Steve Blackbird (TardoFreak)

PROGRAMMAZIONE (3): I COSTRUTTI DI BASE

21 September 2016

Si può scrivere qualsiasi tipo di algoritmo utilizzando tre costrutti di base:

- **assegnamento** per assegnare ad una variabile un valore o il risultato di un'espressione.
- **controllo del flusso** per decidere, in base ad una **condizione** se fare una cosa o farne un'altra.
- **controllo dei cicli** per fare eseguire più volte una porzione di programma in base ad una condizione

Assegnamento

Come accennato prima l'assegnamento serve per assegnare un valore ad una variabile. Per fare questo si usa un **operatore** chiamato **operatore di assegnamento** che è il carattere =, cioè il carattere di "uguale". Se vogliamo assegnare il valore 123 alla nostra variabile totaleProdotti dobbiamo scrivere

```
totaleProdotti = 123;
```

Il numero 123 è chiamato **costante numerica**. Se invece vogliamo assegnare alla variabile "totaleProdotti" una espressione che coinvolge altre variabili, per esempio vogliamo che "totaleProdotti" contenga il risultato di una moltiplicazione fra le due variabili "numeroScatole" e "prodottiPerScatola" dobbiamo scrivere

```
totaleProdotti = numeroScatole * prodottiPerScatola;
```

che significa dire al calcolatore "esegui la moltiplicazione fra *numeroScatole* e *prodottiPerScatola* e memorizza il risultato nella variabile *totaleProdotti*. Il carattere asterisco * indica l'operazione di moltiplicazione, ed è un **operatore aritmetico**.
Oltre agli operatori aritmetici esistono anche gli **operatori logici** utilizzati nelle **espressioni logiche**, cioè quel tipo di operazioni che danno come risultato un valore che può essere **vero** o **falso**. Ad esempio la comparazione fra due numeri per sapere se uno è più grande di un altro, per ipotesi a > b dà come risultato **true** (vero) se il valore contenuto nella variabile "a" è maggiore di quello contenuto nella variabile "b". Se i due valori sono uguali o "b" è maggiore di "a" il risultato sarà **false** (falso). Prendiamo la seguente porzione di codice

```
int a;
int b;
boolean maggiore;
```

```
a = 15;
b = 20;
maggiore = a > b;
```

ed analizziamola nel dettaglio riga per riga come se parlassimo in modo discorsivo al calcolatore

1. Userò una variabile di tipo int che chiamo "a".
2. Userò una variabile di tipo int che chiamo "b".
3. Userò una variabile di tipo **boolean** che chiamo "maggiore". Le variabili di questo tipo sono chiamate **variabili logiche** e possono contenere solo i valori true e false.
4. Assegna alla variabile "a" il valore 15
5. Assegna alla variabile "b" il valore 20
6. Compara i valori delle variabili "a" e "b" per sapere se "a" è maggiore di "b" e assegna il risultato (in questo caso è false perché "a" non è maggiore di "b") alla variabile "maggiore".

Nota: in Java nella dichiarazione di variabile è anche possibile assegnare il risultato di una espressione.
Nota per il C: in C non esistono le variabili di tipo boolean. Il valore di una qualsiasi variabile intera usata in una espressione logica viene considerato vero se diverso da 0, false se uguale a 0.

Controllo del flusso

Il controllo del flusso, o **esecuzione condizionata**, serve per scegliere se eseguire un'istruzione o un blocco e si ottiene mediante l'uso della parola chiave **if**. La sintassi del costrutto "if" è:
if (<*espressione logica*> **)** <*istruzione*>
Un esempio chiarirà meglio il concetto

```
if (a == b) a = 0;
if (a > b) {
  int temp = a;
  a = b;
  b = temp;
}
```

Nella prima linea stiamo dicendo al calcolatore "se il valore della variabile *a* è uguale al valore della variabile *b*, allora assegna il valore 0 alla variabile *a*". L'operatore logico che verifica l'uguaglianza di due valori è il doppio uguale ==. I due caratteri di uguale devono essere uno accanto all'altro, senza spazi, altrimenti il compilatore li tradurrà come due operatori di assegnamento generando un errore di sintassi. Notare che l'istruzione di assegnamento, essendo una singola istruzione, termina con il punto e virgola.
Nella seconda linea stiamo dicendo al calcolatore "se il valore della variabile *a* è maggiore del valore della variabile *b* esegui le istruzioni comprese tra le due parentesi graffe (cioè il blocco). Quindi le tre istruzioni contenute nel blocco verranno eseguite, altrimenti saranno "saltate" e non

verranno eseguite.
Quest'ultimo è un esempio della potenza della programmazione strutturata perché all'interno di un blocco potremmo trovarci un'altra istruzione di if per eseguire o meno un altro eventuale blocco, o addirittura un intero programma! Abbiamo visto che la parola chiave "if" ci permette di fare eseguire o no un blocco o un'istruzione se la condizione racchiusa fra le parentesi tonde dà come risultato true, allora una domanda sorge spontanea: è possibile dirgli anche cosa fare nel caso che la condizione dia come risultato false? La risposta è sì, e lo si implementa mediante la parola chiave **else**. Vediamo un esempio completo di tutto quel che serve per poterlo vedere funzionare. In pratica prendiamo il primo programma in Java e modifichiamolo come segue

```
package progjava;

public class ProgJava {

  public static void main(String[] args) {
    int a = 9;
    int b = 5;
    if (a > b) {
      int temp = a;
      a = b;
      b = temp;
      System.out.println("Scambio effettuato");
    }
    else System.out.println("Nessuno scambio");
  }
}
```

Le prime due istruzioni sono due dichiarazioni di variabile con assegnamento. Subito prima di incontrare la "if" le variabili "a" e "b" conterranno rispettivamente i valori 9 e 5. Quindi essendo "a" è maggiore di "b" e verranno eseguite le istruzioni all'interno del blocco. Queste tre istruzioni scambiano fra di loro i valori contenuti in "a" e "b". Per prima cosa viene dichiarata una variabile intera "temp" a cui verrà assegnato il valore della variabile "a", alla variabile "a" viene assegnato il valore di "b", e per finire a "b" viene assegnato il valore di "temp". L'istruzione successiva è una chiamata al metodo System.out.println per visualizzare sul terminale la scritta "Scambio effettuato". Facendo partire il programma otterremo questo output

```
run:
Scambio effettuato
BUILD SUCCESSFUL (total time: 0 seconds)
```

Ora modifichiamo il programma assegnando, invece che 9 il valore 2 ad a

```
  int a = 2;
```

Ora "a" **non è** maggiore di "b", quindi non verrà eseguito il blocco ma l'istruzione che ci dice che lo scambio non è stato effettuato. Facendo partire il programma otteniamo questo output

```
run:
Nessuno scambio
BUILD SUCCESSFUL (total time: 0 seconds)
```

Quindi il costrutto if ... else è
if (<*espressione logica*> **)** <*istruzione*> **else** <*istruzione*>
Dove la parte "else" è facoltativa.

Passiamo ora ad un esempio pratico dell'utilità di questo costrutto. Supponiamo di avere tre variabili "a","b", e "c" che contengono tre valori assegnati direttamente. Dobbiamo scrivere un programma che trovi il valore più alto fra i tre e lo scambi con il valore di "a". In pratica facciamo **emergere il massimo** fra "a","b" e "c" e facciamo in modo che il massimo venga posto in "a".
Supponiamo che i valori siano
a = 2, b = 6, c = 17
al termine dell'esecuzione dobbiamo trovarci per esempio in questa situazione a = 17, b = 6, c = 2
L'importante, alla fine, è che i tre valori siano tutti e tre presenti ma il valore massimo sia nella variabile "a". Un procedimento che fa questo lavoro può essere questo:

1. confronto "b" con "c"
2. se "c" è maggiore di "b" scambio i valori in modo da ritrovami in "b" il massimo fra "b" e "c".
3. confronto "a" con "b"
4. se "b" è maggiore di "a" scambio i valori in modo da ritrovami in "a" il massimo fra "a" e "b", che risulta così essere il massimo fra i tre.

Tradotto in Java

```
package progjava;

public class ProgJava {

  public static void main(String[] args) {
    int a = 2;
    int b = 6;
    int c = 17;
    if (c > b) {
      int temp = b;
      b = c;
      c = temp;
    }
    if (b > a) {
```

```
      int temp = a;
      a = b;
      b = temp;
    }
    System.out.print("Il valore della variabile a vale ");
    System.out.println(a);
  }
}
```

L'output di questo programma è

```
run:
Il valore della variabile a vale 17
BUILD SUCCESSFUL (total time: 1 second)
```

Nota: la differenza fra System.out.print e System.out.println è che la prima scrive il testo (o il numero) ma non va a capo, la seconda stampa e poi va a capo. Il risultato è quello che si vede.
Per completezza, con poche altre istruzioni, possiamo scrivere un programma di **ordinamento**. In effetti se abbiamo già fatto emergere il massimo è sufficiente confrontare nuovamente "b" con "c" ed eventualmente scambiarli fra loro per fare in modo che i tre valori siano ordinati in modo decrescente. Quindi aggiungiamo questo confronto e scriviamo anche le istruzioni per visualizzare i valori di "b" e "c".

```
package progjava;

public class ProgJava {

  public static void main(String[] args) {
    int a = 2;
    int b = 6;
    int c = 17;
    if (c > b) {
      int temp = b;
      b = c;
      c = temp;
    }
    if (b > a) {
      int temp = a;
      a = b;
      b = temp;
    }
    if (c > b) {
      int temp = b;
      b = c;
```

```
      c = temp;
    }
    System.out.print("Il valore della variabile a vale ");
    System.out.println(a);
    System.out.print("Il valore della variabile b vale ");
    System.out.println(b);
    System.out.print("Il valore della variabile c vale ");
    System.out.println(c);
  }
}
```

Che dà come output

```
run:
Il valore della variabile a vale 17
Il valore della variabile b vale 6
Il valore della variabile c vale 2
BUILD SUCCESSFUL (total time: 0 seconds)
```

Controllo del ciclo

Questo costrutto serve per far ripetere più volte una parte di programma in base ad una condizione. Serve per implementare conteggi per esempio, oppure per continuare a fare un qualcosa fino a quando una condizione è verificata. In un programma che gira su micro si potrebbe far ripetere una serie di operazioni per tutto il tempo in cui un pulsante è premuto. Ci sono diversi modi ed istruzioni in Java per il controllo del ciclo ma, essendo questo un tutorial sui fondamenti della programmazione e non un manuale di riferimento, ne adotterò (per ora) uno solo, quello che si implementa con la parola chiave **while**. Un semplice esempio può essere un programma che conta alla rovescia fino a quando il valore di una variabile raggiunge lo zero, ma lo zero non lo facciamo vedere. Supponiamo di avere una variabile che chiamiamo "i" e che valga 10. Lo scopo di questo programma è stampare 10 numeri (da 10 a 1). Se i vale 25 stamperà al rovescio i numeri da 25 ad 1. Possiamo descrivere questo algoritmo in modo discorsivo con questi passi

1. dichiaro la variabile intera "i" e le assegno un numero
2. fino a quando il valore di "i" è maggiore di zero eseguo queste istruzioni
   - Stampo il valore di "i"
   - Decremento il valore di "i"

Una cosa molto semplice che scritta in Java diventa

```
package progjava;
```

```
public class ProgJava {

  public static void main(String[] args) {
    int i = 10;
    while(i > 0) {
      System.out.println(i);
      i = i - 1;
    }
  }
}
```

L'output del programma

```
run:
10
9
8
7
6
5
4
3
2
1
BUILD SUCCESSFUL (total time: 0 seconds)
```

Il costrutto while è:
while(<*espressione logica*> **)** <*istruzione*>
Quando il calcolatore incontra un costrutto "while" prima verifica che l'espressione logica dia come risultato true e, in caso affermativo, esegue l'istruzione. Al termine dell'istruzione il calcolatore non prosegue con le istruzioni successive ma ritorna al while e ripete quanto descritto da capo.
L' espressione logica richiesta dal costrutto è chiamata **condizione di ingresso e di permanenza**. Il calcolatore rimane, per così dire, "intrappolato" nell'esecuzione ripetuta dell'istruzione fino a quando la condizione di permanenza è true.
E' un costrutto potente ma insidioso. Se la condizione di permanenza non diventa mai false il calcolatore eseguirà il ciclo all'infinito! E' quindi opportuno prestare la massima attenzione a fare in modo che l'istruzione, che poi solitamente è un blocco, cambi in qualche modo il risultato della condizione di permanenza per permettere la cosiddetta **uscita** dal ciclo, o **terminazione**.
Nell'esempio riportato la condizione di permanenza è che "i" sia maggiore di zero. C'è qualcosa all'interno del blocco che fa in modo che il valore di "i" diventi zero? Si, c'è, ed è l'istruzione che decrementa "i" (i = i - 1;) quindi prima o poi i varrà zero ed il ciclo terminerà.
Ma cosa succede se assegniamo ad "i" il valore 0 o addirittura un valore negativo? Succede che **il blocco non viene eseguito nemmeno una volta**.

I cicli, come i blocchi, possono essere **annidati**, possono esserci quindi cicli all'interno di altri cicli. Nell'esempio precedente abbiamo stampato i numeri da 10 ad 1, in questo esempio stampiamo invece la tavola pitagorica con i numeri da 1 a 10. Prima di scrivere il sorgente analizziamo il problema.
Cosa deve innanzitutto fare questo programma? Deve innanzi tutto stampare il prodotto di una riga che chiameremo "r" con una colonna che chiameremo "c". Quindi possiamo già scrivere su un foglio di carta l'istruzione per stampare questo valore

```
System.out.print(r * c);
```

Il metodo "System.out.print" accetta come parametri attuali stringhe, numeri o un qualsiasi cosa che abbia un valore, espressioni comprese. "r * c" è un'espressione e quindi la scriviamo direttamente fra le parentesi tonde.
Proseguendo potremmo pensare di stampare (cioè di visualizzare) una riga, dovremmo cioè scrivere un ciclo che ci stampi tutti i valori di una riga. Per fare questo un ciclo che faccia variare il valore della colonna da 1 a 10. All'interno di questo ciclo c'è l'istruzione per la stampa. Ma siccome vogliamo che i valori siano separati fra loro almeno di uno spazio, dobbiamo stampare anche quello. Per ultimo vogliamo anche andare a capo alla fine della riga, cioè dopo essere usciti dal ciclo.

```
int c = 1;
while(c < 11) {
  System.out.print(r * c);
  System.out.print(" ");
  c = c + 1;
}
System.out.println();
```

1. Per prima cosa dichiariamo la nostra variabile "c" e le assegniamo il valore iniziale, cioè 1.
2. La seconda linea dice "fino a quando il valore di *c* sarà minore di 11 fai quello che c'è scritto nel blocco".
3. All'interno del blocco prima stampiamo il valore senza andare a capo, poi uno spazio sempre senza andare a capo. L'ultima istruzione incrementa di 1 il valore di "c" per stampare la colonna successiva.
4. Usciti dal ciclo la chiamata al metodo "System.out.println()" senza argomenti otterrà l'effetto di non stampare niente e andare a capo all'inizio della riga successiva.

Il passo successivo sarebbe stampare le righe da 1 a 10. La parte che stampa una riga ce l'abbiamo già quindi è sufficiente porla all'interno di un ciclo che scandisca le righe da 1 a 10 ed il gioco è fatto. Questo è il programma completo.

```
package progjava;

public class ProgJava {
```

```
  public static void main(String[] args) {
    int r = 1;
    while(r < 11) {
      int c = 1;
      while(c < 11) {
        System.out.print(r * c);
        System.out.print(" ");
        c = c + 1;
      }
      System.out.println();
      r = r + 1;
    }
  }
}
```

La prima istruzione del corpo del metodo main è la dichiarazione della nostra variabile "r" a cui assegniamo il valore di partenza 1. Dopo troviamo il ciclo che scandisce le righe "r", e che contiene al suo interno il ciclo che stampa le righe e va a capo al termine della stampa, ed infine una istruzione per incrementare il valore della riga. Ragionato in questo modo non esiste nessun motivo valido per cui questo programma non debba essere compilato senza errori e che non faccia esattamente quello che deve fare. Infatti si compila al primo colpo e ci dà questo come output

```
run:
1 2 3 4 5 6 7 8 9 10
2 4 6 8 10 12 14 16 18 20
3 6 9 12 15 18 21 24 27 30
4 8 12 16 20 24 28 32 36 40
5 10 15 20 25 30 35 40 45 50
6 12 18 24 30 36 42 48 54 60
7 14 21 28 35 42 49 56 63 70
8 16 24 32 40 48 56 64 72 80
9 18 27 36 45 54 63 72 81 90
10 20 30 40 50 60 70 80 90 100
BUILD SUCCESSFUL (total time: 1 seconds)
```

Nota: nel progettare questo programma verrebbe la tentazione di calcolare il prodotto fra riga e colonna e memorizzarlo in una variabile temporanea. Secondo il mio punto di vista è un errore perché questa variabile temporanea "male non gli farebbe" ma non servirebbe a niente. E quello che non serve non si scrive! Io utilizzo una regola: se si usa una variabile per memorizzare un valore intermedio, vuol dire che questa variabile è di sicuro utilizzata altrove. In caso contrario scrivo l'espressione direttamente dove deve essere.

Passiamo ora ad un altro esempio dove useremo sia il controllo del flusso che il controllo del ciclo: un programma che moltiplichi fra di loro due numeri interi e ne visualizzi il risultato, ma senza utilizzare l'operatore della moltiplicazione, è ammesso solo l'uso dell'addizione (o sottrazione).
Un modo per risolvere questo problema è quello di ottenere il risultato della moltiplicazione mediante addizioni successive. In effetti scrivere 3 * 4 significa dire "3 volte 4", quindi 3 + 3 + 3 + 3. Si potrebbe scrivere un ciclo che sommi ad una variabile che conterrà il risultato il valore del moltiplicando tante volte quanto è il valore del moltiplicatore. Se chiamiamo "m" il moltiplicando ed "n" il moltiplicatore ed "r" il risultato l' operazione di base sarebbe questa

```
r = r + m;
```

Ma siccome dobbiamo farla eseguire n volte inseriamo Questa istruzione in un ciclo come quello di prima

```
while(n > 0) {
  r = r + m;
  n = n - 1;
}
```

Sembrerebbe tutto a posto ma non abbiamo tenuto conto dei segni perché "m" o "n" o entrambi potrebbero avere valore negativo. Nel caso in cui m sia negativo non ci sono problemi perché la somma sarà negativa, il problema sorge quando "n" è negativo perché non sarebbe mai maggiore di zero e quindi il ciclo non sarebbe eseguito nemmeno una volta. La soluzione è rendere "n" positivo controllandone il segno e memorizzandolo da qualche parte (insomma dobbiamo ricordarci se "n" è negativo o no). Se è negativo gli cambiamo segno e "ce lo ricordiamo" usando una variabile boolean che chiamiamo "segnoMeno" che vale true se "n" è negativo. Se "n" è negativo dovremo poi cambiare segno al risultato. Il risultato è questo programma

```
package progjava;

public class ProgJava {

  public static void main(String[] args) {
    int m = 12;  // moltiplicando
    int n = -5;  // moltiplicatore
    int r = 0;   // risultato
    boolean segnoMeno = false;
    if (n < 0) {
      segnoMeno = true;
      n = -n;
    }
    while(n > 0) {
      r = r + m;
```

```
      n = n - 1;
    }
    if (segnoMeno == true) r = -r;
    System.out.println(r);
  }
}
```

1. Le prime quattro linee sono le dichiarazioni delle variabili ed il loro assegnamento iniziale.
2. La linea seguente (quella con il primo if) fa un test: se il valore di "n" è minore di zero (quindi ha segno -) esegue il contenuto del blocco. Più precisamente assegna il valore true a "segnoMeno" e cambia segno ad "n". Il segno meno - utilizzato in questo modo viene chiamato **operatore unario**. Se "n" è positivo il blocco non viene eseguito e "segnoMeno" avrà il valore di false.
3. Arriviamo quindi al ciclo di "while" che abbiamo già visto prima. Il valore di "m" viene sommato ad "r" (che inizialmente vale 0) per "n" volte.
4. La linea seguente serve per l'eventuale cambio di segno. Se "n" è negativo (cioè "segnoMeno" è true) cambiamo segno al risultato.
5. L'ultima istruzione che viene eseguita è quella di visualizzare il valore del risultato.

L' output di questo programma è il seguente

```
run:
-60
BUILD SUCCESSFUL (total time: 1 second)
```

Considerazioni finali

Con queste conoscenze si può progettare qualsiasi algoritmo. Il prossimo passo sarà quello di introdurre alcuni concetti importanti come i metodi, e strutture dati per poter utilizzare questi costrutti con problemi di complessità via via crescente, ed altro ancora.
Curiosità, ricerca ed approfondimento sono le parole chiave per sfruttare al meglio la sintesi scritta fino ad ora. E' necessario uno sforzo personale per imparare, ad esempio, altri costrutti per il controllo dei cicli o per sperimentare i vari tipi di variabili , provare la visibilità ...
I manuali di riferimento, o libri/corsi che sono anche manuali di riferimento sarebbero da leggere; magari lasciati sul mobiletto nel bagno, da leggere nei momenti di ... relax.

Nota Immagine di copertina tratta dal sito www.oldwildweb.com

Estratto da "http://www.electroyou.it/mediawiki/index.php?title=UsersPages:Tardofreak:programmazione-2"