Programmazione in Java (3)

Mauro Lorenzutti

Scaletta

- Variabili e assegnamenti
- Espressioni e tipi di dato predefiniti
- Operatori e strutture di controllo
- Introduzione agli oggetti
- Creare una classe

Variabile

- È un nome che indica una locazione di memoria usata per contenere un valore

```
public class Esempio1 {

   public static void main(String[] args){
      int variabile = 5;
      System.out.println(variabile);
   }
}
```

- Perché il "main" è static?

Assegnamenti 1/2

- Consiste nell'assegnare un valore ad una variabile

```
public class Esempio2 {

   public static void main(String[] args){
      int v1 = 10;
      System.out.println("v1: "+v1);
      int v2 = v1;
      System.out.println("v2: "+v2);
      v1 = 15;
      System.out.println("v1: "+v1);
      System.out.println("v2: "+v2);
   }
}
```

Assegnamenti 2/2

- Java è un linguaggio fortemente tipizzato

```
public class Esempio2 {

   public static void main(String[] args){
      int v1 = 10;
      v1 = "Stringa"; // NO!!!!
   }
}
```

Costanti

- Sono particolari "variabili"
- Mantengono il proprio valore per tutta la loro esistenza
- Quando si usano?

```
final int CAPACITA_MASSIMA = 100
```

Espressioni e tipi di dato predefiniti

- Definizioni
- Tipi di dato predefiniti
- Espressioni aritmetiche
- Precedenza fra gli operatori
- Conversioni di tipo

Definizioni

- In Java i dati sono rappresentati come:
  - *Dati primitivi*
  - *Oggetti*
- Un *tipo di dato* definisce un insieme di valori e le operazioni che si possono eseguire su tali valori

Tipi di dato predefiniti

Tipo	Occupazione di memoria	Valore minimo	Valore massimo
byte	8 bit	-128	127
short	16 bit	-32.768	32767
int	32 bit	-2.147.483.648	2.147.483.647
long	64 bit	~ -9E18	~ 9E18
float	32 bit	~ -3,4E38 (7 cifre significative)	~ +3,4E38 (7 cifre significative)
double	64 bit	~ -1,7E308 (15 cifre significative)	~ -1,7E308 (15 cifre significative)
char	16 bit	Codice Unicode (65536 caratteri)	
boolean	{true, false}		

Literal

- Un *literal* è un valore esplicito usato nel programma

```
public class Esempio3 {
   public static void main(String[] args){
      byte v1 = 32;
      short v2 = 32;
      int v3 = 32;
      long v4 = 32L;
      float v5 = 33.0F;
      double v6 = 33.0;
      char v7 = 'a';
      boolean v8 = true;
   }
}
```

Espressioni aritmetiche

- +, -, *, /, %
- Attenzione al /

```
public class Esempio4 {
   public static void main(String[] args){
      int v1 = 33;
      int v2 = 5;
      int v3 = v1/v2;
      double v4 = 33;
      double v5 = v4/v2;
      double v6 = (double)v1/v2;
      }
}
```

Precedenze

- Le operazioni seguono regole di precedenza
  - risultato = 12 + 2 * 4 // =20
  - risultato = (12 + 2) * 4 // =56

Conversioni di tipo

- 2 tipi:
  - Larga
  - Stretta
- 3 casi:
  - Conversione durante un assegnamento
  - Promozione in un'espressione aritmetica
  - Casting

Conversione durante un assegnamento

```
public class Esempio5 {

   public static void main(String[] args){
      float v1;
      int v2 = 33;

      v1 = v2;
   }
}
```

Promozione in un'espressione aritmetica

```
public class Esempio6 {

   public static void main(String[] args) {
      float risultato;
      float somma = 10F;
      int soci = 5;

      risultato = somma / soci;
   }
}
```

Casting

```
public class Esempio4 {
   public static void main(String[] args){
      int v1 = 33;
      int v2 = 5;
      double v3 = (double)v1/v2;
      char v4 = (char)v1;
      v2 = (int)v4
      }
}
```

- Attenzione ai boolean: non possono essere convertiti

Operatori e strutture di controllo

- If
- Operatori di confronto e operatori vari
- Switch
- While
- Do
- For
- Break & continue

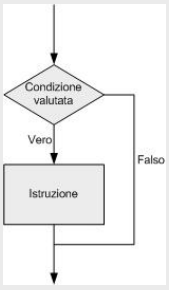
If

- Istruzione condizionale

```
If (condizione)
   istruzione
```

- Alternativa

```
if (condizione)
   istruzione 1
else if
   istruzione 2
else
   istruzione default
```



Operatori di confronto

Operatore	Significato
==	Uguale a
!=	Diverso da
<	Minore di
<=	Minore o uguale a
>	Maggiore di
>=	Maggiore o uguale di

Esempio

```
if (v1>v2){
    System.out.println(v1+" è maggiore di "+v2);
}   else if (v1<v2) {
    System.out.println(v1+" è minore di "+v2);
}   else    {
    System.out.println(v1+" è uguale a "+v2);
}
```

Operatori logici

Operatore	Descrizione
!	Not
&&	And
||	Or

```
if (v1>v2 || v1<v2)
    System.out.println(v1+" è diverso da "+v2);
if (v1>v2 && v1<v2)
    System.out.println("ERRORE");
```

Numeri in virgola mobile

- Attenzione al confronto (precisione)
- Non si usa l'uguaglianza

```
if (v1==v2)
    System.out.println("Impreciso");
if ((v1-v2)<0.00001)
    System.out.println("Più preciso");
```

Esempio

- Cosa c'è di sbagliato?

```
if (totale==MAX)
   if (totale<somma)
        System.out.println("totale è uguale a MAX e minore di
                somma");
else
   System.out.println("totale è diverso da MAX");
```

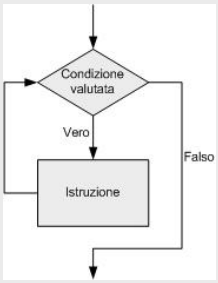
Switch

- Valuta un'espressione e ne confronta il valore con le clausole (valuta solo l'uguaglianza)

```
char c1 = 'd';
switch (c1) {
   case 'a':
      System.out.println("Il carattere è una a");
      break;
   case 'b':
      System.out.println("Il carattere è una b");
      break;
   default:
      System.out.println("Il carattere è una c");
      break;
}
```

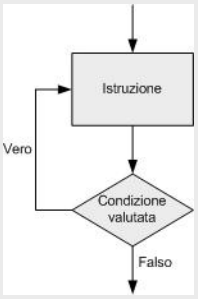
While

```
int i=0;
while(i<10){
    System.out.println(i);
    i++;
}
```



Do

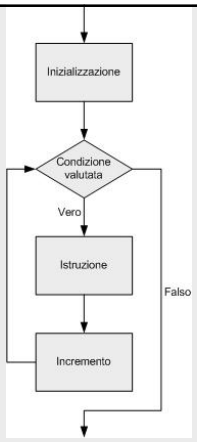
```
int j=0;
do  {
   System.out.println(j);
   j++;
}   while (j<10);
```



For

- Ciclo da eseguire un numero predefinito di volte

```
for (int h=0; h<10; h++){
   System.out.println(h);
}
```



Break & continue

- Istruzioni "delicate"
- Break: per interrompere un ciclo
- Continue: per saltare alla prossima iterazione
- Da evitare, attenzione…

Operatori di incremento e decremento

```
int count = 0;
int total;
count++;    // count = count+1
++count;    // count = count+1

count--;    // count = count-1
--count;    // count = count-1

total = count++;    //total=0; count=1;
total = ++count;    //total=2; count=2;
```

Operatori di assegnamento e condizionale

- Qualche esempio

```
total += 5;                 // total = total + 5
total *= count + count;     // total = total * (count + count)
```

- Operatore condizionale

```
total = (total>10) ? total-10 : 5;

if (total>10)
   total -= 10;
else
   total = 5;
```

Introduzione agli oggetti

- Definizioni
- Un esempio: le stringhe
- Librerie e package

Definizioni 1/2

- In Java i dati sono rappresentati come:
  - *Dati primitivi*
  - *Oggetti*
- Gli oggetti sono usati per rappresentare concetti più specializzati e complessi

Definizioni 2/2

- Un oggetto è definito dalla sua *classe* (~ il tipo dell'oggetto)
- Una classe contiene dei metodi
  - I metodi rappresentano le operazioni che si possono compiere sugli oggetti della classe
- Un oggetto è un'*astrazione*, i dettagli dell'implementazione sono irrilevanti ai fini dell'utilizzo

Le stringhe

- Una stringa di caratteri è un oggetto definito dalla classe *String*

```
public class Esempio8 {

   public static void main(String[] args) {
      String v1 = "Questa è una stringa";
      String v2 = "Questa è una seconda stringa";
      String v3 = v1+v2;
   }
}
```

Sequenze di escape 1/2

Sequenza	Descrizione
\b	Carattere di cancellazione
\t	Carattere di tabulazione
\n	Nuova linea
\r	Ritorno carrello
\"	Doppi apici
\'	Apice singolo
\\	Barra inversa

Sequenze di escape 2/2

```
public class Esempio9 {

   public static void main(String[] args) {
      String v1 = "Per andare a capo: \"\\n\"\n";
      String v2 = "Sequenza di \"escape\"";
      String v3 = v1+v2;
      System.out.println(v3);
   }
}
```

La stringa come oggetto 1/3

- Una variabile può contenere un *riferimento a un oggetto*

```
String variabile;
```

- Per creare un oggetto si usa l'operatore *new* (istanziazione)

```
variabile = new String("Testo");
```

- Un oggetto è un'*istanza* di una classe particolare
- L'operatore *new* chiama il costruttore della classe

La stringa come oggetto 2/3

- Dopo aver istanziato l'oggetto si può usare l'operatore . (*dot*) per usarne i metodi

```
variabile.toUpperCase();
```

- Per conoscere i metodi applicabili → API (*Application Programmer Interface*)

La stringa come oggetto 3/3

- Attenzione al confronto!

```
if (v1==v2)
    System.out.println("Confronto errato");
if (v1.equals(v2))
   System.out.println("Confronta uguaglianza stringhe");
if (v1.equalsIgnoreCase(v2))
   System.out.println("Confronta uguaglianza stringhe
        ignorando maiuscole/minuscole");
```

Librerie di classi e package

- Una *libreria di classi* è un insieme di classi (fra loro collegate) a disposizione del programmatore
- La classe *String* fa parte della libreria standard fornita con Java
- In Java le classi sono raggruppate in package
- La classe *String* fa parte del package `java.lang`

La clausola `import` 1/3

- Le classi del package `java.lang` sono automaticamente disponibili
- Per usare classi di altri package sono possibili due strade:
  - Specificare il nome del package insieme al nome della classe
  - Usare la clausola import

La clausola `import` 2/3

- Prima alternativa:

```
public class Esempio10 {

   public static void main(String[] args) {
      java.util.Random r = new java.util.Random();
      System.out.println(r.nextInt());
   }
}
```

La clausola `import` 3/3

- Seconda alternativa:

```
import java.util.Random;
// import java.util.*;

public class Esempio11 {
    public static void main(String[] args) {
        Random r = new Random();
        System.out.println(r.nextInt());
    }
}
```

Esercizio 1

- Progettare e implementare un programma che stampa un sottoinsieme dei caratteri Unicode e il relativo valore numerico. Stampare una tabella ben formattata per i valori da 32 a 126.

Esercizio 2

- Progettare e implementare un'applicazione che stampi la somma di tutti gli interi pari compresi fra 2 e un intero random, estremi inclusi. Stampare a video la somma e l'intero random. Prevedere un messaggio di errore se l'intero random è minore di 2.

Creare una classe

- Definizioni
- Anatomia di una classe
- Anatomia di un metodo
- Una classe per i numeri complessi

Definizioni

- Un oggetto è composto da:
  - *Varibili*: definiscono lo stato dell'oggetto
  - *Metodi*: definiscono il comportamento dell'oggetto
- Un oggetto è definito da una classe
- Una classe è il modello su cui viene istanziato un oggetto

Anatomia di una classe

public class NomeClasse{

Variabili di istanza	(private)
Costruttore	(public)
Metodi	(public)
Supporto	(private)

}

Dati di istanza

- La classe definisce un insieme di variabili
- Ad ogni istanziazione della classe viene riservato uno spazio di memoria per ogni variabile d'istanza
- Il loro valore può essere diverso anche fra oggetti della stessa classe
- I metodi sono condivisi fra tutti gli oggetti di una classe

Incapsulamento

- Un oggetto deve *governarsi da sé*: le sue variabili può modificarle solo lui
- Un oggetto dovrebbe essere **incapsulato** rispetto a tutto il resto del sistema e dovrebbe interagire col resto del programma solo attraverso i propri metodi (l'interfaccia dell'oggetto)

Modificatori di visibilità

- Sono applicati ai metodi e alle variabili(…)
- public
- private
- protected (vedrete dopo l'ereditarietà)

Anatomia di un metodo

- La *dichiarazione di un metodo* specifica il codice da eseguire ad ogni invocazione del metodo
- Programmi *driver*:
  - Programmi guida, programmi che guidano l'uso di altre parti

L'istruzione return

- Un metodo deve restituire un dato di un certo tipo (primitivo, oggetto o *void*)
- Un metodo restituisce un valore attraverso l'istruzione *return*

```
public int getRandomInt()  {
   Random r = new Random();
   return r.nextInt();
}
```

Parametri

- Ad un metodo possono essere forniti dei *parametri*
  - Le liste di parametri dell'invocazione e della dichiarazione devono coincidere
  - I tipi devono coincidere
- Parametri formali: i nomi dei parametri nella dichiarazione
- Parametri attuali (o argomenti): i valori passati al metodo alla sua invocazione

Dati locali e visibilità

- La *visibilità* di una variabile è la parte di programma in cui può essere usata
- Una variabile dichiarata in un metodo è locale a tale metodo
- I parametri formali sono locali al metodo

Costruttore

- È un metodo particolare invocato al momento dell'istanziazione dell'oggetto
- Serve per inizializzare gli oggetti
- Due particolari:
  - Deve avere lo stesso nome della classe
  - Non restituisce nulla
- Non è obbligatorio

Una classe per i numeri complessi

- Esercizio: definire una classe *NumeroComplesso* per la gestione dei numeri complessi
- Prevedere i metodi per il calcolo di:
  - somma
  - moltiplicazione
  - complesso coniugato
  - modulo
  - … e per la stampa a video

Una classe per l'input da tastiera

- EasyIn
- InputStreamReader
- BufferedReader
- IOException