Istruzioni di ripetizione in Java

Programmazione
Corso di laurea in Comunicazione digitale

Istruzioni di ripetizione

- *Le istruzioni di ripetizione* consentono di eseguire la stessa istruzione molte volte
- Si chiamano anche *cicli*
- Come le istruzioni condizionali, i cicli sono controllati da espressioni booleane
- In Java i cicli: `while`, `do-while` e `for`
- I diversi tipi di ciclo sono approriati in situazioni diverse

Istruzione `while`

- Sintassi dell'istruzione *while*:

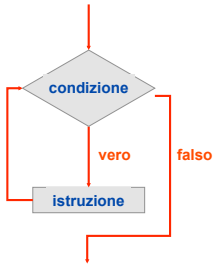
```
while ( condizione )
    istruzione;
```

`while` è parola riservata

Se *condizione* è vera, viene eseguita *istruzione*
Quindi si valuta ancora *condizione*

istruzione viene eseguita ripetutamente fino a che la *condizione* non diventa falsa

Semantica del ciclo `while`



Istruzione `while`

- Se la condizione di un ciclo `while` è inizialmente falsa, il ciclo non viene mai eseguito
- Quindi un ciclo `while` può essere eseguito 0 o più volte
- Counter.java

  Average.java e Average2.java per ovviare al possibile problema dell'overflow

  WinPercentage.java

Istruzione `do-while`

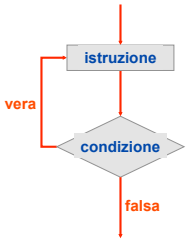
- La sintassi dell'istruzione *do*:

```
do
{
    istruzione;
}
while ( condizione )
```

Le parole riservate `do` `while`

L'istruzione viene inizialmente eseguita una volta, poi la condizione è valutata e l'istruzione viene ripetutamente eseguita fino a che la condizione non diventa falsa

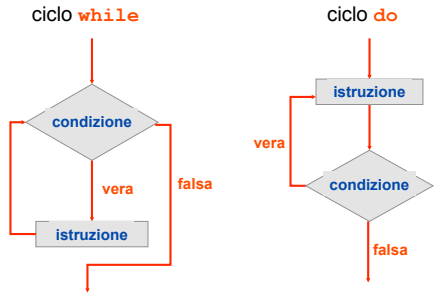
Semantica del ciclo `do-while`



Istruzione `do-while`

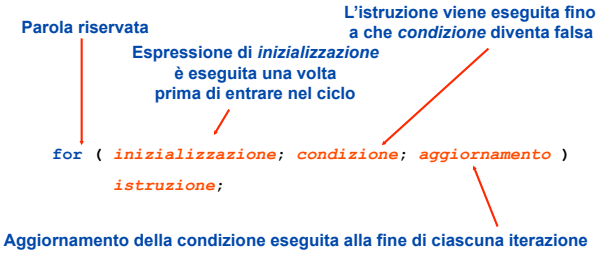
- Il ciclo `do-while` è simile al ciclo `while`, tranne che la condizione è valutata dopo che il corpo del ciclo viene eseguito
- Il corpo del ciclo viene sempre eseguito almeno 1 volta
- `Counter2.java`
- `ReverseNumber.java`
- `TestPrimo.java`

Confronto tra i cicli `while` e `do`



Istruzione `for`

- La sintassi dell'istruzione `for`



Istruzione `for`

- Un ciclo `for` è equivalente al ciclo `while`:

```
inizializzazione;
while ( condizione )
{
   istruzione;
   aggiornamento;
}
```


Semantica del ciclo for

Variabili definite nel for

- La variabile che controlla il ciclo `for` può essere definita all'interno del ciclo e sarà visibile solo nel ciclo
- La sua durata di vita è legata a quella del ciclo
- A ciclo finito la variabile non esiste più

```
for (int i = 1; i<=n; i++) …
```


Istruzione for

- La condizione di un ciclo `for` viene valutata prima di eseguire il ciclo, come nel ciclo `while`
- Di conseguenza, il corpo del ciclo `for` può essere eseguito 0 o più volte
- Questo ciclo è indicato per eseguire istruzioni un numero di volte specifico che può essere determinato a priori

- Counter3.java
- Multiples.java
- Palindrome.java

Cicli infiniti

- Il corpo di un ciclo deve alla fine rendere falsa la condizione perchè il ciclo si fermi
- Altrimenti è un *ciclo infinito*, che viene eseguito finché l'utente non interrompe il programma
- Forever.java
- Un errore logico non infrequente
- Assicuratevi che i vostri cicli abbiano sempre termine

Cicli innestati

- Anche i cicli possono essere innestati
  - Il corpo del ciclo contiene un altro ciclo
  - Come nelle istruzioni `if-else`
- Ogni nuovo ingresso nel ciclo esterno causa un'intera esecuzione del ciclo interno
- PalindromeTester.java

Cicli innestati

```
String s = "";
for (int i = 1; i <= MAX; i++) {
   for (int j = 1; j <= i; j++)
      s = s + '*';
   s = s + '\n';
}
```

Il ciclo interno dipende dalla varabile che controlla il ciclo esterno.

CicloInnestato.java
Stars.java

Ancora istruzione for

- Ogni espressione nella dichiarazione di un ciclo `for` è opzionale
  - manca l'espressione di inizializzazione: nessuna inizializzazione viene effettuata
  - manca l'espressione della condizione: si considera che sia sempre vera, e si realizza un ciclo infinito
  - manca l'espressione di aggiornamento: non si esegue nessun aggiornamento
- Ma il carattere `;` è sempre necessario anche quando manca l'espressione corrispondente

Dichiarazioni multiple

- Si possono definire all'interno del `for` più variabili

```
for (int i=0, j=5; i<=10; i++, j--) …
```

- Per la leggibilità del codice rimane meglio

```
int j = 5;
for (int i=0; i<=10; i++)
{
   j--;
}
```


L'istruzione vuota

- Nei cicli `for` si possono avere situazioni tipo:

```
for (anno=1;
     (saldo += saldo*interesse/100) <
                                obiettivo;
      anno++
     );
```

- Alla fine del ciclo il dato importante è il valore della variabile anno
- Ricordarsi però il `;`, altrimenti l'struzione successiva diventa il corpo del ciclo

Contare le iterazioni

- I limiti del ciclo sono asimmetrici

```
for (int i=0; i<10; i++)
```

la variabile di controllo `i` varia `0 <= i < 10`

- I limiti sono simmetrici

```
for (int i=0; i<=10; i++)
```

la variabile di controllo `i` varia `0 <= i <= 10`

- Per cicli asimmetrici è più facile il conteggio dell'iterazioni
- Quanti n umeri ci sono tra 0 e 10 estremi inclusi?
- Questo errore dovuto a +1 è frequente

Esempi di ciclo

- In `SommaFrazioni.java` ciclo per acquisire frazioni e riportarne la somma
- Azzerare la somma
- Eseguire il ciclo
  - Acquisire una frazione
  - Aggiornare la somma

  finchè ci sono frazioni
- Scrivere la somma ottenuta nel ciclo

Elaborazione di dati in input

- Spesso capita di dover elaborare una serie di numeri che vengono letti da input

```
boolean fatto = false;
while (!fatto) {
   String riga = leggi la riga;
   if (i dati sono terminati)
        fatto = true;
   else
        elabora i dati
}
```


Elaborazione di dati in input

- La verifica della fine dei dati avviene all'interno del ciclo
- Prima occore cercare di leggere qualcosa
- *Comincia a fare il lavoro, verifica che tutto sia a posto, procedi nel lavoro*
- Ecco perché serve anche una variabile booleana
- `DataSet.java` e `LeggiDecimali.java`

Leggere i dati di input

- La classe `JOptionPane` offre un metodo di classe `showInputDialog` che mostra una finestra di dialogo per ricevere dati di tipo diverso
- La finestra restituisce i dati sottoforma di stringa
- Se si devono leggere numeri le stringhe vanno trattate con i metodi delle classi involucro
  `Integer.parseInt(String)` e `Double.parseDouble(String)`
- La classe offre altri metodi
  `showConfirmDialog`, `showMessageDialog` ...
  Es: **`LeggiNumeri.java`**

Migliorare la leggibilità del ciclo

- Assegnamento con effetto collaterale

```
while (
(riga =
JOptionPane.showInputDialog("input…"))
!= null)
{
   elabora il dato
}
```

- In generale gli effetti collaterali sono da evitare, ma in questo caso fa risparmiare l'uso della variabile boolean per controllare il ciclo `while`
- Es **`LeggiNumeri.java`**

Enunciato `break` e `continue`

- L'istruzione `break` serve per interrompere un ciclo
  - Nell'istruzione `switch` interrompe l'esecuzione limitandola all'istruzioni del blocco corrispondente all'etichetta che eguaglia l'espressione che controlla il ciclo
- L'istruzione `continue` interrompe il passo corrente del ciclo e lo riprende con il passo successivo

Esempi

```
for (int i = 0; i < 100; i++) {
  if (i == 15) break;
  if (i % 5 != 0) continue;
  System.out.println (i);
}

int i = 0;
while (true) {
  i++;
  int j = i * 4;
  if (j > 60) break;
  if (i % 10 == 0) continue;
  System.out.println (i);
}
```

BreakContinue.java

Migliorare la leggibilità del ciclo

- Uso dell'istruzione `break`

```
while (true) {
String riga =
JOptionPane.showInputDialog("input…")
if (riga != null) {
   elabora il dato }
else
    break;
}
```


Esempi

Palindrome:

```
for (int i = 0; i < f; i++, f--)
  if (s.charAt(i) != s.charAt(f)){
     palindrome = false;
     break;}
```

- Interrompe il ciclo quando incontra 2 car ≠
- Occorre comunque fare un test successivo `(palindrome)` per sapere come si è usciti dal ciclo

Esempi

Sommari numeri pari da input, 0 per terminare:

```
do {
  x = in.readInt();
  if (x == 0) break;
  if (x % 2 != 0) continue;
  somma += x; }
while (true);
```

- Con `break` si termina il ciclo
- Con `continue` si termina il passo corrente

Cercare in una stringa di caratteri

- Spesso si devono esaminare i singoli caratteri di una stringa
- Uso dei metodi `charAt()` e `length()`

```
for (int i=0; i<s.length(); i++) {
   char car = s.charAt(i);
   elabora car
}
```

- Es: ContaVocali.java

Cercare in una riga di input

- L'input spesso viene letto in un'unica riga che può contenere diversi dati di input e che quindi va decomposta
- la riga: 1.5 100 4.23 4
  potrebbe dare errore, anche se l'intenzione è chiara
- Va usata la classe `StringTokenizer` del pacchetto `java.util`

La classe `StringTokenizer`

- La classe `StringTokenizer` è definita nel pacchetto `java.util`
- Un oggetto `StringTokenizer` separa una stringa di caratteri in sottostringhe più piccole (*tokens*)
- Il costruttore `StringTokenizer` riceve come parametro la stringa originale da separare
- Per default, il tokenizer separa la stringa di input agli spazi bianchi
- Ogni invocazione del metodo `nextToken` riporta il prossimo token nella stringa di input

Uso della classe `StringTokenizer`

- Costruire un oggetto della classe

```
StringTokenizer tokenizer = new
            StringTokenizer(input);
```

- Invocare il metodo `nextToken` per ottenere un elemento alla volta

```
tokenizer.nextToken();
```

- Attenzione: se non ci sono più elementi nell'input si ha errore

```
while (tokenizer.hasMoreTokens()) {
   fai cose e vedi gente
}
```


I separatori di token

- Di default il separatore di elementi per la classe `StringTokenizer` è lo spazio
- È possibile cambiare il separatore
  - Con virgole o punti
- Va specificato nel costruttore

```
new StringTokenizer(input, ",");
```

- In questo caso gli spazi vengono presi come parte integrante del token

Esempi

- `LeggiDecimaliToken.java` e `DataSet.java`
- `PigLatin.java` e `PigLatinTranslator.java`
- `ExamGrades.java`