Iterazioni

Problema

- 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?
  - Abbiamo bisogno di un programma che **ripeta** degli enunciati (capitalizzazione degli interessi annuali, incremento del conto degli anni), **finche'** non si realizza la condizione desiderata

Algoritmo che risolve il problema

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

L'enunciato *while*

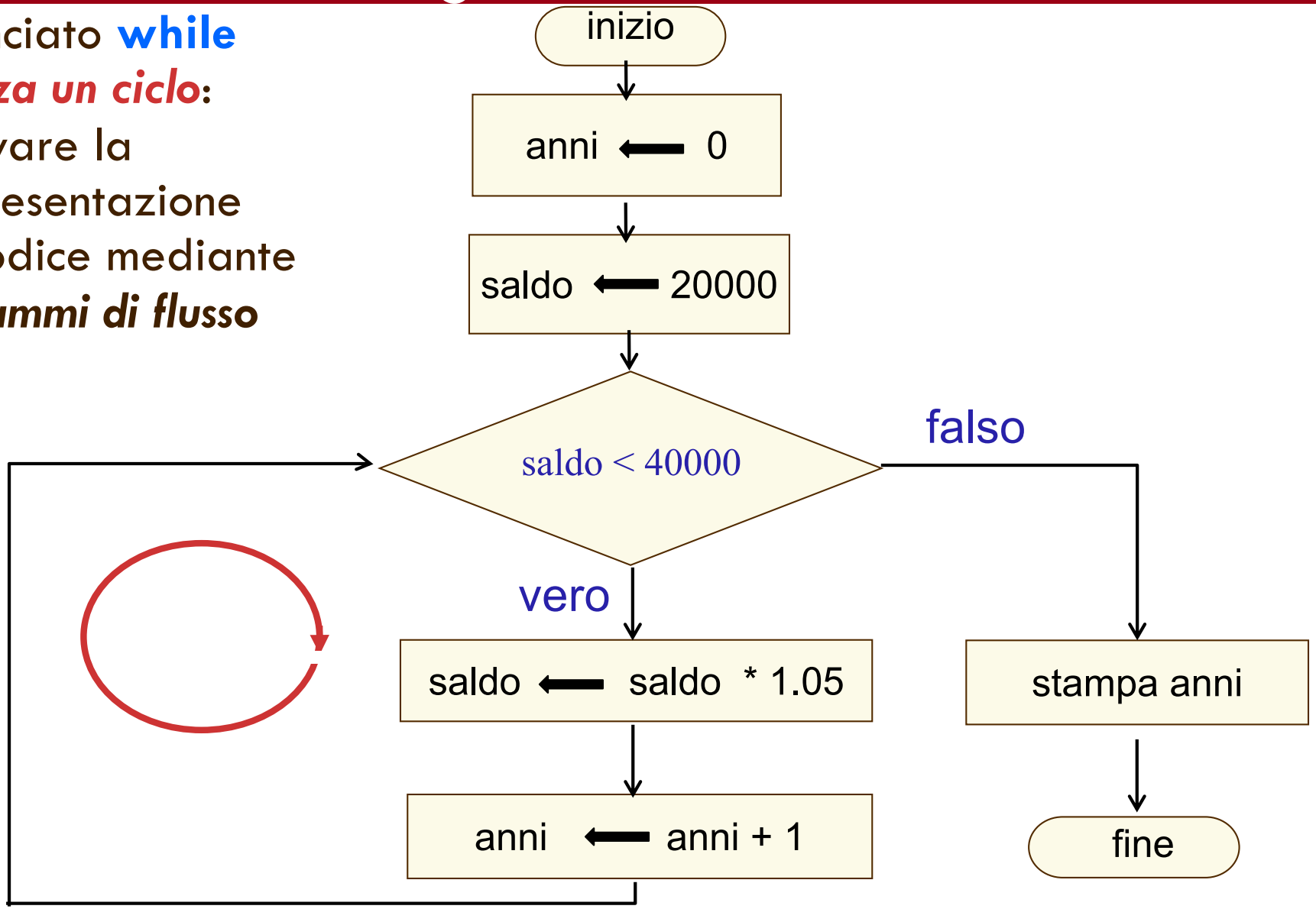
- L'enunciato **while** consente la realizzazione di programmi che devono ***eseguire ripetutamente una serie di azioni finché è verificata una condizione***
- Sintassi:

```
while (condizione)
    enunciato
```

- Scopo: eseguire un enunciato finché la condizione è vera
- Nota: il corpo del ciclo while può essere un enunciato qualsiasi, quindi anche un blocco di enunciati

Rappresentazione dell'algoritmo con diagramma di flusso

L'enunciato **while** ***realizza un ciclo***: osservare la rappresentazione del codice mediante ***diagrammi di flusso***



Soluzione 1: un programma semplice

```
public class DoubleInvestment
{
   public static void main(String[] args)
   {   final double INITIAL_BALANCE = 20000;
       final double RATE = 5;
       double balance = INITIAL_BALANCE;
       int year = 0;

       while (balance < 2*INITIAL_BALANCE)
       {  double interest = balance * RATE / 100;
          balance = balance + interest;
          year++;
       }

       System.out.println("L'investimento "
          + "raddoppia in " + year + " anni");
   }
}
```

Soluzione 2: usando BankAccount

```
public class SimpleInvestmentTester
{
   public static void main(String[] args)
   {
      final double INITIAL_BALANCE = 20000;
      final double rate = 5;
      // definizione e inizializzazione
      BankAccount account = new BankAccount(INITIAL_BALANCE);
      int year = 0;  // anno

      while (account.getBalance() < 2*INITIAL_BALANCE)
      {  double interest = account.getBalance()*rate/100;
         account.deposit(interest);  // modifica saldo
         year++;
      }

      System.out.println("L'investimento "
            + "raddoppia in " + year + " anni");
   }
}
```

Cicli infiniti

- Esistono errori logici che impediscono la terminazione di un ciclo, generando un ciclo infinito
- l'esecuzione del programma continua ininterrottamente

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

Cicli infiniti

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

- In questi casi bisogna arrestare il programma con un comando del sistema operativo, oppure addirittura riavviare il computer

CTRL-C

Cicli for

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 è necessario che l'incremento sia di una sola unità, né che sia positivo, né che sia intero

```
for (double x = 2; x > 0; x = x - 0.3){
enunciati
}
```

L'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 è 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
```

Esempio: invertire una stringa

ciao => oaic

- Idea: in una stringa invertita il primo carattere diventa l'ultimo, il secondo il penultimo, il terzo il terz'ultimo, e cosi' via.
- Posso fare una scansione della stringa, da primo all'ultimo carattere e costruire man mano il reverse
  - Assegno alla stringa reverse il primo carattere s1
  - Quando leggo s2 devo ottenere la stringa s2s1, quindi concateno s2 in testa a reverse (che ora vale s2s1)
  - Quando leggo s3 lo concateno in testa a reverse (che ora vale s3s2s1), etc

Esempio: invertire una stringa

```
import java.util.Scanner;

public class ReverseTester
{  public static void main(String[] args){
      Scanner in = new Scanner(System.in);
      System.out.println("Inserire una stringa");
      String s = in.nextLine();
      String reverseS = "";
      for(int i=0; i<s.length(); i++)
      {  // seleziono il carattere in posizione i
         char ch = s.charAt(i);
         // aggiungo all'inizio
         reverseS = ch + reverseS;
      }
      System.out.println(s+" invertita e'
   "+reverseS);
      in.close();
   }
}
```

Osservazione

```
import java.util.Scanner;

public class ReverseTester{
   public static void main (String[] args){
      Scanner in = new Scanner(System.in);
      System.out.println("Inserire una stringa");
      String s = in.nextLine();
      String reverseS = "";
      int sLength = s.length();
      for(int i=0; i<sLength; i++)
      {  // seleziono il carattere in posizione i
         char ch = s.charAt(i);
         // aggiungo all'inizio
         reverseS = ch + reverseS;
      }
      System.out.println(s+" invertita e'"+reverseS);
   }
}
```

Esecuzione

```
s = "ciao";  reverseS = "";
for (int i = 0; i < s.length(); i++)
{  char ch = s.charAt(i);
   reverseS = ch + reverseS;
}
```

```
i = 0
   ch = s.charAt(0) = 'c'
   reverseS = 'c' + "" = "c"
```

```
i = 1
   ch = s.charAt(1) = 'i'
   reverseS = 'i' + "c" = "ic"
```

```
i = 2
   ch = s.charAt(2) = 'a'
   reverseS = 'a' + "ic" = "aic"
```

```
i = 3
   ch = s.charAt(3) = 'o'
   reverseS = 'o' + "aic" = "oaic"
```

Visibilità delle variabili

- Se il valore finale di una variabile di controllo del ciclo deve essere visibile al di fuori del corpo del ciclo, bisogna definirla **prima** del ciclo
- Poiché una variabile definita nell'inizializzazione di un ciclo non è più definita dopo il corpo del ciclo, è possibile usare di nuovo **lo stesso nome** in altri cicli

```
double b =  … ;
for(int i=1; i<10 && b<3; i++)
{  int c = . . .
   modifica b e c
}
// qui b e' visibile, mentre i e c non lo sono
System.out.println(b); // ok
System.out.println(i); // errore in compilazione
System.out.println(c); // errore in compilazione
//NB: i e c non sono piu' visibili: posso ridefinirle
for(int i=3; i>0; i--){
   System.out.println(i); // ok
}
```

Acquisire in input una sequenza di dati

Obiettivo

- Elaborare una quantità arbitraria di dati
  - Se è noto il numero di dati
    - Acquisire il numero n di dati (variabile int)
    - Usare un ciclo per chiedere un numero equivalente di inserimento di dati
  - Se non è noto il numero di dati
    - Sentinella
      - Valore non valido
      - Tipo diverso di dato
    - Segnale di fine inserimento

Leggere una sequenza di dati

- Molti problemi di elaborazione richiedono la ***lettura di una sequenza di dati in ingresso***
  - ad esempio: calcolare la somma di **DIECI** numeri

```
import java.util.Scanner;
public class Sum1
{  public static void main(String[] args)
   {
      Scanner console = new Scanner(System.in);
      double sum = 0;
      for(int i=0; i<10; i++)
      {
         sum = sum + console.nextDouble();
      }
      System.out.println("Somma: " + sum);
      console.close();
   }
}
```

Leggere una sequenza di dati

- Potevo usare anche il while

```
import java.util.Scanner;
public class Sum1
{  public static void main(String[] args)
   {
      Scanner console = new Scanner(System.in);
      double sum = 0;
      int i=0;
      while(i<10)
      {  i++;
         sum = sum + console.nextDouble();
      }
      System.out.println("Somma: " + sum);
      console.close();
   }
}
```

Leggere una sequenza di dati

- Per rendere il programma più flessibile, si può **chiedere all'utente la lunghezza della sequenza**

```
import java.util.Scanner;
public class Sum2
{
   public static void main(String[] args)
   {  Scanner console = new Scanner(System.in);
      System.out.println("Quanti numeri?");
      final int NUM = console.nextInt();
      double sum = 0;
      for(int i=0; i<NUM; i++)
      {  sum = sum + console.nextDouble();
      } // è ancora un "ciclo a contatore"
        // ma il conteggio non è più prefissato
      System.out.println("Somma: " + sum);
      console.close();
   }
}
```

Leggere una sequenza di dati

- Può succedere che l'utente non sappia ***quanti saranno*** i dati che fornirà in ingresso
  - ad esempio perché li sta acquisendo a sua volta da uno strumento di misura e la durata dell'esperimento non è prefissata
  - oppure, più in generale, perché è scomodo o impossibile contarli prima di iniziare a scriverli

Leggere una sequenza di dati

- **Problema:** leggere una sequenza di dati in ingresso *finché i dati non sono finiti*
- **Soluzione:** usare un **approccio "a sentinella"**
  - L'utente introduce **un dato "non valido"** che segnala la fine dei veri dati
    - Ad esempio, un numero **negativo** per terminare una sequenza di numeri **positivi**

Calcolare la somma di numeri positivi

```
import java.util.Scanner;
public class Sum3
{  public static void main(String[] args)
   {  Scanner console = new Scanner(System.in);
      double sum = 0;
      System.out.println("Introduci un numero da
         sommare (numero negativo per terminare)");
      double next = console.nextDouble();
      // usa una sentinella non positiva
      while (next > 0)
      {  sum = sum + next;
         System.out.println("Introduci un numero…");
         next = console.nextDouble();
      }
      System.out.println("Somma: " + sum);
      console.close();
      // controlliamo cosa avviene nel caso limite:
      // se il primo valore è subito la sentinella,
      // visualizza zero: corretto
   }
}
```

Approccio a sentinella

- **Problema:** non sempre è possibile individuare tali valori "non validi", cioè ***non appartenenti al dominio di elaborazione del problema***
  - Un programma che somma numeri deve poter accettare qualunque numero…
- Si può terminare la sequenza di dati con una stringa non numerica (cioè usando **una sentinella di tipo diverso dai dati**) ma questo complica il codice che legge i numeri, perché non si può usare **nextDouble ma devo leggere una generica stringa e convertirla!**

Sentinella di tipo diverso dai dati

```
import java.util.Scanner;
public class Sum4
{  public static void main(String[] args)
   {  Scanner console = new Scanner(System.in);
      double sum = 0;
      final String SENTINELLA = "STOP";
      boolean done = false; // ho finito?
      while (!done) // strategia usata molto spesso
      {  System.out.println("Prossimo numero (o STOP per
            terminare)");
         String read = console.nextLine();
         if (read.equals(SENTINELLA)){
            done = true; // il ciclo terminerà
         }
         else{
            // devo convertire la stringa read in numero
            sum = sum + Double.parseDouble(read);
         }
      }
      System.out.println("Somma: " + sum);
      console.close();
   }
}
```

Leggere una sequenza di dati

- A volte, però, un programma non può porre limiti ragionevoli al dominio… in alcuni casi non si può definire una sentinella!
  - es. un programma che chiede all'utente di digitare un testo generico su più righe, come un romanzo… che sentinella uso ?!?!

Leggere una sequenza di dati

- **Problema:** leggere una sequenza di dati in ingresso *finché i dati non sono finiti*

- **Soluzione senza sentinella:** i metodi

  hasNext( )
  hasNextLine( )
  hasNextInt( )
  hasNextDouble( )

della classe **Scanner** restituiscono il valore **false** ***se e solo se i dati in ingresso sono terminati***

oppure se il tipo di dati inserito e' diverso da quello specificato dai metodi hasNextInt e hasNextDouble

Leggere una sequenza di dati

```
Scanner console = new Scanner(System.in);
...
while (console.hasNextDouble())
{   double nextValue = console.nextDouble();
    sum = sum + nextValue;
}
console.close();
```

- Quando i dati in ingresso sono terminati, il metodo `hasNextDouble()` restituisce il valore booleano `false` e, di conseguenza, l'esecuzione del ciclo si conclude
- **Problema ancora aperto:**
  come fa l'utente a comunicare che i dati sono terminati?
  Potrebbe non esserci un dato pronto da leggere semplicemente perché l'utente "ci sta pensando"…

Leggere una sequenza di dati

- Dopo aver messo in esecuzione il programma, si introducono tramite la tastiera i dati che il programma deve acquisire
- Al termine dell'elenco, occorre ***comunicare al sistema operativo*** che l'input da tastiera, destinato al programma in esecuzione, è terminato
  - in una finestra DOS/Windows bisogna digitare **Ctrl+Z** e premere Invio
  - in una ***shell*** di Unix bisogna digitare **Ctrl+D**
- Il flusso **System.in** di Java ***non leggerà questi caratteri speciali***, ma riceverà dal sistema operativo la segnalazione che l'input è terminato e, di conseguenza, **hasNextDouble()** restituirà **false**

Leggere una sequenza di dati

- I metodi

  hasNext( )
  hasNextLine( )
  hasNextInt( )
  hasNextDouble( )

  della classe **Scanner** restituiscono il valore **false** **se e *solo se i dati in ingresso sono terminati***

- Sono metodi **bloccanti** come i corrispondenti metodi **next…**
- Anche se li esaminano, **non "consumano" i dati in ingresso**, che rimangono disponibili per i metodi **next…**

Take home message

- Cicli: while
  - Fare molta attenzione con la condizione di terminazione
    - Se si "pianta": CTRL+C
- Utilizzo di cicli per leggere sequenze di input
  - Leggo sentinella
    - Valore non valido
    - Leggo tipo diverso di dato e uso metodi per convertire in numero
  - Uso metodi predicativi: hasNextLine(), hasNext()
    - Blocco il flusso con CTRL+D, CTRL+Z