Impariamo a programmare in Java™

Lezione 6

Gr.it

Decisioni

Gestire un conto corrente

```
double balance = 10000; // saldo iniziale
System.out.println("Quanto vuoi prelevare?”);
double amount = console.nextDouble();
balance = balance – amount;
System.out.println("Saldo: " + balance);
```

Il programma precedente consente di prelevare tutto il denaro che si vuole

- ✓ il saldo **`balance`** può diventare negativo

```
balance = balance – amount;
```

- ✓ È una situazione assai poco realistica!
- ✓ Il programma ***deve controllare il saldo ed agire di conseguenza***, consentendo il prelievo oppure no

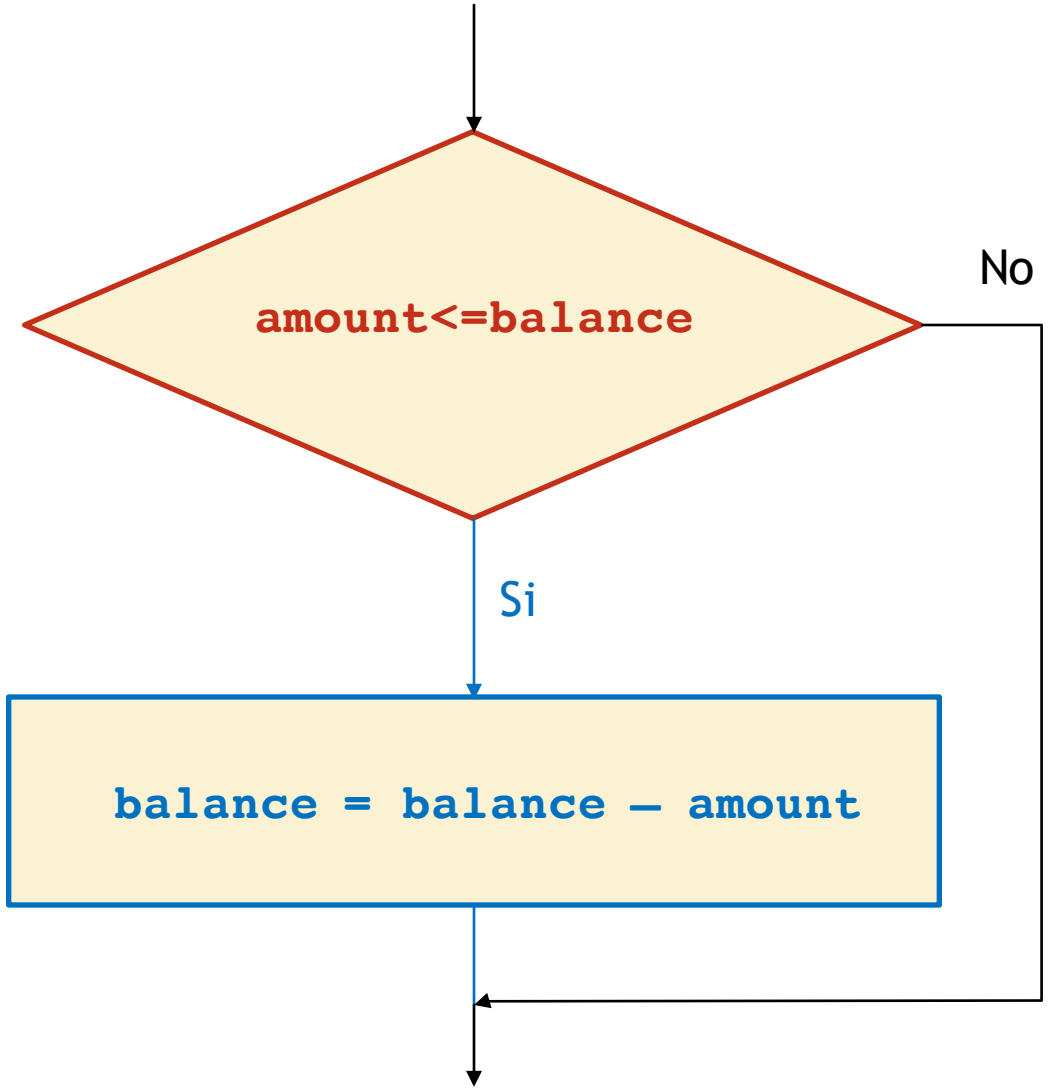
L'enunciato *if*

```
if (amount<=balance)
   balance = balance – amount;
```

L'enunciato ***if*** si usa per realizzare una decisione ed è diviso in due parti

- ✓ una ***verifica***
- ✓ un ***corpo***

Il corpo viene eseguito se e solo se la verifica ha successo



Un nuovo problema

Proviamo ora ad emettere un messaggio d'errore in caso di prelievo non consentito

```
if (amount <= balance)
   balance = balance - amount;
if (amount > balance)
   System.out.println("Conto scoperto");
```

Un nuovo problema

Proviamo ora ad emettere un messaggio d'errore in caso di prelievo non consentito

```
if (amount <= balance)
   balance = balance - amount;
if (amount > balance)
   System.out.println("Conto scoperto");
```

Problema: se si modifica la prima verifica, bisogna ricordarsi di modificare anche la seconda (es. viene concesso un fido sul conto, che può "andare in rosso")

Un nuovo problema

Proviamo ora ad emettere un messaggio d'errore in caso di prelievo non consentito

```
if (amount <= balance)
   balance = balance - amount;
if (amount > balance)
   System.out.println("Conto scoperto");
```

Problema: se si modifica la prima verifica, bisogna ricordarsi di modificare anche la seconda (es. viene concesso un fido sul conto, che può "andare in rosso")

Problema: se il corpo del primo **if** viene eseguito, la verifica del secondo **if** usa il ***nuovo*** valore di **balance**, introducendo un errore logico quando si preleva più della metà del saldo disponibile

La clausola *else*

Per realizzare un'***alternativa***, si utilizza la clausola **else** dell'enunciato **if**

```
if (amount <= balance)
   balance = balance - amount;
else
   System.out.println("Conto scoperto");
```

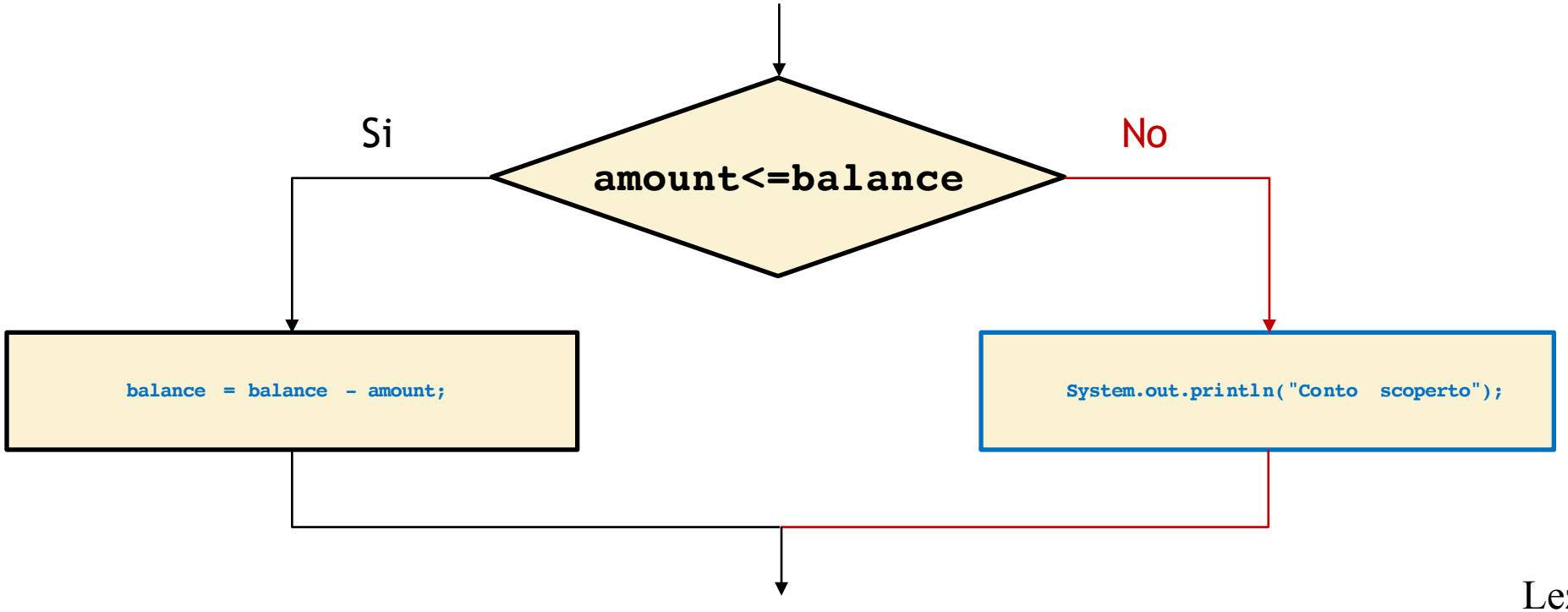
Vantaggio: ora c'è ***una sola verifica***

- ✓ se la verifica ha successo, viene eseguito il ***primo*** corpo dell'enunciato **if/else**
- ✓ ***altrimenti***, viene eseguito il ***secondo*** corpo

Non è un costrutto sintattico ***necessario***, ma è utile

La clausola *else*

```
if (amount <= balance)
   balance = balance - amount;
else
   System.out.println("Conto scoperto");
```



L'enunciato *if*

Sintassi:

```
if (condizione)
   enunciato1;
```

```
if (condizione)
   enunciato1;
else
   enunciato2;
```

- ✓ Scopo: eseguire ***enunciato1*** se e solo se la ***condizione*** è **vera**; se è presente la clausola **else**, eseguire ***enunciato2*** se e solo se la ***condizione*** è **falsa**
- ✓ Spesso il corpo di un enunciato **if** è costituito da ***più enunciati*** da eseguire ***in sequenza***; racchiudendo tali enunciati tra una coppia di parentesi graffe **{ }** si crea un ***blocco di enunciati***, che può essere usato come corpo

```
if (amount <= balance)
{ balance = balance - amount;
System.out.println("Prelievo accordato");}
```

Confrontare Valori

Confrontare Valori

Le ***condizioni*** dell'enunciato **if** sono molto spesso dei ***confronti tra due valori***

```
if (x >= 0)
```

Gli ***operatori di confronto*** si chiamano ***operatori relazionali***

Attenzione: negli operatori costituti da due caratteri ***non*** vanno inseriti spazi intermedi

>	Maggiore
>=	Maggiore o uguale
<	Minore
<=	Minore o uguale
==	Uguale
!=	Diverso

Operatori Relazionali

Fare molta attenzione alla differenza tra l'operatore relazionale == e l'operatore di assegnazione =

```
a = 5; // assegna 5 ad a

if (a == 5)  // esegue enunciato
 enunciato // se a è uguale a 5
```

Confronto di Stringhe

Confronto di Stringhe

Per confrontare stringhe si usa il metodo **equals**

```
if (s1.equals(s2))
```

Per confrontare stringhe ignorando la differenza tra maiuscole e minuscole si usa

```
if (s1.equalsIgnoreCase(s2))
```

Non usare mai l'operatore di uguaglianza per confrontare stringhe!

Usare sempre **equals**

Se si usa l'operatore uguaglianza, il successo del confronto sembra essere deciso in maniera "casuale", in realtà dipende da come è stata progettata la Java Virtual Machine e da come sono state costruite le due stringhe

Attenzione perché NON è un errore di sintassi

Ordinamento Lessicografico

Se due stringhe sono diverse, è possibile conoscere la relazione che intercorre tra loro secondo l'***ordinamento lessicografico***, simile al comune ordinamento alfabetico

Il confronto lessicografico tra stringhe si esegue con il metodo **compareTo**

```
if (s1.compareTo(s2) < 0)
```

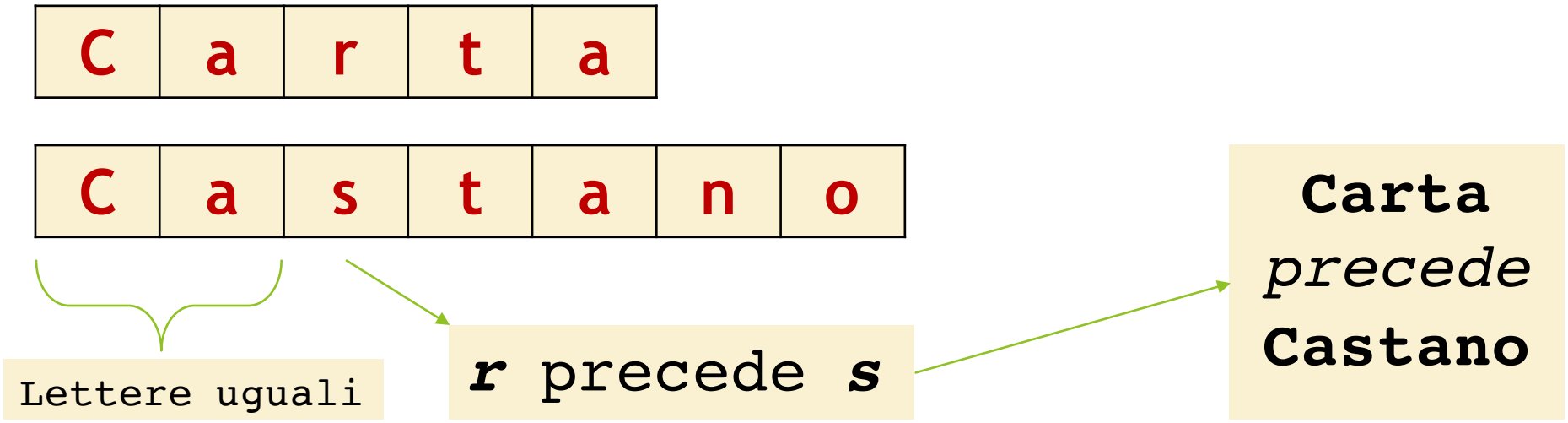
Il metodo **compareTo** restituisce un valore **int**

- ✓ ***negativo*** se **s1** ***precede*** **s2** nell'ordinamento
- ✓ ***positivo*** se **s1** ***segue*** **s2** nell'ordinamento
- ✓ ***zero*** se **s1** e **s2** sono ***identiche***

Confronto Lessicografico

Partendo dall'inizio delle stringhe, si confrontano a due a due i caratteri in posizioni corrispondenti, finché una delle stringhe termina oppure due caratteri sono diversi

- ✓ se una stringa termina, essa precede l'altra
- ✓ se terminano entrambe, sono uguali
- ✓ altrimenti, l'ordinamento tra le due stringhe è uguale all'ordinamento alfabetico tra i due caratteri diversi



Confronto Lessicografico

Il confronto lessicografico genera un ordinamento ***simile*** a quello di un comune dizionario

L'ordinamento tra caratteri in Java è ***simile*** all'ordinamento alfabetico comune, con qualche differenza… anche perché tra i caratteri non ci sono solo le lettere! Ad esempio

- ✓ i numeri precedono le lettere
- ✓ tutte le lettere maiuscole precedono tutte le lettere minuscole
- ✓ il carattere di "spazio bianco" precede tutti gli altri caratteri

Esercizio

Scriviamo un programma che gestisca un conto bancario.
L'utente (unico per ora) accede al sistema inserendo il "Nome", il programma dopo aver controllato il nome controllerà il pin di 5 cifre.
Dopo aver effettuato tutti i controlli chiederà all'utente quale cifra prelevare. Dopo aver controllato che la cifra sia prelevabile stampare a video il valore del prelievo e il saldo rimanente.

Dati del problema:

- ✓ Nome: Gianmaria
- ✓ Pin: 14689
- ✓ Saldo iniziale: 18345,74€

Alternative Multiple

Sequenza di confronti

Se si hanno più di due alternative, si usa una sequenza di confronti

```
if (richter >= 8)
   System.out.println("Terremoto molto forte");
else if (richter >= 6)
   System.out.println("Terremoto forte");
else if (richter >= 4)
   System.out.println("Terremoto medio");
else if (richter >= 2)
   System.out.println("Terremoto debole”);
else
   System.out.println("Terremoto molto debole");
```

Esercizio

Trova l'algoritmo risolutivo per una equazione di secondo grado.

$$ax^2 + bx + c = 0 \Rightarrow \begin{cases} x_{1,2} = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a} & \text{se} \quad b^2 - 4ac \geq 0 \\ x_{1,2} = \dfrac{-b \pm i\sqrt{b^2 - 4ac}}{2a} & \text{se} \quad b^2 - 4ac < 0 \end{cases}$$

Per calcolare le due soluzioni bisogna prima calcolare il valore del determinante "b^2 - 4ac".

Se il determinante è nullo si segnala che le soluzioni sono coincidenti.

Se il determinante è negativo segnalare che le soluzioni sono complesse.

Espressioni Booleane

Tipi di dato *booleano*

Ogni espressione ha un valore

x + 10 espressione aritmetica → valore numerico
x < 10 espressione relazionale → valore booleano

Un'espressione relazionale ha un valore vero o falso (*true o false*)

I valori *true* e *false* non sono numeri, ne oggetti, ne classi: appartengono ad un tipo di dati diverso, detto booleano, dal nome del matematico George Boole (1815-1864), pioniere della logica

I valori booleani sono un tipo di *dato fondamentale* in Java, come quelli numerici

Gli operatori booleani o logici

Gli operatori booleani o logici servono a svolgere operazioni su valori booleani:

```
if (x > 10 && x < 20)
// esegue se x è maggiore di 10 e minore di 20
```

L'operatore **&&** (and, e) combina due o più condizioni in una sola, che risulta vera se e solo se sono tutte vere.

L'operatore **||** (or, oppure) combina due o più condizioni in una sola, che risulta vera se e solo se almeno una vera.

L'operatore **!** (not, non) inverte il valore di un'espressione booleana

Gli operatori booleani o logici

Cosa succede in questa espressione se x=15

```
if ((x > 10 && x < 20) || x > 30)
```

Gli operatori booleani o logici

A	B	A && B
true	true	true
true	false	false
false	qualsiasi	false

A	! A
true	false
false	true

CORTOCIRCUITO

A	B	A || B
true	qualsiasi	true
false	true	true
false	false	false

PROBLEMA

Riprendiamo un problema visto in precedenza, per il quale abbiamo individuato un algoritmo senza averlo ancora realizzato.

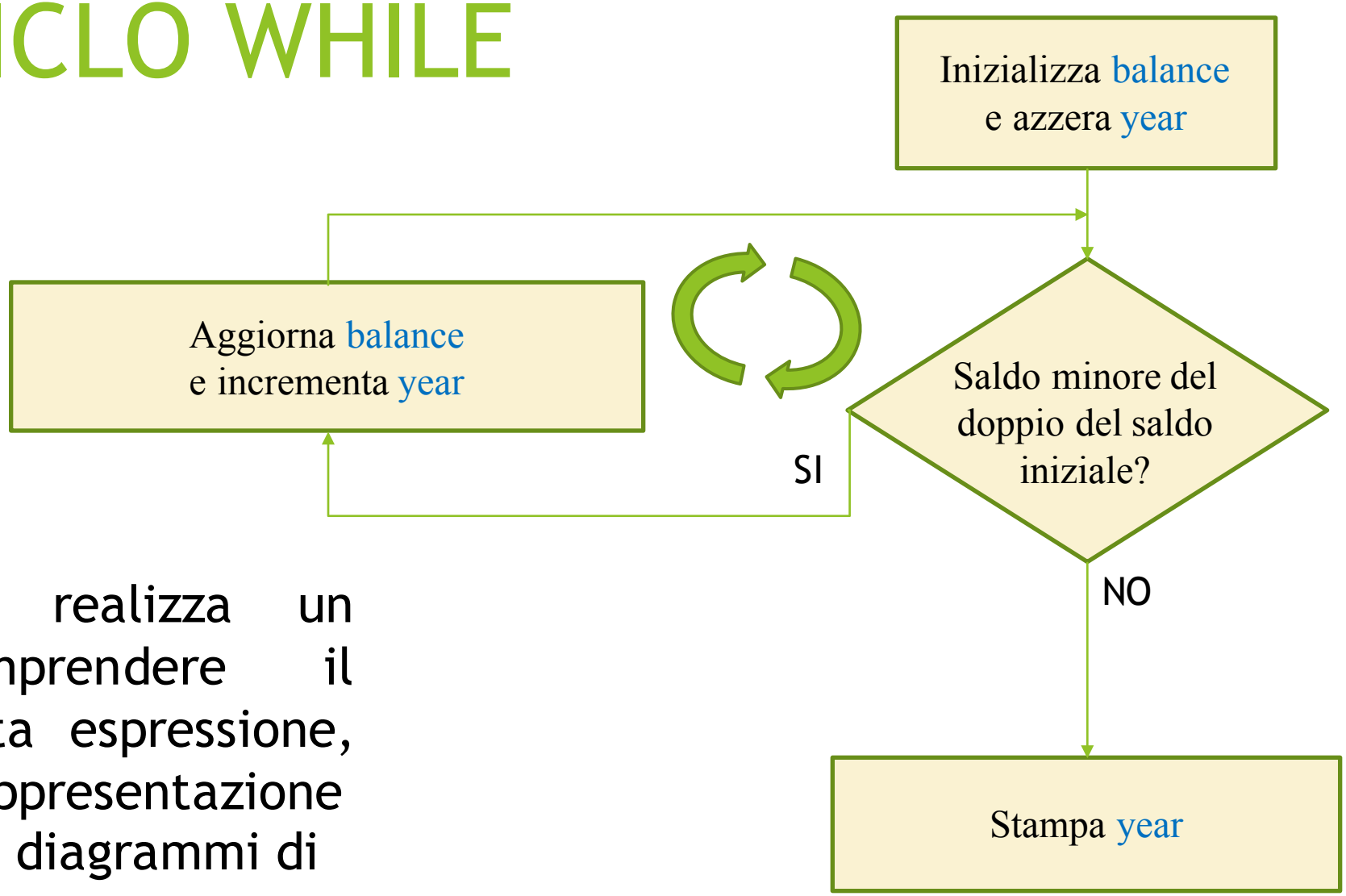
Problema: Avendo depositato ventimila euro in un conto bancario che produce il 5% di interessi all'anno, capitalizzati annualmente, quanti anni occorrono affinché il saldo del conto arrivi al doppio della cifra iniziale?

SOLUZIONE

1. All'anno 0 il saldo 20000
2. Ripetere i passi 3 e 4 finché il saldo minore del doppio di 20000, poi passare al punto 5.
3. Aggiungere 1 al valore dell'anno corrente.
4. Il nuovo saldo il valore del saldo precedente moltiplicato per 1.05 (cioè aggiungiamo il 5%).
5. Il risultato il valore dell'anno corrente.

L'enunciato ***while*** consente la realizzazione di programmi che devono ***eseguire ripetutamente una serie di azioni finché è verificata una condizione***

IL CICLO WHILE



L'enunciato while realizza un ciclo: per comprendere il significato di questa espressione, utile osservare la rappresentazione del codice mediante diagrammi di flusso

SOLUZIONE

```
public class DoubleInvestment
    { public static void main(String[] args)
        { final double initialBalance = 20000; //final indica una costante, non una variabile
        final double rate = 5; //final indica una costante, non una variabile
        double balance = initialBalance;
        int year = 0;
        while (balance < 2 * initialBalance)
            { double interest = balance * rate / 100;
            balance = balance + interest;
            year++; }
    System.out.println("L'investimento  iniziale di “+ initialBalance + “ raddoppia in " +
year + " anni.");}
    }
```

L'ENUNCIATO WHILE

Sintassi

```
while (condizione)
    enunciato
```

Scopo: eseguire un ***enunciato*** finché la ***condizione*** vera

Nota: il ***corpo*** del ciclo **while** può essere un enunciato qualsiasi, quindi anche un ***blocco di enunciati***

CICLI INFINITI

Esistono errori logici che ***impediscono la terminazione di un ciclo***, generando un ***ciclo infinito***

l'esecuzione del programma continua Ininterrottamente.

Bisogna arrestare il programma con un comando del sistema operativo, oppure addirittura riavviare il computer

```
int year = 0;
while (year < 20)
      { double interest = balance * rate / 100;
      balance = balance + interest;
      // qui manca year++
      }
```

Cicli *for*
e
cicli *do*

CICLO *for*

Molti cicli hanno questa forma.

```
i = inizio;
while (i < fine)
      { enunciati
      i++ }
```

Per comodità esiste il ciclo for equivalente.

```
for (i=inizio; i < fine; i++)
      { enunciati}
```

Non necessario che l'incremento sia di una sola unità, ne che sia positivo, ne che sia intero.

ENUNCIATO *for*

Sintassi.

```
for (inizializzazione; condizione; aggiornamento)
      { enunciato }
```

Scopo: eseguire un'inizializzazione, poi ripetere l'esecuzione di un enunciato ed effettuare un aggiornamento finché la condizione vera.

Nota: l'inizializzazione può contenere la definizione di una variabile, che sarà visibile soltanto all'interno del corpo del ciclo.

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

CICLO *do*

Capita spesso di dover eseguire il corpo di un ciclo almeno una volta, per poi ripeterne l'esecuzione se verificata una particolare condizione.

Esempio tipico: leggere un valore in ingresso, ed eventualmente rileggerlo finché non viene introdotto un valore "valido".

Ciclo *do*

Si può usare un ciclo while innaturale.

```
// si usa un'inizializzazione "ingiustificata"
double rate = 0;
while (rate <= 0)
{ System.out.println("Inserire il tasso:");
rate = console.nextDouble();
}
```

Ma per comodità esiste il ciclo do

```
double rate;
do
{ System.out.println("Inserire il tasso:");
rate = console.nextDouble();
} while (rate <= 0);
```