Linguaggi di Programmazione I – Lezione 9

Prof. Marcello Sette
mailto://marcello.sette@gmail.com
http://sette.dnsalias.org

6 maggio 2008

Panoramica della lezione

Variabili

Espressioni

Casting di primitivi

Enunciati branch

Enunciati loop

Esercizi

Questionario

Variabili 3 / 31

Ambiti

- Variabili definite all'interno di un metodo sono dette **locali**; esse sono create quando il metodo è invocato e sono distrutte alla sua terminazione; esse DEVONO essere inizializzate esplicitamente prima di essere usate.
- Variabili usate come parametro di metodi, visto il meccanismo di passaggio di parametri (IN per copia), sono variabili locali.
- Variabili definite all'esterno di un metodo sono create quando viene costruito un oggetto. Ve ne sono di due tipi:
  - **Variabili di classe**: sono dichiarate usando il modificatore `static`; sono create nel momento in cui una classe è caricata in memoria (esistono a prescindere dalle istanze); continuano ad esistere finché la classe rimane in memoria; una variabile valida per tutte le istanze (discuteremo meglio in seguito).
  - **Variabili di istanza**: quelle dichiarate senza modificatore `static`; sono create durante la costruzione di una istanza; continuano ad esistere finché l'oggetto esiste; una variabile diversa per ciascuna istanza.

Esempio

```
public class EsempioAmbito {
  private int i=1;

  public void primoMetodo() {
    int i=4, j=5;
    this.i = i + j;
    secondoMetodo(7);
  }

  public void secondoMetodo(int i) {
    int j=8;
    this.i = i + j;
  }
}

public class TestEsempioAmbito {
  public static void main (String[] args) {
    EsempioAmbito ambito = new EsempioAmbito();

    ambito.primoMetodo();
  }
}
```


Inizializzazioni

- Nessuna variabile può essere usata prima di essere inizializzata!
- Le variabili non locali sono inizializzate dalla JVM, nel momento in cui la classe è caricata in memoria o in cui è allocato spazio per il nuovo oggetto, ai seguenti valori:

Variabile	Valore
byte	0
short	0
int	0
long	0L
float	0.0F
double	0.0D
char	'\u0000'
boolean	false
riferimenti	null

- Le variabili locali (quelle dei metodi) DEVONO essere inizializzate manualmente prima dell'uso. Esempio:

```
public void faQualcosa() {
  int x=(int)(Math.random()*9);
  int y;
  int z;
  if (x > 4) {
    y = 9;
  }
  z = y + x;
  // Errore: possibile uso
  // prima della
  // inizializzazione
}
```

Operatori

Simili a quelli di C o C++. In ordine decrescente di precedenza:

Separatori	. [] () ; ,

Associat.	Operatori
R → L	++ -- + - ~ !
L → R	* / %
L → R	+ -
L → R	<< >> >>>
L → R	< > <= >= instanceof
L → R	== !=
L → R	&
L → R	^
L → R	|
L → R	&&
L → R	||
R → L	?:
R → L	= *= /= %= += -=
	&= ^= |= <<= >>= >>>=

- logici

- Operatori booleani: ! (NOT), & (AND), | (OR), ^ (XOR).
- Operatori booleani con corto circuito: || (OR), && (AND).
- Sono usati come segue:

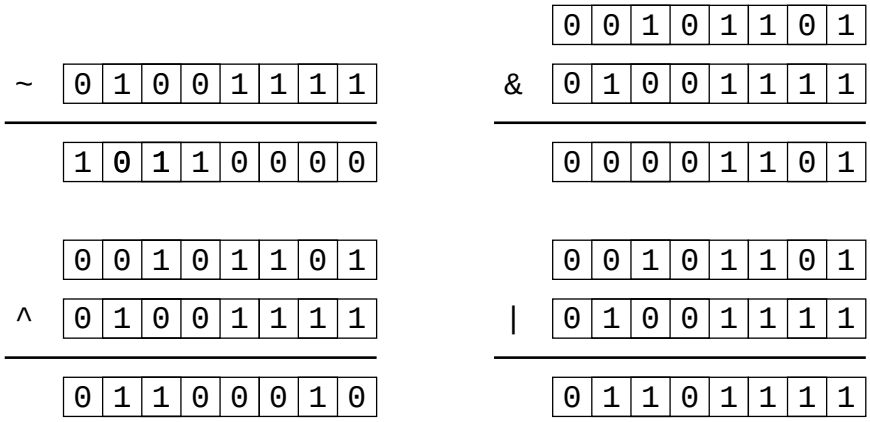
```
int i = 1;
if (i)      // genera un errore
if (i != 0) // corretto

MiaData d;
if ((d != null) && (d.giorno > 31)) {
  // manipola d
}
```

- di bit

- Operatori di manipolazioni di bit su interi: ~ (Complemento ad 1), & (AND), | (OR), ^ (XOR).
- Esempi su `byte`

```
byte a = 45;
byte b = 79;
System.out.println("~b  = " + (byte)(~b));   // -80
System.out.println("a&b = " + (byte)(a&b));  //  13
System.out.println("a^b = " + (byte)(a^b));  //  98
System.out.println("a|b = " + (byte)(a|b));  // 111
```



- right shift

- Nello shift a destra aritmetico o con segno (>>) il bit di segno viene conservato:
  - 128 >> 1 vale 64.
  - 128 >> 3 vale 16.
  - −128 >> 3 vale -16.
- Nello shift a destra logico o senza segno (>>>) il bit di segno è azzerato nello spostamento:
  - −1 >>> 30 vale 3.
- Se l'operando di sinistra è un `int`, prima di applicare lo shift viene ridotto l'operando di destra modulo 32; se l'operando di sinistra è un `long`, prima di applicare lo shift viene ridotto l'operando di destra modulo 64. `x >>> 64` non modifica il valore di `x`.
- L'operatore >>> è ammesso solo per i tipi interi `int` e `long`; se viene usato per `short` o `byte`, questi valori sono promossi, con estensione di segno, ad `int` prima di applicare lo shift.

- left shift

Per lo shift a sinistra non c'è il problema del bit di segno:

- 128 << 1 vale 256.
- 16 << 3 vale 128.
- −1 << 1 vale -2.
- −1 << −31 vale -2.

Concatenazione di stringhe

- L'operatore + esegue la concatenazione di due oggetti `String`, producendo un nuovo oggetto `String`.
- Esempio:

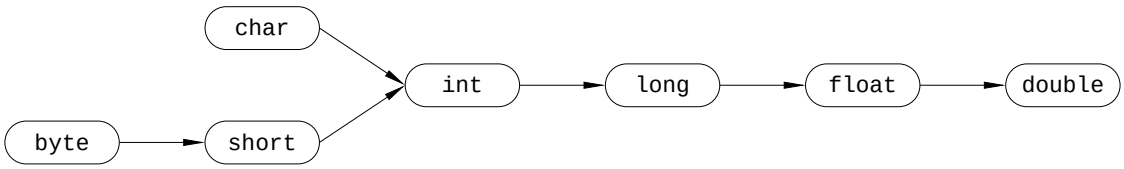
```
String titolo = "Dr.";
String nome = "Valentino" + " " + "Rossi";
String nomecompleto = titolo + " " + nome;
```

- Un argomento deve essere un oggetto `String`.
- Gli altri argomenti sono convertiti a `String` automaticamente, invocando il metodo `toString`.

Casting di primitivi 14 / 31

Conversioni implicite

- Una *conversione di tipo per assegnazione* avviene quando si assegna un valore ad una variabile di tipo differente.
- Ciò può avvenire o con un operatore di assegnazione, o durante il passaggio di parametri ad un metodo.
- Il tipo `boolean` non può essere convertito MAI.
- Le conversioni legali (cioè quelle che avvengono automaticamente) sono quelle per cui esiste un percorso nel grafo:



Conversioni esplicite

- **Regola:** se è possibile la perdita di informazioni in una assegnazione o in un passaggio di parametri, il programmatore DEVE confermare l'assegnazione con un "cast" esplicito.
- In un cast esplicito, il valore in eccesso viene troncato.
- Per esempio, l'assegnazione da `long` a `int` richiede un cast esplicito:

```
long grande = 77L;
int piccolo = grande;       // Sbagliato, richiede cast
int piccolo = (int) grande; // OK

int piccolo = 77L;          // Sbagliato, richiede cast
int piccolo = (int) 77L;    // OK, ma ...
int piccolo = 77;           // pure OK
                            // (default: 77 litterale int)
```

- **Eccezione:** Il casting esplicito non è necessario per i litterali interi (non i floating point) che ricadono nel range legale del tipo di destinazione:

```
int i = 12;
byte b = 12;    // OK
byte b = 128;   // Illegale: e' superiore a 127
byte b = i;     // Illegale: vale 12 ma non e' un litterale
float x = 3.14; // Illegale: eccezione non valida per f.p.
```

Promozione aritmetica

- In una espressione aritmetica, le variabili sono automaticamente promosse ad una forma più estesa per non causare perdita di informazioni.
- Le regole per operatori unari:
  - Se l'operando è un `byte`, uno `short` o un `char`, esso è convertito ad `int` (a meno che l'operatore non sia `++` o `--`, nel qual caso non avviene nessuna conversione).
  - Altrimenti, nessuna conversione.
- Le regole per operatori binari:
  - Se uno degli operandi è un `double`, allora l'altro operando è convertito a `double`.
  - Se uno degli operandi è un `float`, allora l'altro operando è convertito a `float`.
  - Se uno degli operandi è un `long`, allora l'altro operando è convertito a `long`.
  - Altrimenti, entrambi gli operandi sono convertiti ad `int`.
- Che succede nella valutazione delle espressioni seguenti:

```
short s=9;
int i=10;
float f=11.1f;
double d=12.2;
if (-s * i >= f / d) {}
```


Enunciati branch

`if`, `else`

```
if (<boolean expression>) {
  <statement>*
}

if (<boolean expression>) {
  <statement>*
} else if (<boolean expression>) {
  <statement>*
} else {
  <statement>*
}
```

Attenzione all'espressione condizionale: DEVE ESSERE UNA ESPRESSIONE BOOLEANA, NON UN INTERO COME IN C/C++.

switch

```
switch (<expr>) {
  case <constant1>:
    <statement>*
    break;
  case <constant2>:
    <statement>*
    break;
  default:
    <statement>*
    break;
}
```

dove:

- <expr> deve essere compatibile per assegnazione con `int` (`byte`, `short`, `char` sono promossi automaticamente);
- l'etichetta opzionale <default> è usata per specificare il segmento di codice da eseguire quando il valore di <expr> non corrisponde a nessuno dei valori delle stanze `case`;
- se non è presente l'enunciato opzionale `break`, l'esecuzione continua nella stanza `case` successiva.

Enunciati loop

for

```
for (<init_expr>; <bool_expr>; <alt_expr>) {
  <statement>*
}
```

Esempio:

```
for (int i=0; i<10; i++) {
  System.out.println("Sono nel loop");
}
System.out.println("Fine");
```


while

```
while (<bool_expr>) {
  <statement>*
}
```

Esempio:

```
int i=0;
while (i<10) {
  Sytem.out.println("Sono nel loop");
  i++;
}
System.out.println("Fine");
```

do/while

```
do {
  <statement>*
} while (<bool_expr>);
```

Esempio:

```
int i=0;
do {
  Sytem.out.println("Sono nel loop");
  i++;
} while (i<10);
System.out.println("Fine");
```


Controlli speciali

- `break [label];`
  usata per uscire in modo prematuro da un enunciato `switch`, da enunciati di loop o da blocchi etichettati.
- `continue [label];`
  usata per saltare direttamente alla fine del corpo di un loop.
- `label: <statement>`
  usata per identificare un qualunque enunciato verso il quale può essere trasferito il controllo.

Esercizi 26 / 31

Esercizi

1. Estensione dell'esercizio della lezione precedente (pacchetto `banca`).

Questionario 27 / 31

D 1

In questa successione di enunciati, quale linea non compila?

A. `byte b = 5;`
B. `char c = '5';`
C. `short s = 55;`
D. `int i = 555;`
E. `float f = 555.5f;`
F. `b = s;`
G. `i = c;`
H. `if (f > b)`
I. `  f = i;`

LP1 – Lezione 9 28 / 31

D 2

Il codice seguente viene compilato correttamente?

```
byte b = 2;
byte b1 = 3;
b = b * b1;
```

A. Si
B. No

LP1 – Lezione 9 29 / 31

D 3

Nel codice seguente, quali sono i possibili tipi per la variabile `result`?

```
byte b = 11;
short s = 13;
result = b * ++s;
```

A. `byte`, `short`, `int`, `long`, `float`, `double`
B. `boolean`, `byte`, `short`, `char`, `int`, `long`, `float`, `double`
C. `byte`, `short`, `char`, `int`, `long`, `float`, `double`
D. `byte`, `short`, `char`
E. `int`, `long`, `float`, `double`

LP1 – Lezione 9 30 / 31

D 4

```
 1. class Cruncher {
 2.   void crunch(int i) {
 3.     System.out.println("int version");
 4.   }
 5.   void crunch(String s) {
 6.     System.out.println("String version");
 7.   }
 8.
 9.   public static void main(String args[]) {
10.     Cruncher crun = new Cruncher();
11.     char ch = 'p';
12.     crun.crunch(ch);
13.   }
14. }
```

Quale delle seguenti affermazioni è vera?

A. La linea 5 non compila, poiché i metodi `void` non possono essere sovrapposti.
B. La linea 12 non compila, poiché nessuna versione di `crunch()` ha un argomento `char`.
C. Il codice compila correttamente, ma viene lanciata una eccezione alla linea 12.
D. Il codice compila correttamente e viene prodotto il seguente output: `int version`
E. Il codice compila correttamente e viene prodotto il seguente output: `String version`