



Decisioni e iterazioni

Istruzione if

L'istruzione if consente a un programma di eseguire azioni diverse a seconda della natura dei dati da elaborare

Un pannello dell'ascensore “salta” il tredicesimo piano.

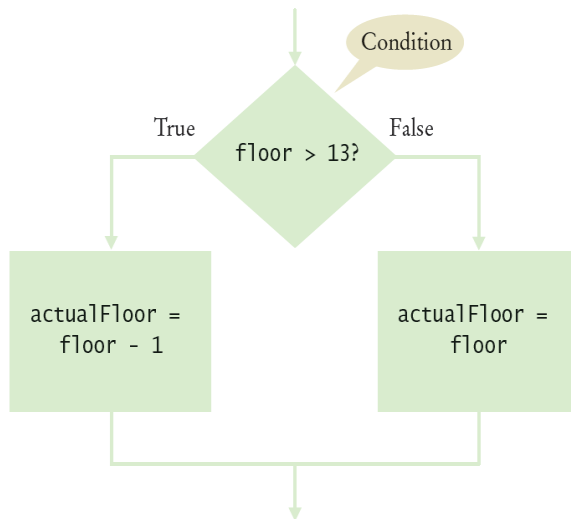
Il piano in realtà non manca: il computer che controlla l'ascensore regola i numeri dei piani sopra il 13.

```
int actualFloor;  
if (floor > 13)  
{  
    actualFloor = floor - 1;  
}  
else  
{  
    actualFloor = floor;  
}
```

Istruzione if

Flowchart con due rami

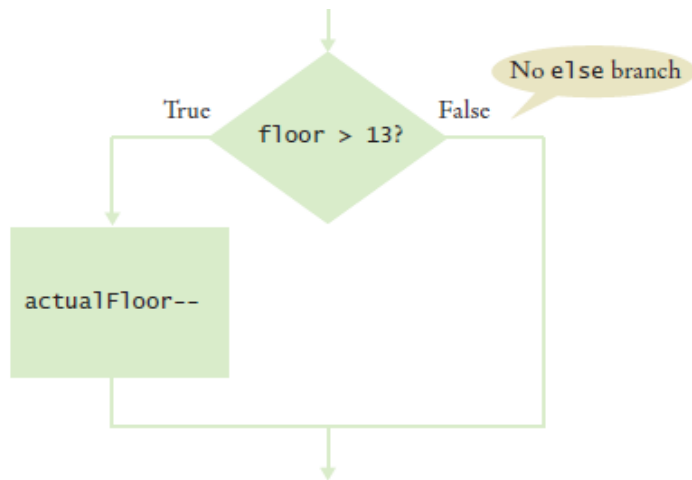
Puoi includere tutte le dichiarazioni che desideri in ogni ramo.



```
int actualFloor;  
if (floor > 13)  
{  
    actualFloor = floor - 1;  
}  
else  
{  
    actualFloor = floor;  
}
```

Istruzione if

Quando non c'è niente da fare nel ramo else, si omette del tutto



```
int actualFloor;  
if (floor > 13)  
{  
    actualFloor --;  
}
```

Istruzione if sintassi

Syntax

```
if (condition)
{
    statements
}
```

```
if (condition) { statements1 }
else { statements2 }
```

A condition that is true or false.
Often uses relational operators:
== != < <= > >=

Le parentesi graffe non sono necessarie se il corpo contiene un solo enunciato ma è bene metterle

```
if (floor > 13)
{
    actualFloor = floor - 1;
}
else
{
    actualFloor = floor;
}
```

Non mettere il punto e virgola qui!

If the condition is true, the statement(s) in this branch are executed in sequence; if the condition is false, they are skipped.

Omit the else branch if there is nothing to do.

Allineare le graffe è una buona idea

If the condition is false, the statement(s) in this branch are executed in sequence; if the condition is true, they are skipped.

```
1 import java.util.Scanner;
2
3 /**
4  * This program simulates an elevator panel that skips the 13th floor.
5  */
6 public class ElevatorSimulation
7 {
8     public static void main(String[] args)
9     {
10         Scanner in = new Scanner(System.in);
11         System.out.print("Floor: ");
12         int floor = in.nextInt();
13
14         // Adjust floor if necessary
15
16         int actualFloor;
17         if (floor > 13)
18         {
19             actualFloor = floor - 1;
20         }
21         else
22         {
23             actualFloor = floor;
24         }
25
26         System.out.println("The elevator will travel to the actual floor "
27             + actualFloor);
28     }
29 }
```

Evita duplicati

Se si ha un codice duplicato in ogni ramo, si sposta fuori da if dichiarazione.

Da non fare:

```
if (floor > 13){  
    actualFloor = floor - 1;  
    System.out.println("Actual floor: " + actualFloor);  
}else{  
    actualFloor = floor;  
    System.out.println("Actual floor: " + actualFloor);  
}
```

Evita duplicati

Renderà il codice molto più facile da mantenere.

Le modifiche dovranno essere apportate solo in un posto

```
if (floor > 13) {  
    actualFloor = floor - 1;  
} else {  
    actualFloor = floor;  
}  
System.out.println("Actual floor: " + actualFloor);
```

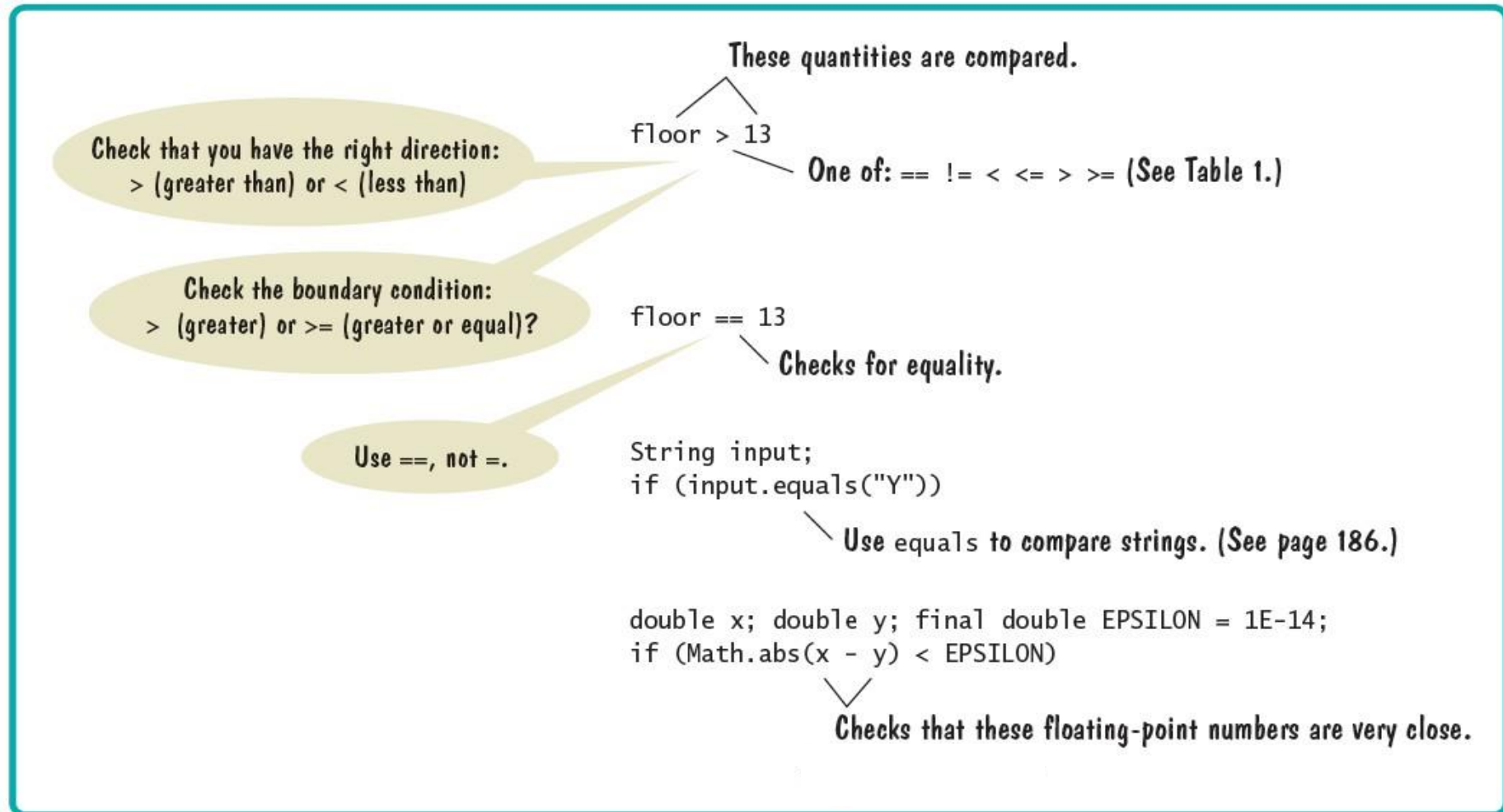

Operatori relazionali

Gli operatori relazionali hanno una precedenza inferiore rispetto agli operatori aritmetici:

`floor - 1 < 13`

Java	Math Notation	Description
<code>></code>	$>$	maggiore di
<code>>=</code>	\geq	maggior o uguale di
<code><</code>	$<$	minore di
<code><=</code>	\leq	minore o uguale
<code>==</code>	$=$	uguale
<code>!=</code>	\neq	diverso

Confronti:



Confronti di numeri in virgola mobile

Vediamo:

```
double r = Math.sqrt(2);  
double d = r * r - 2;  
if (d == 0){  
    System.out.println("sqrt(2) squared minus 2 is 0");  
}else{  
    System.out.println("sqrt(2) squared minus 2 is not 0 but " +  
        d);  
}
```

Se stampo: sqrt(2) non è 0 ma
4.440892098500626E-16

Ciò è dovuto a errori di arrotondamento
Quando si confrontano numeri in virgola
mobile, non verificare l'uguaglianza. Controlla
se sono abbastanza vicini.

Confronti di numeri in virgola mobile

Per evitare errori di arrotondamento, non utilizzare `==` per confrontare i numeri a virgola mobile.

Per confrontare i numeri in virgola mobile, verifica se sono abbastanza vicini:
 $|x - y| \leq \epsilon$ (di solito impostato a 10^{-14})

```
final double EPSILON = 1E-14;
if (Math.abs(x - y) <= EPSILON)
{
    // x is approximately equal to y
}
```

Confronti di stringhe- compareTo()

Il metodo compareTo confronta le stringhe in ordine lessicografico (ordine del dizionario).

`string1.compareTo(string2) < 0`

significa: string1 viene prima di string2 nel dizionario

`string1.compareTo(string2) > 0`

significa: string1 viene dopo string2 nel dizionario

`string1.compareTo(string2) == 0`

significa: stringa1 e stringa2 sono uguali

Confronti di stringhe- compareTo()

c a r

c a r t

c a t

Letters r comes
match before t

*Lexicographic
Ordering*

Ordine Lessicografico

Differenze nell'ordinamento e nell'ordinamento del dizionario in Java

- Tutte le lettere maiuscole vengono prima delle lettere minuscole. "Z" viene prima di "a"
- Il carattere spazio viene prima di tutti i caratteri stampabili
- I numeri vengono prima delle lettere
- L'ordine dei segni di punteggiatura varia

Per vedere quale dei due termini viene prima nel dizionario, considera la prima lettera in cui differiscono



Confronto tra oggetti

L'operatore `==` verifica se due riferimenti a oggetti sono identici, se si riferiscono allo stesso oggetto.

```
Rectangle box1 = new Rectangle(5, 10, 20, 30);  
Rectangle box2 = box1;  
Rectangle box3 = new Rectangle(5, 10, 20, 30);
```

`box1 == box2` è true

`box1 == box3` è false

Si usa il metodo `equals()` per verificare se due rettangoli hanno lo stesso contenuto

`box1.equals(box3)`

Hanno lo stesso angolo in alto a sinistra e la stessa larghezza e altezza

Avvertenza: gli uguali devono essere definiti per la classe

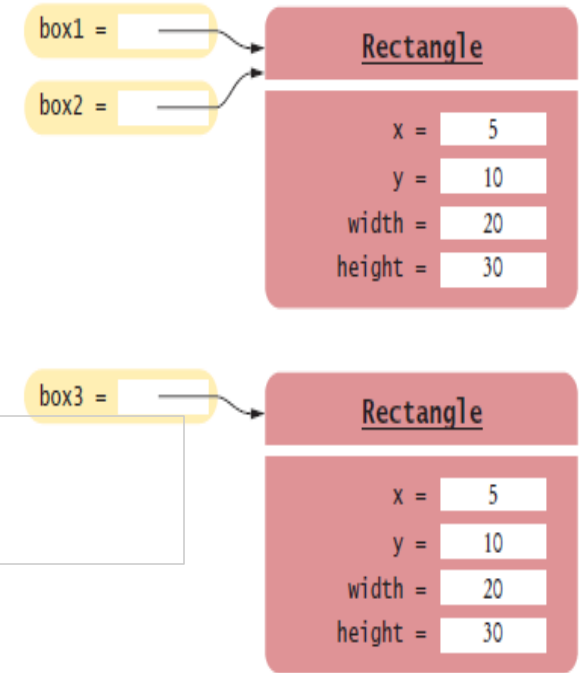





Table 2 Relational Operator Examples

Expression	Value	Comment
<code>3 <= 4</code>	true	3 is less than 4; <= tests for “less than or equal”.
 <code>3 =< 4</code>	Error	The “less than or equal” operator is <=, not =<. The “less than” symbol comes first.
<code>3 > 4</code>	false	> is the opposite of <=.
<code>4 < 4</code>	false	The left-hand side must be strictly smaller than the right-hand side.
<code>4 <= 4</code>	true	Both sides are equal; <= tests for “less than or equal”.
<code>3 == 5 - 2</code>	true	== tests for equality.
<code>3 != 5 - 1</code>	true	!= tests for inequality. It is true that 3 is not 5 – 1.
 <code>3 = 6 / 2</code>	Error	Use == to test for equality.
<code>1.0 / 3.0 == 0.33333333</code>	false	Although the values are very close to one another, they are not exactly equal. See Section 5.2.2.
 <code>"10" > 5</code>	Error	You cannot compare a string to a number.
<code>"Tomato".substring(0, 3).equals("Tom")</code>	true	Always use the equals method to check whether two strings have the same contents.
<code>"Tomato".substring(0, 3) == ("Tom")</code>	false	Never use == to compare strings; it only checks whether the strings are stored in the same location. See Common Error 5.2 on page 192.

Esercizio

Quali delle seguenti è vera, con $a=3$ e $b=4$?

- $a + 1 \leq b$
- $a + 1 \geq b$
- $a + 1 \neq b$



Esercizio

Cosa c'è di sbagliato?

```
if (scoreA = scoreB)
{
    System.out.println("Tie");
}
```

Esercizio

Due modi per testare che una stringa è vuota



```
Stringa = "1";  
String b = "one";  
double x = 1;  
double y = 3 * (1.0 / 3);
```

Quali sono sbagliate?

1. a == "1"

2. a == null

3. a.equals("")

4. a == b

5. a == x

6. x == y

7. x - y == null

8. x.equals(y)

ALTERNATIVE MULTIPLE

- È possibile combinare più istruzioni if per valutare decisioni complesse. Si utilizzano istruzioni if multiple per implementare più alternative.



ALTERNATIVE MULTIPLE

Esempio:

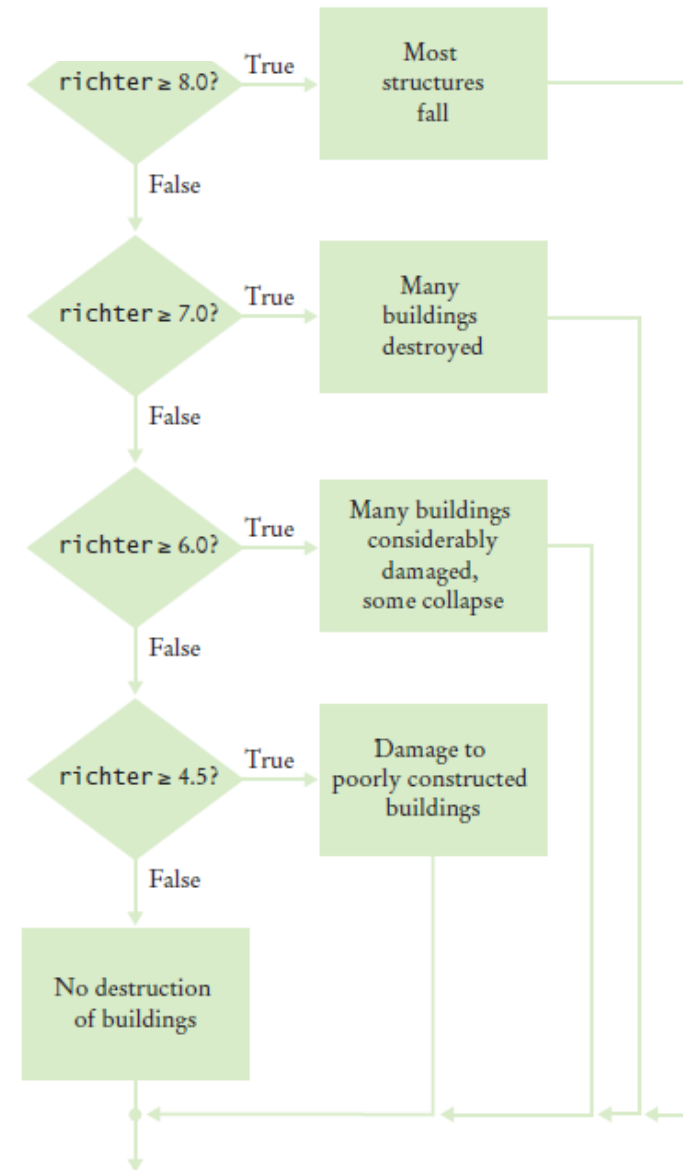
Danno causato da terremoto di una data magnitudo sulla scala Richter

```
if (richter >= 8.0){  
    description = "Most structures fall";  
}else if (richter >= 7.0){  
    description = "Many buildings destroyed";  
}else if (richter >= 6.0){  
    description = "Many buildings considerably damaged, some collapse";  
}else if (richter >= 4.5){  
    description = "Damage to poorly constructed buildings";  
}else{  
    description = "No destruction of buildings";  
}
```

Table 3 Richter Scale	
Value	Effect
8	Most structures fall
7	Many buildings destroyed
6	Many buildings considerably damaged, some collapse
4.5	Damage to poorly constructed buildings

ALTERNATIVE MULTIPLE FLOWCHART

Figure 4
Multiple Alter



Alternative multiple

L'ordine del if e else if conta

Quando si utilizzano istruzioni if multiple, bisogna verificare le condizioni generali dopo condizioni più specifiche.

Errore:

```
if (richter >= 4.5) {  
    description = "Damage to poorly constructed buildings";  
}else if (richter >= 6.0){  
    description = "Many buildings considerably damaged, some  
collapse";  
}else if (richter >= 7.0){  
    description = "Many buildings destroyed";  
}else if (richter >= 8.0){  
    description = "Most structures fall";  
}
```

Alternative multiple

In questo esempio, devi usare la sequenza if/elseif/else, non solo più istruzioni if indipendenti

Errore:

```
if (richter >= 8.0) // Didn't use else
{
    description = "Most structures fall";
}
if (richter >= 7.0)
{
    description = "Many buildings destroyed";
}
if (richter >= 6.0)
{
    description = "Many buildings considerably damaged, some collapse";
}
if (richter >= 4.5)
{
    "Damage to poorly constructed buildings";
}
```

Le alternative non sono più esclusive.

Enunciato switch

Per confrontare un singolo valore a diversi valori alternativi, invece di una serie di enunciati if/else if/ else → switch

Esempio:

```
int digit=..;
switch(digit){
    case 1: digitName= " one "; break;
    case 2: digitName= " two "; break;
    case 3: digitName= " three "; break;
    case 4: digitName= " four "; break;
    case 5: digitName= " five "; break;
    ...
    default: digitName=" "; break;
}
```

Enunciato switch

Per confrontare un singolo valore a diversi valori alternativi, invece di una serie di enunciati if/else if/ else → switch

PROBLEMA: i valori presenti nelle verifiche devono essere costanti e possono essere interi o caratteri (anche stringhe a partire da Java 7). È necessario mettere il break altrimenti fa più rami diversi fino all'incontro del break, non sempre è un problema MA LA MAGGIOR PARTE DELLE VOLTE è UN PROBLEMA!!

A large orange circle is positioned on the left side of the slide, partially cut off by the edge.

DIRAMAZIONI ANNIDATE

Insieme nidificato di istruzioni:

Un'istruzione if all'interno di un'altra

Esempio: imposta federale (USA) sul reddito

L'imposta dipende dallo stato civile e
dal reddito

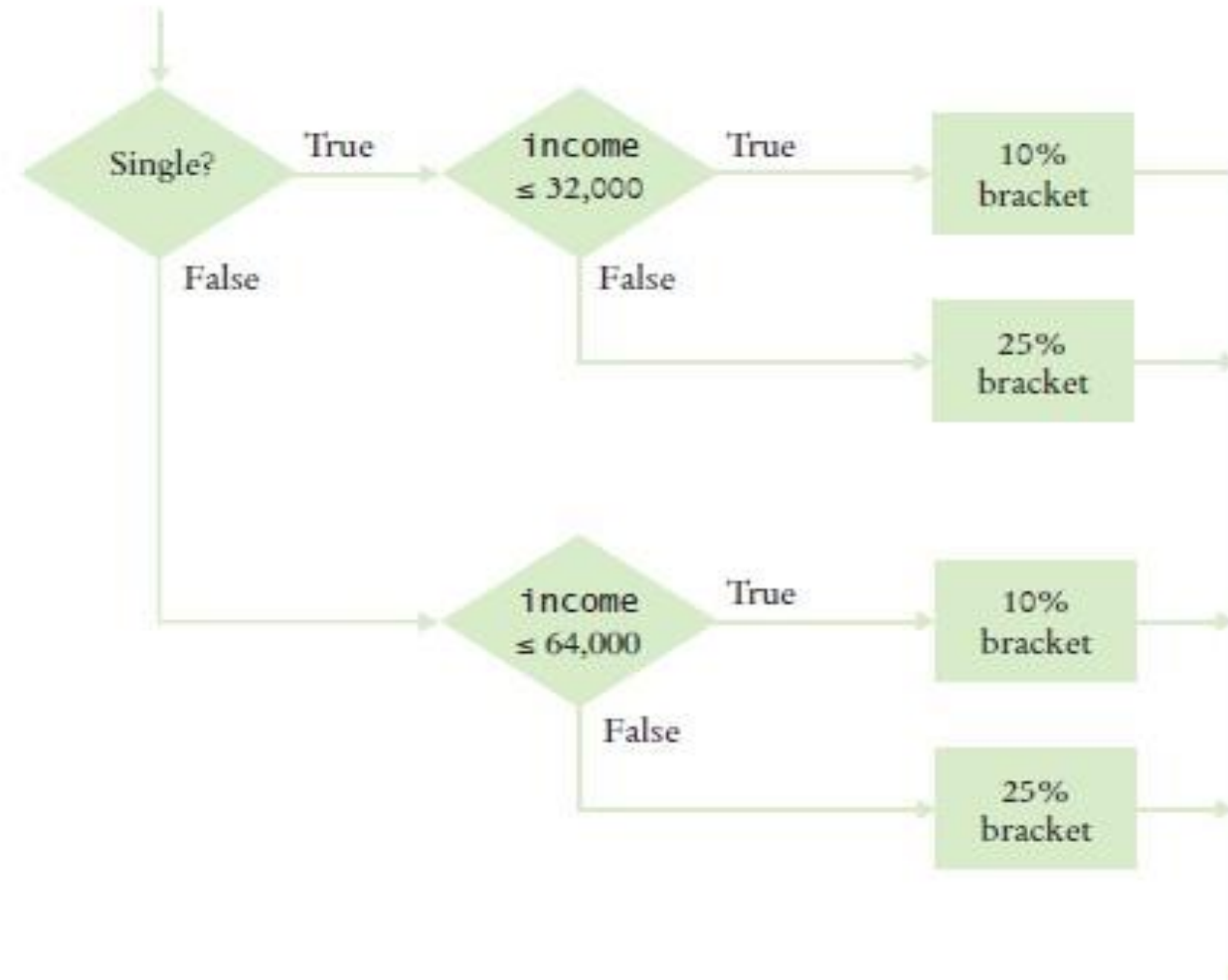


DIRAMAZIONI ANNIDATE

Table 4 Federal Tax Rate Schedule

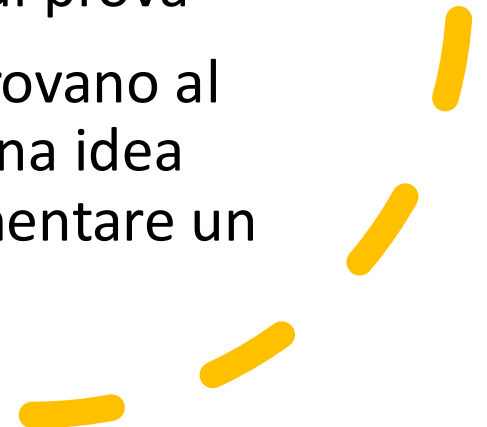
If your status is Single and if the taxable income is	the tax is	of the amount over
at most \$32,000	10%	\$0
over \$32,000	$\$3,200 + 25\%$	\$32,000
If your status is Married and if the taxable income is	the tax is	of the amount over
at most \$64,000	10%	\$0
over \$64,000	$\$6,400 + 25\%$	\$64,000

DIRAMAZIONI ANNIDATE



Preparare casi di prova con senso!

- Black-box testing: un metodo di test che non tiene conto della struttura dell'implementazione. (come se fossimo l'utente: se il programma funziona con ogni tipo di dato → bene, bravo!)
- White-box testing: utilizza le informazioni sulla struttura di un programma.
Eseguire unit test (collaudo) di ciascun metodo
- La copertura del codice è una misura di quante parti di un programma sono state testate.
Guarda ogni ramo if/else per vedere che ognuno di essi è raggiunto da qualche caso di prova
- I casi limite sono casi di prova che si trovano al confine di input accettabili. È una buona idea progettare casi limite prima di implementare un programma.



Preparare
casi di prova
con senso!

- Un piano per la classe TaxReturn
- Ci sono due possibilità per lo stato civile e due scaglioni fiscali per ogni stato, ottenendo quattro casi di prova
- Bisogna verificare un certo numero di casi limite, come un reddito che si trova al confine tra due scaglioni oppure un reddito zero. Bisogna testare un input non valido, come un reddito negativo



Preparare casi di prova con senso!

Elenco di casi di prova e risultati previsti:

Test Case	Married	Expected Output	Comment
30,000	N	3,000	10% bracket
72,000	N	13,200	$3,200 + 25\%$ of 40,000
50,000	Y	5,000	10% bracket
104,000	Y	16,400	$6,400 + 25\%$ of 40,000
32,000	N	3,200	boundary case
0		0	boundary case

Variabili booleane e operatori

Per memorizzare la valutazione di una condizione logica che può essere vera o falsa, si utilizza una variabile booleana.

Il tipo di dati booleano ha esattamente due valori, indicati con false e true.

```
boolean failed = true;
```

Più avanti nel tuo programma, puoi usare il valore per prendere una decisione

```
if (failed) {  
    . . .  
}
```



Una variabile booleana è anche chiamata flag perché può essere up (true) o down (false).

Operatori booleani

- Spesso è necessario combinare valori booleani quando si prendono decisioni complesse
- Un operatore che combina condizioni booleane è detto operatore booleano.
 - L'operatore && è chiamato and. Restituisce vero solo quando entrambe le condizioni sono vere.
 - Il || operatore è chiamato or. Restituisce il risultato vero se almeno una delle condizioni è vera.

Operatori booleani

Table 5 Boolean Operator Examples

Expression	Value	Comment
<code>0 < 200 && 200 < 100</code>	false	Only the first condition is true.
<code>0 < 200 200 < 100</code>	true	The first condition is true.
<code>0 < 200 100 < 200</code>	true	The is not a test for “either-or”. If both conditions are true, the result is true.
<code>0 < x && x < 100 x == -1</code>	<code>(0 < x && x < 100) x == -1</code>	The && operator has a higher precedence than the operator (see Appendix B).
 <code>0 < x < 100</code>	Error	Error: This expression does not test whether x is between 0 and 100. The expression <code>0 < x</code> is a Boolean value. You cannot compare a Boolean value with the integer 100.
 <code>x && y > 0</code>	Error	Error: This expression does not test whether x and y are positive. The left-hand side of && is an integer, x, and the right-hand side, <code>y > 0</code> , is a Boolean value. You cannot use && with an integer argument.
<code>!(0 < 200)</code>	false	<code>0 < 200</code> is true, therefore its negation is false.
<code>frozen == true</code>	frozen	There is no need to compare a Boolean variable with true.
<code>frozen == false</code>	!frozen	It is clearer to use ! than to compare with false.

Not

- Per invertire una condizione utilizzare l'operatore not booleano
- l'operatore ! accetta una singola condizione
- Restituisce vero se tale condizione è falsa
- Restituisce falso se la condizione è vera
- Per verificare se la variabile booleana frozen è falsa:

```
if (!frozen) { System.out.println("Not frozen"); }
```



Iterazioni

Loop

- Una parte di un programma viene ripetuta più e più volte, fino a raggiungere un obiettivo specifico
- Per calcoli che richiedono passaggi ripetuti
- Per l'elaborazione di input costituiti da molti elementi di dati



WHILE Loop

Problema di investimento

Mettiamo \$ 10.000 in un conto bancario che guadagna il 5% di interesse all'anno. Quanti anni ci vogliono perché il saldo del conto sia il doppio dell'investimento originale?

L'algoritmo:

Inizia con un valore dell'anno pari a 0, una colonna per gli interessi e un saldo di \$ 10.000.

anno	interesse	saldo
0		\$10,000

Ripeti i seguenti passaggi finché il saldo è inferiore a \$ 20.000.

Aggiungi 1 al valore dell'anno. Calcola l'interesse come $\text{saldo} \times 0,05$ (ovvero 5 per cento di interesse).

Aggiungi gli interessi al saldo.

Riporta il valore dell'ultimo anno come risposta.

WHILE Loop

Come puoi "Ripetere i passaggi mentre il saldo è inferiore a \$ 20.000?"

Con un'istruzione while

Sintassi

```
while (condizione) {  
    istruzioni;  
}
```

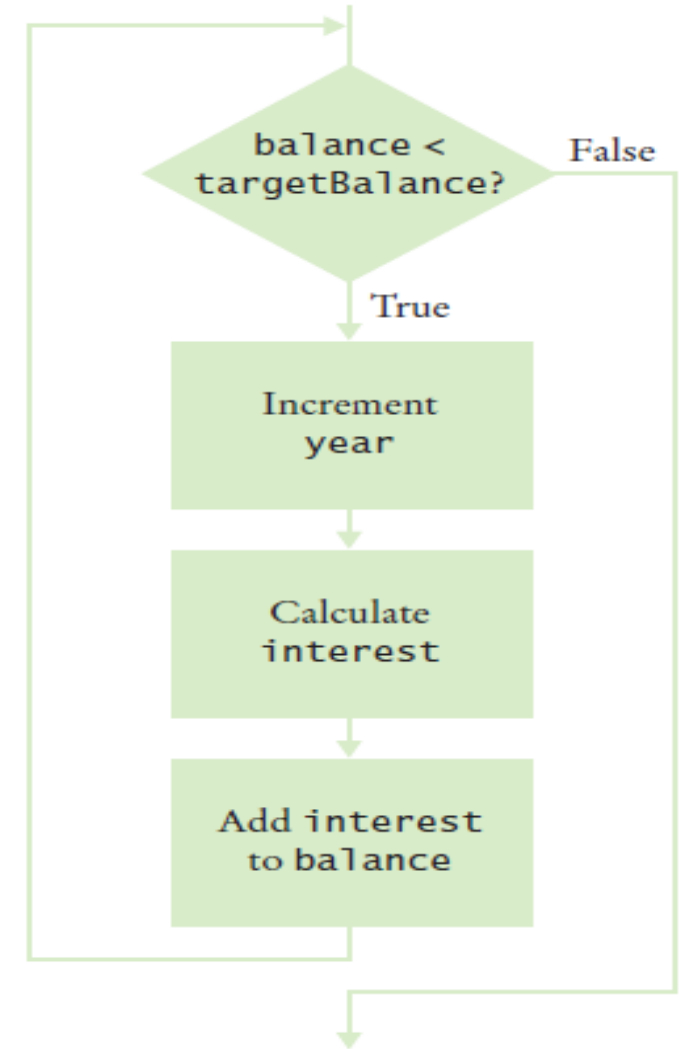
Se la condizione è vera, le istruzioni nel ciclo while vengono eseguite.

A large yellow triangle is positioned in the bottom right corner of the slide, pointing towards the top right.

WHILE Loop

Risultato:

```
while (balance < targetBalance) {  
    year++;  
    double interest = balance * RATE /100;  
    saldo = balance + interest;  
}
```



WHILE Loop

Syntax `while` (*condition*)
 {
 statements
 }

This variable is declared outside the loop and updated in the loop.

If the condition never becomes false, an infinite loop occurs.

This variable is created in each loop iteration.


`double balance = 0;`


`.`
`.`
`.`

`while` (`balance < targetBalance`)
{


`double interest = balance * RATE / 100;`
`balance = balance + interest;`


}

Beware of "off-by-one" errors in the loop condition.
 See page 244.

Don't put a semicolon here!
 See page 182.

These statements are executed while the condition is true.

 Lining up braces is a good idea.
See page 181.

Braces are not required if the body contains a single statement, but it's good to always use them.
 See page 181.

WHILE Loop

Una variabile dichiarata all'interno di un corpo del ciclo:

La variabile viene creata per ogni iterazione del ciclo e rimossa dopo la fine di ogni iterazione

```
while (balance < targetBalance) {  
    year++;  
    double interest = balance * RATE / 100;  
    balance = balance + interest;  
}  
  
// interest non è più dichiarato qui
```

Step by step esecuzione

1 Check the loop condition

balance = 10000

years = 0

```
while (balance < targetBalance)
{
    years++;
    double interest = balance * rate / 100;
    balance = balance + interest;
}
```

The condition is true

2 Execute the statements in the loop

balance = 10500

years = 1

interest = 500

```
while (balance < targetBalance)
{
    years++;
    double interest = balance * rate / 100;
    balance = balance + interest;
}
```

3 Check the loop condition again

balance = 10500

years = 1

```
while (balance < targetBalance)
{
    years++;
    double interest = balance * rate / 100;
    balance = balance + interest;
}
```

The condition is still true

⋮

4 After 15 iterations

balance = 20789.28

years = 15

```
while (balance < targetBalance)
{
    years++;
    double interest = balance * rate / 100;
    balance = balance + interest;
}
```

The condition is
no longer true

5 Execute the statement following the loop

balance = 20789.28

years = 15

```
while (balance < targetBalance)
{
    years++;
    double interest = balance * rate / 100;
    balance = balance + interest;
}
System.out.println(years);
```

Table 1 while Loop Examples

Loop	Output	Explanation
<pre>i = 0; sum = 0; while (sum < 10) { i++; sum = sum + i; Print i and sum; }</pre>	<pre>1 1 2 3 3 6 4 10</pre>	When sum is 10, the loop condition is false, and the loop ends.
<pre>i = 0; sum = 0; while (sum < 10) { i++; sum = sum - i; Print i and sum; }</pre>	<pre>1 -1 2 -3 3 -6 4 -10 . . .</pre>	Because sum never reaches 10, this is an “infinite loop” (see Common Error 6.2 on page 248).
<pre>i = 0; sum = 0; while (sum < 0) { i++; sum = sum - i; Print i and sum; }</pre>	(No output)	The statement <code>sum < 0</code> is false when the condition is first checked, and the loop is never executed.
<pre>i = 0; sum = 0; while (sum >= 10) { i++; sum = sum + i; Print i and sum; }</pre>	(No output)	The programmer probably thought, “Stop when the sum is at least 10.” However, the loop condition controls when the loop is executed, not when it ends (see Common Error 6.1 on page 247).
<pre>i = 0; sum = 0; while (sum < 10) ; { i++; sum = sum + i; Print i and sum; }</pre>	(No output, program does not terminate)	Note the semicolon before the <code>{</code> . This loop has an empty body. It runs forever, checking whether <code>sum < 10</code> and doing nothing in the body.

Esempi

Errori: cicli infiniti

- Esempio: dimenticando di aggiornare la variabile che controlla il ciclo

```
int years = 1;
while (years <= 20)
{
    double interest = balance * RATE / 100;
    balance = balance + interest;
}
```

- Esempio: incrementando invece di decrementare

```
int years = 20;
while (years > 0)
{
    double interest = balance * RATE / 100;
    balance = balance + interest;
    years++;
}
```

Errore off-by-one

- Errore off-by-one: un ciclo esegue una volta di troppo o una di meno.
Esempio:

```
int years = 0;
while (balance < targetBalance)
{
    years++;
    balance = balance * (1 + RATE / 100);
}
System.out.println("The investment doubled after "+ year + " years.");
```

- L'anno dovrebbe iniziare da 0 o 1?
- Il test dovrebbe essere < 0 o \leq ?

Evitare l'off-by-one

Guarda uno scenario con valori semplici:

saldo iniziale: \$ 100

tasso di interesse: 50%

dopo l'anno 1, il saldo è di \$ 150

dopo l'anno 2 è di \$ 225, è oltre \$ 200 quindi l'investimento è raddoppiato dopo 2 anni → il ciclo è stato eseguito due volte, incrementando ogni volta gli anni

Pertanto: l'anno deve iniziare da 0, non da 1.

tasso di interesse: 100%

dopo un anno: il saldo è $2 * \text{saldo iniziale}$ il ciclo dovrebbe fermarsi

Quindi: deve usare $<$ non $<=$

Pensa, non compilare e provare a caso!!!!

Problem solving

Una simulazione dell'esecuzione del codice in cui si eseguono istruzioni e si tiene traccia dei valori delle variabili. Che valore viene visualizzato?

```
int n = 1729;
int sum = 0;
while (n > 0)
{
    int digit = n % 10;
    sum = sum + digit;
    n = n / 10;
}
System.out.println(sum);
```

Problem solving


Primo step:

There are three variables: `n`, `sum`, and `digit`.

<code>n</code>	<code>sum</code>	<code>digit</code>

Secondo step:


The first two variables are initialized with 1729 and 0 before the loop is entered.

```
 int n = 1729;  
int sum = 0;  
while (n > 0)  
{  
    int digit = n % 10;  
    sum = sum + digit;  
    n = n / 10;  
}  
System.out.println(sum);
```

<code>n</code>	<code>sum</code>	<code>digit</code>
1729	0	

Problem solving

Terzo step:




```
int n = 1729;
int sum = 0;
while (n > 0)
{
    int digit = n % 10;
    sum = sum + digit;
    n = n / 10;
}
System.out.println(sum);
```

n	sum	digit
1729	0	
	9	9

Quarto step:

Cross out the old values and write the new ones under the old ones.




```
int n = 1729;
int sum = 0;
while (n > 0)
{
    int digit = n % 10;
    sum = sum + digit;
    n = n / 10;
}
System.out.println(sum);
```

n	sum	digit
1729	0	
172	9	9

Problem solving

Terzo step:




```
int n = 1729;
int sum = 0;
while (n > 0)
{
    int digit = n % 10;
    sum = sum + digit;
    n = n / 10;
}
System.out.println(sum);
```

n	sum	digit
1729	0	
	9	9

Quarto step:

Cross out the old values and write the new ones under the old ones.




```
int n = 1729;
int sum = 0;
while (n > 0)
{
    int digit = n % 10;
    sum = sum + digit;
    n = n / 10;
}
System.out.println(sum);
```

n	sum	digit
1729	0	
172	9	9

Problem solving

Quinto step:

Now check the loop condition again.



```
int n = 1729;  
int sum = 0;  
while (n > 0)  
{  
    int digit = n % 10;  
    sum = sum + digit;  
    n = n / 10;  
}  
System.out.println(sum);
```

n	sum	digit
1729	0	
172	9	9
17	11	2

Sesto step:

Repeat the loop once again, setting digit to 7, sum to $11 + 7 = 18$, and n to 1.

n	sum	digit
1729	0	
172	9	9
17	11	2
1	18	7


Problem solving

settimo step:

Enter the loop for one last time. Now digit is set to 1, sum to 19, and n becomes zero.

n	sum	digit
1729	0	
172	9	9
17	11	2
1	18	1
0	19	1

Ottavo step:




```
int n = 1729;
int sum = 0;
while (n > 0)
{
    int digit = n % 10;
    sum = sum + digit;
    n = n / 10;
}
System.out.println(sum);
```

Because n equals zero,
this condition is not true.

Problem solving

Nono step:

The condition $n > 0$ is now false. Continue with the statement after the loop.

```
int n = 1729;  
int sum = 0;  
while (n > 0)  
{  
    int digit = n % 10;  
    sum = sum + digit;  
    n = n / 10;  
}  
 System.out.println(sum);
```

n	sum	digit	output
1729	0		
172	9	9	
17	17	2	
1	18	1	
0	19	1	19

Decimo step:

La somma che è 19, è stampata in output

FOR loop

- Per eseguire una sequenza di istruzioni un dato numero di volte:
- Si potrebbe usare un ciclo while controllato da un contatore

```
int counter = 1; // inizializzo il contatore
while (counter <= 10)
// controllo il contatore
{
    System.out.println(counter);
    counter++; // aggiorno il contatore
}
```

Potrei usare il ciclo For→

```
for (int counter = 1; counter <= 10; counter++)
{
    System.out.println(counter);
}
```

For loop

Syntax **for** (*initialization; condition; update*)
 {
 statements
 }

These three
expressions should be related.
See page 255.

This **initialization**
happens once
before the loop starts.

The **condition** is
checked before
each iteration.

This **update** is
executed after
each iteration.

```
for (int i = 5; i <= 10; i++)  
{  
    sum = sum + i;  
}
```

The variable **i** is
defined only in this for loop.
See page 257.



This loop executes 6 times.
See page 256.



For loop

L'inizializzazione viene eseguita una volta, prima di entrare nel ciclo. La condizione viene verificata prima di ogni iterazione. L'aggiornamento viene eseguito dopo ogni iterazione.

1 Initialize counter	<pre>for (int counter = 1; counter <= 10; counter++) { System.out.println(counter); }</pre>
2 Check condition	<pre>for (int counter = 1; counter <= 10; counter++) { System.out.println(counter); }</pre>
3 Execute loop body	<pre>for (int counter = 1; counter <= 10; counter++) { System.out.println(counter); }</pre>
4 Update counter	<pre>for (int counter = 1; counter <= 10; counter++) { System.out.println(counter); }</pre>
5 Check condition again	<pre>for (int counter = 1; counter <= 10; counter++) { System.out.println(counter); }</pre>

For loop

- Un ciclo for può contare alla rovescia invece che verso l'alto

```
for (int counter = 10; counter >= 0; counter--) . .
```

- L'incremento o il decremento non devono necessariamente essere in passi di 1

```
for (int counter = 0; counter <= 10; counter = counter + 2) . . .
```

For loop

Se la variabile contatore è definita nell'intestazione del ciclo → Non esiste dopo il ciclo

```
for (int counter = 1; counter <= 10; counter++)  
{  
    . . .  
}  
// counter non è più dichiarata qui
```

Se dichiari la variabile contatore prima del ciclo, Puoi continuare a usarlo dopo il ciclo

```
int counter;  
for (counter = 1; counter <= 10; counter++)  
{  
    . . .  
}  
// counter è ancora dichiarata qui
```

For loop

Se voglio attraversare tutti i caratteri di una stringa:

```
for (int i = 0; i < str.length(); i++)  
{  
    char ch = str.charAt(i);  
    --- ch---  
}
```

La variabile contatore i inizia da 0 e il ciclo termina quando i raggiunge la lunghezza della stringa.

For loop

Per calcolare la crescita del nostro conto di risparmio in un periodo di anni si utilizza un ciclo for perché l'anno variabile inizia da 1 e quindi si sposta con incrementi costanti fino a raggiungere l'obiettivo

```
for (int year = 1; year <= numberOfYears; year++)  
{  
    Update balance.  
}
```

Year	Balance
1	10500.00
2	11025.00
3	11576.25
4	12155.06
5	12762.82

For loop

Table 2 for Loop Examples

Loop	Values of i	Comment
<code>for (i = 0; i <= 5; i++)</code>	0 1 2 3 4 5	Note that the loop is executed 6 times. (See Programming Tip 6.3 on page 260.)
<code>for (i = 5; i >= 0; i--)</code>	5 4 3 2 1 0	Use <code>i--</code> for decreasing values.
<code>for (i = 0; i < 9; i = i + 2)</code>	0 2 4 6 8	Use <code>i = i + 2</code> for a step size of 2.
<code>for (i = 0; i != 9; i = i + 2)</code>	0 2 4 6 8 10 12 14 ... (infinite loop)	You can use <code><</code> or <code><=</code> instead of <code>!=</code> to avoid this problem.
<code>for (i = 1; i <= 20; i = i * 2)</code>	1 2 4 8 16	You can specify any rule for modifying <code>i</code> , such as doubling it in every step.
<code>for (i = 0; i < str.length(); i++)</code>	0 1 2 ... until the last valid index of the string <code>str</code>	In the loop body, use the expression <code>str.charAt(i)</code> to get the <code>i</code> th character.

Do-while loop

- Esegue il corpo di un ciclo almeno una volta ed esegue il test del ciclo dopo l'esecuzione del corpo.
- Utilizzato spesso per la convalida dell'input
- Es: Per forzare l'utente a inserire un valore inferiore a 100

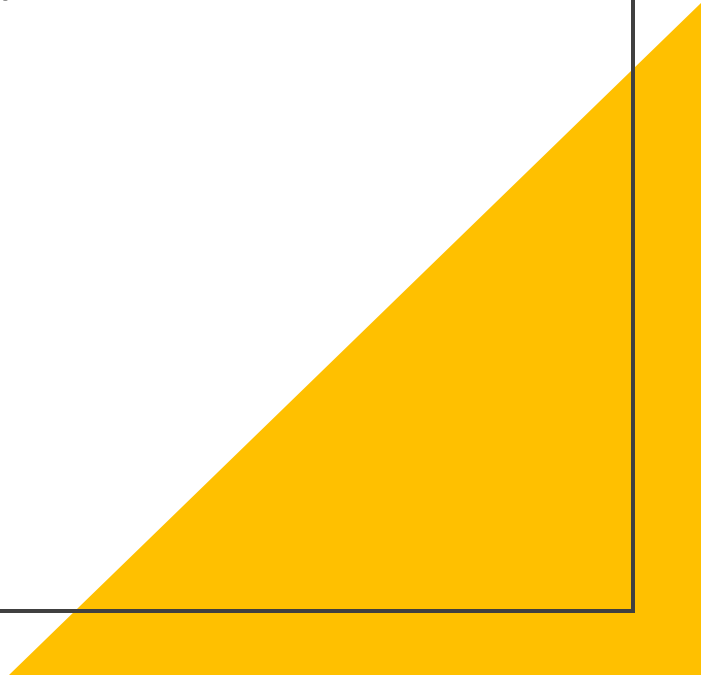
```
int value;  
do  
{  
    System.out.print("Enter an integer < 100: ");    value =  
    in.nextInt();  
}  
while (value >= 100);
```

Valori sentinella

- Un valore sentinella indica la fine di un set di dati, ma non fa parte dei dati.
- Se 0 non può essere parte del set di dati continua ad accettare input finché uno 0 non termina la sequenza
- Se 0 è valido ma nessun valore può essere negativo usa -1 per indicare la terminazione
- Nell'esercito, una sentinella sorveglia un confine o un passaggio. In informatica, un valore sentinella denota la fine di una sequenza di input o il confine tra le sequenze di input.

Valori sentinella

- Per calcolare la media di una serie di stipendi usiamo -1 per indicare la terminazione
- Dentro il ciclo leggiamo l'input. Se l'input non è -1 lo elaboriamo.
- Vediamo → piccolo problema



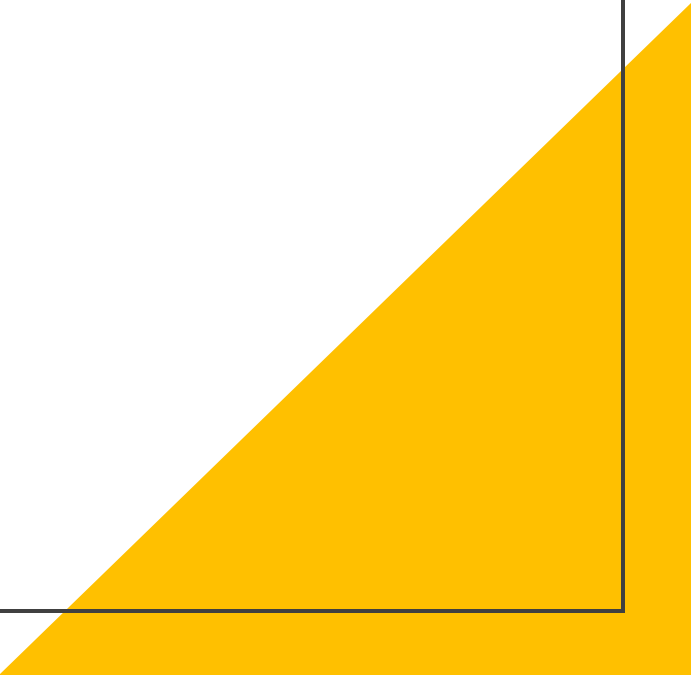
Valori sentinella

Utilizzo di una variabile booleana per controllare un ciclo.

Imposta la variabile prima di entrare nel ciclo

Impostala al contrario per uscire dal ciclo.

```
System.out.print("Enter salaries, -1 to finish: "); boolean  
done = false;  
while (!done){  
    value = in.nextDouble();  
    if (value == -1){  
        done = true;  
    }else{  
        ...  
    }  
}
```

A large yellow right-angled triangle is positioned in the bottom right corner of the slide, pointing towards the top right.