Lezione XXXI
Lu 21-Nov-2005

Esercitazione

Appelli di fine trimestre

Sono aperte le liste di iscrizione

- primo appello 13-Dic-2005
- secondo appello 9-Genn-2006

http://sis.dei.unipd.it/cgi-bin/info/DEI/printaggiungistudente

Cambio orario

Domani Ma 22-Nov-2005
FI1 h 14:20
MAT_A h 16:15

Esercitazione: Compito 24.9.2003

- Frequenza delle parole in un testo

- E' data una interfaccia D che definisce un dizionario di coppie composte da una stringa di caratteri (la chiave) e da un attributo (un numero intero).
- Le coppie sono definite della seguente classe

```
// Coppia.java -- coppie per il dizionario D
public class Coppia
{  public String ch;
   public int att;

   public Coppia(String s, int i)
   {
     ch = s;
     att = i;
   }
}
```

- mentre il dizionario è definito dalla seguente interfaccia:

```
// D.java --  dizionario
import java.util.NoSuchElementException
public interface Dizionario
{
  /**
      inserisce la coppia <chiave, attributo> */
  void inserisci(String chiave, int attributo);

  /** restituisce vero se la chiave x e' presente, falso
      altrimenti  */
  boolean presente(String x);

  /* restituisce l'attributo associato alla chiave x se
     presente, altrimenti lancia l'eccezione
     java.util.NoSuchElementException */
  int cerca(String x) throws NoSuchElementException;

  /* estrae in un array, di lunghezza pari alle
     dimensioni del dizionario, il contenuto del
     dizionario */
  Coppia[] toArray();
}
```

Esercitazione

- Scrivere il codice di una classe D che realizza l'interfaccia Dizionario.
- Scrivere una classe "Main" di prova che, utilizzando un esemplare della classe D, esegua il conteggio della frequenza delle parole (per "parola" si intende una sequenza di caratteri separati da spazi o fineriga) presenti in un file di testo, utilizzando il seguente algoritmo

  - crea un esemplare d del dizionario vuoto
  - finche' ci sono parole in ingresso
    - estrai una parola
    - se la parola non è presente nel dizionario
      - inserisci in d la coppia <parola, 1>
    - altrimenti
      - inserisci nel dizionario la coppia <parola, n+1> essendo n l'attributo corrente di parola
  - estrai il contenuto del dizionario
  - trasferisci sull'uscita standard il valore delle coppie presenti.

Esercitazione

- Al termine della prova il candidato dovra' lasciare nella directory di lavoro i seguenti file:
  - Dizionario.java
  - Coppia.java — la classe che realizza la coppia
  - D.java — la classe che realizza il dizionario
  - Main.java — il programma di prova per il conteggio della frequenza
- Durante la valutazione degli elaborati le classi sviluppate verranno provate eseguendo il conteggio delle parole presenti nel file che contiene il tema d'esame con il comando:
  - **`$java Main < compito.txt`**
- Si possono usare nello svolgimento dell'elaborato solo le seguenti classi della libreria standard:
  - tutte le classi del package java.io
  - le classi Scanner e NoSuchElementException del package java.util

Esercitazione: quale soluzione?

- Soluzioni possibili per il Dizionario

- Array non ordinato
  - inserimento O(n) con verifica di univocità della chiave
  - ricerca O(n)
  - verifica presenza O(n)

- Array Ordinato
  - inserimento O(n)
  - ricerca O(log n)
  - verifica presenza O(log n)

- Tabella hash con bucket
  - inserimento O(n/M)
  - ricerca O(n/M)

Classe Coppia

```
// Coppia.java -- coppie per il dizionario D
public class Coppia
{
   public String ch;
   public int att;

   // costruttore
   public Coppia (String s, int i)
   {
      ch = s;
      att = i;
   }
}
```

Soluzione con array non ordinato

```
// D.java --  dizionario con array non ordinato
import java.util.NoSuchElementException;
public class FixedArrayD implements Dizionario
{ private final int ARRAY_DIM = 1000;
  private Coppia[] v;   // array riempito parzialmente
  private int vSize;    // num. coppie nel dizionario

  //array di lunghezza fissa
  public FixedArrayD()
  { v = new Coppia[ARRAY_DIM];
    vSize = 0;
  }

  public void inserisci(String chiave,      //O(n)
                        int attributo)
  { v[vSize] = new Coppia(chiave, attributo);
    int i;
    for (i = 0; !chiave.equals(v[i].ch); i++)
      ;
    if (i == vSize)
      vSize++;
    else
      v[i] = v[vSize];
  }
```

Sintassi valida perché le variabili di esemplare della classe Coppia sono pubbliche

Attenzione: non c'è il ridimensionamento dinamico

```
  public boolean presente(String x)        //O(n)
  {
     for (int i = 0; i < vSize; i++)
       if (x.equals(v[i].ch))
         return true;
     return false;
  }

  public int cerca(String x) throws        //O(n)
    NoSuchElementException
  {
    for (int i = 0; i < vSize; i++)
      if (x.equals(v[i].ch))
        return v[i].att;
    throw new NoSuchElementException();
  }

  public Coppia[] toArray() //ritorna copie! O(n)
  {
    Coppia[] tmp = new Coppia[vSize];
    for (int i = 0; i < vSize; i++)
      tmp[i] = new Coppia(v[i].ch,v[i].att); //copia
    return tmp;
  }
}
```

Incapsulamento

- Le variabili di esemplare della classe Coppia sono definite pubbliche!
  - E' una violazione del principio dell'incapsulamento
  - E' lecito perche' Java non obbliga a definire private le variabili di esemplare

- Classe Coppia: interna o pubblica?
  - la classe Coppia non può essere realizzata come una classe interna alla classe D, perchè il metodo `toArray()` ritorna un riferimento a un array di oggetti di classe Coppia; la classe Coppia deve essere quindi una classe accessibile all'esterno della classe D, quindi pubblica o con accessibilità di pacchetto.
  - in generale, quando una classe è usata solo all'interno di un'altra classe, è buona prassi definire la prima come classe interna

- I dati memorizzati nella classe D sono sicuri anche se la classe Coppia ha variabili di esemplare pubbliche?
  - Sì, perchè il metodo toArray() non restituisce i riferimenti agli oggetti di classe Coppia memorizzati in un contenitore di classe D, ma restituisce un array di riferimenti a copie, a oggetti cioè diversi creati allo scopo che hanno lo stesso stato (stessi valori delle variabili di esemplare) degli oggetti del contenitore

Incapsulamento

- E’ opportuno definire pubbliche le variabili di esemplare della classe Coppia?
  - **No**, il codice della classe Coppia è molto debole. Se ri-usata in altri contesti (in altre classi, che realizzano metodi che ritornano oggetti di classe Coppia) rende vulnerabili quelle classi.

- E allora, come realizzare la classe Coppia?

```
public class Coppia
{ private String ch;
  private int att;

  public Coppia(String s, int i)
  {  ch = s;
     att = i;
  }

  public String chiave() {return ch;}
  public int attributo() {return att;}
}
```

Incapsulamento

```
public void inserisci(String chiave, int attributo)
{ v[vSize] = new Coppia(chiave, attributo);
  int i;
  for (i = 0; !chiave.equals(v[i].chiave()); i++)
    ;
  if (i == vSize)  vSize++;
  else  v[i] = v[vSize];
}
public boolean presente(String x)
{ for (int i = 0; i < vSize; i++)
    if (x.equals(v[i].chiave()))   return true;
  return false;
}
public int cerca(String x) throws NoSuchElementException
{ for (int i = 0; i < vSize; i++)
    if (x.equals(v[i].chiave()))
      return v[i].attributo();
  throw new NoSuchElementException();
}
```

Incapsulamento

```
public Coppia[] toArray()
{
  Coppia[] tmp = new Coppia[vSize];
  for (int i = 0; i < vSize; i++)
    tmp[i] = new Coppia(v[i].chiave(), v[i].attributo());
  return tmp;
}
```

Incapsulamento

- Nel rispetto del principo dell'incapsulamento è accettabile anche la seguente classe:

```
public class Coppia
{ final public String ch;
  final public int att;

  public Coppia(String s, int i)
  {  ch = s;
     att = i;
  }
}
```

- Le variabili di esemplare costanti sono inizializzate dal costruttore al momento dell'istanza dell'oggetto e successivamente non possono essere modificate
- Nella classe non si possono scrivere metodi modificatori.

Soluzione con tabella hash

```
// D.java --  dizionario con tabella hash
import java.util.NoSuchElementException;

public class HashTableD implements Dizionario
{
  // parte privata della classe
  private final int HASHTABLE_DIM = 97;
  private ListNode[] v;
  private int size; //num. elementi nel dizionario

  public HashTableD() //il costruttore
  {
    v = new ListNode[HASHTABLE_DIM];
    size = 0;

    //iniz. dei bucket con liste concatenate vuote
    for (int i = 0; i < HASHTABLE_DIM; i++)
      v[i] = new ListNode(null, null); //header
  }
```

```
//calcola l'hashCode della stringa String s
private int hash(String s)
{  final int HASH_MULTIPLIER = 31;

   int h = 0;
   for (int i = 0; i < s.length(); i++)
   {
      h = HASH_MULTIPLIER * h + s.charAt(i);
      h = h % HASHTABLE_DIM;
   }

   return h;
}

//classe interna
class ListNode
{
   Coppia element;
   ListNode next;

   ListNode(Coppia e, ListNode n)
   {  element = e;
      next = n;
   }
}
```

Ritorna un intero fra zero e HASHTABLE_DIM – 1, compresi

Soluzione con tabella hash

```
/*
 se la chiave String x non e' presente, ritorna
 l'ultimo nodo del bucket, altrimenti ritorna il
 riferimento al nodo che precede quello contenente la
 chiave String x.
 NB: se la lista concatenata e' vuota ritorna
  l'header del bucket
*/
private ListNode trovaNodo(String x)
{
   ListNode tmp = v[hash(x)]; //header del bucket

   while (tmp.next != null)
     if ((tmp.next.element.ch).equals(x))
       break;
     else
       tmp = tmp.next;

   return tmp;
}
```

```
public void inserisci(String chiave, int attributo)
{  ListNode nodo = trovaNodo(chiave);
   Coppia nuovaCoppia = new Coppia(chiave, attributo);

   if (nodo.next != null)
      nodo.next.element = nuovaCoppia;
   else
   {  nodo.next = new ListNode(nuovaCoppia, null);
      size++;
   }
}

public boolean presente(String x)
{
   ListNode nodo = trovaNodo(x);
   return nodo.next != null;
}

public int cerca(String x) throws NoSuchElementException
{
  ListNode nodo = trovaNodo(x);
    if (nodo.next != null)
      return nodo.next.element.att;

   throw new NoSuchElementException();
}
```

Soluzione con tabella hash

```
public Coppia[] toArray() // restituisce copie!!!
{
  Coppia[] c = new Coppia[size];

  int j = 0;
  for (int i = 0; i < HASHTABLE_DIM; i++)
  {
    ListNode nodo = v[i];
    while (nodo.next != null)
    {
      String chiave = nodo.next.element.ch;
      int attributo = nodo.next.element.att;
      Coppia copia = new Coppia(chiave, attributo);
      c[j++] = copia; //copia della Coppia

      nodo = nodo.next;
    }
  }

  return c;
}
```

Lista Concatenata

- La realizzazione dei bucket che è stata presentata nell'esercizio è **sicura**?
  - **la risposta è sì**, perchè i bucket sono usati solo internamente alla classe HashTableD
    - i metodi pubblici non restituiscono mai riferimenti a bucket e non hanno bucket come parametri espliciti
- In questa condizione, abbiamo potuto realizzare una versione della **lista concatenata** più semplice di quella presentata a lezione
  - quando è richiesto che sia una **classe pubblica**, allora deve essere realizzata come abbiamo fatto a lezione, con la complicazione dell'iteratore, se è necessaria la funzionalità di accedere agli elementi centrali della lista, senza l'iteratore se si accede in testa e in coda (come nel caso della realizzazione dei contenitori Stack e Queue)
  - Quando la lista concatenata può essere definita come **classe interna**, allora è sicuro scrivere il codice come presentato nell'esercizio

Soluzione con array ordinato

```
// Coppia.java -- coppie per il dizionario D
// Ordinamento naturale: per chiave
public class Coppia implements Comparable
{
   final public String ch;
   final public int att;

   public Coppia (String s, int i) {ch = s; att = i; }

   public int compareTo(Object obj)
   {
      Coppia coppia = (Coppia) obj;
      return ch.compareTo(coppia.ch);
   }
}
```

```
// D.java --  dizionario con array ordinato
import java.util.NoSuchElementException;
public class SortedFixedArD
{  private final int ARRAY_DIM = 1000;
   private Coppia[] v;
   private int vSize;

   public SortedFixedArD() //costruttore
   { v = new Coppia[ARRAY_DIM];
     vSize = 0;
   }

   private int trovaIndice(String x) // O(log n)
   {  int da = 0;
      int a = vSize - 1;
      Coppia coppia = new Coppia(x,1);//coppia da cercare
      while (da <= a) //bisezione iterativa
      {  int m = (da + a) / 2;
         if (c.compareTo(v[m]) == 0) //trovato
           return m;
         else if (c.compareTo(v[m]) < 0 )
           a = m -1; //cerco a sinistra
         else
           da = m + 1; //cerco a destra
      }
      return -1; //non trovato
   }
```

Soluzione con array ordinato

```
//O(n)
public void inserisci(String chiave, int attributo)
{
  int i = trovaIndice(chiave); //verifica presenza
  Coppia c = new Coppia(chiave, attributo);

  if (i == -1)
  { int j;
    for(j = vSize; j > 0 && c.compareTo(v[j-1]) < 0; j--)
      v[j] = v[j-1];

    v[j] = c;
    vSize++;
  }
  else
    v[i] = c;
}

public boolean presente(String x) // o(log n)
{
   int i = trovaIndice(x);
   return i != -1;
}
```

Soluzione con array ordinato

```
//O(log n)
  public int cerca(String x) throws NoSuchElementException
  {
    int i = trovaIndice(x);
    if (i != -1)
      return v[i].att;

    throw new NoSuchElementException("Cerca " + x);
  }

  public Coppia[] toArray() //O(n)
  {
     Coppia[] tmp = new Coppia[vSize];

     for (int i = 0; i < vSize; i++)
       tmp[i] = new Coppia(v[i].ch, v[i].att);

     return tmp;
   }
}
```

Classe di prova: Main

```
import java.util.Scanner;
import java.util.NoSuchElementException;

public class MainD
{ public static void main(String[] args)
  { Scanner in = new Scanner(System.in);
    Dizionario htd = new HashTableD();

    while (in.hasNext())
    {
      String parola = in.next();

      if (!htd.presente(parola))
        htd.inserisci(parola, 1);
      else
        htd.inserisci(parola, htd.cerca(parola) + 1);
    }

    System.out.println("\nStampa da HashTableD");
    Coppia[] ar = htd.toArray();
    for (int i = 0; i < ar.length; i++)
      System.out.println(ar[i].ch + " " + ar[i].ch );
  }
}
```

Lezione XXXII
Ma 22-Nov-2005

Ultima lezione
Ma 28-Nov-2005

Esame e programmazione professionale

- Le soluzione proposte sono le soluzioni che siete richiesti di programmare in sede di prova di programmazione per l'esame di Fondamenti di Informatica
- Dovete dimostrare di conoscere le strutture di base e gli algoritmi fondamentali
  - Array riempiti parzialmente
  - Liste Concatenate
  - Ordinamento
  - Ricerca
- Per questo motivo non si permette di usare le classi della libreria standard
- Il pacchetto java.util contiene, infatti, molte classi utili

Esame e programmazione professionale

- ad esempio:

Package: java.util	Struttura dati
`LinkedList`	Lista concatenata
`Stack`	Realizzazione di Pila
`Vector (jsdk 1.0)` `ArrayList (jsdk 1.1)`	Realizzazione di Lista con rango
`HashTable`	Realizzazione di Tabella Hash
`HashMap`	Realizzazione di Dizionario

Uso della libreria standard

- Realizzazione che usa la classe java.util.Vector che realizza una lista con rango

```
import java.util.Vector;
import java.util.NoSuchElementException;

public class VectorD implements Dizionario
{ private Vector<Coppia> cv;

  public VectorD()
  {
    cv = new Vector<Coppia>();
  }

  public void inserisci(String chiave, int attributo)
  { Coppia tmpCoppia = new Coppia(chiave, attributo);
    if (!presente(chiave))  cv.add(tmpCoppia);
    else
    { cv.remove(tmpCoppia);
      cv.add(tmpCoppia);
    }
  }
```

Uso della libreria standard

```
public boolean presente(String x)
  {
    return cv.contains(new Coppia(x, 1));
  }

  public int cerca(String x) throws NoSuchElementException
  {
    int indx = cv.indexOf(new Coppia(x, 1));
    if (indx < 0) throw new NoSuchElementException();
    return cv.get(indx).att;
  }

  public Coppia[] toArray()
  { return cv.toArray(new Coppia[cv.size()]); }
}
```

- La classe Coppia deve sovrascrivere il metodo equals(), nel quale si considerano uguali due istanze che abbiano la stessa chiave
- Il metodo equals() è infatti usato dai metodi della classe Vector quali, ad esempio, indexOf()
- In generale, è poi sempre opportuno sovrascrivere il metodo toString()

Classe Coppia

```
class Coppia implements Comparable
{
   final public String ch;
   final public int att;

   public Coppia (String s, int i) {ch = s; att = i; }

   public int compareTo(Object obj)
   {
      Coppia coppia = (Coppia) obj;
      return ch.compareTo(coppia.ch);
   }

   public String toString()
   {
     return ch + " " + att;
   }

   public boolean equals(Object obj)
   {
     return compareTo(obj) == 0;
   }

}
```

Uso della libreria standard

- Realizzazione che usa la classe java.util.HashMap che realizza un dizionario generico
- In questo caso non serve definire la classe Coppia perché la classe HashMap gestisce internamente le associazioni chiave, attributo
- Definiamo la classe Coppia solo perché il metodo toArray() richiede di restituire un array di oggetti di classe Copia
- Nella classe HashMap la chiave e l'attributo sono entrambi oggetti, quindi trasformiamo il nostro attributo di tipo fondamentale **`int`** in un oggetto di classe wrapper **`Integer`**
- I metodi della classe D risultano tutti semplicissimi, ad eccezione del metodo toArray() nel quale dobbiamo acquisire la descrizione interna dell'associazione chiave, attributo e memorizzarla in un array di oggetti di classe Coppia

Uso della libreria standard

```
import java.util.HashMap;
import java.util.Set;        // interfaccia
import java.util.Map;        // interfaccia
import java.util.Iterator;   // interfaccia
import java.util.NoSuchElementException;

public class HashMapD implements Dizionario
{
  // variabile di esemplare
  private HashMap<String, Integer> hm;

  public HashMapD()
  { hm = new HashMap<String, Integer>(); }

  public void inserisci(String chiave, int attributo)
  { hm.put(chiave, new Integer(attributo)); }

  public boolean presente(String x)
  { return hm.containsKey(x); }

  public int cerca(String x) throws NoSuchElementException
  { if (hm.isEmpty())
      throw new NoSuchElementException();
    return hm.get(x).intValue();
  }
```

Uso della libreria standard

```
 public Coppia[] toArray()
 {
   Set<Map.Entry<String, Integer>> s = hm.entrySet();
   Coppia[] tmp = new Coppia[s.size()];

   Iterator<Map.Entry<String, Integer>> iter =
     s.iterator();

   int i = 0;
   while (iter.hasNext())
   {
     Map.Entry<String, Integer> me = iter.next();
     tmp[i++] = new Coppia(me.getKey(),
                           me.getValue().intValue());
   }
   return tmp;
  }
}
```

Cenni alla Programmazione Generica

- In java per realizzare contenitori in grado di memorizzare oggetti di qualsiasi tipo esistono **due** meccanismi **Ereditarietà** e **Programmazione Generica**
- 1/ Ereditarietà
  - **`Object[] v = new Object[...];`**
  - è, ad esempio una struttura dati in cui possiamo inserire riferimenti a oggetti qualsiasi, perchè tutti gli oggetti derivano dalla classe generica Object attraverso il meccanismo dell'ereditarietà

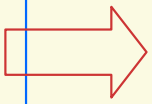
```
...
Object[] v = new Object[100];
String hello = "ciao"; // stringa
v[0] = hello;

Studente s = new Studente("rossi", "12345"); // studente
v[1] = s;
```

  - Con questa tecnica abbiamo costruito i nostri contenitori, come ad esempio, ArStack e LinkedListStack

Cenni alla Programmazione Generica

- Quando si estraggono oggetti dai contenitori, per invocare sull'oggetto i metodi specifici della classe a cui appartiene si effettua una **conversione forzata**

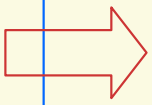
```
...
Stack s = new ArStack();
s.push("Hello, World!");
String str = (String) s.pop();
System.out.println(str.substring(7));
...
```



```
World!
```

- Se la conversione forzata è errata, in compilazione non viene segnalato alcun errore, ma in esecuzione viene generata l'eccezione **ClassCastException**

```
...
Stack s = new ArStack();
s.push(new Integer(1));
...
String str = (String) s.pop();
System.out.println(str.substring(7));
```



```
Class
Cast
Exception!
```

Cenni alla Programmazione Generica

- 2/ **Programmazione Generica java 5.0**
- Nella libreria standard sono definite delle classi generiche (Generics) che possono contenere oggetti qualsiasi ma tutti della stessa classe
- Ad esempio

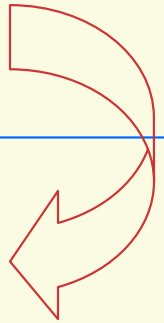
```
...
Vector<String> vStr = new Vector<String>();

Vector<Integer> vInt = new Vector<Integer>();
```

- Nella lista **`vStr`** possono essere inseriti solo riferimenti a oggetti di classe **`String`**
- Nella lista **`vInt`** possono essere inseriti solo riferimenti a oggetti di classe **`Integer`**
- La classe degli oggetti che possono essere contenuti viene scritta fra parentesi angolari, dei tipi e dei costruttori.

Cenni alla Programmazione Generica

- Se in un contenitore generalizzato particolarizzato per una certa classe si cerca di inserire un oggetto di classe diversa, l'errore viene segnalato dal compilatore

```
...
Vector<String> vStr = new Vector<String>();

Integer zero = new Integer(0);
vStr.add(zero);
...
```



```
cannot find symbol
symbol: java.util.Vector(java.lang.Integer)
location: java.util.Vector(java.lang.String)
        vStr.add(new Integer(0));
            ^
```

Sistema Operativo

Il sistema Operativo

- Fornisce al programma un ambiente uniforme, indipendentemente dalle caratteristiche Hardware del calcolatore
- E’ un codice che mette a disposizione le risorse del computer in una maniera unificata
- Realizza un’astrazione della macchina reale
- La Java Virtual Machine estende il concetto di sistema operativo fornendo una macchina ideale, indipendentemente dal computer e dal sistema operativo sottostante.

Le risorse di un computer

- Rappresentano i componenti necessari al suo funzionamento, quali
  - La CPU
  - La Memoria
  - I dispositivi di I/O
- Il sistema operativo interagisce direttamente con tali componenti:
  - Processo per l'utilizzo della CPU
  - Memoria virtuale per la gestione della memoria
  - Dispositivi (device) per la gestione delle operazioni di I/O

Gestione dei processi

- Un sistema operativo multi-processo (*multitasking)* è in grado di gestire più **processi**
- Un processo corrisponde a un programma in esecuzione
- Un programma tipicamente alterna sequenze di istruzioni CPU a operazioni di I/O
- Durante l'esecuzione di una operazione di I/O il programma non utilizza la CPU, che pertanto può essere assegnata dal sistema operativo a un alto processo

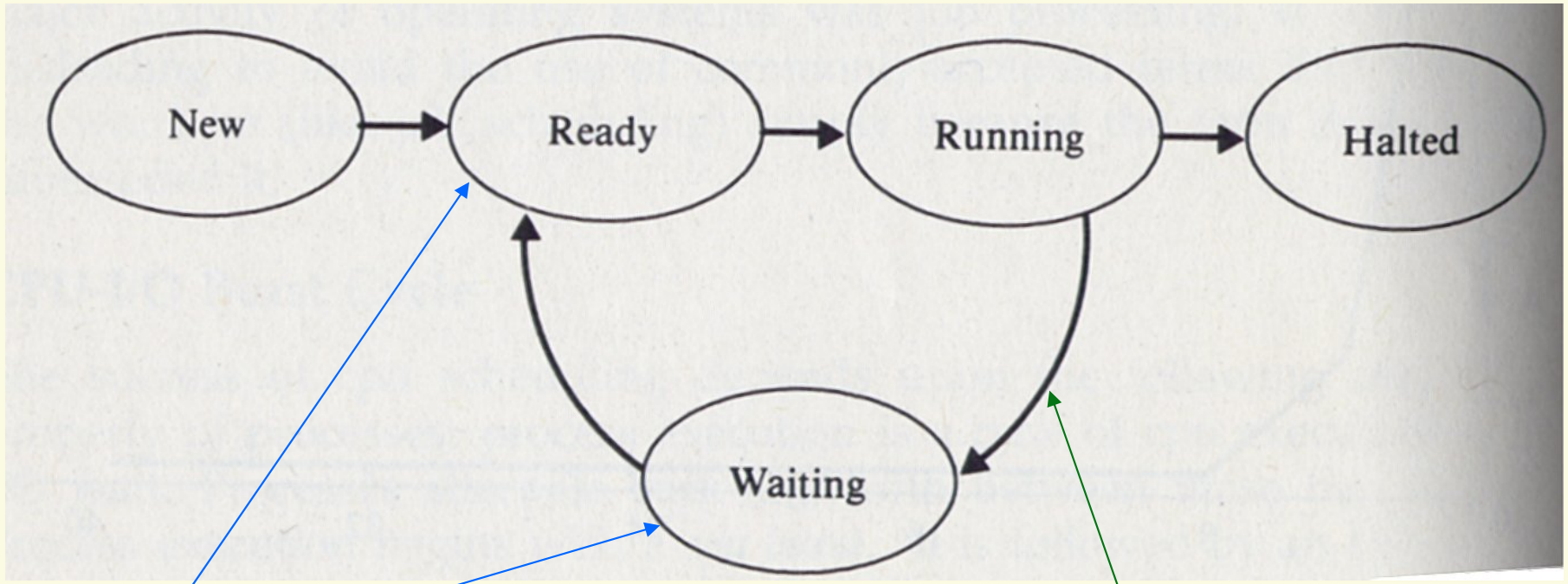
Il contesto del processo

- Rappresenta tutte le informazioni che descrivono la “fotografia” corrente del programma in esecuzione
- Alcune di queste informazioni sono:
  - Il contenuto della memoria assegnata al processo
  - Il contenuto attuale dei registri della CPU, incluso il registro Program Counter, che determina lo stato di avanzamento del programma
  - Informazioni accessorie utilizzate dal sistema operativo per la gestione del processo, quali la priorità corrente
- Tali informazioni, escluso il contenuto della memoria sono memorizzate dal sistema operativo nel ***Process Control Block*** associato .

Gli stati di un processo

- Definiscono se un processo è in esecuzione, o on attesa
- Gli stati sono:
  - **New**: processo creato, non ancora entrato in esecuzione
  - **Running**: il processo sta utilizzando la CPU
  - **Waiting**: il processo è in attesa della terminazione di un'operazione di I/O
  - **Ready**: il processo potrebbe utilizzare la CPU, ma la CPU è assegnata a un altro processo
  - **Halted**: il processo ha finito la sua esecuzione o è stato stoppato dal sistema operativo (ad esempio in conseguenza alla generazione di un'eccezione)

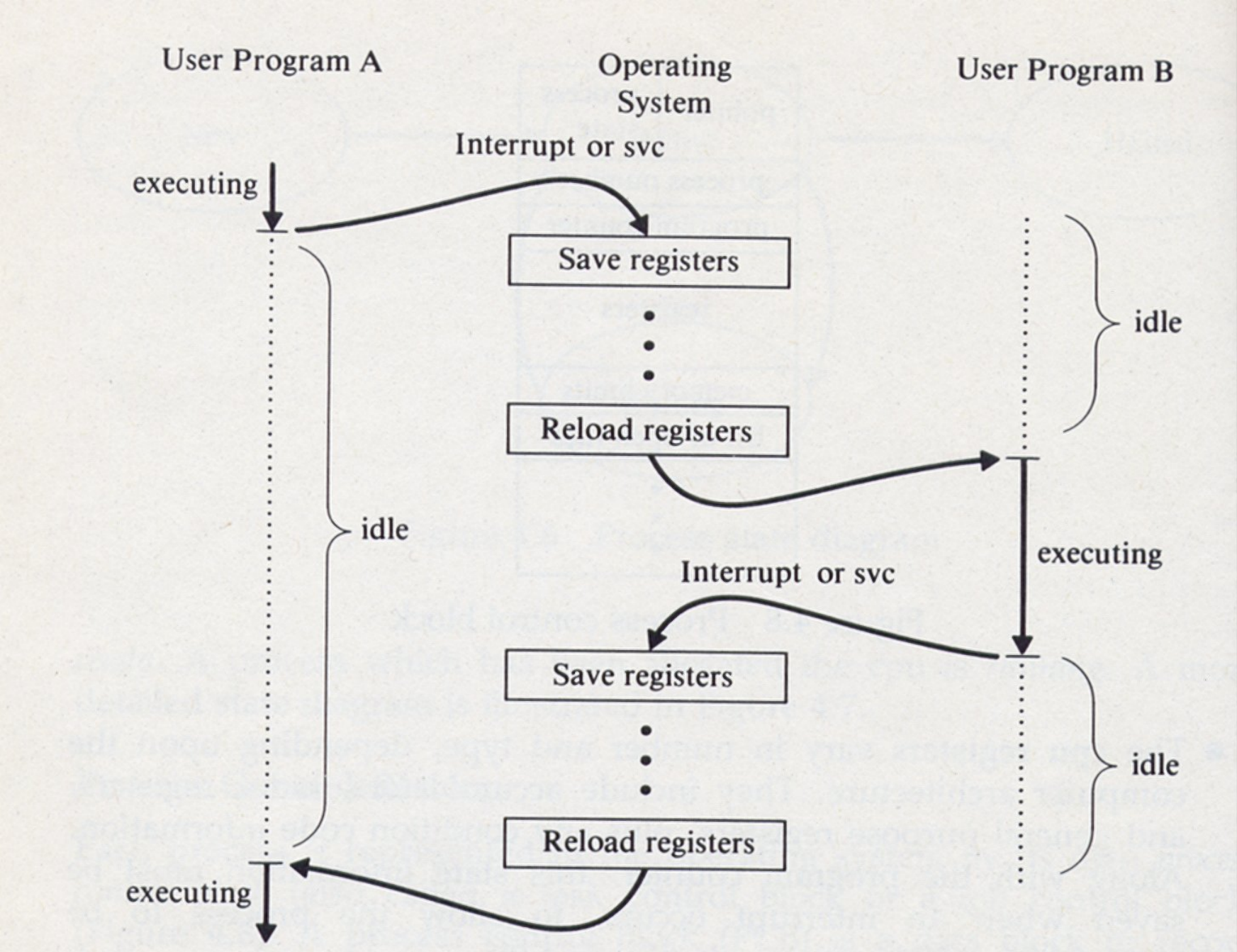
La sequenza degli stati



stato

transizione

L’operazione di Context Switch



La schedulazione (scheduling) dei processi

- Il sistema operativo ha generalmente più di un processo candidato per l'utilizzo della CPU (Ready)
- Tra le caratteristiche da tener conto nella determinazione di un efficiente algoritmo di scheduling sono le seguenti:
  - *Utilizzo della CPU*: si cerca di massimizzare l'uso della CPU
  - *Throughput*: massimizzazione dei processi eseguiti (terminati) per unità di tempo
  - *Tempo di risposta*: minimizzazione del tempo di attesa in stato Ready per ogni processo

Prima implementazione: FIFO

- L’utilizzo di una coda First-In-First out rappresenta l’implementazione più semplice, ma anche la meno efficiente.
- Si supponga, ad esempio, che nella coda siano tre processi P1, P2, P3, con il processo P1 inserito per primo nella coda. Si supponga inoltre che il processo P1 impiegherà la CPU per un tempo di 100ms, mentre gli altri due processi impiegheranno la CPU per un tempo di 1 ms.
- In questo caso il tempo medio di attesa risulterà pari a (100 + 101)/3 ms.
- Se invece venissero eseguiti prima P2 e P3, il tempo medio di attesa sarebbe, a parità di throughput: (1 + 2) /3 ms

Shortest Job First

- Con riferimento all'esempio precedente, si può intuire che l'organizzazione ideale assegnerebbe la CPU per prima al processo che la utilizzerà per il tempo minore.
- Tale approccio non è tuttavia realizzabile in pratica.
- E' possibile dare una stima τ_{n+1} del tempo di esecuzione previsto dopo n passi a partire dalla stima precedentemente fatta (τ_n) e dal tempo effettivamente impiegato al passo precedente t_n:

$$\tau_{n+1} = \alpha\, t_n + (1- \alpha)\tau_n$$

La priorità dei processi

- In molte situazioni reali, esistono processi che hanno una “importanza” maggiore, e pertanto per tali processi risulta preferibile una maggiore prontezza del sistema operativo nell’assegnazione della CPU, a scapito di altri processi meno importanti
- Ciò si può ottenere assegnando una priorità ai processi.
- Il sistema operativo assegnerà la CPU al processo Ready di priorità più alta.

Priorità fisse e variabili

- L’utilizzo di priorità fisse assegnante ai processi è adeguato per alcune classi di utilizzi, tra cui i sistemi real-time.
- In altri utilizzi, quali gestione multi-utente, tale politica non è ottimale in quanto rischia di creare lunghi tempi di attesa per alcuni processi.
- In un sistema time sharing si vuole invece che la CPU venga assegnata ai vari utenti in modo da minimizzare il tempo di risposta.
- Ciò può essere realizzato tramite l’utilizzo di priorità variabili: la priorità di processi che utilizzano pesantemente la CPU viene abbassata, mentre la priorità di processi che eseguono prevalentemente I/O viene elevata

La gestione della priorità variabile.

- Il meccanismo di cambio della priorità avviene interrompendo periodicamente il processore, e verificando il processo che sta correntemente utilizzando la CPU.
- Ad ogni interruzione (tick) la priorità del processo corrente può essere decrementata, mentre la priorità degli altri processi può essere incrementata.
- In questa maniera si “premiano” i processi che eseguono prevalentemente I/O, con l’assunzione che tali processi utilizzeranno la CPU per un tempo breve prima di tornare nello stato wait

La gestione della priorità variabile (cont.)

- Conseguentemente, lo scheduler gestirà più code FIFO di processi, una per ogni priorità.
- La CPU verrà assegnata al primo processo della coda corrispondente alla priorità massima
- Ad ogni tick, i processi possono cambiare coda, come conseguenza del loro “comportamento”
- Tale gestione pertanto fa si’ che il processo selezionato per l’utilizzo della CPU la impegnerà probabilmente per un tempo non più lungo rispetto agli altri processi che in quel momento sono ready.

La *gestione* (scheduling) dei processi

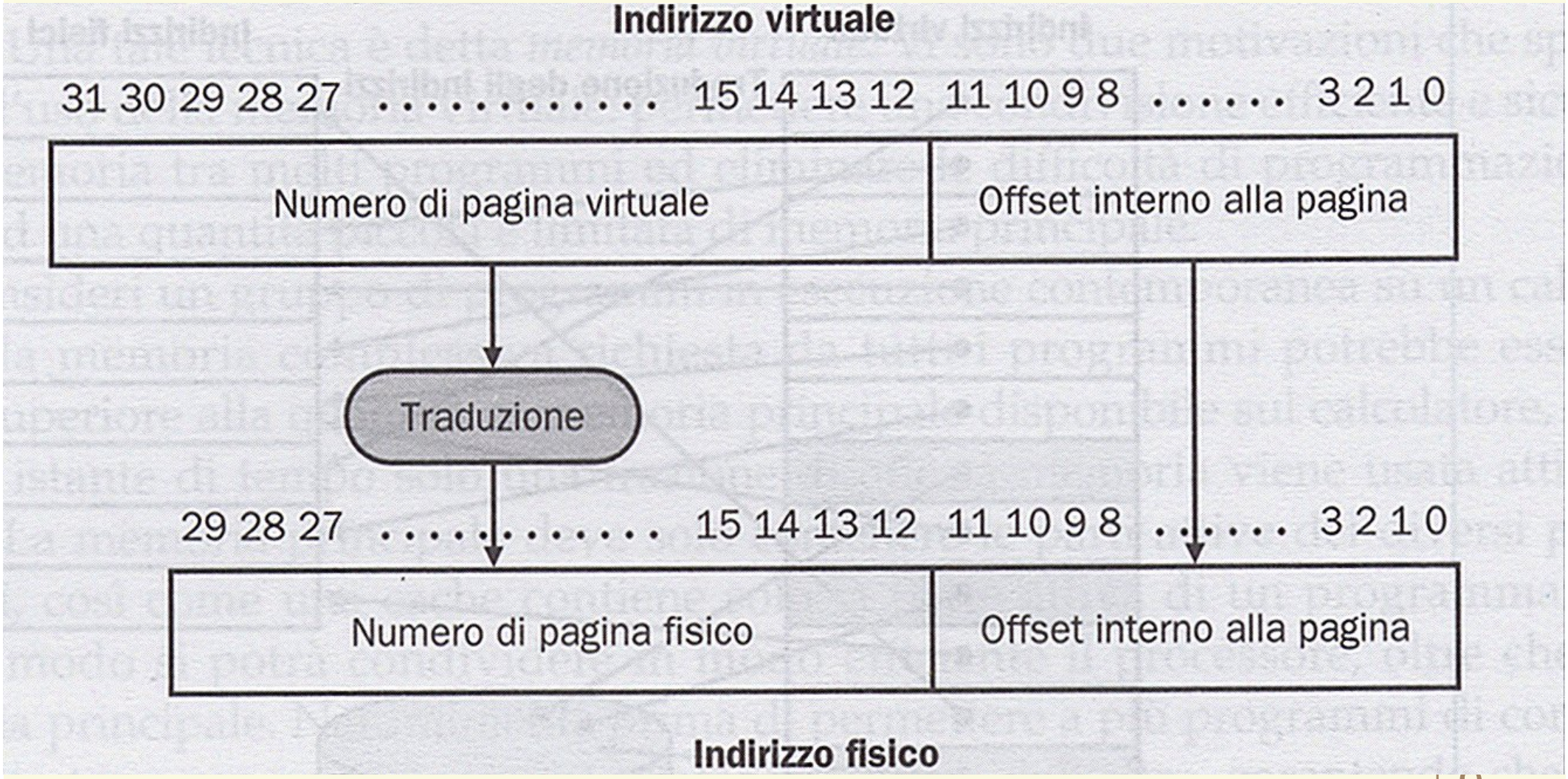
- Il sistema operativo ha generalmente più di un processo candidato per l'utilizzo della CPU (Ready)
- Tra le caratteristiche da tener conto nella determinazione di un efficiente algoritmo di scheduling sono le seguenti:
  - *Utilizzo della CPU*: si cerca di massimizzare l'uso della CPU
  - *Throughput*: massimizzazione dei processi eseguiti (terminati) per unità di tempo
  - *Tempo di risposta*: minimizzazione del tempo di attesa in stato Ready per ogni processo

La memoria virtuale

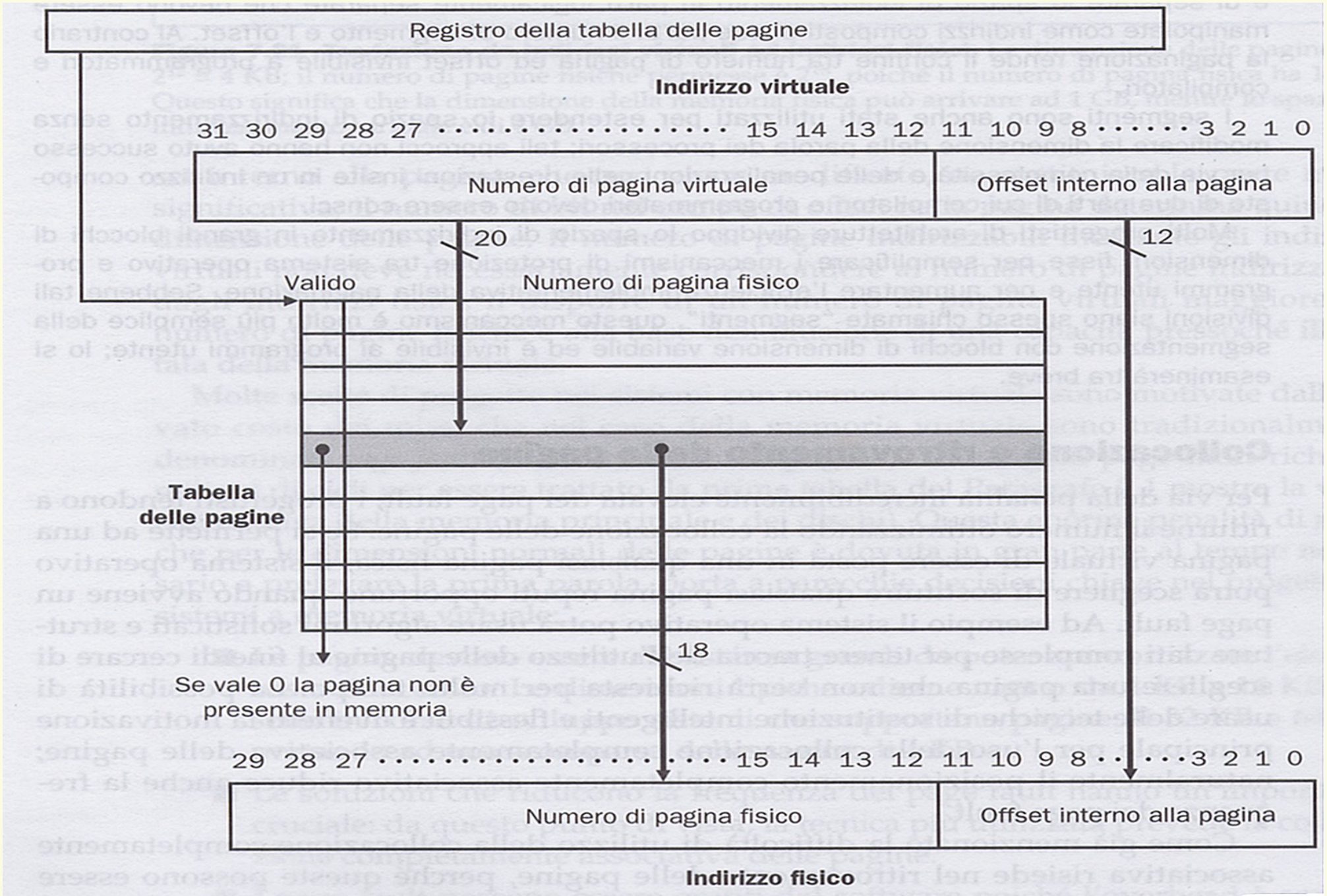
- Concetto di base: fornire una mappatura flessibile tra indirizzi **logici** o **virtuali** ed indirizzi fisici
- Ciò consente il funzionamento di più processi che utilizzano lo stesso spazio di indirizzamento
- Consente inoltre di gestire la ***paginazione***, ovvero l'utilizzo del disco per parcheggiare porzioni di memoria

La traduzione tra indirizzi virtuali e fisici

- L'unità base di memoria è la pagina.
- Dimensioni tipiche di una pagina sono 1K bytes e 4K bytes



Il meccanismo di traduzione



Il Page Fault

- Si ha un ***page fault*** quando la pagina non è attualmente residente in memoria
- Il processore segnala la condizione tramite un'eccezione
- In risposta il sistema operativo provvede a:
  - Sospendere il processo che ha causato il page fault
  - Trovare una pagina libera in memoria
  - Trasferire il blocco dal disco a tale pagina
  - Porre i bits più significativi dell'indirizzo della pagine nella Page Table Entry corrispondente
  - Svegliare il processo che ripeterà l'istruzione che ha causato il page fault

I dispositivi di I/O

- Possono essere di diversi tipi, quali:
  - Dischi
  - Rete
  - Tastera, Video
  - Interfacce seriali RS232 o USB)
  - Interfacce parallele
- In generale un dispositivo viene presentato dal sistema operativo tramite un'interfaccia semplificata in cui siano disponibili le operazioni di :
  - Open
  - Read
  - Write
  - Control

Lezione XXXIII
Me 23-Nov-2005

Esercitazione

Esercitazione (compito 2-12-2002)

- Una coda ordinata (CO) è definita dalla seguente classe:

```
// CO.java --  coda ordinata
public class CO
{ /* parte privata della classe */
  public CO() {...}
  public void accoda(Comparable x) {...}
  public Object togli() {...}
  public boolean vuota() {...}
}
```

- nella quale il metodo accoda(x) inserisce l'oggetto x (che realizza l'interfaccia Comparable) nella coda in modo tale che il metodo togli() restituisca sempre l'oggetto più piccolo presente nella coda
- Il metodo vuota() restituisce un booleano vero se la coda è vuota, falso se la coda contiene oggetti. Il costruttore CO() crea una coda ordinata vuota.

Esercitazione

- Completare la classe CO con la parte privata e con il codice che realizza i metodi e il costruttore
- La classe CO non deve avere altri metodi pubblici oltre a quelli definiti sopra

- Scrivere una classe Main di prova che:
  - legga un file di testo composto da righe formate di parole (le parole sono sequenze di caratteri alfabetici separate da uno piu' spazi) inserendo le singole parole in una coda ordinata
  - trasferisca il contenuto della coda in una seconda coda ordinata eliminando le parole ripetute e vuoti la seconda coda trasferendo le parole sull'uscita standard una parola per riga
  - Il nome del file di ingresso deve essere passato al metodo main( ) come parametro della riga di comando.

Analisi: la classe CO

- Il metodo togli() deve restituire sempre l'oggetto piu' piccolo presente nella coda

- Per questo ci sono due soluzioni possibili:

A. Il metodo accoda() non si cura dell'ordinamento degli oggetti, mentre il metodo togli() esegue la ricerca del minimo scandendo tutta la coda.

   – In questo caso possiamo realizzare, al meglio, i metodi accoda() e togli() con andamenti asintotici O(1) e O(n), rispettivamente.

Analisi: la classe CO

B. Il metodo accoda() inserisce l'oggetto in modo da mantenere ordinato l'insieme dei dati, mentre il metodo togli() restituisce l'oggetto piu' piccolo

- In questo caso possiamo realizzare, al meglio, i metodi accoda() e togli() con andamenti asintotici O(n) e O(1), rispettivamente, usando nel metodo accoda l'algoritmo di ***ordinamento per inserimento*** (insertionSort)

- l'ordinamento effettuato dal metodo accoda() avviene in un insieme gia' ordinato

- Il metodo togli() non modifica l'ordinamento nell'insieme

Soluzione con catena

- Scegliamo come contenitore la struttura dati ***catena***
- Il metodo accoda() mantenga i dati ordinati in senso crescente in modo che il dato piu' piccolo si trovi nel primo nodo
- Il metodo togli() restituisce il dato contenuto nel primo nodo
- Realizziamo la classe Nodo come ***classe interna*** della classe CO

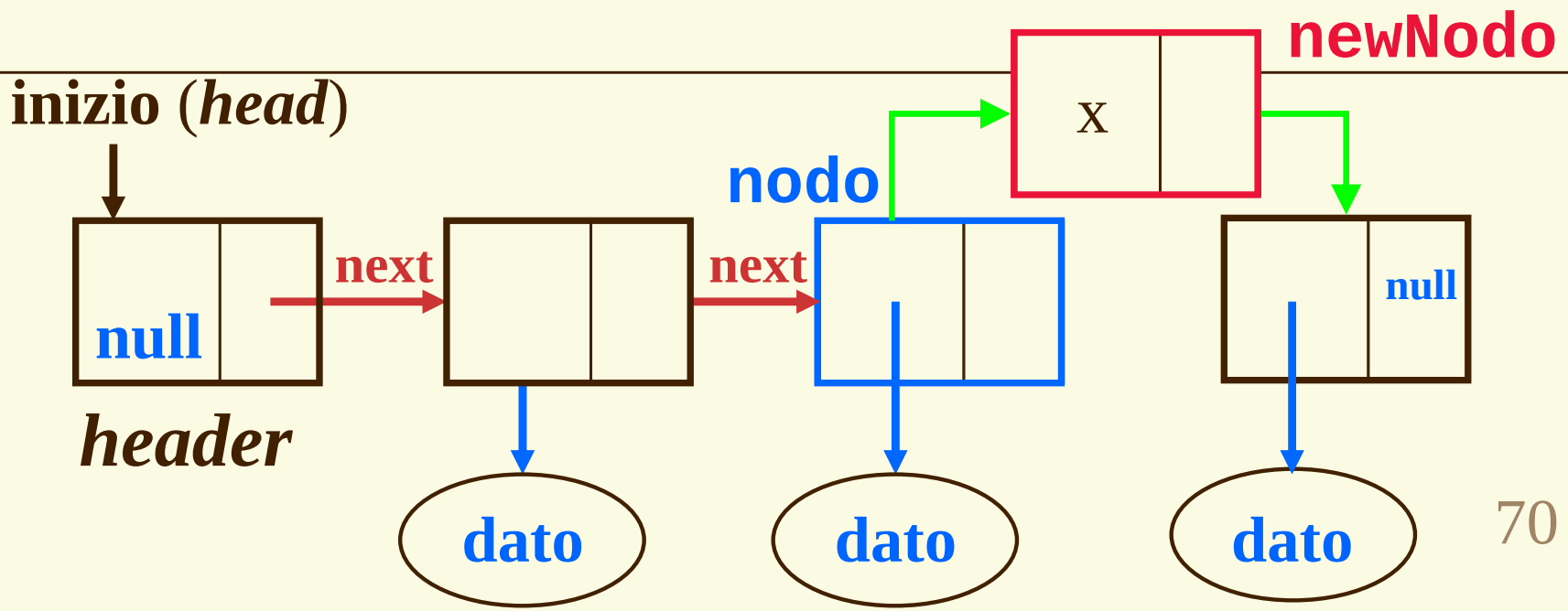
```
public class CO
{
  // parte privata
  private Nodo head;

  private class Nodo //Classe interna
  {  Object element;
     Nodo next;

     Nodo(Object obj, Nodo nextNode)//costruttore
     {
        element = obj;
        next = nextNode;
     }
  }


  public CO() //Costruttore
  { head = new Nodo (null,null);//catena vuota
  }
```

Non ci serve il riferimento *tail* all'ultimo nodo

Nella classe interna le variabili d'istanza non sono definite private
Non viola l'incapsulamento!

Catena con nodo header

```
. . .
  public void accoda(Comparable x)
  { Nodo nodo = head;
    //ricerca di posizione di inserimento
    while(nodo.next!= null
      &&  x.compareTo(nodo.next.element) > 0)
         nodo = nodo.next;

    //inserimento ordinato di nuovo nodo
    Nodo newNodo = new Nodo(x, nodo.next);
    nodo.next = newNodo;
   }
```



```
. . .
  public Object togli()
  {  if(vuota())
        return null;

     // primo dato della catena
     Object minObj = head.next.element;

     // il primo nodo diventa head
     head.next.element = null;
     head = head.next;

     return minObj;
  }

  public boolean vuota()
  {
     return head.next == null;
  }
} // fine della classe
```

Soluzione con array

- Scegliamo come contenitore la struttura dati ***array riempito parzialmente***
- Il metodo accoda() mantenga i dati ordinati in senso decrescente in modo che il dato piu' piccolo si trovi nella locazione di indice arraySize -1
- Il metodo togli() restituisce il dato contenuto nella locazione di indice arraySize -1

```
public class CO
{
  private Object[] obj;
  private int objSize;

  public CO() // Costruttore
  {  obj = new Object[1];
     objSize = 0;
  }

  private void resize() //Ridimensionamento
  {
    Object[] newObj = new Object[obj.length*2];

    for(int i = 0; i < obj.length; i++)
      newObj[i] = obj[i];

    obj = newObj;
  }
```

```
…

    public void accoda(Comparable x)
    {
        //Ridimensionamento array
        if(objSize >= obj.length)
            resize();

        //Ricerca posizione di inserimento
        int i;
        for (i = objSize; i > 0
            && x.compareTo (obj[i - 1]) > 0; i--)
                obj [i] = obj [i - 1];

        //Inserimento nell'array
        obj[i] = x;
        objSize++;
    }
```

```
…
  public Object togli()
  {
    if (vuota ())
      return null;

    Object minObj = obj[objSize - 1];
    obj[objSize-1] = null; //garbage collector
    objSize--;
    return minObj;
  }

  public boolean vuota()
  {
    return objSize == 0;
  }
 } // fine della classe
```

Classe di prova

```
import java.io.IOException;
import java.io.FileReader;
import java.util.Scanner;

public class MainCO
{ public static void main(String[] args)
    throws IOException
  {
    // Verifica argomenti su riga di comando
    if(args.length < 1)
    {
      System.out.println(
        "uso: $java MainCO nomeFile");
      return;
    }
```

Classe di prova

```
// lettura file
Scanner in = new Scanner(
   new FileReader(args[0]));
CO storeCO = new CO();

while (in.hasNextLine())
{
  Scanner tok = new Scanner(in.nextLine());

  while (tok.hasNext())
    storeCO.accoda(tok.next());
  tok.close();
}

in.close();
```

```
    CO uniqueCO = new CO();

    //eliminazione degli oggetti non unici
    Object lastObj = null;
    if (!storeCO.vuota())
    {  lastObj = storeCO.togli();
       uniqueCO.accoda((Comparable)lastObj);
    }
    while(!storeCO.vuota()) {
      Object current = storeCO.togli();
      if(((Comparable)current).compareTo(lastObj)
        != 0)
      {  uniqueCO.accoda((Comparable)current);
         lastObj = current;
      }
    }
    while(!uniqueCO.vuota()) //stampa
      System.out.println(uniqueCO.togli());
  }
}
```

```
    . . .
   CO uniqueCO = new CO();

   //eliminazione degli oggetti non unici
   Comparable lastObj = null;
   if (!storeCO.vuota())
   {
      lastObj = (Comparable)storeCO.togli();
      uniqueCO.accoda(lastObj);
   }
   while(!storeCO.vuota()) {
     Comparable current = (Comparable)storeCO.togli();
     if(current.compareTo(lastObj) != 0)
     {  uniqueCO.accoda (current);
         lastObj = current;
     }
   }
   while(!uniqueCO.vuota()) //stampa
     System.out.println(uniqueCO.togli());
  }
}
```

alternativa

Prova del programma

- Usare un file di prova come file di ingresso, ad esempio un file come i seguenti

prova1.txt	*prova2.txt*
uno due tre	3 4 5
tre quattro cinque	1 2 3
cinque sei sette	5 6 7

- Inserire in punti chiave del codice istruzioni di stampa a standard output per il debugging
  - ad esempio nel motodo accoda(), dopo l'inserimento del dato, si stampino il valore della variabile objSize e i dati ordinati

```
//Inserimento nell'array
obj[i] = x;
objSize++;
//System.out.println("objSize = " + objSize);
//for (int k =0, k < objSize; k++)
//  System.out.println(obj[i]);
```

Esercitazione

- **Traduttore**
- **Per tradurre in modo elementare un testo dall'italiano in inglese e viceversa si vuole realizzare la seguente classe:**

```
// TR.java --  traduttore
public class TR
{
  /* parte privata della classe */
  …
  public TR(String nomeFile) {...}
  public String inInglese(String parolaItaliana) {...}
  public String inItaliano(String parolaInglese) {...}
}
```

Esercitazione

- nella quale il metodo "inInglese(x)" restituisce la parola inglese corrispondente alla parola italiana x
- analogamente il metodo "inItaliano(x)" restituisce la parola italiana corrispondente alla parola x inglese.
- Nel caso non sia disponibile la traduzione della parola cercata i due metodi restituiscono un puntatore "null".
- Il costruttore "TR(nomeFile)“ preleva dal file di testo "nomeFile" le coppie <parolaItaliana,parolaInglese> che stabiliscono per ogni parola la sua traduzione.
- Le coppie sono riportate sul file una per riga e le parole sono separate da uno spazio bianco, sappiamo che ogni parola compare una volta sola nel file e che non ci sono parole italiane uguali a parole inglesi e viceversa.

Esercitazione

- Completare la classe TR con la parte privata e con il codice che realizza i due metodi e il costruttore.

- Nella realizzazione della classa TR non è consentito utilizzare classi "contenitore" quali Collections, HashSet, Hashtable, Vector, LinkedList, ecc. delle librerie di Java.

- Scrivere una classe Main di prova che, creato un esemplare della classe TR, legga dall'ingresso standard un testo in italiano e lo traduca in inglese scrivendo il risultato sull'uscita standard
- Nel caso una parola italiana non possa essere tradotta il programma di prova dovrà riportare nella traduzione la parola italiana originaria preceduta e seguita da un asterisco.

Soluzione minima

```
// TR.java -- traduttore
// soluzione elementare: ricerca per
// scansione di un vettore;
import java.io.*;
import java.util.*;
public class TR
{
  class Coppia
  {
    String italiano;
    String inglese;
    Coppia (String it, String en)
    { italiano = it; inglese = en;) }
  }
  // parte privata
  private Coppia[] v;
  private int n;
```

Soluzione minima

```
public TR(String nome) throws IOException
{
  v = new Coppia[1000]; // array dim. fissa

  Scanner in = new Scanner(new FileReader(nome));
  while (in.hasNextLine())
  {
    Scanner tok = new Scanner(in.nextLine());
    v[n] = new Coppia(tok.next(), tok.next());
    n++;
    tok.close();
  }
  in.close();
}
```

FileNotFoundException
sottoclasse di IOException

Soluzione minima

```
  public String inItaliano(String x)
  {
    for (int i = 0; i < n; i++)
      if (x.compareTo(v[i].inglese) == 0)
         return v[i].italiano;
    return null;
  }

  public String inInglese(String x)
  {
    for (int i = 0; i < n; i++)
      if (x.compareTo(v[i].italiano) == 0)
         return v[i].inglese;

    return null;
  }
} // Fine della classe
```

Soluzione con tabella hash

```
// TR1.java -- traduttore
// soluzione efficiente realizzata con una
// tabella hash
import java.io.*;
import java.util.*;
public class TR1
{
   class Nodo
   {
      String chiave;
      String attributo;
      Nodo prossimo;

      Nodo (String c, String a, Nodo p)
      {  chiave = c;
         attributo = a;
         prossimo = p;
      }
   }
```

Nodo con due dati

Soluzione con tabella hash

```
// parte privata
private Nodo[] v;
private static final int MAX = 97;

private int h(String ch)
{
   int n = 0;
   for (int i = 0; i < ch.length(); i++)
      n = ((31 * n) + ch.charAt(i)) % MAX;

   return n;
}
```

Soluzione con tabella hash

```
public TR1(String nome) throws IOException
{ v = new Nodo[MAX];
  Scanner in = new Scanner(new FileReader(nome));
  while (in.hasNextLine())
  { Scanner tok = new Scanner(in.nextLine());
    String italiano = tok.next();
    String inglese = tok.next();
    int chr = h(italiano);
    v[chr] = new Nodo(italiano, inglese, v[chr]);
    chr = h(inglese);
    v[chr] = new Nodo(inglese, italiano, v[chr]);
    tok.close();
  }
  in.close();
}
```

FileNotFoundException
sottoclasse di IOException

Soluzione con tabella hash

```
public String inItaliano(String x)
{
   Nodo p = v[h(x)];
   while (p != null)
     if (x.compareTo(p.chiave) == 0)
       return p.attributo;
     else
       p = p.prossimo;

     return null;
 }

 public String inInglese(String x)
 {
    return inItaliano(x);
 }
} // fine della classe
```

Soluzione con doppia tabella hash

```
// TR2.java -- traduttore
// soluzione efficiente realizzata con due
// tabelle hash
import java.io.*;
import java.util.*;
public class TR2
{ class Nodo
  { … }

  private Nodo[] inInglese;
  private Nodo[] inItaliano;
  private static final int MAX = 11;

  private int h (String ch)
  { … }
```

Soluzione con doppia tabella hash

```
public TR2(String nome) throws IOException
{ inItaliano = new Nodo[MAX];
  inItaliano = new Nodo[MAX];

  Scanner in = new Scanner(new FileReader(nome));
  while (in.hasNextLine())
  { Scanner tok = new Scanner(in.nextLine());
    String italiano = tok.next();
    String inglese = tok.next();
    int chr = h(italiano);
    inInglese[chr] = new Nodo(italiano,
      inglese, inInglese[chr]);
    chr = h(inglese);
    inItaliano[chr] = new Nodo(inglese,
      italiano, inItaliano[chr]);
    tok.close();
  }
  in.close();
}
```

FileNotFoundException
sottoclasse di IOException

Soluzione con doppia tabella hash

```
  public String inItaliano(String x)
  {
    Nodo p = inItaliano[h(x)];
    while (p != null)
      if (x.compareTo(p.chiave) == 0)
        return p.attributo;
      else p = p.prossimo;
    return null;
   }
   public String inInglese(String x)
   {
     Nodo p = inInglese[h(x)];
     while (p != null)
       if (x.compareTo(p.chiave) == 0)
         return p.attributo;
       else p = p.prossimo;
     return null;
   }
}
```

Classe di prova

```
import java.io.IOException;
import java.util.Scanner;

public class Main
{ public static void main(String[] arg) throws IOException
  { Scanner in = new Scanner(System.in);
    TR2 diz = new TR2(arg[0]);

    while (in.hasNext())
    { String parola = in.next();
      String traduzione = diz.inInglese(parola);
      if (traduzione == null)
        System.out.print("*" + parola + "* ");
      else
        System.out.print(traduzione + " ");
      System.out.println();
    }
    in.close();
  }
}
```

File di ingreso di prova

prova.txt
uno one due two tre three quattro four

Lezione XXXIV
Gi 24-Nov-2005

ADT Set
Standard Error

Il flusso di errore standard

- Abbiamo visto che un programma Java ha sempre due flussi ad esso collegati
  - il flusso di ingresso standard, **System.in**
  - il flusso di uscita standard, **System.out**

  che vengono forniti dal sistema operativo
- In realtà esiste un altro flusso, chiamato ***flusso di errore standard*** o ***standard error***, rappresentato dall'oggetto **System.err**
  - **System.err** è di tipo **PrintStream** come **System.out**

Il flusso di errore standard

- La differenza tra **System.out** e **System.err** è solo convenzionale
  - si usa **System.out** per comunicare all'utente i risultati dell'elaborazione o qualunque altro messaggio che sia previsto dal corretto e normale funzionamento del programma
  - si usa **System.err** per comunicare all'utente eventuali condizioni di errore (fatali o non fatali) che si siano verificate durante il funzionamento del programma

Il flusso di errore standard

- In condizioni normali (cioè senza redirezione) lo standard error finisce sullo schermo insieme allo standard output
- In genere il sistema operativo consente di effettuare la redirezione dello standard error in modo indipendente dallo standard output
  - in Windows è possibile redirigere soltanto lo standard output, mentre lo standard error rimane verso lo schermo
  - in Unix è possibile redirigere i due flussi verso due file distinti

ADT Set
Realizzazione di un insieme

Insieme

- Il tipo di dati astratto “**insieme**” (***set***) è un contenitore (eventualmente vuoto) di oggetti **distinti** (cioè non contiene duplicati)
  - senza alcun particolare ordinamento
  - senza memoria dell’ordine temporale in cui gli oggetti vengono inseriti o estratti
  - si comporta come un insieme matematico

Insieme

```
public interface Set
    extends Container
{   void add(Object obj);
    boolean contains(Object obj);
    Object[] toArray();
}
```

- Le operazioni consentite sull'insieme sono
  - **inserimento** di un oggetto
    - fallisce silenziosamente se l'oggetto è già presente
  - **verifica della presenza** di un oggetto
  - **ispezione di tutti gli oggetti**, mediante un metodo che restituisce un array di riferimenti agli oggetti contenuti nell'insieme, senza alcun requisito di ordinamento di tale array (anche perché i dati non sono ordinabili...)
- Non esiste un'operazione di **rimozione**
  - si usa la sottrazione tra insiemi (vedi in seguito)

Operazioni sugli insiemi

- Per due insiemi A e B, si definiscono le operazioni
  - **unione** $A \cup B$
    - appartengono all'unione di due insiemi tutti e soli gli elementi che appartengono ad almeno uno dei due insiemi
  - **intersezione** $A \cap B$
    - appartengono all'intersezione di due insiemi tutti e soli gli elementi che appartengono ad entrambi gli insiemi
  - **sottrazione** A - B (oppure anche $A \setminus B$)
    - appartengono all'insieme sottrazione A-B tutti e soli gli elementi che appartengono all'insieme A e non appartengono all'insieme B
    - non è necessario che B sia un sottoinsieme di A

Insieme con array non ordinato

- Quando si scrive una classe, è comodo scrivere prima le firme di tutti i metodi, con un corpo "*vuoto*"

```
public class ArraySet implements Set
{  public void makeEmpty() { }
   public boolean isEmpty() { return true; }
   public Object[] toArray() { return null; }
   public boolean contains(Object x) {
      return true; }
   public void add(Object x) { }
}
```

- Invece di compilare tutto alla fine, si compila ogni volta che si scrive il corpo di un metodo
  - in questo modo, si evita di trovarsi nella situazione in cui il compilatore segnala molti errori

```
public class ArraySet implements Set
{  private final int SET_DIM = 1;
   private Object[] v = new Object[SET_DIM];
   private int vSize = 0;

   public void makeEmpty() { vSize = 0; }
   public boolean isEmpty(){ return vSize == 0;}

   public Object[] toArray()     // O(n)
   {  Object[] x = new Object[vSize];
      for (int i = 0; i < vSize; i++)
        x[i] = v[i];
      return x;
   }

   public boolean contains(Object x)  // O(n)
   {  for (int i = 0; i < vSize; i++)
         if (v[i].equals(x)) return true;
      return false;
   }
…//continua
```

Insieme con array non ordinato

```
public class ArraySet implements Set
{
. . .
   // O(n) (usa contains)
   public void add(Object x)
   {
      if (contains(x)) return;
      if (vSize >= v.length)
         v = resize(v, 2 * v.length);
      v[vSize++] = x;
   }

   private Object[] resize(Object[] v, int n)
   { ... }
}
```

Operazioni su insiemi: unione

```
public static Set union(Set s1, Set s2)
{  Set x = new ArraySet();
   // inseriamo gli elementi del primo insieme
   Object[] v = s1.toArray();
   for (int i = 0; i < v.length; i++)
      x.add(v[i]);
   // inseriamo tutti gli elementi del
   // secondo insieme, sfruttando le
   // proprietà di add (niente duplicati...)
   v = s2.toArray();
   for (int i = 0; i < v.length; i++)
      x.add(v[i]);
   return x;
}
```

- Se **contains** è **O(n)** (e, quindi, lo è anche **add()**), questa operazione è **$O(n^2)$**

Insiemi: intersezione e sottrazione

```
public static Set intersection(Set s1, Set s2)
{  Set x = new ArraySet();
   Object[] v = s1.toArray();
   for (int i = 0; i < v.length; i++)
      if (s2.contains(v[i]))
         x.add(v[i]);
   return x;
}                                          // O(n²)
```

```
public static Set subtract(Set s1, Set s2)
{  Set x = new ArraySet();
   Object[] v = s1.toArray();
   for (int i = 0; i < v.length; i++)
      if (!s2.contains(v[i]))
         x.add(v[i]);
   return x;
}                                          // O(n²)
```

Insieme con array non ordinato

- Riassumendo, realizzando un insieme con un array non ordinato
  - le prestazioni di tutte le primitive dell'insieme sono **O(n)**
  - le prestazioni di tutte le operazioni che agiscono su due insiemi sono **$O(n^2)$**
- Si può facilmente verificare che si ottengono le stesse prestazioni realizzando l'insieme con una catena (prestazioni determinate dalla ricerca)

Insieme di dati ordinabili

- Cerchiamo di capire se si possono ottenere prestazioni migliori quando l'insieme contiene dati ordinabili
- Definiamo l'interfaccia "insieme ordinato"

```
public interface SortedSet extends Set
{  void add(Comparable obj);
   Comparable[] toSortedArray();
}
```

```
public interface Set extends Container
{  void add(Object obj);
   boolean contains(Object obj);
   Object[] toArray();
}
```

Insieme di dati ordinabili

- Realizziamo l'interfaccia **SortedSet** usando un array ordinato
  - dovremo definire due metodi **add()**, uno dei quali ***impedisce l'inserimento di dati non ordinabili***

```
public interface SortedSet extends Set
{  void add(Comparable obj);
   Comparable[] toSortedArray();
}
```

```
public interface Set extends Container
{  void add(Object obj);
   boolean contains(Object obj);
   Object[] toArray();
}
```

Insieme con array ordinato

```
public class ArraySortedSet implements SortedSet
{
   private Comparable[] v = new Comparable[100];
   private int vSize = 0;

   public void makeEmpty()
   {  vSize = 0; }
   public boolean isEmpty()
   {  return vSize == 0; }

   public Comparable[] toSortedArray() // O(n)
   {  Comparable[] x = new Comparable[vSize];
      for (int i = 0; i < vSize; i++)
        x[i] = v[i];
      return x; }

   public Object[] toArray()
   {  return toSortedArray();
   }
   ...
}
```

Insieme con array ordinato

```
public class ArraySortedSet implements SortedSet
{  ...
  public boolean contains(Object x)  // O(log n)
  {  // si può fare una ricerca binaria
      ...
  }

  public void add(Object x)
  {  throw new IllegalArgumentException(); }

  public void add(Comparable x)      // O(n)
  {  if (contains(x)) return;
     if (vSize == v.length)
        v = resize(v, 2*vSize);
     v[vSize++] = x;
     // usiamo insertion sort che è O(n)
     // perché inseriamo in un array ordinato
     ...
  }
}
```

Operazioni su insiemi ordinati

- Gli algoritmi già visti per le operazioni sugli insiemi generici possono essere utilizzati senza alcuna modifica anche per l'insieme ordinato, realizzato con un array ordinato
  - infatti, un **SortedSet** è anche un **Set**
  - la complessità di tutti gli algoritmi (unione, intersezione e sottrazione) rimane **$O(n^2)$**, perché il metodo **add** è rimasto **O(n)**
- Senza sfruttare le informazioni sull'ordinamento dell'insieme, non è possibile ottenere prestazioni migliori...

Operazioni su insiemi ordinati

- Usare l'incapsulamento a oltranza può essere sconveniente…
- In questo caso, sapendo che
  - l'array ottenuto con il metodo **toSortedArray** è ordinato
  - l'inserimento nell'insieme avviene nel metodo **add()** con l'algoritmo di ordinamento per inserzione in un array ordinato

  è possibile scrivere versioni più efficienti dei metodi già visti

Operazioni su insiemi: unione

- Per realizzare l'***unione***, osserviamo che il problema è molto simile alla ***fusione di due array ordinati***
  - come abbiamo visto in **MergeSort**, questo algoritmo di fusione è **O(n)**
- L'unica differenza consiste nella contemporanea eliminazione (cioè nel non inserimento…) di eventuali oggetti duplicati
  - un oggetto presente in entrambi gli insiemi dovrà essere presente una sola volta nell'insieme unione

Operazioni su insiemi: unione

- Effettuando la fusione dei due array ordinati secondo l'algoritmo visto in **MergeSort**, gli oggetti vengono via via inseriti nell'insieme unione che si va costruendo
- Se gli inserimenti avvengono con oggetti ***in ordine crescente***, l'ordinamento per inserzione in un array ordinato che viene usato dal metodo **add()** ha prestazioni **O(1)** per ogni inserimento!
  - il metodo **add()** ha quindi prestazioni **O(log n)**
    - perché invoca **contains()** che è **O(log n)**
  - se gli inserimenti non avvengono in ordine crescente, il metodo **add()** ha prestazioni medie di tipo **O(n)**

Operazioni su insiemi: unione

```
public static SortedSet union(SortedSet s1, SortedSet s2)
{
   SortedSet x = new ArraySortedSet();
   Comparable[] v1 = s1.toSortedArray();
   Comparable[] v2 = s2.toSortedArray();
   int i = 0, j = 0;
   while (i < v1.length && j < v2.length)
      if (v1[i].compareTo(v2[j]) < 0)
         x.add(v1[i++]);
      else if (v1[i].compareTo(v2[j]) > 0)
         x.add(v2[j++]);
      else
      {  x.add(v1[i++]);
         j++;
      }
   while (i < v1.length) x.add(v1[i++]);
   while (j < v2.length) x.add(v2[j++]);
   return x;
} // prestazioni O(n log n) anziché quadratiche
```

Operazioni su insiemi: intersezione

```
public static SortedSet intersection(ArraySortedSet s1,
                                     ArraySortedSet s2)
{
   SortedSet x = new ArraySortedSet();
   Comparable[] v1 = s1.toSortedArray();
   Comparable[] v2 = s2.toSortedArray();
   for (int i = 0, j = 0; i < v1.length; i++)
   {
      while (j < v2.length &&
             v1[i].compareTo(v2[j]) > 0)
        j++;
      if (j == v2.length)
         break;
      if (v1[i].compareTo(v2[j]) == 0)
         x.add(v1[i]);
   }
   return x;
} // prestazioni O(n log n) anziché quadratiche
```

Operazioni su insiemi: sottrazione

```
public static SortedSet subtract(ArraySortedSet s1,
                                 ArraySortedSet s2)
{
   SortedSet x = new ArraySortedSet();
   Comparable[] v1 = s1.toSortedArray();
   Comparable[] v2 = s2.toSortedArray();
   for (int i = 0, j = 0; i < v1.length; i++)
   {
      while (j < v2.length &&
             v1[i].compareTo(v2[j]) > 0)
         j++;
      if (j == v2.length)
         break;
      if (v1[i].compareTo(v2[j]) != 0)
         x.add(v1[i]);
   }
   return x;
} // prestazioni O(n log n) anziché quadratiche
```