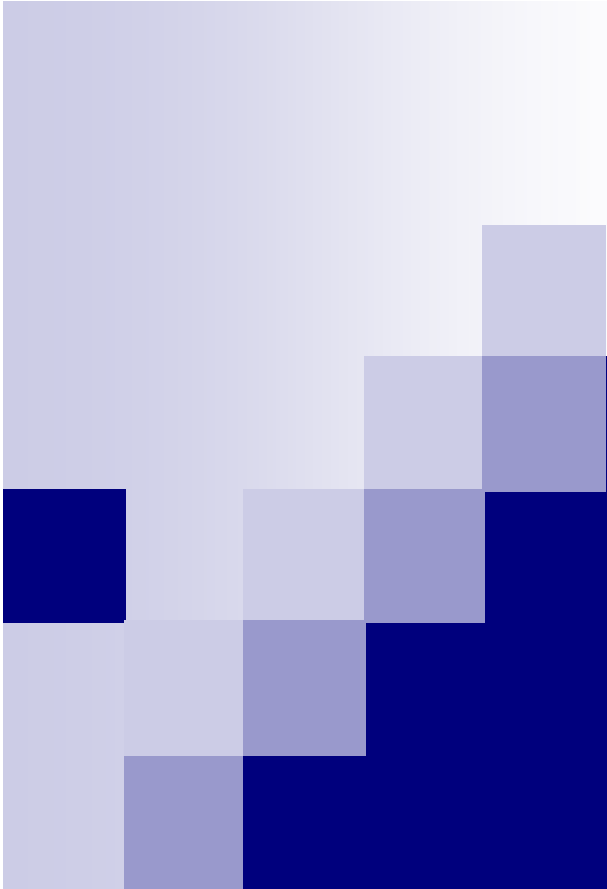




# Costrutti per il controllo del flusso di esecuzione

Fondamenti di informatica L-B  
Laboratorio

Alessandro Falchi  
[afalchi@deis.unibo.it](mailto:afalchi@deis.unibo.it)

# Operatori (1)

| PRECEDENZA | ASSOCIATIVITÀ | OPERATORE       | TIPO/I DEGLI OPERANDI | OPERAZIONE ESEGUITA                 |
|------------|---------------|-----------------|-----------------------|-------------------------------------|
| 15         | L (left)      | .               | object, member        | object member access                |
|            |               | [ ]             | array, int            | array element access                |
|            |               | ( <i>args</i> ) | method, arglist       | method invocation                   |
|            |               | ++ , --         | variable              | post-increment, decrement           |
| 14         | R (right)     | ++, --          | variable              | pre-increment, decrement            |
|            |               | + , -           | number                | unary plus, unary minus             |
|            |               | ~               | integer               | bitwise complement                  |
|            |               | !               | boolean               | boolean NOT                         |
| 13         | R             | new             | class, arglist        | object creation                     |
|            |               | ( <i>type</i> ) | type, any             | cast (type conversion)              |
| 12         | L             | *, /, %         | number, number        | multiplication, division, remainder |
| 11         | L             | + , -           | number, number        | addition, subtraction               |
|            |               | +               | string, any           | string concatenation                |
| 10         | L             | <<              | integer, integer      | left shift                          |
|            |               | >>              | integer, integer      | right shift with sign extension     |
|            |               | >>>             | integer, integer      | right shift with zero extension     |
| 9          | L             | < , <=          | number, number        | less than, less than or equal       |
|            |               | > , >=          | number, number        | greater than, greater than or equal |
|            |               | instanceof      | reference, type       | type comparison                     |

# Operatori (2)

| PRECEDENZA | ASSOCIATIVITÀ | OPERATORE        | TIPO/I DEGLI OPERANDI | OPERAZIONE ESEGUITA                    |
|------------|---------------|------------------|-----------------------|----------------------------------------|
| 8          | L (left)      | = =              | primitive, primitive  | equal (have identical values)          |
|            |               | !=               | primitive, primitive  | not equal (have different values)      |
|            |               | = =              | reference, reference  | equal (refer to same object)           |
|            |               | !=               | reference, reference  | not equal (refer to different objects) |
| 7          | L             | &                | integer, integer      | bitwise AND                            |
|            |               | &                | boolean, boolean      | boolean AND                            |
| 6          | L             | ^                | integer, integer      | bitwise XOR                            |
|            |               | ^                | boolean, boolean      | boolean XOR                            |
| 5          | L             |                  | integer, integer      | bitwise OR                             |
|            |               |                  | boolean, boolean      | boolean OR                             |
| 4          | L             | &&               | boolean, boolean      | conditional AND                        |
| 3          | L             |                  | boolean, boolean      | conditional OR                         |
| 2          | R (right)     | ?:               | boolean, any          | conditional (ternary) operator         |
| 1          | R             | =                | variable, any         | assignment                             |
|            |               | *= , /= , %= ,   | variable, any         | assignment with operation              |
|            |               | += , -= , <=<= , |                       |                                        |
|            |               | >>= , >>>= ,     |                       |                                        |
|            |               | &= , ^= ,  =     |                       |                                        |



## Operatori (3)

|    |                         |                                        |
|----|-------------------------|----------------------------------------|
| >  | maggiore                |                                        |
| >= | maggiore o uguale       |                                        |
| <  | minore                  |                                        |
| <= | minore o uguale         |                                        |
| == | uguale                  | <i>[NB: differenza tra operandi di</i> |
| != | diverso                 | <i>tipo primitivo e riferimento]</i>   |
| && | AND logico condizionale |                                        |
|    | OR logico condizionale  |                                        |
| !  | NOT logico              | <i>[operatore unario]</i>              |
| =  | assegnamento            |                                        |

# Statement (1)

| STATEMENT    | AZIONE               | SINTASSI                                                                                        |
|--------------|----------------------|-------------------------------------------------------------------------------------------------|
| expression   | side effects         | <i>var = expr; expr++; method( ); new Type( );</i>                                              |
| compound     | group statements     | <i>{ statements }</i>                                                                           |
| empty        | do nothing           | <i>;</i>                                                                                        |
| labeled      | name a statement     | <i>label : statement</i>                                                                        |
| variable     | declare a variable   | <i>[final] type name [= value] [, name [= value]] ...;</i>                                      |
| if           | conditional          | <i>if ( expr ) statement [ else statement]</i>                                                  |
| switch       | conditional          | <i>switch ( expr ) { [ case expr : statements ] ... [ default: statements ] }</i>               |
| while        | loop                 | <i>while ( expr ) statement</i>                                                                 |
| do           | loop                 | <i>do statement while ( expr );</i>                                                             |
| for          | simplified loop      | <i>for ( init ; test ; increment ) statement</i>                                                |
| for/in       | collection iteration | <i>for ( variable : iterable ) statement      Java 5.0 ("foreach")</i>                          |
| break        | exit block           | <i>break [ label ] ;</i>                                                                        |
| continue     | restart loop         | <i>continue [ label ] ;</i>                                                                     |
| return       | end method           | <i>return [ expr ] ;</i>                                                                        |
| synchronized | critical section     | <i>synchronized ( expr ) { statements }</i>                                                     |
| throw        | throw exception      | <i>throw expr ;</i>                                                                             |
| try          | handle exception     | <i>try { statements } [ catch ( type name ) { statements } ] ... [ finally { statements } ]</i> |
| assert       | verify invariant     | <i>assert invariant [ : error] ;      Java 1.4.</i>                                             |



## Statement (2)

Uno statement è un singolo comando eseguito dall'interprete Java

```
String city;  
city="Bologna";  
String city="Bologna";    // uno o due statement?  
;                          // statement vuoto
```

```
{    // inizio compound statement  
    stipendio=stipendio+1000;  
    System.out.println("Aumento effettuato!");  
}    // fine compound statement
```



# Costrutti di Controllo

Sono istruzioni che permettono di controllare il flusso del codice, condizionando l'esecuzione di uno o più statement attraverso la valutazione di un'espressione

Vedremo :

1. *if / else*
2. *switch*

(non si considera l'uso dell'istruzione *goto* associata alle *label* perché è una pratica di programmazione che rende il codice poco leggibile e accresce le possibilità di errore)



## Il costrutto if/else (1)

Condiziona l'esecuzione di uno statement (o di un compound statement) alla valutazione di un'espressione

```
if (condizione)           // “condizione” è di tipo boolean
    statement1 // oppure compound statement
else                       // la parte “else” è opzionale
    statement2 // oppure compound statement
```



## Il costrutto if/else (2)

```
if (level >= 5)
    System.out.println("Warning!");
```

```
if (a>10 && b!=0)                // meglio: if ((a > 10) && (b != 0))
    c=a/b;
```

```
if (accesso)                     // di che tipo è "accesso" ?
    System.out.println("Accesso...");
else
    System.out.println("Login errato...");
```



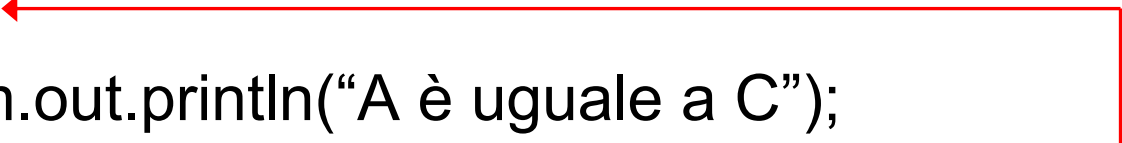
## Il costrutto if/else (3)

```
if (spesa > 0) {                                // stili di indentazione
    cash = cash - spesa;
    System.out.println("Addebito eseguito");
}
else
    System.out.println("Bravo risparmiatore...");
```

```
if (level < 5)
    System.out.println("Tutto ok");              // se avessi messo { e } ?
else {
    System.out.println("Inizio procedura emergenza");
    level = 4;
}
```

## Il costrutto if/else (4)

### Annidamento di costrutti if/else

```
if (a == b) {  
    if (b == c)   
        System.out.println("A è uguale a C");  
else  
    System.out.println("A non è uguale a B");  
}
```

*“else” è riferito all’ “if” immediatamente precedente*

Il codice è sintatticamente corretto? **Si**

E logicamente corretto o errato? **ERRATO!**



## Il costrutto if/else (5)

### Annidamento di costrutti if/else

```
if (a == b) {  
    if (b == c)  
        System.out.println("A è uguale a C");  
} // qui termina lo statement eseguito se a == b è vero  
else {  
    System.out.println("A non è uguale a B");  
}
```



# Il costrutto switch

Selezione multipla sulla base del valore di una variabile  
(analogo al C)

```
switch(expr) {  // “expr” espressione con valore int, short, byte o char
    case const1:
        // zero o più statement...
        break;
    case const2:
        // zero o più statement...
        break;
    ...
    default:
        // zero o più statement...
}
```



# Il costrutto switch

```
switch(punti) {  
    case 3:  
        System.out.println("Vittoria");  
        break;           // salta alla fine dello switch  
    case 1:  
        System.out.println("Pareggio");  
        break;  
    case 0:  
        System.out.println("Sconfitta");  
} // e il break?
```



# Il costrutto switch

```
switch(risposta) {           // di che tipo è "risposta"?
    case 'y':
    case 'Y':
        System.out.println("Inizio formattazione...");
        break;
    case 'n':
    case 'N':
        System.out.println("Formattazione non eseguita");
}
```

```
switch(risposta) {           // cosa cambia rispetto a prima?
    case 'y':
    case 'Y':
        System.out.println("Inizio formattazione...");
        break;
    default:
        System.out.println("Formattazione non eseguita");
}
```



# I Costrutti Ciclici

Sono istruzioni che permettono di ripetere l'esecuzione di uno o più statement in relazione alla valutazione di una condizione.

Java prevede tre costrutti ciclici:

- while
- do/while
- for



# Il ciclo while (1)

Sintassi:

```
while (espressione)
    statement           // semplice o composto
```

Il valore boolean di “espressione” viene valutato **prima** di ogni iterazione:

- se espressione è **true**, statement viene processato e, al termine, si ritorna alla valutazione dell’espressione;
- se espressione è **false**, il flusso di esecuzione prosegue a partire dal primo statement successivo al ciclo while

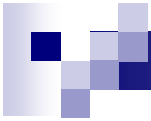


## Il ciclo while (2)

```
int conta=0;
while (conta < 10)
    conta=conta+1;
```

```
int celsius=30;
boolean raffredda=true;
while (raffredda) {                                // cosa succede se non metto { e } ?
    System.out.println("Raffreddamento in corso...");
    celsius=celsius-2;
    if (celsius < 10)
        raffredda=false;
}
```

Quante volte sono eseguiti i due cicli?



## Il ciclo while (3)

Qual è l'output del codice qui sotto?

```
int lezioni_in_lab=0;
while (lezioni_in_lab <= 9) {
    if (lezioni_in_lab < 7)
        System.out.println("Uff, è dura la vita...");
    else
        System.out.println("Ormai ci siamo...");
    lezioni_in_lab++;
}
System.out.println("E ora... vai con l'esame!");
```



# Il ciclo do/while (1)

Sintassi:

do

statement      *// semplice o composto*

while (espressione); *// attenzione al ;*

Il valore boolean di “*espressione*” viene valutato **dopo** ogni iterazione:

- se *espressione* è **true**, si esegue una nuova iterazione (statement viene processato), al termine della quale si ripete la valutazione di *espressione*;
- se *espressione* è **false**, il flusso di esecuzione prosegue a partire dal primo statement successivo al ciclo *while*



## Il ciclo do/while (2)

```
int a=0, b=0;  
do {  
    a=a+1;  
} while (a+b < 8);
```

```
int nLezione;  
do {  
    nLezione=1;  
    System.out.println("Sono alla lezione numero: "+nLezione);  
    nLezione = nLezione + 1;  
} while (nLezione <= 10);
```

**ATTENZIONE!**



Quante volte viene eseguito ciascun ciclo?



# Il ciclo for (1)

Sintassi:

```
for(init; test; update)
    statement           // semplice o composto
```

Rispetto ai cicli while e do/while, il ciclo for esplicita gli elementi tipici delle iterazioni, ovvero il contatore e la condizione di iterazione.

- **init** è lo statement di inizializzazione della variabile “contatore”
- **test** è l'espressione che esprime la condizione a cui ciascuna iterazione è subordinata
- **update** è lo statement che altera il valore del contatore



## Il ciclo for (2)

```
for(int count=0 ; count < 10 ; count++)  
    System.out.println("Eseguo l'interazione " + count);
```

```
int count;  
for(count=0 ; count < 10 ; count++)  
    System.out.println("Eseguo l'interazione " + count);
```

Qual è la differenza tra i due cicli for qui sopra?

**Nota:** così come è possibile leggere i valori delle variabili “contatori” dall'interno del ciclo, è anche possibile **alterarne il valore** (accesso in scrittura) senza che l'interprete Java segnali **alcun errore**. Fate attenzione, perché questo modo di operare, pur lecito, compromette leggibilità e correttezza logica del codice!



## Il ciclo for (3)

Init e update possono essere formati da più espressioni separate dalla virgola:

```
for(int n=1, m=5 ; n < 20 ; n++, m--) {  
    if (n > m)  
        System.out.println(n + "è più grande di" + m);  
    else  
        System.out.println(n + "è minore o uguale a" + m);  
}
```

**Nota:** init, test ed update sono parti opzionali dello statement for. Sono dunque lecite, ad esempio, le seguenti:

- `for(int j=5 ; j < 80 ; )`
- `for( ; x > 7 ; )`



# Alcuni casi particolari

```
int conta=1;      // inizializzazione comune ai tre cicli
```

```
for( ; ; ) {  
    System.out.println("Eseguita iterazione " + conta);  
    conta = conta + 1;  
    break;  
}
```

```
while(true) {  
    System.out.println("Eseguita iterazione " + conta);  
    conta = conta + 1;  
    break;  
}
```

```
do {  
    System.out.println("Eseguita iterazione " + conta);  
    conta = conta + 1;  
    break;  
} while(true);
```



# Esercizi

Per ciascuno dei punti seguenti, scrivere un programma Java.

- data una vocale, scrive su standard output una parola che la contiene almeno due volte (es: 'a' → "casa")
- dato un numero intero, scrive i suoi 8 interi successivi [provare i tre costrutti ciclici]
- definita la classe "Mese" con i seguenti metodi  
    getName();                      // ritorna una stringa con il nome del mese  
    getNumeroGiorni();            // ritorna il numero dei giorni  
il programma, ricevuto un numero compreso tra 1 e 12, crea un'istanza della classe "Mese" e ne invoca i metodi per scrivere il messaggio "il mese <nome> ha <n> giorni"
- premessa  
    java.util.Random dado = new java.util.Random(); // istanzia un generatore di numeri casuali  
    dado.nextInt(6); // ritorna un intero casuale tra **0** e **5** compresi  
Il programma conta il numero di lanci necessari a fare un 6.  
*Estensione:* lancio di due dadi. Il programma conta il numero di lanci necessari ad un doppio 6.