Introduzione a Java (2)

Metodologie di Programmazione A.A. 2008-09

Stefano Guerrini

Corso di Laurea in Informatica
Dipartimento di Informatica.
Sapienza Università di Roma
guerrini@di.uniroma1.it

Febbraio 2009

Input/Output: un esempio

Un semplice programma che legge due numeri e stampa il loro prodotto.

```
1  import java.util.*;
2
3  public class Prodotto {
4      public static void main(String[] args) {
5          Scanner in = new Scanner(System.in);
6          System.out.println("Primo numero: ");
7          int num1 = in.nextInt();
8          System.out.println("Secondo numero: ");
9          int num2 = in.nextInt();
10         System.out.println("Il prodotto e' " + num1*num2);
11     }
12 }
```

Input/Output: un altro esempio

Un programma che legge tre numeri e li stampa in ordine.

```
1  import java.util.*;
2
3  public class Ordine {
4      public static void main(String[] args) {
5          Scanner in = new Scanner(System.in);
6          System.out.println("Digita tre numeri: ");
7          double a = in.nextDouble(),
8              b = in.nextDouble(),
9              c = in.nextDouble();
10         String risultato = "I tre numeri ordinati sono ";
11         if (a <= b) {
12             if (b <= c) risultato += a + " " + b + " " + c;
13             else if (a <= c) risultato += a + " " + c + " " + b;
14             else risultato += c + " " + a + " " + b;
15         } else {
16             if (a <= c) risultato += b + " " + a + " " + c;
17             else if (b <= c) risultato += b + " " + c + " " + a;
18             else risultato += c + " " + b + " " + a;
19         }
20         System.out.println(risultato);
21     }
22 }
```

Standard input/output

- Gli stream per lo standard input/output (stdin/stdout del C) in Java sono associati ai seguenti attributi della classe System

  public static final InputStream in
  public static final PrintStream out

- Lo stream per lo standard error output (stderr in C) è invece associato a

  public static final PrintStream err

- Si tratta di variabili pubbliche di classe di System accessibili con

  System.in System.out System.err

- In realtà il nome completo della classe System è java.lang.System, ovvero, System appartiene al package java.lang. Se non ci sono ambiguità, tutte le classi di java.lang sono accessibili direttamente, senza bisogno di specificare il nome del package (vedremo meglio in seguito).
- Tutti gli stream standard sono pronti per essere usati (aperti e pronti a ricevere o fornire dati) e tipicamente
  - System.in corrisponde alla tastiera o a un'altra sorgente di input specificata dall'ambiente o dall'utente (ad esempio, mediante redirezione di un output);
  - System.out corrisponde al display o a un'altra destinazione di output specificata dall'ambiente o dall'utente.

Output: la classe PrintStream

- Questa classe appartiene al package java.io.
- Aggiunge funzionalità a un output stream di più basso livello (ad esempio, OutputStream) permettendo la stampa di valori appartenenti a diversi tipi di dato (a basso livello normalmente si lavora invece su blocchi di byte).
- Il metodo di istanza print stampa valori di tutti i tipi primitivi e stringhe

```
public void print(boolean b)      System.out.print(true)
public void print(char c)         System.out.print('a')
public void print(int i)          System.out.print(18)
public void print(double d)       System.out.print(3.6)
public void print(String s)       System.out.print("Esempio'')
```

```
System.out.print("Il numero " + 3 + " e' dispari.")
```

- Il metodo println è simile a print, ma aggiunge un accapo.

Input: la classe Scanner

- Lo standard input System.in ha tipo InputStream.
- È uno stream di basso livello che permette di leggere blocchi di byte.
- Per semplificare la lettura dei dati, occorre utilizzare una versione di uno stream di input di più alto livello associato a System.in che permetta di leggere direttamente valori dei tipi primitivi o stringhe.
- La classe Scanner del package java.util suddivide l'input ricevuto dallo stream di basso livello in pezzi o token utilizzando gli spazi come separatori (è possibile anche chiedere di usare altri separatori).
- I token possono poi essere convertiti in valori dei tipi richiesti utilizzando gli appositi metodi next.

```
Scanner in = new Scanner(System.in);     crea un nuovo Scanner
                                         associato a System.in
String linea = in.nextLine();            legge una linea
int num = in.nextInt();                  legge un intero
double a = in.nextDouble();              legge un double
String word = in.next();                 legge una parola
```

- Uno scanner può anche essere associato a una stringa usando il costruttore

  public Scanner(String source) Scanner in = **new** Scanner(stringa);

  I token saranno quelli ottenuti dalla lettura della stringa.

Tipi primitivi

Non tutti i dati sono oggetti; non tutte le variabili sono riferimenti a oggetti.
Per ragioni di efficienza, Java fornisce otto tipi primitivi corrispondenti a valori con una diretta rappresentazione nella memoria della JVM.
Le variabili appartenenti ai tipi primitivi rappresentano locazioni di memoria contenenti valori del corrispondente tipo.

dominio	tipo	dim./rappr.	range	default
interi	**int**	32 bit	−2.147.483.648 .. 2.147.483.647	0
	long	64 bit	−9.223.372.036.854.775.808 .. 9.223.372.036.854.775.807	0L 0
	short	16 bit	−32.768 .. 32.767	0
	byte	8 bit	−128 .. 127	0
reali in virg. mob.	**double**	64 bit IEEE 754		0.0
	float	32 bit IEEE 754		0.0F
caratteri	**char**	16 bit Unicode	'\u0000' .. '\uFFFF'	'\u0000'
booleani	**boolean**	1 bit	**true**, **false**	**false**

Notare che tutti i tipi hanno dimensioni fissate, mentre in C sappiamo solo che un long non è più corto di un int, uno short non è più lungo di un int e che i char hanno dimensione di un byte.
Tipi principali: **int** per gli interi e **double** per i reali in virgola mobile.
Suffissi: L = **long**, F = **float**
Esadecimale: 0xFF, 0xFFL Ottale: 0377, 0377L

Char

Literal per caratteri :

- caratteri semplici: 'a', 'A', ';'
- caratteri speciali: '\b', '\t', '\n', '\f', '\r', '\"', '\'', '\\'
- caratteri Unicode (hex): '\u0000', '\u5D0'
- caratteri Latin1 (oct): '\047', '\377'

Java utilizza i caratteri Unicode che hanno dimensione 16 bit (2 byte).
I caratteri Unicode sono una estensione dei caratteri ASCII per rappresentare anche i caratteri alfabetici delle lingue diverse dall'inglese e dalle altre lingue dell'Europa occidentale.
L'estensione è compatibile, ovvero, sui valori compresi tra 0 e 255 i caratteri Unicode coincidono con i caratteri ASCII.
Nella maggior parte delle applicazioni si può ignorare che i caratteri sono Unicode e non ASCII.

Conversioni di tipo (cast)

- Sono permesse conversioni tra i vari tipi di interi, tra i due tipi di reali in virgola mobile e tra interi e reali in virgola mobile.
- I **char** possono essere convertiti da/in valori interi o in virgola mobile.
- Widening conversion: si passa da un tipo più piccolo ad uno più grande (e.g., da interi a reali, da byte a int). Queste conversioni sono eseguite senza problemi.
  short s = 3; **int** i = s;
- Narrowing conversion: si passa da un tipo più grande a uno più piccolo (e.g., da long a int o short, da double a float). Il compilatore non accetta questo di tipo di conversioni se non esplicitamente richieste; se le incontra termina la compilazione con un messaggio di errore.
  int i = 3; **short** s = i; errore
- Una narrowing conversion deve essere esplicitamente chiesta:
  int i = 3; **short** s = (**short**) i; OK.
  double d = 13.456; **int** i = (**int**) d; OK.
  short i = 0x00FF; **byte** b = (**byte**) i; Taglia 2 byte: 255 → -1
- Attenzione alla conversione tra **char** e **short**: hanno la stessa lunghezza, ma i **char** non hanno segno.
  char c = '\uFFFF'; **short** s = (**short**) c; OK. Ma s contiene -1!
- La classe Math contiene metodi che operano sui numeri, incluse le funzioni per l'arrotondamento dei reali in virgola mobile.

Operatori

Praticamente tutti gli operatori del linguaggio C sono definiti in Java con lo stesso significato.

assegnamento semplice	`=`
aritmetici	`+ - * / %`
segno	`+ -`
incremento/decremento	`++ --`
booleani	`&& || !`
relazionali	`== != > >= < <=`
condizionale (if-then-else)	`?:`
sui bit (bitwise)	`~ << >> >>> & ^ |`
autoincremento, autodecremetno, ecc.	`+= -= *= ...`
	`&= |= ... <<= >>= ...`

Controllo del flusso

```
1 if (i > 0 && j != 0) {
2     i += ++j;
3 }
```

```
1 while (i > 0) {
2   j = j * i;
3   i--;
4 }
```

```
1 do {
2   j = j * i;
3   i--;
4 } while (i > 0)
```

```
1 for (int i = 10; i > 0; j *= i) {
2     ...
3 };
```

```
1 swicth (n) {
2   case 1:
3       ...
4       break;
5   case 2:
6       ...
7   ...
8   default:
9       ...
10      break;
11 }
```

```
1 while {
2     ...
3         break;
4     ...
5 }
```

```
1 for (i > 0; i < 10; i++) {
2     ...
3         continue;
4     ...
5 }
```

References

Le variabili per gli oggetti contengono dei puntatori o references.

```
Greeter worldGreeter = new Greeter("World");
```

Quando lo stato di un oggetto si modifica: tutti i riferimenti vedono le modifiche dell'oggetto.

Il riferimento null.

- Non fa riferimento a nessun oggetto.
- Può essere assegnato a qualsiasi variabile che fa riferimento a oggetti, indipendentemente dal suo tipo;
- È il valore di default per le variabili di oggetti.
- Accedere a un riferimento **null** causa un errore (una eccezione di tipo NullPointerException).

Il riferimento this.

- Si riferisce all'oggetto che riceve il messaggio.
- È il parametro implicito di ogni invocazione di metodo

```
1 public void setPrice(int price)
2 {
3     this.price = price;
4 }
```

Passaggio dei parametri (1)

- Call-by-value: come in C, in Java il passaggio dei parametri è per valore.
- Quando viene chiamato, un metodo rivece una copia dei suoi parametri
- Ricordando che gli oggetti della classe Greeter hanno un attributo privato String name, supponiamo di voler definire i seguenti metodi:

```
1  public void copyLengthTo(int n)
2  {
3      n = name.length();
4  }


5  public void copyGreeterTo(Greeter other)
6  {
7      other = new Greeter(name);
8  }
```

Supponendo di avere le seguenti dichiarazioni

```
9  Greeter worldGreeter = new Greeter("World");
10 Greeter daveGreeter = new Greeter("Dave");
11 int length = 0;
```

Le seguenti due chiamate di metodi non hanno alcun effetto

```
12 worldGreeter.copyLengthTo(length); // length rimane a 0
13 worldGreeter.copyGreeterTo(daveGreeter) // daveGreeter immutato
```

Passaggio dei parametri (2)

- Dato che la copia di un riferimento punta sempre allo stesso oggetto, un metodo può modificare l'oggetto corrispondente a un riferimento ricevuto come parametro. Ad esempio:

```
1 public void copyNameTo(Greeter other) { other.name = this.name; }
2 worldGreeter.copyNameTo(daveGreeter);
```

- In qualche modo possiamo dire che il passaggio degli oggetti avviene per riferimento. Attenzione! Tecnicamente si tratta sempre di call-by-value visto che le variabili che hanno una classe come tipo non denotano un oggetto di quella classe, ma un riferimento a un oggetto, e i riferimenti sono passati per valore.
- Non essendo possibile dereferenziare (prendere il puntatore) a una variabile di un tipo base o a un riferimento, in Java non è possibile simulare facilmente il call-by-reference.
  Ricordiamo che in C, il call-by-reference per una variabile di tipo T si può simulare utilizzando un parametro di tipo *T e l'operatore & nella chiamata. Ad esempio:

```
3 void assign(int *i, int j) { *i = j; }
```

  permette di assegnare 0 a i chiamando assign(&i, 0)

Attributi static

- Gli attributi static o attributi di classe sono condivisi da tutti gli oggetti della classe e, se public, accessibili aanche al di fuori della classe.
  Esempio: gli stream di I/O
  ```
  public static final InputStream in;
  public static final PrintStream out;
  ```
  Esempio. Un generatore di numeri random condiviso.

```
1 public class Greeter
2 {
3     . . .
4     private static Random generator;
5     . . .
6 }
```

- Permettono una comunicazione di valori simile a quella ottenuta con le variabili globali in C (da usare solo in casi particolari).
- Sono spesso utilizzati per contenere costanti di rilievo per ogetti della classe o di utilità generale.
  In questo caso, la dichiarazione della variabile è preceduta dalla parola chiave **final**, per indicare che non è possibile cambiarne il valore.
  Esempio. Nella classe Math è definita una costante per π.
  ```
  public static final double PI 3.141592653589793d;
  ```

Metodi static

- Forniscono operazioni di ausilio ai metodi degli oggetti della classe o, se public, di utilità generale.
- Sono indipendenti dai valori degli oggetti della classe e non operano su di essi, ma dipendono esclusivamente dai parametri ricevuti nella chiamata e dagli attributi static.
- Esempio. Math.sqrt
  `public static double sqrt(double a)`
- Esempio: *factory method* (si tratta di un *design pattern*, lo analizzeremo in dettaglio in seguito)

```
1 public static Greeter getRandomInstance()
2 {
3     if (generator.nextBoolean()) // note: generator is static field
4         return new Greeter("Mars");
5      else
6         return new Greeter("Venus");
7 }
```

- Sono chiamati attraverso la classe
  `Greeter g = Greeter.getRandomInstance();`
- Attributi e metodi static vanno usati con attenzione nei programmi OO. Solo in pochi casi sono la scelta migliore e più naturale.

Oveloading di metodi (1)

- Nelle chiamate di metodi, il compilatore Java seleziona quale metodo effettivamente chiamare in base al nome e alla sua segnatura (numero di parametri ricevuti e loro tipo).
- Java permette quindi l'overloading dei metodi.
- Ovvero, una classe può contenere metodi con lo stesso nome ma con differenti liste di parametri (con qualche eccezione legata all'ereditarietà che vedremo in seguito).
- Ad esempio, la classe PrintStream contiene diverse versioni del metodo println , tra cui

```
public void println(int x)
public void println(short x)
public void println(double x)
public void println(String x)
```

- Il compilatore non considera il tipo restituito nel distinguere metodi. Ad esempio, non si può definire nella stessa classe

```
public int getNum(String s);
public long getNum(String s);
```

Oveloading di metodi (2)

- Nel chiamare metodi overloaded occorre garantire che il compilatore riesca a determinare il metodo da chiamare in base al tipo delle espressioni passate nei parametri.

  Esempio. Supponiamo che un oggetto obj abbia due metodi

  ```
  public void esempio(Int i);
  public void esempio(String s);
  ```

  Nessun problema con

  ```
  obj.esempio(new Int(3));
  obj.esempio("alfa");
  ```

  Ma in questo caso

  ```
  obj.esempio(null);
  ```

  quale metodo si vuole chiamare?

Oveloading di metodi (3)

- L'overdoaling è particolarmente utile per i costruttori. Permette di avere diversi modi per creare e inizializzare oggetti di una classe.

```
21      /**
22       * Constructor for ClockDisplay objects. This constructor
23       * creates a new clock set at 00:00.
24       */
25      public ClockDisplay()
26      {
27          hours = new NumberDisplay(24);
28          minutes = new NumberDisplay(60);
29          updateDisplay();
30      }
31
32      /**
33       * Constructor for ClockDisplay objects. This constructor
34       * creates a new clock set at the time specified by the
35       * parameters.
36       */
37      public ClockDisplay(int hour, int minute)
38      {
39          hours = new NumberDisplay(24);
40          minutes = new NumberDisplay(60);
41          setTime(hour, minute);
42      }
```

I package: introduzione

- Un grosso progetto Java può essere composto da centinaia o migliaia di classi scritte da programmatori o società diverse.
- Al fine di poter trovare e utilizzare le classi è pertanto necessario mantenere organizzate le classi raggruppandole in package.
- Essendo impossibile garantire nomi distinti o assenza di conflitti per classi sviluppate da programmatori distinti e in tempi/luoghi diversi i package devono anche garantire degli spazi di nomi o namespace distinti per raggruppare classi correlate.
- I package non contengono solo classi, ma anche interfacce (scheletri di classi senza codice che vedremo in seguito), e più in generale tipi.
- I tipi contenuti in un package sono i membri del package.
- Il nome completo, o fully qualified, di un package è composto da una sequenza di identificatori (lettere minuscole, cifre decimali e '_') separati da '.'

  java.lang com.company.package_name
- Per convenzione, quando possibile, il nome completo di un package si ottiene facendo precedere il nome del package dal dominio internet invertito di chi ha realizzato il package.

Spazi di nomi

- Concettualmente i package possono essere visti come directory distinte.
- Anche praticamente ogni package viene mantenuto in una directory e la sequenze di nomi separati da '.' corrisponde ad un cammino di accesso relativo alla directory di base in cui si trovano i package

  java/lang com/company/package_name
- Anche se organizzati in directory, e quindi memorizzati in una struttura gerarchica, i package non formano una gerarchia. Ad esempio, questi sono tutti package distinti

  java.awt java.awt.color java.awt.font
- java.awt.color, e più in generale ogni package java.awt.xxxx, non è un membro di java.awt né un suo sotto-package.
- Importando java.awt.* non si importa nessun package java.awt.xxxx

Import

- Per utilizzare un membro pubblico di un package, si può
  - far riferimento al membro del package per mezzo del suo nome completo
    java.util.Math java.util.Math.PI
    java.io.InputStream java.io.OutputStream
  - importare il membro del package
    import java.util.Math;
    import java.io.InputStream;
    import java.io.File;
  - importare tutti i membri del package
    import java.util.*;
    import java.io.*;
- Si osservi che si può importare un membro per volta o tutti i membri del package in una sola volta.
  Non si può usare l'asterisco come in una maschera o in una espressione regolare per importare solo parte dei membri del package.
  import java.util.M*; *// non valido!*
  o per selezionare tutti gli elementi con lo stesso nome in diversi package
  import java.*.Math; *// non valido!*

Ambiguità e static import

- Ambiguità. Se si importano due membri con lo stesso nome da package diversi, per accedere a questi membri occorre usare il nome completo.
- Static import. È possibile importare costanti metodi static di una classe

  ```
  import static java.lang.Math.PI;
  import static java.lang.Math.sqrt;
  ```

  o anche tutte le costanti e i metodi statici contemporaneamente

  ```
  import static java.lang.Math.*
  ```

```
1  // StaticImportTest.java
2  // Using static import to import static methods of class Math.
3    import static java.lang.Math.*;
4
5  public class StaticImportTest
6  {
7       public static void main( String args[] )
8       {
9            System.out.printf( "sqrt( 900.0 ) = %.1f\n", sqrt( 900.0 ) );
10           System.out.printf( "ceil( -9.8 ) = %.1f\n", ceil( -9.8 ) );
11           System.out.printf( "log( E ) = %.1f\n", log( E ) );
12           System.out.printf( "cos( 0.0 ) = %.1f\n", cos( 0.0 ) );
13      } // end main
14 } // end class StaticImportTest
```

Come creare un package

- Se non si assegna nessun package a una classe, questa finisce in un package senza nome.
- Per creare un package occorre scegliere un nome, ad esempio
  it.uniroma1.di.guerrini.esercizi
- creare la corrispondente struttura di directory, per l'esempio
  it/uniroma1/di/guerrini/esercizi
- Scrivere tutti i file sorgenti dei tipi che si vogliono inserire nel package nella directory esercizi
- Inserire una dichiarazione di package
  package it.uniroma1.di.guerrini.esercizi ;
  all'inizio di ogni file sorgente che contiene il tipo da includere nel package
- La dichiarazione del pacakge deve essere la prima linea del file sorgente (ma può essere precedeuta da commenti o linee bianche).
- In un file sorgente può esserci al massimo una dichirazione di package.
- La variabile di sistema CLASSPATH può essere usata per specificare la radice delle directory dei package.

I package nelle Java API

I nomi dei package delle API di Java cominciano con java o javax.

I package base della piattaforma Java:

- java.lang (i suoi tipi vengono importati automaticamente)
- java.util
- java.io

Altri package che useremo

- java.applet
- java.awt
- javax.swing
- java.math

Altri package di uso frequente

- java.beans
- java.net
- javax.net
- javax.sql
- javax.xml