Gestione di Basi di Dati mediante l'approccio *File Processing*

Prof. Giuseppe Polese

Corso di *Basi di Dati*

Anno Accademico 2005/06

File e classi predefinite

- File su disco: un insieme di dati sul disco cui è associato un nome
  - Persistenza, i dati vivono oltre il tempo dell'elaborazione
  - Grande capacità, dipende solo dalla capienza del disco
  - Condivisibilità, più applicazioni/utenti possono accedere agli stessi file

3

File

- I file su disco hanno due attributi
  - Un contenuto (i dati)
  - Un nome (del file)
- Contenuto
  - Qualsiasi: compito d'esame, album fotografico, programma Java
- Operazioni
  - Creazione, cancellazione, cambiamento del nome, cambiamento del contenuto, lettura del contenuto

4

La classe File

- Modella: il nome di un file su disco
- Costruttore: File(nomeDelFile)
  - Nome del file: reference ad un oggetto String
- Metodi:
  - delete()
  - Restituisce: niente
  - Azione: cancella il file dal disco
  - renameTo(nuovoNome)
  - nuovoNome: reference ad un oggetto File
  - Restituisce: niente
  - Azione: rinomina il file

5

Esempio

```
import java.io.*;
// Program to delete the file named “junk”
class ZapJunk {
        public static void main(String a[]) throws Exception {
                File f;
                f = new File ("junk");
                f.delete();
        }
}
```

6

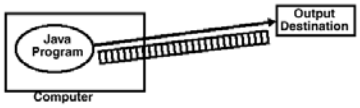
Esempio

```
import java.io.*;
// Program to rename the file named “today”
// to “yesterday”
class TodayBecomesYesterday {
        public static void main(String a[]) throws Exception {
                File fOld, fNew;
                fOld = new File ("today");
                fNew = new File ("yesterday");
                fOld.renameTo(fNew);
        }
}
```

7

Scrittura su file

- Output Stream
  - Una sequenza di dati da un programma ad un dispositivo d'uscita
    - Monitor, file, casse acustiche, connessione di rete



Varianti

Bufferizzati / non bufferizzati

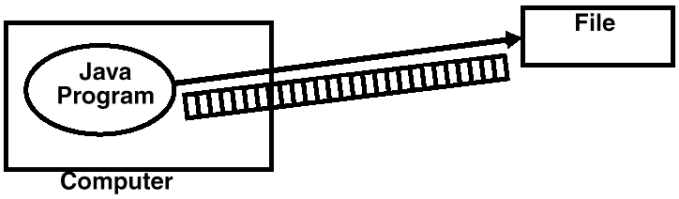
Unità dei dati: byte, string, linee …

In Java classi diverse modellano Stream diversi

8

La classe FileOutputStream

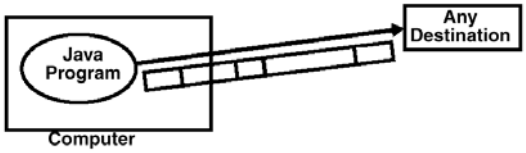
- Modella: stream di bytes inviati ad un file su disco
- Costruttore: FileOutputStream(file)
  - file – reference ad un oggetto File
- Metodi: nessun metodo per la scrittura di stringhe



9

La classe PrintStream

- Modella: stream di stringhe e linee inviati ad una destinazione qualsiasi
- Costruttore: PrintStream(fileOutputStream)
  - NON è l'unico costruttore
- Metodi (… già noti…)
  - print(String), println(String), println()



10

Mettiamo tutto insieme

- Creare un oggetto File che modella il nome di un file su disco
- Creare un oggetto FileOutputStream che modella il flusso verso il file su disco
- Creare un oggetto PrintStream che modella uno stream di stringhe e linee che fluiscono verso il file su disco
- Scrivere i dati sul file inviando messaggi print e println all'oggetto PrintStream

11

Mettiamo tutto insieme

```
import java.io.*;
class WriteAddress {
        public static void main(String a[]) throws Exception{
                File usFile;
                FileOutputStream usFS;
                PrintStream usPS;
                usFile = new File ("address");
                usFS = new FileOutputStream(usFile);
                usPS = new PrintStream(usFS);
                usPS.println("Università di Salerno");
                usPS.println(“Facoltà di Scienze");
                usPS.println(“Via S. Allende, 43");
                usPS.println("84081 Baronissi (SA), Italy");
        }
}
```

12

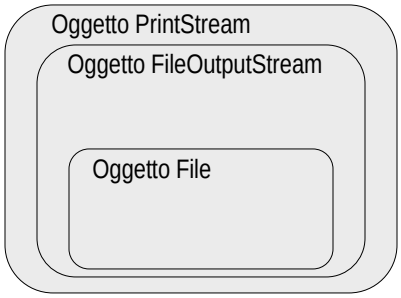
Stile composizione

```
import java.io.*;
// Program to write address in a file named "address"
class WriteAddress {
        public static void main(String a[]) throws Exception {
                PrintStream usPS;
                usPS = new PrintStream(
                                new FileOutputStream(
                                        new File ("address")));
                usPS.println("Università di Salerno");
                usPS.println(“Facoltà di Scienze");
                usPS.println(“via S. Allende, 43");
                usPS.println("84081 Baronissi (SA), Italy");
        }
}
```

13

Composizione

L'oggetto PrintStream
ha un oggetto
FileOutputStream
ha un oggetto
File



- I metodi print e println di PrintStream inviano Stringhe all'oggetto File utilizzando metodi di FileOutputStream

14

Esempio

```
import java.io.*;
class Program1Backup {
        public static void main(String arg[]) throws Exception {
                PrintStream bPS;
                FileOutputStream bFS;
                File bF;
                bF = new File("backup");
                bFS = new FileOutputStream(bF);
                bPS = new PrintStream(bFS);
                System.out.println("First program");
                bPS.println("First program");
                System.out.println(".. not last.");
                bPS.println(".. not last.");
        }
}
```

15

Commenti (I)

- Le operazioni su file interagiscono con l'ambiente
- Cosa succede se:
  - Mancanza di diritti in scrittura …
- La frase **throws Exception** è obbligatoria e segnala la possibilità del verificarsi di errori irrecuperabili
  - Più dettagli nel seguito …

16

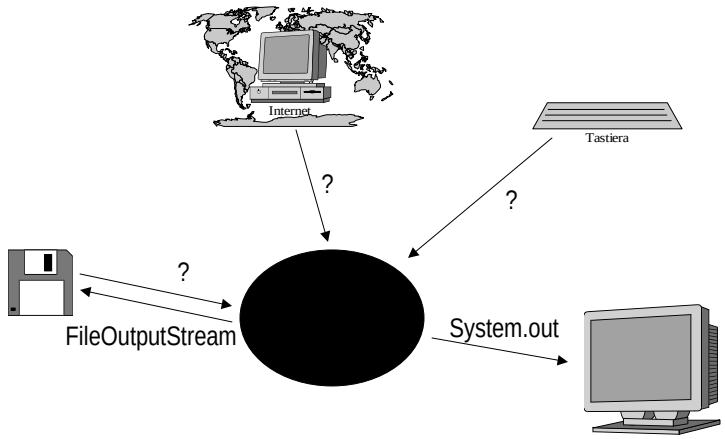
Commenti (II)

- Perché non dotare la classe FileOutputStream dei metodi print e println ?
- Principio:

  Separazione delle responsabilità
- FileOutputStream
  - Far fluire i byte …
- PrintStream
  - Determinare l'organizzazione dei dati che fluiscono
  - Può essere utilizzata per destinazioni diverse da file su disco

17

Panoramica sull'I/O



18

Panoramica sugli ingressi

- Stesso approccio dell'uscita
  - Diverse classi per modellare i diversi aspetti dei flussi di dati in ingresso
- Classi che modellano sorgenti di flussi di dati in ingresso in byte
  - FileInputStream
  - BufferedInputStream
- Classi che modellano sequenze di caratteri
  - InputStreamReader
- Classi che modellano sequenze di linee
  - BufferedReader

19

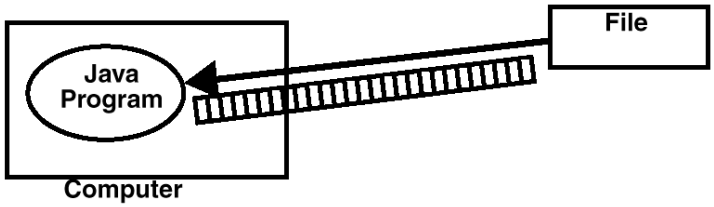
Strategia

- Trovare o creare un oggetto FileInputStream o BufferedInputStream
- Creare un oggetto InputStreamReader
- Creare un oggetto BufferedReader
- Inviare messaggi readLine all'oggetto BufferedReader

20

La classe FileInputStream

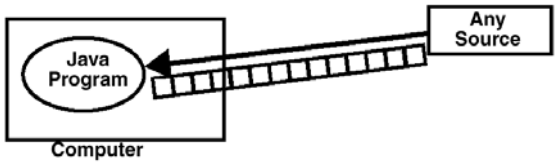
- Modella: flussi di byte che arrivano da un file su disco
- Costruttore: **FileInputStream(file)**
  - file – reference ad un oggetto File
- Metodi: nessuno per trattare stringhe



21

La classe InputStreamReader

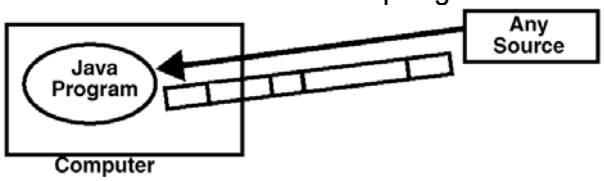
- Modella: flussi di caratteri che arrivano da un qualunque dispositivo di ingresso
- Costruttori
  - InputStreamReader(fileInputStream)
  - InputStreamReader(bufferedInputStream)
- Metodi: ancora nessuno per le stringhe



22

La classe BufferedReader

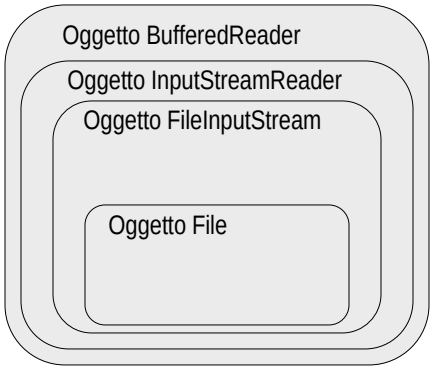
- Modella: flussi di linee che arrivano da un qualsiasi dispositivo di ingresso
- Costruttore: BufferedReader(inputStreamReader)
- Metodi: readLine()
  - Restituisce un reference ad un oggetto String che contiene i caratteri che compongono la linea



23

Composizione

L'oggetto BufferedReader
ha un oggetto
InputStreamReader
ha un oggetto
FileInputStream
ha un oggetto
File



24

Esempio

```
import java.io.*;
// Program to display first line of a file named "winners"
class Head1 {
        public static void main(String a[]) throws Exception {
                File wFile;
                FileInputStream wFS;
                InputStreamReader wIS;
                BufferedReader wBR;
                String line1;
                wFile = new File ("winners");
                wFS = new FileInputStream(wFile);
                wIS = new InputStreamReader(wFS);
                wBR = new BufferedReader(wIS);
                line1 = wBR.readLine();
                System.out.println(line1);
        }
}
```

25

Esempio: stile composizione

```
import java.io.*;
// Program to display first line of a file named "winners"
class Head1 {
        public static void main(String a[]) throws Exception {
                BufferedReader wBR;
                String line1;
                wBR = new BufferedReader(
                                new InputStreamReader(
                                        new FileInputStream(
                                        new File ("winners"))));
                line1 = wBR.readLine();
                System.out.println(line1);
        }
}
```

26

La tastiera

- System.in
  - Un reference ad oggetto predefinito di tipo BufferedInputStream associato alla tastiera
  - Può essere utilizzato per definire oggetti InputStreamReader etc. al fine di leggere dalla tastiera piuttosto che dal file

27

Esempio

```
import java.io.*;
// reads a line from the keyboard and says it doesn't
// want that!
class DoNotLike {
        public static void main(String a[]) throws Exception {
                InputStreamReader keyIS;
                BufferedReader keyBR;
                String line;
                keyIS = new InputStreamReader(System.in);
                keyBR = new BufferedReader(keyIS);
                line = keyBR.readLine();
                System.out.print("I don't want ");
                System.out.println(line);
        }
}
```

28

I/O interattivi

- Obiettivo: Introdurre messaggi di prompt che indichino all'utente cosa digitare (**interazione**)
  - Es: Please enter your name:

29

Esempio

```
import java.io.*;
class Chatty {
        public static void main(String a[]) throws Exception {
                InputStreamReader keyIS;
                BufferedReader keyBR;
                String name;
                keyIS = new InputStreamReader(System.in);
                keyBR = new BufferedReader(keyIS);
                System.out.println("What's your name?");
                name = keyBR.readLine();
                System.out.print("Well HELLO there, ");
                System.out.println(name);
        }
}
```

30

Problema

- Scrivere un programma che chieda all'utente il nome di un file su disco e ne visualizzi la prima linea
  1. Identificare gli oggetti necessari

     Poiché il trattamento degli oggetti richiede opportune variabili di riferimento …
  2. Scrivere le dichiarazioni delle variabili necessarie
  3. Scrivere il codice che istanzia gli oggetti necessari

31

Soluzione

1. Oggetti necessari
BufferedReader per la keyboard
InputStreamReader per la keyboard
PrintStream per il monitor
String per il nome di file, String per la prima linea del file
File per il file su disco
FileInputStream, InputStreamReader e BufferedReader per il file su disco
2. Dichiarazione di variabili di referenza:

```
BufferedReader keyBR, fBR;
InputStreamReader keyISR, fISR;
FileInputStream fIS;
File f;
PrintStream stdout;
String fname, line1;
```

Soluzione

Provate a riscrivere
Lo stesso programma
con lo stile composizione

```
import java.io.*;
class ShowFirstLine {
        public static void main(String a[]) throws Exception {
                BufferedReader keyBR, fBR;
                InputStreamReader keyISR, fISR;
                FileInputStream fIS;
                File f;
                PrintStream stdout;
                String fname, line1;
                stdout = System.out;
                keyISR = new InputStreamReader(System.in);
                keyBR = new BufferedReader(keyISR);
                stdout.print("File? ");
                fname = keyBR.readLine();
                f = new File (fname);
                fIS = new FileInputStream(f);
                fISR = new InputStreamReader(fIS);
                fBR = new BufferedReader(fISR);
                line1 = fBR.readLine();
                stdout.println(line1);
        }
}
```

33

Problema

- Copiare un file di due linee
  - Ovvero le prime due linee di un file

```
import java.io.*;
class CopyFile2 {
        public static void main(String arg[]) throws Exception {

                InputStreamReader keyISR;
                BufferedReader keyBR;
                PrintStream stdout;
```

34

Soluzione

```
String fName1;
File f1;
FileInputStream fis1;
InputStreamReader isr1;
BufferedReader rdr1;

String fName2;
File f2;
PrintStream ps2;
FileOutputStream fos2;

String s;
```

35

Soluzione

```
stdout = System.out;
keyISR = new InputStreamReader(System.in);
keyBR = new BufferedReader(keyISR);

stdout.print("file to copy: ");
fName1 = keyBR.readLine();
fName2 = fName1.concat(".copy");

f1 = new File(fName1);
fis1 = new FileInputStream(f1);
isr1 = new InputStreamReader(fis1);
rdr1 = new BufferedReader(isr1);
```

36

Soluzione

```
                f2 = new File(fName2);
                fos2 = new FileOutputStream(f2);
                ps2 = new PrintStream(fos2);

                s = rdr1.readLine();
                ps2.println(s);
                s = rdr1.readLine();
                ps2.println(s);
            }
        }
```

37

Il metodo flush

- Trasferire una stringa da un programma ad un dispositivo di uscita richiede del tempo
- Risulta più efficiente trasferire tutte le stringhe in una volta sola, o comunque a lotti
- PrintStream raggruppa in un'area di memoria tutte le stringhe da visualizzare (**buffer**)
- Conseguenza: il prompt potrebbe NON essere visualizzato quando desiderato !
- Il metodo flush (senza argomenti) forza l'invio delle stringhe dal buffer al dispositivo

38

Esempio

- System.out.println("Digitare il numero di matricola");
  inputLine = keybd.readLine();
- Potrebbe NON funzionare come desiderato
- System.out.println("Digitare il numero di matricola");
  System.out.flush();
  inputLine = keybd.readLine();

39

Esempio

```
import java.io.*;
class NoFlush { //potrebbe non funzionare come atteso !
        public static void main(String arg[]) throws IOException {
                InputStreamReader isr;
                BufferedReader keyboard;
                String inputLine;
                isr = new InputStreamReader(System.in);
                keyboard = new BufferedReader(isr);
                System.out.println("Type in a word to be pluralized, please ");
                inputLine = keyboard.readLine();
                System.out.print(inputLine);
                System.out.println("s");
        }
}
```

40

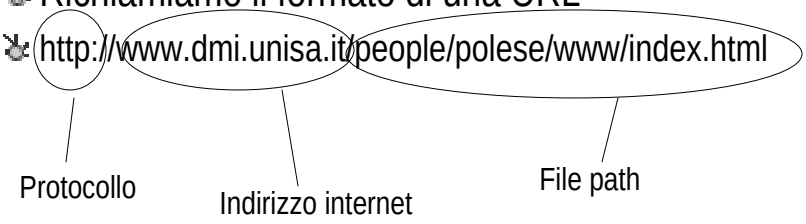
Esempio

```
import java.io.*;
class WithFlush {  //Funziona sempre
        public static void main(String arg[]) throws IOException {
                InputStreamReader isr;
                BufferedReader keyboard;
                String inputLine;
                isr = new InputStreamReader(System.in);
                keyboard = new BufferedReader(isr);
                System.out.println("Type in a word to be pluralized, please ");
                System.out.flush();
                inputLine = keyboard.readLine();
                System.out.print(inputLine);
                System.out.println("s");
        }
}
```

41

Ingresso da rete

- Sapete già cosa sia una rete, e cosa è internet
- Richiamiamo il formato di una URL
- http://www.dmi.unisa.it/people/polese/www/index.html



42

Ingresso da rete

- Concettualmente simile a quello da file
- Una classe URL che modella URL
  - È usata per creare una connessione ad URL ed ottenere un reference ad un oggetto di tipo InputStream
- Quindi …
- => InputStreamReader => BufferedReader …

43

La classe URL

- Modella: URL – Uniform Resource Locator
- Costruttore: URL(unaURL)
  - unaURL – reference ad un oggetto String
- Metodi
  - openStream()
    - Restituisce un reference ad un oggetto InputStream
- Es.
  - URL u;
    u = new URL(“http://www.yahoo.com”);
    InputStream ic = u.openStream();

44

Esempio

```
import  java.net.*;
import java.io.*;

class BnWWW {
   public static void main(String[] arg) throws Exception {
                        URL u = new URL("http://www.unisa.it”);
                        InputStream ins = u.openStream();
                        InputStreamReader isr = new InputStreamReader(ins);
                        BufferedReader BN = new BufferedReader(isr);
                        System.out.println(BN.readLine());
                        System.out.println(BN.readLine());
                        System.out.println(BN.readLine());
                        System.out.println(BN.readLine());
                        System.out.println(BN.readLine());
            }
}
```

45

Esempio: Database di canzoni (1)

- Problema
  - WOLD, una stazione radio locale, vuole costruire un database di canzoni, per automatizzare le richerche.
  - Si è creato un file in cui sono stati inseriti degli elementi composti dai titoli e dai compositori delle canzoni.
  - Si intende dare al disk-jockey la possibilità di cercare nel database tutte le canzoni di un particolare artista.

46

Database di canzoni (2)

- Scenario d'esempio

Inserisci il nome del file contenente il database di canzoni:
ClassicRock.db
File ClassicRock.db loaded.
Inserisci l'artista da cercare: Beatles
Canzoni dei Beatles trovate:
Back in the USSR
Paperback writer
She Loves You
Inserisci l'artista da cercare: Mozart
Nessuna canzone di Mozart trovata

47

Determinare gli oggetti primari

- Nomi: song DB, song, file, entry, title, artist
- Artist e title sono parti di song, che è sussidiaria di song library
- File e entry (in un file) rappresentano solo dati da leggere
- Classe primaria: SongDB

```
class SongDB {
...
}
```

48

Determinare il comportamento desiderato

- Capacità di creare un SongDB
  - Costruttore
- Necessità di cercare le canzoni di un artista
  - Un metodo lookUp

49

Definire l'interfaccia

- Tipico codice di utilizzo

```
SongDB classical = new SongDB("classical.db");
SongDB jazz = new SongDB("jazz.db");
classical.lookUp("Gould");
classical.lookUp("Marsalas");
jazz.lookUp("Corea");
jazz.lookUp("Marsalas");
```

- Abbiamo bisogno della seguente interfaccia

```
class SongDB {
    public SongDB(String songFileName) {...}
    void lookUp(String artist) throws Exception {...}
    …
}
```

50

Definire le variabili di istanza

- Ogni volta che viene invocato lookUp crea un nuovo BufferedReader associato al file su disco specificato dal nome del file di canzoni (passato al costruttore).
- Questo nome deve quindi essere mantenuto in una variabile d'istanza

```
class SongDB {
    public SongDB(String songFileName) { // costruttore
                this.songFileName = songFileName;
    }
    ...         // Metodo lookup
    String songFileName;      // variabile d'istanza
}
```

51

Implementazione del metodo lookup

```
void lookUp(String artist) throws Exception {
        BufferedReader br = new BufferedReader(
                            new InputStreamReader(
                                    new FileInputStream (
                                            new File(songFileName))));
         Song song = Song.read(br);  // necessità di una classe Song
         while(song != null) {
                 if (artist.equals(song.getArtist()))
                         System.out.println(song.getTitle());
                 song = Song.read(br);
         }
    }
```

52

La classe Song

- L'interfaccia e le variabili d'istanza

```
class Song {
    // Metodi
    static public Song read(BufferedReader br) throws Exception {…}
    public Song(String title, String artist) {…}
    String getTitle() {…}
    String getArtist() {…}

    // Variabili d'istanza
    String title, artist;
}
```

53

La classe Song

- Implementazione del metodo read

```
static public Song read(BufferedReader br) throws Exception
{
        String title = br.readLine();
        if (title == null)
                return null;
        String artist = br.readLine();
        return new Song(title, artist);
}
```

54

La classe Song

- Implementazione del costruttore e degli altri metodi

```
public Song(String title, String artist) {
        this.title = title;
        this.artist = artist;
}

String getTitle() {
        return this.title;
}

String getArtist() {
        return this.artist;
}
```

55

Discussione

- In questo esempio Song è subordinato a SongDB
  - Quindi si è preferito concentrarsi prima su SongDB e poi si è vista la necessità di introdurre la classe Song (perché usata nell'implementazione dei metodi)
  - Il procedimento di sviluppo delle classi è di natura iterativa …

56

Gestione di valori multipli

- Il metodo lookUp deve scorrere il file ogni volta che viene invocato
- Per migliorare l'efficienza si può pensare di mantenere in memoria il contenuto del file
- Non si conosce a priori il numero di canzoni nel file
  - non si conosce il numero di variabili da dichiarare
- Occorre dichiarare una collezione di oggetti
  - Un gruppo di oggetti che può essere dichiarato come entità singola

57

Esempi di applicazione

- Leggere una lista di nomi e stamparla in ordine inverso
- Leggere dei nomi e stamparli in ordine alfabetico
- Verificare se i voti degli studenti sono al di sopra o al di sotto della media
- Restituire le diverse parole contenute in un file con il relativo numero di occorrenze

- In questi casi non è solo un problema di efficienza …

58

Collezioni di oggetti: Vector

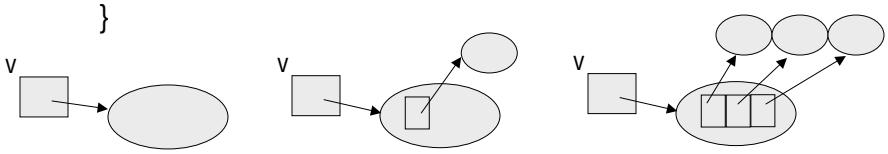
- Operazioni caratteristiche
  - Creare una nuova collezione (costruttore)
  - Aggiungere un oggetto ad una collezione
  - Elaborare gli oggetti della collezione
- Java fornisce diverse classi collezione
  - Contenute nel pacchetto java.util
- La più semplice classe collezione è la classe Vector
  - Creazione: Vector v = new Vector();

59

Aggiungere oggetti a un Vector

- Leggiamo alcuni oggetti String e aggiungiamoli al Vector v

```
String s;
s = br.readline();  // Legge la prima stringa
while (s != null) {
    v.addElement(s);        // Elaborazione: aggiunge s a v
    s = br.readline();      // Legge la stringa successiva
  }
```



Vector v = new Vector(); v.addElement(s); Più applicazioni di addElement

60

Muoversi all'interno di un Vector

- *Attraversare* la collezione di oggetti e *visitare* (elaborare) ciascun elemento
  - E' un processo ripetitivo e quindi viene usata una struttura di ciclo

    while(*ci sono ancora elementi da visitare*)

    visitare l'elemento successivo
- Per poter visitare tutti gli elementi di un Vector è necessario:
  - Raggiungere in qualche modo gli elementi
  - Ottenere l'elemento successivo
  - Verificare se ci sono altri elementi da visitare
- La classe Vector non fornisce questi metodi …

61

Enumeration

- Java offre una classe *Enumeration* che modella l'attraversamento
  - In realtà non si tratta proprio di una classe
  - Ogni classe collezione fornisce un metodo che crea e restituisce un riferimento a un oggetto della classe Enumeration
  - L'oggetto Enumeration fornisce i metodi per ottenere l'elemento successivo e verificare se ci sono altri elementi
- La classe Vector contiene un metodo, *elements*, che restituisce un riferimento ad un oggetto Enumeration
  - Enumeration elements() {…} // Restituisce un Enumeration per Vector
- Enumeration fornisce i seguenti metodi
  - boolean hasMoreElements() // Restituisce true se ci sono altri elementi da // visitare; restituisce false altrimenti
  - Object nextElement() //Restituisce un riferimento all'elemento successivo

62

La classe Object

- nextElement restituisce un riferimento ad un oggetto di classe Object
- La classe predefinita Object modella un qualsiasi oggetto
  - I riferimenti a Name, Employee, String, ecc. sono anche riferimenti a Object
- In questo modo nextElement può restituire riferimenti a oggetti di qualsiasi classe
  - C'è un prezzo da pagare per questa flessibilità …

63

Lavoro supplementare

- Il prezzo da pagare per tale flessibilità:
  - Effettuare un'operazione di cast davanti ad ogni invocazione di nextElement

```
Name n = new Name("Gerald", "Weiss");
Vector v = new Vector();
v.addElement(n);
Enumeration e = v.elements();      // trasforma un Vector in una Enumeration
Name n2 = (Name) e.nextElement();  // nextElement restituisce un riferimento
                                   // ad Object che viene trasformato in un
                                   // riferimento a Name mediante cast
```

cast

  - Assicurarsi che in ogni oggetto Vector si aggiungono riferimenti ad un solo tipo di oggetto

64

Il ciclo di attraversamento

- Il ciclo di attraversamento di una collezione

```
Enumeration enum = ottenere un riferimento Enumeration dalla collezione
while(enum.hasMoreElements()) {
    ClassName x = (ClassName) enum.nextElement();    // Estrae gli elementi
    elaborare x
}
```

- Un esempio: stampiamo le String di un Vector

```
Enumeration enum = v.elements();
while(enum.hasMoreElements()) {
    String s = (String) enum.nextElement();
    System.out.print(s);
  }
```

65

Rivisitiamo la classe SongDB

- Miglioriamo l'implementazione lasciando inalterata l'interfaccia.
  - Il contenuto del file viene caricato in un Vector dal costruttore
  - Il metodo lookUp attraverserà la collezione

```
import java.io.*;
import java.util.*;    // perché stavolta usiamo un Vector

class SongDB {
    public SongDB(String songFileName) throws Exception {...}
    void lookUp(String artist) {...}
    // Variabili di istanza
    private Vector songColl;
}
```

66

Il Costruttore

```
public SongDB(String songFileName) throws Exception {
    songColl = new Vector();
    BufferedReader br = new BufferedReader(
                            new InputStreamReader(
                                    new FileInputStream (
                                            new File(songFileName))));
        Song song = Song.read(br);
        while(song != null) {
                songColl.addElement(song);
                song = Song.read(br);
        }
}
```

67

Il metodo lookUp

```
void lookUp(String artist) {
        Enumeration enum = songColl.elements();
        while(enum.hasMoreElements()) {
                Song song = (Song) enum.nextElement();
                if (artist.equals(song.getArtist()))
                        System.out.println(song.getTitle());
        }
    }
```

68

Esempio2: Database Studenti e loro media voti

- Si supponga di avere un database di studenti universitari, gestito tramite un file che memorizza i tipi dati di un registro.
- Per ogni studente il file contiene la coppia di linee:
  nome
  media
- Vogliamo stabilire quali studenti siano sopra e quali sotto la media della classe

69

Determinazione dell'oggetto primario

- I nomi sono: *file del registro*, *studente*, *nome* e *media*.
  - Nome e media sono subordinati a studente che a sua volta è subordinato a file del registro
- Introduciamo la classe StudentRoster, associata al file d'ingresso
  - Notare l'analogia con la classe SongDB
  - In entrambi i casi gli oggetti subordinati sono letti dal file associato all'oggetto primario

70

Determinazione del comportamento

- Ovviamente abbiamo un costruttore
  - StudentRoster
- E un metodo che percorre il registro e valuta gli studenti (ci dice se stanno sopra o sotto la media)
  - evaluate
- Non è stato ancora detto quando si intende calcolare la media, né se si userà una collezione
  - Questi aspetti riguardano l'implementazione dei metodi e la definizione delle variabili d'istanza

71

Definizione dell'interfaccia

- Utilizzo dei metodi

```
StudentRoster roster = new StudentRoster("CS1.f98");
roster.evaluate();
```

- L'interfaccia

```
class StudentRoster {
   public StudentRoster(String rosterFileName) throws Exception {… }
   public void evaluate() {… }
}
```

72

Definizione delle variabili d'istanza (1)

- Occorre valutare la media della classe
  - Può essere fatto dal costruttore e l'informazione può essere mantenuta in una variabile d'istanza (in tal modo può essere usata dal metodo evaluate)
- Le informazioni relative agli studenti sono richieste due volte:
  - Per calcolare la media (nel costruttore)
  - Per valutare gli studenti (nel metodo evaluate)
  - Conviene usare una variabile di istanza (una collezione)

73

Definizione delle variabili d'istanza (2)

```
class StudentRoster {
        StudentRoster(String rosterFileName) { … }
        void evaluate() {… }

        private Vector studentColl;
        private int classAverage;
      }
```

74

Implementazione del costruttore

```
StudentRoster(String rosterFileName)
                                    throws Exception {
        studentColl = new Vector();
        BufferedReader br =
                new BufferedReader(new InputStreamReader(
                        new FileInputStream(new File(rosterFileName))));
        int total = 0;          // somma le medie
        int count = 0;          // conta gli studenti
        Student student = Student.read(br); // introduzione della classe Student
        while (student != null) {       // struttura di ciclo lettura/elaborazione
                total += student.getAverage();
                count++;
                studentColl.addElement(student);
                student = Student.read(br);
        }
        classAverage = total / count;
}
```

75

Implementazione di evaluate

```
void evaluate() {
        Enumeration enum = studentColl.elements();
        while (enum.hasMoreElements()) {  // struttura di ciclo enumerare
                Student student = (Student)enum.nextElement();
                System.out.print(student.getName());
                System.out.print(" is performing ");
                if (student.getAverage() >= classAverage)
                        System.out.println("above average");
                else
                        System.out.println("below average");
        }
}
```

76

Uso della classe

```
class Evaluator {
    public static main(String[] args) throws Exception {
                StudentRoster roster1 = new StudentRoster(“CS1.f98”);
                roster1.evaluate();
                StudentRoster roster2 = new StudentRoster(“CS2.f98”);
                roster2.evaluate();
    }
```

- Considerazioni conclusive
  - Occorre costruire la classe Student (farlo come esercizio …)
  - Possibili diverse implementazioni (ad esempio, scorrere più volte il file invece di mantenere la collezione in memoria) …
  - in pratica un file non è altro che una collezione residente su disco