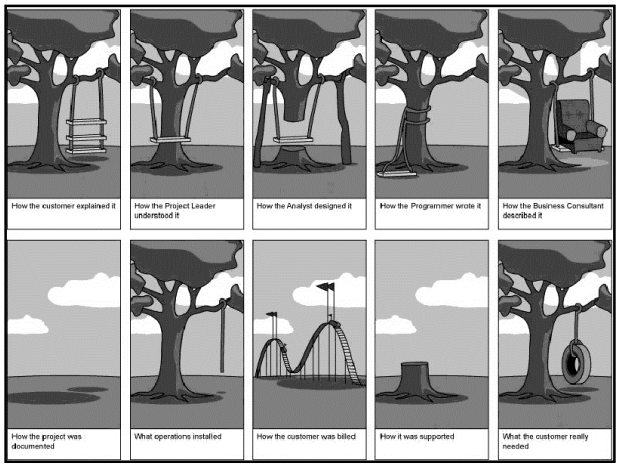


La programmazione strutturata - III

Stefano Mizzaro

Dipartimento di matematica e informatica
Università di Udine
http://www.dimi.uniud.it/mizzaro
mizzaro@dimi.uniud.it

Programmazione, lezione 6
13 ottobre 2004

Dove siamo

- Strutture di controllo della programmazione strutturata
  - Sequenza
    - `;` e `{}`
  - Selezione
    - **`if, if/else, switch/case`**
  - Iterazione (ripetizione)
    - **`while, do/while, for`**
    - **`(break, continue)`**
- Sviluppo incrementale

Oggi

- Esempi ed esercizi
- **`do/while`**
- **`for`**
- Cicli annidati
- Esempi, esercizi, errori tipici
- Programmazione strutturata

Esempi ed esercizi

- Scrivete un programma che visualizza tutti i numeri fra 1 e 100 che siano multipli di 2 o di 3

```
i = 1;
while (i <= 100) {
  if (i % 2 == 0 || i % 3 == 0)
    System.out.println(i);
  i = i + 1;
}
```


Esempi ed esercizi

- Scrivete un programma che visualizza tutti i numeri fra 1 e 100 che siano multipli di 2 e di 3

```
i = 1;
while (i <= 100) {
  if (i % 2 == 0 && i % 3 == 0)
    System.out.println(i);
  i = i + 1;
}
```

- MEGLIO! Multipli di 2 e di 3?? Ah, multipli di 6!!

```
i = 6;
while (i <= 100) {
  System.out.println(i);
  i = i + 6;
}
```

Il `do/while`

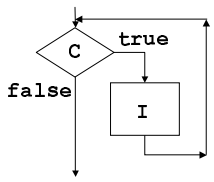
- Un'altra istruzione di iterazione
- `do I while (C);`
  - "Fai `I` mentre `C` è vera"
- Differenza dal `while`:
  - Il controllo del valore di `C` viene effettuato alla fine dell'iterazione, anziché all'inizio
  - Con `while (C) I`, potrebbe darsi che `I` non sia eseguita neanche una volta
  - Con `do I while (C)`, sicuramente `I` viene eseguita almeno una volta

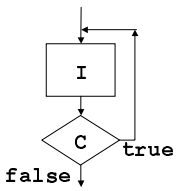
Diagramma di flusso

- Differenze fra `while` e `do/while`

```
while (C)
  I
```

```
do {
  I
} while (C);
```





Esempio

- Stampare i numeri da 1 a 10 con un `do/while`

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

```
i = 0;
do {
  i = i + 1;
  System.out.println(i);
} while (i < 10);
```


Puntualizzazioni

- Le parentesi graffe sono superflue ("`do`" e "`while`" fanno già da parentesi)
  - Meglio metterle per leggibilità
  - "`} while (C)`" non è senz'altro un inizio di ciclo `while` (sarebbe "`while (C)`")
- `do/while` è superfluo, ma fa comodo
- Si usa meno del `while` perché "non uniforme"

Simulare il `do/while` con il `while`

- Bisogna essere sicuri che `I` sia eseguita almeno una volta…

```
do {
  I
} while (C);
```

⇒

```
I;
while (C) {
  I
}
```

⇒

```
boolean primo;
primo = true;
while (C || primo) {
  primo = false;
  I;
}
```


Simulare il `while` con il `do/ while`

- Più difficile
- Non è detto che `I` sia eseguita almeno una volta…
- … bisogna innanzitutto controllare `C`…

```
while (C) {
  I
}
```

⇒

```
if (C)
  do {
    I
  } while (C);
```

Esempio: controllo input

- Se volete leggere un carattere che DEVE essere '**s**' o '**n**':

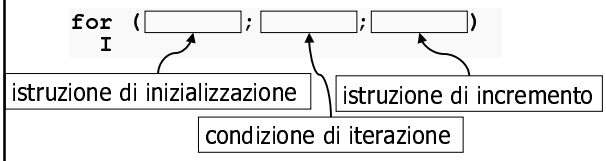
```
do {
  c = System.in.read();
} while ((c != 'n') && (c != 's'));
```

Leggi.unChar()

- …meglio del **while** perché la prima iterazione deve essere "diversa", non uniforme

Il **for**

- Terza e ultima istruzione di iterazione



- (e **I** è l'istruzione che viene iterata)
- Intestazione e corpo del ciclo **for**

Esempi

```
for (i = 0; i <= 10; i = i + 1)
  System.out.println(i);
```

```
for (i = 0; i <= 10; i = i + 1) {
  System.out.print(i + " ");
  System.out.println(i * i);
}
```


Significato del **for**

- Quindi il significato di

```
for(Init; C; Inc) I
```

è:

  - Nell'istruzione di inizializzazione **Init** viene inizializzato il valore di una variabile di controllo
  - Se la condizione di iterazione **C** è **true**, il corpo del ciclo (**I**) viene eseguito
  - Al termine dell'esecuzione di **I** viene eseguita l'istruzione di incremento **Inc**
  - …e **C** e **Inc** "riguardano" la variabile di controllo

Cicli **for** e **while**

- Si può simulare un **for** con un **while**:

```
for (I1; C; I2)
  I3;
```

⟹

```
I1;
while (C) {
  I3;
  I2;
}
```


Vantaggio del **for**

- Gestione unitaria della variabile di controllo
  - Inizializzazione
  - Incremento
  - Controllo
- Con un **while** o con un **do/while** invece devo cercare in giro per il codice
- E anche la dichiarazione può essere fatta insieme all'inizializzazione…

Dichiarazione nel `for`

- Si può dichiarare la variabile di controllo del ciclo nell'intestazione del **`for`**:

```
for (int i = 0; i <= 10; i = i + 2)
  System.out.println(i);
```

- È comodo, ma attenzione: **`i`** esiste solo nel corpo del **`for`**

```
for (int i = 0; i <= 10; i = i + 2)
  System.out.println(i);
System.out.println(i);
```


Incrementi e decrementi…

- L'incremento può non essere unitario…

```
for (i = 0; i <= 10; i = i + 2)
  System.out.println(i);
```

- … o essere un decremento

```
for (i = 10; i >= 1; i = i - 1)
  System.out.println(i);
```


Errori tipici

- … quello che sembra nel corpo del **`for`** non lo è…

```
for (int i = 0; i <= 10; i = i + 1);
  System.out.println(i);
```

- … la condizione è invertita

```
for (int i = 10; i <= 1; i = i - 1)
  System.out.println(i);
```

- …

Uso del `for`

- Non modificate la variabile di controllo nel corpo del **`for`** → usate il **`while`**
- Evitare cose del tipo

```
for ( ; ; )
  I
```

⟹

```
while (true)
  I
```


Operatori di incremento e decremento

- Le forme più tipiche dell'istruzione di incremento in un **`for`** sono
  `i = i + 1` e **`i = i - 1`**
- 2 "istruzioni" più comode
  - **`++i`** (o **`--i`**): pre-incrementa (decrementa) **`i`**
  - **`i++`** (o **`i--`**): post-incrementa (o decrementa) **`i`**
- (e non solo nei **`for`**…)
- Sono operatori…

Sono operatori!

- Quindi possono stare dentro a espressioni
- Anche se spesso sono usati da soli:

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

```
for (int i = 10; i >= 1; i--)
  System.out.println(i);
```

- Pre e post: quanto vale **`i`**?

```
int a, i;
a = 5;
i = a++;
```

```
i = a;
a++;
```

```
int a, i;
a = 5;
i = ++a;
```

```
a++;
i = a;
```

Dove siamo

- Strutture di controllo della programmazione strutturata
  - Sequenza
    - `;` e `{}`
  - Selezione
    - `if, if/else, switch/case`
  - Iterazione (ripetizione)
    - `while, do/while, for`
    - `(break, continue)`
- Sviluppo incrementale

Cicli annidati

- Cicli dentro ad altri cicli

```
for (i = 1; i <= 5; i++) {
  for (j = 1; j <= 10; j++)
    System.out.print('+');
  System.out.println();
}
```

```
for (i = 1; i <= 5; i++) {
  for (j = 1; j <= i; j++)
    System.out.print('+');
  System.out.println();
}
```

- Ragionare "a scatole"…

Quale istruzione uso?

- Selezione (**`if`** o **`switch`**)
  - Condizione e numero di alternative
- Iterazione (**`while, do/while, for`**)
  - **`for`**: cicli "standard"
    - Variabile di controllo che assume valori equidistanti fino a una soglia
    - Numero di iterazioni noto a priori
  - **`while, do/while`**: almeno un'iterazione?

Altri errori tipici (1/2)

- Ciclo infinito
  - Per mancata modifica variabile (dimenticanza…)

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

  - Per errata condizione

```
i = 1;
while (i != 10) {
  System.out.println(i);
  i = i + 2;
}
```


Altri errori tipici (2/2)

- "Errore di uno"
  - Stampare numeri da 1 a 10

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

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

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

```
i = 1;
while (i <= 10) {
  System.out.println(i);
  i++;
}
```

```
for (i = 1;i <= 10;i++)
  System.out.println(i);
```

```
i = 0;
while (i <= 9) {
  i++;
  System.out.println(i);
}
```


La programmazione strutturata

- L'istruzione di salto:
  - Etichetta (label)
  - "`goto`" o "`jump`"
- **`if`** e salto permettono di simulare tutti i cicli
- Ma… e il viceversa?

▪ Sequenza ▪ Selezione ▪ Salto	vs.	▪ Sequenza ▪ Selezione ▪ Iterazione

3 domande…

- Iterazione vs. salto
  1. Si perde qualcosa senza il salto?
- **`while, do/while`** e **`for`**
  2. Ma servono tutte e tre?
- Perché
  3. Ma perché queste scelte in Java (e in altri linguaggi)?

Stefano Mizzaro - Prog. strutt. 3 31

… e 3 risposte

- Böhm e Jacopini (1966)
  1. Con l'iterazione e senza il salto si riescono a rappresentare gli stessi algoritmi
  2. Con una sola fra **`while, do/while`** e **`for`** si simulano le altre 2
     - (bastano sequenza e **`while`**. Ex.: simulare **`if/else`** con **`while`**…)
- Dijkstra (1968)
  3. Salti = minore comprensibilità (spaghetti logic/programming)

Stefano Mizzaro - Prog. strutt. 3 32

Riassunto

- Strutture di controllo, programmazione strutturata
  - Sequenza
    - `;` e `{}`
  - Selezione
    - **`if, if/else, switch/case`**
  - Iterazione (ripetizione)
    - **`while, do/while, for`**
    - **`(break, continue)`**
- Sviluppo incrementale

Stefano Mizzaro - Prog. strutt. 3 33

Prossima lezione

- **`break`** e **`continue`**
- Esempi ed esercizi
- Array

Stefano Mizzaro - Prog. strutt. 3 34