costrutti iterativi: while

```
public class WhileDemo {
  public static void main(String[] args) { new WhileDemo();}
  WhileDemo(){
    String copyFromMe = "Copy this string until you " +
                        "encounter the letter 'g'.";
    StringBuffer copyToMe = new StringBuffer();
    int i = 0;
    char c = copyFromMe.charAt(i);
    while (c != 'g') {
       copyToMe.append(c);
       c = copyFromMe.charAt(++i);
    }
    System.out.println(copyToMe);
  }
}
```

Costrutti iterativi: do, while

```
public class DoWhileDemo {
  public static void main(String[] args) { new DoWhileDemo(); }
  DoWhileDemo(){
    String copyFromMe = "Copy this string until you " +
                        "encounter the letter 'g'.";
    StringBuffer copyToMe = new StringBuffer();
    int i = 0; char c = copyFromMe.charAt(i);
    do {
      copyToMe.append(c);
      c = copyFromMe.charAt(++i);
    } while (c != 'g');
    System.out.println(copyToMe);
  }
}
```

Costrutti iterativi: for

```
for (inizializzazione; istruzione booleana; passo) {
      // istruzioni
}
```

ciascuna delle istruzioni: inizializzazione, istruzione booleana e passo può essere vuota

passo può essere una serie di istruzioni (devono essere separate da una virgola)

Costrutti iterativi: for

```
public class ForDemo {
  public static void main(String[] args) { new ForDemo();}
  ForDemo(){
    for (int i = 0, j=3; i <= 5; i++, j-=2) {
       System.out.println(i + " " + j);
    }
  }
}
```

Esercizio: scrivere un'applicazione che stampi, in ordine inverso, i valori contenuti in un array di float dall'i-esimo al j-esimo e ne calcola il prodotto

Costrutti condizionali: if, else

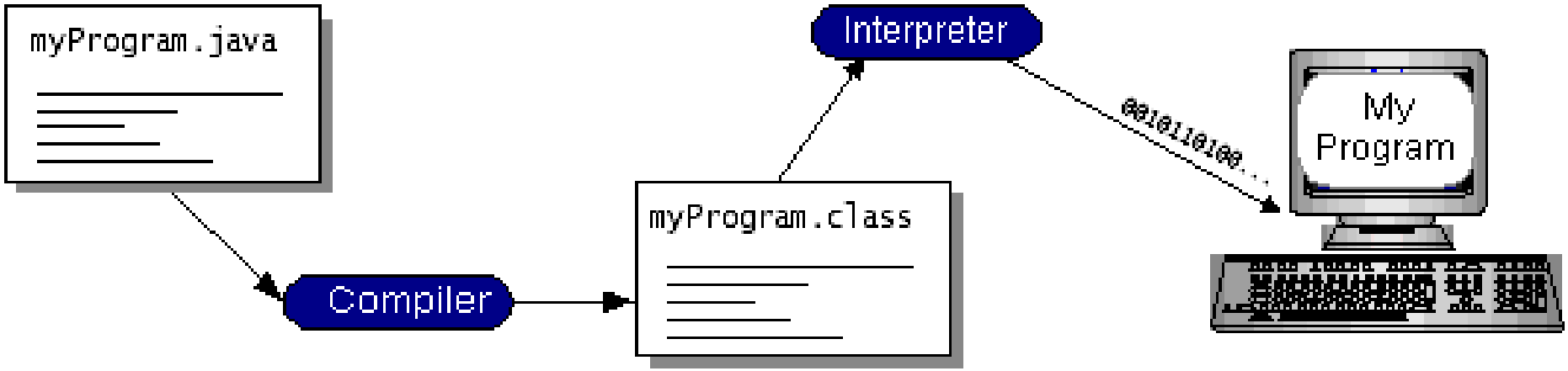
```
public class IfElseDemo {
  public static void main(String[] args) { new IfElseDemo(); }
  IfElseDemo(){
    int punteggio = 76;
    char grade;
    if (punteggio >= 90) { grade = 'A'; }
    else if (punteggio >= 80) { grade = 'B'; } // ...
    else { grade = 'F'; }
    System.out.println("Grade = " + grade);
    System.out.print(“Il punteggio e’ ");
    System.out.println( punteggio<60 ? “basso” : “alto”);
  }
}
```

Costrutti condizionali: switch

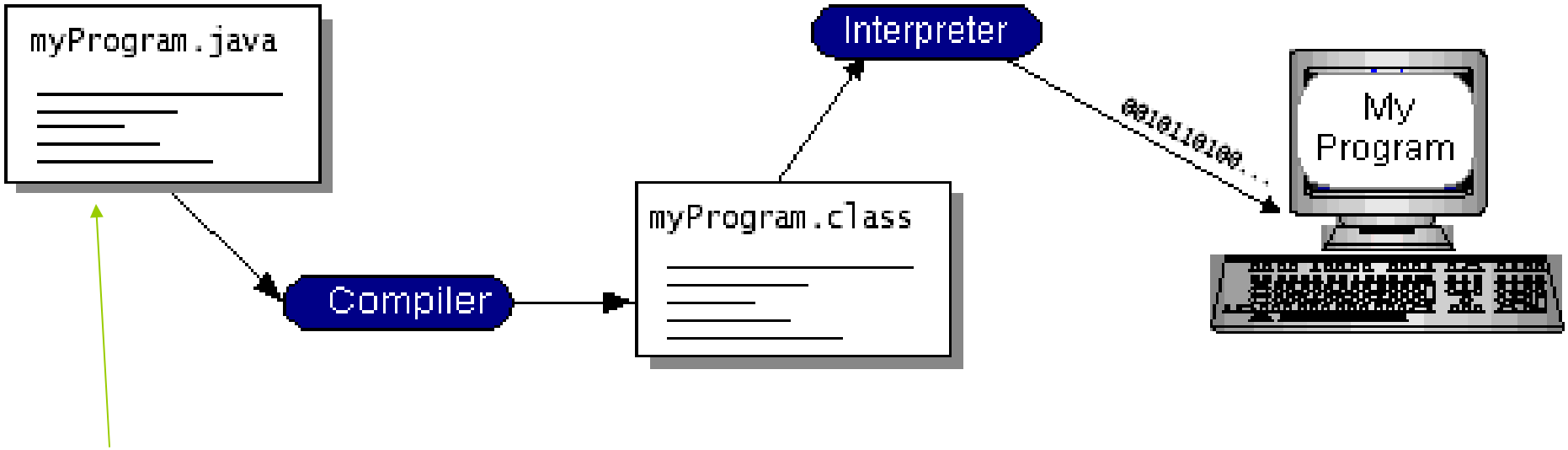
```
public class SwitchDemo {
  public static void main(String[] args) { new SwitchDemo(); }
  SwitchDemo(){
    int mese = 8;
    switch (mese) {
      case 1: System.out.println("Gennaio"); break;
      case 2: System.out.println("Febbraio"); break;
      …
      case 12: System.out.println("Dicembre"); break;
      default: System.out.println("mese non valido!");
    }
  }
}
```

Compilazione ed esecuzione da riga di comando

Un APPLICAZIONE JAVA è un programma stand-alone scritto in linguaggio Java



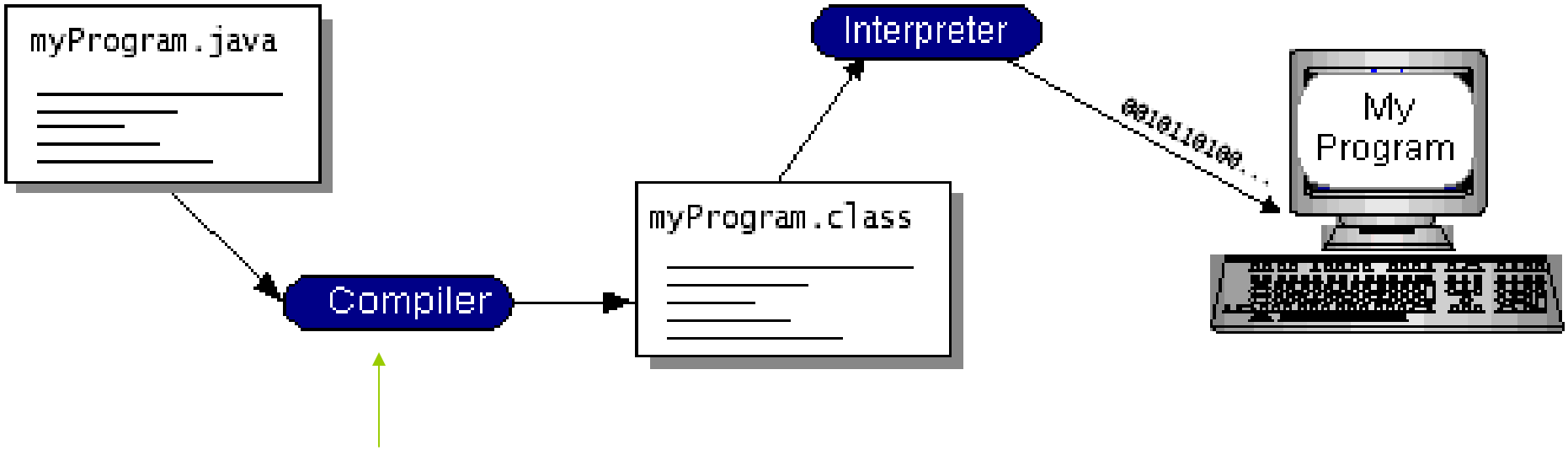
Scrittura di un'applicazione Java



Creazione di un file sorgente.

Un file sorgente è un file di testo, scritto in linguaggio Java.

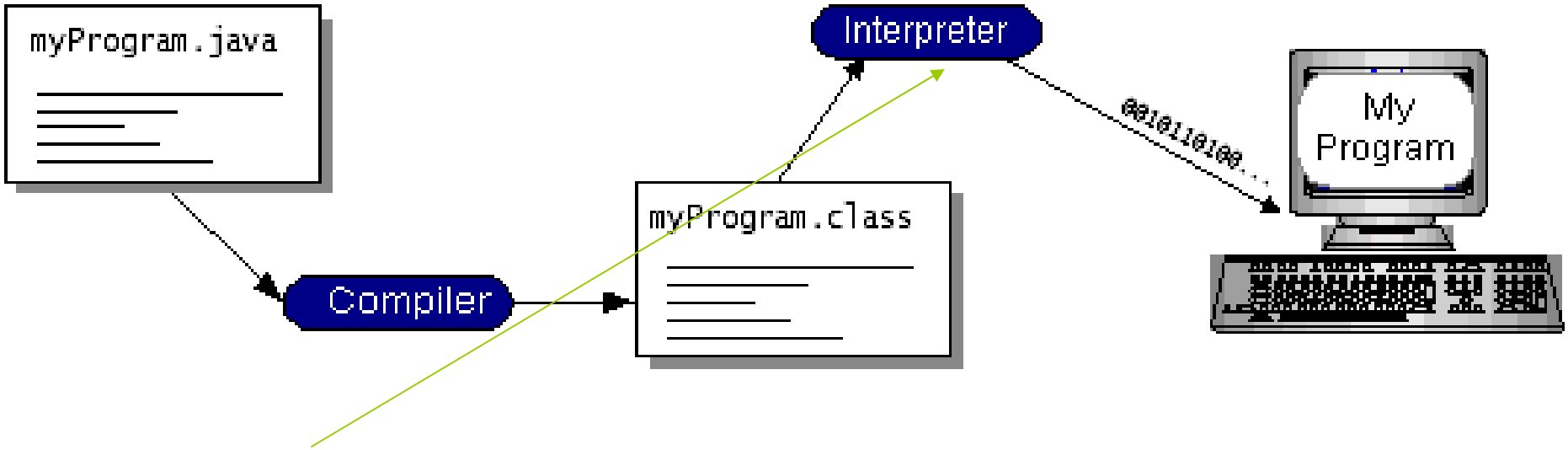
Compilazione



Compilazione del file sorgente in un file bytecode.

Il compilatore java, javac, traduce il testo contenuto nel file sorgente in istruzioni comprensibili alla Java Virtual Machine (Java VM) e le scrive in un file bytecode.

Esecuzione

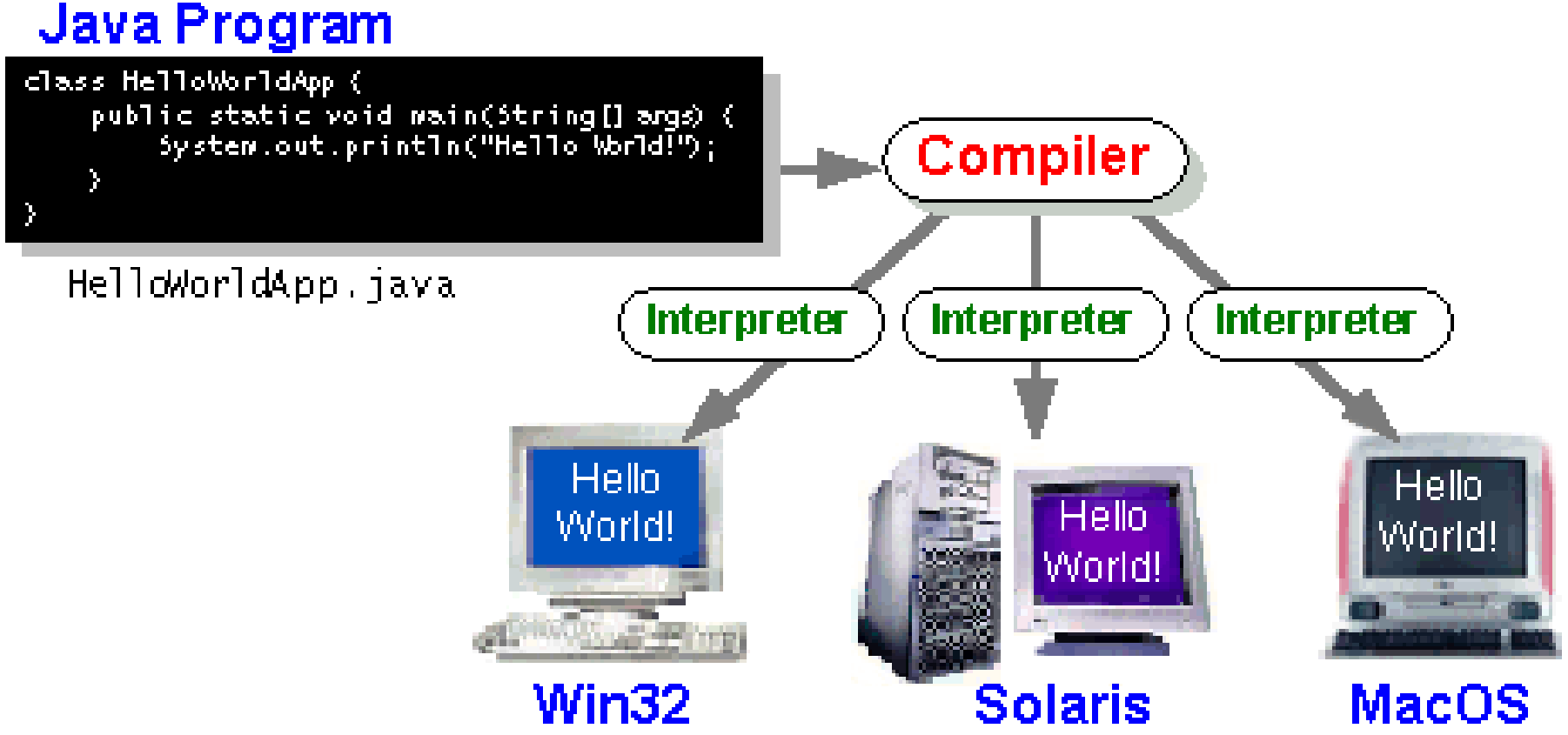


Esecuzione del programma.

La Java VM è implementata da un interprete Java, java. Traduce le istruzioni bytecode (indipendenti dalla piattaforma) in istruzioni comprensibili dal vostro computer e le esegue.

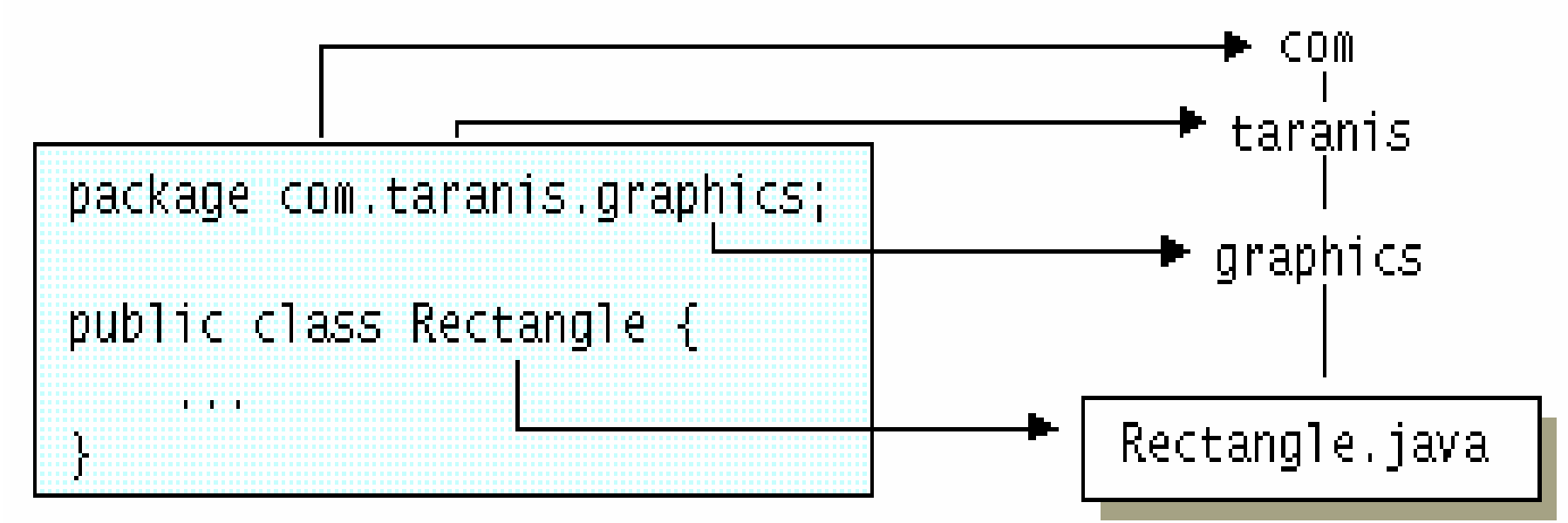
Portabilità del bytecode Java

"write once, run anywhere"



Gestione dei file sorgente (.java) e bytecode (.class)

Gerarchia di directory:



Compilatore Java

javac

Linux: /usr/local/JBuilderx/jdk1.4/bin/javac

Win2000:

C:\jdk1.3.1_04\bin\javac

Compilazione della classe Messaggio:

javac Test.java

Sintassi:

javac [opzioni] file_sorgente

Opzioni di javac

-classpath <classpath>

Specifica dove sono i file bytecode (.class)

-sourcepath <sourcepath>

Specifica dove sono i file sorgente (.java)

-d <directory>

Specifica in quale directory salvare i bytecode prodotti

-verbose

Stampa informazioni aggiuntive durante la compilazione

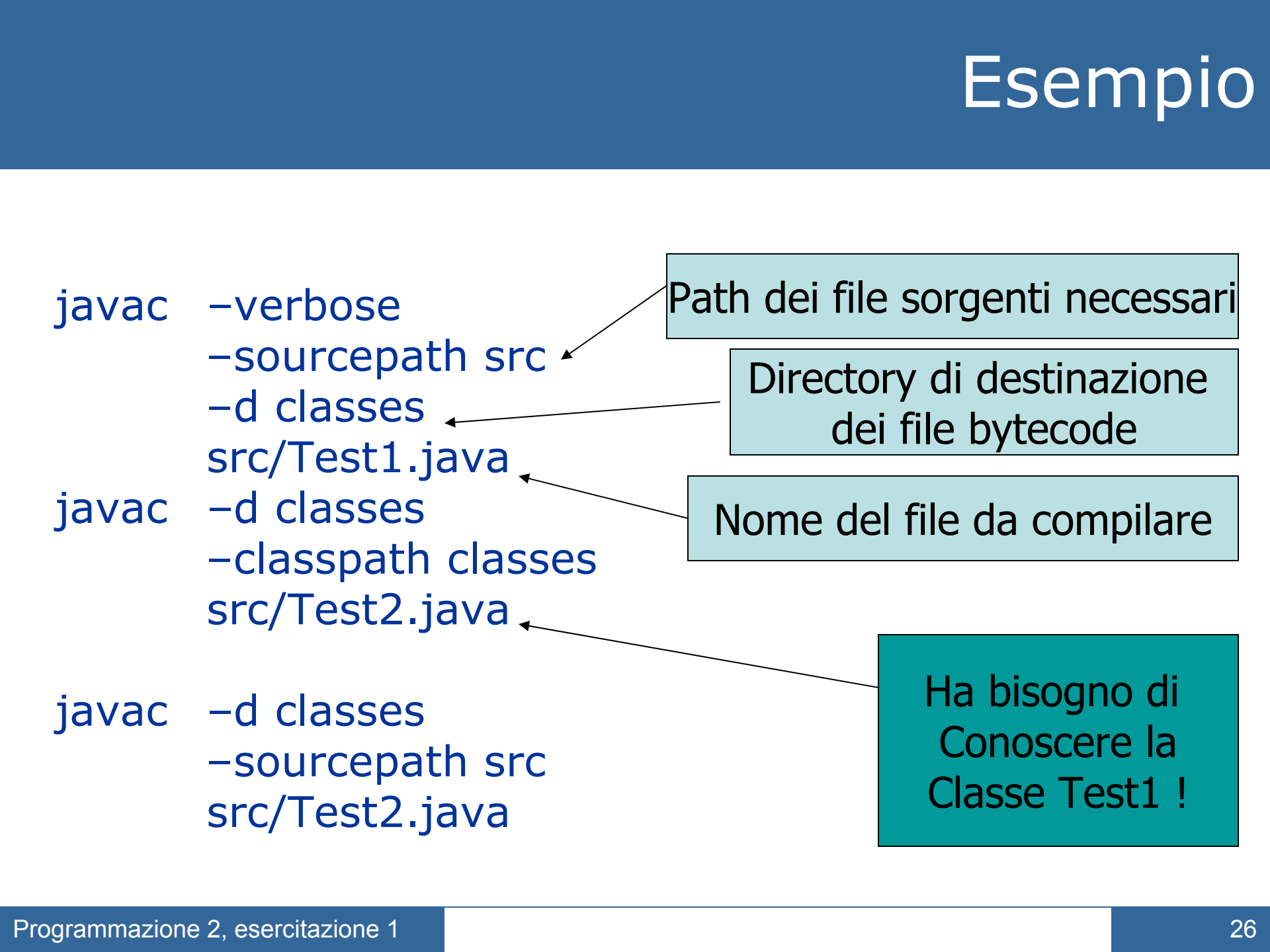
-deprecation

Avverte se sono stati usati metodi deprecati

Esempio

```
javac   –verbose
        –sourcepath src
        –d classes
        src/Test1.java
javac   –d classes
        –classpath classes
        src/Test2.java

javac   –d classes
        –sourcepath src
        src/Test2.java
```



Interprete Java

`java`

Linux:

/usr/local/JBuilderx/jdk1.4/bin/java

Sintassi:

java [opzioni] classe [argomenti]
(opzioni: -classpath –verbose –version –help)

NB: la classe deve aver definito il metodo main:
public static void main(String[] argomenti)

Esempio: java –classpath classes Test

Passaggio di parametri alla riga di comando

Sono delle stringhe passate al programma in fase di esecuzione tramite la riga di comando, dopo il nome del file. Vengono passati automaticamente come parametri al metodo main:

```
public class Hello {
        public static void main (String[] args){
                System.out.println("Hello "+args[0]+"!");
        }
}
```

```
>java Hello Novella
Hello Novella!
>
```