UNIVERSITA' DEGLI STUDI DI BRESCIA
Facoltà di Economia

Eserciziario del corso di Algoritmi e Programmazione

Enrico Angelelli , Denis Ruggeri
angele@eco.unibs.it , ruggeri@eco.unibs.it

A.A. 2008/09

Indice

1 Scheda 01 - Algoritmi **7**

1.1 Analisi del problema: Tassazione persone fisiche 8

1.2 Analisi del problema: No tax area 14

1.3 Analisi del problema: Interesse composto 18

1.4 Proposte di lavoro 20

2 Scheda 02 - Operatori e tipi primitivi **21**

2.1 Operatori di confronto 22

2.2 Operatori logici 24

2.3 Metodi per l'Input/Output 27

2.4 Tipi primitivi 31

2.5 Proposte di lavoro 34

3 Scheda 03 - Costrutti della programmazione strutturata **35**

3.1 Istruzione IF 36

3.2 Istruzione SWITCH 38

3.3 Istruzione WHILE 40

3.4 Istruzione DO....WHILE 44

3.5 Istruzione FOR 46

3.6 Proposte di lavoro 48

4 Scheda 04 - Array **49**

4.1 Struttura a menu con DO...WHILE e SWITCH 50

4.2 Array: inserimento,visualizzazione,elaborazione 53

4.3 Array: ricerca semplice, determinazione massimo, elaborazione 56

4.4 Array: booleani, relazione posizionali multiple 60

4.5 Proposte di lavoro 65

5 Scheda 05 - Array multidimensionali, file, database **67**
5.1 Array multidimensionali: Test di selezione . . . 68
5.2 Gestione files . . . 72
5.3 Gestione database . . . 76
5.4 Proposte di lavoro . . . 80

6 Scheda 06 - Metodi, package **81**
6.1 Introduzione ai metodi . . . 82
6.2 Metodi classici su array e metodi ricorsivi . . . 87
6.3 Package: utilizzo del package vettori . . . 94
6.4 Package: creazione del package formule . . . 96
6.5 La documentazione . . . 98
6.6 Proposte di lavoro . . . 100

7 Scheda 07 - Oggetti **101**
7.1 Oggetti: Impiegato . . . 102
7.2 Oggetto: Veicolo . . . 107
7.3 Oggetto: Conto corrente . . . 109
7.4 Proposte di lavoro . . . 113

8 Scheda 08 - Collezioni **115**
8.1 Collezioni: Listino azionario . . . 116
8.2 Collezioni: Portafoglio clienti . . . 122
8.3 Mappe: Indagine clienti . . . 126
8.4 Proposte di lavoro . . . 131

9 Scheda 09 - Aggregazioni **133**
9.1 Oggetti: aggregazioni . . . 134
9.2 Proposte di lavoro . . . 142

10 Scheda 10 - Oggetti e database **143**
10.1 Visione d'insieme: Gestione del conto corrente con database . . . 144

11 *Appendice uno - Appello 11/01/05* **155**
11.1 Appello 11/01/05 . . . 156

12 *Appendice due - Appello 10/01/06* **163**
12.1 Appello 10/01/06 . . . 164

13 *Appendice tre - Appello 09/01/07* **173**
13.1 Appello 09/01/07 174

14 *Appendice quattro - Appello 08/01/08* **183**
14.1 Appello 08/01/08 184

15 *Appendice cinque - Progetto Algoritmi A.A. 05/06* **193**
15.1 Progetto A.P. 2005/2006 194

16 *Appendice A - Il pacchetto grafico MicroCad* **197**
16.1 Il package Microcad 198

16.2 La classe Avvisi 202

Capitolo 1

Argomenti trattati

- Analisi del problema: Tassazione persone fisiche
- Analisi del problema: No tax area
- Analisi del problema: Interesse composto
- Proposte di lavoro

1.1 Analisi del problema: Tassazione persone fisiche

Il sistema in vigore dal 1 gennaio 2003 (fino alla riforma del 2005) prevedeva, cinque scaglioni di reddito e relative aliquote. La tabella riassuntiva di aliquote e scaglioni di reddito imponibile è la seguente :

Scaglione	Aliquota
fino a 15.000 euro	23%
oltre 15.000 euro e fino a 29.000 euro	29%
oltre 29.000 euro e fino a 32.600 euro	31%
oltre 32.600 euro e fino a 70.000 euro	39%
oltre 70.000 euro	45%

La progressività dell'imposta viene garantita in base ad un sistema di deduzioni. Il reddito imponibile viene ottenuto come mostra lo schema seguente:

Reddito Complessivo
+ Crediti d'imposta su dividendi
- Oneri deducibili
- Deduzione abitazione principale
- No tax area
=
Reddito Imponibile

L'imposta determinata è un'imposta lorda dalla quale devono essere tolti tutti gli oneri detraibili e le detrazioni d'imposta per carichi famigliari. Si riportano due esempi di tassazione, rimandando lo studio approfondito e l'analisi delle casistiche di deduzione e detrazione alle discipline di pertinenza.

Esempio 1
Lavoratore dipendente, senza famigliari a carico, con reddito annuo di 20.000 euro ed oneri deducibili di 1.500 euro.

Reddito Complessivo	20.000,00	
		- 1.500,00 (Oneri deducibili)
		- 4.326,75 (No tax area)
Reddito Imponibile	14.173,25	
Imposta Dovuta	- 3.259,85	(Aliquota 23%)

Esempio 2
Contribuente con coniuge e un figlio a carico, un reddito da lavoro dipendente di 28.000 euro

Reddito Complessivo	28.000,00	
		- 1.586,85 (No tax area)
Reddito Imponibile	26.413,75	
Imposta		- 6.759,99 (Irpef)
Imposta	6.759,99	
		-130,00 (Detrazione reddito lavoro dipendente)
		-1.013,06 (Detrazione per famigliari a carico)
Imposta dovuta	5.616,93	

Analisi del problema.

Premesso che la presenza di detrazioni e deduzioni è un fattore strettamente legato alla soggettività fiscale dei vari contribuenti, è possibile analizzare un procedimento per calcolare l'ammontare dell'Irpef dovuta. L'algoritmo per arrivare a determinare l'imposizone fiscale appare piuttosto elementare. I dati necessari come ingresso sono quelli relativi a reddito complessivo, detrazioni e deduzioni. Dal reddito complessivo si determina il reddito imponibile (sottraendo dal reddito complessivo le deduzioni previste). Il reddito imponibile viene sottoposto ad una selezione per individuare la fascia di contribuzione relativa e sulla base di questa viene applicata la percentuale di tassazione prevista. Dall'imposta così ottenuta vengono tolte le detrazioni. La tassazione avviene come percentuale in modo progressivo:

Esempio 1
Reddito Imponibile di 10.000 euro

Il reddito rientra totalmente nella prima fascia quindi l'imposta applicata sarà 10.000*0,23 = 2300 euro di imposta lorda

Esempio 2
Reddito Imponibile di 20.000 euro
La fascia di pertinenza è la seconda quindi con una percentuale del 29%. Risulta assoggettato a questa aliquota l'importo che eccede le fasce precedenti. Il reddito che rientra nella prima fascia fino al massimale della fascia è 15.000 euro quindi l'imposizione fiscale sarà 15.000*0,23=3450 euro. Il reddito che rientra nella seconda fascia fino al massimale della fascia sarà 20.000-15000 (rientrato nella tassazione precedente)=5.000 euro. A questo importo viene applicata l'aliquota del 29%, quindi 5.000*0,29=1450 euro. Il totale dell'imposta dovuta sarà quindi 3450+1450 euro = 4900 euro come riepilogato dallo schema sottostante:

Reddito imponibile	20.000		
scaglione di competenza	importo competenza	aliquota competenza	ammontare
fino a 15.000 euro	15000	23%	3450
oltre 15.000 euro e fino a 29.000 euro	5000 (20.000-15000)	29%	1450
Imposta lorda			4900

Esempio3
Reddito di 80.000 euro

Reddito imponibile	80.000		
scaglione di competenza	importo competenza	aliquota competenza	ammontare
fino a 15.000 euro	15.000	23%	3.450
oltre 15.000 euro e fino a 29.000 euro	14.000	29%	4.060
oltre 29.000 euro e fino a 32.600 euro	3.600	31%	1.116
oltre 32.600 euro e fino a 70.000 euro	37.400	39%	14.586
oltre 70.000 euro	10.000	45%	4.500
Imposta lorda			27.712

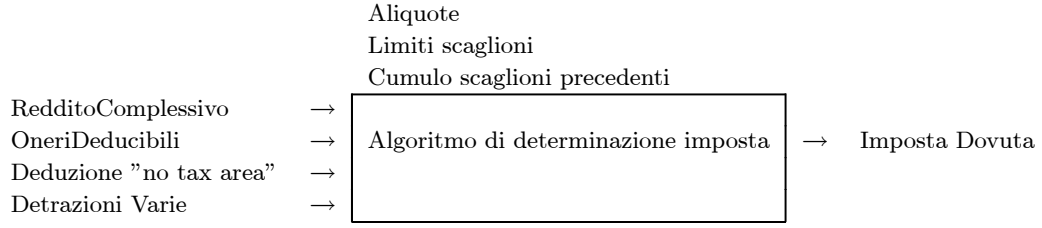
Dalle tabelle relative agli esempi precedenti è possibile notare come l'imposta sia costituita da una parte fissa relativa alla quota degli scaglioni precedenti più una certa parte determinata dall'aliquota dello scaglione corrente, è quindi possibile schematizzare una tabella di calcolo come segue:

Scaglione di competenza	Aliquota	Imposta sui redditi intermedi
fino a 15.000 euro	23%	23% sull'intero importo
oltre 15.000 euro e fino a 29.000 euro	29%	3.450,00 + 29% sulla parte eccedente i 15.000 euro
oltre 29.000 euro e fino a 32.600 euro	31%	7.510,00 + 31% sulla parte eccedente i 29.000 euro
oltre 32.600 euro e fino a 70.000 euro	39%	8.626,00 + 39% sulla parte eccedente i 32.600 euro
oltre 70.000 euro	45%	23.212,00 + 45% sulla parte eccedente i 70.000 euro

gli esempi 1,2,3 vengono ridotti come segue:

Esempio 1	Reddito: 10.000	Imposta: 10.000*0,23 = 2300,00
Esempio 2	Reddito: 20.000	Imposta: 3.450,00+ (20.000-15000)*0,29 = 4.900,00
Esempio 3	Reddito: 80.000	Imposta: 23.212,00+ (80.000-70.000)*0,45 = 27.712,00

In questo algoritmo: reddito complessivo, oneri deducibili, deduzione relativa alla "no tax area", detrazioni da reddito di lavoro dipendente e per famigliari a carico sono le quantità in ingresso mentre l'ammontare dell'aliquota è il risultato della procedura di calcolo. L'importo limite degli scaglioni, le aliquote e il cumulo degli scaglioni precedenti, possono essere considerati delle costanti nel contesto della procedura di calcolo.
Schematicamente possiamo rappresentare questa situazione come segue:



Analisi dell'algoritmo tramite Meta Linguaggio

Algoritmo:	CalcolaImposta	
Dati di input:	redditoComplessivo	Numero reale
	oneriDeducibili	Numero reale
	deduzioniNoTaxArea	Numero reale
	detrazioneLavoro	Numero reale
	detrazioneFamigliari	Numero reale
Costanti di sistema	LIMITE_SCAGLIONE_UNO	Numero reale
	ALIQUOTA_SCAGLIONE_UNO	Numero intero
	LIMITE_SCAGLIONE_DUE	Numero reale
	ALIQUOTA_SCAGLIONE_DUE	Numero intero
	LIMITE_SCAGLIONE_TRE	Numero reale
	ALIQUOTA_SCAGLIONE_TRE	Numero intero
	LIMITE_SCAGLIONE_QUATTRO	Numero reale
	ALIQUOTA_SCAGLIONE_QUATTRO	Numero intero
	ALIQUOTA_SCAGLIONE_CINQUE	Numero intero
	CUMULO_SCAGLIONE_UNO	Numero reale
	CUMULO_SCAGLIONE_DUE	Numero reale
	CUMULO_SCAGLIONE_TRE	Numero reale
	CUMULO_SCAGLIONE_QUATTRO	Numero reale
Variabili utilizzate	impostaLorda	Numero reale
	impostaDovuta	Numero reale
Risultato	redditoImponibile	Numero reale

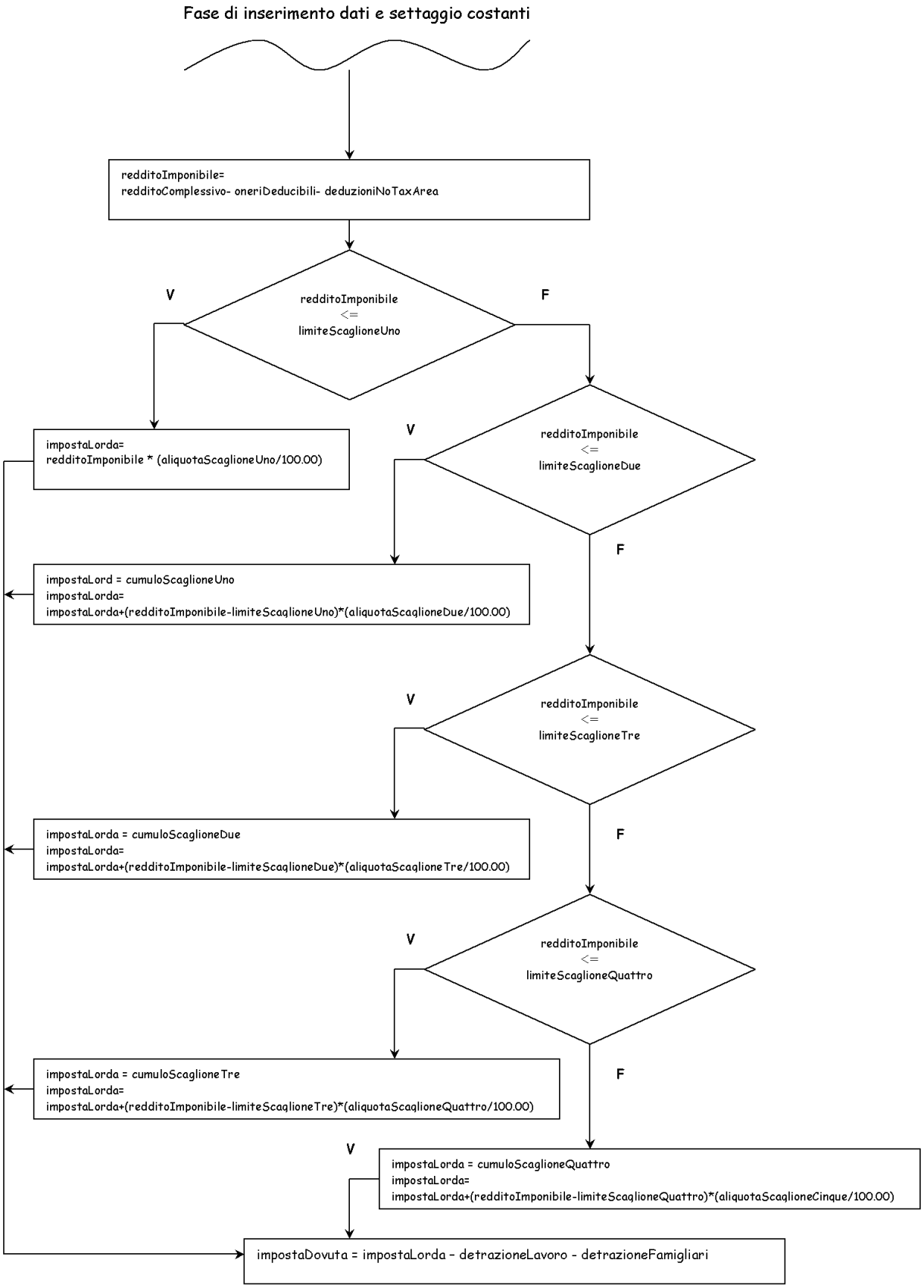
```
INIZIO_ALGORITMO

LIMITE_SCAGLIONE_UNO <- 15.000,00
ALIQUOTA_SCAGLIONE_UNO <- 23
LIMITE_SCAGLIONE_DUE <- 29.000,00
ALIQUOTA_SCAGLIONE_DUE <- 29
LIMITE_SCAGLIONE_TRE <- 32.600,00
ALIQUOTA_SCAGLIONE_TRE <- 31
LIMITE_SCAGLIONE_QUATTRO <- 70.000,00
ALIQUOTA_SCAGLIONE_QUATTRO <- 39
ALIQUOTA_SCAGLIONE_CINQUE <- 45
CUMULO_SCAGLIONE_UNO <- 3.450,00
CUMULO_SCAGLIONE_DUE <- 7.510,00
CUMULO_SCAGLIONE_TRE <- 8.626,00,
CUMULO_SCAGLIONE_QUATTRO <- 23.212,00
redditoImponibile <- 0
impostaLorda <- 0
redditoImponibile<-redditoComplessivo-oneriDeducibili-deduzioniNoTaxArea
SE redditoImponibile<=LIMITE_SCAGLIONE_UNO
  ALLORA
    impostaLorda<-(redditoImponibile)*(ALIQUOTA_SCAGLIONE_UNO/100)
  ALTRIMENTI
      SE redditoImponibile <= LIMITE_SCAGLIONE_DUE
        ALLORA
          impostaLorda <- CUMULO_SCAGLIONE_UNO
          impostaLorda <- impostaLorda+
          (redditoImponibile-LIMITE_SCAGLIONE_UNO)*(ALIQUOTA_SCAGLIONE_DUE/100)
        ALTRIMENTI
          SE redditoImponibile <= LIMITE_SCAGLIONE_TRE
            ALLORA
              impostaLorda <- CUMULO_SCAGLIONE_DUE
              impostaLorda <- impostaLorda+
              (redditoImponibile-LIMITE_SCAGLIONE_DUE)*(ALIQUOTA_SCAGLIONE_TRE/100)
```

```
            ALTRIMENTI
              SE redditoImponibile <= LIMITE_SCAGLIONE_QUATTRO
                ALLORA
                  impostaLorda <- CUMULO_SCAGLIONE_TRE
                  impostaLorda <- impostaLorda+
                  (redditoImponibile-LIMITE_SCAGLIONE_TRE)*(ALIQUOTA_SCAGLIONE_QUATTRO/100)
                ALTRIMENTI
                  impostaLorda <- CUMULO_SCAGLIONE_QUATTRO
                  impostaLorda <- impostaLorda+
                  (redditoImponibile-LIMITE_SCAGLIONE_QUATTRO)*(ALIQUOTA_SCAGLIONE_CINQUE/100)
              FINESE
          FINESE
      FINESE
FINESE
impostaDovuta<- ImpostaLorda-detrazioniLavoro-detrazioniFamigliari
Stampa impostaDovuta

FINE_ALGORITMO
```

Analisi dell'algoritmo tramite Flow Chart



Con la Finanziaria del 2005 il sistema di tassazione è stato modificato come segue:

L'IRPEF DEL 2005: LE NUOVE REGOLE

Come si è accennato precedentemente, la legge Finanziaria per il 2005 ha introdotto il secondo modulo di riforma dell'Irpef che prevede principalmente:

- nuove aliquote e scaglioni di reddito;
- trasformazione delle detrazioni per familiari a carico in deduzioni dal reddito;
- soppressione delle detrazioni, introdotte con il primo modulo della riforma fiscale, che erano concesse in relazione alla tipologia di reddito posseduta (lavoro dipendente, pensione, lavoro autonomo e impresa) e solo se il reddito complessivo era compreso in determinate fasce;
- estensione della clausola di salvaguardia.

(Fonte: Agenzia delle Entrate)

Tab. 15 - LA TABELLA IRPEF 2005 PER IL CALCOLO DELL'IMPOSTA

Redditi	Aliquote	Imposta sui redditi intermedi
fino a 26.000 euro	23%	23% sull'intero importo (pari a 5.980 euro)
oltre 26.000 fino a 33.500 euro	33%	5.980 + 33% sulla parte eccedente 26.000 euro
oltre 33.500 fino a 100.000 euro	39%	8.455 + 39% sulla parte eccedente 33.500 euro
oltre 100.000 euro	39% + 4%	34.390 + 4% sulla parte eccedente 100.000 euro

(Fonte: Agenzia delle Entrate)

1.2 Analisi del problema: No tax area

Un passaggio ulteriore nell'algoritmo della deteminazione dell'imposta può essere costituito dalla determinazione in modo automatico della deduzione relativa alla "no tax area" che è stata introdotta per garantire un prelievo fiscale progressivo nelle persone fisiche. La "no tax area" (tecnicamente definita come "deduzione per assicurare la progressività dell'imposizione") è quella parte di reddito non sottoposta a tassazione. A tutti i contribuenti si riconosce, indipendentemente dalla tipologia di reddito posseduto e dal periodo di lavoro svolto durante l'anno una "deduzione base" del reddito pari a 3.000 euro. Questo importo aumenta a seconda della tipologia di reddito come segue:

+ 4.500 euro per i lavoratori dipendenti
+ 4.000 euro per i pensionati
+ 1.500 euro per i lavoratori autonomi

in relazione al periodo temporale in cui si è lavorato.

In linea teorica possono essere applicate le seguenti deduzioni:

Deduzione teorica massima	Tipologia contribuente
7.500 euro	Lavoratori dipendenti (3000 base + 4500)
7.000 euro	Pensionati (3000 base + 4000)
4.500 euro	Lavoratori autonomi e imprenditori (3000 base + 1500)

Quindi un lavoratore dipendente con un reddito fino a 7.500 euro non pagherà nulla come irpef, mentre superata questa soglia, la misura della deduzione effettivamente spettante diminuirà progressivamente in relazione al reddito complessivo fino ad azzerarsi del tutto. La deduzione spettante può essere determinata con il seguente procedimento:

Algoritmo determinazione No Tax Area

→ si somma all'importo di 26.000 euro la massima deduzione "no tax area" prevista per quel profilo di contribuente

→ si sommano le deduzioni relative all'abitazione principale e gli altri oneri deducibili (contributi previdenziali ed assistenziali, assegni coniuge separato, divorziato etc ect)

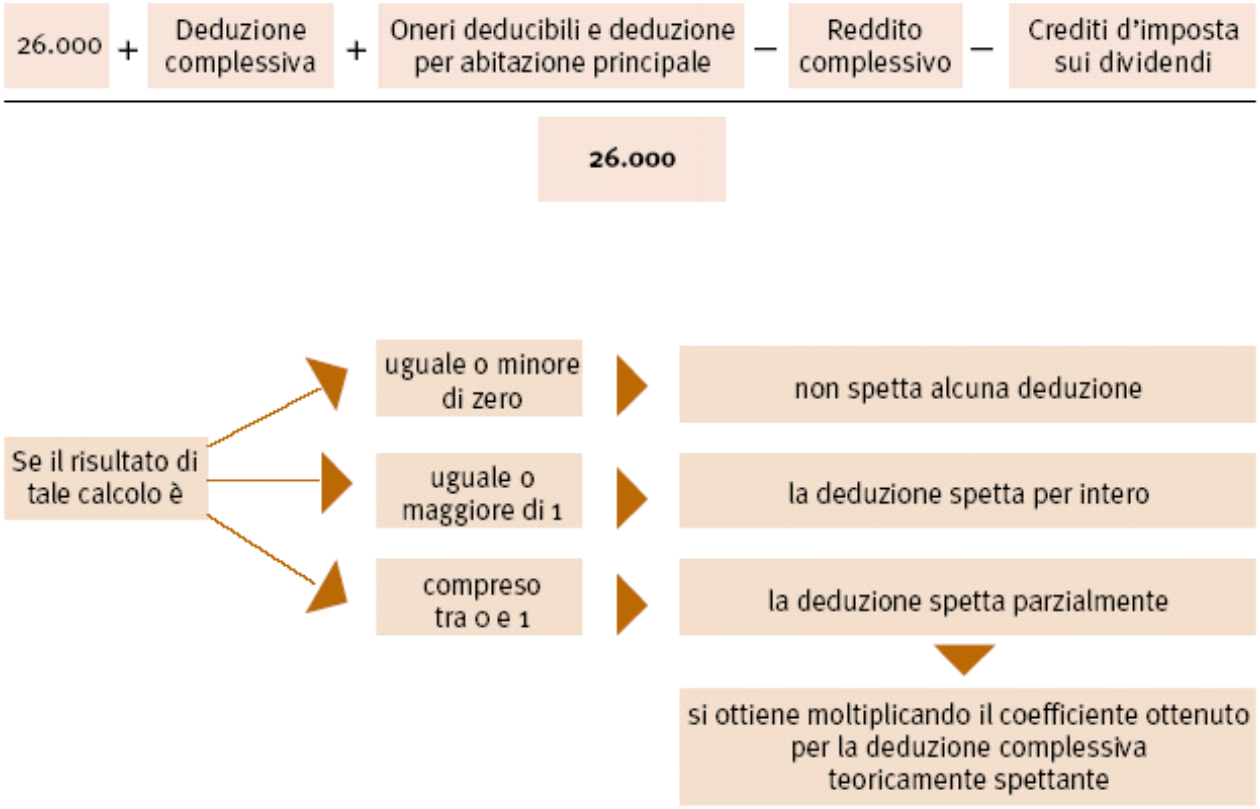
→ si sottrae il reddito complessivo ed eventuali crediti d'imposta sui dividendi

→ si divide il tutto per 26.000

→ a seconda del risultato:

se il risultato è $<=$ a zero non spetta alcuna deduzione
se il risultato è $>=$ a uno la deduzione spetta per intero
se il risultato è compreso fra zero e uno allora la deduzione spetta parzialmente in una misura calcolata moltiplicando tale risultato (considerando le prime quattro cifre ed applicando il troncamento) per la deduzione teoricamente spettante

In questo modo il contribuente deve solo fornire al procedimento di calcolo l'indicazione sulla propria tipologia contributiva e i dati sul reddito e delegare all'algoritmo la determinazione della deduzione di "no tax area".

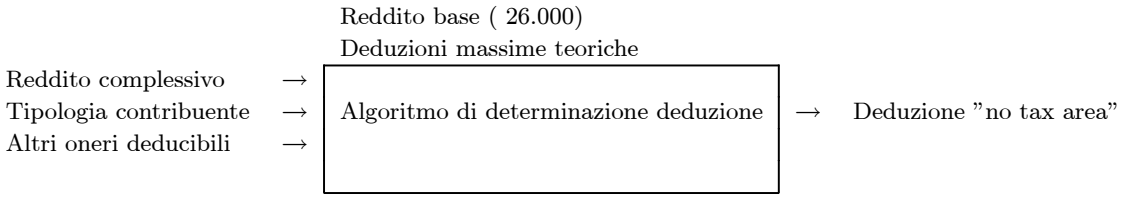


Esempio 1
Lavoratore dipendente con reddito annuo di 36.000 euro ed oneri deducibili per 1.500 euro, il calcolo diventa:
$\frac{26:000+7:500+1:500-36:000}{26:000} = -0;0385$
essendo il risultato minore di zero non spetta nessuna deduzione relativa alla "no tax area"

Esempio 2
Pensionato con reddito annuo di 8.000 euro ed oneri deducibili per 1.500 euro, il calcolo diventa:
$\frac{26:000+7:000+1:500-8:000}{26:000} = 1:0192$
essendo il risultato $>= 1$ la deduzione relativa alla "no tax area" spetta per intero

Esempio 3
Lavoratore dipendente con reddito annuo di 20.000 euro ed oneri deducibili per 1.500 euro, il calcolo diventa:
$\frac{26:000+7:500+1:500-20:000}{26:000} = 0:5769$
La deduzione spettante sarà quindi:
7.500 (deduzione teorica massima) * 0:5769 (percentuale di pertinenza) = 4.326,75

Analisi del problema.
Le componenti necessarie per la determinazione della "no tax area" sono: il reddito complessivo, la tipologia di contribuenza (dipendente,pensionato,autonomo) e gli altri oneri deducibili. Il reddito base per il calcolo (26.000) e i massimali di deduzione teorica possono essere considerati come delle costanti del sistema. Consideriamo intervalli temporali annuali tralasciando la determinazione di pro quota relativi a durate contributive inferiori all'anno.



Per comodità possiamo condiderare le tipologie di contribuzione associate ad un codice numerico come segue:

Tipologia contribuente	Codice associato
Lavoratore dipendente	0
Pensionato	1
Lavoratore autonomo	2

Analisi dell'algoritmo tramite Meta Linguaggio

Algoritmo:	CalcolaNoTaxArea	
Dati di input:	redditoComplessivo	Numero reale
	oneriDeducibili	Numero reale
	tipologiaContribuente	Numero reale
Costanti di sistema	REDDITO_BASE	Numero reale
	DEDUZIONE_MAX_DIPENDENTE	Numero reale
	DEDUZIONE_MAX_PENSIONATO	Numero reale
	DEDUZIONE_MAX_AUTONOMO	Numero reale
Variabili utilizzate	massimaleDiCompetenza	Numero reale
	coefficienteDeduzione	Numero reale
Risultato	deduzioneNoTaxArea	Numero reale

```
INIZIO_ALGORITMO
REDDITO_BASE <- 26.000,00
DEDUZIONE_MAX_DIPENDENTE <- 7.500,00
DEDUZIONE_MAX_PENSIONATO <- 7.000,00
DEDUZIONE_MAX_AUTONOMO <- 4.000,00
massimaleDiCompetenza <- 0
deduzioneNoTaxArea <- 0
SE tipologiaContribuente = 0
  ALLORA
    massimaleDiCompetenza <- DEDUZIONE_MAX_DIPENDENTE
  ALTRIMENTI
    SE tipologiaContribuente = 1
      ALLORA
        massimaleDiCompetenza <- DEDUZIONE_MAX_PENSIONATO
      ALTRIMENTI
        massimaleDiCompetenza <- DEDUZIONE_MAX_AUTONOMO
    FINESE
FINESE
coefficienteDeduzione <- (REDDITO_BASE+massimaleDiCompetenza+oneriDeducibili-redditoComplessivo)
                         ------------------------------------------------------------------------
                                                    REDDITO_BASE
SE coefficienteDeduzione <= 0
  ALLORA
    deduzioneNoTaxArea <- 0
  ALTRIMENTI
    SE coefficienteDeduzione >=1
      ALLORA
        deduzioneNoTaxArea <- massimaleDiCompetenza
      ALTRIMENTI
        deduzioneNoTaxArea <- massimaleDiCompetenza*coefficienteDeduzione
    FINESE
FINESE
Stampa deduzioneNoTaxArea

FINE_ALGORITMO
```

Lavoro in aula / laboratorio

- Realizzare il flowchart relativo alla determinazione della No tax area

1.3 Analisi del problema: Interesse composto

Supponendo di versare la somma di 10.000 euro in banca ad un tasso del'1% annuo in quanto si può stimare il deposito dopo 4 anni ? (Tralasciamo per ora considerazioni tecnico/fiscali come variazione tasso d'interesse e tassazione a fine anno degli interessi attivi).

Il primo anno la somma a disposizione sarà:
Capitale a disposizione = Capitale iniziale + Interessi maturati sul capitale iniziale
$10000 + 10000 * \frac{1}{100} = 10\,100$
Il secondo anno la somma sarà:
Capitale a disposizione = Capitale inizio anno + Interessi maturati sul capitale inizio anno
$10100 + 10100 * \frac{1}{100} = 10\,201$
Il terzo anno la somma sarà:
Capitale a disposizione = Capitale inizio anno + Interessi maturati sul capitale inizio anno
$10201 + 10201 * \frac{1}{100} = 10{:}303$:
Il quarto anno la somma sarà:
Capitale a disposizione = Capitale inizio anno + Interessi maturati sul capitale inizio anno
$10303 + 10303 * \frac{1}{100} = 10{:}406$:
L'espressione del singolo calcolo annuale può essere riscritta attraverso il raccoglimento come segue
primo anno
$10000 + 10000 * \frac{1}{100} = 10{:}000 * (1 + \frac{1}{100}) = 10{:}100$
secondo anno
$10{:}100 * (1 + \frac{1}{100}) = 10{:}000 * (1 + \frac{1}{100}) * (1 + \frac{1}{100}) = 10{:}201$
terzo anno
$10{:}201 * (1 + \frac{1}{100}) = 10{:}000 * (1 + \frac{1}{100}) * (1 + \frac{1}{100}) * (1 + \frac{1}{100}) = 10{:}303$
quarto anno
$10{:}303 * (1 + \frac{1}{100}) = 10{:}000 * (1 + \frac{1}{100}) * (1 + \frac{1}{100}) * (1 + \frac{1}{100}) * (1 + \frac{1}{100}) = 10{:}406$
che può essere espressa come
$10{:}000 * (1 + \frac{1}{100})^4 = 10{:}406$

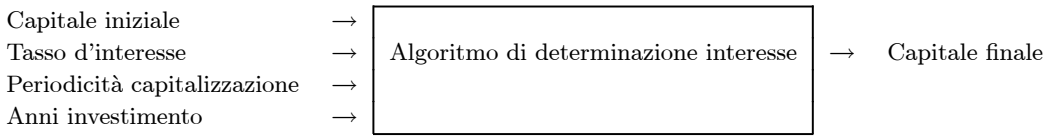
Generalizzando, il valore futuro di un investimento di CI (capitale iniziale) che frutta interessi ad un tasso annuo r (reinvestiti) m volte l'anno per un periodo di t anni determina un CF (capitale finale) pari a:

$$CF = CI * (1 + \frac{r}{m})^{mt}$$

Nell'esempio precedente:
CI=10.000, r=1%, m=1, t=4
$CF = 10{:}000(1 + 0;01)^4 = 10{:}406$

Esempio di deposito a risparmio
Nell'aprile 2002 ING Direct pagava un interesse del 4.3% sul deposito a risparmio denominato conto arancio. Depositando 2.000 euro con capitalizzazione degli interessi trimestrale qual'è l'ammontare dopo sei anni del deposito e quanto si è maturato come interessi[1]
CI= 2.000, r=0,043, m=4, t=6
$CF = 2{:}000 * (1 + \frac{0{:}043}{4})^{4*6} = 2{:}585;12$ euro
Interesse=CF-CI
$Interesse = 2{:}585;12 - 2{:}000 = 585;12$ euro

Analisi del problema.
Una volta forniti i valori in ingresso (CI, r, m, t) bisogna iterare il procedimento di calcolo per un numero di volte pari a mt. Ad ogni passaggio il capitale accumulato fino a qual momento viene moltiplicato per il fattore $(1 + \frac{r}{m})$.



[1] Fonte: Corriere della Sera, 2 aprile 2002, p. 5

Analisi dell'algoritmo tramite Meta Linguaggio

Algoritmo:	CalcolaCapitaleFinale	
Dati di input:	capitaleIniziale	Numero reale
	tassoInteresse	Numero reale
	periodCapitalizzazioni	Numero intero
	anniInvestimento	Numero intero
Variabili utilizzate	capitaleFinale	Numero reale
	contaPeriodi	Numero reale
Risultato	totalePeriodi	Numero reale

```
INIZIO_ALGORITMO

capitaleFinale <- capitaleIniziale
contaPeriodi <- 0
totalePeriodi <- anniInvestimento*periodCapitalizzazioni
QUANDO contaPeriodi<totalePeriodi ESEGUI
  capitaleFinale<-capitaleFinale*(1+(tassoInteresse/100)/periodCapitalizzazioni)
  contaPeriodi=contaPeriodi+1
RIPETI
Stampa capitaleFinale

FINE_ALGORITMO
```

Lavoro in aula / laboratorio

- Realizzare il flowchart relativo all'algortimo di determinazione dell'interesse composto
- Modificare l'algoritmo dell'interesse composto in modo che il tasso d'interesse si incrementi dello 0,01% ad ogni periodo t

1.4 Proposte di lavoro

Proposta Uno

Il capitale CI che lasciamo in gestione alla banca matura gli interessi secondo una percentuale fissa annuale. Ogni anno sugli interessi maturati viene effettuato un prelievo tributario del 12.5%. Modificare l'algoritmo dell'interesse composto in modo da considerare la tassazione sugli interessi attivi.

Proposta Due

Il capitale CI che lasciamo in gestione alla banca matura gli interessi attivi in questo modo:

- se la somma è inferiore a 5.000 euro 1% fisso annuale
- se la somma è compresa fra 5.000 e 10.000 euro 1,5% fisso annuale
- se la somma è superiore a 10.000 euro 2% fisso annuale

Determinare l'algoritmo per il calcolo dell'interesse composto

Proposta Tre

Si supponga di applicare una tassazione proporzionale secondo le seguenti regole differenziate per tipologia di contribuente:

Dipendente	22%	Reddito tassabile (Reddito-no tax area)
Pensionato	18%	Reddito tassabile
Autonomo	25%	Reddito tassabile

per tutte le tipologie è stabilita una soglia di non tassabilità (no tax area) pari a:

Dipendente	30%	Reddito
Pensionato	32%	Reddito
Autonomo	28%	Reddito

Determinare l'algoritmo per la determinazione dell'imposta

Capitolo 2

Argomenti trattati

- Operatori di confronto
- Operatori logici
- Metodi per l'Input/Output
- Tipi primitivi
- Proposte di lavoro

2.1 Operatori di confronto

Questa serie di operatori interviene ogni qualvolta è necessario stabilire una relazione d'ordine. Il risultato dell'applicazione di un operatore di confronto fra due variabili, fra due valori o fra una variabile e un valore, confrontabili fra loro, restituisce un valore booleano che può essere VERO se la relazione d'ordine è verificata o FALSO se la relazione d'ordine non è verificata. Questo risultato booleano viene solitamente utilizzato all'interno di costrutti di selezione.

I classici operatori sono:

Operatore	Relazione d'ordine
>	maggiore
>=	maggiore o uguale
<	minore
<=	minore o uguale
==	uguale

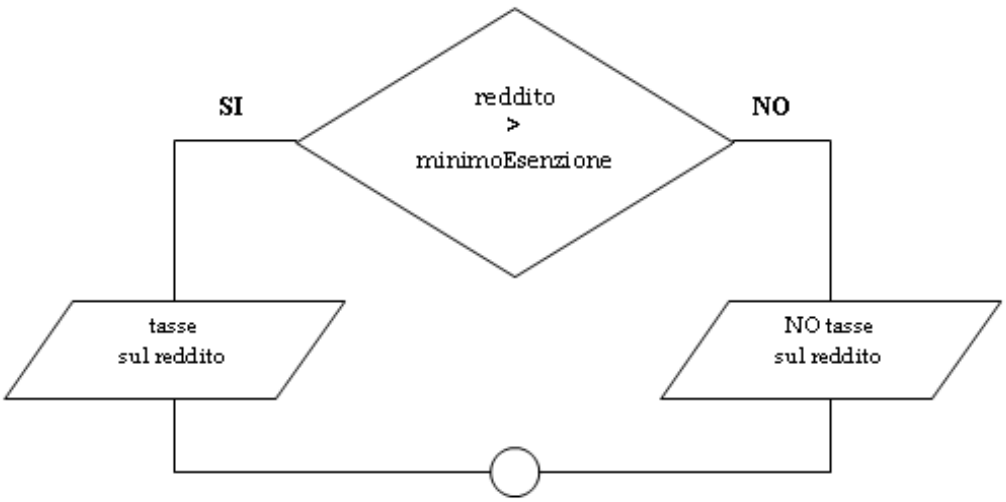
Esempio1

reddito <– 20.000
minimoEsenzione <– 10.000
Relazione booleana
reddito> minimoEsenzione

(Viene confrontato il contenuto della locazione di memoria associata alla variabile reddito con il contenuto della locazione di memoria associata alla variabile minimoEsenzione e restituito il valore VERO o FALSO). La relazione precedente assume il valore VERO in quanto il confronto che viene effettuato è il seguente: 20.000 > 10.000 che rappresenta una verità.

Possibile utilizzo

```
SE reddito>minimoEsenzione
  ALLORA
    "Devi pagare le tasse sul reddito"
  ALTRIMENTI
    "Il tuo reddito rientra nell'esenzione"
FINESE
```



Esempio2

Vogliamo determinare se la proposta che viene fatta sul mutuo a tasso fisso è coerente con le condizioni di mercato. Supponendo il tasso medio di mercato per i mutui a tasso fisso pari a 4.15%:

tassoMutuoProposto <– 4.5
Relazione booleana
tassoMutuoProposto <= 4.15

Il questo caso la relazione non è verificata e il valore booleano in uscita è FALSO. (In linguaggio naturale la risposta alla domanda il tasso proposto è minore del tasso medio di mercato è negativa.)

Possibile utilizzo

```
SE tassoMutuoProposto <= 4.15
  ALLORA
    "Tasso proposto sul mutuo buono rispetto al mercato"
  ALTRIMENTI
    "Tasso proposto sul mutuo alto rispetto al mercato"
FINESE
```

Esempio3

Supponiamo che il programma per il controllo automatico della correttezza delle coordinate bancarie abbia impostato come regola che un conto corrente bancario è corretto se il CIN corrisponde al carattere "C" (esempio):

codiceCin <– "B"
codiceControlloIncrociato <– "C"
Relazione booleana
codiceCin = CodiceControlloIncrociato
La relazione di uguaglianza stretta non è verificata quindi il valore restituito è FALSO

Possibile utilizzo

```
SE codiceCin=codiceControlloIncrociato
  ALLORA
    "Il controllo incrociato sul codice conto corrente non rivela imprecisioni"
  ALTRIMENTI
    "Il controllo incrociato sul codice conto corrente rivela imprecisioni"
FINESE
```

2.2 Operatori logici

Gli operatori logici vengono applicati a uno o più valori booleani e restituiscono un valore di tipo booleano.I principali operatori logici sono riassunti nella tabella che segue:

Operatore	Funzione dell'operatore
NOT	Negazione del valore logico d'ingresso
AND	Verifica del rispetto contemporaneo di tutte le affermazioni logiche proposte
OR	Verifica del rispetto di almeno una delle affermazioni logiche proposte

Esempio1(**NOT**)

L'operatore NOT restituisce il valore booleano opposto al valore booleano in ingresso secondo la seguente tabella :

Valore	NOT(Valore)
FALSO	VERO
VERO	FALSO

Supponiamo di avere una variabile booleana che indica se il contribuente ha diritto alla detrazione per abitazione principale nel pagamento dell'ICI (tale variabile assumerà valore VERO se l'utente ha diritto e valore FALSO se l'utente non ha diritto):

dirittoDetrazione <– VERO

Operatore Logico

NOT[1](dirittoDetrazione)

In questo caso la valutazione della relazione NOT(dirittoDetrazione) restituisce come valore booleano la negazione del valore booleano in ingresso quindi NOT(VERO) corrisponde ad un valore booleano di FALSO.

Possibile utilizzo

Logica positiva (senza NOT)

```
SE dirittoDetrazione=VERO
  ALLORA
    "Detrarre dall'imposta da pagare la quota per l'abitazione principale"
  ALTRIMENTI
    "Nessun diritto alla detrazione abitazione principale"
FINESE
```

Logica negata (con il NOT)

```
SE NOT(dirittoDetrazione=VERO)
  ALLORA
    "Nessun diritto alla detrazione abitazione principale"
  ALTRIMENTI
    "Detrarre dall'imposta da pagare la quota per l'abitazione principale"
FINESE
```

Esempio2 (**AND**)

L'operatore AND[2] restituisce un valore booleano VERO se entrambi i valori booleani in ingresso sono veri, secondo la seguente tabella :

primoValore	secondoValore	primoValore AND secondoValore
FALSO	FALSO	FALSO
FALSO	VERO	FALSO
VERO	FALSO	FALSO
VERO	VERO	VERO

Una situazione in cui dobbiamo avere entrambe le condizioni contemporaneamente verificate può essere la seguente:

[1]In Java l'operatore booleano NOT viene codificato con !

[2]In Java l'operatore booleano AND viene codificato con &&

se un contribuente ha un reddito E se il comune di residenza applica l'addizionale comunale allora si deve pagare tale addizionale comunale.

Sintetizzato in tabella:

redditoPresente	previstaAddizionaleComunale	pagamentoAddizionale
FALSO	FALSO	FALSO
FALSO	VERO	FALSO
VERO	FALSO	FALSO
VERO	VERO	VERO

Possibile utilizzo

```
SE (redditoPresente=VERO)AND(previstaAddizionaleComunale=VERO)
  ALLORA
    "Calcola l'imposta comunale da pagare"
  ALTRIMENTI
    "Imposta non dovuta per mancanza di almeno un requisito"
FINESE
```

Esempio3 (**OR**)

L'operatore OR [3]restituisce un valore booleano VERO se almeno uno dei valori booleani in ingresso è vero secondo la seguente tabella:

primoValore	secondoValore	primoValore OR secondoValore
FALSO	FALSO	FALSO
FALSO	VERO	VERO
VERO	FALSO	VERO
VERO	VERO	VERO

Una situazione in cui dobbiamo avere almeno una condizione vera può essere la seguente:

se un commerciale vende molto riceve una gratifica tramite le provvigioni OPPURE se l'azienda decide di distribuire ai dipendenti in modo indiscriminato un premio di produzione, allora per il venditore ci saranno entrate extra

In sostanza il venditore beneficia di entrate aggiuntive se vende molto oppure se l'azienda distribuisce il premio di produzione in modo indiscriminato oppure si verificano entrambe le cose. Sintetizzato in tabella:

pagataGratifica	premioDiProduzioneAziendale	entrataMensileExtra
FALSO	FALSO	FALSO
FALSO	VERO	VERO
VERO	FALSO	VERO
VERO	VERO	VERO

Possibile utilizzo

```
SE (pagataGratifica=VERO)OR(premioDiProduzioneAziendale=VERO)
  ALLORA
    "Prevista entrata mensile extra"
  ALTRIMENTI
    "Nessuna entrata aggiuntiva prevista"
FINESE
```

Esempio4

Gli operatori logici possono essere combinati fra loro. Se l'azienda XY vuole espandersi E contemporaneamente ci sono mercati appetibili OPPURE, il venditore migliore è andato in pensione, allora bisogna potenziare la struttura commerciale. In questo caso devono essere verificare entrambe le condizioni di volontà di espansione e presenza di mercati appetibili per poter fornire parere positivo sul potenziamento della struttura commerciale, oppure questo potenziamento è legato ad un evento univoco come il pensionamento del venditore.

[3]In Java l'operatore booleano OR viene codificato con ||

espansione	mercatiDisponibili	pensioneVenditore	espansione && mercatiDisponibili	(espansione && mercatiDisponibili) || pensioneVenditore
FALSO	FALSO	FALSO	FALSO	FALSO
FALSO	FALSO	VERO	FALSO	VERO
FALSO	VERO	FALSO	FALSO	FALSO
FALSO	VERO	VERO	FALSO	VERO
VERO	FALSO	FALSO	FALSO	FALSO
VERO	FALSO	VERO	FALSO	VERO
VERO	VERO	FALSO	VERO	VERO
VERO	VERO	VERO	VERO	VERO

Possibile utilizzo

```
SE ((volontaEspansione=VERO) AND (mercatiAppetibili=VERO)) OR (venditorePensionato=VERO)
  ALLORA
    "Si deve potenziare la struttura commerciale"
  ALTRIMENTI
    "Nessuna variazione nell'azienda"
FINESE
```

Lavoro in aula / laboratorio

- Analizzare la seguente situazione e costruire la tabella di verità
  il bilancio aziendale migliora se aumentano i profitti OPPURE diminuiscono le perdite

- Analizzare la seguente situazione e costruire la tabella di verità e pseudocodifica
  un consumatore aumenta la quota di reddito destinata ai beni sussidiari se aumenta la sua capacità reddituale OPPURE diminuiscono i prezzi dei beni sussidiari OPPURE indipendentemente dalle vicende economiche un bene che prima era sussidiario nel paniere del consumatore diviene un bene primario

2.3 Metodi per l'Input/Output

All'interno del package unibs.eco.dmq.basicIO ci sono tre classi che agevolano la gestione delle operazioni relative all'interfaccia utente:

Classe Avvisi;
Classe Scrittore;
Classe Lettore.

Il dettaglio dei vari metodi associati alle classi può essere consultato attraverso l'opportuna documentazione creata con il javadoc. I metodi principali per effettuare delle operazioni di I/O sui dispositivi standard (tastiera, video) sono:

Input	
Metodo	Funzionalità
Lettore.tastiera.leggiDouble()	Permette la lettura di un tipo Double da tastiera
Lettore.tastiera.leggiInt()	Permette la lettura di un tipo Int da tastiera
Lettore.tastiera.leggiFloat()	Permette la lettura di un tipo Float da tastiera
Lettore.tastiera.leggiString()	Permette la lettura di un tipo String da tastiera
Lettore.tastiera.leggiRiga()	Permette la lettura di una riga intera da tastiera leggendo fino alla fine della riga, ignorando seperatori intermedi (es. spazi)

Output	
Metodo	Funzionalità
Scrittore.video.println([variabile])	Visualizza il contenuto tra parentesi, seguito da un ritorno a capo automatico
Scrittore.video.print([variabile])	Visualizza il contenuto tra parentesi senza ritorno a capo

(La variabile tra parentesi può essere anche una stringa statica es. "tasso crescita", in questo caso deve essere racchiusa fra doppio apice)

Esempio1
Si vogliono reperire dal dispositivo stadard di Input (tastiera) i dati relativi a nome e tasso di inflazione relativi a due nazioni, comunicando i dati appena ottenuti sul dispositivo standard di Output (video)

Listato Java (VisualizzaTassi.java)

```
01    import java.io.*;
02    import unibs.eco.dmq.basicIO.*;
03    public class VisualizzaTassi{
04      public static void main(String[] args){
05        double
06          inflazioneNazioneUno,
07          inflazioneNazioneDue;
08        String
09          nomeNazioneUno,
```

```
10          nomeNazioneDue;
11        /* Inizio */
12        Scrittore.video.println("[Inserire Nome Prima Nazione]:");
13        nomeNazioneUno=Lettore.tastiera.leggiString();
14        Scrittore.video.println("Inserire tasso di inflazione di "+nomeNazioneUno);
15        inflazioneNazioneUno=Lettore.tastiera.leggiDouble();
16        Scrittore.video.println("[Inserire Nome Seconda Nazione]:");
17        nomeNazioneDue=Lettore.tastiera.leggiString();
18        Scrittore.video.println("Inserire tasso di inflazione di "+nomeNazioneDue);
19        inflazioneNazioneDue=Lettore.tastiera.leggiDouble();
20        Scrittore.video.println("Nazione: "+nomeNazioneUno+" Tasso: "+inflazioneNazioneUno);
21        Scrittore.video.println("Nazione: "+nomeNazioneDue+" Tasso: "+inflazioneNazioneDue);
22      }
23    }
```

Commento al codice

Righe 01-02

Tramite le istruzioni di import il compilatore identifica e carica le classi necessarie all'interno del codice. Ogni parte del nome nel package identifica una cartella gerarchicamente più interna;

Riga 03

Viene definita la classe principale. Ogni programma Java è composto da almeno una definizione di classe definita dal programmatore. Il nome della classe è chiamato identificatore e per convenzione inizia con la lettera maiuscola. Un identificatore è costituito da una serie di caratteri (lettere, numeri, trattini di sottolineatura, simboli di dollaro) che non contengono spazi, il primo dei quali non deve essere un numero;

Riga 04

Questa riga fa parte di ogni applicazione Java. Le applicazioni Java iniziano la loro esecuzione con il metodo main. I metodi sono in grado di eseguire delle attività e di ritornare delle informazioni una volta completate queste attività. La parola chiave void indica che il metodo main svolgerà delle attività senza che venga restituita alcuna informazione;

Righe 05-10

Vengono definite le variabili. Le variabili sono locazioni di memoria identificate da un tipo e da un nome. Il tipo della variabile ne condiziona il comportamento e l'utilizzo all'interno del programma.Per convenzione la prima lettera di una variabile è minuscola e la scelta del nome deve essere autoesplicativa;

Riga 11

Il testo racchiuso fra /* e */ non viene considerato dal compilatore e viene utilizzato per introdurre commenti nel programma;

Riga 12

Viene chiamato il metodo *Scrittore.video.println* che permette di scrivere a video quanto viene passato come parametro. In questo caso viene passata una stringa fissa che viene racchiusa fra doppi apici;

Riga 13

Il medoto *Lettore.tastiera.leggiString* permette di reperire tramite la tastiera un contenuto di tipo alfanumerico che viene memorizzato nella variabile nomeNazioneUno;

Riga 14

Viene richiamato ancora il metodo Scrittore.video.println, in questo caso come parametro viene passata la concatenazione fra la scritta fissa *"Inserire tasso di inflazione di "* e il contenuto della variabile *nomeNazioneUno* (che viene automaticamente convertito in stringa);

Riga 15

Altro metodo per l'Input da tastiera che permette la lettura di un tipo double;

Righe 16-21

Metodi analizzati precedentemente;

Righe 22-23

Parentesi graffe di chiusura. Ogni blocco di codice contenente più di un istruzione deve essere racchiuso fra parentesi graffe.

Esempio 2

Codice Java relativo all' Esempio 2 - Operatori di confronto

Listato Java (RelazioneTassi.java)

```
import java.io.*;
import unibs.eco.dmq.basicIO.*;
public class RelazioneTassi{
  public static void main(String[] args) {
    /* Definisco una costante che contiene il tasso medio di mercato sui mutui */
    final double
      TASSO_MEDIO_MERCATO=4.15;
    double
      tassoProposto;
    /*Inizio*/
    Scrittore.video.println("[Inserire Tasso Proposto]:");
    tassoProposto=Lettore.tastiera.leggiDouble();
    if(TASSO_MEDIO_MERCATO>tassoProposto)
      Scrittore.video.println("Tasso proposto favorevole");
    else
      Scrittore.video.println("Tasso proposto non favorevole");
  }
}
```

(Attraverso la definizione final possiamo definire la variabile TASSO_MEDIO_MERCATO come una costante di programma. Il vantaggio di utilizzare le costanti di programma è legato alla possibilità di modificare il valore solamente nella fase iniziale dell'assegnazione statica, ottenendo la ripercussione automatica di tale modifica in tutto il programma)

Lavoro in aula / laboratorio

- Modificare il listato RelazioneTassi.java variando la costante TASSO_MEDIO_MERCATO (ricompilare e provare)
- Modificare il listato RelazioneTassi.java rispettando la logica e il dettato del problema utilizzando come operatore di confronto il <=

Esempio 3

Codice Java relativo all' Esempio 2 - Operatori logici

(Con il codice che segue viene rappresentata la tabella di tutte le casistiche possibili relative al pagamento dell'addizionale comunale)

Listato Java (PagaAddizionaleComunale.java)

```
import java.io.*;
import unibs.eco.dmq.basicIO.*;
public class PagaAddizionaleComunale{
  public static void main(String[] args) {
    boolean
      redditoPresente,
      previstaAddizionaleComunale,
      pagamentoAddizionale;
      /*Inizio*/
      redditoPresente=false;
      previstaAddizionaleComunale=false;
      pagamentoAddizionale=redditoPresente && previstaAddizionaleComunale;
      Scrittore.video.println("Reddito:"+redditoPresente+" Addizionale:"
      +previstaAddizionaleComunale+" Pagamento:"+pagamentoAddizionale);
```

```
        redditoPresente=false;
        previstaAddizionaleComunale=true;
        pagamentoAddizionale=redditoPresente && previstaAddizionaleComunale;
        Scrittore.video.println("Reddito:"+redditoPresente+" Addizionale:"
        +previstaAddizionaleComunale+" Pagamento:"+pagamentoAddizionale);
        redditoPresente=true;
        previstaAddizionaleComunale=false;
        pagamentoAddizionale=redditoPresente && previstaAddizionaleComunale;
        Scrittore.video.println("Reddito:"+redditoPresente+" Addizionale:"
        +previstaAddizionaleComunale+" Pagamento:"+pagamentoAddizionale);
        redditoPresente=true;
        previstaAddizionaleComunale=true;
        pagamentoAddizionale=redditoPresente && previstaAddizionaleComunale;
        Scrittore.video.println("Reddito:"+redditoPresente+" Addizionale:"
        +previstaAddizionaleComunale+" Pagamento:"+pagamentoAddizionale);
    }
}
```

```
Reddito:false Addizionale:false Pagamento:false
Reddito:false Addizionale:true Pagamento:false
Reddito:true Addizionale:false Pagamento:false
Reddito:true Addizionale:true Pagamento:true
Press any key to continue...
```

Lavoro in aula / laboratorio

- Realizzare il codice java per la problematica relativa all'espansione aziendale analizzata precedentemente (Esempio4 - Operatori logici)

2.4 Tipi primitivi

La scelta del tipo di variabile è molto importante nella fase di passaggio da pseudocodifica/flowchart a quella di traduzione nel linguaggio di programmazione. Durante la fase di analisi si sono presi in considerazione alcuni elementi che possono aiutare nella scelta quali ad esempio:

→ Tipo di valori che la variabile assume (Numerici, Alfanumerici, Singoli, Aggregazioni,...)

→ Operazioni associate (Algebriche, Confronto, Estrazione sottoinsiemi,....)

→ Range di valori (Dipendente dal valore massimo e minimo che la variabile può assumere)

Esempio 1
Dal problema analizzato è specificato che la variabile debba contenere numeri senza decimali, che le operazioni associate ad essa nell'algoritmo siano la differenza e la somma, che il valore massimo assunto può essere 5.000 e il minimo -6.000. Verifico se definendo la variabile di tip int vengono rispettate tutte le condizioni contemporaneamente (AND):

Numeri senza decimali	VERO
Differenza e somma sono operazioni compatibili con variabili intere	VERO
Range fra -6.000 e 5.000 supportato dalle variabili intere	VERO

Quindi posso definire in sede di linguaggio la variabile come tipo int.

Esempio 2
Dal problema analizzato è richiesto che la variabile contenga numeri o lettere e le operazioni ad essa associate siano la concatenazione e il confronto. Posso definire in sede di linguaggio la variabile come tipo String

Esempio 3
Che variabile utilizzo per contenere il risultato di una media aritmetica ??? Il risultato è frutto di un calcolo che prevede una divisione per cui se voglio essere ragionevolmente sicuro di non perdere precisione nel risultato utilizzo una variabile float o double

Esempio 4
Che variabile utilizzo per contenere lo stato di un contribuente che può essere persona fisica o persona giuridica ??? Se mi interessa soltanto lo stato e sono possibili solo due situazioni posso utilizzare una variabile booleana ad esempio personaFisica che sarà TRUE se il contribuente è una persona fisica e sarà FALSE se il contribuente è una persona giuridica (meglio NON è una persona fisica)

Tipo int (operazioni riconducibili anche ai tipi: long/short/byte**)**

Listato Java (TipoInt.java)

```
import java.io.*;
import unibs.eco.dmq.basicIO.*;
public class TipoInt{
      public static void main(String[] args) {
          int
          miaVariabile;
          /* Assegnazione statica */
            miaVariabile=2;
          /* Visualizzazione */
          Scrittore.video.println("mia variabile "+miaVariabile);
          /*Lettura da tastiera*/
          Scrittore.video.println("inserisci intero numerico");
          miaVariabile=Lettore.tastiera.leggiInt();
          Scrittore.video.println("mia variabile "+miaVariabile);
          /*Assegnazione da calcolo*/
```

```
          miaVariabile=2+5;
          Scrittore.video.println("risultato di 2+5 = "+miaVariabile);
          /* Incrementi Unitari*/
          miaVariabile=miaVariabile+1;
          miaVariabile++;
          miaVariabile+=1;
          Scrittore.video.println("mia variabile "+miaVariabile);
          /* Confronto */
          if (miaVariabile>=0)
            Scrittore.video.println("mia variabile positiva");
          else
            Scrittore.video.println("mia variabile negativa");
       }
}
```

Tipo float

Listato Java (TipoFloat.java)

```
import java.io.*;
import unibs.eco.dmq.basicIO.*;
public class TipoFloat{
      public static void main(String[] args) {
        float
          miaVariabile;
          /* Assegnazione statica */
          miaVariabile=4.25F;
          /* Visualizzazione */
          Scrittore.video.println("mia variabile "+miaVariabile);
          /*Lettura da tastiera*/
          Scrittore.video.println("inserisci numero");
          miaVariabile=Lettore.tastiera.leggiFloat();
          Scrittore.video.println("mia variabile "+miaVariabile);
          /*Assegnazione da calcolo*/
          miaVariabile=2.4F+2.5F;
          Scrittore.video.println("risultato di 2/5+1/5 = "+miaVariabile);
          /* Incrementi di tre elementi*/
          miaVariabile=miaVariabile+3;
          miaVariabile+=3;
          Scrittore.video.println("mia variabile "+miaVariabile);
          /* Confronto */
          if (miaVariabile>=0)
            Scrittore.video.println("mia variabile positiva");
          else
            Scrittore.video.println("mia variabile negativa");
       }
}
```

Tipo string

Listato Java (TipoString.java)

```
import java.io.*;
import unibs.eco.dmq.basicIO.*;
public class TipoString{
      public static void main(String[] args) {
          String
```

```
          miaVariabile;
        /* Assegnazione statica */
        miaVariabile="assegnazione";
        /* Visualizzazione */
        Scrittore.video.println("mia variabile : "+miaVariabile);
        /*Lettura da tastiera*/
        Scrittore.video.println("inserisci stringa");
        miaVariabile=Lettore.tastiera.leggiString();
        Scrittore.video.println("mia variabile : "+miaVariabile);
        /*Assegnazione da concatenazione*/
        miaVariabile="uno"+"due"+"tre";
        Scrittore.video.println("mia variabile : "+miaVariabile);
        /* Operazioni su stringhe */
        miaVariabile="12aAbBCCff12";
        Scrittore.video.println("Lunghezza variabile : "+miaVariabile.length());
        /* Trasformazioni maiuscolo-minuscolo */
        Scrittore.video.println("Tutto maiuscolo : "+miaVariabile.toUpperCase());
        /* Sottostringa */
        Scrittore.video.println("Tutto maiuscolo : "+miaVariabile.substring(5,8));
        /* Confronto*/
        if (miaVariabile.equals("fdfHTT"))
          Scrittore.video.println(miaVariabile+" = a "+"fdfHTT");
        else
          Scrittore.video.println(miaVariabile+" <> da "+"fdfHTT");
      }
}
```

Lavoro in aula / laboratorio

- Realizzare il codice Java per leggere nomi e quotazioni di due valori azionari e segnalare a video il nome e la quotazione del titolo con quotazione maggiore
- Realizzare il codice Java per leggere tre tassi di inflazione e visualizzare il tasso di inflazione medio

2.5 Proposte di lavoro

Proposta Uno

Realizzare come pseudocodifica (ed eventualmente codice Java) la tabella di valori booleani generata dalla seguente problematica: se si verifica un rincaro nel prezzo del petrolio OPPURE si verifica un aumento dei tassi d'interesse, allora è ipotizzabile che il tasso d'inflazione cresca;

Proposta Due

Utilizzando le classi di Input/Output apposite, inserire il PIL relativo a tre nazioni ed elencare quali fra queste nazioni hanno un PIL superiore al PIL medio delle tre;

Proposta Tre

Supponiamo di scrivere un algoritmo per il controllo di un codice numerico intero che abbia le seguenti regole: il codice è valido se è un numero pari OPPURE divisibile per 7, inoltre il numero deve essere positivo E minore di 100.

(La verifica sulla divisibilità può essere fatta usando l'operatore % che restituisce il resto della divisione fra numeri interi. Es. il controllo sui numeri pari, può essere fatto "testando" lo stato della variabile booleana divisibileX2 dopo la sequente assegnazione divisibileX2=(variabile%2)==0, che riporterà VERO nel caso in cui sia verificata l'uguaglianza del resto della divisione fra la variabile e il valore 2 con il valore 0);

Capitolo 3

Argomenti trattati

- Istruzione If
- Istruzione Switch
- Istruzione While
- Istruzione Do...While
- Istruzione For
- Proposte di lavoro

3.1 Istruzione IF

Un esempio di algortimo che utilizza if annidati può essere quello analizzato nella scheda 01 relativo alla determinazione della tassazione delle persone fisiche.

Listato Java (CalcolaImposta.java)

```
import java.io.*;
import unibs.eco.dmq.basicIO.*;
public class CalcolaImposta{
  public static void main(String[] args) {
  /* Costanti limite scaglione */
  final double
    LIMITE_SCAGLIONE_UNO=15000.00,
    LIMITE_SCAGLIONE_DUE=29000.00,
    LIMITE_SCAGLIONE_TRE=32600.00,
    LIMITE_SCAGLIONE_QUATTRO=70000.00;
  /* Costanti cumuli scaglione */
  final double
    CUMULO_SCAGLIONE_UNO=3450.00,
    CUMULO_SCAGLIONE_DUE=7510.00,
    CUMULO_SCAGLIONE_TRE=8626.00,
    CUMULO_SCAGLIONE_QUATTRO=23212.00;
  /* Costanti aliquote */
  final int
    ALIQUOTA_SCAGLIONE_UNO=23,
    ALIQUOTA_SCAGLIONE_DUE=29,
    ALIQUOTA_SCAGLIONE_TRE=31,
    ALIQUOTA_SCAGLIONE_QUATTRO=39,
    ALIQUOTA_SCAGLIONE_CINQUE=45;
  double
    redditoComplessivo,
    oneriDeducibili,
    deduzioniNoTaxArea,
    detrazioneLavoro,
    detrazioneFamigliari,
    impostaLorda,
    impostaDovuta,
    redditoImponibile;
  /* Inizio */
    impostaLorda=0.0;
    impostaDovuta=0.0;
    redditoImponibile=0.0;
    Scrittore.video.println("[Inserire reddito complessivo]:");
    redditoComplessivo=Lettore.tastiera.leggiDouble();
    Scrittore.video.println("[Inserire oneri deducibili]:");
    oneriDeducibili=Lettore.tastiera.leggiDouble();
    Scrittore.video.println("[Inserire deduzioni No Tax Area]:");
    deduzioniNoTaxArea=Lettore.tastiera.leggiDouble();
    Scrittore.video.println("[Inserire detrazione lavoro]:");
    detrazioneLavoro=Lettore.tastiera.leggiDouble();
    Scrittore.video.println("[Inserire detrazione famigliari]:");
    detrazioneFamigliari=Lettore.tastiera.leggiDouble();
    redditoImponibile=redditoComplessivo-oneriDeducibili-deduzioniNoTaxArea;
    if (redditoImponibile<=LIMITE_SCAGLIONE_UNO)
      impostaLorda=redditoImponibile*(ALIQUOTA_SCAGLIONE_UNO/100.00);
    else if (redditoImponibile<=LIMITE_SCAGLIONE_DUE){
```

```
      impostaLorda=CUMULO_SCAGLIONE_UNO;
      impostaLorda=impostaLorda+
        (redditoImponibile-LIMITE_SCAGLIONE_UNO)*(ALIQUOTA_SCAGLIONE_DUE/100.00);
    }
    else if(redditoImponibile<=LIMITE_SCAGLIONE_TRE){
      impostaLorda=CUMULO_SCAGLIONE_DUE;
      impostaLorda=impostaLorda+
        (redditoImponibile-LIMITE_SCAGLIONE_DUE)*(ALIQUOTA_SCAGLIONE_TRE/100.00);
    }
    else if(redditoImponibile<=LIMITE_SCAGLIONE_QUATTRO){
      impostaLorda=CUMULO_SCAGLIONE_TRE;
      impostaLorda=impostaLorda+
        (redditoImponibile-LIMITE_SCAGLIONE_TRE)*(ALIQUOTA_SCAGLIONE_QUATTRO/100.00);
    }
    else{
      impostaLorda=CUMULO_SCAGLIONE_QUATTRO;
      impostaLorda=impostaLorda+
        (redditoImponibile-LIMITE_SCAGLIONE_QUATTRO)*(ALIQUOTA_SCAGLIONE_CINQUE/100.00);
    }
    impostaDovuta=impostaLorda-detrazioneLavoro-detrazioneFamigliari;
    Scrittore.video.println("[Imposta dovuta]:"+impostaDovuta);
  }
}
```

```
[Inserire reddito complessivo]:
28000
[Inserire oneri deducibili]:
0
[Inserire deduzioni No Tax Area]:
1586.25
[Inserire detrazione lavoro]:
130
[Inserire detrazione famigliari]:
1013.06
[Imposta dovuta]:5616.9275
Press any key to continue...
```

```
[Inserire reddito complessivo]:
20000
[Inserire oneri deducibili]:
1500
[Inserire deduzioni No Tax Area]:
4326.75
[Inserire detrazione lavoro]:
0
[Inserire detrazione famigliari]:
0
[Imposta dovuta]:3259.8475000000003
Press any key to continue...
```

Note al listato

L'utilizzo delle costanti (variabili dichiarate final) permette una forte parametrizzazione da parte del programmatore. Supponiamo che venga decisa una riduzione dell'aliquota relativa allo scaglione uno dal 23% al 21%, il programmatore dovrà cambiare semplicemente la variabile final da 23 a 21 lasciando inalterato il resto del programma.

Lavoro in aula / laboratorio

- Realizzare il codice Java relativo all'algoritmo di calcolo della "No tax area" analizzato nella scheda 01

3.2 Istruzione SWITCH

L'utilizzo dello switch si presta alla realizzazione di strutture a menu o nelle catene di if esclusivi ogni qualvolta il test preveda la valutazione di uguaglianza su un valore di tipo enumerativo.

Esempio di utilizzo dell'istruzione switch :

i movimenti di chiusura annuale (attribuzione spese, calcolo interessi, ...) relativi ad un conto corrente bancario sono strettamente legati alle condizioni associate al conto. Supponiamo di avere le seguenti tipologie di conto(le combinazioni fra interessi attivi, spese per operazioni e spese fisse sono puramente esemplificative):

Conto 1

Caratteristiche: paga interessi attivi, non sono previste operazioni sul conto e non sono previste spese di chiusura (può essere pensato come un conto a giacenza temporalmente vincolato).

Conto 2

Caratteristiche: paga interessi attivi, prevede un numero massimo di operazioni gratuite per poi pagare un fisso per operazione, prevede spese fisse di chiusura.(può essere pensato come il conto corrente standard).

Conto 3

Caratteristiche: non paga interessi attivi, prevede un numero massimo di operazioni gratuite per poi pagare un fisso per operazione, non prevede spese fisse di chiusura. (può essere pensato come un conto di servizio a giacenza medio bassa).

Listato Java (ChiusuraAnnuale.java)

```
import java.io.*;
import unibs.eco.dmq.basicIO.*;
public class ChiusuraAnnuale{
  public static void main(String[] args) {
    final int
      SPESE_FISSE_ANNUALI=35;
    final int
      MAX_OPERAZIONI=10;
    final double
      INTERESSE_ATTIVO=1.25;
    final double
      COSTO_OPERAZIONE=1.5;
    int
      tipoConto,
      operazioniEffettuate;
    double
      saldoConto,
      parzialeSpese;
    /*Inizio*/
      Scrittore.video.println("Saldo C/C:");
      saldoConto=Lettore.tastiera.leggiDouble();
      Scrittore.video.println("Tipo di conto:");
      Scrittore.video.println("1 - 2 - 3");
      tipoConto=Lettore.tastiera.leggiInt();
      Scrittore.video.println("Numero operazioni effettuate");
      operazioniEffettuate=Lettore.tastiera.leggiInt();
      switch(tipoConto){
      /* Conto 1:*/
        case 1:
          saldoConto=saldoConto*(1.0+(INTERESSE_ATTIVO)/100.0);
          break;
      /* Conto 2:*/
        case 2:
          saldoConto=saldoConto*(1.0+(INTERESSE_ATTIVO)/100.0);
```

```
          saldoConto=saldoConto-SPESE_FISSE_ANNUALI;
          if (operazioniEffettuate>MAX_OPERAZIONI){
            parzialeSpese=(operazioniEffettuate-MAX_OPERAZIONI)*COSTO_OPERAZIONE;
            saldoConto=saldoConto-parzialeSpese;
          }
          break;
      /* Conto 3:*/
        case 3:
          if (operazioniEffettuate>MAX_OPERAZIONI){
            parzialeSpese=(operazioniEffettuate-MAX_OPERAZIONI)*COSTO_OPERAZIONE;
            saldoConto=saldoConto-parzialeSpese;
          }
          break;
      }
      if (saldoConto>=0)
        Scrittore.video.println("Saldo positivo pari a : "+saldoConto);
      else
        Scrittore.video.println("Saldo negativo pari a :"+saldoConto);
  }
}
```

```
Saldo C/C:
50000
Tipo di conto:
1 - 2 - 3
1
Numero operazioni effettuate
10
Saldo positivo pari a : 50625.0
Press any key to continue...
```

```
Saldo C/C:
20000
Tipo di conto:
1 - 2 - 3
2
Numero operazioni effettuate
20
Saldo positivo pari a : 20200.0
Press any key to continue...
```

```
Saldo C/C:
40000
Tipo di conto:
1 - 2 - 3
3
Numero operazioni effettuate
30
Saldo positivo pari a : 39970.0
Press any key to continue...
```

Lavoro in aula / laboratorio

- Introdurre nel listato precedente il tipo conto 4 con le seguenti caratteristiche:

  paga interessi attivi solo se viene superata una certa giacenza (definita come costante)

  il costo delle operazioni è costante fino ad un numero prefissato (definiti come costanti) superato il quale tutte le operazioni sono gratuite

3.3 Istruzione WHILE

L'istruzione while permette l'iterazione di una o più istruzioni fino al verificarsi di una condizione di uscita. Il numero di iterazioni potrebbe anche essere uguale a 0 nel caso in cui la condizione d'uscita risulti subito verificata. E' essenziale che una volta entrati in un ciclo iterativo siano presenti delle istruzioni che modificano le variabili oggetto della condizione d'uscita o delle istruzioni di breaking del ciclo per evitare di finire in loop (ciclo senza termine).

Esempio di ciclo iterativo:

La concezione della probabilità classica definita da Laplace è:
"si definisce probabilità di un evento il rapporto tra il numero m dei casi favorevoli all'evento e il numero n dei casi possibili, purchè i casi siano tutti equiprobabili: $p(E)=\frac{m}{n}$"

La definizione frequentista invece afferma che:
"sia n il numero (molto grande) di volte che si è ripetuto un dato esperimento in condizioni identiche; sia m il numero di volte che si è presentato un dato evento, allora la probabilità del verificarsi è $p=\frac{m}{n}$"
In questa fase non verranno approfondite le altre definizioni di probabilità (soggettivistica, assiomatica), si vuole semplicemente definire un algoritmo che permetta di rispondere alla domanda: qual è la probabilità che lanciando una moneta esca testa?
In base alla definizione classica si porebbe affermare $\frac{1}{2}$ ma, il nostro scopo è realizzare una simulazione che permetta attraverso la definizione frequentista, di confermare la definizione classica. Tramite il linguaggio Java si vuole simulare il lancio di una moneta un numero arbitrario di volte per verificare che aumentando il numero delle prove, la probabilità che esca testa o croce si avvicina a $\frac{1}{2}$

Listato Java (ProbaFreq.java)

```
import java.io.*;
import unibs.eco.dmq.basicIO.*;
public class ProbaFreq{
 public static void main(String[] args) {
   int
     numeroTesta,
     numeroCroce,
     numeroProve,
     contaProve;
   double
     estratto;
   /*Inizio*/
     numeroTesta=0;
     numeroCroce=0;
     contaProve=0;
     Scrittore.video.println("Numero di lanci da effettuare :");
     numeroProve=Lettore.tastiera.leggiInt();
     while(contaProve<numeroProve){
       estratto=Math.random();
       if (estratto<0.5){
         numeroTesta++;
         Scrittore.video.println("Testa");
       }
       else{
         numeroCroce++;
         Scrittore.video.println("Croce");
       }
       contaProve++;
     }
     Scrittore.video.println("Tot testa "+numeroTesta+ " Frequenza "
       +(float)numeroTesta/numeroProve);
```

```
      Scrittore.video.println("Tot croce "+numeroCroce+ " Frequenza "
        +(float)numeroCroce/numeroProve);
  }
}
```

```
Numero di lanci da effettuare :
15
Testa
Croce
Croce
Croce
Croce
Testa
Croce
Croce
Testa
Croce
Testa
Croce
Testa
Croce
Testa
Tot testa 6 Frequenza 0.4
Tot croce 9 Frequenza 0.6
Press any key to continue...
```

Ulteriore esempio di utilizzo del ciclo while può essere la scansione di una serie di dati memorizzati all'interno di un file. Nell'esempio che segue sono memorizzate all'interno di un semplice file di testo delle informazioni relative ad un gruppo di nazioni per quanto rigurda il rispetto dei parametri UE.

Il trattato di Maastricht ha previsto il rispetto di determinati parametri fra cui:

- Inflazione non superiore di 1,5 punti alla media dei tre paesi con minor tasso di incremento dei prezzi;
- Deficit pubblico non superiore al 3% del Pil;
- Debito pubblico non superiore al 60% del Pil;
- Stabilità del cambio all'interno delle fasce di oscillazione rispetto all'ECU;

Supponiamo che i dati di ogni nazione vengano memorizzati all'interno di un file di testo in questa sequenza:

Nome nazione PIL Ammontare deficit Ammontare debito Tasso inflazione

Es. di file .txt (I valori inseriti sono inventati, il file può essere creato semplicemente con il notepad e salvato in formato txt)

Italia	1000	16	570	2.5
Francia	1250	78	720	1.6
Germania	1500	20	810	1.5
Spagna	900	10	510	2.0
Portogallo	800	70	619	3.2
Olanda	1100	45	590	1.5

Si vuole realizzare tramite linguaggio Java un programma che, letti i dati dei paesi dal file, determini il rispetto del Deficit pubblico e/o del Debito pubblico dei parametri europei e determini l'inflazione media dell'area UE considerata.

Listato Java (ParametriUE.java)

```
import java.io.*;
import unibs.eco.dmq.basicIO.*;
public class ParametriUE{
  public static void main(String[] args) {
    int
      numeroNazioni,
      numeroNazioniElaborate,
      pilNazione,
      deficitNazione,
```

```
      debitoNazione;
    double
      inflazioneNazione,
      mediaInflazioni;
    String
      nomeNazione;
    boolean
      okDeficit,
      okDebito;
    /*Inizio*/
    Lettore fileNazioni=new Lettore("parametri.txt");
    mediaInflazioni=0;
    numeroNazioni=fileNazioni.contaRighe();
    numeroNazioniElaborate=0;
    Scrittore.video.println("Nazione Pil Deficit Debito Inflazione");
    while(numeroNazioniElaborate<numeroNazioni){
      nomeNazione=fileNazioni.leggiString();
      pilNazione=fileNazioni.leggiInt();
      deficitNazione=fileNazioni.leggiInt();
      debitoNazione=fileNazioni.leggiInt();
      inflazioneNazione=fileNazioni.leggiDouble();
      Scrittore.video.println(nomeNazione+" "+pilNazione+" "
        +deficitNazione+" "+debitoNazione+" "+inflazioneNazione);
      okDeficit=(deficitNazione<(pilNazione*0.03));
      okDebito=(debitoNazione<(pilNazione*0.6));
      Scrittore.video.println("Rispetto Deficit = "+okDeficit+" Rispetto Debito = "+okDebito);
      Scrittore.video.println("-------------------------------------------------------");
      mediaInflazioni=mediaInflazioni+inflazioneNazione;
      numeroNazioniElaborate++;
    }
   fileNazioni.chiudi();
   Scrittore.video.println("Inflazione Media "+(float)mediaInflazioni/numeroNazioniElaborate);
  }
}
```

```
Nazione   Pil   Deficit   Debito   Inflazione
Italia     1000     16    570    2.5
Rispetto Deficit = true Rispetto Debito = true
-------------------------------------------------
Francia     1250     78    720    1.6
Rispetto Deficit = false Rispetto Debito = true
-------------------------------------------------
Germania     1500     20    810    1.5
Rispetto Deficit = true Rispetto Debito = true
-------------------------------------------------
Spagna     900     10    510    2.0
Rispetto Deficit = true Rispetto Debito = true
-------------------------------------------------
Portogallo     800     70    619    3.2
Rispetto Deficit = false Rispetto Debito = false
-------------------------------------------------
Olanda     1100     45    590    1.5
Rispetto Deficit = false Rispetto Debito = true
-------------------------------------------------
Inflazione Media 2.05
Press any key to continue...
```

Note al listato

Vengono utilizzate due variabili booleane per verificare il rispetto dei parametri. Ad esempio l'istruzione *okDebito=(debitoNazione< (pilNazione*0.6))* verifica se il debito della nazione è inferiore al 60% del pil nazionale e memorizza la risposta (true o false) nell'apposita variabile booleana che assume quindi il significato logico di rispetto parametro = true o rispetto parametro = false.

Lavoro in aula / laboratorio

- Aggiungere al listato Java precedente le istruzioni per determinare:

  1. nome e valore inflazione della nazione con inflazione maggiore (suggerimento: per ogni riga di file letta verificare se l'inflazione letta è la maggiore fra quelle considerate; se questo è verificato, allora questa inflazione viene memorizzata come inflazione massima e il nome relativo viene memorizzato in una variabile ausiliaria)

  2. elenco nazioni con inflazione superiore all'inflazione media UE

3.4 Istruzione DO....WHILE

Un utilizzo tipico dell'istruzione do...while è l'applicazione nell'inserimento controllato. Tramite questo costrutto è possibile far ripetere delle istruzioni (specificatamente quelle di inserimento) fino a quando le variabili di riferimento assumono valori accettabili. La prevenzione degli effetti che può provocare l'inserimento di valori errari è un aspetto molto importante che deve essere valutato dal programmatore. (Ad esempio il controllo che valori numerici non possano essere al di fuori di un certo range evita crash del programma o risultati inaspettati).

Supponiamo che il codice di controllo di un conto bancario abbia le seguenti regole:

il conto è valido se:

la parte numerica è un numero pari, positivo e inferiore a 1000

la parte letterale è un vocale maiuscola

Listato Java (CodiceControllo.java)

```
import java.io.*;
import unibs.eco.dmq.basicIO.*;
public class CodiceControllo{
  public static void main(String[] args) {
    String
      codiceParteAlfanumerica;
    int
      codiceParteNumerica;
    boolean
      okControllo;
    /*Inizio*/
    do{
      Scrittore.video.println("Inserisci Parte Numerica Codice :");
      codiceParteNumerica=Lettore.tastiera.leggiInt();
      okControllo=(codiceParteNumerica>0)&&
                  (codiceParteNumerica<=1000)&&(codiceParteNumerica%2==0);
      if (okControllo!=true)
        Scrittore.video.println("Parte numerica del codice non corretta");
    }while(okControllo==false);
    do{
      Scrittore.video.println("Inserisci Parte Alfanumerica Codice :");
      codiceParteAlfanumerica=Lettore.tastiera.leggiString();
     /* trasformo in maiuscolo per dimezzare i controlli */
      codiceParteAlfanumerica=codiceParteAlfanumerica.toUpperCase();
      okControllo=
        (codiceParteAlfanumerica.equals("A"))||
        (codiceParteAlfanumerica.equals("E"))||
        (codiceParteAlfanumerica.equals("I"))||
        (codiceParteAlfanumerica.equals("O"))||
        (codiceParteAlfanumerica.equals("U"));
      if (okControllo!=true)
      Scrittore.video.println("Parte Alfanumerica del codice non corretta");
    }while(okControllo==false);
    Scrittore.video.println("Codice [ "+codiceParteAlfanumerica+" "
      +codiceParteNumerica+" ] ABILITATO");
  }
}
```

Lavoro in aula / laboratorio

- Realizzare un listato Java che legge due quotazioni con i seguenti controlli:

  la quotazione è valida se compresa tra un valoreMax e un valoreMin
  (definiti come costanti final dal programmatore)

  le quotazioni sono valide se la loro media non è superiore alla media
  fra valoreMax e valoreMin
  (suggerimento: usare un ciclo do ... while esterno per controllare
  il rispetto della media e due cicli do ... while interni per
  controllare il rispetto delle singole quotazioni)

3.5 Istruzione FOR

L'istruzione for è spesso legata a tutte quelle elaborazioni che prevedono una condizione d'uscita e un incremento fisso (per questo il costrutto è spesso associato all'elaborazione dei vettori) L'utilizzo del costrutto for prevede anche cicli a più condizioni d'uscita e incrementi multipli di variabili.

Supponiamo di volere simulare l'andamento delle quotazioni di un'azione, attraverso l'estrazione di valori casuali compresi fra un valore minimo del periodo e un valore massimo del periodo

Listato Java (EstraiQuotazioni.java)

```
import java.io.*;
import unibs.eco.dmq.basicIO.*;
public class EstraiQuotazioni{
      public static void main(String[] args) {
        int
          quanteQuotazioni,
          contaQuotazioni;
        double
          minAnnuale,
          maxAnnuale,
          maxQuotazioni,
          minQuotazioni,
          sommaQuotazioni,
          quotazioneEstratta,
          mediaQuotazioni;
        String
          nomeAzione;
          sommaQuotazioni=0;
          minQuotazioni=0;
          maxQuotazioni=0;
          sommaQuotazioni=0;
          /*Inizio*/
          Scrittore.video.println("Inserisci nome dell'Azione :");
          nomeAzione=Lettore.tastiera.leggiString();
          do{
            Scrittore.video.println("Numero di estrazioni da effettuare :");
            quanteQuotazioni=Lettore.tastiera.leggiInt();
          }while(quanteQuotazioni<=0);
          do{
            Scrittore.video.println("Quotazione max annuale :");
            maxAnnuale=Lettore.tastiera.leggiDouble();
            Scrittore.video.println("Quotazione min annuale :");
            minAnnuale=Lettore.tastiera.leggiDouble();
          }while(minAnnuale>maxAnnuale);
          minQuotazioni=maxAnnuale;
          for(contaQuotazioni=0;contaQuotazioni<=quanteQuotazioni;contaQuotazioni++){
            do{
              quotazioneEstratta=Math.random()*maxAnnuale;
            }while(quotazioneEstratta<minAnnuale);
            Scrittore.video.println("Quotazione "+quotazioneEstratta);
            sommaQuotazioni=sommaQuotazioni+quotazioneEstratta;
            if (quotazioneEstratta<minQuotazioni) minQuotazioni=quotazioneEstratta;
            if (quotazioneEstratta>maxQuotazioni) maxQuotazioni=quotazioneEstratta;
          }
          mediaQuotazioni=sommaQuotazioni/(float)quanteQuotazioni;
          Scrittore.video.println("--------------------------");
```

```
            Scrittore.video.println("AZIONE : "+nomeAzione);
            Scrittore.video.println("Media quotazioni "+mediaQuotazioni);
            Scrittore.video.println("Min quotazioni "+minQuotazioni);
            Scrittore.video.println("Max quotazioni "+maxQuotazioni);
        }
}
```

```
Inserisci nome dell'Azione :
ENI
Numero di estrazioni da effettuare :
12
Quotazione max annuale :
16
Quotazione min annuale :
14
Quotazione 14.084541796105915
Quotazione 14.961568923719293
Quotazione 14.389093514923148
Quotazione 14.062338303737832
Quotazione 15.725001254981033
Quotazione 15.931711904531412
Quotazione 15.25524103901039
Quotazione 15.557881747075953
Quotazione 14.491882281011991
Quotazione 15.382797359025492
Quotazione 14.75528806634362
Quotazione 15.832289959050858
Quotazione 14.051541275854948
--------------------------
AZIONE : ENI
Media quotazioni 16.206764785447657
Min quotazioni 14.051541275854948
Max quotazioni 15.931711904531412
Press any key to continue...
```

Lavoro in aula / laboratorio

- Realizzare un listato Java che definito un tasso di inflazione come riferimento (tramite dichiarazione final) chieda all'utente l'inserimento di un certo numero di tassi di inflazione (tramite ciclo for) e visualizzi lo scostamento in valore assoluto dei singoli tassi di inflazione con il tasso di riferimento.

3.6 Proposte di lavoro

Proposta Uno

Memorizzare in un file di testo i dati relativi ad un conto corrente bancario in questo modo:

Causale Importo
La causale può essere:
VE - Versamento;
PR - Prelievo.

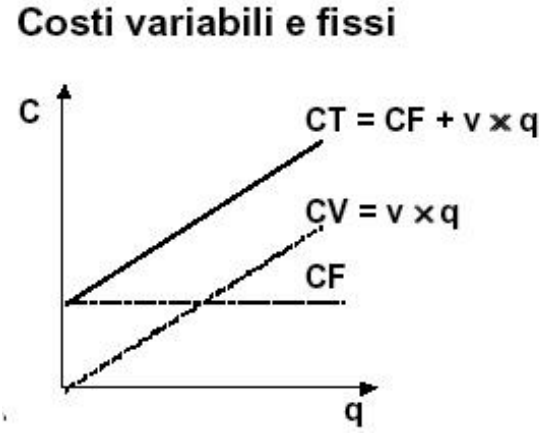
Esempio di file
VE 1.000
PR 1.100
PR 2.000
VE 1.500

Realizzare un algoritmo (e tradurlo in linguaggio Java) che permetta di realizzare le seguenti opzioni:

1 Determinare il saldo del conto(somma versamenti - somma prelievi); (scansione del file)
2 Determinare l'importo medio versato e quelle prelevato; (scansione del file)
3 Determinare l'importo massimo e la sua tipologia (versamento o prelievo).(scansione del file)

Proposta Due

Considerando una funzione dei costi aziendali di questo tipo:
CT=CF+CV
dove CF rappresenta l'insieme dei costi fissi (quindi sostenuti dall'azienda indipendentemente dal volume della produzione) e CF rappresenta l'insieme dei costi variabili (legati al volume della produzione)



realizzare un algoritmo che chieda in ingresso alcuni costi fissi (es. affitto strutture), alcuni costi variabili (es.energia elettrica) e permetta di simulare l'andamento di CT variando il volume di produzione

Capitolo 4

Argomenti trattati

- Stuttura a menu con Do..While e Switch
- Array: inserimento,visualizzazione,elaborazione
- Array: ricerca semplice, determinazione del massimo, elaborazione
- Array: booleani, relazione posizionali multiple
- Proposte di lavoro

4.1 Struttura a menu con DO...WHILE e SWITCH

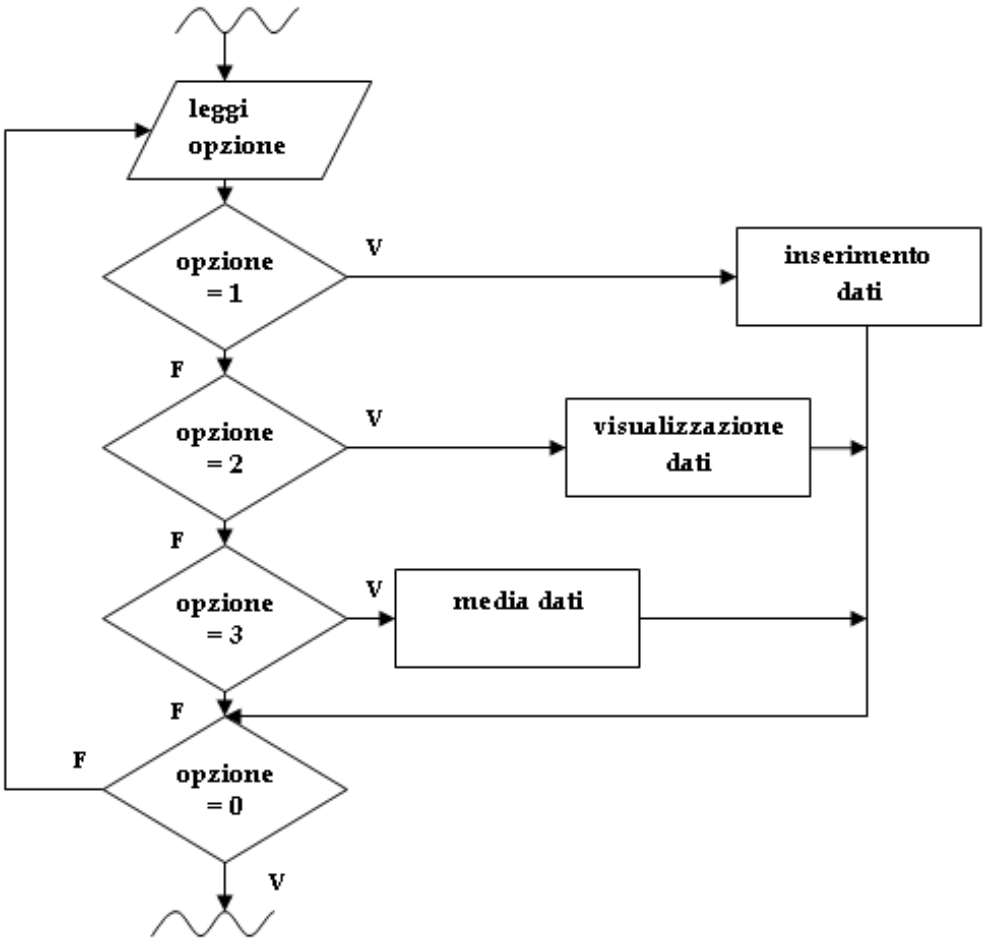
Una classica struttura implementata utilizzando il do..while e lo switch è quella a menu. La struttura ripete la sequenza:

1 esegui le seguenti istruzioni
2 visualizza le opzioni possibili dell'algoritmo
3 scegli l'opzione desiderata
4 esegui le istruzioni associate all'istruzione desiderata
5 se non è stata scelta l'opzione di uscita dal programma riprendi dal punto 1

Tramite questo tipo di struttura è possibile analizzare un algoritmo che prevede varie fasi e/o opzioni concentrandosi su una di queste per volta, provarne il funzionamento e passare alla fase / opzione successiva.

La maggior parte delle operazioni sono inefficaci se non è stata effettuata la fase di inserimento. Il controllo dell'inserimento può essere realizzato tramite un flag booleano inizializzato a falso ad inizio programma e settato a vero una volta che viene fatto l'inserimento. Il seguente esempio introduce una struttura a menu e permette di effettuare le operazioni di inserimento dati, visualizzazione e media secondo lo schema che segue:

ripeti
visualizza opzioni
scegli opzione
in base all' opzione scelta esegui:
inserimento dati;
oppure visualizzazione dati;
oppure media dati.
fino a quanto l'utente decide di terminare l'esecuzione attraverso la scelta dell'operazione 0 (uscita)



Listato Java (MenuGestione.java)

```
import java.io.*;
import unibs.eco.dmq.basicIO.*;
public class MenuGestione{
  public static void main(String[] args) {
    int
      opzione;
    double
      inflazioneNazioneUno,
      inflazioneNazioneDue,
      inflazioneMedia;
    String
      nomeNazioneUno,
      nomeNazioneDue;
    boolean
      insOK;
    /*Inizio*/
    inflazioneNazioneUno=0;
    inflazioneNazioneDue=0;
    nomeNazioneUno="";
    nomeNazioneDue="";
    insOK=false;
    inflazioneMedia=0;
    do{
      Scrittore.video.println("---------------------");
      Scrittore.video.println("1* Inserimento dati");
      Scrittore.video.println("2* Visualizzazione dati");
      Scrittore.video.println("3* Inflazione media");
      Scrittore.video.println("0* USCITA");
      Scrittore.video.println("---------------------");
      opzione=Lettore.tastiera.leggiInt();
      switch(opzione){
        case 1:
          Scrittore.video.println("Inserisci nome prima nazione");
          nomeNazioneUno=Lettore.tastiera.leggiString();
          Scrittore.video.println("Inserisci inflazione prima nazione");
          inflazioneNazioneUno=Lettore.tastiera.leggiDouble();
          Scrittore.video.println("Inserisci nome seconda nazione");
          nomeNazioneDue=Lettore.tastiera.leggiString();
          Scrittore.video.println("Inserisci inflazione seconda nazione");
          inflazioneNazioneDue=Lettore.tastiera.leggiDouble();
          insOK=true;
          break;
        case 2:
          if(insOK==true){
            Scrittore.video.println("Nazione Inflazione");
            Scrittore.video.println("---------------------");
            Scrittore.video.println(nomeNazioneUno+" "+inflazioneNazioneUno);
            Scrittore.video.println(nomeNazioneDue+" "+inflazioneNazioneDue);
          }
          else
            Scrittore.video.println("Inserimento non effettuato");
          break;
        case 3:
          if(insOK==true){
            inflazioneMedia=(inflazioneNazioneUno+inflazioneNazioneDue)/2.0;
```

```
            Scrittore.video.println("Inflazione Media: "+inflazioneMedia);
          }
          else
            Scrittore.video.println("Inserimento non effettuato");
          break;
      }
    }while(opzione!=0);
  }
}
```

```
---------------------
1* Inserimento dati
2* Visualizzazione dati
3* Inflazione media
0* USCITA
---------------------
1
Inserisci nome prima nazione
ITALIA
Inserisci inflazione prima nazione
2.5
Inserisci nome seconda nazione
FRANCIA
Inserisci inflazione seconda nazione
2.2
---------------------
1* Inserimento dati
2* Visualizzazione dati
3* Inflazione media
0* USCITA
---------------------
2
Nazione    Inflazione
---------------------
ITALIA  2.5
FRANCIA  2.2
---------------------
1* Inserimento dati
2* Visualizzazione dati
3* Inflazione media
0* USCITA
---------------------
3
Inflazione Media: 2.35
---------------------
1* Inserimento dati
2* Visualizzazione dati
3* Inflazione media
0* USCITA
---------------------
```

4.2 Array: inserimento,visualizzazione,elaborazione

Si vuole gestire tramite array un insieme di prodotti. Vengono utilizzati due array, uno che memorizza i nomi dei prodotti e l'altro i costi dei prodotti. La relazione fra i due array è di tipo posizionale, cioè fissata una posizione (indice) troviamo il nome del prodotto nell'array dei nomi e il costo corrispondente nell'array dei costi.

Indice	Array nome prodotto		Array costo prodotto
0	Ferro	⇆	100
1	Rame	⇆	210
2	Acciaio	⇆	150
3	Ghisa	⇆	180

La gestione prevede una fase di inserimento, una fase di visualizzazione, il costo medio delle materie prime, determinazione del costo di un prodotto assemblato composto da una certa quantità di due fra le materie prime disponibili.

Listato Java (GestioneArray.java)

```
import java.io.*;
import unibs.eco.dmq.basicIO.*;
public class GestioneArray{
      public static void main(String[] args) {
         final int
          DIM_ARRAY=3;
         double []
           costiMaterie;
         double
           sommaCosti,
           mediaCosti,
           qtaPrimaMateria,
           qtaSecondaMateria,
           costoProdotto;
         String[]
           nomiMaterie;
         boolean
           insOK;
         int
           i,
           opzione,
           primaMateria,
           secondaMateria;
         /*Inizio*/
         insOK=false;
         mediaCosti=0;
         costoProdotto=0;
         qtaPrimaMateria=0;
         qtaSecondaMateria=0;
         primaMateria=0;
         secondaMateria=0;
         costiMaterie=new double[DIM_ARRAY];
         nomiMaterie=new String[DIM_ARRAY];
         do{
           Scrittore.video.println("---------------------");
           Scrittore.video.println("1* Inserimento dati");
           Scrittore.video.println("2* Visualizzazione dati");
```

```
Scrittore.video.println("3* Costo  medio materie prime ");
Scrittore.video.println("4* Costo  prodotto ");
Scrittore.video.println("0* USCITA");
Scrittore.video.println("----------------------");
opzione=Lettore.tastiera.leggiInt();
  switch(opzione){
    case 1:
      for(i=0;i<DIM_ARRAY;i++){
        Scrittore.video.println("Inserire nome prodotto di indice "+i);
        nomiMaterie[i]=Lettore.tastiera.leggiString();
        Scrittore.video.println("Inserire prezzo di "+nomiMaterie[i]);
        costiMaterie[i]=Lettore.tastiera.leggiDouble();
      }
      insOK=true;
      break;
    case 2:
      if(insOK==true){
        for(i=0;i<DIM_ARRAY;i++)
        Scrittore.video.println(nomiMaterie[i]+" "+costiMaterie[i]);
      }
      else
        Scrittore.video.println("Inserimento non effettuato");
      break;
    case 3:
      if(insOK==true){
        sommaCosti=0;
        for(i=0;i<DIM_ARRAY;i++)
          sommaCosti=sommaCosti+costiMaterie[i];
        mediaCosti=sommaCosti/(float)DIM_ARRAY;
        Scrittore.video.println("Media materie prime "+mediaCosti);
      }
      else
        Scrittore.video.println("Inserimento non effettuato");
      break;
    case 4:
      if(insOK==true){
        do{
          Scrittore.video.println("Inserire indice della prima materia prima ");
          primaMateria=Lettore.tastiera.leggiInt();
        }while(primaMateria<0 \vert\vert primaMateria>=DIM_ARRAY);
        do{
          Scrittore.video.println("Inserire indice della seconda materia prima ");
          secondaMateria=Lettore.tastiera.leggiInt();
        }while(secondaMateria<0 \vert\vert secondaMateria>=DIM_ARRAY);
        do{
          Scrittore.video.println("Inserire qta di  "+nomiMaterie[primaMateria]);
          qtaPrimaMateria=Lettore.tastiera.leggiInt();
        }while(qtaPrimaMateria<0);
        do{
          Scrittore.video.println("Inserire qta di  "+nomiMaterie[secondaMateria]);
          qtaSecondaMateria=Lettore.tastiera.leggiInt();
        }while(qtaSecondaMateria<0);
        costoProdotto=costiMaterie[primaMateria]*qtaPrimaMateria+
        costiMaterie[secondaMateria]*qtaSecondaMateria;
        Scrittore.video.println(costoProdotto);
      }
    else
```

```
                Scrittore.video.println("Inserimento non effettuato");
              break;
            }
        }while(opzione!=0);
    }
}
```

```
----------------------
1* Inserimento dati
2* Visualizzazione dati
3* Costo  medio materie prime
4* Costo  prodotto
0* USCITA
----------------------
1
Inserire nome prodotto di indice 0
ferro
Inserire prezzo di ferro
10
Inserire nome prodotto di indice 1
rame
Inserire prezzo di rame
8
Inserire nome prodotto di indice 2
ghisa
Inserire prezzo di ghisa
5
----------------------
1* Inserimento dati
2* Visualizzazione dati
3* Costo  medio materie prime
4* Costo  prodotto
0* USCITA
----------------------
2
ferro 10.0
rame 8.0
ghisa 5.0
----------------------
```

```
----------------------
1* Inserimento dati
2* Visualizzazione dati
3* Costo  medio materie prime
4* Costo  prodotto
0* USCITA
----------------------
3
Media materie prime 7.666666666666667
----------------------
1* Inserimento dati
2* Visualizzazione dati
3* Costo  medio materie prime
4* Costo  prodotto
0* USCITA
----------------------
4
Inserire indice della prima materia prima
0
Inserire indice della seconda materia prima
1
Inserire qta di  ferro
3
Inserire qta di  rame
5
70.0
----------------------
```

il prodotto composto è costituito da 3 parti di ferro (PU 10) e 5 parti di rame (PU 8) per cui il suo costo è dato da:
3 x 10 + 5 x 8 =70

Lavoro in aula / laboratorio

∘Aggiungere nel programma Java precedente le seguenti opzioni:

5 Sconto (chiede la percentuale di sconto e la applica a tutto il listino dei prodotti, eventualmente far riferimento all'opzione effetto inflazione dell'esercizio che segue)

6 Determinare il costo differenziale di magazzino dato dalla diffenza fra il costo materia prima maggiore e il costo materia prima minore

4.3 Array: ricerca semplice, determinazione massimo, elaborazione

Nell'esempio che segue (che riprende la struttura ad array paralleli analizzata precedentemente), oltre alle normali operazioni di inserimento e visualizzazione, viene introdotto un primo algoritmo di ricerca, uno di individuazione del massimo e una procedura di modifica degli elementi dell'array

Listato Java (RicercaArray.java)

```
import java.io.*;
import unibs.eco.dmq.basicIO.*;
public class RicercaArray{
  public static void main(String[] args) {
    final int
      DIM_ARRAY=3;
    double []
      costiProdotti;
    double
      tassoInflazione;
    String []
      nomiProdotti;
    String
      prodottoDaCercare;
    boolean
      insOK;
    int
      i,
      posizioneProdotto,
      opzione;
    /*Inizio*/
    insOK=false;
    costiProdotti=new double[DIM_ARRAY];
    nomiProdotti=new String[DIM_ARRAY];
    do{
      Scrittore.video.println("----------------------");
      Scrittore.video.println("1* Inserimento dati");
      Scrittore.video.println("2* Visualizzazione dati");
      Scrittore.video.println("3* Ricerca semplice ");
      Scrittore.video.println("4* Determinazione massimo ");
      Scrittore.video.println("5* Effetto inflazione");
      Scrittore.video.println("0* USCITA");
      Scrittore.video.println("----------------------");
      opzione=Lettore.tastiera.leggiInt();
      switch(opzione){
        case 1:
               .
               .
               .
        case 2:
               .
               .
               .
        case 3:
          if(insOK==true){
            Scrittore.video.println("Inserire prodotto da cercare");
            prodottoDaCercare=Lettore.tastiera.leggiString();
            posizioneProdotto=-1;
            for(i=0;i<DIM_ARRAY;i++)
```

```
            if(prodottoDaCercare.equals(nomiProdotti[i]))
              posizioneProdotto=i;
          if(posizioneProdotto==-1)
            Scrittore.video.println("Prodotto non presente in elenco");
          else{
            Scrittore.video.println("Il prodotto "+nomiProdotti[posizioneProdotto]);
            Scrittore.video.println("si trova nella posizione "+posizioneProdotto);
            Scrittore.video.println("e costa "+costiProdotti[posizioneProdotto]);
          }
        }
        else
          Scrittore.video.println("Inserimento non effettuato");
        break;
      case 4:
        if(insOK==true){
          posizioneProdotto=0;
            for(i=1;i<DIM_ARRAY;i++)
              if(costiProdotti[posizioneProdotto]<costiProdotti[i])
                posizioneProdotto=i;
              Scrittore.video.println("Il prodotto "+nomiProdotti[posizioneProdotto]);
              Scrittore.video.println("in posizione "+posizioneProdotto);
              Scrittore.video.println("ha il prezzo massimo pari a "+
                costiProdotti[posizioneProdotto]);
            }
        else
          Scrittore.video.println("Inserimento non effettuato");
        break;
      case 5:
        if(insOK==true){
          Scrittore.video.println("Inserire tasso d'inflazione ");
          tassoInflazione=Lettore.tastiera.leggiDouble();
          for(i=0;i<DIM_ARRAY;i++)
            costiProdotti[i]=costiProdotti[i]*(1.0+tassoInflazione/100.0);
        }
        else
          Scrittore.video.println("Inserimento non effettuato");
        break;
    }
  }while(opzione!=0);
 }
}
```

```
----------------------
1* Inserimento dati
2* Visualizzazione dati
3* Ricerca semplice
4* Determinazione massimo
5* Effetto inflazione
0* USCITA
----------------------
1
Inserire nome prodotto di indice 0
rame
Inserire prezzo di rame
11
Inserire nome prodotto di indice 1
ferro
Inserire prezzo di ferro
12
Inserire nome prodotto di indice 2
acciaio
Inserire prezzo di acciaio
9
```

```
----------------------
1* Inserimento dati
2* Visualizzazione dati
3* Ricerca semplice
4* Determinazione massimo
5* Effetto inflazione
0* USCITA
----------------------
3
Inserire prodotto da cercare
ferro
Il prodotto ferro
si trova nella posizione 1
e costa 12.0
```

```
----------------------
1* Inserimento dati
2* Visualizzazione dati
3* Ricerca semplice
4* Determinazione massimo
5* Effetto inflazione
0* USCITA
----------------------
4
Il prodotto ferro
in posizione 1
ha il prezzo massimo pari a 12.0
```

```
----------------------
1* Inserimento dati
2* Visualizzazione dati
3* Ricerca semplice
4* Determinazione massimo
5* Effetto inflazione
0* USCITA
----------------------
5
Inserire tasso d'inflazione
2.4
----------------------
1* Inserimento dati
2* Visualizzazione dati
3* Ricerca semplice
4* Determinazione massimo
5* Effetto inflazione
0* USCITA
----------------------
2
rame 11.264
ferro 12.288
acciaio 9.216000000000001
```

Note al listato

L'algoritmo di ricerca semplice viene realizzato ponendo la variabile che memorizza la posizione in cui si trova l'elemento uguale al valore arbitrario -1. Se durante la ricerca viene individuato l'elemento all'interno dell'array, tale variabile assume il valore dell'indice di scorrimento del vettore. Alla fine della scansione, se la variabile di posizione dell'elemento è rimasta uguale a -1, l'elemento non è stato individuato, mentre se la variabile ha un valore diverso, l'elemento è stato trovato e si trova proprio nella posizione corrispondente al valore della variabile.(Per ora non affrontiamo il problema di più elementi che soddisfano la ricerca all'interno dell'array).

L'Algoritmo di individuazione del massimo prevede la classica tecnica della sentinella. Viene memorizzata nell'indice dell'ipotetico valore massimo la posizione del primo elemento e si comincia a scorrere l'array. Se uno degli elementi dell'array è maggiore di quello riferito alla posizione del massimo allora la posizione del massimo diventa quella dell'indice dell'elemento appena considerato.(Per ora non affrontiamo il problema di più elementi che hanno valore uguale al massimo. La prassi in questo caso è, una volta inviduato il massimo, effettuare una nuova scansione sequenziale per selezionate i molteplici valori massimi).

L'Algoritmo di modifica elementi per adeguamento all'inflazione prevede una semplice scansione degli elementi con aggiornamento dei loro valori in base al tasso d'inflazione.

Lavoro in aula / laboratorio

- Aggiungere nel programma Java precedente le seguenti opzioni:

  7 Elenco di tutti i prodotti il cui prezzo è superiore al prezzo medio di listino

4.4 Array: booleani, relazione posizionali multiple

Nell'esempio che segue si vogliono gestire tramite array delle transazioni commerciali internazionali. Questi movimenti possono essere movimenti in uscita (Esportazioni) o movimenti in entrata (Importazioni) e possono essere in una delle seguenti valute: Euro o Dollari. La tipologia del movimento (Importazione/Esportazione) e la valuta (Euro/Dollari) sono individuate attraverso due array paralleli in relazione posizionale fra loro e con quello principale che contiene l'ammontare del movimento commerciale. Gli array tipoMovimenti e valutaMoviventi sono di tipo booleano dove i valori true e false sono associati al significato indicato nella figura sottostante:

tipoMovimenti

true	Importazione
false	Esportazione

valutaMovimenti

true	Euro
false	Dollari

Indice	tipoMovimenti		valutaMovimenti		importoMovimenti
0	true	⇆	false	⇆	100
1	true	⇆	true	⇆	200
2	false	⇆	true	⇆	300

Ad esempio considerando i valori dei tre vettori con indice pari a 1, ho un movimento di Importazione (tipoMovimenti[1]=true), in Euro (valutaMovimenti[1]=true) pari a 200.

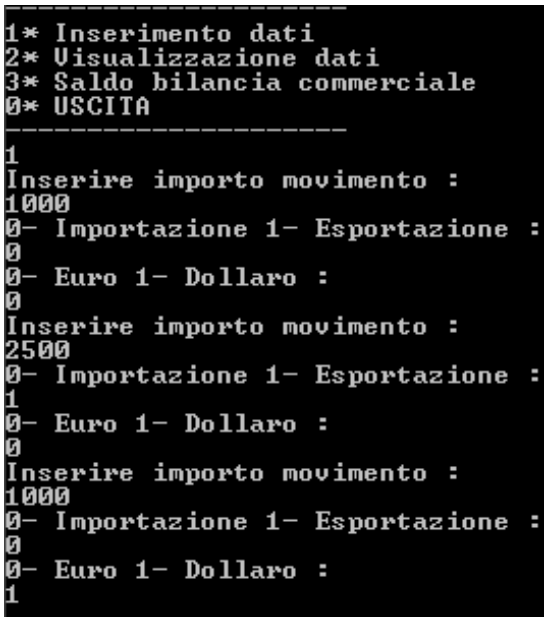
Oltre alle normali operazioni di inserimento e visualizzazione si vuole determinate il saldo della bilancia commerciale (Esportazioni-Importazioni) in una delle due valute.

Listato Java (ImportArray.java)

```
import java.io.*;
import unibs.eco.dmq.basicIO.*;
public class ImportArray{
  public static void main(String[] args) {
    final int
      DIM_ARRAY=3;
    double[]
      importoMovimenti;
    double
      sommaImportazioni,
      sommaEsportazioni,
      saldoBilancia,
      movimentoMaggiore;
    boolean[]
    /* true-Importazione false-Esportazione*/
      tipoMovimenti,
    /* true-Euro false-Dollari*/
      valutaMovimenti;
    boolean
       tipoMaggiore,
       valutaMaggiore,
       valutaScelta,
```

```
   insOK;
String
   valutaStampa,
   tipoStampa;
int
   i,
   sceltaTipo,
   opzione;
/*Inizio*/
insOK=false;
importoMovimenti=new double [DIM_ARRAY];
tipoMovimenti=new boolean [DIM_ARRAY];
valutaMovimenti=new boolean [DIM_ARRAY];
  do{
    Scrittore.video.println("----------------------");
    Scrittore.video.println("1* Inserimento dati");
    Scrittore.video.println("2* Visualizzazione dati");
    Scrittore.video.println("3* Saldo bilancia commerciale");
    Scrittore.video.println("0* USCITA");
    Scrittore.video.println("----------------------");
    opzione=Lettore.tastiera.leggiInt();
      switch(opzione){
        case 1:
          for(i=0;i<DIM_ARRAY;i++){
            Scrittore.video.println("Inserire importo movimento : ");
            importoMovimenti[i]=Lettore.tastiera.leggiDouble();
            Scrittore.video.println("0- Importazione 1- Esportazione : ");
            sceltaTipo=Lettore.tastiera.leggiInt();
            tipoMovimenti[i]=sceltaTipo==0;
            Scrittore.video.println("0- Euro 1- Dollaro : ");
            sceltaTipo=Lettore.tastiera.leggiInt();
            valutaMovimenti[i]=sceltaTipo==0;
          }
        insOK=true;
        break;
        case 2:
          if(insOK==true){
            for(i=0;i<DIM_ARRAY;i++){
              Scrittore.video.println("Movimento commerciale nr : "+i);
              if (valutaMovimenti[i])
                valutaStampa="Euro";
              else
                valutaStampa="Dollari";
              if (tipoMovimenti[i])
                tipoStampa="Importazione";
              else
                tipoStampa="Esportazione";
              Scrittore.video.println("Valuta : "+valutaStampa);
              Scrittore.video.println("Tipo : "+tipoStampa);
              Scrittore.video.println("Ammontare : "+importoMovimenti[i]);
              Scrittore.video.println("--------------------");
            }
          }
          else
            Scrittore.video.println("Inserimento non effettuato");
          break;
        case 3:
```

```
            if(insOK==true){
              sommaImportazioni=0;
              sommaEsportazioni=0;
              saldoBilancia=0;
              Scrittore.video.println("0- Euro 1- Dollaro : ");
              sceltaTipo=Lettore.tastiera.leggiInt();
              valutaScelta=sceltaTipo==0;
              if (valutaScelta)
                valutaStampa="Euro";
              else
                valutaStampa="Dollari";
              for(i=0;i<DIM_ARRAY;i++)
                if(valutaMovimenti[i]==valutaScelta)
                  if(tipoMovimenti[i])
                    sommaImportazioni=sommaImportazioni+importoMovimenti[i];
                  else
                    sommaEsportazioni=sommaEsportazioni+importoMovimenti[i];
              saldoBilancia=sommaEsportazioni-sommaImportazioni;
              if(saldoBilancia==0)
                Scrittore.video.println("Equilibrio nella bilancia commerciale");
              else if (saldoBilancia<0)
                Scrittore.video.println("Deficit commerciale di: "+Math.abs(saldoBilancia)
                +" "+valutaStampa);
              else
                Scrittore.video.println("Surplus commerciale di: "+saldoBilancia
                +" "+valutaStampa);
            }
            else
              Scrittore.video.println("Inserimento non effettuato");
          break;
        }
      }while(opzione!=0);
  }
}
```



Commento al listato

Nelle operazioni di inserimento non essendo possibile l'inserimento diretto di variabili booleane viene chiesto un codice numerico che vale 0 per memorizzare un valore booleano true e 1 (più precisamento diverso da 0) per memorizzare un valore booleano false. L'assegnazione dei valori booleani agli array avviene tramite la seguente uguaglianza: *tipoMovimenti[i]=sceltaTipo==0* (nell'array tipoMovimenti alla posizione i viene memorizzato il risultato booleano della verifica dell'uguaglianza nell'espressione sceltaTipo==0).

Nelle operazioni di determinazione del saldo della bilancia commerciale, la scelta principale riguarda il tipo di valuta di cui si vuole determinare il saldo. Il risultato potra essere: equilibrio/avanzo/disavanzo commerciale.Nel caso di disavanzo commerciale essendo il saldo negativo viene applicato l'operatore matematico di valore assoluto.

Lavoro in aula / laboratorio

- Aggiungere nel programma Java precedente le seguenti opzioni:

  4 Movimento commerciale maggiore
  (visualizzare insieme al movimento maggiore anche la sua tipologia di Importazione/Esportazione e valuta Euro/Dollari, ricordare che valuta e tipo hanno lo stesso indice del movimento maggiore)

4.5 Proposte di lavoro

Proposta Uno

Realizzare, utilizzando i vettori, l'inserimento dei nomi e dei tassi di cambio da Euro ad altre valute. Gestire le operazioni di inserimento, visualizzazione e trasformazione, dato in ingresso un valore in Euro, in tutte le altre valute prese in considerazione

Esempio tassi cambio Euro/valute al 30/08/04

Dollaro US	1,2028
Yen Giapponese	132,47
Sterlina Inglese	0,6710
Franco Svizzero	1,5400
Dollaro Australiano	1,7162
Dollaro Canadese	1,5839

Proposta Due

Un programma di rilevazione memorizza su file le quotazioni di due azioni ad intervalli di due ore. Le contrattazioni di borsa iniziano alle ore 9:30 e finiscono alle ore 17:30 (semplificato) per cui ci sono 4 rilevazioni giornaliere. Se un'azione non è quotata (es.sospensione rialzo/ribasso) non viene registrato il dato relativo all'azione sul file. Il formato del record sarà: nome azione, rilevamento, quotazione.

Esempio file

```
FIAT  1  5.83
ENI   1  16.92
FIAT  2  6.08
ENI   2  16.95
ENI   3  16.98
ENI   4  16.97
```

(Per l'azione "ENI" abbiamo le quattro rilevazioni mentre per l'azione FIAT abbiamo due rilevazioni probabilmente dovute ad una sospensione per eccesso di rialzo visto il brusco movimento di prezzo dalla rilevazione da 5.83 a 6.08)

Si vuole realizzare un'applicazione java che permetta:

il trasferimento delle informazioni da file ad array (è sufficiente disporre di due variabili stringa dove memorizzare il nome delle azioni ed associati a queste due array per memorizzare la quotazioni associando il numero di rilevazione all'indice es. indice 0 rilevazione 1,indice 1 rilevazione 2 .....)

Nome	Nome
FIAT	ENI
Quotazione	Quotazione
5.83	16.92
6.08	16.95
0	16.98
0	16.97

la determinazione del valore medio delle due azioni;

la determinazione della quotazione massima e minima raggiunte con indicazione dell'intervallo di contrattazione.

Capitolo 5

Argomenti trattati

- Array multidimensionali: Test di selezione

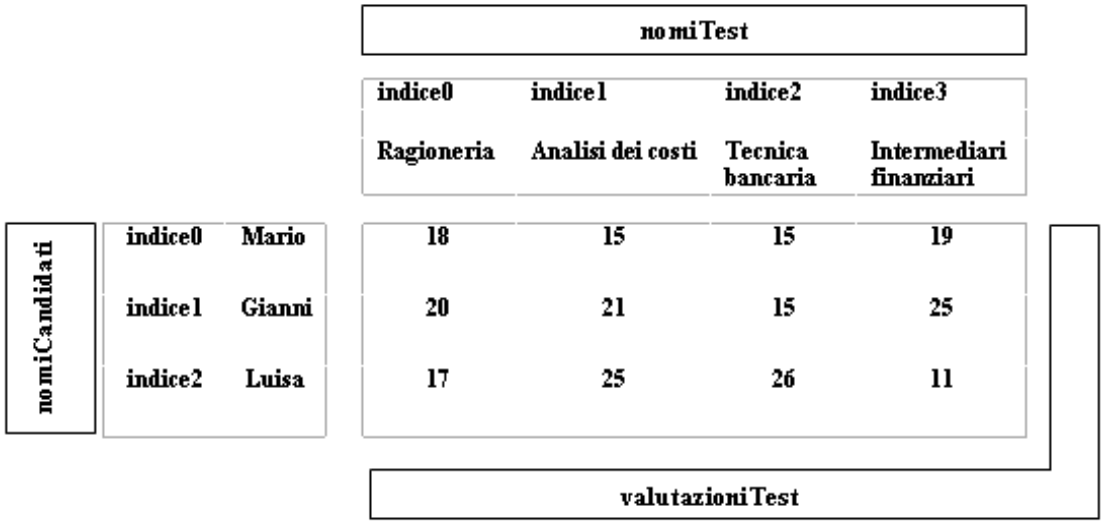
- Gestione files

- Gestione database

- Proposte di lavoro

5.1 Array multidimensionali: Test di selezione

Nell'esempio che segue si vogliono schedare i risultati ottenuti da alcuni candidati in un test di selezione che prevede un certo numero di prove. I nomi dei candidati sono memorizzati nell'array nomiCandidati con dimensione DIM_RIGA, i nomi delle prove del test sono memorizzati come stringhe fisse nell'array DIM_COLONNA, mentre i risultati sono memorizzari in un array bidimensionale con dimensione DIM_RIGA x DIM_COLONNA. Le relazioni posizionali sono per riga fra l'array dei nomi candidati e la matrice risultati (cioè data una posizione di riga, ho in quella posizione il nome del candidato nell'array nomi e tutti i risultati dei test relativi a quel nome nella matrice risultati) e per colonna fra l'array nomi delle prove e la matrice risultati (cioè data una posizione di colonna ho il nome della prova nell'array prove e tutti i risultati dei test relativi alla prova nella matrice risultati) come rappresentato nella figura sottostante:



Oltre alle normali procedure di inserimento dati e visualizzazione si vuole determinate il candidato migliore (quello che mediamente ha fatto meglio i test) e i candidati migliori da selezionare per le singole discipline (tutti quelli che hanno avuto punteggio massimo).

Listato Java (TestArray.java)

```
import java.io.*;
import unibs.eco.dmq.basicIO.*;
public class TestArray{
  public static void main(String[] args) {
    final int
      DIM_RIGA=3;
    final int
      DIM_COLONNA=4;
    int [][]
      valutazioniTest;
    double
      mediaCandidato,
      mediaMassima;
    String []
      nomiCandidati,
      nomiTest={"Ragioneria","Analisi_dei_costi","Tecnica_Bancaria","Intermediari_Finanziari"};
    String
      materiaSelezionata;
    boolean
      insOK;
      int
```

```
    i,
    j,
    indiceMateria,
    massimoCandidato,
    indiceMediaMassima,
    opzione;
  /*Inizio*/
    insOK=false;
    valutazioniTest=new int [DIM_RIGA][DIM_COLONNA];
    nomiCandidati=new String[DIM_RIGA];
    do{
      Scrittore.video.println("---------------------");
      Scrittore.video.println("1* Inserimento dati");
      Scrittore.video.println("2* Visualizzazione dati");
      Scrittore.video.println("3* Candidato con media maggiore");
      Scrittore.video.println("4* Selezione candidato");
      Scrittore.video.println("0* USCITA");
      Scrittore.video.println("---------------------");
      opzione=Lettore.tastiera.leggiInt();
      switch(opzione){
      case 1:
        for(i=0;i<DIM_RIGA;i++){
          Scrittore.video.println("Nome Candidato : ");
          nomiCandidati[i]=Lettore.tastiera.leggiString();
          for(j=0;j<DIM_COLONNA;j++){
            Scrittore.video.println(" Prova di "+nomiTest[j]+" inserire valutazione");
            valutazioniTest[i][j]=Lettore.tastiera.leggiInt();
          }
        }
        insOK=true;
        break;
      case 2:
        if(insOK==true){
          for(i=0;i<DIM_RIGA;i++){
            Scrittore.video.println("Valutazioni Candidato : "+nomiCandidati[i]);
            for(j=0;j<DIM_COLONNA;j++)
              Scrittore.video.println(nomiTest[j]+" "+valutazioniTest[i][j]);
              Scrittore.video.println("----------------------");
            }
          }
        else
          Scrittore.video.println("Inserimento non effettuato");
        break;
      case 3:
        mediaCandidato=0;
        mediaMassima=0;
        indiceMediaMassima=0;
        if(insOK==true){
          for(i=0;i<DIM_RIGA;i++){
            mediaCandidato=0;
            for(j=0;j<DIM_COLONNA;j++)
              mediaCandidato=mediaCandidato+valutazioniTest[i][j];
              mediaCandidato=mediaCandidato/(float)DIM_COLONNA;
              if (mediaCandidato>mediaMassima){
                mediaMassima=mediaCandidato;
                indiceMediaMassima=i;
              }
```

```
            }
          Scrittore.video.println("Media Maggiore: "
            +nomiCandidati[indiceMediaMassima]+" "+mediaMassima);
        }
        else
          Scrittore.video.println("Inserimento non effettuato");
        break;
      case 4:
          if(insOK==true){
            Scrittore.video.println("Per quale disciplina vuoi individuare il candidato ??");
            materiaSelezionata=Lettore.tastiera.leggiString();
            indiceMateria=-1;
            for (j=0;j<DIM_COLONNA;j++)
              if (materiaSelezionata.equals(nomiTest[j]))
                indiceMateria=j;
            if(indiceMateria==-1)
              Scrittore.video.println("Materia "+materiaSelezionata+" non presente");
            else{
              massimoCandidato=valutazioniTest[0][indiceMateria];
                for(i=1;i<DIM_RIGA;i++)
                  if(massimoCandidato<valutazioniTest[i][indiceMateria])
                    massimoCandidato=valutazioniTest[i][indiceMateria];
                for(i=0;i<DIM_RIGA;i++)
                  if(valutazioniTest[i][indiceMateria]==massimoCandidato)
                    Scrittore.video.println(nomiCandidati[i]+" ha il massimo :"+
                      massimoCandidato+" in "+materiaSelezionata);
            }
          }
          else
            Scrittore.video.println("Inserimento non effettuato");
          break;
      }
    }while(opzione!=0);
  }
}
```

```
--------------------
1* Inserimento dati
2* Visualizzazione dati
3* Candidato con media maggiore
4* Selezione candidato
0* USCITA
--------------------
2
Valutazioni Candidato : Gigi
Ragioneria  11
Analisi_dei_costi  12
Tecnica_Bancaria  24
Intermediari_Finanziari  25
----------------------
Valutazioni Candidato : Gianni
Ragioneria  28
Analisi_dei_costi  18
Tecnica_Bancaria  22
Intermediari_Finanziari  23
----------------------
Valutazioni Candidato : Maria
Ragioneria  22
Analisi_dei_costi  24
Tecnica_Bancaria  23
Intermediari_Finanziari  21
----------------------
```

```
--------------------
1* Inserimento dati
2* Visualizzazione dati
3* Candidato con media maggiore
4* Selezione candidato
0* USCITA
--------------------
3
Media Maggiore: Gianni 22.75
```

```
--------------------
1* Inserimento dati
2* Visualizzazione dati
3* Candidato con media maggiore
4* Selezione candidato
0* USCITA
--------------------
4
Per quale disciplina vuoi individuare il candidato ??
Ragioneria
Gianni ha il massimo :28 in Ragioneria
```

Note al listato

L'array relativo ai nomi dei test viene inizializzato con delle stringhe fisse in sede di dichiarazione (vengono evitati spazi nei nomi delle prove per evitare problemi in inserimento, oppure l'inserimento può essere effettuato con il metodo leggiRiga che considera l'invio con terminazione della stringa e non gli spazi).

Nella determinazione del candidato con media maggiore viene determinata per ogni riga la media delle valutazioni del candidato che viene successivamente confrontata con le medie dei restanti candidati.(Per semplicità non viene considerato il caso di candidati con media uguale).

Nella determinazione dei migliori candidati per ogni prova, viene inizialmente richiesto per quale prova si voglioni selezionare i candidati, se la prova è fra quella previste nella schedatura viene determinato il massimo riferito all'indice corrispondente. Per essere sicuro di elencare tutti i candidati che hanno ottenuto il massimo viene fatta una nuova scansione.

Lavoro in aula / laboratorio

- Ampliare il listato Java precedente introducendo l'opzione 5 che individua il nome del candidato che ha ottenuto la valutazione migliore in assoluto e la disciplina in cui tale valutazione è stata ottenuta
- Individuare la disciplina in cui i candidati sono andati meglio (ricerca del massimo sulle medie delle discipline)

5.2 Gestione files

La comodità dell'utilizzo dei vettori è legata al forte svantaggio di essere strutture residenti in memoria RAM il cui contenuto viene perso all'uscita dell'elaborazione e/o allo spegnimento del calcolatore. La possibilità di memorizzare i dati in modo permanente permettere di risolvere questa problematica.

Il package unibs mette a disposizione alcuni metodi per l'elaborazione dei files.

Principali operazioni per la SCRITTURA dei files con il package unibs

(le operazioni di scrittura possono essere fortemente limitate a seconda dei sistemi operativi o dinamiche amministrative per problemi di sicurezza)

Dichiarazione file per scrittura	Scrittore fileMovimentiScrivi
Istanza oggetto file	fileMovimentiScrivi=new Scrittore(nomeFile) se il file è preesistente viene perso il contenuto precedente
Scrittura variabile sul file	fileMovimentiScrivi.print(tipoMovimento)
Chiusura file	fileMovimentiScrivi.chiudi()

Principali operazioni per la LETTURA dei files con il package unibs

Dichiarazione file per lettura	Lettore fileMovimentiLeggi
Istanza oggetto file	fileMovimentiLeggi=new Lettore(nomeFile)
Lettura variabile dal file	importoMovimento=fileMovimentiLeggi.leggiInt()
Chiusura file	fileMovimentiLeggi.chiudi()

Un esempio di utilizzo dei files può essere il seguente:

si vogliono gestire la informazioni relative ad una serie di movimenti finanziari. Ogni movimento finanziario è costituito dalla coppia: tipoMovimento (dare / avere) e importo movimento. I movimenti devono poter essere memorizzati in modo permanente, devono poter essere letti e si devono poter eseguire alcune operazioni (saldo, media importi).

Una volta realizzato il programma di gestione è possibile creare un file dei movimenti che può essere trasferito (mail, supporto di memorizzazione) per la lettura ad un altro programma che ne conosce il tracciato record oppure è possibile leggere files provenienti dall'esterno (mail, supporto di memorizzazione) con un formato record compatibile.

Listato Java(GestioneFile.java)

```
import java.io.*;
import unibs.eco.dmq.basicIO.*;
/* Realizzazione gestione dei dati di magazzino */
public class GestioneFile{
  public static void main(String[] args) {
    String
      nomeFile,
      tipoMovimento;
    int
      i,
      importoMovimento,
      numeroMovimenti,
      sommaDare,
      sommaAvere,
      sommaMovimenti,
      opzione;
    double
      mediaMovimenti;
    Lettore
      fileMovimentiLeggi;
    Scrittore
      fileMovimentiScrivi;
```

```
/* Determino il nome del file */
if(args.length==0)
  nomeFile="movimenti.txt";
else
  nomeFile=args[0];
do{
  Scrittore.video.println("---------------------");
  Scrittore.video.println("1* Inserimento movimenti finanziari");
  Scrittore.video.println("2* Visualizzazione movimenti finanziari");
  Scrittore.video.println("3* Saldo finanziario");
  Scrittore.video.println("4* Media dei movimenti");
  Scrittore.video.println("0* Uscita");
  Scrittore.video.println("---------------------");
  opzione=Lettore.tastiera.leggiInt();
  switch(opzione){
    case 1:
         fileMovimentiScrivi=new Scrittore(nomeFile);
         do{
           Scrittore.video.println("Tipo movimento [d-dare a-avere u-uscita]");
           tipoMovimento=Lettore.tastiera.leggiString();
           if (!tipoMovimento.equals("u")){
             Scrittore.video.println("Inserisci importo movimento");
             importoMovimento=Lettore.tastiera.leggiInt();
             fileMovimentiScrivi.print(tipoMovimento);
             fileMovimentiScrivi.print(" ");
             fileMovimentiScrivi.println(importoMovimento);
           }
         }while(!tipoMovimento.equals("u"));
         fileMovimentiScrivi.chiudi();
         break;
    case 2:
         fileMovimentiLeggi=new Lettore(nomeFile);
         numeroMovimenti=fileMovimentiLeggi.contaRighe();
         for(i=0;i<numeroMovimenti;i++){
           tipoMovimento=fileMovimentiLeggi.leggiString();
           Scrittore.video.print(tipoMovimento+" ");
           importoMovimento=fileMovimentiLeggi.leggiInt();
           Scrittore.video.println(importoMovimento);
         }
         fileMovimentiLeggi.chiudi();
         break;
    case 3:
         fileMovimentiLeggi=new Lettore(nomeFile);
         numeroMovimenti=fileMovimentiLeggi.contaRighe();
         sommaDare=0;
         sommaAvere=0;
         for(i=0;i<numeroMovimenti;i++){
           tipoMovimento=fileMovimentiLeggi.leggiString();
           importoMovimento=fileMovimentiLeggi.leggiInt();
           if (tipoMovimento.equals("a"))
             sommaAvere=sommaAvere+importoMovimento;
           if (tipoMovimento.equals("d"))
             sommaDare=sommaDare+importoMovimento;
         }
         fileMovimentiLeggi.chiudi();
         if(sommaDare>sommaAvere)
           Scrittore.video.println("Surplus Dare pari a "+(sommaDare-sommaAvere));
```

```
            else
              Scrittore.video.println("Surplus Avere pari a "+(sommaAvere-sommaDare));
            break;
        case 4:
            fileMovimentiLeggi=new Lettore(nomeFile);
            numeroMovimenti=fileMovimentiLeggi.contaRighe();
            sommaMovimenti=0;
            for(i=0;i<numeroMovimenti;i++){
              tipoMovimento=fileMovimentiLeggi.leggiString();
              importoMovimento=fileMovimentiLeggi.leggiInt();
              sommaMovimenti=sommaMovimenti+importoMovimento;
            }
            mediaMovimenti=sommaMovimenti/(double)numeroMovimenti;
            Scrittore.video.println("Media dei movimenti "+mediaMovimenti);
            break;
      }
    }while(opzione!=0);
  }
}
```

Es dato il seguente file di movimenti:

d 100
d 101
a 200
a 210
d 300
d 100
a 115

ottengo i seguenti risultati

```
----------------------
1* Inserimento movimenti finanziari
2* Visualizzazione movimenti finanziari
3* Saldo finanziario
4* Media dei movimenti
0* Uscita
----------------------
3
Surplus Dare pari a 76
```

```
----------------------
1* Inserimento movimenti finanziari
2* Visualizzazione movimenti finanziari
3* Saldo finanziario
4* Media dei movimenti
0* Uscita
----------------------
4
Media dei movimenti 160.85714285714286
```

Commento al listato

Nella dichiarazione del metodo main viene passato come parametro una array di stringhe (public static void main(String[] args)). E' possibile in questo modo passare dei valori all'atto dell'esecuzione del programma. Ciascuno dei valori passati viene memorizzato nelle posizioni successive dell'array args. Ad esempio richiamando da console il programma media in questo modo: java media 5 7, verranno memorizzati in args[0] il valore 5 e in args[1] il valore 7. All'interno del programma di gestione files, se viene passato insieme alla chiamata del programma un nome file allora questo nome diventa il nome da utilizzare altrimenti viene utilizzato un nome generico(movimenti.txt). Una chiamata del programma di questo tipo: java GestioneFile archivio.txt determina come nome da utilizzare per il file archivio.txt. Nel caso in cui non venga passato nessun parametro viene stabilito come nome di default movimenti.txt.

```
if(args.length==0)
      nomeFile="movimenti.txt";
    else
      nomeFile=args[0];
```

Nella fase di memorizzazione dei valori viene stabilito che il tracciato record è costituito da un carattere contente il tipo di movimento e l'importo. Per separare i due campi del record viene memorizzato uno spazio mentre per identificare la fine del record viene effettuata una scrittura dell'ultimo campo con un metodo println.

```
fileMovimentiScrivi.print(tipoMovimento);
fileMovimentiScrivi.print(" ");
fileMovimentiScrivi.println(importoMovimento);
```

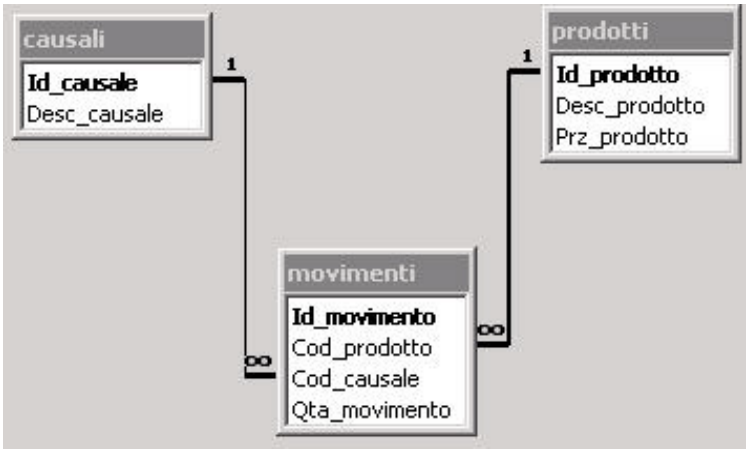
Lavoro in aula / laboratorio

- Generare un file movimenti in modo casuale e passarlo ad un altro gruppo di lavoro per l'elaborazione tramite il programma GestioneFile

  (Suggerimento: la generazione sia del tipo movimento che dell'importo può essere fatta con il metodo Math.random. Es se il valore estratto è < 0.5 allora memorizzo un movimento dare altrimenti memorizzo un movimento avere. Per gli importi, una volta stabilito l'importo massimo, la generazione causuale può essere fatta semplicemente con Math.random*importoMassimo)

5.3 Gestione database

Il package unibs dispone di una serie di metodi per poter interagire con delle basi di dati. Supponiamo di voler gestire una base di dati realizzata con MS-Access che memorizza i movimenti di carico e scarico di un magazzino (tralasciamo le nozioni relative all'integrità referenziale e forme di normalizzazione oggetto di altre discipline). Il database è costituito da una tabella principale movimenti e da due tabelle correlate che contengono le informazioni dettagliate dei codici memorizzati nella tabella principale. Lo schema delle relazioni è il seguente:



La singola riga movimento sarà un insieme di campi di questo tipo:

Id_movimento	Cod_prodotto	Cod_causale	Qta_movimento
1	4	1	11

come logico nella riga movimento vengono memorizzati i codici di riferimento ai prodotti ed ai movimenti secondo le pù semplici regole di gestione delle basi di dati. Attraverso questi codici è possibile accedere in fase di manipolazione/interrogazione delle tabelle alle informazioni associate. Nell'esempio sopra,

a Cod_prodotto = 4 corriponderà una riga nella tabella prodotti di questo tipo

Id_prodotto	Desc_prodotto	Prz_prodotto
4	ghisa	10

a Cod_causale = 1 corriponderà una riga nella tabella causali di questo tipo

Id_causale	Desc_causale
1	Uscita magazzi

unendo le informazioni, il movimento di magazzino interessato è della ghisa, con prezzo unitario di 10 euro in uscita per una quantità di 11 unità.

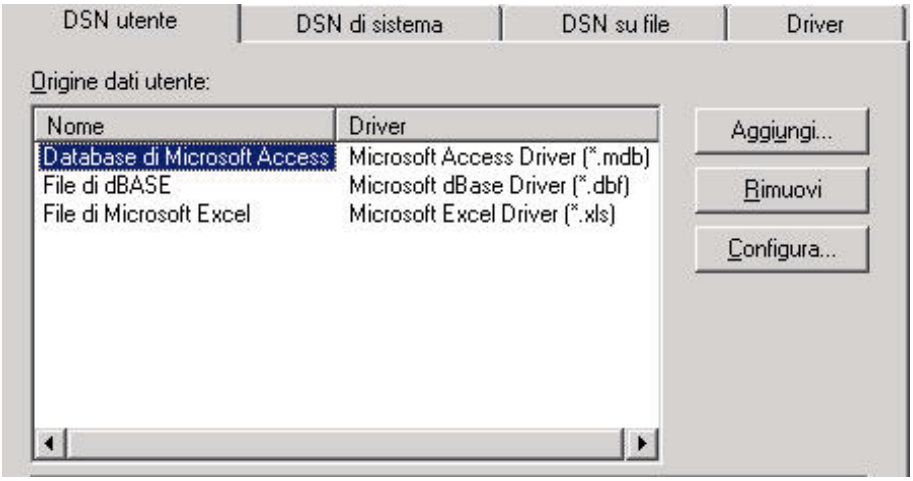
Dopo aver generato il database è necessario registrarlo nella sezione ODBC del sistema operativo:

- attraverso il sistema operativo bisogna selezionare lo strumenti origine dati ODBC

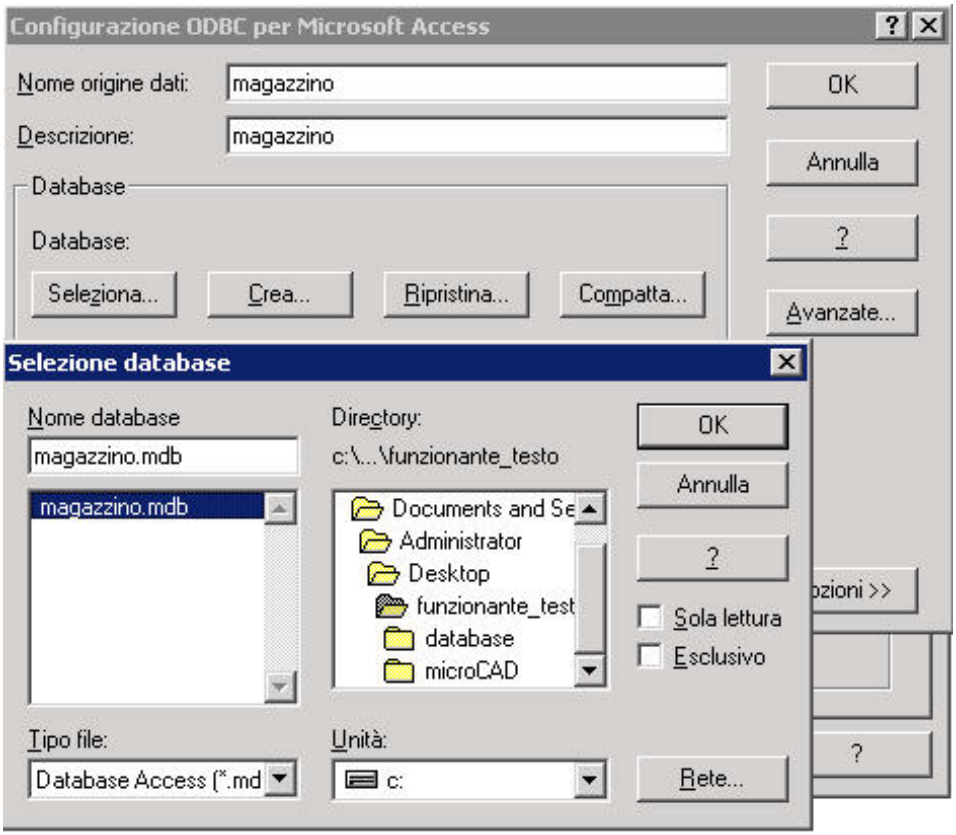
(con Windows XP il percorso è:

)

- una volta selezionata l'origine dati e necessario specificare una origine dati (MS-Access) come DSN utente tramite l'opzione aggiunta



- è necessario specificare il nome e la descrizione dell'orgine dati e attravero l'opzione seleziona collegare il nostro database access



Una volta impostata l'interfaccia ODBC è possibile sfruttare i metodi della classe database del package unibs per elaborare/gestire il database magazzino

Listato Java(AccessoDB.java)

```
import unibs.eco.dmq.database.*;
public class AccessoDB {
  public static void main (String[] argv){
    String matriceDati[][];
    String vettoreTestata[];
    String queryDatabase;
    String nomeDatabase;
```

```
    int i;
    nomeDatabase = "magazzino";
    queryDatabase = "select * from prodotti";
    vettoreTestata= null;
    matriceDati=null;
    /* attiva connessione al database */
    Database.apriConnessione(nomeDatabase);
    /* seleziona i nomi dei campi della selezione specificata */
    if (Database.eseguiQuery(queryDatabase)==null)
      vettoreTestata=Database.estraiNomiColonne();
    /* visualizza i nomi dei campi */
    for (i=0;i<vettoreTestata.length;i++)
      System.out.print(vettoreTestata[i]+" ");
    /* seleziona i dati della selezione specificata */
    if (Database.eseguiQuery(queryDatabase)==null)
      matriceDati=Database.estraiMatrice();
    System.out.println(" ");
    /* visualizza i dati della selezione */
    for (i=0;i<matriceDati.length;i++){
      for (int j=0;j<matriceDati[i].length;j++)
        System.out.print(" "+matriceDati[i][j]+" ");
      System.out.println("");
    }
    /* aggiunta di un prodotto */
    queryDatabase="insert into prodotti values ('6','Bronzo','12')";
    if (Database.eseguiUpdate(queryDatabase)==null){
      queryDatabase = "select * from prodotti";
      if (Database.eseguiQuery(queryDatabase)==null)
        matriceDati=Database.estraiMatrice();
    }
    System.out.println(" ");
    /* visualizza i dati della selezione */
    for (i=0;i<matriceDati.length;i++){
      for (int j=0;j<matriceDati[i].length;j++)
        System.out.print(" "+matriceDati[i][j]+" ");
        System.out.println("");
    }
    /* chisura connessione */
    Database.chiudiConnessione();
  }
 }
```

```
Id_prodotto Desc_prodotto Prz_prodotto
   1         rame         12.6
   2         ferro         11.0
   3         acciaio         12.0
   4         ghisa         10.0
   5         plastica         18.0

   1         rame         12.6
   2         ferro         11.0
   3         acciaio         12.0
   4         ghisa         10.0
   5         plastica         18.0
   6         Bronzo         12.0
Press any key to continue...
```

Commento al listato

La connessione al database specificato avviene attraverso il metodo *apriConnessione* passando come parametro il nome del database.

```
Database.apriConnessione(nomeDatabase)
```

La query di interrogazione avviene attraverso il metodo *eseguiQuery* che ha come parametro le istruzioni SQL relative all'interrogazione voluta. Il risultato della selezione può essere analizzato con il metodo *estraiNomiColonne* che permette di reperire i nomi dei campi del database ed il metodo *estraiMatrice* che contiene tutte le informazioni relative ai valori contenuti nella selezione.

```
queryDatabase = "select * from prodotti"
if (Database.eseguiQuery(queryDatabase)==null)
      matriceDati=Database.estraiMatrice()
```

Una query di modifica può essere quella definita dal metodo *eseguiUpdate* che nell'esempio precedente aggiunge una riga alla tabella prodotti. L'elenco dei dati aggiornati si può ottenere tramite una nuova query di selezione

```
queryDatabase="insert into prodotti values ('6','Bronzo','12')"
if (Database.eseguiUpdate(queryDatabase)==null){
      queryDatabase = "select * from prodotti";
      if (Database.eseguiQuery(queryDatabase)==null)
        matriceDati=Database.estraiMatrice()
```

All'uscita del programma viene chiusa la connessione interessata

Lavoro in aula / laboratorio

- Inserire nel listato java precedente le istruzioni necessarie per determinare il prezzo medio dei prodotti presenti (suggerimento: per convertire la stringa risultato del prezzo in numero usate il metodo Float.parseFloat)
- Visualizzare solo i prodotti con prezzo superiore a 10 euro (suggerimento: esplicitare con la clausula where dell'istruzione SQL la condizione desiderata)
- Realizzare un listato java che permetta tramite menu la gestione della tabella causali con le seguenti opzioni:
  elenco causali esistenti
  aggiunta nuova causale

.

5.4 Proposte di lavoro

Proposta Uno

Memorizzare su file i movimenti finanziari di gestione famigliare che possono essere:
SP - spesa
BO - bollette
ST - stipendi
caricare i dati in apposite strutture dati e soddisfare le seguenti richieste:
Riportare il saldo famigliare
Riportare il movimento finanziariamente più rilevante
Riportare l'elenco dei movimenti, la causale e relativo ammontare ordinati per causale (SP,BO,ST)

Proposta Due

Il sistema di monitoraggio aziendale prevede l'invio ogni settimana tramite file delle ore di lavoro(giornaliere) effettuate dai dipendenti. L'informazione registrata sul file è la seguente: nome dipendente,ore. Nel caso in cui il dipendente sia assente(ferie,malattia) viene registrato un movimento con ore di lavoro pari a -1 se è in ferie, -2 se è in malattia. Il file sarà quindi di questo tipo (esempio con due dipendenti):

ROSSI 8
ROSSI 9
ROSSI 8
ROSSI 9
ROSSI 8
ROSSI 9
VERDI 8
VERDI 8
VERDI 9
VERDI 8
VERDI -1
VERDI 8

L'elaborazione prevede il caricamento delle informazioni in una tabella che ha il numero di righe pari al numero dei dipendenti e il numero delle colonne pari a 6 (giorni lavorativi settimanali) che contiene le ore lavorate nei vari giorni (ad ogni indice viene correlato un giorno es. indice 0 giorno 1, indice 1 giorno 2). Questa tabella è in relazione posizionale con un vettore di stringhe contenente i nomi dei dipendenti. Nell'esempio visto prima le strutture generate saranno:

ROSSI	8	9	8	9	8	9
VERDI	8	8	9	8	-1	8

le funzionalità (metodi) da rendere disponibili all'ufficio personale sono:

- dato il nome di un dipendente fornire le ore lavorate in settimana (attenti a "saltare" i valori negativi)
- elencare i dipendenti e il numero di ore mediamente lavorate in settimana
- elencare i dipendenti assenti per ferie o malattia specificando se si trattava di ferie o malattia e il giorno
- dato un giorno settimanale elencare i dipendenti che hanno fatto straordinari (lavorato più di 8 ore)

Proposta tre

Predisporre un programma Java per gestire un conto corrente semplificato agganciato ad una base dati costituita da due tabelle:

Tabella Movimenti

Id_Progressivo Cod_Causale Importo
Tabella Causale
Id_Causale Desc_Causale

Capitolo 6

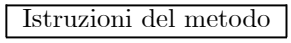
Argomenti trattati

- Introduzione ai metodi
- Metodi classici su array e metodi ricorsivi
- Package: utilizzo del package vettori
- Package: creazione del package formule
- La documentazione
- Proposte di lavoro

6.1 Introduzione ai metodi

Possiamo fare una classificazione logica dei metodi in base ai parametri d'ingresso ed ai valori di ritorno. E' possibile effettuare un'analisi delle funzionalità del metodo pensandolo come una scatola chiusa contenente le istruzioni necessarie a risolvere i nostri algoritmi in termini di: contenuti necessari all'algoritmo per l'esecuzione e valori che vengono restituiti a fine elaborazione. Con questa suddivisione possiamo ottenere 4 combinazioni:

Tipo 0



Questo tipo di metodo non necessita di parametri d'ingresso e non restituisce valori alla fine dell'elaborazione. Nella traduzione Java ci troveremo di fronte a

dichiarazioni di questo tipo:

```
public static void opzioneErrata()
```

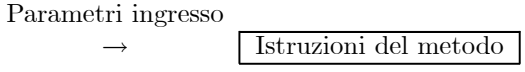
e utilizzi di questo tipo:

```
opzioneErrata();
```

La mancanza del tipo di ritorno viene evidenziata restituendo il tipo neutro void e la mancanza dei parametri viene evidenziata nel non specificare nulla fra la tonde della dichiarazione.

Un esempio può essere una metodo che fornisce avvisi o indicazioni tramite scritte statiche.

Tipo 1



Questo tipo di metodo necessita di parametri d'ingresso e non restituisce valori alla fine dell'elaborazione. Nella traduzione Java ci troveremo di fronte a

dichiarazioni di questo tipo:

```
public static void tabellina(double numero)
public static void visualizzazione(final double rendimentiOttenuti[])
```

e utilizzi di questo tipo:

```
tabellina(3);
visualizzazione(rendimentiOttenuti);
```

La mancanza del tipo di ritorno viene evidenziata restituendo il tipo neutro void. Il parametro d'ingresso viene specificato all'interno delle parentesi. I tipi di dato primitivi vengono passati per valore, quindi viene creata una copia temporanea di memoria che non produce effetti sulla variabile orginaria. I tipi derivati come potrebbero essere gli oggetti sono passati per riferimento.(Il passaggio del parametro per riferimento permette la modifica del parametro stesso)

Un esempio può essere il metodo per la visualizzazione degli elementi di un array oppure la visualizzazione della tabellina di un certo numero che viene passato come parametro.

Tipo 2

Questo tipo di metodo non necessita di parametri d'ingresso e restituisce dei valori alla fine dell'elaborazione. Nella traduzione Java ci troveremo di fronte a

dichiarazioni di questo tipo:

```
public static int menu ()
```

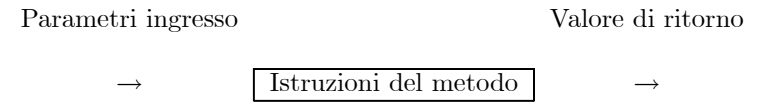
e utilizzi di questo tipo:

opzione=menu();

Il tipo di ritorno viene espressamente specificato, mentre la mancanza di parametri d'ingresso viene evidenziata nel non specificare nulla fra la tonde della dichiarazione.

Un esempio di questo metodo può essere la visualizzazione delle opzioni di un menu che restituisce l'opzione scelta.

Tipo 3



Questo tipo di metodo necessita di parametri d'ingresso ed è finalizzato a restituire un risultato in uscita. Nella traduzione Java ci troveremo di fronte a

dichiarazioni di questo tipo:

public static **int** massimo(**final double** rendimentiOttenuti[])

e utilizzi di questo tipo:

valoreMassimo=massimo(rendimentiOttenuti);

Il tipo di ritorno è espressamente specificato come pure il parametro.

Per meglio comprendere la gestione dei metodi e la concezione top-down di analisi degli algoritmi proviamo ad analizzare e risolvere il seguente problema:

si vuole gestire l'inserimento, la visualizzazione e l'individuazione del range di variazione (valore massimo - valore minimo) di una serie di rendimenti finanziari

Pensando ad una struttura a menu e ipotizzando che ci siano i metodi: inserimento, visualizzazione, massimo e minimo, la soluzione top-down del nostro problema può essere la seguente:

ESEGUI
  LEGGI opzione
  SE opzione = 1 allora INSERIMENTO
  ALTRIMENTI SE opzione = 2 allora VISUALIZZAZIONE
  ALTRIMENTI SE opzione = 3 allora MASSIMO
  ALTRIMENTI SE opzione = 4 allora RANGE_VARIAZIONE
  FINE SE
QUANDO opzione <> 0

Una volta "spezzettato" l'algoritmo in sottoproblematiche più agevoli da affrontare ci si concentra nell'analizzare e risolvere i sottoproblemi. Concentriamoci sul sottoproblema di INSERIMENTO. Da una prima analisi si può verificare che lo scopo dell'inserimento deve essere quello di scorrere il vettore caricando in ogni locazione un valore inserito dall'utente. Il vettore deve essere reso poi disponibile al metodo principale (main).Questa soluzione può essere ottenuta ricordando che essendo il vettore un dato non primitivo, nella chiamata al metodo verrà passato il suo riferimento, per cui ogni modifica effettuata all'interno del metodo, si rifletterà anche nel metodo main. In questo caso, il parametro passato, viene utilizzato anche per ottenere il valore di ritorno (vettore inizializzato con i dati utente).

Listato Java (TipiMetodi.java)

```
import java.io.*;
import unibs.eco.dmq.basicIO.*;
public class TipiMetodi{
  public static void main(String[] args) {
    final int
      DIM_ARRAY=3;
    double []
      rendimentiOttenuti;
    boolean
      insOK;
```

```
    int
      i,
      indiceMassimo,
      indiceMinimo,
      opzione;
 /*Inizio*/
 insOK=false;
 rendimentiOttenuti=new double[DIM_ARRAY];
 do{
     opzione=menu();
     switch(opzione){
       case 1:
         inserimento(rendimentiOttenuti);
         insOK=true;
         break;
       case 2:
         if(insOK==true)
           visualizzazione(rendimentiOttenuti);
         else
           Scrittore.video.println("Inserimento non effettuato ");
         break;
       case 3:
         if(insOK==true){
           indiceMassimo=massimo(rendimentiOttenuti);
           Scrittore.video.println("Massimo "+rendimentiOttenuti[indiceMassimo]);
           Scrittore.video.println("in posizione "+indiceMassimo);
         }
         else
           Scrittore.video.println("Inserimento non effettuato ");
         break;
       case 4:
         if(insOK==true){
           indiceMassimo=massimo(rendimentiOttenuti);
           indiceMinimo=minimo(rendimentiOttenuti);
           Scrittore.video.println("Il range da massimo a minimo "
            +(rendimentiOttenuti[indiceMassimo]-rendimentiOttenuti[indiceMinimo]));
         }
         else
           Scrittore.video.println("Inserimento non effettuato ");
         break;
       default:
         if (opzione!=0)
           opzioneErrata();
     }
  }while(opzione!=0);
}
/* Metodo opzioneErrata
* Parametro ingresso : nessuno(void)
* Valore restituito : nessuno(void)
*/
public static void opzioneErrata(){
  Scrittore.video.println("---------------------");
  Scrittore.video.println("L'opzione selezionata");
  Scrittore.video.println("risulta disabilitata");
  Scrittore.video.println("---------------------");
}
/* Metodo menu
```

```
* Parametro ingresso : nessuno(void)
* Valore restituito : opzione scelta(int)
*/
public static int menu(){
  int
    opzione;
  Scrittore.video.println("---------------------");
  Scrittore.video.println("1* Inserimento dati");
  Scrittore.video.println("2* Visualizzazione dati");
  Scrittore.video.println("3* Rendimento massimo");
  Scrittore.video.println("4* Range di variazione");
  Scrittore.video.println("0* USCITA");
  Scrittore.video.println("---------------------");
  opzione=Lettore.tastiera.leggiInt();
  return opzione;
}
/* Metodo inserimento
* Parametro ingresso : rendimentiOttenuti(array interi) x riferimento
* Valore restituito : nessuno (void)
*/
public static void inserimento(double rendimentiOttenuti[]){
  int
    i;
  for(i=0;i<rendimentiOttenuti.length;i++){
    Scrittore.video.println("Inserire rendimento di indice : "+i);
    rendimentiOttenuti[i]=Lettore.tastiera.leggiDouble();
  }
  return;
}
/* Metodo visualizzazione
* Parametro ingresso : rendimentiOttenuti(array interi) x riferimento
* Valore restituito : nessuno (void)
*/
public static void visualizzazione(final double rendimentiOttenuti[]){
  int
    i;
  for(i=0;i<rendimentiOttenuti.length;i++){
    Scrittore.video.println("Rendimento : "+i+" pari a "+rendimentiOttenuti[i]);
    Scrittore.video.println("----------------------");
  }
  return;
}
/* Metodo massimo
* Parametro ingresso : rendimentiOttenuti(array interi) x riferimento
* Valore restituito : indice massimo
*/
public static int massimo(final double rendimentiOttenuti[]){
  double
    massimo;
  int
    indiceMassimo,
    i;
  massimo=rendimentiOttenuti[0];
  indiceMassimo=0;
  for(i=1;i<rendimentiOttenuti.length;i++){
    if(massimo<rendimentiOttenuti[i]){
      massimo=rendimentiOttenuti[i];
```

```
       indiceMassimo=i;
     }
   }
   return indiceMassimo;
 }
 /* Metodo minimo
 * Parametro ingresso : rendimentiOttenuti(array interi) x riferimento
 * Valore restituito : indice minimo
 */
 public static int minimo(final double rendimentiOttenuti[]){
.......
.......
.......
 }
}
```

Lavoro in aula / laboratorio

- Completare il metodo relativo alla determinazione del minimo
- Introdurre l'opzione 5 che permette di visualizzare i primi N (dove N è dato in input dall'utente) rendimenti significativi.
  (suggerimento, basta effettuare un ordinamento e successivamente una visualizzazione degli N elementi. Per problemi relativi all'algoritmo di ordinamento vedere l'esempio che segue)

6.2 Metodi classici su array e metodi ricorsivi

Supponiamo di avere delle materie prime riposte in più magazzini. Periodicamente l'addetto al magazzino memorizza su file di testo il nome della materia e la sua giacenza aggiornata. L'ufficio responsabile del sistema produttivo e organizzativo, in base agli ordini che deve soddisfare, aggiorna periodicamente un file che contiene per ogni materia prima la giacenza minima da mantenere. Lo schema può essere rapresentato in questo modo:

Schema magazzino

magazzinoUno

Rame	100
Zinco	200
Plastica	150
Ferro	245
Acciaio	185

magazzinoDue

Legno	100
Alluminio	200
Pittura	150
Vetro	245

giacenzeMinime

Alluminio	100
Pittura	250
Acciaio	120
Rame	80
Legno	100
Zinco	100
Plastica	130
Ferro	240
Vetro	250

Si vuole permettere agli uffici centrali dell'azienda di:

reperire e unire le informazioni relative alle materie prime e loro giacenza aggiornata tramite accesso ai files di testo dei singoli magazzini.(supponiamo che tutti i magazzini gestiscano materie prime diverse);

elencare le materie prime ordinate in modo decrescente per giacenza;

verificare tramite controllo incrociato fra i dati delle giacenze reali e i limiti di giacenza minima se la materia contenuta nei magazzini è in situazione di carenza oppure di surplus rispetto alle esigenze aziendali.

Listato Java (GestioneArray.java)

```
import java.io.*;
import unibs.eco.dmq.basicIO.*;
/* Realizzazione gestione dei dati di magazzino */
public class GestioneArray{
 public static void main(String[] args) {
 /* Array delle giacenze */
 double []
   giacenzeMagazzino,
   giacenzeMinime;
 /* Array dei nomi delle materie */
 String []
   nomiMaterieMagazzino,
   nomiMaterieGiacenza;
 int
   i,
   opzione,
   numeroDatiMagazzinoUno,
   numeroDatiMagazzinoDue,
   numeroDatiGiacenze;
```

```
Lettore
  fileMagazzinoUno,
  fileMagazzinoDue,
  fileMagazzinoGiacenze;
/* Determinazione numero di dati da memorizzare */
fileMagazzinoUno=new Lettore("magazzino1.txt");
numeroDatiMagazzinoUno=fileMagazzinoUno.contaRighe();
/**/
fileMagazzinoDue=new Lettore("magazzino2.txt");
numeroDatiMagazzinoDue=fileMagazzinoDue.contaRighe();
/**/
fileMagazzinoGiacenze=new Lettore("giacenze.txt");
numeroDatiGiacenze=fileMagazzinoGiacenze.contaRighe();
/* Allocazione array */
giacenzeMagazzino=new double[numeroDatiGiacenze];
giacenzeMinime=new double[numeroDatiGiacenze];
nomiMaterieMagazzino=new String[numeroDatiGiacenze];
nomiMaterieGiacenza=new String[numeroDatiGiacenze];
/**/
/* Inizializzazione array sulla base dei files di dati*/
for(i=0;i<numeroDatiGiacenze;i++){
  nomiMaterieGiacenza[i]=fileMagazzinoGiacenze.leggiString();
  giacenzeMinime[i]=fileMagazzinoGiacenze.leggiDouble();
  if(i<numeroDatiMagazzinoUno){
    nomiMaterieMagazzino[i]=fileMagazzinoUno.leggiString();
    giacenzeMagazzino[i]=fileMagazzinoUno.leggiDouble();
  }
  else{
    nomiMaterieMagazzino[i]=fileMagazzinoDue.leggiString();
    giacenzeMagazzino[i]=fileMagazzinoDue.leggiDouble();
  }
}
fileMagazzinoUno.chiudi();
fileMagazzinoDue.chiudi();
fileMagazzinoGiacenze.chiudi();
/* ------------------------------------------------*/
do{
  opzione=menu();
  switch(opzione){
    case 1:
      Scrittore.video.println("Magazzino Unificato");
      visualizzazione(giacenzeMagazzino,nomiMaterieMagazzino);
      Scrittore.video.println("------------------------");
      Scrittore.video.println("Giacenze Minime");
      visualizzazione(giacenzeMinime,nomiMaterieGiacenza);
      break;
    case 2:
      ordinamentoValori(giacenzeMagazzino,nomiMaterieMagazzino);
      Scrittore.video.println("Ordinamento effettuaro");
      break;
    case 3:
    /* per sicurezza faccio l'ordinamento prerequisito della ric.bin.*/
      ordinamentoNomi(giacenzeMinime,nomiMaterieGiacenza);
      rispettoGiacenze(giacenzeMagazzino,giacenzeMinime,
      nomiMaterieMagazzino,nomiMaterieGiacenza);
      break;
  }
```

```
 }while(opzione!=0);
}
 /* Menu gestione */
 public static int menu(){
   int
     opzione;
   Scrittore.video.println("---------------------");
   Scrittore.video.println("1* Visualizzazione dati");
   Scrittore.video.println("2* Ordinamento magazzino per giacenza");
   Scrittore.video.println("3* Rispetto giacenze");
   Scrittore.video.println("0* USCITA");
   Scrittore.video.println("---------------------");
   opzione=Lettore.tastiera.leggiInt();
   return opzione;
 }
 /* Visualizzazione coppie Nomi - Giacenze */
 public static void visualizzazione(double valori[], String nomi[]){
   int i;
   for(i=0;i<valori.length;i++)
     Scrittore.video.print(nomi[i]+" "+valori[i]+" ; ");
   Scrittore.video.println();
   return;
 }
 /* Ordinamento semplice, l'array numerico comanda l'ordinamento */
 public static void ordinamentoValori(double valori[], String nomi[]){
   int i,
       j;
   double
     tempDouble;
   String
     tempString;
   for(i=0;i<valori.length-1;i++)
     for(j=i+1;j<valori.length;j++)
       if(valori[i]<valori[j]){
         tempDouble=valori[i];
         valori[i]=valori[j];
         valori[j]=tempDouble;
         tempString=nomi[i];
         nomi[i]=nomi[j];
         nomi[j]=tempString;
       }
   return;
 }
 /* Ordinamento semplice, l'array stringa comanda l'ordinamento */
 public static void ordinamentoNomi(double valori[], String nomi[]){
   int i,
       j;
   double
     tempDouble;
   String
     tempString;
   for(i=0;i<valori.length-1;i++)
     for(j=i+1;j<valori.length;j++)
       if(nomi[i].compareTo(nomi[j])>0){
         tempDouble=valori[i];
         valori[i]=valori[j];
         valori[j]=tempDouble;
```

```
        tempString=nomi[i];
        nomi[i]=nomi[j];
        nomi[j]=tempString;
      }
  return;
 }
 /* Ricerca binaria ricorsiva */
 public static int ricercaBinaria(String nomi[], String nomeDaCercare,int basso , int alto){
  int medio;
  medio = (alto+basso)/ 2;
  if (alto < basso)
    return -1;
  else if (nomi[medio].equals(nomeDaCercare))
    return medio;
  else if (nomi[medio].compareTo(nomeDaCercare)<0)
    return ricercaBinaria(nomi, nomeDaCercare, medio + 1, alto);
  return ricercaBinaria(nomi, nomeDaCercare, basso, medio - 1);
 }
 /*Determinazione rispetto delle giacenze */
 public static void rispettoGiacenze(double valoriMagazzino[],double valoriGiacenza[],
  String nomiMagazzino[], String nomiGiacenza[]){
  int
    i,
    indiceMateriale;
  double
    rispettoGiacenza;
  for(i=0;i<nomiGiacenza.length;i++){
    indiceMateriale=
    ricercaBinaria(nomiGiacenza,nomiMagazzino[i],0,nomiGiacenza.length);
    if(indiceMateriale!=-1){
      rispettoGiacenza=valoriMagazzino[i]-valoriGiacenza[indiceMateriale];
      if (rispettoGiacenza<0)
        Scrittore.video.println(nomiMagazzino[i]+" giacenza "+valoriMagazzino[i]
          +" minimo "+valoriGiacenza[indiceMateriale]+" CARENZA MAGAZZINO");
      else
        Scrittore.video.println(nomiMagazzino[i]+" giacenza "+valoriMagazzino[i]
          +" minimo "+valoriGiacenza[indiceMateriale]+" SURPLUS MAGAZZINO");
  }
  else
    Scrittore.video.println("Errata corrispondenza magazzino archivio giacenze per "
      +nomiMagazzino[i]);
  }
 return;
 }
}
```

```
----------------------
1* Visualizzazione dati
2* Ordinamento magazzino per giacenza
3* Rispetto giacenze
0* USCITA
----------------------
1
Magazzino Unificato
Rame 100.0 ; Zinco 200.0 ; Plastica 150.0 ; Ferro 245.0 ; Acciaio 185.0 ; Legno
100.0 ; Alluminio 200.0 ; Pittura 150.0 ; Vetro 245.0 ;
-------------------------
Giacenze Minime
Legno 100.0 ; Alluminio 100.0 ; Pittura 250.0 ; Vetro 250.0 ; Rame 80.0 ; Zinco
100.0 ; Plastica 130.0 ; Ferro 240.0 ; Acciaio 120.0 ;
```

```
----------------------
1* Visualizzazione dati
2* Ordinamento magazzino per giacenza
3* Rispetto giacenze
0* USCITA
----------------------
3
Ferro giacenza 245.0 minimo 240.0 SURPLUS MAGAZZINO
Vetro giacenza 245.0 minimo 250.0 CARENZA MAGAZZINO
Alluminio giacenza 200.0 minimo 100.0 SURPLUS MAGAZZINO
Zinco giacenza 200.0 minimo 100.0 SURPLUS MAGAZZINO
Acciaio giacenza 185.0 minimo 120.0 SURPLUS MAGAZZINO
Pittura giacenza 150.0 minimo 250.0 CARENZA MAGAZZINO
Plastica giacenza 150.0 minimo 130.0 SURPLUS MAGAZZINO
Legno giacenza 100.0 minimo 100.0 SURPLUS MAGAZZINO
Rame giacenza 100.0 minimo 80.0 SURPLUS MAGAZZINO
```

Commento al listato

Il programma fa leva su due coppie di vettori principali. La prima è la coppia nomi materie - giacenze rilevate, la seconda è la coppia nomi materie - giacenze minime. Nelle prima parte si procede all'inizializzazione di questi vettori nel seguente modo:

si determina il numero di dati memorizzati sui files;
si alloca lo spazio degli array in base a questo numero;
si trasferisce il contenuto dei files negli array, unendo le informazioni che provengono separatamente dai magazzini.

visualizzazione

```
public static void visualizzazione(double valori[], String nomi[])
```

In questo metodo di visualizzazione, viene sfruttato il fatto che i valori da stampare in entrambi i casi siano coppie di vettori del tipo stringa e numero. Tramite le due chiamate:

```
visualizzazione(giacenzeMagazzino,nomiMaterieMagazzino);
visualizzazione(giacenzeMinime,nomiMaterieGiacenza);
```

vengono di volta in volta passati i riferimenti delle coppie di vettori interessati alla visualizzazione.Le istruzioni all'interno del metodo fanno riferimento ai parametri generici valori e nomi. Nella prima chiamata, l'array valori viene inizializzato con il riferimento dell'array giacenzeMagazzino e l'array nomi viene inizializzato con il riferimento dell'array nomiMaterieMagazzino. Nella seconda chiamata, l'array valori viene inizializzato con il riferimento dell'array giacenzeMinime e l'array nomi viene inizializzato con il riferimento dell'array nomiMaterieGiacenza.

ordinamentoValori

```
public static void ordinamentoValori(double valori[], String nomi[])
```

Come il metodo di visualizzazione anche questo metodo elabora i vettori valori e nomi. Durante l'algoritmo, essendo i due array nomi e valori in stretta relazione posizionale fra loro, è necessario per rispettare questo legame scambiare sia i valori che i nomi.

ordinamentoNomi

```
public static void ordinamentoNomi(double valori[], String nomi[])
```

A differenza del metodo ordinamentoValori il vettore di riferimento per l'ordinamento è quello contenente i nomi. Viene utilizzato il metodo compareTo del tipo String per determinare una relazione di ordinamento fra gli elementi.

ricercaBinaria

```
public static int ricercaBinaria(String nomi[], String nomeDaCercare, int basso , int alto)
```

Il metodo ricerca binaria è realizzato in modo ricorsivo con una doppia possibilità di uscita:

nel caso in cui l'elemento venga trovato viene restituita la posizione di riferimento;
nel caso in cui l'indice superiore e inferiore si invertano l'elemento non è stato trovato e viene restituito -1.

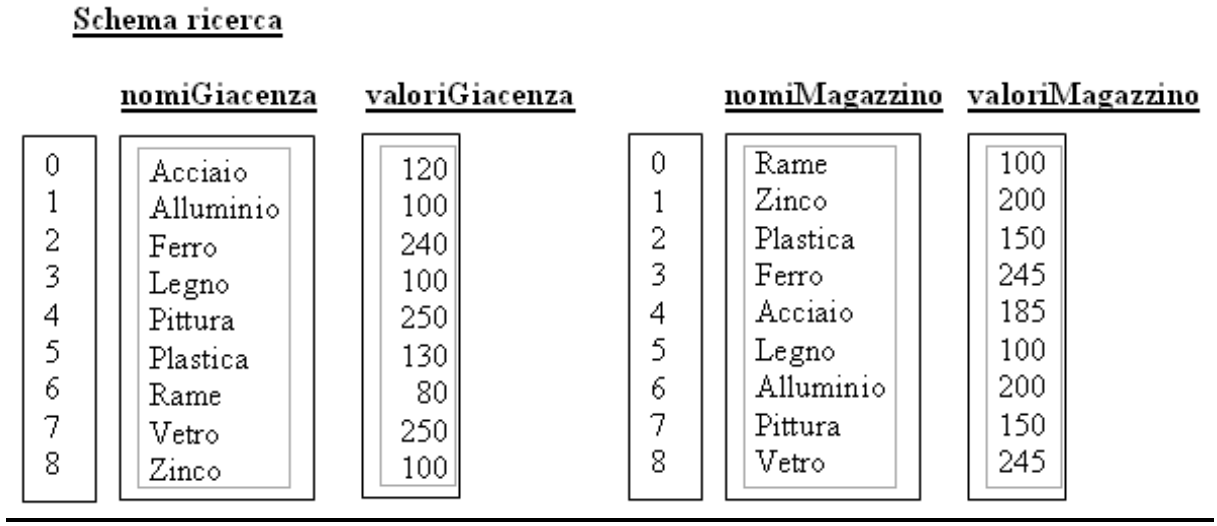
Ad ogni passaggio viene dimezzato tramite i parametri alle chiamate ricorsive il range degli elementi da considerare fino ad attenere una delle due condizioni d'uscita. Prima della chiamata del metodo, nella classe principale viene richiamato l'ordinamento, in quanto i dati ordinati sono un prerequisito della ricerca binaria.

rispettoGiacenze

```
public static void rispettoGiacenze(double valoriMagazzino[],double valoriGiacenza[],
   String nomiMagazzino[], String nomiGiacenza[])
```

Il metodo rispettoGiacenze verifica lo stato della giacenza attuale rispetto a quella minima richiesta in questo modo: per ogni materia del vettore nomiMagazzino viene fatta la ricerca binaria per inviduare la posizione nel vettore nomiGiacenza. Essendo i vettori valoriMagazzino e valoriGiacenza in relazione posizionale con quelli che riportano i loro nomi, una volta inviduati correttamente gli indici, è possibile confrontare tali valori numerici e determinare lo status di carenza o surplus di magazzino rispetto alla giacenza minima.

Vediamo un esempio pratico(gli array nomiGiacenza e valoriGiacenza si presentano già ordinati in base ai nomiGiacenza per poter effettuare la ricerca binaria). Lo schema dei nostri array in memoria è il seguente:



Step dell'algoritmo

1 indiceScansione=0
2 nomiMagazzino[indiceScansione] è Rame , valoriMagazzino[indiceScansione] è 100
3 chiamo la ricerca binaria sul vettore nomiGiacenza passando come elemento da cercare nomiMagazzino[indiceScansione] (cioè Rame)
4 la ricerca binaria trova nomiMagazzino[indiceScansione] in posione 6 di nomiGiacenza viene quindi memorizzato in indiceRicerca il valore 6
5 tramite il test valoriMagazzino[indiceScansione]> valoriGiacenza[indiceRicerca] verifico il rispetto giacenza
6 in questo caso valoriMagazzino[0] pari a 100 > valoriGiacenza[6] pari a 80 quindi giacenza sufficiente
7 questo procedimento viene ripetuto per tutte le materie prime

.

Lavoro in aula / laboratorio

Il magazzino deve essere gestito nel modo più ottimale possibile, per questo motivo si vogliono rendere disponibili le seguenti due opzioni:

- Elenco Surplus Reali (la materia è in surplus se la giacenza attuale è superiore del 10% rispetto alla giacenza minima, è necessario segnalare queste situazioni per evitare un aumento dei costi di stoccaggio e deperimento merci)
- Elenco Deficit Reali (consideriamo la materia vicino alla soglia di giacenza quando il suo valore è inferiore del 10% rispetto alla giacenza minima, è necessario segnalare queste situazioni per provvedere per tempo al reintegro della giacenza minima)

6.3 Package: utilizzo del package vettori

il package vettori include una serie di algoritmi utili per la gestione dei vettori (si rimanda alla documentazione del package):

Metodi di utilità generale
public static void inserimento(int[] a, int n, int pos, int dato)
public static void eliminazione(int[] a, int n, int pos)
public static void scambioPosizione(int[] a, int p1, int p2)
public static boolean isOrdinato(int[] a, int n)
public static int[] duplica(int[] a)
public static void stampa(int[] a, int n)
public static int[] creaCasuale(int n, int valoreMax)
Algoritmi di ricerca
public static int getPosizioneMassimo(int[] a, int n)
public static int getPosizioneMinimo(int[] a, int n)
public static int ricercaSequenziale(int[] a, int n, int dato)
public static int ricercaBinaria(int[] a, int n, int dato)
Algoritmi di ordinamento
public static void inserimentoOrdinato(int[] a, int n, int dato)
public static void ordinaPerInserimento(int[] a, int n)
public static void ordinaPerSelezione(int[] a, int n)
public static void bubbleSort(int[] a, int n)
public static void mergeSort(int[] a, int primo, int ultimo)
public static void merge(int[] a, int primo, int medio, int ultimo)

Un esempio dell'utilizzo del package vettori è il seguente confronto fra metodi di ordinamento su un vettore di numeri generati casualmente:

Listato Java (TestMetodiVettori.java)

```
import vettori.*;
public class TestMetodiVettori
{
 public static void main(String[] args)
 {
 int n = 30000;
 // crea un array con n elementi e lo riempie di numeri
 // scelti casualmente fra 0 e 100
 int[] x = MetodiArray.creaCasuale(n,100);
 System.out.println("Dimensione array: " + n);
 System.out.println("Ordinato ? " + MetodiArray.isOrdinato(x,n));
 int[] a0 = MetodiArray.duplica(x);
 int[] a1 = MetodiArray.duplica(x);
 int[] a2 = MetodiArray.duplica(x);
 int[] a3 = MetodiArray.duplica(x);
 long ti,tf;
 System.out.print("Ordinamento per fusione: ");
 ti = System.currentTimeMillis();
 MetodiArray.mergeSort(a0,0,n-1);
 tf = System.currentTimeMillis();
 System.out.println("ordinato in " + (tf-ti) + "ms");
 System.out.print("Ordinamento per inserimento: ");
 ti = System.currentTimeMillis();
 MetodiArray.ordinaPerInserimento(a1,n);
 tf = System.currentTimeMillis();
 System.out.println("ordinato in " + (tf-ti) + "ms");
 System.out.print("Ordinamento per selezione: ");
 ti = System.currentTimeMillis();
```

```
 MetodiArray.ordinaPerSelezione(a2,n);
 tf = System.currentTimeMillis();
 System.out.println("ordinato in " + (tf-ti) + "ms");
 System.out.print("Bubble Sort: ");
 ti = System.currentTimeMillis();
 MetodiArray.bubbleSort(a3,n);
 tf = System.currentTimeMillis();
 System.out.println("ordinato in " + (tf-ti) + "ms");
 }
}
```

```
Dimensione array: 30000
Ordinato ? false
Ordinamento per fusione: ordinato in 20ms
Ordinamento per inserimento: ordinato in 1242ms
Ordinamento per selezione: ordinato in 1572ms
Bubble Sort: ordinato in 4677ms
Press any key to continue...
```

6.4 Package: creazione del package formule

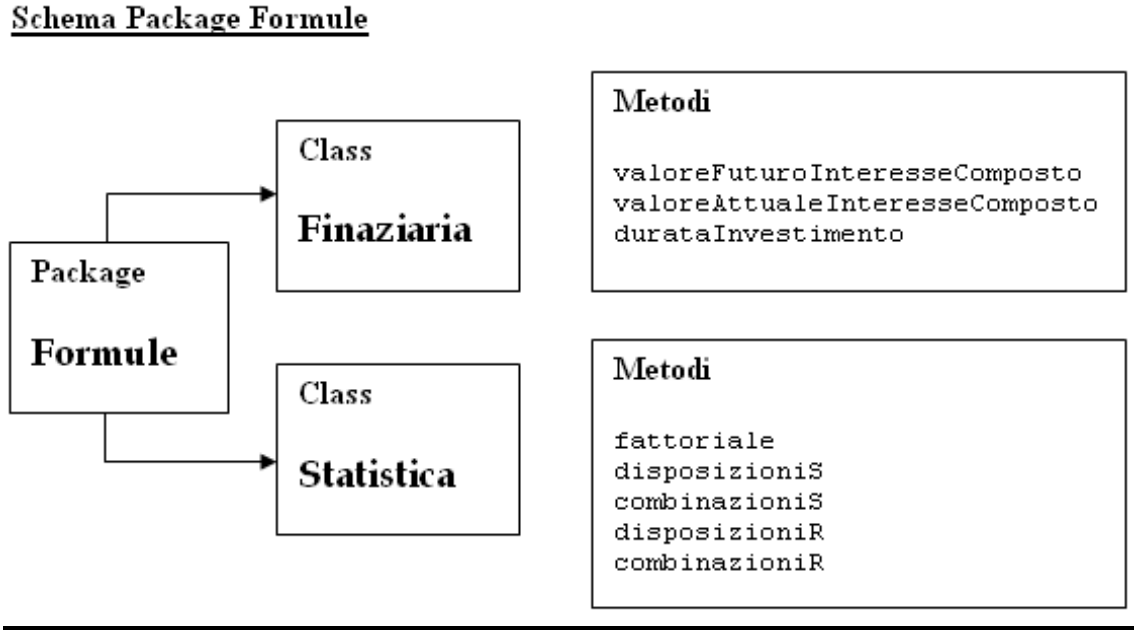
Un Package può essere considerato come un insieme di classi (e non solo) raggruppate tra di loro in modo logico. Per poter dichiarare che una classe fa parte di un certo package occorre inserire come prima riga di comando del file java l'istruzione:

package nome_package

E' necessario che i files compilati contenti il bytecode siano memorizzati in una cartella che ha lo stesso nome del package.

Vediamo praticamente la costruzione di un package che può esserci utile. Il package da realizzare vuole mettere a disposizione delle classi che contengono metodi per effettuare operazioni di tipo finanziario e statistico. Il nostro package può essere chiamato formule in modo da poter logicamente includere sia classi di tipo finanziario che statistico.

La struttura può essere rappresentata come segue:



La classe **Finanziaria** all'interno del package formule sarà la seguente:

Listato Java (Finanziaria.java)

```
package formule;
public abstract class Finanziaria{
  private Finanziaria(){}
  public static double valoreFuturoInteresseComposto(double va,double r,double m, int t){
    return va*Math.pow((1+r/m),m*t);
  }
  public static double valoreAttualeInteresseComposto(double vf,double r,double m, int t){
    return vf/Math.pow((1+r/m),m*t);
  }
  public static double durataInvestimento(double vf,double va,double r,double m){
    return Math.log(vf/va)/(m*Math.log(1+r/m));
  }
}
```

Mentre la classe **Statistica** all'interno del package formule sarà la seguente

Listato Java (Statistica.java)

```
package formule;
 public abstract class Statistica{
 private Statistica(){}
  public static double fattoriale(double n){
    double fatt;
    fatt=1.0;
    while (n>1.0) {
      fatt=fatt*n;
      n=n-1.0;
    }
    return fatt;
  }
 public static double disposizioniS(double n,double k){
   return (fattoriale(n)/fattoriale(n-k));
 }
 public static double combinazioniS(double n,double k){
   return (fattoriale(n)/(fattoriale(n-k)*fattoriale(k)));
 }
 public static double disposizioniR(double n,double k){
   return Math.pow(n,k);
 }
 public static double combinazioniR(double n,double k){
   return fattoriale(n+k-1)/(fattoriale(n-1)*fattoriale(k));
 }
}
```

La sottocartella di riferimento per il package deve essere raggiungibile attraverso i percorsi di default in cui è installato l'ambiente di esecuzione del Java (JRE). I percorsi di default per la ricerca delle classi nel package devono essere impostati nella variabile di sistema CLASSPATH.

L'accesso alle classi di un package (e quindi ai metodi afferenti ad esse) avviene specificando con l'istruzione import il package interessato es. *import formule.** oppure specificando il nome del package prima di classe e metodo, es *formule.Statistica.fattoriale(5)*. Possiamo provare alcuni metodi del package formule con il seguente listato:

Listato Java (TestFormule.java)

```
import java.io.*;
import unibs.eco.dmq.basicIO.*;
import formule.*;
public class TestFormule{
      public static void main(String[] args) {
        /*prova funzioni statistiche */
         Scrittore.video.println("fattoriale di 5 ="+formule.Statistica.fattoriale(5));
         Scrittore.video.println("disposizioni 6 a 3 ="+Statistica.disposizioniS(6,3));
         Scrittore.video.println("combinazioni 10 a 4 ="+Statistica.combinazioniS(10,4));
         Scrittore.video.println("va 1000 tasso 0.02 reinv annuali 1 anni 1 ="
           +Finanziaria.valoreFuturoInteresseComposto(1000,0.02,1,1));
 }
}
```

```
fattoriale di 5 120.0
disposizioni 6 a 3 120.0
combinazioni 10 a 4 210.0
va 1000 tasso 0.02 reinv annuali 1 anni 1 1020.0
Press any key to continue...
```

I package sono utlizzati principalmente per:

raggruppare classi con una determinata gerarchia (all'interno delle classi ci sono rapporti di ereditarietà, astrazione ..);

raggruppare classi logicamente collegate (es le classi che svolgono le formule finanziarie e statistiche nel package formule).

Passi per la creazione di un package:

1. progettare l'insieme delle classi logicamente collegate o con vincoli gerarchici;
2. realizzare le singole classi, specificando come prima riga di codice il riferimento al pakage di appartenenza;
3. generare i bytecode memorizzando i *.class* all'interno di una cartella con lo stesso nome del package;
4. creare la documentazione del package attraverso il javadoc in modo da inseririla in una cartella docs all'interno del package.

6.5 La documentazione

La produzione di package riutilizzabili e gararchicamente o logicamente più complessi prevede una cura particolare della documentazione che va distribuita insieme al package. La documentazione viene generata automaticamente dallo strumento javadoc interpretando le righe di documentazione presenti nei sorgenti .java delle classi del package.La documentazione deve essere scritta fra i simboli /** e */. All'interno possono essere utilizzate delle keywords fra cui le più importanti sono @param per specificare i parametri del metodo e @return per specificare il valore di ritorno del metodo.

Esempi di documentazione

```
/**
 * Metodo disposizioniS:
 * permette di calcolare le disposioni semplici
 * di n elementi di classe k
 * @param n numero elementi
 * @param k classe di raggruppamento
 * @return numero disposizioni semplici
 */
 public static double disposizioniS(double n,double k){
 return (fattoriale(n)/fattoriale(n-k));
 }
 /**
 * Metodo valoreFuturoInteresseComposto:
 * permette di calcolare il valore futuro di un investimento
 * VA che frutta interessi al tasso annuo r, reinvestiti
 * m volte l'anno per un periodo di t anni
 * @param VA valore attuale investimento
 * @param r tasso annuo
 * @param m reinvestimenti annuali
 * @param t anni durata investimento
 * @return valore futuro investimento
 */
 public static double valoreFuturoInteresseComposto(double va,double r,double m, int t){
 return va*Math.pow((1+r/m),m*t);
 }
```

Invocando lo strumento javadoc e passando il nome del file .java vengono create le pagine html della documentazione relativa nella directory specificata.

Esempio:

javadoc -d doc *.java

dove doc è il nome della cartella dove si creeranno i documenti

*.java prende tutti i file java che devono essere coinvolti nella documentazione

I commenti iniziano con /** e finiscono con */.
Oltre ai commenti si possono inserire altri campi
@param indica un parametro del metodo
@return indica cosa ritorna il metodo
@throws indica che il metodo può generare eccezioni
@author nome dell'autore
@version versione del software
@since versione che ha introdotto questa funzione
@deprecated metodo o var che non dovrebbero essere piu usati
@see indica un collegamento ipertestuale

Lavoro in aula / laboratorio

- Ampliare la classe statistica introducendo i metodi:

  - Media aritmetica (passato come parametro un array di valori numerici)
  - Scarto quadratico medio (passato come parametro un array di valori numerici)

- Realizzare una classe TestStatistica che permette di testare tutti i metodi della classe statistica

- Realizzare una classe TestFinanziaria che permette di testare tutti i metodi della classe finanziaria

6.6 Proposte di lavoro

Proposta Uno

Realizzare un package che permette di calcolare la tassazione Irpef e la NoTaxArea. Provvedere a generare la sua documentazione.

Proposta Due

Ampliare il package delle formule introducendo una nuova classe Aziendale che contenga il metodo per calcolare il break even point:
(viene rimandata l'analisi approfondita alle discipline di riferimento)

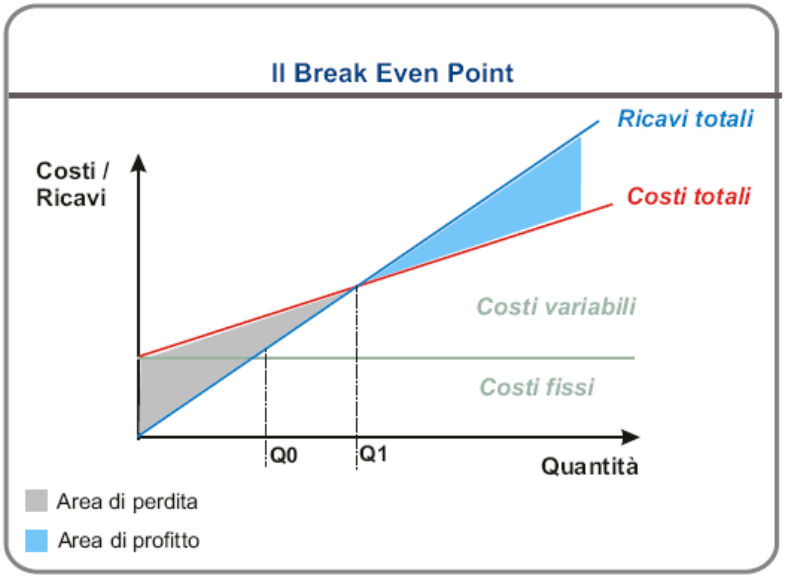
L'analisi del punto di pareggio è molto diffusa per la sua facilità di utilizzo e per il fatto che la sua costruzione non richiede una grossa mole di informazioni e può essere applicata anche in assenza di un sistema di contabilità analitica.

L'analisi del Break Even, tradotta in pratica, individua quanto l'azienda deve fatturare affinché, con una determinata struttura di costi fissi e con un certo margine di contribuzione, possa raggiungere, in un dato periodo, il pareggio di risultato. Rappresenta la soglia minima di fatturato al di sotto della quale l'azienda non può arrivare senza mettere in dubbio la sua sopravvivenza.

L'espressione che solitamente si usa per indicare il punto di pareggio è:

$$\frac{CF}{1 - (CV / R)}$$

Dove:
CF: Costi Fissi
CV: Costi Variabili



R: Ricavi

Il denominatore esprime il margine di contribuzione in percentuale.
Con questa formula è possibile calcolare velocemente il valore di fatturato di pareggio in funzione di ogni possibile variazione dei costi fissi e variabili.

Capitolo 7

Argomenti trattati

- Oggetti : Impiegato
- Oggetti : Veicolo
- Oggetti : Conto corrente
- Proposte di lavoro

7.1 Oggetti: Impiegato

Possiamo definire un oggetto come un insieme di dati e funzioni che ne definiscono lo stato ed il comportamento. In Java le classi sono dei modelli per gli oggetti, ovvero sono delle strutture dati utilizzate per definire le caratteristiche e le funzionalità di un determinato tipo di oggetto. I dati dell'oggetto sono gli attributi, i comportamenti sono i metodi.

Esempio Uno (Impiegato)

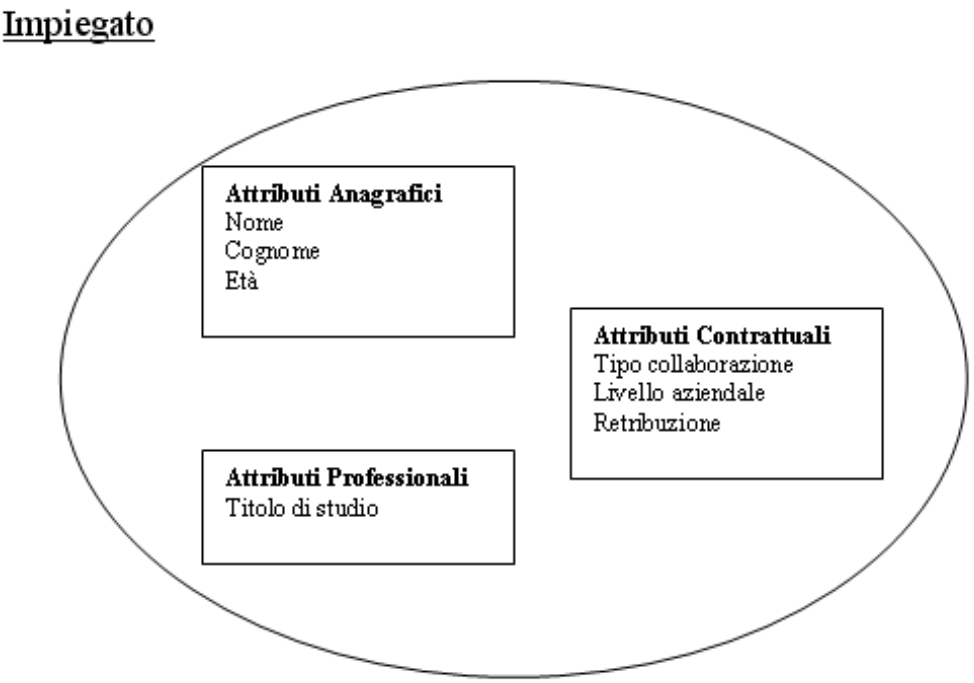
Analizziamo le caratteristiche che potrebbe avere un impiegato dal punto di vista della realtà aziendale. L'impiegato è caratterizzato da alcuni dati di tipo:

anagrafico (nome, cognome, età.);

professionale (titolo di studio);

contrattuale (
forma di collaborazione: tempo determinato, indeterminato, a progetto, ecc;
livello aziendale;
retribuzione.
)

Riepilogando, gli attributi che rappresentano l'impiegato all'interno dell'azienda, possono essere schematizzati come segue:



L'entità impiegato sarà caratterizzata dagli attributi e dai comportamenti legati a questi attributi. Saranno previsti metodi di comportamento che permettono l'accesso e la modifica degli attributi dell'oggetto impiegato. Ad esempio, posso istanziare la classe impiegato definendo l'oggetto impiegatoAziendale ed andando ad inizializzarne gli attributi o in fase di costruzione dell'oggetto o con metodi di accesso agli attributi privati. Le componenti della classe Impiegato possono essere definiti attraverso il linguaggio Java nel seguente modo:

Attributi

/* Attributi Anagrafici */

```
private String cognome, nome;
private int anni;
```

/* Attributi Contrattuali */

```
private String tipoCollaborazione;
private String livelloAziendale;
private double retribuzione;
```

/* Attributi Professionali */

```
private String titoloStudio;
```

Costruttori

Consideriamo tre tipologie di costruttore per l'oggetto Impiegato:

Costruttore Generico (senza parametri in fase di creazione)

Definisco un costruttore generico che permetta l'inizializzazione degli attributi anche quando non viene settato nessun parametro in fase di creazione dell'oggetto. (Questo può essere legato ad esigenze particolari, ad esempio se volessi valutare quale sarebbe l'impatto economico dell'assunzione di un nuovo impegato di 5 livello, potrei creare un impiegato generico e settare solamente gli attributi livelloAziendale e retribuzione)

```
Impiegato()
{
  this.cognome = "Anonimo";
  this.nome = "Utente";
  this.anni = 0;
  this.tipoCollaborazione="Nessuna";
  this.livelloAziendale="Nessuno";
  this.retribuzione=0;
  this.titoloStudio="Nessuno";
}
```

Costruttore Completo (tutti i parametri sono valorizzati in fase di creazione)

```
Impiegato( String cognome, String nome, int anni,
           String tipoCollaborazione, String livelloAziendale, double retribuzione,
           String titoloStudio)
{
  this.cognome = cognome;
  this.nome = nome;
  this.anni = anni;
  this.tipoCollaborazione=tipoCollaborazione;
  this.livelloAziendale=livelloAziendale;
  this.retribuzione=retribuzione;
  this.titoloStudio=titoloStudio;
}
```

Costruttore "Parziale" (solamente alcuni parametri sono valorizzati in fase di creazione)

Definisco un terzo costruttore che può essere utile all'azienda nella fase preliminare all'assunzione, dove vengono memorizzate solo informazioni anagrafiche ed il titolo di studio

```
 Impiegato( String cognome, String nome, int anni, String titoloStudio)
 {
   this.cognome = cognome;
   this.nome = nome;
   this.anni = anni;
   this.tipoCollaborazione="Nessuna";
   this.livelloAziendale="Nessuno";
   this.retribuzione=0;
   this.titoloStudio=titoloStudio;
 }
```

Metodi get / set

Per ogni attributo privato di cui voglio rendere possibile la modifica (*set*) o il reperimento del valore attuale (*get*) definisco la coppia di metodi get/set come segue:

/* Accesso attributo privato anni per reperire il valore*/

```
 public int getAnni()
 {
   return anni;
 }
```

/* Inizializzazione attributo privato anni */

```
 public void setAnni(int anni)
 {
   this.anni = anni;
 }
```

Altri Metodi

In questo caso non sono previsti o descritti particolari comportamenti dell'oggetto. Può essere utile rendere disponibile un metodo che permetta la visualizzazione dei dati dell'impiegato. Viene utilizzato il metodo *System.out.println* al posto di *Scrittore.video.println* per evitare di rendere dipendente la classe impiegato dal pacchetto proprietario unibs.eco.dmq.basicIO.

```
/* Visualizzazione scheda Impiegato c */
 public void stampaScheda()
 {
   System.out.println("Cognome :"+cognome);
   System.out.println("Nome : "+nome);
   System.out.println("Anni : "+anni);
   System.out.println("Collaborazione : "+tipoCollaborazione);
   System.out.println("Livello Aziendale : "+livelloAziendale);
   System.out.println("Retribuzione :"+retribuzione);
   System.out.println("Titolo studio : "+titoloStudio+"\n");
 }
```

L'utilizzo di questa classe può essere quello riportato nel listato testImpiegato:

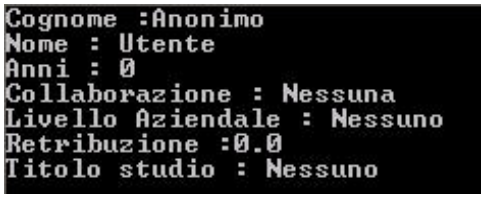
Listato Java (TestImpiegato.java)

```
import unibs.eco.dmq.basicIO.*;
class TestImpiegato
{
 public static void main(String[] args)
 {
  String
```

```
  nomeimpiegato,
  cognomeimpiegato,
  collaborazioneimpiegato,
  titoloStudioimpiegato,
  livelloimpiegato;
double
  stipendioimpiegato;
int
   anniimpiegato;
Impiegato
  impiegatoProva;
```

/* Chiamata al costruttore senza parametri */

```
impiegatoProva=new Impiegato();
impiegatoProva.stampaScheda();
```



/* Chiamata al costruttore con tutti i parametri */

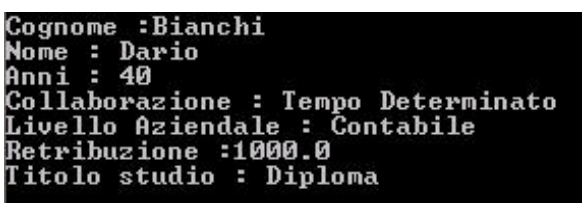
```
impiegatoProva=new Impiegato("Rossi","Mario",35,"Tempo Indeterminato",
 "Disegnatore",1200,"Laurea");
impiegatoProva.stampaScheda();
```

/* Chiamata al costruttore solo con indicazioni anagrafiche */

```
impiegatoProva=new Impiegato("Bianchi","Dario",40,"Diploma");
impiegatoProva.stampaScheda();
```

/* Accesso come inizializzazione agli attributi privati */

```
impiegatoProva.setRetribuzione(1000);
impiegatoProva.setTipoCollaborazione("Tempo Determinato");
impiegatoProva.setLivelloAziendale("Contabile");
impiegatoProva.stampaScheda();
```



/* Accesso per recuperare i valori degli attributi */

```
  nomeimpiegato=impiegatoProva.getNome();
  cognomeimpiegato=impiegatoProva.getCognome();
  titoloStudioimpiegato=impiegatoProva.getTitoloStudio();
  Scrittore.video.println(nomeimpiegato);
  Scrittore.video.println(cognomeimpiegato);
  Scrittore.video.println(titoloStudioimpiegato);
```

```
Dario
Bianchi
Diploma
```

```
 }
}
```

Lavoro in aula / laboratorio

- Implementare un programma Java che permetta di gestire un array di oggetti del tipo impiegato con le seguenti opzioni:

  1 Inserimento dati impiegati;

  2 Visualizzazione dati impiegati;

  3 Salvataggio dati degli impiegati su file;

  4 Determinazione della retribuzione aziendale media;

  5 Individuazione dell'Impiegato con retribuzione massima e visualizzazione della sua scheda;

  6 Inviduazione del range di età degli impiegati (dato dalla differenza fra l'età dell'impiegato più anziano e quello più giovane).

7.2 Oggetto: Veicolo

Si vuole realizzare un modello matematico che permette di calcolare lo spazio di frenata di un autoveicolo. Lo spazio di frenata è definito dalla seguente formula:

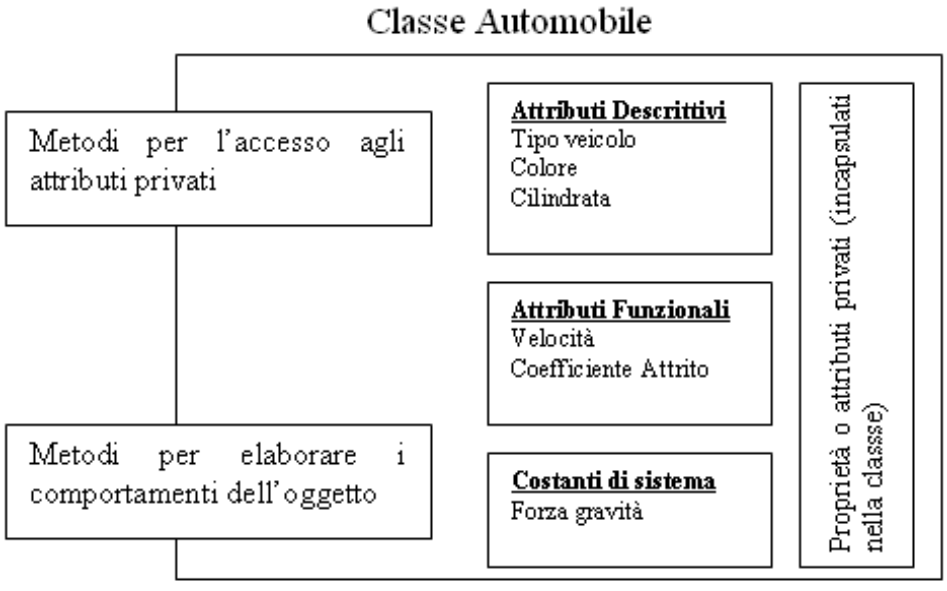
$$spazioFrenata = \frac{velocità^2}{2*g*coefficienteAttrito}$$

Un metodo per affrontare la programmazione OOP (Abbott, Booch, Lorensen) suggerisce di evidenziare all'interno della realtà che si va ad analizzare (desumendole dall'analisi descrittiva della problematica):

- le entità che potrebbero diventare una classe;
- le proprietà e/o gli attributi di questa classe;
- i comportamenti associati.

Nel nostro esempio l'entità classe potrebbe essere quella relativa al veicolo. La classe veicolo può essere caratterizzata da:

attributi descrittivi come il tipo (automobile, autocarro, ..), la cilindrata (1100,1600, ...), il colore (blu, rosso, ..);
attributi funzionali al problema da risolvere (velocità, coefficienteAttrito);
costanti di sistema (forza di gravità).



Nella definizione della classe gli attributi/proprietà della classe sono definiti come privati. L'accesso agli attributi privati per rispettare l'incapsulamento, deve essere fatto attraveso una serie di metodi set e il contenuto può essere reperito attraverso una serie di metodi get. In questo modo è il programmatore/progettista che decide quanta autonomia lasciare sulla propria classe. Il calcolo dello spazio di frenata è competenza di un metodo che applica, partendo dal contenuto attuale degli attributi, la formula riportata precedentemente.

Lavoro in aula / laboratorio

Realizzare la classe Automobile definendo:

→ Attributi e proprietà
→ Costruttori
→ Metodi di accesso (get / set)
→ Metodi di utilità (es. visualizz. dati)
→ Metodi funzionali (spazio frenata)

Realizzare la classe TestAutomobile che utilizzi oggetti Automobile

7.3 Oggetto: Conto corrente

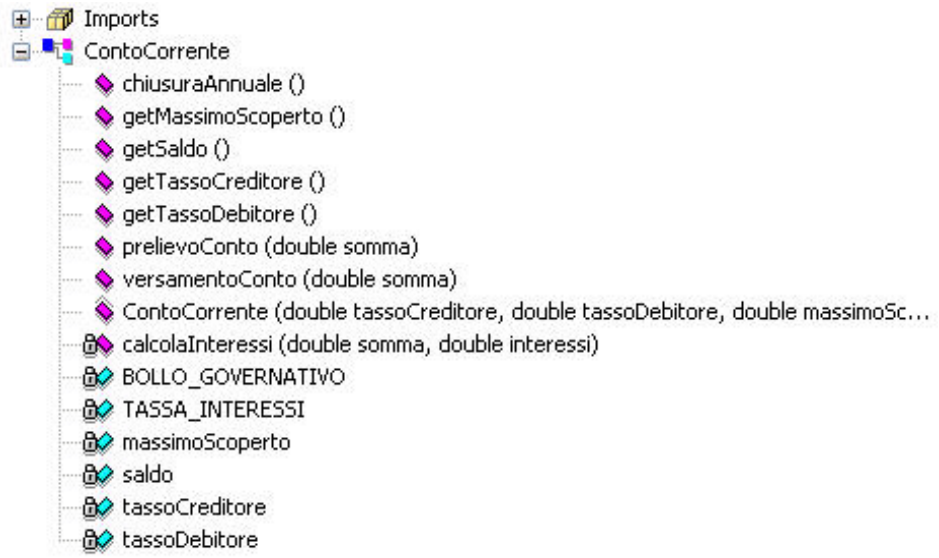
Analizziamo la realtà (semplificata) relativa ad un conto corrente. Lo stato del conto corrente è determinato dal saldo del conto stesso mentre i metodi da una serie di operazioni che modificano in qualche modo questo saldo e una serie di operazioni che permettono l'accesso agli attributi incapsulati. Nella fase di creazione del conto corrente è necessario specificare:

- giacenza iniziale
- massimo scoperto
- ammontare del tasso creditore
- ammontare del tasso debitore

Le operazioni possibili sul conto sono:

- versamento
- prelievo (se non andiamo oltre il massimo scoperto)
- visualizzazione del saldo
- operazioni di chiusura annuale

Per semplicità si limitano le operazioni di chiusura annuale alla tassazione degli interessi sul saldo attuale (non in base al sistema dei numeri creditori/debitori) applicando di volta in volta interessi creditori o debitori e il bollo governativo.



Attributi costanti

questi attributi sono dichiarari costanti in quanto non dipendono dalle peculiarità dei vari conti costruiti ma sono stabiliti a livello centrale per tutti i conti in maniera uguale

```
private static double BOLLO_GOVERNATIVO=25.56
private static double TASSA_INTERESSI=0.27
```

Attributi

```
private double tassoCreditore
private double tassoDebitore
private double massimoScoperto
private double saldo
```

Costruttore

```
ContoCorrente(double tassoCreditore, double tassoDebitore, double massimoScoperto,double saldo)
 {
  this.tassoCreditore = tassoCreditore;
  this.tassoDebitore = tassoDebitore;
  this.massimoScoperto = massimoScoperto;
  this.saldo=saldo;
 }
```

Metodi Get

```
 public double getSaldo()
 {
   return saldo;
 }
 public double getTassoCreditore()
 {
    return tassoCreditore;
 }
 public double getTassoDebitore()
 {
    return tassoDebitore;
 }
 public double getMassimoScoperto()
 {
    return massimoScoperto;
 }
```

Metodi per operazioni sul conto

Versamento

```
public void versamentoConto(double somma)
 {
   this.saldo = saldo+somma;
 }
```

Prelievo (valore booleano in uscita per controllare stato prelievo)

```
 public boolean prelievoConto(double somma)
 {
   if ((saldo-somma)>(-massimoScoperto)){
    this.saldo = saldo-somma;
    return true;
   }
   else
    return false;
 }
```

Chiusura Annuale

```
public void chiusuraAnnuale()
 {
 double interesse;
 if (saldo>0){
   interesse=calcolaInteressi(saldo,tassoCreditore);
   interesse=interesse*(double)(1-TASSA_INTERESSI);
   saldo=saldo+interesse;
 }
 else{
```

```
   interesse=calcolaInteressi(saldo,tassoDebitore);
   saldo=saldo-interesse;
 }
 saldo=saldo-BOLLO_GOVERNATIVO;
 }
```

Metodi ausiliari per gli altri metodi dell'oggetto

```
E' necessario importare Math per poter accedere al metodo abs
private double calcolaInteressi(double somma,double interessi)
 {
   return Math.abs(somma*interessi);
 }
```

Listato TestContoCorrente(TestContoCorrente.java)

```
import unibs.eco.dmq.basicIO.*;
class TestContoCorrente
{
  public static void main(String[] args)
    {
      ContoCorrente
        contoProva;
      double
        tassoCreditore,
        tassoDebitore,
        massimoScoperto,
        saldo;
      boolean
        okOperazione;
      /* Creazione conto */
      tassoCreditore=0.018;
      tassoDebitore=0.95;
      massimoScoperto=5000;
      saldo=12000;
      contoProva=new ContoCorrente(tassoCreditore,tassoDebitore,massimoScoperto,saldo);
      /*Visualizzo il saldo */
      Scrittore.video.println("Situazione Conto:");
      Scrittore.video.println("Saldo "+contoProva.getSaldo());
      Scrittore.video.println("Tasso attivo "+contoProva.getTassoCreditore());
      Scrittore.video.println("Tasso passivo "+contoProva.getTassoDebitore());
      Scrittore.video.println("Massimo scoperto "+contoProva.getMassimoScoperto());
      /* Versamento */
      Scrittore.video.println("\nVerso 2500");
      contoProva.versamentoConto(2500);
      Scrittore.video.println("Saldo "+contoProva.getSaldo());
      /* Prelievo */
      Scrittore.video.println("\nPrelievo 10000");
      okOperazione=contoProva.prelievoConto(10000);
      if (okOperazione)
        Scrittore.video.println("Saldo "+contoProva.getSaldo());
      else
        Scrittore.video.println("Operazione non effettuabile si supera il massimo scoperto");
      /* Prelievo */
      Scrittore.video.println("\nPrelievo 10000");
```

```
      okOperazione=contoProva.prelievoConto(10000);
      if (okOperazione)
        Scrittore.video.println("Saldo "+contoProva.getSaldo());
      else
        Scrittore.video.println("Operazione non effettuabile si supera il massimo scoperto");
      /* Operazioni di chiusura annuale */
      Scrittore.video.println("\nChiusura annuale");
      contoProva.chiusuraAnnuale();
      Scrittore.video.println("Saldo "+contoProva.getSaldo());
    }
}
```

```
Situazione Conto:
Saldo 12000.0
Tasso attivo 0.018
Tasso passivo 0.95
Massimo scoperto 5000.0

Verso 2500
Saldo 14500.0

Prelievo 10000
Saldo 4500.0

Prelievo 10000
Operazione non effettuabile si supera il massimo scoperto

Chiusura annuale
Saldo 4533.57
Press any key to continue...
```

Lavoro in aula / laboratorio

- Aggiungere e gestire un attributo libero/vincolato. Se il conto è vincolato non sono permessi prelievi e la banca a fine anno applica un bonus di interessi attivi dello 0.025 calcolati sul saldo finale (oltre ai normali interessi creditori)

7.4 Proposte di lavoro

Proposta Uno

Analizzare e realizzare una semplice classe magazzino che permetta di monitorare le giacenze di oggetti di tipo Prodotto. Si devono prevedere entrate e uscite di magazzino.(Es. array paralleli di Prodotti e giacenze aggiornate)

Proposta Due

Modificare la classe automobile come segue:

Inserire l'attributo km/litro (ovviemente semplificato che non tiene del tipo di traffico o percorso);

Inserire il metodo spesa del viaggio che gestisca come parametri il numero di chilometri percorsi e il prezzo del carburante.

Capitolo 8

Argomenti trattati

- Collezioni : Listino azionario

- Collezioni : Portafoglio clienti

- Mappe : Indagine clienti

8.1 Collezioni: Listino azionario

Si vuole gestire una struttura informatica che permetta di memorizzare, per eventuali elaborazioni, i dati di fine giornata delle azioni presenti sul listino di Milano. I dati minimi da memorizzare per le singole azioni sono: il codice azione (per comodità definito numerico), la descrizione, il settore di appartenenza, la quotazione di chiusura e il volume scambiato. La struttura deve presentare le seguenti caratteristiche:

non è importante l'odine di memorizzazione dei dati;

deve essere possibile l'aggiunta di un dato;

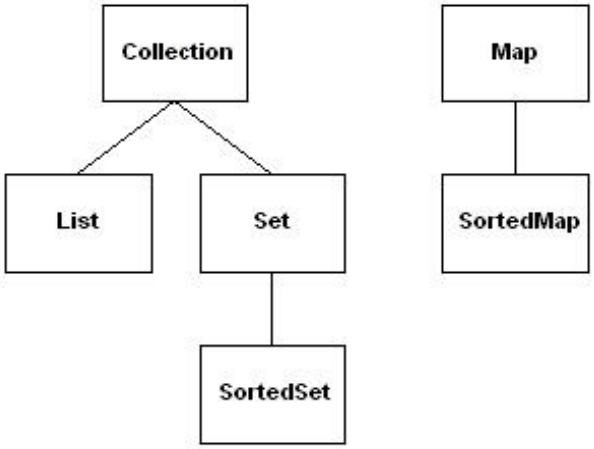
deve essere possibile la cancellazione di un dato;

deve essere possibile l'elenco dei dati memorizzati;

deve essere possibile la verifica di appartenenza di un dato all'insieme.

Riprendendo concetti di matematica, un insieme non può contenere elementi duplicati. Se un oggetto è già presente nell'insieme il tentativo di aggiungerlo deve essere ignorato come pure deve essere ignorato il tentativo di rimozione di un elemento non presente.

L'interfaccia Set estende l'interfaccia Collection, non prevede la presenza di elementi duplicati e viene implementata praticamente con la classe HashSet:



Interface	Implementation			
Set	HashSet		TreeSet	
List		ArrayList		LinkedList
Map	HashMap		TreeMap	

Elenco dei metodi della classe HashSet

Set
+add(element : Object) : boolean +addAll(collection : Collection) : boolean +clear() : void +contains(element : Object) : boolean +containsAll(collection : Collection) : boolean +equals(object : Object) : boolean +hashCode() : int +iterator() : Iterator +remove(element : Object) : boolean +removeAll(collection : Collection) : boolean +retainAll(collection : Collection) : boolean +size() : int +toArray() : Object[] +toArray(array : Object[]) : Object[]

Oggetto azione

Attributi

```
private int codice;
private String descrizione;
private double quotazione;
private long volume;
private String settore;
```

Costruttore

```
public Azione (int codice,String descrizione,double quotazione,long volume,String settore)
 {...}
```

Metodi Get/Set

Medoti per la visualizzazione del contenuto

Override toString

```
public String toString()
 {
   return codice+" "+descrizione+" "+quotazione+" "+volume+" "+settore;
 }
```

Viene ridefinito il metodo toString per avere una codifica in formato alfanumerico di tutte le informazioni contenute nell'oggetto

Metodi necessari per la collection

Override equals

```
public boolean equals(Object azioneConfronto)
 {
 Azione altraAzione=(Azione)azioneConfronto;
 return this.codice==altraAzione.codice
                  && this.descrizione.equals(altraAzione.descrizione)
                  && this.quotazione==altraAzione.quotazione
                  && this.volume==volume
                  && this.settore.equals(altraAzione.settore);
 }
```

Viene fatto l'*override* sul metodo equals della classe Object. Ogni volta che verrà effettuata un'operazione di verifica di uguaglianza fra due oggetti di tipo azione, verrà invocato il metodo personalizzato appena codificato che restituisce il valore vero se i due oggetti confrontati hanno valore uguale in tutti i campi.

Override hashCode

```
public int hashCode()
 {
  int hash1=descrizione.hashCode();
  int hash2=new Integer(codice).hashCode();
  int hash3=new Double(quotazione).hashCode();
  int hash4=new Long(volume).hashCode();
  int hash5=settore.hashCode();
  final int HAS_MOLT=50;
  int hash=HAS_MOLT*hash1+hash2+hash3+hash4+hash5;
  return hash;
 }
```

Altro ovveride viene fatto sul metodo che calcola l'hashCode che viene richiamato automaticamente nella gestione della collezione.

Collezione listino

La borsa valori di Milano può essere vista come una collezione di Azioni. Possono essere quotate nuove azioni, possono essere rimosse azioni in quotazione, si può ottenere un listato delle azioni etc etc. La costruzione della collezione può essere manuale, inserendo i valori di volta in volta oppure da file recuperendo le azioni da un file di testo.

(Esempio:Azioni.txt)

100	Generali	21.24	11290	Assicurativo
102	Fiat	5.957	80045	Auto
105	BcaCarige	2.885	268579	Bancario
108	Unicredito	3.91	600000	Bancario
110	Mediaset	8.41	14313	Editoriale
112	Beghelli	0.57	84677	Elettronico
115	Finmeccanica	0.542	235855	Elettronico
130	Gefran	3.93	2960	Elettronico

Le operazioni che devono essere permesse all'operatore sono:

inserimento manuale dei dati delle azioni (nel caso in cui non fossero recuperabili da file oppure l'azione abbia avuto comportamenti particolari come sospensioni e riammissioni)

inserimento automatico (si suppone che un altro applicativo si colleghi al database della borsa, recuperi le informazioni richieste e le salvi in formato file testuale)

rimozione manuale (l'operatore può rimuovere manualmente un'azione fornendo il codice)

visualizzazione azioni listino

Listato Java (Listino.java)

```
import java.util.*;
import unibs.eco.dmq.basicIO.*;
public class Listino{
 public static void main(String[] args) {
 int
   i,
   opzione;
 /* Allocazione insieme */
 Set
   listino=new HashSet();
 /* ------------------------------------------------*/
 do{
   opzione=menu();
   switch(opzione){
     case 1:
       aggiuntaManuale(listino);
       break;
     case 2:
       aggiuntaFile(listino);
       break;
     case 3:
       rimozione(listino);
       break;
     case 4:
       visualizza(listino);
       break;
```

```
  }
}while(opzione!=0);
}
/* Menu gestione */
public static int menu(){
  int
    opzione;
  Scrittore.video.println("---------------------");
  Scrittore.video.println("1* Aggiunta manuale azioni listino");
  Scrittore.video.println("2* Aggiunta da file azioni listino");
  Scrittore.video.println("3* Eliminazione azioni listino");
  Scrittore.video.println("4* Lista azioni listino");
  Scrittore.video.println("0* USCITA");
  Scrittore.video.println("---------------------");
  opzione=Lettore.tastiera.leggiInt();
  return opzione;
}
/* Aggiunta azioni */
public static void aggiuntaManuale(Set listino){
  Azione
    azioneListino;
  int
    codice;
  String
    descrizione;
double
    quotazione;
long
    volume;
String
    settore;
Scrittore.video.println("Codice Azione: ");
codice=Lettore.tastiera.leggiInt();
Scrittore.video.println("Nome Azione: ");
descrizione=Lettore.tastiera.leggiString();
Scrittore.video.println("Quotazione Azione: ");
quotazione=Lettore.tastiera.leggiDouble();
Scrittore.video.println("Volume Azione: ");
volume=Lettore.tastiera.leggiInt();
Scrittore.video.println("Settore Azione: ");
settore=Lettore.tastiera.leggiString();
azioneListino=new Azione(codice,descrizione,quotazione,volume,settore);
listino.add(azioneListino);
return;
}
public static void aggiuntaFile(Set listino){
Azione
  azioneListino;
int
  i;
int
  nrRighe;
int
  codice;
String
  descrizione;
double
```

```
  quotazione;
long
  volume;
String
  settore;
Lettore fileAzioni=new Lettore("azioni.txt");
nrRighe=fileAzioni.contaRighe();
i=0;
while(i<nrRighe){
  codice=fileAzioni.leggiInt();
  descrizione=fileAzioni.leggiString();
  quotazione=fileAzioni.leggiDouble();
  volume=fileAzioni.leggiInt();
  settore=fileAzioni.leggiString();
  azioneListino=new Azione(codice,descrizione,quotazione,volume,settore);
  listino.add(azioneListino);
  i++;
}
fileAzioni.chiudi();
return;
}
public static void rimozione(Set listino){
Iterator
  iterazione=listino.iterator();
Azione
  azioneListino;
int
  codice;
boolean
  cancellato;
Scrittore.video.println("Codice Azione da cancellare: ");
codice=Lettore.tastiera.leggiInt();
cancellato=false;
while (iterazione.hasNext()&&cancellato==false){
  azioneListino=(Azione)iterazione.next();
  if(codice==azioneListino.getCodice()){
    cancellato=true;
    listino.remove(azioneListino);
  }
}
if (cancellato)
  Scrittore.video.println("Cancellazione effettuata");
else
  Scrittore.video.println("Nessuna azione con cod: "+codice);
return;
}
public static void visualizza(Set listino){
  Iterator
    iterazione=listino.iterator();
  Azione
    azioneListino;
  while (iterazione.hasNext()){
    azioneListino=(Azione)iterazione.next();
    Scrittore.video.println(azioneListino);
  }
return;
}
```

```
}
```

Note:
metodo aggiuntaManuale
dopo aver costruito l'oggetto Azione viene semplicemente inserito nel listino attraverso il metodo listino.add. La locazione dove viene inserito e il controllo sul fatto che sia già presente nel listino vengono gestiti automaticamente e si basano sugli override dei metodi equals e hashCode

```
azioneListino=new Azione(codice,descrizione,quotazione,volume,settore)
listino.add(azioneListino)
```

metodo aggiunta
il procedimento di gestione dell'inserimento è lo stesso del metodo aggiuntaManuale, gli oggetti vengono costruiti prendendo i dati da un file di testo

metodo rimozione
viene utilizzato l'oggetto

```
Iterator iterazione=listino.iterator()
```

che permette di iterare la scansione degli elementi della collezione di nome listino. La condizione di esecuzione specificata nel ciclo while è la presenza contemporanea di ulteriori elementi da scansionare (iterazione.hasNext()) e il non aver trovato e cancellato l'elemento da cancellare (cancellato == false). L'assegnamento all'oggetto azioneListino da parte dell'iteratore necessita di un casting. Una volta individuato l'elemento da cancellare viene invocato il metodo remove.

```
while(iterazione.hasNext()&&cancellato==false){
  azioneListino=(Azione)iterazione.next();
  if(codice==azioneListino.getCodice()){
    cancellato=true;
    listino.remove(azioneListino);
  }
}
```

metodo visualizza
viene utlizzata la stessa procedura di scansione utilizzata per il metodo di rimozione. La visualizzazione con l'istruzione:

```
Scrittore.video.println(azioneListino)
```

è possibile in quanto è stato effettuato l'override sul metodo toString

Lavoro in aula / laboratorio

- Implementare (sfruttando il metodo contains della classe HashSet) la verifica dell'esistenza dei dati relativi ad un'azione nel listino milanese
- Determinare il volume medio della contrattazioni delle azioni del listino
- Determinare il numero delle azioni contrattate sul listino milanese (sfruttare il metodo size della classe HashSet)
- Inserire un controllo che segnali se la collezione è vuota (usare il metodo boolean isEmpty())
- Inserire un metodo che permetta di visualizzare le azioni appartenenti ad un settore azionario scelto dall'utente

8.2 Collezioni: Portafoglio clienti

La banca XY S.p.a. consente ai propri clienti la gestione delle intermediazioni in borsa on-line. Ogni giorno il sistema automaticamente memorizza per ogni cliente su un file di testo le informazioni della giornata relative alle azioni intermediate. La banca vuole:

A visionare i dati relativi alle azioni intermediate dal cliente
B visionare le azioni che sono state oggetto di contrattazione per tutti i clienti
C visionare le azioni intermediate nella giornata

E' possibile realizzare la gestione attraverso l'oggetto azione definito precedentemente, gli insiemi e le proprietà degli insiemi. Supponendo di avere i seguenti file di log relativi alle azioni intermediate:

PortafoglioUno

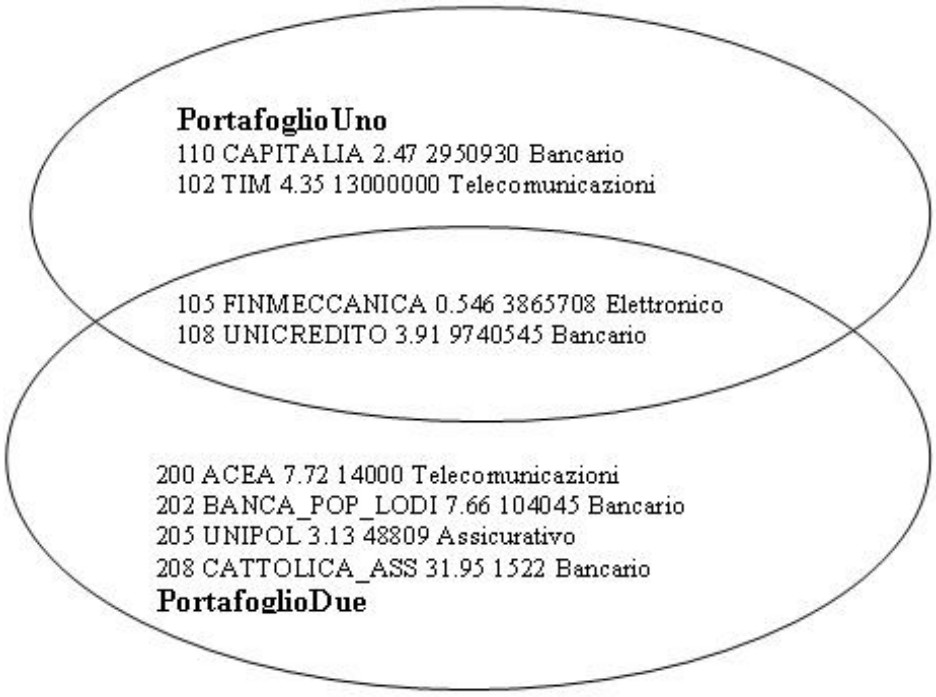
102	TIM	4.35	13000000	Telecomunicazioni
105	FINMECCANICA	0.546	3865708	Elettronico
108	UNICREDITO	3.91	9740545	Bancario
110	CAPITALIA	2.47	2950930	Bancario

PortafoglioDue

200	ACEA	7.72	14000	Telecomunicazioni
202	BANCA_POP_LODI	7.66	104045	Bancario
205	UNIPOL	3.13	48809	Assicurativo
208	CATTOLICA_ASS	31.95	1522	Bancario
105	FINMECCANICA	0.546	3865708	Elettronico
108	UNICREDITO	3.91	9740545	Bancario

A - Il punto A viene risolto caricando i files relativi ai logs in collezioni di tipo hashSet

B - Il punto B non è altro che un'operazione di intersezione insiemistica fra i portafogli che permette di ottenere le azioni comuni a tutti i clienti e quindi intermediate da tutti



C - Il punto C è un'operazione di unione insiemistica fra i portafogli dove il portafoglioUnione contiene tutte le azioni intermediate senza duplicazioni.

Le operazioni relative ai punti B, C vengono realizzate tramite i seguenti metodi:

addAll → **Unione**
retainAll → **Intersezione**

Listato Java (OperazioniInsiemi.java)

```
import java.util.*;
import unibs.eco.dmq.basicIO.*;
public class OperazioniInsiemi{
  public static void main(String[] args) {
    int
      i,
      opzione;
    /* Allocazione insieme */
    Set
      portafoglioUno=new HashSet();
    Set
      portafoglioDue=new HashSet();
    /* ------------------------------------------------*/
    do{
      opzione=menu();
      switch(opzione){
        case 1:
          caricamentoPortafoglio(portafoglioUno,"portafoglioUno.txt");
          caricamentoPortafoglio(portafoglioDue,"portafoglioDue.txt");
          break;
        case 2:
          visualizza(portafoglioUno);
          break;
        case 3:
          visualizza(portafoglioDue);
          break;
        case 4:
          intersezionePortafoglio(portafoglioUno,portafoglioDue);
          break;
        case 5:
          unionePortafoglio(portafoglioUno,portafoglioDue);
          break;
      }
    }while(opzione!=0);
  }
  /* Menu gestione */
  public static int menu(){
    int
      opzione;
      Scrittore.video.println("---------------------");
      Scrittore.video.println("1* Caricamento portafoglio");
      Scrittore.video.println("2* Lista portafoglio uno");
      Scrittore.video.println("3* Lista portafoglio due");
      Scrittore.video.println("4* Lista interserzione portafogli");
      Scrittore.video.println("5* Lista unione portafogli");
      Scrittore.video.println("0* USCITA");
      Scrittore.video.println("---------------------");
      opzione=Lettore.tastiera.leggiInt();
      return opzione;
  }
  /* Aggiunta azioni */
  public static void caricamentoPortafoglio(Set listino,String nomeFile){
    Azione
      azionePortafoglio;
    int
```

```
      i,
      nrRighe;
    int
      codice
  String
     descrizione;
  double
     quotazione;
  long
     volumi;
  String
     settore;
  Lettore
     fileAzioni=new Lettore(nomeFile);
  nrRighe=fileAzioni.contaRighe();
  i=0;
  while(i<nrRighe){
    codice=fileAzioni.leggiInt();
    descrizione=fileAzioni.leggiString();
    quotazione=fileAzioni.leggiDouble();
    volumi=fileAzioni.leggiInt();
    settore=fileAzioni.leggiString();
    azionePortafoglio=new Azione(codice,descrizione,quotazione,volumi,settore);
    listino.add(azionePortafoglio);
    i++;
  }
  fileAzioni.chiudi();
  return;
 }
 public static void intersezionePortafoglio(Set portafoglioUno, Set portafoglioDue){
  Set
    portafoglioIntersezione=new HashSet();
  portafoglioIntersezione.addAll(portafoglioUno);
  portafoglioIntersezione.retainAll(portafoglioDue);
  visualizza(portafoglioIntersezione);
  return;
 }
 public static void unionePortafoglio(Set portafoglioUno, Set portafoglioDue){
  Set
    portafoglioUnione=new HashSet();
  portafoglioUnione.addAll(portafoglioUno);
  portafoglioUnione.addAll(portafoglioDue);
  visualizza(portafoglioUnione);
  return;
 }
 public static void visualizza(Set listino){
  Iterator
    iterazione=listino.iterator();
  Azione
    azionePortafoglio;
  while (iterazione.hasNext()){
    azionePortafoglio=(Azione)iterazione.next();
    Scrittore.video.println(azionePortafoglio);
  }
  return;
 }
}
```

Unione

```
1* Caricamento portafoglio
2* Lista portafoglio uno
3* Lista portafoglio due
4* Lista interserzione portafogli
5* Lista unione portafogli
0* USCITA
----------------------
5
105  FINMECCANICA  0.546  3865708  Elettronico
205  UNIPOL  3.13  48809  Assicurativo
110  CAPITALIA  2.47  2950930  Bancario
200  ACEA  7.72  14000  Telecomunicazioni
102  TIM  4.35  13000000  Telecomunicazioni
208  CATTOLICA_ASS  31.95  1522  Bancario
108  UNICREDITO  3.91  9740545  Bancario
202  BANCA_POP_LODI  7.66  104045  Bancario
----------------------
```

Intersezione

```
----------------------
1* Caricamento portafoglio
2* Lista portafoglio uno
3* Lista portafoglio due
4* Lista interserzione portafogli
5* Lista unione portafogli
0* USCITA
----------------------
4
105  FINMECCANICA  0.546  3865708  Elettronico
108  UNICREDITO  3.91  9740545  Bancario
```

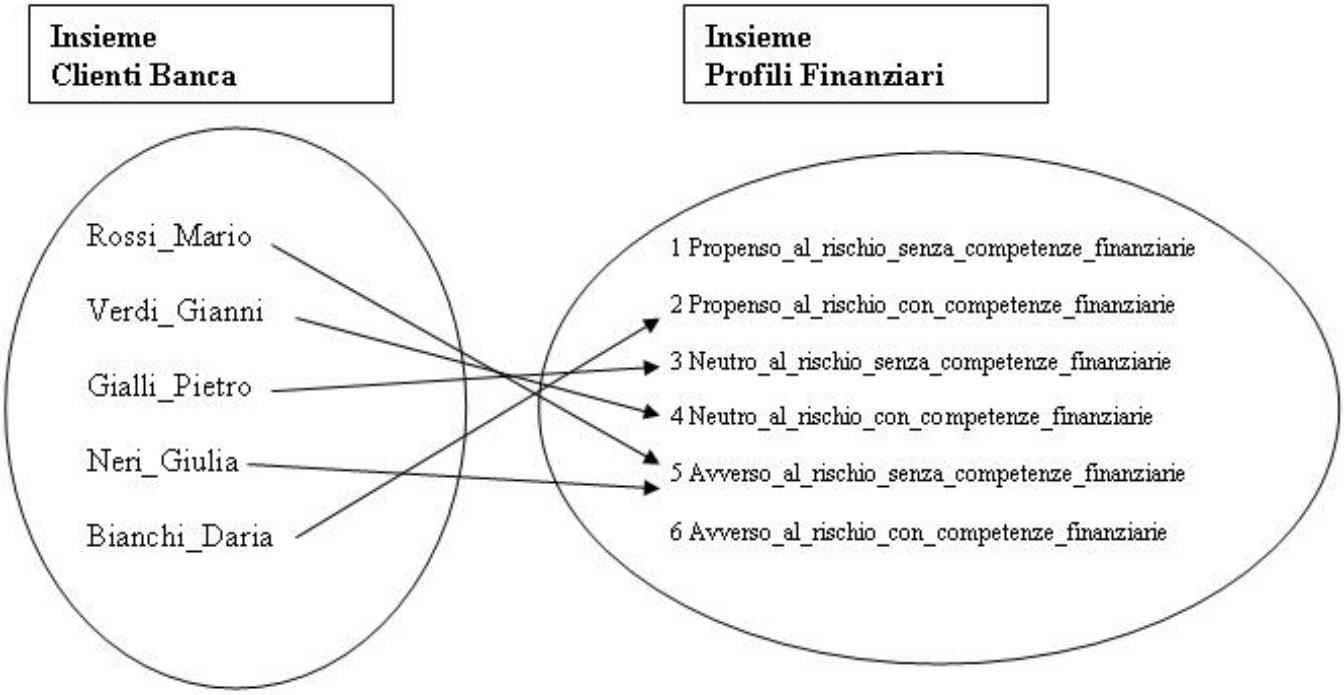
Lavoro in aula / laboratorio

- Implementare il metodo unionePortafoglio che permette la visualizzazione dell'unione insiemistica dei portafogli senza utilizzare il metodo addAll
- Implementare il metodo intersezionePortafoglio che permette la visualizzazione dell'intersezione insiemistica dei portafogli senza utilizzare il metodo retainAll
- Indicare il settore azionario più intermediato dai clienti

8.3 Mappe: Indagine clienti

Il centro finanziario della Banca XY decide di lanciare dei nuovi fondi comuni di investimento. Si vuole fornire ai direttori di ciascuna delle filiali (che hanno una conoscenza maggiore della clientela locale) uno strumento che permetta di associare ad ogni cliente della banca un profilo di rischio in modo che sia possibile inviare del materiale pubblicitario adeguato e si possano fare statistiche sulla tipologia di clientela.

La Mappa è un tipo di dati che memorizza associazioni tra chiavi e valori. Considerando i clienti come le chiavi della mappa ed i valori come i profili di rischio posso associare la mappa schematizzata sotto:



Ogni chiave nella mappa ha un valore, ma un valore può essere associato a più chiavi (Es. il profilo con codice 5 è comune sia a Rossi_Mario che a Neri_Giulia)

Per l'insieme relativo ai profili finanziari, viene creato un Oggetto profilo che costituisce la base per una collezione HasSet, mentre per le chiavi vengono utilizzare delle stringhe semplici riferite ai singoli clienti.

Oggetto Profilo (Profilo.java)

Attributi

```
private int codice;
private String descrizione;
```

Costruttore

```
public Profilo (int codice,String descrizione)
```

Metodi Get/Set

Metodi necessari per la collection

Override equals

```
public boolean equals(Object profiloConfronto)
{
 Profilo altroProfilo=(Profilo)profiloConfronto;
 return this.codice==altroProfilo.codice
                     &&
                     this.descrizione.equals(altroProfilo.descrizione);
}
```

Override hashCode

```
 public int hashCode()
 {
  int hash1=descrizione.hashCode();
  int hash2=new Integer(codice).hashCode();
  final int HAS_MOLT=20;
  int hash=HAS_MOLT*hash1+hash2;
  return hash;
 }
```

Override toString

```
 public String toString()
 {
 return codice+" "+descrizione;
 }
}
```

Supponiamo che per uniformare il lavoro delle filiali la sede centrale predisponga un file con una serie di profili finanziari standard di questo tipo:

Codice	**Profilo**
1	Propenso_al_rischio_senza_competenze_finanziarie
2	Propenso_al_rischio_con_competenze_finanziarie
3	Neutro_al_rischio_senza_competenze_finanziarie
4	Neutro_al_rischio_con_competenze_finanziarie
5	Avverso_al_rischio_senza_competenze_finanziarie
6	Avverso_al_rischio_con_competenze_finanziarie

che deve essere la base per l'HasSet profili di ogni filiale (la quale può comunque definire ed aggiungere dei profili propri in relazione alle esigenze)

Listato Java (StatisticheClienti.java)

```
import java.io.*;
import java.util.*;
import unibs.eco.dmq.basicIO.*;
public class StatisticheClienti{
 public static void main(String[] args) {
   int
     i,
     opzione;
 Set profiliContribuente=new HashSet();
 Map mappaClientiProfili=new HashMap();
 caricamentoProfili(profiliContribuente);
 do{
   opzione=menu();
   switch(opzione){
     case 1:
       aggiuntaMappa(mappaClientiProfili,profiliContribuente);
       break;
     case 2:
       listaMappa(mappaClientiProfili);
       break;
   }
 }while(opzione!=0);
 }
 /* Menu gestione */
```

```
 public static int menu(){
   int
     opzione;
   Scrittore.video.println("----------------------");
   Scrittore.video.println("1* Inserimento dati");
   Scrittore.video.println("2* Lista dati");
   Scrittore.video.println("0* USCITA");
   Scrittore.video.println("----------------------");
   opzione=Lettore.tastiera.leggiInt();
   return opzione;
 }
 /* Aggiunta dati nella mappa */
 public static void aggiuntaMappa(Map mappaClientiProfili,Set profiliContribuente){
 Profilo
   profiloRischio;
 String
   nomeContribuente;
 int
   codiceProfilo;
 Scrittore.video.println("Inserire nome del cliente");
/* leggi riga permette di avere spazi fra stringhe di input */
 nomeContribuente=(String)Lettore.tastiera.leggiRiga();
 Scrittore.video.println("Inserire profilo di rischio fra quelli disponibili");
 Scrittore.video.println(" ");
 visualizzaProfili(profiliContribuente);
 codiceProfilo=Lettore.tastiera.leggiInt();
 profiloRischio=selezionaProfilo(profiliContribuente,codiceProfilo);
 if (profiloRischio!=null)
   mappaClientiProfili.put(nomeContribuente,profiloRischio);
 else
   Scrittore.video.println("Profilo non gestito");
 return;
 }
 public static void listaMappa(Map mappaClientiProfili){
   String
     nomeContribuente;
   Profilo
     profiloContribuente;
   Set
     keySet=mappaClientiProfili.keySet();
   Iterator
     iterazione=keySet.iterator();
   while (iterazione.hasNext()){
     nomeContribuente=(String)iterazione.next();
     profiloContribuente=(Profilo)mappaClientiProfili.get(nomeContribuente);
     Scrittore.video.println(nomeContribuente+" "+profiloContribuente);
   }
 return;
 }
 public static void visualizzaProfili(Set profiliContribuente){
   Iterator
     iterazione=profiliContribuente.iterator();
   Profilo
     profiloContribuente;
   while (iterazione.hasNext()){
     profiloContribuente=(Profilo)iterazione.next();
     Scrittore.video.println(profiloContribuente);
```

```
    }
    return;
  }
  public static Profilo selezionaProfilo(Set profiliContribuente,int codiceSelezione){
    Iterator
      iterazione=profiliContribuente.iterator();
    Profilo profiloContribuente,
      profiloSelezionato;
    int
      codice;
    profiloSelezionato=null;
    while (iterazione.hasNext()&&profiloSelezionato==null){
      profiloContribuente=(Profilo)iterazione.next();
      codice=profiloContribuente.getCodice();
      if (codice==codiceSelezione)
        profiloSelezionato=profiloContribuente;
    }
    return profiloSelezionato;
  }
  public static void caricamentoProfili(Set profiliContribuente){
    Profilo profiloContribuente;
    int i,
    nrRighe;
    int codice;
    String descrizione;
    Lettore fileProfili=new Lettore("tabellaProfili.txt");
    nrRighe=fileProfili.contaRighe();
    i=0;
    while(i<nrRighe){
      codice=fileProfili.leggiInt();
      descrizione=fileProfili.leggiString();
      profiloContribuente=new Profilo(codice,descrizione);
      profiliContribuente.add(profiloContribuente);
      i++;
    }
    fileProfili.chiudi();
    return;
  }
}
```

Commento ai metodi principali

caricamentoProfili

Vengono caricati da file i profili di rischio principali generando un HasSet di oggetti Profilo

selezionaProfilo

Viene individuato l'oggetto Profilo corrispondente ad un determinato codice profilo selezionato dell'utente.

aggiuntaMappa

Vengono aggiunte (se nuove) o aggiornate (se esistenti) le associazioni chiavi-valori

listaMappa

Con il metodo keySet viene restituito l'insieme delle chiavi che può essere percorso con un iterator. Sfruttando il metodo get della mappa posso accedere al valore che corrisponde alla chiave

Lavoro in aula / laboratorio

- Implementare il metodo per permettere l'aggiunta o la cancellazione di nuovi profili di rischio
- Implementare un metodo per permettere al direttore della filiale di salvare su file di testo la associazioni clienti-valori a fine giornata
- Implementare il metodo per permettere una visualizzazione con i dati raggruppati per profilo di rischi (tutti i clienti associati al profilo di rischio1, tutti i clienti associati al profilo di rischio 2, ecc...)

8.4 Proposte di lavoro

Proposta Uno

L'azienda tessile XY distribuisce una serie di prodotti caratterizzati da:

- codifica prodotto
- descrizione materiale
- costo al metro

(Es. 120 Lana 4)
L'azienda grazie alla diffusione della sua rete commerciale riesce a diffondersi in: Europa, Asia, America.
Realizzare in linguaggio Java un'applicazione che permetta: la gestione dell'oggetto tessuto e la gestione degli insiemi Europa, Asia, America. L'Azienda vuole inoltre conoscere i prodotti che riesce a vendere in tutte le piazze (intersezione) e i prodotti che produce ed hanno mercato (unione)

Proposta Due

La direzione della ditta XY chiede all'ufficio risorse umane di fornire una struttura che associ il livello all'interno dell'azienda (operaio, impiegato, quadro ..) alle persone. Realizzare un'applicazione Java che permetta questa associazione.

Capitolo 9

Argomenti trattati

- Oggetti: aggregazioni

- Proposte di lavoro

9.1 Oggetti: aggregazioni

Esempio fattura di vendita

Si vuole analizzare la realtà relativa a una fattura di vendita per una serie di prodotti all'ingrosso schematizzata nella figura che segue:

Venditore

Magazzini Vari S.r.l.
Via Bianchi 45 - 25100
Brescia

Compratore

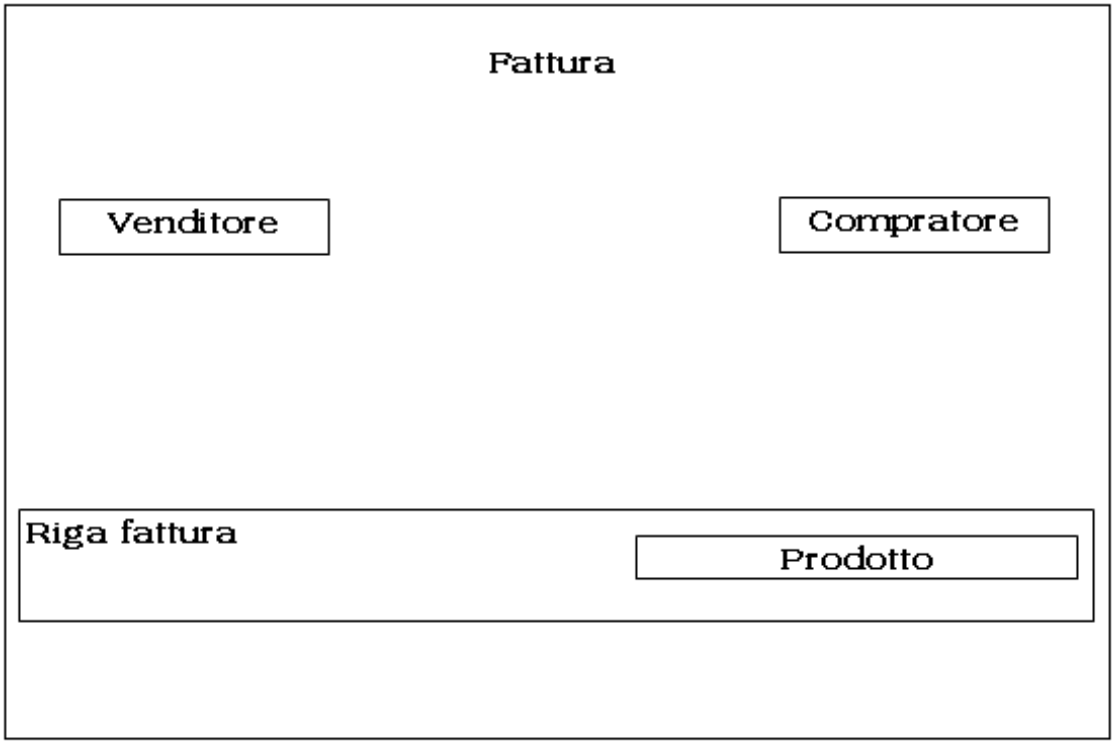
IperMio s.r.l.
Via Rossi 36 -25100
Brescia

FatturaNr
01

Qta	Descrizione Merce	Prezzo unitario	Importo	Sconto	Netto Scontato	Iva
1,00	Ghisa	11,40	11,40	0,00	11,40	10,00
2,00	Ferro	13,00	26,00	50,00	13,00	20,00

iva10	1,14		
iva20	2,60		
totIva	**3,74**	**totImpo**	**24,40**
		totFattura	**28,14**

Analizzando le varie componenti possiamo considerare l'oggetto fattura così composto: sono presenti per ogni fattura i dati commerciali relativi al venditore ed al compratore, sono presenti delle righe relative ai singoli movimenti commerciali con informazioni relative alle quantità oggetto dello scambio, descrizioni del prodotto, prezzo unitario, aliquota iva e sconto di riga. Schematizzando:



Oggetto Prodotto.

Le informazioni necessarie per il prodotto sono:

1 codice prodotto;
2 descrizione prodotto;
3 prezzo unitario;
4 aliquota iva associata

I comportamenti del prodotto sono legati alla definizione di un costruttore che permetta il passaggio di tutti i parametri oppure di un costrutture che passato un parametro, ricerchi all'interno di un file-archivio i parametri relativi (per semplicità espositiva useremo per la ricerca il campo descrizione anche se la ricerca, nel caso in cui il codice prodotto sia definito come chiave univoca di identificazione del prodotto, andrebbe fatta sul codice prodotto). In questa sede non ci occupiamo dei controlli relativi all'esistenza del prodotto sul file (la non esistenza viene segnalata attraverso il settaggio del codice prodotto a -1 in modo da permettere al programmatore eventuali controlli). Devono poi essere resi disponibili i metodi set/get per i vari attributi.

Attributi

```
int codice
String descrizione
double prezzoUnitario
int aliquota
```

Costruttori
Costruttore standard

```
Prodotto (int codice,String descrizione,double prezzoUnitario,int aliquota)
```

Costruttore da file

```
Prodotto (String descrizione)
 {
   boolean trovato;
   if (!leggiProdotto(descrizione)){
     this.codice = -1;
     this.descrizione = "-1";
     this.prezzoUnitario=0;
     this.aliquota=0;
   }
 }
```

Il metodo di ricerca all'interno del file viene definito privato in quanto è di utilizzo esclusivo del costruttore La ricerca viene realizzata attraverso un ciclo while con due condizioni d'uscita: raggiunta la fine del file oppure trovato l'elemento cercato. Viene restituito un valore booleano per segnalare l'individuazione o meno del prodotto in archivio.

```
private boolean leggiProdotto(String prodotto)
 {
  int i,
      nrRighe;
  String
      codControllo;
  boolean
      trovato;
  trovato=false;
  Lettore fileProdotti=new Lettore("prodotti.txt");
  nrRighe=fileProdotti.contaRighe();
  i=0;
  while((i<nrRighe)&&(!trovato)){
     codice=fileProdotti.leggiInt();
     descrizione=fileProdotti.leggiString();
     prezzoUnitario=fileProdotti.leggiDouble();
     aliquota=fileProdotti.leggiInt();
     if (prodotto.equals(descrizione))
       trovato=true;
     i++;
  }
  fileProdotti.chiudi();
  return trovato;
 }
```

Metodi get/set sugli attributi

.
.

Metodo per stampa scheda prodotto

.
.

Oggetto Riga

Ogni riga della fattura commerciale è costituita da un oggetto di tipo prodotto al quale si aggiungono come attributi la quantità e lo sconto relative ad ogni riga commerciale per il dato prodotto.

Attributi

```
Prodotto prod
double qta
double sconto
```

Costruttori

```
/* Costruttore impostando sconto=0 */
 Riga (Prodotto prod, double qta)
 {...}
 /* Costruttore impostando lo sconto */
 Riga (Prodotto prod,double qta,double sconto)
 {...}
```

Metodi get/set sugli attributi

.

.

Metodo per stampa scheda prodotto

.

.

Oggetto Soggetto

Per l'applicazione considerata i soggetti economici hanno delle caratteristiche simili che possono essere schematicamente: la ragione sociale e l'indirizzo. Nel caso in cui si vogliano gestire con un unico programma degli archivi di compratori e di venditori è possibile inserire come attributo un codice che permette di evidenziare se il soggetto economico è un venditore o un compratore. Per rendere più realistica la simulazione commerciale è possibile definire un costruttore che passato un riferimento del soggetto economico (codice o ragione sociale) recuperi le altre informazioni di competenza da un file (lo stesso procedimento che è stato seguito precedentemente in uno dei costruttori dell'oggetto prodotto).

Attributi

```
String ragioneSociale
String indirizzo
String tipo
```

Costruttori

```
Soggetto (String ragioneSociale, String indirizzo,String tipo)
 {...}
```

Metodi get/set sugli attributi

.

.

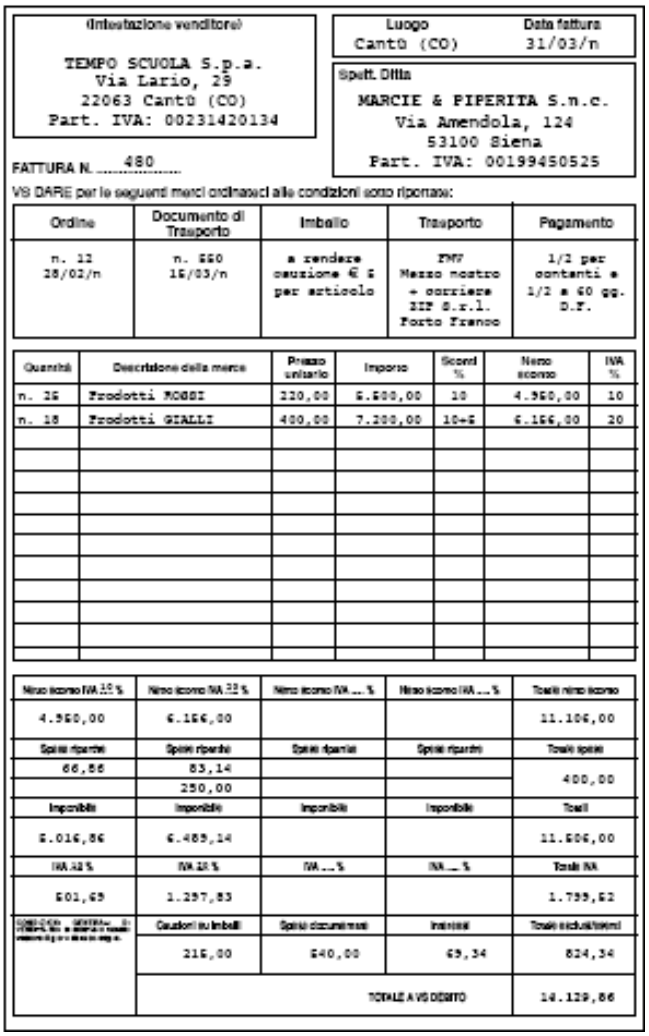
Metodo per stampa scheda prodotto

.

.

Oggetto *Fattura*

L'oggetto fattura costituisce l'aggregazione degli oggetti precedenti. Per "costruire" l'oggetto fattura è necessario conoscere quali sono i soggetti economici (compratore e venditore) ed il numero massimo di movimenti commerciali che possono intercorrere fra loro. Una volta creato l'oggetto fattura è possibile aggiungere le righe commerciali, visualizzare la fattura, calcolare gli importi di competenza (iva, imponibili, sconti)



(Intestazione venditore)

TEMPO SCUOLA S.p.a.
Via Lario, 29
22063 Cantù (CO)
Part. IVA: 00231420134

Luogo	Data fattura
Cantù (CO)	31/03/n

Spett. Ditta

MARCIE & PIPERITA S.n.c.
Via Amendola, 124
53100 Siena
Part. IVA: 00199450525

FATTURA N. 480

Vi DARE per le seguenti merci ordinateci alle condizioni sotto riportate:

Ordine	Documento di Trasporto	Imballo	Trasporto	Pagamento
n. 12 28/02/n	n. 550 15/03/n	a rendere cauzione € 5 per articolo	FMP Mezzo nostro + corriere ZIP S.r.l. Porto Franco	1/2 per contanti e 1/2 a 60 gg. D.F.

Quantità	Descrizione della merce	Prezzo unitario	Importo	Sconti %	Netto sconto	IVA %
n. 25	Prodotti ROSSI	220,00	5.500,00	10	4.950,00	10
n. 18	Prodotti GIALLI	400,00	7.200,00	10+5	6.156,00	20

Netto sconto IVA 10 %	Netto sconto IVA 20 %	Netto sconto IVA ... %	Netto sconto IVA ... %	Totale netto sconto
4.950,00	6.156,00			11.106,00
Spese riporto	**Spese riporto**	**Spese riporto**	**Spese riporto**	**Totale spese**
66,86	83,14			400,00
	250,00			
Imponibile	**Imponibile**	**Imponibile**	**Imponibile**	**Totali**
5.016,86	6.489,14			11.506,00
IVA 10 %	**IVA 20 %**	**IVA ... %**	**IVA ... %**	**Totale IVA**
501,69	1.297,83			1.799,52
	Cauzioni su imballi	**Spese documento**	**Insoluti**	**Totale esclusioni**
	215,00	540,00	69,34	824,34
			TOTALE A VS DEBITO	14.129,86

Le informazioni importanti per la fattura possono essere essenzialmente suddivise in tre componenti:

informazioni di testata
- numero della fattura
- indirizzo e ragione sociale del venditore
- indirizzo e ragione sociale del compratore

informazioni di riga
- prodotto (è un oggetto Prodotto)
- quantità di prodotto oggetto del movimento
- sconto

informazioni di fine fattura
- imponibile
- iva
- totale fattura

Attributi

```
int
    numero /* numero della fattura */
    indiceMovimenti /* numero di righe di fattura effettivamente inserite */
Soggetto
    fornitore (dati fornitore)
    cliente (dati cliente)
Riga
    righeMovimenti[]
```

Costruttore

```
Fattura (int numero,Soggetto fornitore,Soggetto cliente,int nrMovimenti)
 {
  this.numero=numero;
  this.fornitore = fornitore;
  this.cliente = cliente;
  this.righeMovimenti=new Riga[nrMovimenti];
  this.indiceMovimenti=0;
 }
```

Metodi get/set sugli attributi

.
.

Metodo per stampa scheda prodotto

.
.

Altri Metodi

```
public void aggiungiMovimento(Riga movimento)
 {
  if (indiceMovimenti<righeMovimenti.length){
    righeMovimenti[indiceMovimenti]=movimento;
    indiceMovimenti=indiceMovimenti+1;
   }

 }
private double imponibileRiga(double przRiga,double qtaRiga,double scontoRiga){
  double scRiga;
  scRiga=(1-scontoRiga/100.0);
  return  przRiga*qtaRiga*scRiga;
 }
public double calcolaIva()
 {
  int i;
  int alqRiga;
  double przRiga;
  double scontoRiga;
  double qtaRiga;
  double iva;
  iva=0;
  for (i=0;i<indiceMovimenti;i++){
    przRiga=righeMovimenti[i].getProd().getPrezzoUnitario();
    alqRiga=righeMovimenti[i].getProd().getAliquota();
    scontoRiga=righeMovimenti[i].getSconto();
    qtaRiga=righeMovimenti[i].getQta();
    iva=iva+(float)(imponibileRiga(przRiga,qtaRiga,scontoRiga)*alqRiga)/100.0;
   }
   return iva;
 }
public double calcolaImponibile()
 {
  int i;
  double przRiga;
  double scontoRiga;
  double qtaRiga;
  double imponibile;
  imponibile=0;
```

```
  for (i=0;i<indiceMovimenti;i++){
   przRiga=righeMovimenti[i].getProd().getPrezzoUnitario();
   scontoRiga=righeMovimenti[i].getSconto();
   qtaRiga=righeMovimenti[i].getQta();
   imponibile=imponibile+imponibileRiga(przRiga,qtaRiga,scontoRiga);
   }
  return imponibile;
 }
```

Dopo aver implementato le classi Prodotto e Fattura e definito un file di riferimento per i prodotti archiviati,

Es. prodotti .txt

Cod	Desc	Prz	Alq
200	Alluminio	13.5	20
210	Mais	12	10
220	Frumento	11	10
230	Ferro	13	20

è possibile realizzare un listato per l'utilizzo dell'oggetto Fattura:

Listato TestFattura (Listato TestFattura.java)

```
class TestFattura
{
 public static void main(String[] args)
 {
 int
   codiceprodotto,
   aliquota;
 String
   descrizione;
 double
   prezzoUnitario,
 iva,
   imponibile;
 Soggetto
   fornitoreProva,
   clienteProva;
 Prodotto
   prodottoProva;
 Riga
   rigaProva;
 Fattura
   fatturaProva;
 /**/
 fornitoreProva=new Soggetto("Pinco srl","Via bianchi 4","F");
 clienteProva=new Soggetto("Gialli sas","Via verdi 5","C");
 /**/
 fatturaProva=new Fattura(1,fornitoreProva,clienteProva,2);
 /**/
 prodottoProva=new Prodotto(100,"Ghisa",11.4,10);
 rigaProva=new Riga(prodottoProva,1);
 fatturaProva.aggiungiMovimento(rigaProva);
 /**/
 prodottoProva=new Prodotto("Ferro");
 rigaProva=new Riga(prodottoProva,2,50);
 fatturaProva.aggiungiMovimento(rigaProva);
```

```
 /**/
 fatturaProva.stampaScheda();
 iva=fatturaProva.calcolaIva();
 imponibile=fatturaProva.calcolaImponibile();
 System.out.println("Imponibile fattura "+imponibile);
 System.out.println("Iva fattura "+iva);
 System.out.println("TOT FATTURA "+(imponibile+iva));
 }
}
```

che genera la seguente schermata:

```
Fattura nr: 1
--------------------
Ragione Sociale : Pinco srl Indirizzo : Via bianchi 4 Tipo : F
--------------------
Ragione Sociale : Gialli sas Indirizzo : Via verdi 5 Tipo : C
--------------------
Cod   Qta Desc   PrzUni Sc Aliq
--------------------
100   1.0 Ghisa  11.4   0.0  10
230   2.0 Ferro  13.0   50.0  20
--------------------
Imponibile fattura 24.4
Iva fattura 3.74
TOT FATTURA 28.14
```

Lavoro in aula / laboratorio

- Completare la classe Prodotto
- Completare la classe Fattura
- Aggiungere il metodo scontoFattura alla classe fattura per poter applicare una percentuale di sconto
- Gestire un riepilogo Iva con la separazione per aliquote

9.2 Proposte di lavoro

Proposta Uno

Uno studio di elaborazione dati vuole generare i prospetti contabili dei sui clienti. L'oggetto cliente è costituito da: codice cliente, nominativo cliente, indirizzocliente. I clienti sono memorizzati in un file di testo secondo lo schema previsto:

Es

```
200  Rossi_Gianni   Via_Verdi_20_Brescia
210  Verdi_Luigi    Via_Bianchi_15_Sarezzo
220  Gialli_Maria   Via_Neri_12_Iseo
```

L'oggetto movimento è costituito da: codice cliente (cui il movimento si riferisce), tipologia movimento (entrata/uscita) e importo. I movimenti sono memorizzati in un file di testo secondo lo schema previsto:

Es

```
200  Entrata  30000
200  Entrata  20000
200  Uscita   10000
210  Uscita   25000
210  Entrata  11000
220  Entrata  21000
220  Uscita   22000
220  Entrata  10000
```

Supponiamo per semplicità che venga applicata solamente la tassazione Irpef (analizzatata nella scheda01) e non siano presenti detrazioni o deduzioni. La gestione deve prevedere come risultato finale una visualizzazione di questo tipo:

```
Prospetto contabile
----------------------
Cliente
NR 200
Rossi_Gianni
Via_Verdi_20_Brescia
-------------------
Movimenti
-------------------
Entrata 30000
Entrata 20000
Uscita  10000
-------------------
TotEntrate 50000
TotUscite 10000
-------------------
SaldoPositivo 40000
Irpef 11506
-------------------
NettoGestione 28494
```

Capitolo 10

Argomenti trattati

- Visione d'insieme: Gestione del conto corrente con database

- Proposte di lavoro

10.1 Visione d'insieme: Gestione del conto corrente con database

Si vuole realizzare un'applicazione java che permetta di gestire tramite un oggetto denominato ContoCorrente alcune operazioni classiche del conto corrente quali:

- elenco dei movimenti del conto corrente;
- aggiuta movimenti al conto corrente.

Il ContoCorrente interagisce con un database dove vengono memorizzate le informazioni.

Il database *banca*, realizzato con MS-ACCESS contiene:

- le informazioni relative ai conti correnti gestiti nella banca;
- i movimenti sui conti correnti;
- le informazioni sui correntisti.

Il database banca

Le tabelle gestite nel database banca sono le seguenti:

Tabella correntisti

Nome campo	Tipo dati	
id_correntista	Contatore	Contatore - chiave primaria
nom_correntista	Testo	Nome correntista
cog_correntista	Testo	Cognome correntista
cod_professione	Numerico	Codice professione

Tabella professioni

Nome campo	Tipo dati	
id_correntista	Contatore	Contatore - chiave primaria
nom_correntista	Testo	Nome correntista
cog_correntista	Testo	Cognome correntista
cod_professione	Numerico	Codice professione

Tabella tipo conti

Nome campo	Tipo dati	
id_tipo	Contatore	Contatore - primaria
des_tipo	Testo	Descrizione tipo conto

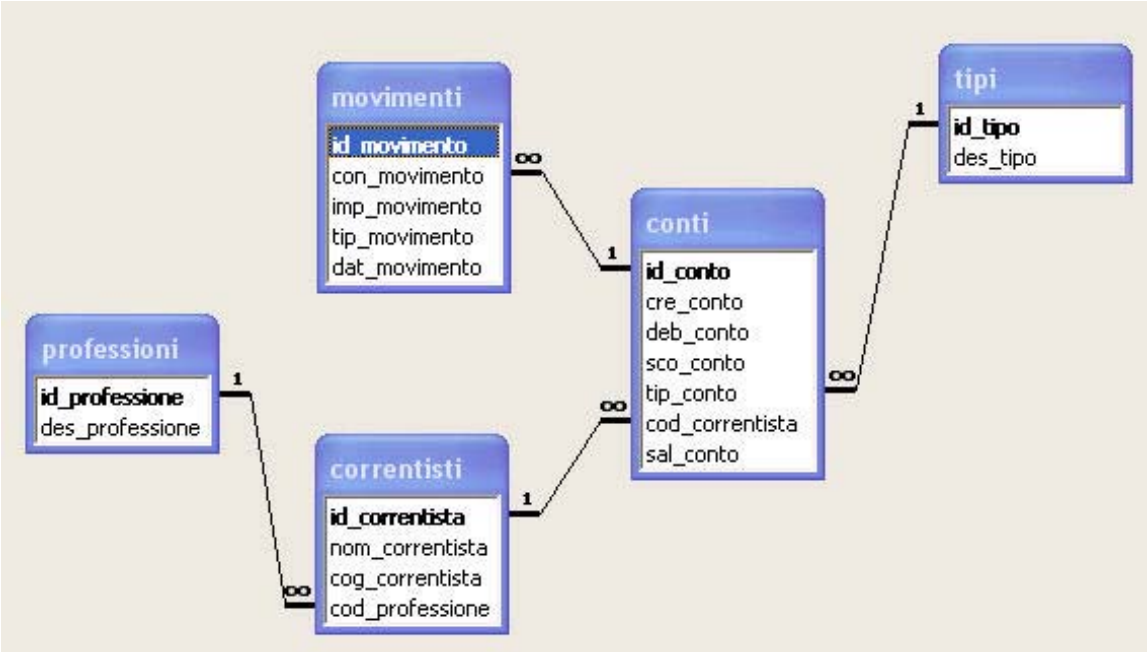
Tabella movimenti

Nome campo	Tipo dati	
id_movimento	Contatore	Contatore - chiave primaria
con_movimento	Numerico	Codice conto movimentato
imp_movimento	Numerico	Importo movimento
tip_movimento	Testo	Tipo movimento dare/avere
dat_movimento	Data/ora	Data movimento

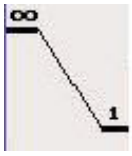
Tabella conti

Nome campo	Tipo dati	
id_conto	Contatore	Contatore - chiave primaria
cre_conto	Numerico	Tasso creditore
deb_conto	Numerico	Tasso debitore
sco_conto	Numerico	Massimo scoperto
tip_conto	Numerico	Codice tipo conto
cod_correntista	Numerico	Codice correntista
sal_conto	Numerico	Saldo conto corrente

con le seguenti relazioni:



La relazione viene evidenziata graficamente con la cardinalità:
(fra tabella movimenti e conti)



indica una relazione molti a uno (Esempio: molti movimenti su un conto corrente se letta da sinistra a destra, un conto corrente con molti movimenti se letta da destra a sinistra).

Gli oggetti gestiti dall'applicazione java sono:

→ Correntista
→ ContoCorrente

Non viene replicata completamente in memoria tutta la struttura del database, in particolare non vengono creati oggetti relativi alle tabelle tipi movimenti, movimenti, professioni perchè lo scopo del mini-gestionale è gestire le operazioni del conto corrente attraverso l'interazione ODBC sul database banca. Le righe movimenti hanno un senso in quanto create da metodi dell'oggetto ContoCorrente e le tabelle professioni e tipi movimento vengono incorporate nella gestione dall'oggetto conto corrente come elementi descrittivi.

Oggetto Correntista

Attributi

Cognome
Nome
Professione

Metodi

Get
Set
toString

L'oggetto Correntista viene incorporato nell'oggetto Conto Corrente. La creazione dell'oggetto Correntista è concomitante alla creazione dell'oggetto ContoCorrente. Nel database la professione viene gestita rispettando le regole di normalizzazione in una tabella separata (professioni) e nella tabella correntisti c'è solo il codice di rimando. Nella creazione dell'oggetto Correntista invece viene riportata nell'attributo Professione la decodifica estesa della professione.

Oggetto Conto Corrente

L'oggetto Conto Corrente è molto simile a quello analizzato nella scheda 07. Si evidenziano le variazioni principali:

Attributi aggiuntivi

titolareConto

Il titolare conto è un oggetto che contiene le informazioni anagrafiche (nome, cognome) e professionali (professione) del correntista.

Metodi gestiti

Costruttore

```
ContoCorrente(int codice,double tassoCreditore, double tassoDebitore,
  double massimoScoperto, String tipoConto,Correntista titolareConto,
  double saldo)
 {
   this.tassoCreditore = tassoCreditore;
   this.tassoDebitore = tassoDebitore;
   this.massimoScoperto = massimoScoperto;
   this.saldo=saldo;
   this.codice=codice;
   this.tipoConto=tipoConto;
   this.titolareConto=titolareConto;
 }
```

Il costruttore del conto è sostanzialmente immutato, viene aggiunta la gestione dell'attributo titolareConto. La grossa differenza è che le informazioni passate al costruttore non derivano più da input dell'utente o dal caricamento di files, ma dall'interazione tramite il database banca.

Metodo aggiungiMovimento

```
public boolean aggiungiMovimento(double importoMovimento,String tipoMovimento,
                                 String dataMovimento)
 {
   String queryDatabase;
   String nomeDatabase;
   boolean eseguito=true;
   nomeDatabase = "banca";
   Database.apriConnessione(nomeDatabase);
   if (tipoMovimento.equals("d") && (saldo-importoMovimento)<(-massimoScoperto))
     eseguito=false;
   else{
     queryDatabase =
                "insert into movimenti"+
                " (con_movimento, imp_movimento, tip_movimento, dat_movimento) "+
                "VALUES ("'+codice+"','+importoMovimento+",'"+tipoMovimento+"','"
                +dataMovimento+"');";
     Database.eseguiUpdate(queryDatabase);
     if (tipoMovimento.equals("d"))
       saldo=saldo-importoMovimento;
     else
       saldo=saldo+importoMovimento;
     queryDatabase =
                "update conti set conti.sal_conto="+saldo+" "+
                "where conti.id_conto = "+codice+";";
     Database.eseguiUpdate(queryDatabase);
     Database.chiudiConnessione();
   }
 return eseguito;
}
```

Il metodo aggiungi Movimento ha come parametri l'importoMovimento, il tipoMovimento e la dataMovimento. Viene restituito un valore boolean che può costituire un primo controllo per la corretta esecuzione del

metodo. Nel caso in cui il movimento sia di tipo "dare" viene controllata la non superabilità del massimoScoperto. Se a causa del movimento si superasse la soglia di massimoScoperto il movimento non viene effettuato e il metodo restituisce un valore false.

L'esecuzione del metodo aggiungiMovimento ha tre conseguenze:

1

La scrittura nel database banca nella tabella movimenti del movimento relativo tramite la query SQL :

```
insert into movimenti
(con_movimento, imp_movimento, tip_movimento, dat_movimento)
values ('codice',importoMovimento,'tipoMovimento','dataMovimento');
```

Non viene passato l'id_movimento in quanto in Access questo campo è stato definito come campo contatore e si autoincrementa. Nella definizione della data è stato indicato come formato : gg/mm/aa

2

L'aggiornamento dell'attributo saldo dell'oggetto ContoCorrente sulla base del tipo movimento (dare/avere)

```
if (tipoMovimento.equals("d"))
  saldo=saldo-importoMovimento;
else
  saldo=saldo+importoMovimento;
```

3

L'aggiornamento nel database banca, nella tabella conti, del saldo aggiornato del conto corrente oggetto del movimento tramite la query SQL relativa

```
update conti set conti.sal_conto=saldo
where conti.id_conto = codice;
```

Metodo listaMovimenti

```
public boolean listaMovimenti(String dataPartenza,String dataArrivo)
 {
   String matriceDati[][]=null;
   String queryDatabase,nomeDatabase;
   int i,codiceMovimento;
   double importoMovimento;
   String tipoMovimento,dataMovimento=null,dataStampa=null;
   boolean eseguito=true;
   nomeDatabase = "banca";
   Database.apriConnessione(nomeDatabase);
   if (dataPartenza==null&&dataArrivo==null){
     queryDatabase =
             "select id_movimento, imp_movimento, tip_movimento, dat_movimento "+
             "from movimenti "+
             "where movimenti.con_movimento = "+codice+";";
   }
   else{
     queryDatabase =
             "select id_movimento, imp_movimento, tip_movimento, dat_movimento "+
             "from movimenti "+
             "where movimenti.con_movimento = "+codice+
             "and (movimenti.dat_movimento between #"+dataPartenza+"# and #"
             +dataArrivo+"#);";
   }
   if (Database.eseguiQuery(queryDatabase)==null){
     matriceDati=Database.estraiMatrice();
     for (i=0;i<matriceDati.length;i++){
       codiceMovimento=Integer.parseInt(matriceDati[i][0]);
```

```
      importoMovimento=Double.parseDouble(matriceDati[i][1]);
      tipoMovimento=matriceDati[i][2];
      dataMovimento=matriceDati[i][3].substring(0,10);
      dataStampa=dataMovimento.substring(8,10)+"/"+dataMovimento.substring(5,7)+
      "/"+dataMovimento.substring(0,4);
      System.out.println(codiceMovimento+" "+importoMovimento+" "+tipoMovimento+
      " "+dataStampa);
    }
  }
  else
    eseguito=false;
  Database.chiudiConnessione();
  return eseguito;
}
```

Il metodo listaMovimenti prevede la visualizzazione di tutti i movimenti relativi al conto compresi fra i due parametri dataPartenza e dataArrivo. Nel caso in cui le date di parzializzazione movimenti non vengano specificate la query SQL è la seguente:

```
select id_movimento, imp_movimento, tip_movimento, dat_movimento
from movimenti
where movimenti.con_movimento = codice;
```

se invece sono specificate parzializzazioni la query diventa:

```
select id_movimento, imp_movimento, tip_movimento, dat_movimento
from movimenti
where movimenti.con_movimento = codice
and (movimenti.dat_movimento between #dataPartenza# and #dataArrivo#);
```

I dati che vengono importati dalla query di interrogazione vengono sottoposti a parsing nel caso di interi e double, mentre nel caso del tipo data, vengono prelevati i primi 10 caratteri della stringa restituita dal database (che restituisce la data in formato esteso compreso di ora) e ricompattati per la stampa nel formato gg/mm/aaaa.

Metodi per la gestione dell'HashSet

Volendo organizzare tutti i conti che possono essere gestiti attraverso una collezione è necessario l'override dei metodi equals e hashCode

```
public boolean equals(Object contoConfronto)
 {
   ContoCorrente altroConto=(ContoCorrente)contoConfronto;
   return this.codice==altroConto.codice;
 }

 public int hashCode()
 {
   int hash1=new Integer(codice).hashCode();
   final int HAS_MOLT=50;
   int hash=HAS_MOLT*hash1;
   return hash;
 }
```

Essendo definito il codice conto come contatore e chiave primaria, ho la certezza che a livello di database non ci siano conti con lo stesso codice conto, per cui posso basare sia il metodo equals che il metodo hashCode sul campo codice

Un semplice programma che utilizza gli oggetti citati precedentemente è riportato di seguito:

Listato Java (OperazioniConto.java)

```
import java.util.*;
import unibs.eco.dmq.basicIO.*;
import unibs.eco.dmq.database.*;
public class OperazioniConto{
 public static void main(String[] args) {
   int
     i,opzione;
   Set
     contiCorrenti=new HashSet();
   boolean
     datiCaricati=false;
   /* ------------------------------------------------*/
   do{
     opzione=menu();
     switch(opzione){
       case 1:
               caricamentoConti(contiCorrenti);
               datiCaricati=true;
               break;
       case 2:
               if(datiCaricati)
                 visualizzaConti(contiCorrenti);
               else
                 Scrittore.video.println("Conti non caricati");
               break;
       case 3:
               if(datiCaricati)
                 movimentaConti(contiCorrenti);
               else
                 Scrittore.video.println("Conti non caricati");
               break;
       case 4:
               if(datiCaricati)
                 listaMovimenti(contiCorrenti);
               else
                 Scrittore.video.println("Conti non caricati");
               break;
       case 5:
               if(datiCaricati)
                 listaMovimentiD(contiCorrenti);
               else
                 Scrittore.video.println("Conti non caricati");
               break;
       case 0:
               break;
       default:
               Scrittore.video.println("Opzione non gestita");
               break;
     }
   }while(opzione!=0);
 }
 /* Menu gestione */
 public static int menu(){
   int
     opzione;
   Scrittore.video.println("-----------------------------");
   Scrittore.video.println("1* Caricamento Conti Gestibili");
```

```
  Scrittore.video.println("2* Lista Conti");
  Scrittore.video.println("3* Movimenta Conti");
  Scrittore.video.println("4* Lista Movimenti");
  Scrittore.video.println("5* Lista Movimenti fra Date");
  Scrittore.video.println("0* USCITA");
  Scrittore.video.println("------------------------------");
  opzione=Lettore.tastiera.leggiInt();
  return opzione;
}
/* Caricamento hash set conti correnti */
public static void caricamentoConti(Set contiCorrente){
  ContoCorrente
    conto;
  String
    matriceDati[][],vettoreTestata[],queryDatabase,nomeDatabase;
  int
    i,j,codice;
  double
    tassoCreditore,tassoDebitore,massimoScoperto;
  String
    tipoConto,nomeCorrentista,cognomeCorrentista,professioneCorrentista;
  double
    saldo=0.0;
  Correntista
    titolareConto;
  vettoreTestata= null;
  matriceDati=null;
  nomeDatabase = "banca";
  queryDatabase =
    "select conti.id_conto,conti.cre_conto,conti.deb_conto,conti.sco_conto,"+
    "tipi.des_tipo,correntisti.nom_correntista,correntisti.cog_correntista,"+
    "professioni.des_professione,conti.sal_conto "+
    "from professioni,conti,tipi,correntisti "+
    "where tipi.id_tipo = conti.tip_conto "+
    "and correntisti.id_correntista = conti.cod_correntista "+
    "and professioni.id_professione=correntisti.cod_professione;";
  Database.apriConnessione(nomeDatabase);
  if (Database.eseguiQuery(queryDatabase)==null){
    matriceDati=Database.estraiMatrice();
    for (i=0;i<matriceDati.length;i++){
      codice=Integer.parseInt(matriceDati[i][0]);
      tassoCreditore=Double.parseDouble(matriceDati[i][1]);
      tassoDebitore=Double.parseDouble(matriceDati[i][2]);
      massimoScoperto=Double.parseDouble(matriceDati[i][3]);
      tipoConto=matriceDati[i][4];
      nomeCorrentista=matriceDati[i][5];
      cognomeCorrentista=matriceDati[i][6];
      professioneCorrentista=matriceDati[i][7];
      saldo=Double.parseDouble(matriceDati[i][8]);
      titolareConto=new Correntista(nomeCorrentista,cognomeCorrentista,professioneCorrentista);
      conto=new ContoCorrente(codice,tassoCreditore,tassoDebitore,massimoScoperto,
                              tipoConto,titolareConto,saldo);
      contiCorrente.add(conto);
    }
  }
  Database.chiudiConnessione();
  return;
```

```
}

public static void movimentaConti(Set contiCorrente){
  ContoCorrente
    conto=null;
  boolean
    trovaConto=false,esito;
  String
    dataMovimento,tipoMovimento;
  int
    codice;
  double
    importoMovimento;
  Scrittore.video.println("Inserisci il codice del conto da movimentare");
  codice=Lettore.tastiera.leggiInt();
  conto=cercaConto(codice,contiCorrente);
  if (conto!=null){
    Scrittore.video.println("Inserisci il l'importo del movimento");
    importoMovimento=Lettore.tastiera.leggiDouble();
    Scrittore.video.println("Dare/Uscita [d] o Avere/Entrata [a]");
    tipoMovimento=Lettore.tastiera.leggiString();
    Scrittore.video.println("Data del movimento in formato gg/mm/aaaa");
    dataMovimento=Lettore.tastiera.leggiString();
    esito=conto.aggiungiMovimento(importoMovimento,tipoMovimento,dataMovimento);
    if(!esito)
      Scrittore.video.println("Movimento non effettuato");
  }
  else
    Scrittore.video.println("Conto non gestito");
  return;
}

public static void listaMovimenti(Set contiCorrente){
  ContoCorrente
    conto=null;
  int
    codice;
  Scrittore.video.println("Inserisci il codice del conto di cui vuoi visualizzare i movimenti");
  codice=Lettore.tastiera.leggiInt();
  conto=cercaConto(codice,contiCorrente);
  if (conto!=null){
    Scrittore.video.println("---------------------------------------");
    Scrittore.video.println(conto);
    Scrittore.video.println("---------------------------------------");
    conto.listaMovimenti();
    Scrittore.video.println("---------------------------------------");
  }
  else
    Scrittore.video.println("Conto non gestito");
  return;
}

public static void listaMovimentiD(Set contiCorrente){
  ContoCorrente
    conto=null;
  int
    codice;
```

```
  String
    dataPartenza=null,dataArrivo=null;
  Scrittore.video.println("Inserisci il codice del conto di cui vuoi visualizzare i movimenti");
  codice=Lettore.tastiera.leggiInt();
  conto=cercaConto(codice,contiCorrente);
  if (conto!=null){
    Scrittore.video.println("Data partenza in formato gg/mm/aaaa");
    dataPartenza=Lettore.tastiera.leggiString();
    Scrittore.video.println("Data arrivo in formato gg/mm/aaaa");
    dataArrivo=Lettore.tastiera.leggiString();
    Scrittore.video.println("---------------------------------------");
    Scrittore.video.println(conto);
    Scrittore.video.println("---------------------------------------");
    conto.listaMovimenti(dataPartenza,dataArrivo);
    Scrittore.video.println("---------------------------------------");
  }
  else
    Scrittore.video.println("Conto non gestito");
  return;
}

public static ContoCorrente cercaConto(int codice, Set contiCorrente){
  boolean
    trovaConto=false;
  ContoCorrente
    conto=null,selezionaConto=null;
  Iterator
    iterazione=contiCorrente.iterator();
  while (iterazione.hasNext()&&trovaConto==false){
    conto=(ContoCorrente)iterazione.next();
    if (conto.getCodice()==codice){
      trovaConto=true;
      selezionaConto=conto;
    }
  }
  return selezionaConto;
}

public static void visualizzaConti(Set contiCorrente){
  Iterator
    iterazione=contiCorrente.iterator();
  ContoCorrente
    conto;
  while (iterazione.hasNext()){
    conto=(ContoCorrente)iterazione.next();
    Scrittore.video.println("---------------------------------------");
    Scrittore.video.println(conto);
  }
  return;
}
}
```

Commento al listato OperazioniConto

I conti correnti gestibili vengono memorizzati in una struttura di tipo HashSet perchè le caratteristiche di questa struttuta concordano con le necessità dell'esempio (non è rilevante l'ordine con cui vengono memorizzati, deve essere possibile l'aggiunta di nuovi elementi e la scansione degli elementi memorizzati).

Metodi principali

Metodo caricamentoConti

Il metodo caricamentoConti carica l'hashSet con i dati risultanti da una query sul database banca. Il risultato della query viene caricato in una matrice bidimensionale di stringhe, per i dati numerici è quindi necessario un parsing. Ogni riga della matrice risultato contiene i dati necessari per la creazione del conto corrente associato. Viene creato come oggetto anche il correntista che viene passato al costruttore dell'oggetto conto corrente.

Metodo movimentaConti

Il metodo movimentaConti dopo aver verificato l'esistenza del conto da movimentare, chiede i parametri della riga movimento non impliciti negli attributi del conto: importo, tipo movimento e data movimento. Una volta acquisiti i parametri viene inovocato il metodo aggiungi movimento

Metodi listaMovimenti e listaMovimentiD

Il metodo listaMovimenti (listaMovimentiD) dopo aver verificato l'esistenza del conto da interrogare, richiama il metodo listaMovimenti senza parametri per una visualizzazione indiscriminata dei movimenti sul conto in una data qualsiasi oppure, nel caso del metodo listaMovimentiD richiama il metodo listaMovimenti passando come parametri di parzializzazione la data di partenza e quella di arrivo dell'intervallo desiderato

Metodo cercaConto

E' un metodo di supporto che riceve come parametri d'ingresso il codice conto e l'HashSet e restituisce il conto richiesto se appartiene all'insieme oppure null se il conto richiesto non vi appartiene.

Alcuni controlli

Se non è stato riempito l'HashSet nello switch non si abilitano altre opzioni se non quella di caricamento, tramite la variabile booleana datiCaricati che assume il valore true all'atto del caricamento

```
if(datiCaricati)
  visualizzaConti(contiCorrenti);
else
  Scrittore.video.println("Conti non caricati");
```

Nello operazioni permesse dallo switch viene segnalato il tentativo di utilizzare opzioni non gestite con l'opzione default

```
default:
       Scrittore.video.println("Opzione non gestita");
```

Viene utilizzato il valore restituito dal metodo cercaConto per effettuare le operazioni oppure segnalare la non presenza del conto fra quelli gestiti

```
conto=cercaConto(codice,contiCorrente);
if (conto!=null){
  .
  .
}
else
  Scrittore.video.println("Conto non gestito");
```

Il metodo aggiungiMovimento restituisce un valore booleano false se il movimento non viene effettuato. Il controllo all'interno del conto corrente prevede che il movimento non venga effettuato se si tratta di un'uscita e se tale uscita porterebbe al superamento della soglia di massimo scoperto. Posso quindi effettuare un test sulla variabile booleana esito

```
esito=conto.aggiungiMovimento(importoMovimento,tipoMovimento,dataMovimento);
if(!esito)
  Scrittore.video.println("Movimento non effettuato");
```

Lavoro in aula / laboratorio

- Controllare in fase di inserimento che il tipo del movimento sia o solo dare "d" o solo avere "a"
- Per il lancio di nuovi prodotti si vuole riportare un elenco delle professioni dei correntisti e il loro saldo
- Riportare il saldo medio
- Permettere in fase di visualizzazione la parzializzazione dei movimenti in base al tipo (solo movimenti dare "d" ed avere "a"

Capitolo 11

Argomenti trattati

- Appello 11/01/05

11.1 Appello 11/01/05

Teoria

A) Descrivere l'algoritmo di ricerca binaria. In particolare indicare quali condizioni sono richieste sui dati, come opera e la sua complessità computazionale.

B) Descrivere la struttura delle liste concatenate. Discuterne vantaggi e svantaggi rispetto agli array.

Algoritmi e codice

1 Errori sintattici / logici

Il seguente programma Java deve simulare il lancio di una moneta un certo numero di volte. Il numero di prove viene introdotto dall'utente. Nel caso in cui l'estrazione tramite il metodo random restituisca un valore inferiore a 0.5 viene considerata l'uscita di testa altrimenti l'uscita di croce. Alla fine delle estrazioni il metodo deve comunicare i risultati della simulazione.

versione da correggere

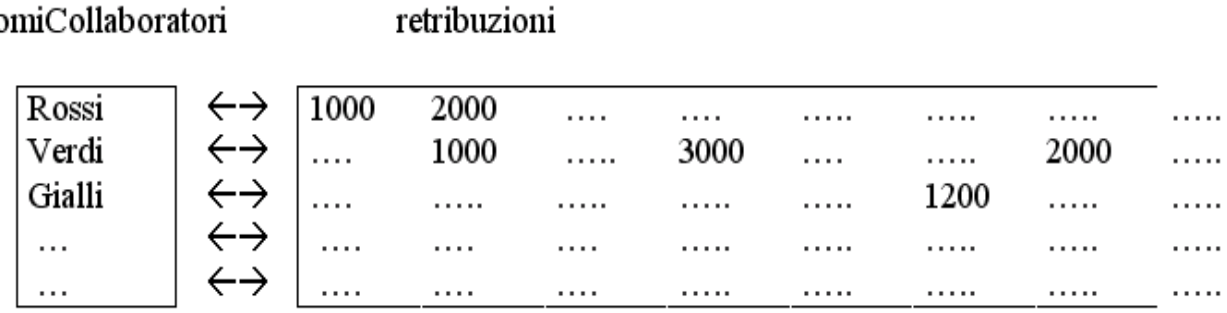
```
import unibs.eco.dmq.basicIO
public class ProbaFreq{
 public static void main(String[] args) {
  int
  numeroTesta;
  numeroCroce;
  numeroProve;
  contaProve;
  double
  estratto;
  /*Inizio*/
  numeroTesta=0;
  numeroCroce=0;
  contaProve=0;
  Scrittore.video.println("Numero di lanci da effettuare :");
  Lettore.tastiera.leggiInt()=numeroProve;
  while(contaProve<>numeroProve){
    estratto=Math.random();
    if (estratto<0.5)
      numeroTesta++;
      Scrittore.video.println("Testa");
    else{
      numeroCroce++;
      Scrittore.video.println("Croce");
    }
    contaProve++1;
  }
  Scrittore.video.println("Tot testa "+numeroTesta+ " Frequenza "
  +(float)numeroTesta/numeroProve);
  Scrittore.video.println("Tot croce "+numeroCroce+ " Frequenza "
  +(float)numeroCroce/numeroProve);
 }
}
```

versione corretta

```
import java.io.*;
import unibs.eco.dmq.basicIO.*;
public class ProbaFreq{
 public static void main(String[] args) {
  int
  numeroTesta,
  numeroCroce,
  numeroProve,
  contaProve;
  double
  estratto;
  /*Inizio*/
  numeroTesta=0;
  numeroCroce=0;
  contaProve=0;
  Scrittore.video.println("Numero di lanci da effettuare :");
  numeroProve=Lettore.tastiera.leggiInt();
  while(contaProve<numeroProve){
    estratto=Math.random();
    if (estratto<0.5){
      numeroTesta++;
      Scrittore.video.println("Testa");
    }
    else{
      numeroCroce++;
      Scrittore.video.println("Croce");
    }
    contaProve++;
  }
  Scrittore.video.println("Tot testa "+numeroTesta+ " Frequenza "
  +(float)numeroTesta/numeroProve);
  Scrittore.video.println("Tot croce "+numeroCroce+ " Frequenza "
  +(float)numeroCroce/numeroProve);
 }
}
```

2 Algoritmi classici

L'ufficio "Retribuzioni collaboratori" registra annualmente l'ammontare delle 12 retribuzioni mensili attraverso le strutture nomiCollaboratori e retribuzioni in relazione posizionale fra loro



Implementare il metodo

```
public static double totaleCollaboratore( String collaboratoreSelezionato,
                                          String nomiCollaboratori[ ],
```

```
                                        double retribuzioni[ ][ ]) {
    ...
    completare
    ...
}
```

che dato in ingresso il nome di un soggetto (collaboratoreSelezionato), l'array con i nomi di tutti i collaboratori (nomiCollaboratori) e l'array delle retribuzioni (retribuzioni) restituisce l'ammontare della retribuzione complessiva del soggetto indicato. Si assume, per semplicità, che non ci siano casi di omonimia e nel caso in cui il nome indicato non corrisponda ad un collaboratore effettivamente presente nell'elenco deve essere restituito come totale il valore -1.

Esempi
– se richiesto nominativo Rossi, il metodo restituirà la somma 1000+2000
– se richiesto nominativo Violetti, il metodo restituirà -1

possibile implementazione

```
public static double totaleCollaboratore(String collaboratoreSelezionato,
                                         double retribuzioni[][],
                                         String nomiCollaboratori[])
 { int i,j;
   double totale=-1;
   for (i=0;i<nomiCollaboratori.length;i++)
     if (collaboratoreSelezionato.equals(nomiCollaboratori[i]))
       {
         totale=0;
         for(j=0;j<retribuzioni[i].length;j++)
           totale=totale+retribuzioni[i][j];
       }
 return totale;
 }
```

3 Oggetti

A livello molto sintetico gli scambi commerciali di una nazione possono essere rappresentati dal sistema Importazioni/Esportazioni. Schematicamente l'oggetto avrà Nazione come caratteristiche principali il nome nazione e il saldo commerciale; le operazioni svolte sono l'importazione e l'esportazione espresse in termini di quantità (positive) di denaro. L'importazione diminuisce il saldo commerciale, l'esportazione lo aumenta.

A) Implementare in Java la classe Nazione (con le proprietà, e i metodi necessari)

possibile implementazione

```
class Nazione
{
 private String nome;
 private double saldo;
 Nazione(String nome, double saldo)
 {
   this.nome=nome;
   this.saldo=saldo;
 }
 public double getSaldo()
 {
   return saldo;
```

```
 }
 public String getNome()
 {
   return nome;
 }
 public void importazione(double somma)
 {
   this.saldo = saldo-somma;
 }
 public void esportazione(double somma)
 {
   this.saldo = saldo+somma;
 }
 public String toString()
 {
   return nome+" "+saldo;
 }
}
```

B) Completare il seguente codice Java con struttura a menù:

1) caricamento in un vettore di oggetti Nazione i cui dati (nome e saldo commerciale) sono reperiti da un file di testo "FileDati.txt";

2) visualizzazione delle nazioni gestite: nome e saldo commerciale attuale;

3) effettuazione di un'operazione commerciale (Importazione / Esportazione di un certo importo). Questa operazione richiede l'inserimento del nome della nazione, del tipo di operazione e della somma (positiva) di denaro coinvolta (si veda esempio riportato sotto). Utilizzare un metodo di ricerca che dato in ingresso un nome nazione restituisce null se la nazione con nome corrispondente non è disponibile altrimenti restituisce l'oggetto Nazione desiderato

codice da completare

```
import unibs.eco.dmq.basicIO.*;
public class TestNazione{
 public static void main(String[] args) {
 int i,opzione;
 Nazione[] nazioni = null;
 boolean datiCaricati=false;
 do{
   opzione=menu();
   switch(opzione){
     case 1: nazioni = caricamentoNazioni("FileDati.txt");
             datiCaricati=true;
             break;
     case 2: if(datiCaricati)
               visualizzaNazioni(nazioni);
             else
               Scrittore.video.println("Nazioni non caricate");
             break;
     case 3: if(datiCaricati)
               movimentaNazione(nazioni);
             else
               Scrittore.video.println("Nazioni non caricate");
             break;
     case 0: break;
     default: Scrittore.video.println("Opzione non gestita");
              break;
```

```
   }
  }while(opzione!=0);
 }
 public static int menu(){
   int opzione;
   Scrittore.video.println("------------------------------");
   Scrittore.video.println("1* Caricamento Nazioni");
   Scrittore.video.println("2* Visualizzazione Nazioni");
   Scrittore.video.println("3* Operazione commerciale");
   Scrittore.video.println("0* USCITA");
   Scrittore.video.println("------------------------------");
   opzione=Lettore.tastiera.leggiInt();
   return opzione;
 }
 public static Nazione[] caricamentoNazioni(String nomeFile){
   // completare
 }

 public static void visualizzaNazioni(Nazione [] nazioni){
   // completare
 }
 public static void movimentaNazione(Nazione [] nazioni){
   // completare
 }
 public static Nazione cercaNazione(Nazione[] nazioni,String nomeNazione){
  // completare
 }
}
```

possibile implementazione

```
public static Nazione[] caricamentoNazioni(String nomefile){
  Lettore fileNazioni=new Lettore(nomefile);
  int nrRighe=fileNazioni.contaRighe();
  String descrizione;
  double saldo;
  Nazione[] nazioni;
  nazioni=new Nazione[nrRighe];
  int i=0;
  while((i<nrRighe)){
   descrizione=fileNazioni.leggiString();
   saldo=fileNazioni.leggiDouble();
   nazioni[i]=new Nazione(descrizione,saldo);
   i++;
  }
  fileNazioni.chiudi();
  return nazioni;
}

public static void visualizzaNazioni(Nazione [] nazioni){
  int i;
  for (i=0;i<nazioni.length;i++)
    Scrittore.video.println(nazioni[i]);
    return;
}
```

```
public static void movimentaNazione(Nazione [] nazioni){
  Nazione nazioneSelezionata=null;
  String nomeNazione;
  int tipoMovimento;
  double sommaMovimento;
  Scrittore.video.println("Nazione da movimentare");
  nomeNazione=Lettore.tastiera.leggiString();
  nazioneSelezionata=cercaNazione(nazioni,nomeNazione);
  if (nazioneSelezionata!=null){
    do{
      Scrittore.video.println("0 - Importazione 1 - Esportazione");
      tipoMovimento=Lettore.tastiera.leggiInt();
    }while (tipoMovimento!=0&&tipoMovimento!=1);
    Scrittore.video.println("Somma");
    sommaMovimento=Lettore.tastiera.leggiDouble();
    if (tipoMovimento==0)
      nazioneSelezionata.importazione(sommaMovimento);
    else
      nazioneSelezionata.esportazione(sommaMovimento);
    }
 else
   Scrittore.video.println("Nazione non gestita");
 return;
}

public static Nazione cercaNazione(Nazione[] nazioni,String nomeNazione){
  int i;
  Nazione trovaNazione=null;
  for (i=0;i<nazioni.length&&trovaNazione==null;i++)
    if(nomeNazione.equals(nazioni[i].getNome()))
      trovaNazione=nazioni[i];
return trovaNazione;
}
```

```
1* Caricamento Nazioni
2* Visualizzazione Nazioni
3* Operazione commerciale
0* USCITA
--------------------------------
2
Italia  200.0
Francia  100.0
Spagna  50.0
```

```
--------------------------------
1* Caricamento Nazioni
2* Visualizzazione Nazioni
3* Operazione commerciale
0* USCITA
--------------------------------
3
Nazione da movimentare
Italia
0 - Importazione 1 - Esportazione
0
Somma
100
--------------------------------
1* Caricamento Nazioni
2* Visualizzazione Nazioni
3* Operazione commerciale
0* USCITA
--------------------------------
2
Italia  100.0
Francia  100.0
Spagna  50.0
```

Capitolo 12

Argomenti trattati

- Appello 10/01/06

12.1 Appello 10/01/06

A) Strumenti di analisi
SOLO per chi NON ha superato la prova intermedia

Analizzare la problematica relativa al calcolo del montante secondo la formula:

$$M = C\ (1 + i)^n$$

Legenda: C = capitale
n = numero intero di periodi
i = tasso di interesse

e realizzare il flowchart relativo.

Nota: l'esecutore del flowchart può eseguire solo le 4 operazioni aritmetiche di base: somma, differenza, prodotto e divisione

B) Algoritmi e codice

B1 Errori sintattici e/o logici
SOLO per chi NON ha superato la prova intermedia

Il seguente programma Java permette l'inserimento e la visualizzazione di un array monodimensionale. La gestione avviene tramite un menu che permette di selezionare le opzioni desiderate.

versione da correggere

```
import unibs.eco.dmq.basicIO.*;
public class ArrayMono{
  public static void main(String[] args) {
    final int = DIM_ARRAY=7;
    double[] quotazioni;
    int i; opzione;
    quotazioni=new String[DIM_ARRAY];
    do{
      Scrittore.video.println("1* Inserimento dati");
      Scrittore.video.println("2* Visualizzazione dati");
      Scrittore.video.println("0* Uscita");
      Lettore.tastiera.leggiInt(opzione);
      switch(opzione){
        case '1':
          for(i=0;i>DIM_ARRAY;i++){
            Scrittore.video.println("Inserire quotazione "+i);
            quotazioni[x]=Lettore.tastiera.leggiDouble();
          }
          brek;
        case '2':
          for(i=0;i<DIM_ARRAY;i--)
            Scrittore.video.println(quotazioni[i]);
          brek;
      }
    }while(opzione< >0);
  }
}
```

versione corretta

```
import unibs.eco.dmq.basicIO.*;
public class ArrayMono{
  public static void main(String[] args) {
    final int
      DIM_ARRAY=7;
    double[]
      quotazioni;
    int
       i,opzione;
    quotazioni=new double[DIM_ARRAY];
    do{
      Scrittore.video.println("---------------------");
      Scrittore.video.println("1* Inserimento dati");
      Scrittore.video.println("2* Visualizzazione dati");
      Scrittore.video.println("0* Uscita");
      Scrittore.video.println("---------------------");
      opzione=Lettore.tastiera.leggiInt();
        switch(opzione){
          case 1:
            for(i=0;i<DIM_ARRAY;i++){
              Scrittore.video.println("Inserire quotazione "+i);
              quotazioni[i]=Lettore.tastiera.leggiDouble();
            }
            break;
          case 2:
            for(i=0;i<DIM_ARRAY;i++)
              Scrittore.video.println(quotazioni[i]);
            break;
        }
    }while(opzione!=0);
  }
}
```

B2 Algoritmi classici

a) OBBLIGATORIO per chi NON ha superato la prova intermedia
FACOLTATIVO per gli altri

Implementare un metodo Java che riceve in input un array di tipo int e determina se i valori dell'array sono disposti in modo simmetrico.

Esempio:

0,1,1,0 (simmetrico)
1,2,5,2,1 (simmetrico)
1,2,4,4,1 (non simmetrico)

L'informazione richiesta può essere associata ad un tipo booleano: Simmetrico o Non simmetrico. Il metodo restituisce valore true se l'array è simmetrico e restituisce vale false se l'array è asimmetrico.

```
public static boolean statusVettore(int vettore[]){
  boolean
    OkSimmetrico=true;
  int
    i,lung,lungMezzi;
  OkSimmetrico=true;
  lung=vettore.length;
  lungMezzi=lung/2;
  for(i=0;(i<lungMezzi)&&(OkSimmetrico==true);i++)
    if(vettore[i]!=vettore[lung-1-i])
      OkSimmetrico=false;
  return OkSimmetrico;
}
```

b) OBBLIGATORIO per TUTTI

Un array bidimensionale 8x8 di tipo base int viene utilizzato per rappresentare la scacchiera del gioco della dama. I codici numerici utilizzati nell'array sono i seguenti:

0 casella vuota
1 casella occupata da una pedina nera
2 casella occupata da una pedina bianca

Si ricorda la regola secondo cui nel gioco della dama ci sono, sulla scacchiera, al massimo 12 pedine bianche e 12 pedine nere; inoltre ogni pedina (bianca o nera) occupa una casella nera.

Implementare un metodo Java che riceve in input un array bidimensionale 8x8 di tipo base int (che rappresenta la scacchiera) e restituisce come risultato il codice numerico:

0 se la posizione e il numero delle pedine sulla scacchiera sono corrette
1 se c'è almeno una pedina che occupa una casella bianca
2 se ci sono più di 12 pedine nere o più di 12 pedine bianche

(Suggerimento: una casella nera si riconosce in quanto la somma dei suoi indici di riga e colonna è un numero dispari.)

Esempi:

Per completezza viene proposta nella soluzione l'individuazione separata dell'errore sul numero delle pedine bianche(3) e sul numero di quelle nere(2).

```
public static int statusScacchiera(int scacchiera[][]){
  int
    OkScacchiera=0;
  int
    i,j,opzione,contaNeri=0,contaBianchi=0;
  for(i=0;(i<scacchiera.length)&&(OkScacchiera==0);i++)
    for(j=0;(j<scacchiera[0].length)&&(OkScacchiera==0);j++)
      if((((i+j)%2==0)&& scacchiera[i][j]!=0))
        OkScacchiera=1;
      else{
        if(scacchiera[i][j]==1)
          contaNeri++;
        if(scacchiera[i][j]==2)
          contaBianchi++;
        if(contaNeri>12)
          OkScacchiera=2;
        if(contaBianchi>12)
          OkScacchiera=3;
      }
  return OkScacchiera;
}
```

Il possible utilizzo all'interno di un programma può essere il seguente:

```
import java.io.*;
import unibs.eco.dmq.basicIO.*;
public class Dama{
  public static void main(String[] args){
  //************************************
  //* Valori della variabile Scacchiera
  //* 0 - Casella vuota
  //* 1 - Pedina Nera
  //* 2 - Pedina Bianca
  //************************************
  int[][]
  scacchiera= {{0,1,0,1,0,1,0,1},
              {1,0,1,0,1,0,1,0},
              {0,1,0,1,0,1,0,1},
              {0,0,0,0,0,0,0,0},
              {0,0,0,0,0,0,0,0},
```

```
            {2,0,2,0,2,0,2,0},
            {0,2,0,2,0,2,0,2},
            {2,0,2,0,2,0,2,0}};
//************************************
//* Valori della variabile OkScacchiera
//* 0 - Scacchiera OK
//* 1 - Pedine in posizione scorretta
//* 2 - Numero errato pedine nere
//* 3 - Numero errato pedine bianche
//************************************
int
  OkScacchiera=0;
int
  i,j,opzione,contaNeri=0,contaBianchi=0;
do{
  Scrittore.video.println("----------------------");
  Scrittore.video.println("1* Visualizzazione scacchiera");
  Scrittore.video.println("2* Status Tastiera");
  Scrittore.video.println("0* USCITA");
  Scrittore.video.println("----------------------");
  opzione=Lettore.tastiera.leggiInt();
    switch(opzione){
      case 1:
        Scrittore.video.println("----------------------");
        Scrittore.video.println("0 - Casella vuota");
        Scrittore.video.println("1 - Pedina Nera");
        Scrittore.video.println("2 - Pedina Bianca");
        Scrittore.video.println("----------------------");
        // Visualizzazione scacchiera
        for(i=0;i<scacchiera.length;i++){
          for(j=0;j<scacchiera[0].length;j++)
            Scrittore.video.print(" "+scacchiera[i][j]);
            Scrittore.video.println(" ");
        }
        Scrittore.video.println("----------------------");
        break;
      case 2:
        OkScacchiera=statusScacchiera(scacchiera);
        switch(OkScacchiera){
          case 0:
            Scrittore.video.println("Tastiera e pedine corrette");
            break;
          case 1:
            Scrittore.video.println("Pedina in posizione scorretta");
            break;
          case 2:
            Scrittore.video.println("Errore sul numero neri");
            break;
          case 3:
            Scrittore.video.println("Errore sul numero bianchi");
            break;
        }
      break;
    }
  }while(opzione!=0);
}
public static int statusScacchiera(int scacchiera[][]){
```

```
  .......
  }
}
```

C) Oggetti
OBBLIGATORIO per TUTTI

La banca XXYY distribuisce alla sua clientela, per scopi promozionali, dei contenitori di monete.I vari modelli di contenitore sono caratterizzati da un codice modello, una capienza massima e possono essere di colori diversi. Per fini organizzativi e di trasporto il colore è ininfluente, due contenitori quindi sono uguali se hanno codice uguale e capienza massima uguale.

C1) Implementare in Java la classe Raccoglitore (con le proprietà,i metodi get/set e gli altri metodi necessari ::: .)

C2) Implementare in Java la classe TestRaccoglitore che permetta di:

1) Inizializzare due oggetti di tipo Raccoglitore
2) Verificarne l'uguaglianza in base ai criteri definiti dalla Banca
3) Versare una somma nel Raccoglitore uno, versare una somma nel Raccoglitore Due (la giacenza corrente + la somma versata non devono superare la capacità massima)
4) Visualizzare la somma complessiva a disposizione (somma delle giacenze dei due Raccoglitori)

Esempi

```
codice raccoglitore 1
AA01
capienza raccoglitore 1
500
codice raccoglitore 2
AA01
capienza raccoglitore 2
500
--------------------
AA01 Blu 500
AA01 Rosso 500
--------------------
stesso modello strutturale
--------------------
versati 200 in racc.uno
versati 300 in racc. due
--------------------
Somma totale raccolta 500
```

```
codice raccoglitore 1
AA01
capienza raccoglitore 1
400
codice raccoglitore 2
AA02
capienza raccoglitore 2
400
--------------------
AA01 Blu 400
AA02 Rosso 400
--------------------
modello strutturale diverso
--------------------
versati 200 in racc.uno
versati 300 in racc. due
--------------------
Somma totale raccolta 500
```

```
codice raccoglitore 1
AA33
capienza raccoglitore 1
200
codice raccoglitore 2
AA33
capienza raccoglitore 2
200
--------------------
AA33 Blu 200
AA33 Rosso 200
--------------------
stesso modello strutturale
--------------------
versati 200 in racc.uno
versamento di 300 non permesso
--------------------
Somma totale raccolta 200
```

C1) Classe Raccoglitore

```
class Raccoglitore{
  /* Definizione attributi privati classe */
  private String idRaccoglitore;
  private String colore;
  private int saldoMonete;
  private int capienzaMax;
  /* Costruttore con i parametri */
```

```
  Raccoglitore(String idRaccoglitore,String colore,int capienzaMax){
    this.idRaccoglitore=idRaccoglitore;
    this.saldoMonete = 0;
    this.colore=colore;
    this.capienzaMax=capienzaMax;
  }
  /* Metodi */
  public int getSaldoMonete(){
    return saldoMonete;
  }
  public boolean versaMonete (int qtaMonete){
    boolean okVersamento=true;
    if((saldoMonete+qtaMonete)<=capienzaMax)
      this.saldoMonete=this.saldoMonete+qtaMonete;
    else
      okVersamento=false;
    return okVersamento;
  }
  public boolean equals(Object Confronto){
    Raccoglitore altroRaccoglitore=(Raccoglitore)Confronto;
    return this.idRaccoglitore.equals(altroRaccoglitore.idRaccoglitore)
    &&
    this.capienzaMax==altroRaccoglitore.capienzaMax;
  }
  public String toString(){
    return idRaccoglitore+" "+colore+" "+capienzaMax;
  }
}
```

C2) Classe test Raccoglitore

```
import unibs.eco.dmq.basicIO.*;
class TestRaccoglitore{
  public static void main(String[] args){
    String
      codiceRaccoglitore;
    int
      moneteDaVersare,capienzaMax;
    Raccoglitore
      raccoglitoreUno,raccoglitoreDue;
    Scrittore.video.println("codice raccoglitore 1");
    codiceRaccoglitore=Lettore.tastiera.leggiString();
    Scrittore.video.println("capienza raccoglitore 1");
    capienzaMax=Lettore.tastiera.leggiInt();
    raccoglitoreUno=new Raccoglitore(codiceRaccoglitore,"Blu",capienzaMax);
    Scrittore.video.println("codice raccoglitore 2");
    codiceRaccoglitore=Lettore.tastiera.leggiString();
    Scrittore.video.println("capienza raccoglitore 2");
    capienzaMax=Lettore.tastiera.leggiInt();
    raccoglitoreDue=new Raccoglitore(codiceRaccoglitore,"Rosso",capienzaMax);
    // Test uguaglianza modello raccoglitori
    // raccoglitore = quanto id = e capienzamax =
    Scrittore.video.println("--------------------");
    Scrittore.video.println(raccoglitoreUno);
    Scrittore.video.println(raccoglitoreDue);
    Scrittore.video.println("--------------------");
```

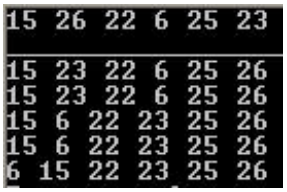
```
    if(raccoglitoreUno.equals(raccoglitoreDue))
      Scrittore.video.println("stesso modello strutturale");
    else
      Scrittore.video.println("modello strutturale diverso");
    Scrittore.video.println("--------------------");
    // Versamento Monete nel raccoglitore uno e due e totale versamenti
    if(!raccoglitoreUno.versaMonete(200))
      Scrittore.video.println("versamento di "+200+" non permesso");
    else
      Scrittore.video.println("versati 200 in racc.uno");
    if(!raccoglitoreDue.versaMonete(300))
      Scrittore.video.println("versamento di "+300+" non permesso");
    else
      Scrittore.video.println("versati 300 in racc. due");\qquad
    Scrittore.video.println("--------------------");
    Scrittore.video.println("Somma totale raccolta "+(raccoglitoreUno.getSaldoMonete()
    +raccoglitoreDue.getSaldoMonete()));
  }
}
```

D) Applicazione di algoritmi
OBBLIGATORIO x TUTTI

Descrivere sinteticamente la logica di funzionamento dell'algoritmo di ordinamento per selezione.
Dato il seguente array:

15;26;22;6;25;23

rappresentare la posizione dei dati all'interno dell'array al termine di ogni iterazione dell'algoritmo



Capitolo 13

Argomenti trattati

- Appello 09/01/07

13.1 Appello 09/01/07

A) Strumenti di analisi
SOLO per chi NON ha superato la prova intermedia

Analizzare la problematica relativa alla determinazione e visualizzazione dei primi N (chiesto come input) numeri della successione di Fibonacci.

Si ricorda che

$$F_0=1$$
$$F_1=1$$
$$F_n=F_{n-1}+F_{n-2}$$

(esempio con N=5 => 1,1,2,3,5)
e *realizzare il FlowChart relativo* (soluzione di tipo iterativo)

B) Algoritmi e codice

B1 Errori sintattici e/o logici
SOLO per chi NON ha superato la prova intermedia

Il seguente programma Java permette la verifica del rispetto dei parametriUE (deficit sotto il 3% del Pil e debito sotto il 60% del Pil) di una serie di nazioni i cui dati sono reperiti da file di testo.

versione da correggere

```
import unibs.eco.dmq.basicIO.*;
public class ParametriUE{
  public static void main(String[] args){
   int numeroNazioni, numeroNazioniElaborate, pilNazione, deficitNazione,debitoNazione;
   dabl inflazioneNazione, mediaInflazioni;
   String nomeNazione;
   boolean okDeficit; okDebito;
   Lettore fileNazioni=new Lettore("parametri.txt");
   mediaInflazioni="0";
   numeroNazioni=fileNazioni.contaRighe();
   numeroNazioniElaborate=0;
   Scrittore.video.println("Nazione   Pil   Deficit   Debito   Inflazione");
   while(numeroNazioniElaborate<!numeroNazioni){
    nomeNazione=fileNazioni.leggiString();
    pilNazione=fileNazioni.leggiInt();
    deficitNazione=fileNazioni.leggiInt();
    debitoNazione=fileNazioni.leggiInt();
    inflazioneNazione=fileNazioni.leggiDouble();
    Scrittore.video.println(nomeNazione+"     "+pilNazione+"     "+deficitNazione
    +"    "+debitoNazione+"    "+inflazioneNazione);
    okDeficit=[deficitNazione<(pilNazione*0.03)];
    okDebito=[debitoNazione<(pilNazione*0.6)];
    Scrittore.video.println("Rispetto Deficit = "-okDeficit+" Rispetto Debito = "-okDebito);
    mediaInflazioni=mediaInflazioni+inflazioneNazione;
    numeroNazioniElaborate++1;
   }
   fileNazioni.chiuditi();
  Scrittore.video.println("Inflazione Media "+(float)mediaInflazioni/numeroNazioniElaborate);
 }
}
```

versione corretta

```
import java.io.*;
import unibs.eco.dmq.basicIO.*;
public class ParametriUE{
  public static void main(String[] args) {
   int numeroNazioni,numeroNazioniElaborate,pilNazione,deficitNazione,debitoNazione;
   double inflazioneNazione,mediaInflazioni;
   String nomeNazione;
   boolean okDeficit,okDebito;
   Lettore fileNazioni=new Lettore("parametri.txt");
   mediaInflazioni=0;
   numeroNazioni=fileNazioni.contaRighe();
    numeroNazioniElaborate=0;
    Scrittore.video.println("Nazione   Pil   Deficit   Debito   Inflazione");
    while(numeroNazioniElaborate<numeroNazioni){
     nomeNazione=fileNazioni.leggiString();
     pilNazione=fileNazioni.leggiInt();
     deficitNazione=fileNazioni.leggiInt();
     debitoNazione=fileNazioni.leggiInt();
     inflazioneNazione=fileNazioni.leggiDouble();
     Scrittore.video.println(nomeNazione+"     "+pilNazione+"     "+deficitNazione
     +"    "+debitoNazione+"    "+inflazioneNazione);
     okDeficit=(deficitNazione<(pilNazione*0.03));
     okDebito=(debitoNazione<(pilNazione*0.6));
     Scrittore.video.println("Rispetto Deficit = "+okDeficit+" Rispetto Debito = "+okDebito);
     mediaInflazioni=mediaInflazioni+inflazioneNazione;
     numeroNazioniElaborate++;
    }
    fileNazioni.chiudi();
    Scrittore.video.println("Inflazione Media "+(float)mediaInflazioni/numeroNazioniElaborate);
  }
}
```

B2 Algoritmi classici
a) OBBLIGATORIO per tutti

Il CED della Banca Sgancia SPA fornisce annualmente su file di testo i dati del fatturato delle singole filiali per ogni trimestre.Il file di testo è organizzato come segue:

NomeFiliale FatturatoTrim1 FatturatoTrim2 FatturatoTrim3 FatturatoTrim4

Es.

brescia 100 200 100 200
como 300 200 100 500
verona 100 100 200 100
milano 120 150 200 180

Il programma di gestione è organizzato a menu. Viene richiesto il completamento del metodo relativo al caricamento dei dati delle trimestrali da file in un vettore, fornendo come parametri d'ingresso il nome della filiale ed il nome del file archivio.

Vengono riportate:

variabili principali:

```
 int []datiTrimestrale=null;
 String nomeFiliale=null, nomeFileIn="in.txt";
```

istruzioni di chiamata del metodo caricaDati dal menu

```
Scrittore.video.println("Nome Filiale ?? ");
nomeFiliale=Lettore.tastiera.leggiString();
```

```
datiTrimestrale=caricaDati(nomeFiliale,nomeFileIn);
 if(datiTrimestrale==null)
   Scrittore.video.println("Filiale non gestita ");
 else{
   Scrittore.video.println("Dati caricati");
   insOK=true;
 }
```

Completare il metodo caricaDati

```
public static int[] caricaDati(String nomeFiliale,String nomeFileIn){
..COMPLETARE
}
```

possibile implementazione

```
 public static int[] caricaDati(String nomeFiliale,String nomeFileIn){
  int i,j,nrFiliali;
  String nomeFilialeFile=null;
  int[] dati;
  boolean trovato=false;
  dati=new int[4];
  Lettore fileMovimentiLeggi;
  fileMovimentiLeggi=new Lettore(nomeFileIn);
  nrFiliali=fileMovimentiLeggi.contaRighe();
  for(i=0;i<nrFiliali&&trovato==false;i++){
    nomeFilialeFile=fileMovimentiLeggi.leggiString();
    if(nomeFiliale.equals(nomeFilialeFile))
      trovato=true;
      for(j=0;j<4;j++)
        dati[j]=fileMovimentiLeggi.leggiInt();
  }
  fileMovimentiLeggi.chiudi();
  if (trovato) return dati;
  else return null;
}
```

Esempio

```
1* Caricamento dati
2* Visualizzazione dati
----------------------
1
Nome Filiale ??
como
Dati caricati
----------------------
1* Caricamento dati
2* Visualizzazione dati
----------------------
2
Filiale :como
Trim: 0 Fatturato :300
Trim: 1 Fatturato :200
Trim: 2 Fatturato :100
Trim: 3 Fatturato :500
```

```
1* Caricamento dati
2* Visualizzazione dati
----------------------
1
Nome Filiale ??
perugia
Filiale non gestita
```

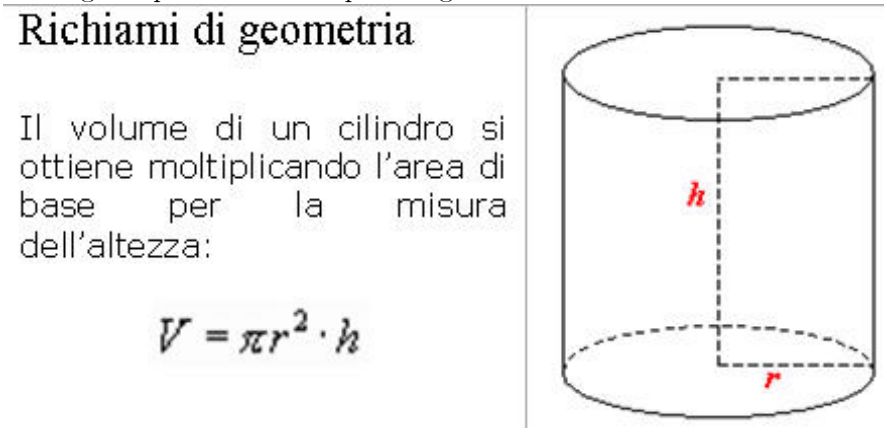
b) SOLO x chi NON ha superato la prova intermedia

Scrivere il metodo fatturato medio che restituisce il valore del fatturato medio prendendo come parametro d'ingresso il vettore datiTrimestre caricato nelmetodo precedente.

C) Oggetti

OBBLIGATORIO per TUTTI

La Società Agricoli Snc è una grossa azienda agricola che utilizza per lo stoccaggio dei prodotti dei silos (magazzini) di forma cilindrica. Al fine della capacità di immagazzinamento, per il software gestionale, due silos sono uguali quando hanno capacità uguale.



C1)

Implementare in Java la classe Silos (con le proprietà,i metodi get/set, e gli altri metodi necessari es l'equals, il toString ..). I comportamenti del Silos sono il carico (la giacenza corrente sommata alla qta che si vuol caricare deve rispettare la capienza) e lo scarico (posso togliere materiale solo se ho materiale a sufficienza per soddisfare la richiesta). Il magazzino prevede come attributo anche la località di ubicazione

possibile implementazione

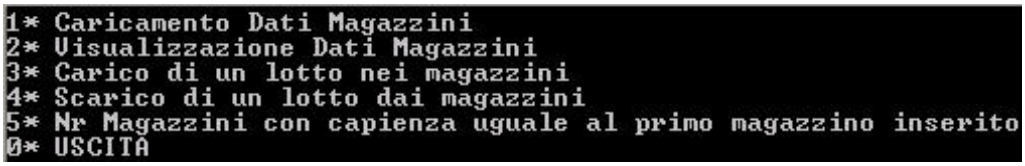
```
class Silos{
  private String ubicazione;
  private double giacenza;
  private double raggio;
  private double altezza;
  Silos(String ubicazione, double giacenza,double raggio,double altezza){
    this.ubicazione=ubicazione;
    this.giacenza=giacenza;
    this.raggio=raggio;
    this.altezza=altezza;
  }
  public double getGiacenza(){
    return giacenza;
  }
  public boolean entrata(double qta){
    boolean eseguito;
    eseguito=getCapienza()>=qta+giacenza;
    if (eseguito)
      this.giacenza = giacenza+qta;
    return eseguito;
  }
  public boolean uscita(double qta){
    boolean eseguito;
    eseguito=giacenza-qta>=0;
    if (eseguito)
      this.giacenza = giacenza-qta;
    return eseguito;
}
private double getCapienza(){
  return 3.1416*raggio*raggio*altezza;
}
public boolean equals(Object Confronto){
  Silos altroSilos=(Silos)Confronto;
  return  this.getCapienza()==altroSilos.getCapienza();
```

```
}
public String toString(){
  return ubicazione+" r "+raggio+" h  "+altezza+" Giac  "+giacenza+" Cap  "+getCapienza();
}
}
```

C2)

Implementare in Java la classe TestSilos organizzata a metodi che risponda alle seguenti richieste (L'esempio che segue è con vettore di 3 Magazzini di tipo Silos)

Menu generale dell'applicazione



C2.1) Caricare i dati nei magazzini

Prototipo x caricare i dati nel magazzini:

```
public static Silos caricamentoMagazzino(){
..COMPLETARE
}
```

possibile implementazione

```
public static Silos caricamentoMagazzino(){
  Silos magazzino;
  String ubicazione;
  double altezza,raggio,giacenza;
  Scrittore.video.println("Ubicazione");
  ubicazione=Lettore.tastiera.leggiString();
  Scrittore.video.println("Altezza");
  altezza=Lettore.tastiera.leggiDouble();
  Scrittore.video.println("Raggio");
  raggio=Lettore.tastiera.leggiDouble();
  Scrittore.video.println("Giacenza iniziale");
  giacenza=Lettore.tastiera.leggiDouble();
  magazzino=new Silos(ubicazione,giacenza,raggio,altezza);
  return magazzino;
}
```

Che viene chiamato in questo modo dal programma di gestione:

```
for (i=0;i<NR_MAGAZZINI;i++)
  magazzini[i]=caricamentoMagazzino();
```

Esempio

```
1
Ubicazione
brescia
Altezza
50
Raggio
100
Giacenza iniziale
0
Ubicazione
como
Altezza
100
Raggio
40
Giacenza iniziale
1000
Ubicazione
bergamo
Altezza
200
Raggio
50
Giacenza iniziale
5000
```

C2.2) Visualizzazione dei dati partendo dall'array caricato
(Cap è la capienza di ogni magazzino)
Prototipo x visualizzare i dati nei magazzini

```
public static void visualizzaMagazzini(Silos [] magazzini){
..COMPLETARE
}
```

possibile implementazione

```
public static void visualizzaMagazzini(Silos [] magazzini){
  int i;
  for (i=0;i<magazzini.length;i++)
    Scrittore.video.println(magazzini[i]);
  return;
}
```

Esempio

```
brescia r 100.0 h  50.0 Giac  0.0 Cap  1570799.9999999998
como r 40.0 h  100.0 Giac  1000.0 Cap  502656.00000000006
bergamo r 50.0 h  200.0 Giac  5000.0 Cap  1570799.9999999998
```

C2.3) Stabilita una quantità che deve essere caricata in ogni magazzino fare le operazioni di carico
(evidenziando il buon esitto dell'operazione)
Prototipo x caricare i magazzini

```
public static void caricaMagazzini(Silos [] magazzini, double qtaMovimentata){
..COMPLETARE
}
```

possibile implementazione

```
public static void caricaMagazzini(Silos [] magazzini, double qtaMovimentata){
  int i;
  for (i=0;i<magazzini.length;i++)
    if(magazzini[i].entrata(qtaMovimentata))
      Scrittore.video.println("Magazzino "+i+" Caricato");
    else
      Scrittore.video.println("Magazzino "+i+" Non Caricato");
  return;
}
```

Esempio

```
Qta da caricare in tutti i magazzini
1000000
Magazzino 0 Caricato
Magazzino 1 Non Caricato
Magazzino 2 Caricato
```

verifica dell'effettiva esecuzione dell'operazione tramite visualizzazione

```
2
brescia r 100.0 h  50.0 Giac  1000000.0 Cap  1570799.9999999998
como r 40.0 h  100.0 Giac  1000.0 Cap  502656.000000000006
bergamo r 50.0 h  200.0 Giac  1005000.0 Cap  1570799.9999999998
```

C2.4) Stabilita una quantità che deve essere scaricata da ogni magazzino fare le operazioni di scarico
Prototipo x scaricare i magazzini

```
public static void scaricaMagazzini(Silos [] magazzini, double qtaMovimentata){
..COMPLETARE
}
```

possibile implementazione

```
public static void scaricaMagazzini(Silos [] magazzini, double qtaMovimentata){
  int i;
  for (i=0;i<magazzini.length;i++)
    if(magazzini[i].uscita(qtaMovimentata))
      Scrittore.video.println("Magazzino "+i+" Scaricato ");
    else
      Scrittore.video.println("Magazzino "+i+" Non Scaricato ");
    return;
}
```

Esempio

```
Qta da prelevare da tutti i magazzini
5000
Magazzino 0 Scaricato
Magazzino 1 Non Scaricato
Magazzino 2 Scaricato
```

verifica dell'effettiva esecuzione dell'operazione tramite visualizzazione

```
2
brescia r 100.0 h  50.0 Giac  995000.0 Cap  1570799.9999999998
como r 40.0 h  100.0 Giac  1000.0 Cap  502656.000000000006
bergamo r 50.0 h  200.0 Giac  1000000.0 Cap  1570799.9999999998
```

C2.5) Determinare il numero di magazzini uguali (capienza uguale) al primo inserito
Prototipo x verificare l'uguaglianza

```
public static double contaUguali(Silos [] magazzini){
..COMPLETARE
}
```

possibile implementazione

```
public static double contaUguali(Silos [] magazzini){
  int j;
  int contaUguali=0;
  for (j=1;j<magazzini.length;j++)
    if(magazzini[0].equals(magazzini[j]))
      contaUguali++;
  return contaUguali;
}
```

Esempio

```
5
Magazzini con uguale capienza :1.0
```

D) Interpretazione del codice
OBBLIGATORIO x TUTTI

Considerando il seguente vettore di valori booleani (con DIM_ARRAY = 6) :

```
valoriBooleani[0]=true;
valoriBooleani[1]=false;
valoriBooleani[2]=true;
valoriBooleani[3]=false;
valoriBooleani[4]=false;
valoriBooleani[5]=true;
```

determinare la sequenza dei valori visualizzati dal seguente codice:

```
for(i=0;i<DIM_ARRAY;i++){
   risultato=valoriBooleani[i]||(i%2==0);
   if(risultato)
     Scrittore.video.println(i);
}
```

Risultato:
0,2,4,5

Capitolo 14

Argomenti trattati

- Appello 08/01/08

14.1 Appello 08/01/08

A) Algoritmi e codice

A1 Errori sintattici e/o logici
SOLO per chi NON ha superato la prova intermedia

Individuare e correggere gli errori sintattici e/o logici presenti nel seguente listato java relativo al caricamento, visualizzazione ed ordinamento di un array.

versione da correggere

```
import unibs.eco.dmq.basicIO.*;
public class ArrayOrdinamento{
  public static void main(String[] args) {
  final int DIM_ARRAY==5;
  double [5] quotazioni;
  double tempScambio;
  int i,j,opzione;
  quotazioni=new double[];
  do{
    Scrittore.video.println("----------------------");
    Scrittore.video.println("1* Inserimento dati");
    Scrittore.video.println("2* Visualizzazione dati");
    Scrittore.video.println("3* Ordinamento");
    Scrittore.video.println("0* Uscita");
    Scrittore.video.println("----------------------");
    opzioneLettore.tastiera.leggiInt();
    switch(opzione){
      case uno: for(i=0;i<DIM_ARRAY;i++){
                  Scrittore.video.println("Inserire quotazione "+i);
                  quotazioni[i]=Lettore.tastiera.leggiDouble();
                }
                break;
      case due: for(i=0;i<DIM_ARRAY;i++)
                  Scrittore.video.println(quotazioni[i]);
                break;
      case tre: for(i=0;i<DIM_ARRAY-1;i++)
                  for(j=X+1;j<DIM_ARRAY;j++)
                    if(quotazioni[i]>quotazioni[j]){
                      tempScambio=quotazioni[];
                      quotazioni[]=quotazioni[];
                      quotazioni[]=tempScambio;
                }
                break;
  }
 }while(opzione!=0);
}
}
```

versione corretta

```
import unibs.eco.dmq.basicIO.*;
public class ArrayOrdinamento{
  public static void main(String[] args) {
  final int DIM_ARRAY=5;
  double [] quotazioni;
  double tempScambio;
  int i,j,opzione;
  quotazioni=new double[DIM_ARRAY];
  do{
    Scrittore.video.println("----------------------");
    Scrittore.video.println("1* Inserimento dati");
    Scrittore.video.println("2* Visualizzazione dati");
    Scrittore.video.println("3* Ordinamento");
    Scrittore.video.println("0* Uscita");
    Scrittore.video.println("----------------------");
    opzione=Lettore.tastiera.leggiInt();
      switch(opzione){
        case 1: for(i=0;i<DIM_ARRAY;i++){
                  Scrittore.video.println("Inserire quotazione "+i);
                  quotazioni[i]=Lettore.tastiera.leggiDouble();
                }
                break;
        case 2: for(i=0;i<DIM_ARRAY;i++)
                  Scrittore.video.println(quotazioni[i]);
                break;
        case 3: for(i=0;i<DIM_ARRAY-1;i++)
                  for(j=i+1;j<DIM_ARRAY;j++)
                    if(quotazioni[i]>quotazioni[j]){
                      tempScambio=quotazioni[i];
                      quotazioni[i]=quotazioni[j];
                      quotazioni[j]=tempScambio;
                }
        break;
      }
  }while(opzione!=0);
 }
}
```

A2 Algoritmi classici

SOLO per chi NON ha superato la prova intermedia

L'Ufficio valutario della Banca InvestiSicuro SPA registra periodicamente le movimentazioni valutarie su file di testo. Le valute gestite sono: Sterlina (codice S); Dollaro (codice D); Euro (codice E). Il file di testo è organizzato come segue: CodiceValuta Movimento

Esempio di file:

S 1000
D 1500
E 2500
D 6000
S 1100
S 1200
D 2100
E 1050

Il programma di gestione è organizzato a menu. Viene richiesto il completamento dei metodi relativi alle seguenti operazioni:

1) determinazione della media valutaria. Parametri di input: codice valuta e nome file dati; Output: media dei movimenti per la valuta indicata (con opportuna segnalazione per valute non gestite)

```
public static double mediaValuta(String nomeValuta,String nomeFileIn) {
 .. completare ..
}
```

2) individuazione del codice valuta con valore movimentato maggiore.Parametri di Input: nome file dati; Output: codice valuta con il maggior valore movimentato.

```
public static String maxValuta(String nomeFileIn) {
 .. completare ..
 }
```

possibile implementazione

```
public static double mediaValuta(String nomeValuta,String nomeFileIn){
   String valutaFile;
   int somma=0,i,nrMovimenti,movimentoFile,movimentiOK=0;
   Lettore fileMovimentiLeggi;
   fileMovimentiLeggi=new Lettore(nomeFileIn);
   nrMovimenti=fileMovimentiLeggi.contaRighe();
   for(i=0;i<nrMovimenti;i++){
     valutaFile=fileMovimentiLeggi.leggiString();
     movimentoFile=fileMovimentiLeggi.leggiInt();
     if(valutaFile.equals(nomeValuta)){
       somma=somma+movimentoFile;
       movimentiOK=movimentiOK+1;
     }
   }
   fileMovimentiLeggi.chiudi();
   if (movimentiOK==0) return (-1);
   else return ((double)somma/(double)movimentiOK);
  }

public static String maxValuta(String nomeFileIn){
   String valutaFile,nomeValutaMax=null;
   int massimo=0,i,nrMovimenti,movimentoFile,movimentiOK;
   Lettore fileMovimentiLeggi;
   fileMovimentiLeggi=new Lettore(nomeFileIn);
   nrMovimenti=fileMovimentiLeggi.contaRighe();
   for(i=0;i<nrMovimenti;i++){
     valutaFile=fileMovimentiLeggi.leggiString();
     movimentoFile=fileMovimentiLeggi.leggiInt();
     if(massimo<movimentoFile){
       massimo=movimentoFile;
       nomeValutaMax=valutaFile;
    }
   }
   fileMovimentiLeggi.chiudi();
   return nomeValutaMax;
}
```

Esempio con i dati precedenti

```
------------------------
1* Media Dati Valuta
2* Valuta con movimento valutario massimo
---------------------
1
Nome Valuta ??
E
Valuta: E Media Movimenti: 1775.0
------------------------
1* Media Dati Valuta
2* Valuta con movimento valutario massimo
---------------------
1
Nome Valuta ??
L
Valuta non gestita
------------------------
1* Media Dati Valuta
2* Valuta con movimento valutario massimo
---------------------
2
Valuta con movimento massimo: D
```

B) Oggetti
OBBLIGATORIO per TUTTI

La Fondi&Fondi SPA gestisce delle forme di investimento. I singoli fondi della società sono caratterizzati da:

- Una descrizione del fondo;
- Un ammontare massimo di capitale investibile nel fondo;
- La composizione del fondo, attraverso tre percentuali che rappresentano la quota massima rispetto al totale investibile in: azioni, obbligazioni e titoli di stato.

I fondi sono vincolati per cui non sono permesse operazioni di prelievo. Sono consentite operazioni di versamento nel fondo, specificando il tipo di settore (A-azioni, O-obbligazioni, T-titoli di stato) e l'ammontare desiderato (l'operazione è consentita solo se si rispettano le percentuali fissate altrimenti tutta l'operazione è esclusa con la segnalazione all'utente).

B1)
Implementare in Java tutta la classe Fondo (con le proprietà, get/set, ed i metodi necessari. I comportamenti principali dell'oggetto Fondo sono:
Metodo per movimentare Input: importo e codice investimento (A-azioni, O-obbligazioni, T-titoli di stato), Output: codice di operazione effettuata

```
public boolean movimentoFinanziario(double qta,char cod) {
 .. completare ..
}
```

Metodo per visualizzare il dettaglio fondo

```
public void schedaDettaglio() {
 .. completare ..
}
```

```
Fondo: MIX_FONDO01
Massimo gestito: 10000.0
Massimo azionario: 3000.0 % massima sul totale 30.0
attuale azionario: 1000.0 % attuale azionario 10.0
Massimo obbligaz: 3000.0 % massima sul totale 30.0
attuale obbligaz: 2000.0 % attuale obbligaz 20.0
Massimo titoli: 4000.0 % massima sul totale 40.0
attuale titoli: 1500.0 % attuale titoli 15.0
```

possibile implementazione

```
class Fondo{
 private String descrizione;
 private double massimoFondo;
 private double giaAzi,giaObb,giaTit;
 private double perAzi,perObb,perTit;
 private double maxAzi,maxObb,maxTit;
 Fondo(String descrizione, double massimoFondo,
    double giaAzi,double giaObb,double giaTit,
    double perAzi,double perObb,double perTit)
 {
   this.descrizione=descrizione;
   this.massimoFondo=massimoFondo;
   this.giaAzi=giaAzi;
   this.giaObb=giaObb;
   this.giaTit=giaTit;
   this.perAzi=perAzi;
   this.perObb=perObb;
   this.perTit=perTit;
   maxAzi=massimoFondo*perAzi/100.0;
   maxObb=massimoFondo*perObb/100.0;
   maxTit=massimoFondo*perTit/100.0;
 }
 public double getMassimoFondo(){
   return massimoFondo;
 }
 public double getGiaAzi(){
  return giaAzi;
 }
 public double getGiaObb(){
  return giaObb;
 }
 public double getGiaTit(){
  return giaTit;
 }

 // Codici
 // A - azioni
 // O - obbligazioni
 // T - titoli di stato
 public boolean movimentoFinanziario(double qta,char cod)
 {
   boolean eseguito=false;
   switch (cod){
     case 'A': if (giaAzi+qta<=maxAzi){
                 giaAzi=giaAzi+qta;
                 eseguito=true;
               }
               break;
     case 'O': if (giaObb+qta<=maxObb){
                 giaObb=giaObb+qta;
                 eseguito=true;
               }
               break;
     case 'T': if (giaTit+qta<=maxTit){
```

```
                giaTit=giaTit+qta;
                eseguito=true;
            }
            break;
   }
   return eseguito;
 }

 public void schedaDettaglio()
 {
   System.out.println("Fondo: "+descrizione);
   System.out.println("Massimo gestito: "+massimoFondo);
   System.out.println("Massimo azionario: "+maxAzi+ " % massima sul totale "
   +perAzi);
   System.out.println("attuale azionario: "+giaAzi+ " % attuale azionario "
   +giaAzi/(massimoFondo)*100.0);
   System.out.println("Massimo obbligaz: "+maxObb+ " % massima sul totale "
   +perObb);
   System.out.println("attuale obbligaz: "+giaObb+ " % attuale obbligaz "
   +giaObb/(massimoFondo)*100.0);
   System.out.println("Massimo titoli: "+maxTit+ " % massima sul totale "
   +perTit);
   System.out.println("attuale titoli: "+giaTit+ " % attuale titoli "
   +giaTit/(massimoFondo)*100.0);
   System.out.println("-------------------------------------------------");
 }
}
```

B2)
Implementare in Java i metodi della classe TestFondo per rispondere alle seguenti richieste

```
1* Caricamento Dati Fondi
2* Visualizzazione Schede Fondi
3* Utilizzo percentuale medio dei fondi
0* USCITA
```

B2.1) Caricare i dati nei magazzini
Caricare i dati dei fondi da file di testo in un array di oggetti

```
public static Fondo[] caricaFondi(String nomeFile) {
 .. completare ..
}
```

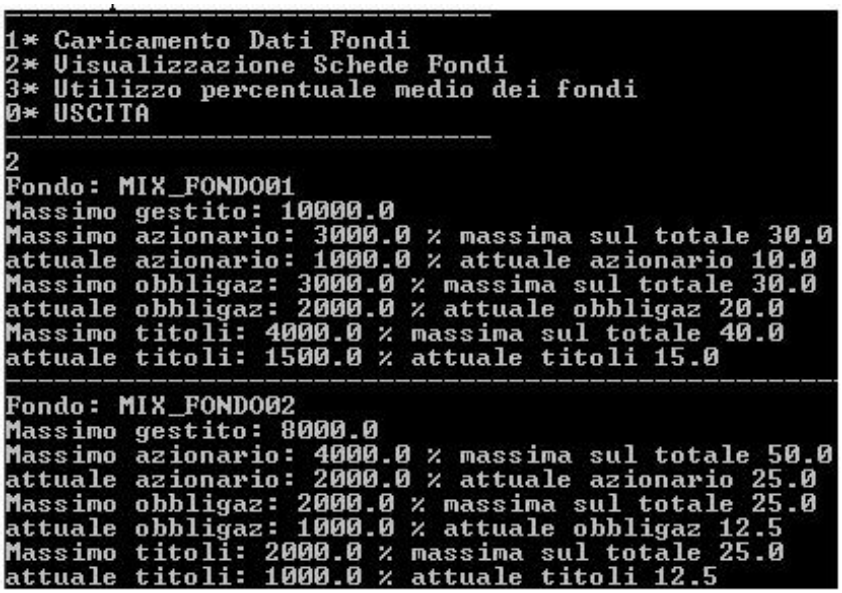
Il file con i dati ha il seguente formato:
Descrizione AmmontareMax GiacCorrAz GiacCorrOb GiacCorrTi PerMaxAz PerMaxOb PerMaxTit
Esempio

MIX_FONDO01 10000 1000 2000 1500 30 30 40
MIX_FONDO02 8000 2000 1000 1000 50 25 25

B2.2) Visualizzare i dettagli dei fondi (utilizzare metodo schedaDettaglio)

```
public static void visualizzaDettaglio(Fondo [] fondi) {
 .. completare ..
}
```

Esempio:



B2.3) Determinare l'utilizzo percentuale medio dei fondi gestiti dalla Fondi&Fondi

```
public static double perMediaUtilizzo(Fondo [] fondi){
 .. completare ..
}
```

Esempio con i dati precedenti: Primo fondo utilizzo % 10+20+15 =totale 45%
Secondo fondo utilizzo % 25+12,5+12,5= totale 50%
Utilizzo percentuale medio = (45+50)/2 = 47,5%

possibile implementazione

```
public static Fondo[] caricaFondi(String nomeFile){
  Lettore fileFondi;
  fileFondi=new Lettore(nomeFile);
  int nrFondi=fileFondi.contaRighe();
  Fondo [] fondi;
  fondi=new Fondo[nrFondi];
  int opzione,i;
  String descrizione;
  double massimoFondo;
  double giaAzi,giaObb,giaTit;
  double perAzi,perObb,perTit;
  for (i=0;i<nrFondi;i++){
    descrizione=fileFondi.leggiString();
    massimoFondo=fileFondi.leggiDouble();
    giaAzi=fileFondi.leggiDouble();
    giaObb=fileFondi.leggiDouble();
    giaTit=fileFondi.leggiDouble();
    perAzi=fileFondi.leggiDouble();
    perObb=fileFondi.leggiDouble();
    perTit=fileFondi.leggiDouble();
    fondi[i]=new Fondo(descrizione,massimoFondo,
    giaAzi,giaObb,giaTit,
    perAzi,perObb,perTit);
  }
  return fondi;
 }
```

```
public static void visualizzaDettaglio(Fondo [] fondi){
  int i;
  for (i=0;i<fondi.length;i++)
    fondi[i].schedaDettaglio();
  return;
}

public static double perMediaUtilizzo(Fondo [] fondi){
  int i;
  double somma=0;
  double media;
  for (i=0;i<fondi.length;i++)
    somma=somma+(fondi[i].getGiaAzi()+fondi[i].getGiaObb()
    +fondi[i].getGiaTit())/fondi[i].getMassimoFondo();
  return (somma*100.0)/(double)i;
}
```

C) Algoritmi ricorsivi
OBBLIGATORIO x TUTTI

C1 Implementare un metodo double termine(int n) per il calcolo dei termini della successione:

$$a_n = \begin{cases} 1 \text{ se } n = 0; \\ 2 \text{ se } n = 1; \\ 0 \text{ se } n = 2; \\ \dfrac{2a_{n-3} + a_{n-2} + 3a_{n-1}}{6} \text{ se } n \geq 3; \end{cases}$$

possibile implementazione

```
public static double termine(int n){
  switch (n){
    case 0:return 1;
    case 1:return 2;
    case 2:return 0;
    default: return ((2*termine(n-3)+termine(n-2)+3*termine(n-1))/6);
  }
 }
```

C2. Implementare in modo ricorsivo un metodo boolean simmetrico(int[] a, int p, int u) per determinare se la porzione di un array numerico di interi dalla posizione p alla posizione u risulta simmetrica.
Esempi: dato il vettore a={7, 2, 5, 2, 7}
la chiamata simmetrico(a,0,4) restituisce true
la chiamata simmetrico(a,1,3) restituisce true
la chiamata simmetrico(a,1,4) restituisce false
possibile implementazione

```
public static boolean simmetrico(int[]a, int p, int u){
 if(p>=u)
   return true;
 else
   if(a[p]!=a[u])
     return false;
```

```
  else
    return simmetrico(a,p+1,u-1);
}
```

D) Alberi

OBBLIGATORIO x TUTTI

5.A)

Generare un Albero Binario di Ricerca (ABR) inserendo in sequenza le chiavi numeriche:
18, 40, 10, 20, 19, 30, 15, 12, 8.

5.B)

Successivamente stampare le chiavi dei nodi dell'albero nell'ordine prodotto da una visita in post-ordine

Capitolo 15

Argomenti trattati

- Progetto A.P. 2005/2006

15.1 Progetto A.P. 2005/2006

Si vuole gestire attraverso gli strumenti analizzati durante il corso di A.P. la seguente realtà aziendale:

L'azienda Algoritmi S.P.A. svolge attività di compravendita e gestisce la memorizzazione dei movimenti di entrata ed uscita del suo magazzino attraverso un database ACCESS.

Il database è costituito dalle seguenti tabelle:

Movimenti

(che contiene tutti i dettagli di un movimento di magazzino quali: identificativo movimento, prodotto movimentato, quantità movimentata, data movimento, tipo movimento (entrata/uscita), soggetto del movimento, ..)

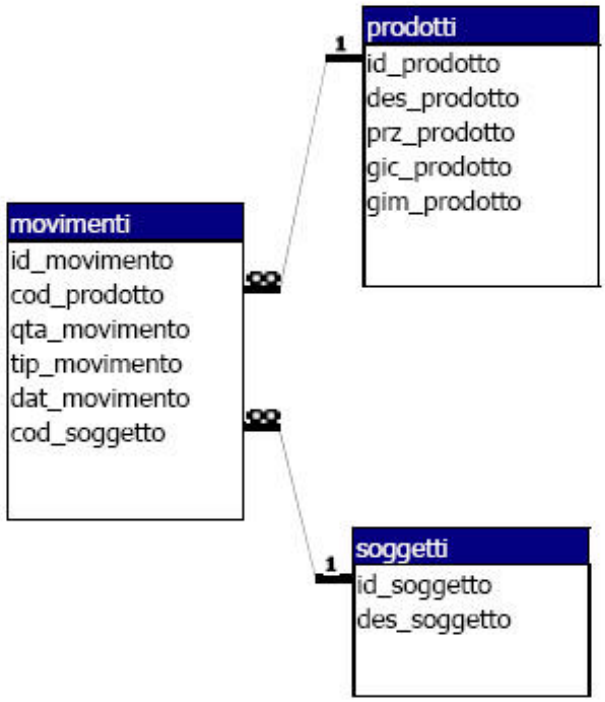
Prodotti

(che contiene tutti i dettagli che caratterizzano il singolo prodotto quali: identificativo prodotto, descrizione prodotto, prezzo unitario, giacenza attuale, giacenza minima)

Soggetti del movimento

(contiente tutti i dati dei clienti / fornitori utili alla società)

relazionate fra di loro nel seguente modo:



Per lo sviluppo la società rende disponibile i files:

magazzino.mdb

che contiene tutte le specifiche delle tabelle magazzino e una serie di dati campione per lo sviluppo dell'applicazione

giacenze.txt

che contiene per una serie di prodotti le giacenze minime indicate dal settore approvvigionamento nel formato: codice prodotto e giacenza minima

Alla società deve essere fornito un mini-gestionale scritto in java, prendendo come riferimento principale l'oggetto Prodotto. Di questo gestionale è possibile fornire tre versioni:

- base
- intermedia
- completa

che non differiscono tra di loro per la complessità ma per il numero di opzioni gestite. Il gestionale nella versione completa comprende anche le opzioni delle versioni intermedia e base, il gestionale nella versione intermedia contiene anche le opzioni della versione base.

Versione BaseVersione Base

Elencare tutti i prodotti gestiti dalla società
visualizzare per ogni prodotto (tramite un metodo sull'oggetto) le componenti principali:
codice Prodotto,descrizione Prodotto,prezzo Prodotto,giacenza Attuale,giacenza Minima

Effettuare un movimento su uno dei prodotti gestiti
deve essere richiesto il codice prodotto e, se questo esiste nell'insieme dei prodotti gestiti, effettuato il movimento generando una riga nella tabella movimenti sul database e aggiornando conseguentemente la giacenza attuale sia dell'oggetto in memoria che sulla tabella prodotti (tramite un metodo sull'oggetto). Il metodo deve controllare, gestire ed eventualmente segnalare le seguenti situazioni:
1) un movimento di uscita che possa rendere negativa la giacenza attuale; (annulla operazione)
2) un movimento di uscita che faccia scendere la giacenza attuale al di sotto della giacenza minima (segnala un avviso)
3) un movimento di uscita/entrata con una valorizzazione finanziaria spropositata (prezzo x quantità) al di sopra di 30.000E; (chiede conferma)
4) un movimento che indichi un soggetto inesistente nella tabella Access soggetti; (annulla operazione).

Segnalazione su file dei prodotti sotto giacenza minima
si vuole produrre per l'ufficio approvvigionamenti un elenco tramite file .txt dei prodotti la cui giacenza Attuale sia inferiore alla giacenza Minima. Nel file vengono riportati: il codice prodotto, la descrizione prodotto e la quantità di merce da reintegrare per rientrare nella giacenza minima

Versione Intermedia

Aggiornamento del campo giacenze tramite file
il file giacenze.txt viene fornito dal settore pianificazione ed inventario e contiene la giacenza minima di una serie di prodotti. In base ai dati di questo file deve essere aggiornato il campo relativo nell'oggetto, sia in memoria che sulla tabella del database (tramite un metodo sull'oggetto). Il file contiene:
codice prodotto e giacenza minima associata

Elenco delle movimentazioni del magazzino in un intervallo temporale
devono essere elencati i dettagli dei movimenti di magazzino, specificando a quali prodotti si riferiscono, compresi fra una data di partenza e una data d'arrivo indicate dall'utente (tramite un metodo sull'oggetto).

Sconto del 20% di tutti i prodotti mai movimentati
il prezzo unitario di tutti i prodotti che non hanno mai subito movimentazioni deve essere scontato del 20%. Deve essere aggiornato sia sul prezzo dell'oggetto contenuto in memoria che la tabella relativa nel database (tramite un metodo sull'oggetto)

Versione Avanzata

Elenco valorizzato movimenti di un prodotto
si deve visualizzare la valorizzazione finanziaria (prezzo prodotto x quantità) dei movimenti di un prodotto il cui codice è scelto dall'utente fra quelli gestiti (tramite un metodo sull'oggetto)

Elenco prodotti e transazioni medie effettuate
si vuole riportare un elenco di tutti i prodotti gestiti indicando il numero di transazioni effettuale e la quantità media intermediata(tramite un metodo sull'oggetto)

Capitolo 16

Argomenti trattati

- Package Microcad
- Classe Avvisi

16.1 Il package Microcad

Il package Microcad contiene delle classi che agevolano la manipolazione grafica di alcune figure semplici e composte. La classe Lavagna fornisce un supporto per disegnare figure bidimensionali. Le proprietà ed i metodi associati alle varie classi sono reperibili nella documentazione del package Le figure rappresentabili sulla lavagna sono tutte quelle che derivano dalla classe Figura del package microCAD. La lavagna utilizza una finestra costruita con dimensioni iniziali 300 X 300 pixel (larghezza X altezza). In seguito può essere ridimensionata.

Alla lavagna è associato un sistema di coordinate cartesiane così definito:

Asse X orientato verso destra

Asse Y orientato verso l'alto

Analizziamo le istruzioni per definire una Lavagna, definire un piano cartesiano collocare un oggetto grafico sulla lavagna

Esempio (ProvaLavagna.java)

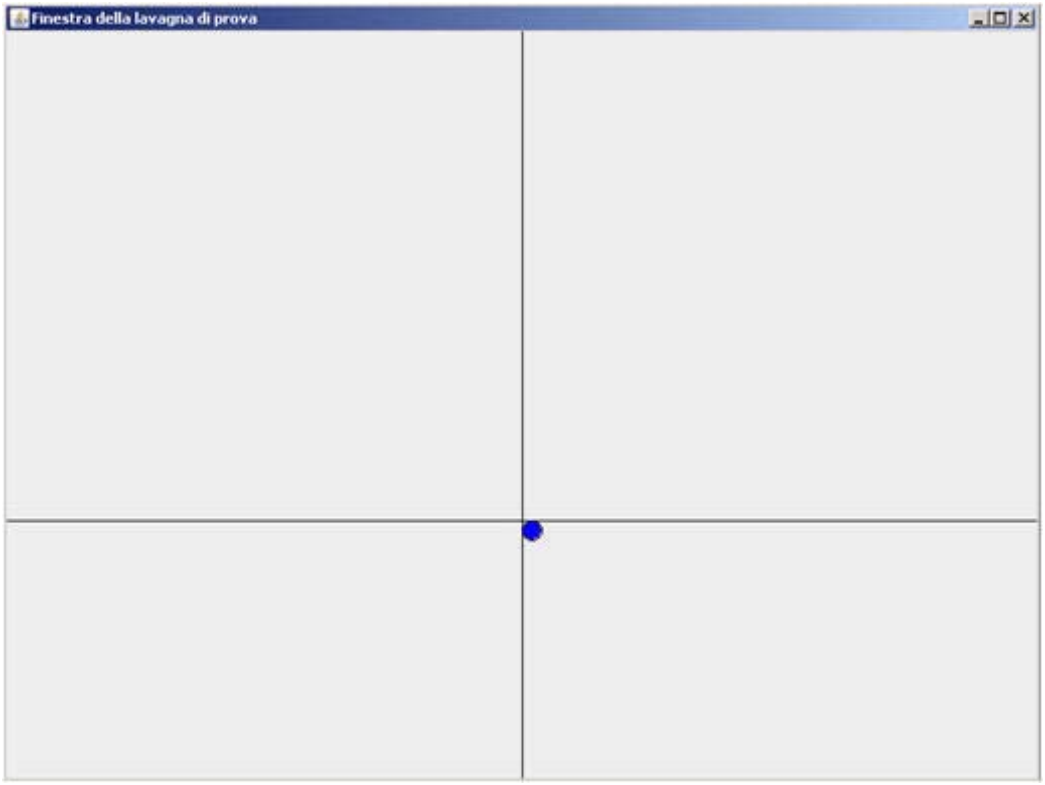
```
package unibs.eco.dmq.esempibase;
import java.awt.Color;
import unibs.eco.dmq.microCAD.Lavagna;
import unibs.eco.dmq.microCAD.Cerchio;
public class ProvaLavagna {
  public static void main(String[] argv) {
    Lavagna lavagna = new Lavagna("Finestra della lavagna di prova");
    lavagna.settaPianoCartesiano(-100, 100, -50);
    Cerchio c=new Cerchio(2);
    lavagna.aggiungi(c,0,0);
 }
}
```

La definizione del piano cartesiano avviene nel seguente modo: nella finestra grafica disponibile ci saranno come x logiche le coordinate che vanno da xmin = -100 a xmax = 100, e come y logiche ymin = -50 e ymax ricavato di conseguenza sulla base della dimensione disponibile

attraverso il metodo

```
lavagna.disegnaAssi();
```

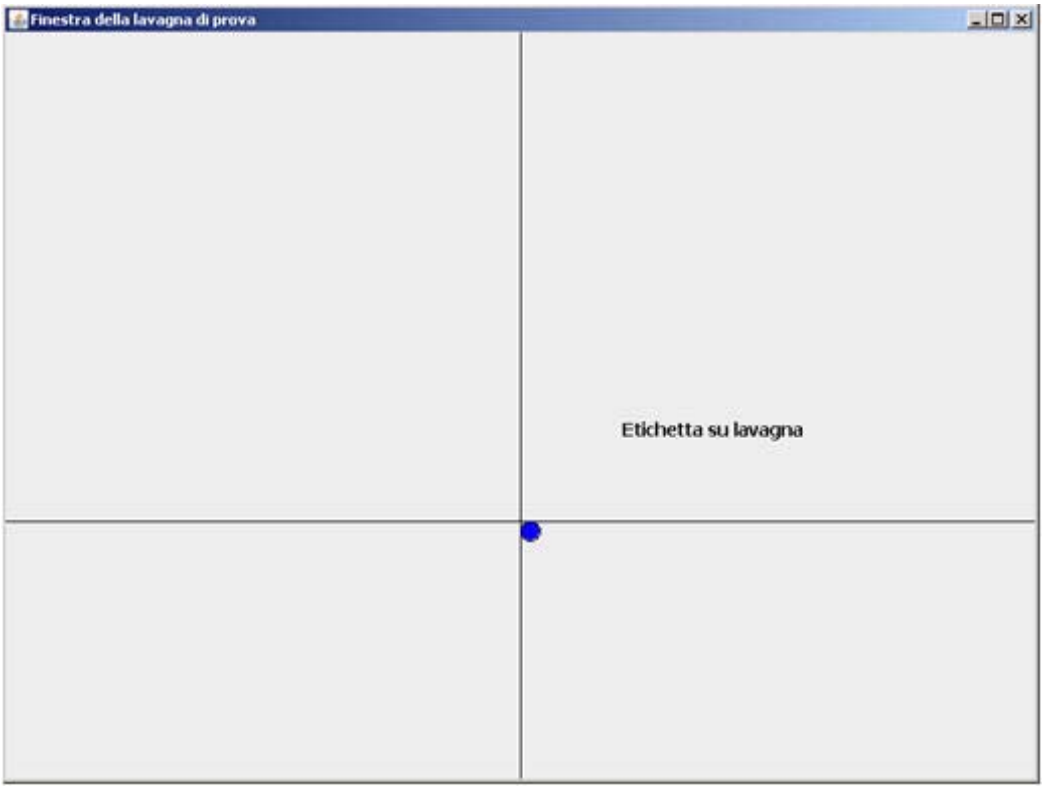
è possibile disegnare gli assi cartesiani di riferimento



il cerchio è posizionato sulla lavagna considerando come posizione di ancoraggio l'angolo superiore sinistro del quadrato virtuale che lo contiene.

è possibile aggiungere alla lavagna delle etichette testuali

```
Etichetta1 e=new Etichetta1("Etichetta su lavagna");
lavagna.aggiungi(e,20,20);
```



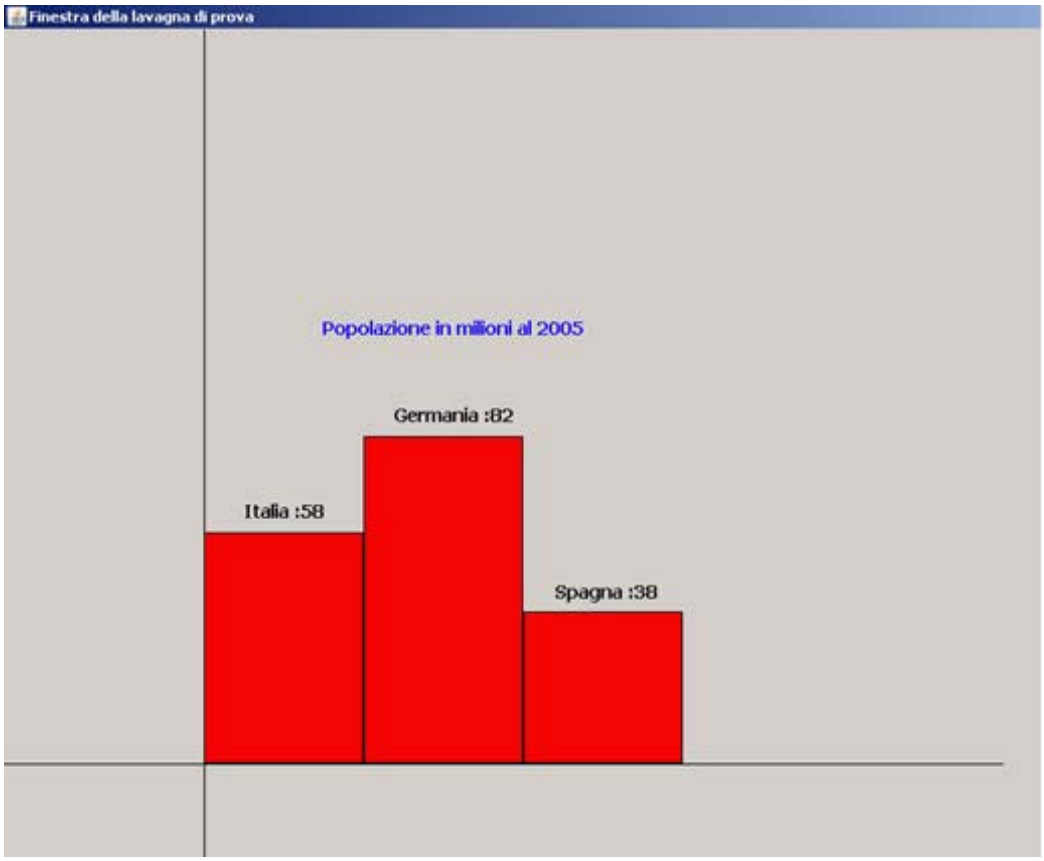
esistono una serie di figure geometriche gestibili per ottenere semplici applicazioni grafiche.

Esempio: attraverso etichette ed oggetti di tipo rettangolo si vuole ottenere una semplice trasposizione grafica dei dati relativi alla popolazione in milioni registrate nel 2005 per le nazioni Italia,Germania,Spagna

Esempio (ProvaFigure.java)

```
import java.awt.Color;
import unibs.eco.dmq.microCAD.*;
public class ProvaFigure{
    public static void main(String[] argv) {
        Lavagna lavagna = new Lavagna("Finestra della lavagna di prova");
        lavagna.settaPianoCartesiano(-50, 200, -50);
        Etichetta1 e_generale=new Etichetta1("Popolazione in milioni al 2005",Color.blue);
        Etichetta1 e_italia=new Etichetta1("Italia :58");
        int a_italia=58;
        Rettangolo r_italia=new Rettangolo(40,a_italia);
        Etichetta1 e_germania=new Etichetta1("Germania :82");
        int a_germania=82;
        Rettangolo r_germania=new Rettangolo(40,a_germania);
        Etichetta1 e_spagna=new Etichetta1("Spagna :38");
        int a_spagna=38;
        Rettangolo r_spagna=new Rettangolo(40,a_spagna);
        lavagna.disegnaAssi();
        lavagna.aggiungi(e_generale,30,a_germania+30);
        lavagna.aggiungi(r_italia,20,a_italia/2);
        lavagna.aggiungi(e_italia,10,a_italia+8);
        lavagna.aggiungi(r_germania,60,a_germania/2);
        lavagna.aggiungi(e_germania,48,a_germania+8);
        lavagna.aggiungi(r_spagna,100,a_spagna/2);
        lavagna.aggiungi(e_spagna,88,a_spagna+8);
   }
}
```

(il punto di ancoraggio della figura rettangolo sulla lavagna è il centro del rettangolo ...)


Finestra della lavagna di prova
Popolazione in milioni al 2005
Germania :82
Italia :58
Spagna :38

16.2 La classe Avvisi

La classe Avvisi, definita all'interno della sezione basicIO, serve a visualizzare dei pannelli grafici con messassi per l'utilizzatore del programma ed a leggere dati in input. Non è necessario creare alcun oggetto per questa classe.

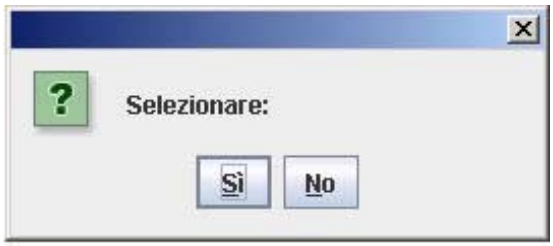
Esempio (ProvaAvvisi.java)

```
package unibs.eco.dmq.esempibase;
import unibs.eco.dmq.basicIO.*;
public class ProvaAvvisi{
 public static void main(String[] argv) {
 Avvisi.mostraMessaggio("avvviso di prova");
 }
}
```

è possibile utilizzare la classe Avvisi anche per chiedere conferme attraverso domande vero/falso

Esempio (ProvaAvvisi1.java)

```
package unibs.eco.dmq.esempibase;
import unibs.eco.dmq.basicIO.*;
public class ProvaAvvisi1{
    public static void main(String[] argv) {
       boolean rispo=Avvisi.chiediConferma("Selezionare:");
       Avvisi.mostraMessaggio("Selezionato: "+rispo);
    }
}
```



Le lettura di dati in input può essere fatta nel seguente modo:

Esempio (ProvaAvvisi2.java)

```
package unibs.eco.dmq.esempibase;
import unibs.eco.dmq.basicIO.*;
public class ProvaAvvisi2{
  public static void main(String[] argv) {
    int num1=Avvisi.leggiInt("Inserire primo numero :");
    int num2=Avvisi.leggiInt("Inserire secondo numero :");
```

```
    Avvisi.mostraMessaggio("Somma : "+(num1+num2));
 }
}
```