Lab. Programmazione (CdL Informatica) & Informatica (CdL Matematica) a.a. 2022-23

Monica Nesi

Università degli Studi dell'Aquila

25 Ottobre 2022

Costrutto iterativo in Java

Iterazione o ciclo (loop):
questo costrutto permette di ripetere (*iterare*) l'esecuzione di un'istruzione (in generale, una sequenza di istruzioni) fintantoché una data condizione logica è vera. Non appena tale condizione diventa falsa, l'iterazione termina e *si esce dal ciclo*.

Per questo costrutto esistono più comandi in Java.

Il primo comando che vediamo è l'istruzione `while`, la cui sintassi può essere data come segue:

while (<*BEspr*>) <*Istr*>

dove <*BEspr*> denota l'insieme delle espressioni booleane e <*Istr*> denota l'insieme delle istruzioni (comandi) del linguaggio.

N.B. Come per il comando condizionale, l'espressione booleana deve essere racchiusa tra parentesi tonde.

Comando while: semantica

La semantica del comando `while` può essere data tramite un diagramma di flusso (coincide con quella del costrutto iterativo utilizzato nelle prime lezioni).

Dato un generico comando

$$\text{while } (E) \ C$$

la semantica è la seguente:

1. dapprima si valuta l'espressione booleana E, detta la *guardia* del comando iterativo;
2. se E è vera, allora viene eseguito il comando C e poi *si torna in ciclo* (ovvero, si ricomincia con la valutazione del comando iterativo);

altrimenti (ovvero, E è falsa) si *esce dal ciclo* terminando l'esecuzione del comando `while`.

Comando while: corpo

Il comando C che costituisce il *corpo* (*body*) del ciclo può essere una *singola* istruzione oppure una *sequenza di istruzioni* racchiusa in un *blocco*, ovvero tra una coppia di parentesi graffe.

In modo simile al comando condizionale, se nel corpo di un comando iterativo esistono più istruzioni *non* racchiuse in un blocco, l'interpretazione di Java è di considerare come corpo del ciclo *solo la prima istruzione* della sequenza.

Dato il comando

$$\begin{array}{l} \text{while } (E) \\ \quad C1 \\ C2 \end{array}$$

solo $C1$ viene considerato come corpo del ciclo e quindi verrà iterata la sua esecuzione fintantoché E è vera. L'istruzione $C2$ viene eseguita una volta terminata l'esecuzione del comando `while`.

Comando while: corpo (cont.)

Se si vuole iterare l'esecuzione di $C1$ seguita da $C2$, occorre introdurre un blocco:

```
while (E) {
  C1
  C2
}
```

Il corpo di un comando `while` può essere eseguito:

- un numero *finito* di volte (in particolare, 0 volte se l'espressione booleana E è subito falsa);

- un numero *infinito* di volte (ciclo infinito), se l'espressione booleana E non diventa mai falsa.

Comando while: esempi

```
int x=0;
while (x<7) {
  System.out.println(x);
  x=x+2;
}
```

```
int i=1;
while (i<=3) {
  System.out.println(i*i);
  i++;
}
```

Comando for

Il secondo comando iterativo che vediamo è l'istruzione `for`, la cui sintassi può essere data come segue:

for (<*Iniz*>; <*BEspr*>; <*Incr*>) <*Istr*>

dove:

- <*Iniz*> indica l'insieme di inizializzazioni di variabili (con eventuale dichiarazione del tipo);
- <*BEspr*> denota l'insieme delle espressioni booleane;
- <*Incr*> denota l'insieme di comandi di incremento/decremento che è possibile applicare sulle variabili che regolano il ciclo;
- <*Istr*> denota l'insieme delle istruzioni (comandi) del linguaggio.

Comando for: semantica

Dato un generico comando

$$\texttt{for}\ \ (\mathit{Iniz}; E; \mathit{Incr})\ \ C$$

la semantica è la seguente:

1. si esegue l'inizializzazione *Iniz*;
2. si valuta l'espressione booleana E (*guardia* del `for`);
3. se E è vera, allora viene eseguito il comando C, poi si esegue la modifica *Incr* e quindi *si torna in ciclo* a valutare la guardia E; altrimenti (ovvero, E è falsa) si *esce dal ciclo* terminando l'esecuzione del comando `for`.

La semantica del comando `for` può essere data tramite un diagramma di flusso.

Comando for: esempi

```
for (int i=0; i<6; i++)
  System.out.println(i);
```

```
for (int j=4; j>=0; j--)
  System.out.println(j+j);
```

```
int x;
for (x=1; x<10; x=x+2)
  System.out.println(x);
System.out.println(x);
```

Comando do-while

Il terzo (ed ultimo) comando iterativo che vediamo è l'istruzione `do-while`, la cui sintassi può essere data come segue:

do <*Istr*> while (<*BEspr*>);

dove <*BEspr*> denota l'insieme delle espressioni booleane e <*Istr*> denota l'insieme delle istruzioni (comandi) del linguaggio.

N.B. Di nuovo, l'espressione booleana deve essere racchiusa tra parentesi tonde.

Comando do-while: semantica

Dato un generico comando

$$\text{do } C \text{ while } (E);$$

la semantica è la seguente:

1. si esegue il comando C (che pertanto viene eseguito *almeno una volta*);
2. si valuta l'espressione booleana E:

- se E è vera, allora *si torna in ciclo* (ovvero, si ricomincia con la valutazione del comando iterativo);
- altrimenti (ovvero, E è falsa) si *esce dal ciclo* terminando l'esecuzione del comando `do-while`.

La semantica del comando `do-while` può essere data tramite un diagramma di flusso.

Comando do-while: esempio

```
int x=1;
do {
   System.out.println(x);
   x=x+2;
}
while (x<10);
System.out.println(x);
```

Blocchi e variabili locali in un blocco

Anche per i comandi `for` e `do-while`, abbiamo che:

i) il corpo *C* del ciclo può essere una singola istruzione oppure una sequenza di istruzioni racchiusa in un blocco;

ii) se nel corpo esistono più istruzioni *non* racchiuse in un blocco, l'interpretazione di Java è di considerare come corpo del ciclo *solo la prima istruzione* della sequenza.

Come nel caso del comando condizionale, una volta aperto un blocco in un comando iterativo, è possibile dichiarare identificatori *locali* a tale blocco.

Valgono gli stessi discorsi che abbiamo fatto per i blocchi in un comando condizionale, la dichiarazione di nuovi identificatori e le regole di visibilità degli identificatori (*scoping rules*).

Confronto tra i comandi while e for

I tre comandi iterativi che abbiamo visto possono essere usati indifferentemente in Java. In questo corso verranno utilizzati soprattutto i comandi `while` e `for`.

Vediamo quindi un confronto tra questi due comandi.

Un generico comando `for`

for $(Iniz; E; Incr)$
C

è *equivalente* al seguente comando `while`

$Iniz;$
while (E) {
C
$Incr;$
}

Confronto tra i comandi while e for: esempio

I due comandi sono equivalenti nel senso che, a partire da uno stesso stato della macchina, lo stato risultante dall'esecuzione dei due comandi è lo stesso. I due comandi iterativi

```
for (int i=0; i<5; i++)
   System.out.println(i);
```

e

```
int i=0;
while (i<5) {
   System.out.println(i);
   i++;
}
```

sono *equivalenti* in quanto, a partire da uno stesso stato della memoria, apportano le stesse modifiche sulle variabili e danno luogo alla stampa degli stessi valori.

Confronto tra i comandi while e for: esempio (cont.)

Non sono però equivalenti per quanto riguarda la *visibilità* delle variabili. Infatti, nel primo frammento di codice la variabile `i` è visibile solo nel `for`, mentre nel secondo frammento `i` è visibile anche al di fuori del `while`.

I due frammenti seguenti sono equivalenti anche relativamente alla visibilità delle variabili:

```
int i=0;     \\ oppure int i;
for (i=0; i<5; i++)
  System.out.println(i);
```

e

```
int i=0;
while (i<5) {
  System.out.println(i);
  i++;
}
```

Esempio del fattoriale

Scrivere un programma in Java che calcola il fattoriale di un numero $n \in \mathbb{N}$.

Si ricorda la definizione del fattoriale:

$0! = 1$
$n! = n \times (n-1) \times \ldots \times 2 \times 1$ per $n \geq 1$.

Se $n < 0$, il programma stampa un messaggio di errore.

Occorre
- un ciclo per calcolare il prodotto dei primi n numeri ($n \geq 1$);
- una variabile in cui accumulare tale prodotto, inizializzata a 1.

Nota: 1 è anche il valore del fattoriale nei casi $n = 0$ ed $n = 1$.

Un programma per il fattoriale

```
class Fattoriale {
 public static void main (String[] args) {
  int n = Integer.parseInt(args[0]);
  if (n<0) {
    System.out.println("Numero␣negativo");
  }
  else {
    int f=1;
    for (int i=1; i<=n; i++) {
      f=f*i;
    }
    System.out.println("Il␣fattoriale␣di␣"
    +n+"␣e'␣"+f+".");
  }
 }
}
```

Un programma per il fattoriale (cont.)

Dopo aver compilato il programma, l'esecuzione di

```
java Fattoriale 5
```

dà luogo alla stampa:

```
Il fattoriale di 5 e' 120.
```

Nota: i casi $n = 0$ ed $n = 1$ possono essere trattati separatamente prima del `for` con un comando condizionale nel ramo `else`.

Nella soluzione proposta questi casi sono gestiti come *casi particolari* del comando `for`.

Cicli annidati: primo esempio

I comandi iterativi possono contenere nel loro corpo altri comandi iterativi.

Sia dato il seguente programma:

```
class Ciclo1 {
 public static void main (String[] args) {
  for (int i=0; i<2; i++) {
    System.out.println("i␣=␣"+i);
    for (int j=3; j>0; j--) {
      System.out.println("j␣=␣"+j);
    }
  }
 }
}
```

Dire che cosa viene stampato in uscita.

Cicli annidati: output del primo esempio

L'output del programma precedente è il seguente:

```
i = 0
j = 3
j = 2
j = 1
i = 1
j = 3
j = 2
j = 1
```

Una volta fissato il valore della variabile `i`, il ciclo `for` interno viene eseguito 3 volte.

Cicli annidati: primo esempio con un while interno

Se il `for` interno viene sostituito da un comando `while`,
il main diventa:

```
for (int i=0; i<2; i++) {
   System.out.println("i␣=␣"+i);
   int j=3;
   while (j>0) {
      System.out.println("j␣=␣"+j);
      j--;
   }
}
```

Frammento di codice *equivalente* al precedente
(verificare che si ha lo stesso output).

Cicli annidati: secondo esempio

Sia dato il seguente programma:

```
class Ciclo2 {
 public static void main (String[] args) {
  int i=7;
  while (i>0) {
    for (int j=1; j<i; j=j+2) {
      System.out.print(i*j + "␣");
    }
    System.out.println();
    i--;
  }
  System.out.println(i);
 }
}
```

Dire che cosa viene stampato in uscita.

Cicli annidati: output del secondo esempio

L'output del programma precedente è il seguente:

```
7  21  35
6  18  30
5  15
4  12
3
2

0
```

Nota: la "riga" di stampa vuota corrisponde all'esecuzione per `i=1` e `j=1`, per cui `j<i` è falso e la stampa nel `for` interno non viene eseguita.

Il valore finale 0 è stampato dall'ultimo comando di stampa all'uscita dal comando `while`.

Cicli annidati: terzo esempio

Sia dato il seguente programma:

```
class Stars {
 public static void main (String[] args) {
  int n = Integer.parseInt(args[0]);
  for (int i=0; i<n; i++) {
    for (int j=0; j<i+1; j++)
      System.out.print("*");
    System.out.println();
  }
 }
}
```

Dire che cosa viene stampato in uscita.

Cicli annidati: output del terzo esempio

Nel caso $n = 6$ l'output del programma precedente è il seguente:

```
*
**
***
****
*****
******
```

Verificare che cosa viene stampato se l'intestazione del `for` interno viene sostituito da
`for (int j=0; j<n-i; j++)`.