Programmazione II

A.A. 2022-23
Prof. Maria Tortorella

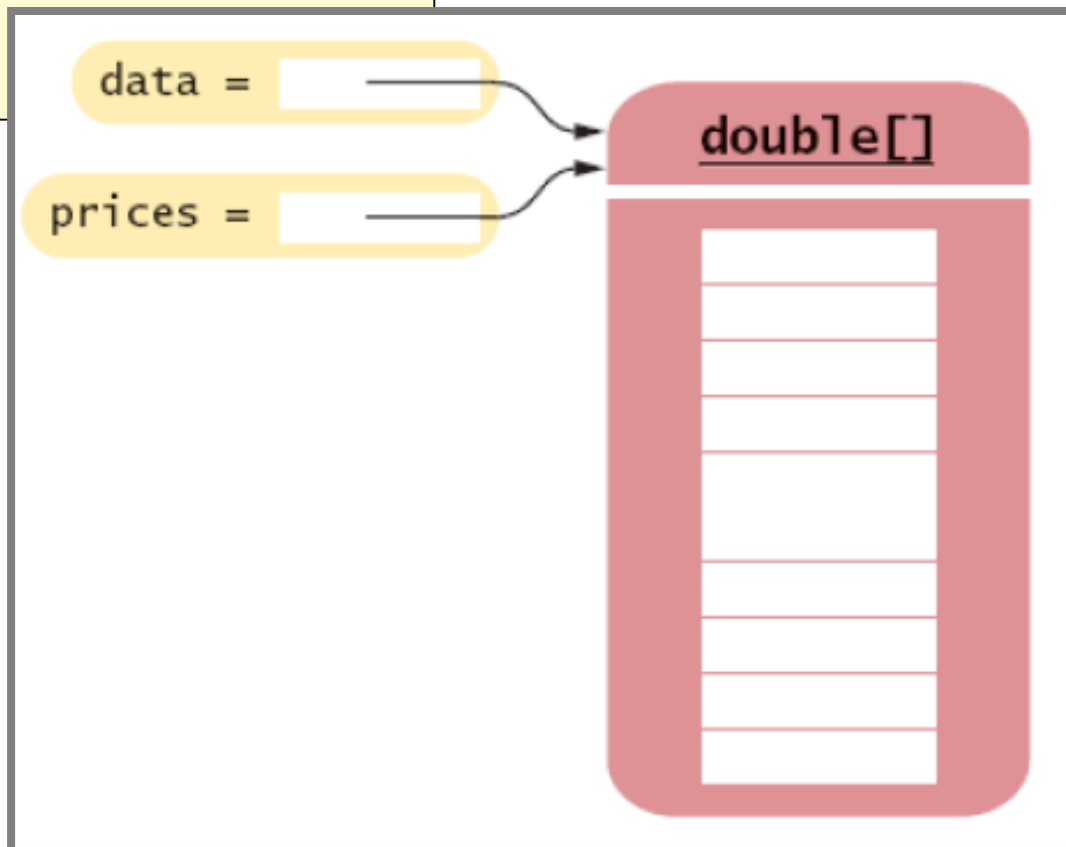
Array ed ArrayList

- Algoritmi fondamentali su array
- Ancora su UML

Copia di array

- Copiare il riferimento ad un array NON comporta la copia dell'array

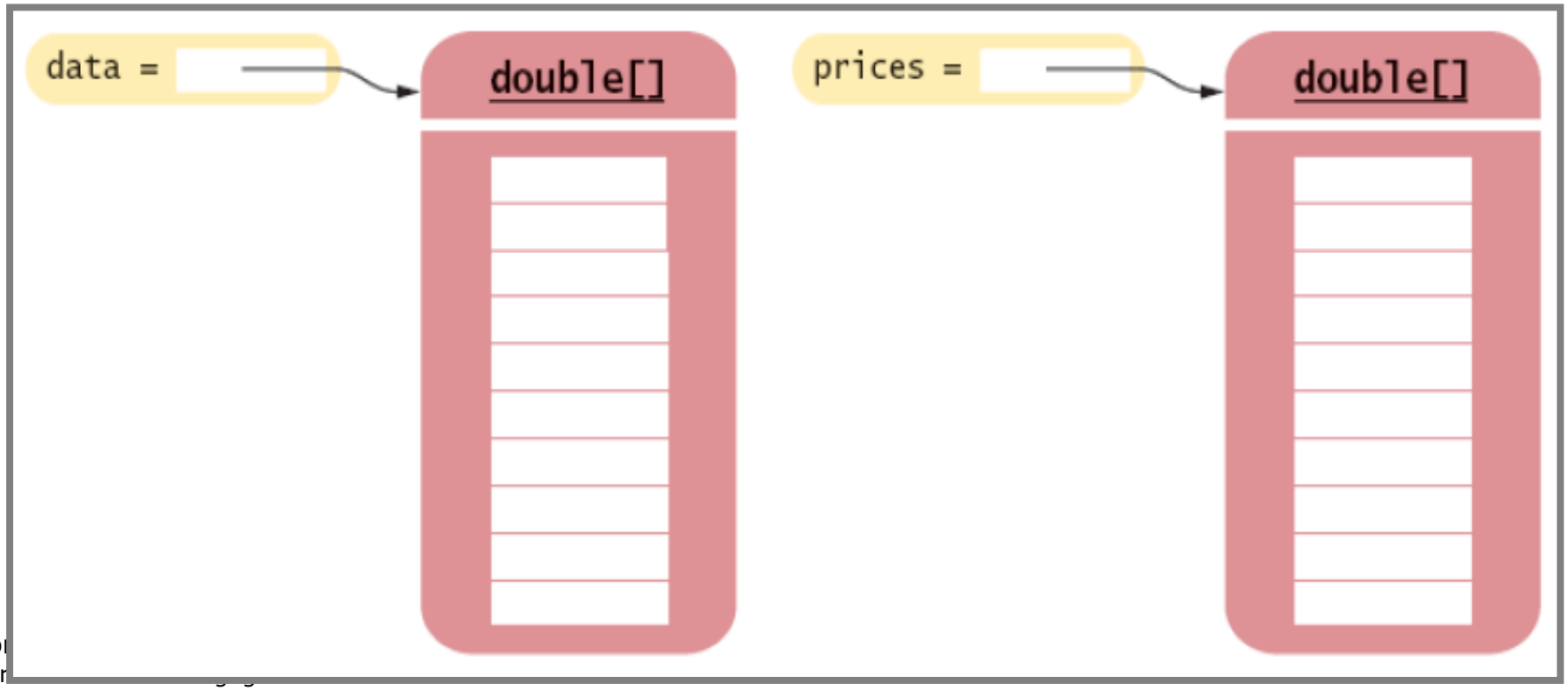
```
double[] data = new double[10];
// fill array . . .
double[] prices = data;
```



Copia mediante clone

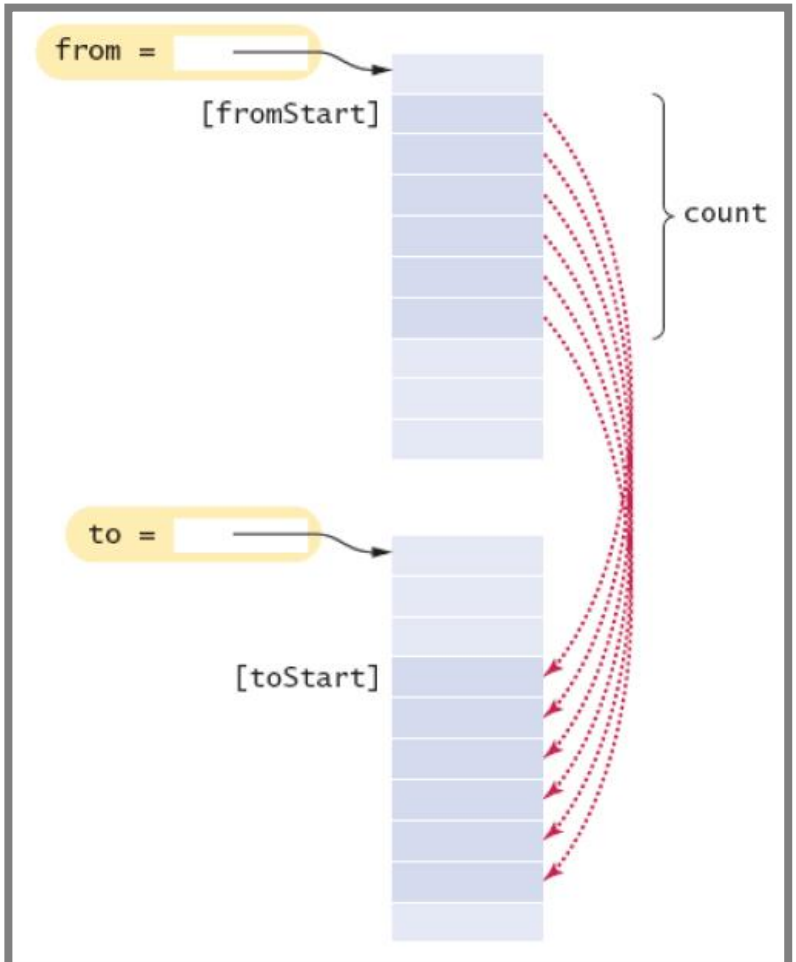
- `clone` consente di fare una copia di una collezione

```
double[] prices = (double[]) data.clone();
```



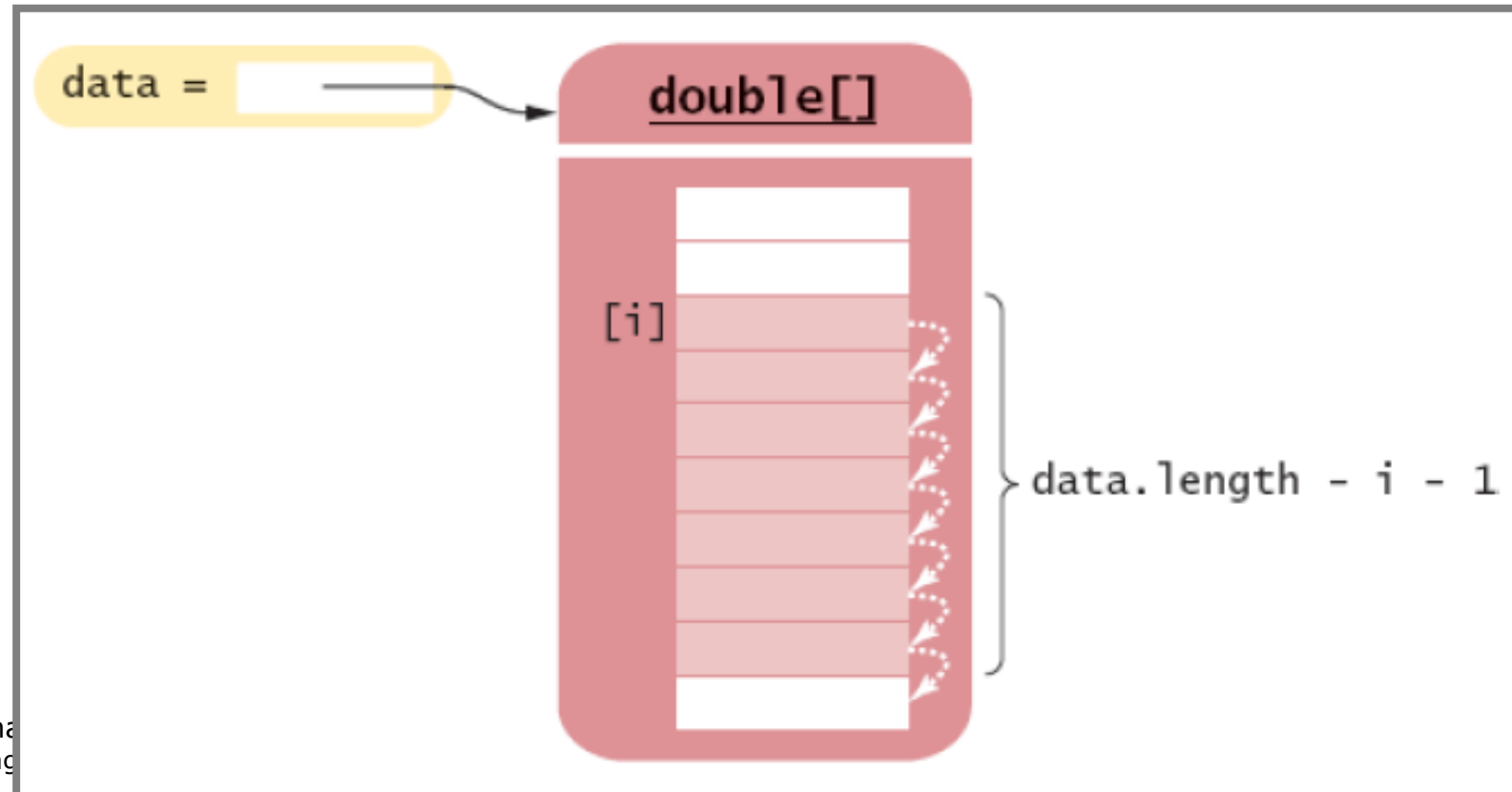
Copiare elementi

```
System.arraycopy(from, fromStart, to, toStart, count);
```



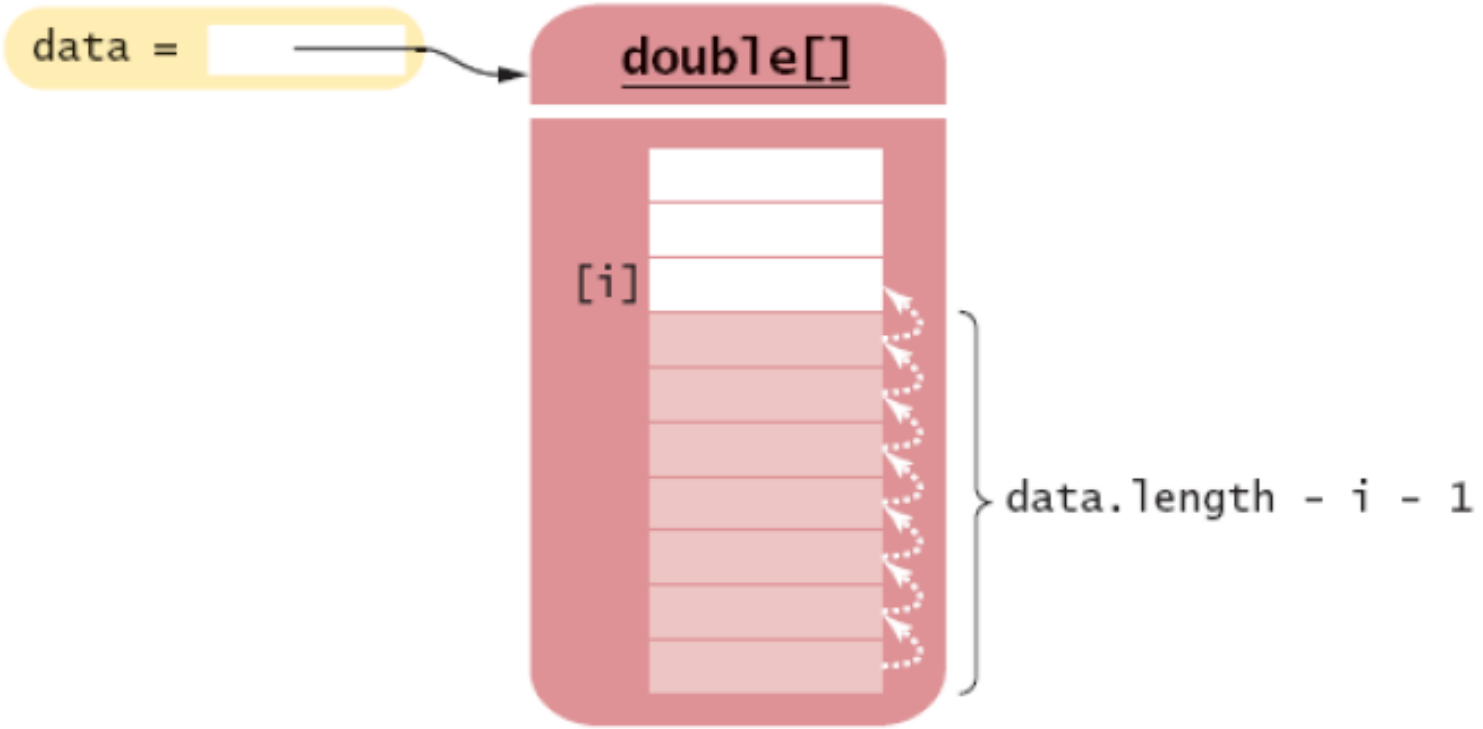
Aggiungere un elemento ad un Array

```
System.arraycopy(data, i, data, i + 1, data.length - i - 1);
data[i] = x;
```



Rimuovere un elemento

```
System.arraycopy(data, i + 1, data, i, data.length - i - 1);
```



Aumentare la dimensione

Creare un nuovo array

```
double[] newData = new double[2 * data.length];
```

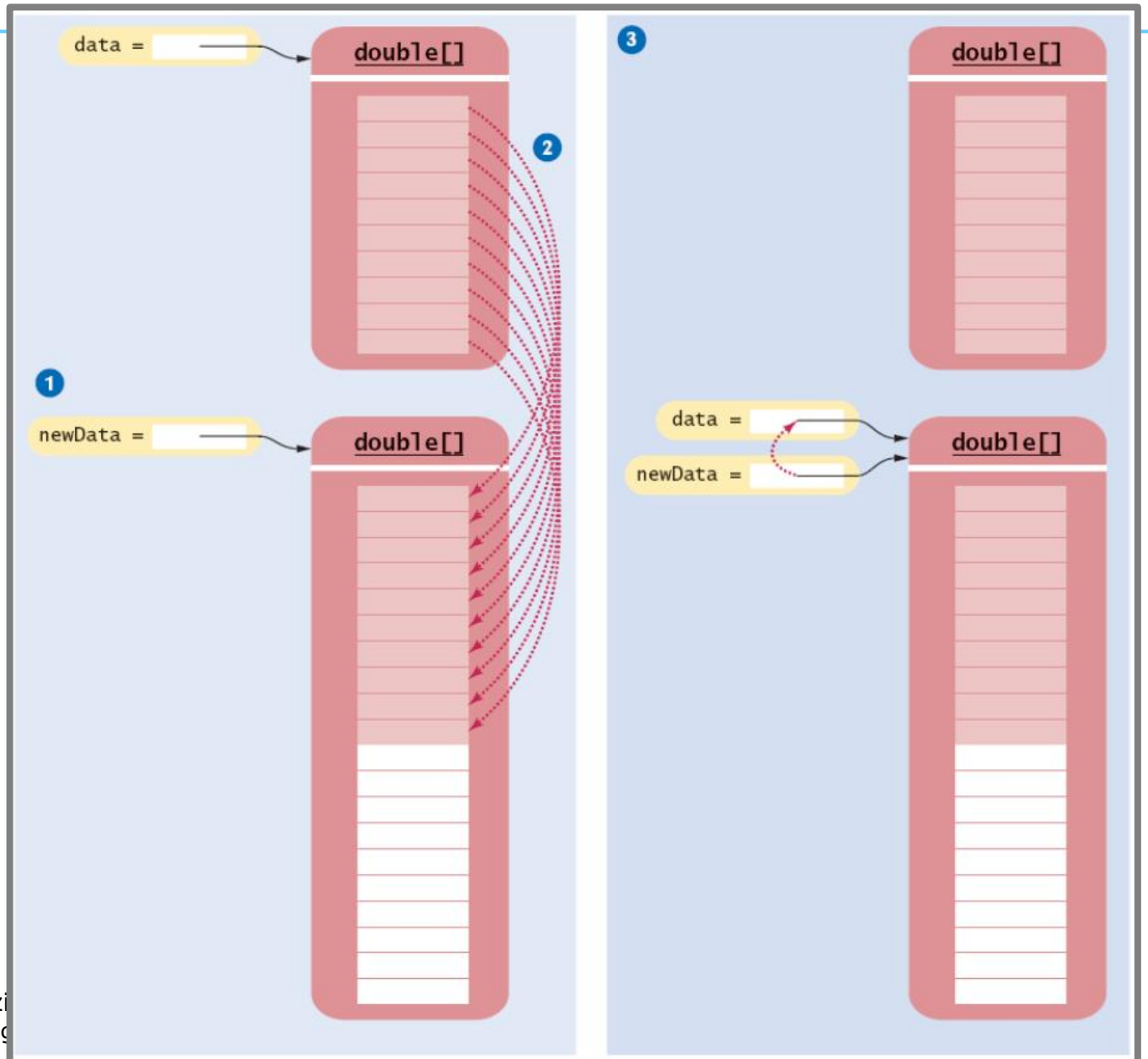
Copiarvi tutti gli elementi

```
System.arraycopy(data, 0, newData, 0, data.length);
```

Memorizzare il riferimento

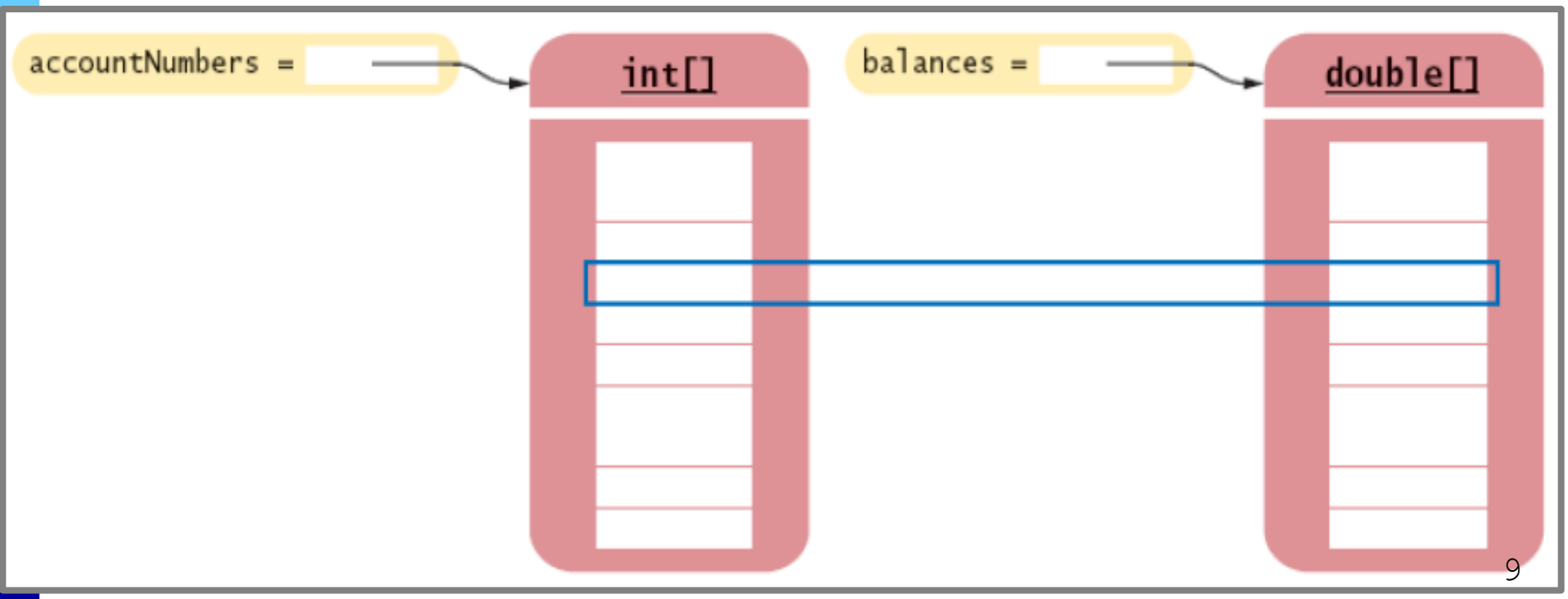
```
data = newData;
```

Aumentare la dimensione



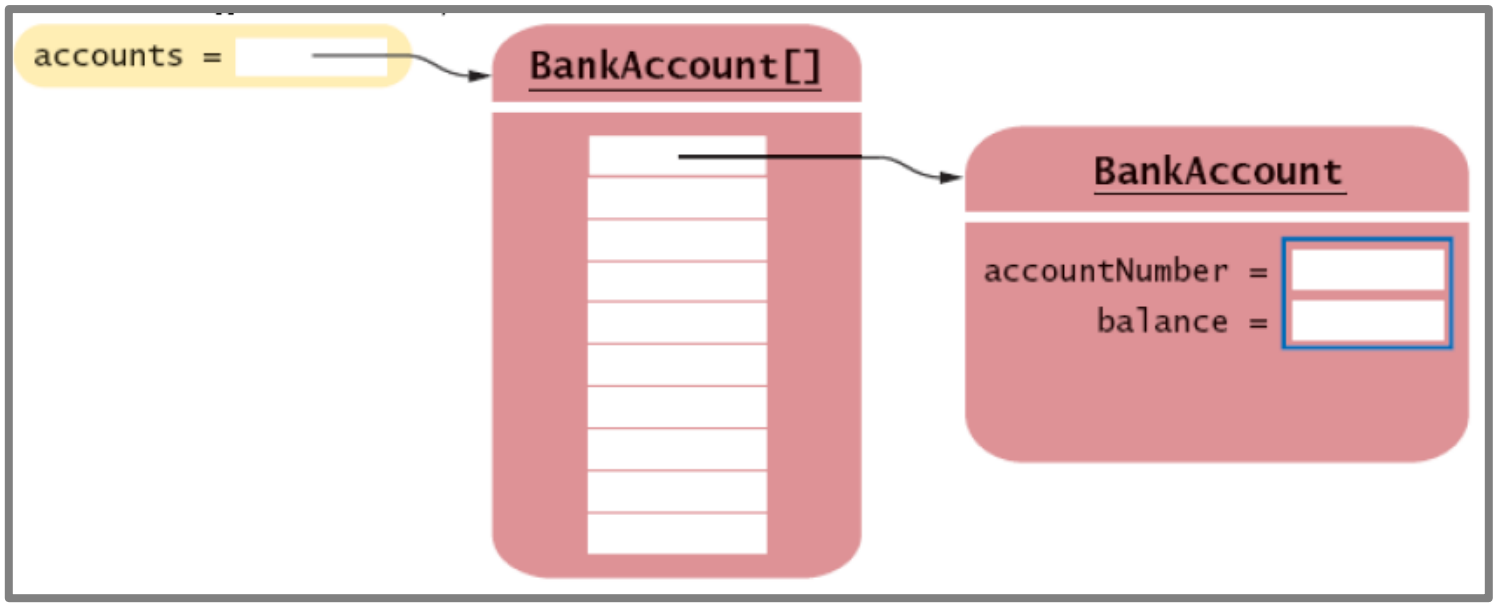
Array paralleli ?

-
```
// Don't do this
int[] accountNumbers;
double[] balances;
```



Array di oggetti !

```
BankAccount[] accounts;
```



Array parzialmente riempiti

- Lunghezza = il numero massimo di elementi in un array
- In genere l'array NON è riempito completamente
- Una seconda variabile memorizza il numero di elementi (riempimento)
- Convenzione:

```
final int DATA_LENGTH = 100;
double[] data = new double[DATA_LENGTH];
int dataSize = 0;
```

Array parzialmente riempiti

- Quando si aggiunge un elemento è necessario incrementare `dataSize`:

```
data[dataSize] = x;
dataSize++;
```

Esercizi

- Implementare di nuovo gli esercizi delle precedenti esercitazioni, creando delle collezioni di oggetti

Esercizio

Modifica al programma precedentemente implementato

Implementare un programma che supporti una segreteria didattica nella gestione delle sedute d'esame di un dato corso di laurea.

Il file "studenti.dat" contiene i dati che riguardano ciascuno studente. Essi sono: nome, cognome, matricola, anno a cui è iscritto, se è fuoricorso o meno. I dati sono memorizzati in base al seguente formato:

matricola	**81700023**
nome studente	**Mario**
cognome studente	**Rossi**
anno d'iscrizione	**2**
fuori corso	**false**
	81700256
	Paolo
	Verdi
	1
	true
	…

Implementare le classi necessarie con l'obiettivo di poter implementare le seguenti funzionalità:

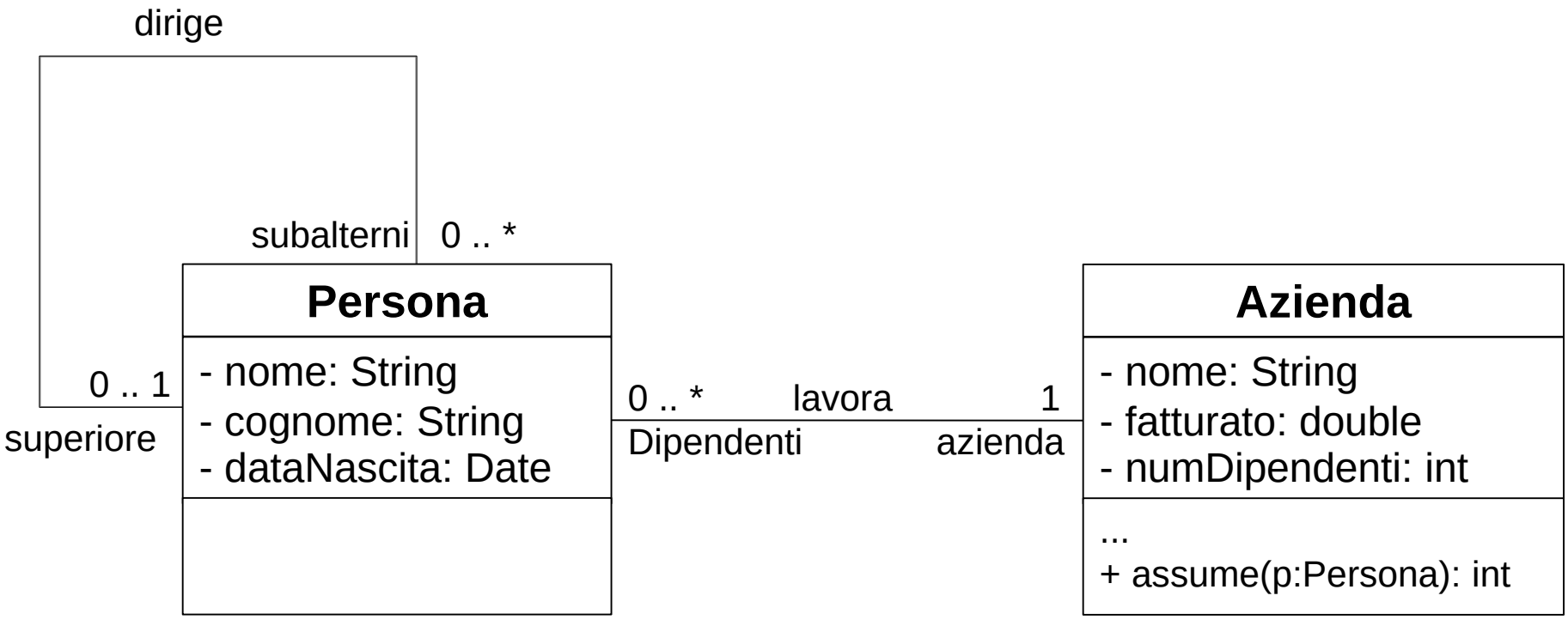
- Listare tutti gli studenti fuori corso
- Listare tutti gli studenti che sono iscritti al terzo anno

Per entrambi, prima cercare, poi listare

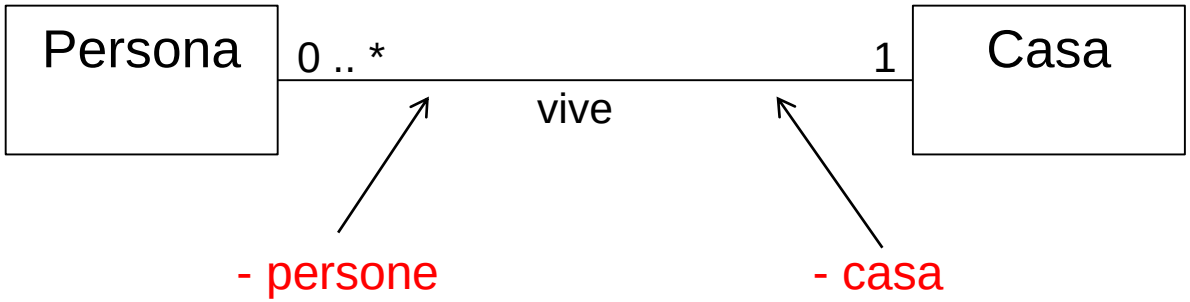
Ancora su UML: le Associazioni

- Un'associazione indica una relazione tra classi
  - ad esempio persona che lavora per azienda
- Un'associazione può avere:
  - un nome (solitamente un verbo)
  - i ruoli svolti dalle classi nell'associazione
- Gli estremi di un'associazione sono "attributi impliciti"
  - hanno visibilità come gli attributi normali
  - hanno una molteplicità
    - 1, 0..1, 1..*, 4, 6-12

Diagramma delle classi



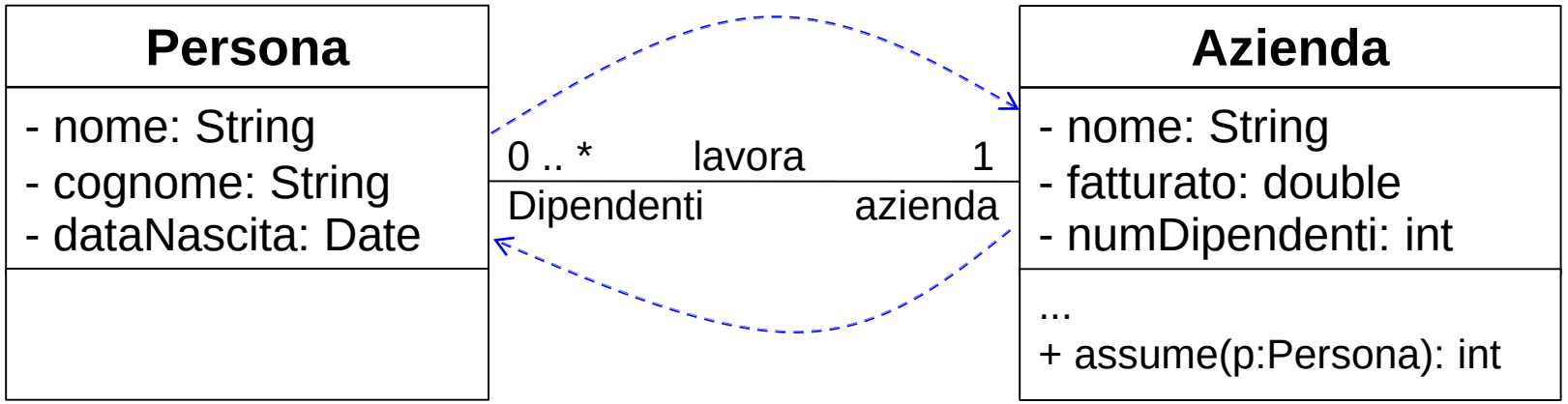
Esempio




```
class Persona {
   …
   private Casa casa;
   …
}
```

```
class Casa{
   ...
   private Persona[] persone;
   ...
}
```

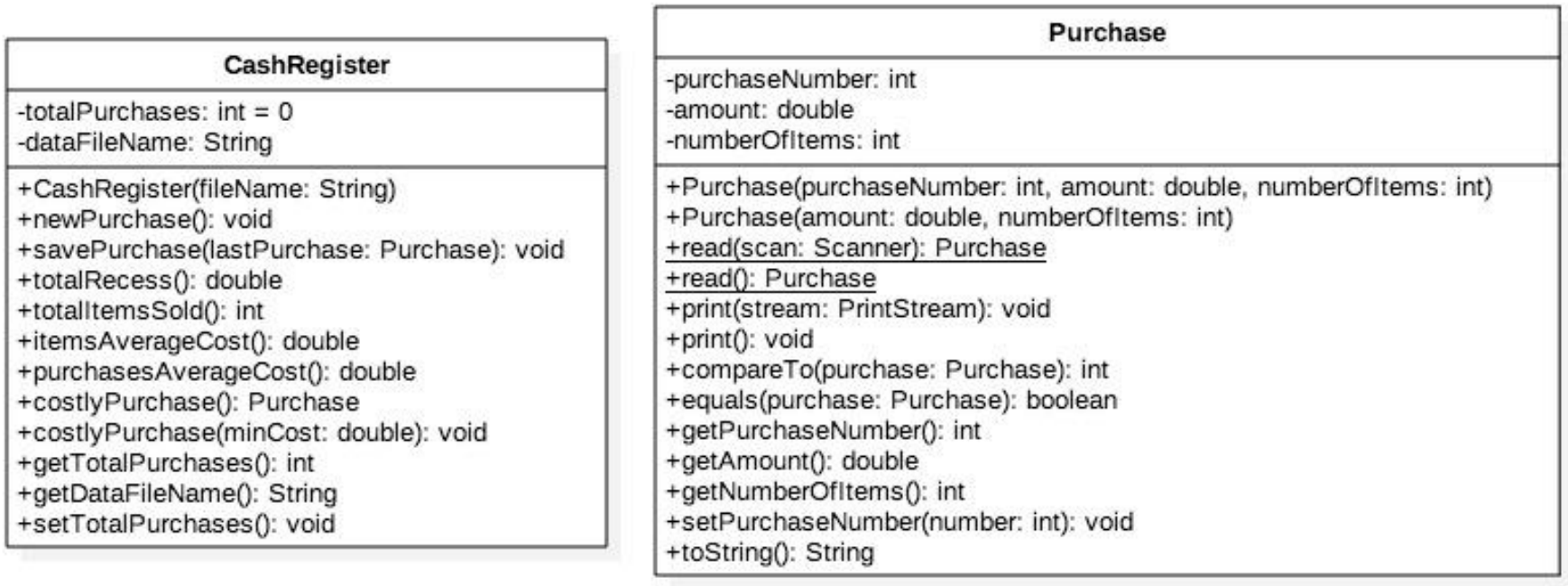
Nel diagramma delle classi UML



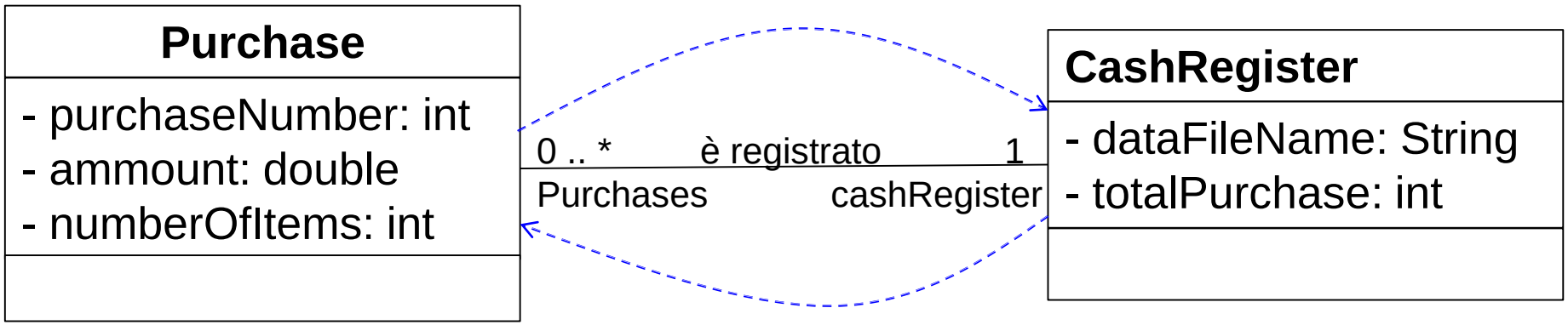

```
class Persona {
   // Methods
   ...
   //Instance variables
   private String nome, cognome;
   private Date dataNascita;
   private Azienda azienda;
}
```

```
class Azienda{
   // Methods
   ...
   //Instance variables
   private String nome;
   private double fatturato;
   private int numDipendenti;
   private Persona[] Impiegati;
}
```

Esempio CashRegister



Esempio CashRegister




```
class Purchase{
   // Methods
   ...
   //Instance variables
   private int purchaseNumber;
   private double ammount;
   private int numberOfItems;
   CashRegister cashRegister;
}
```

```
class CashRegister{
// Methods
   ...
   //Instance variables
   private String dataFileName;
   private int totalPurchase;
   private Purchase[] purchases;
}
```

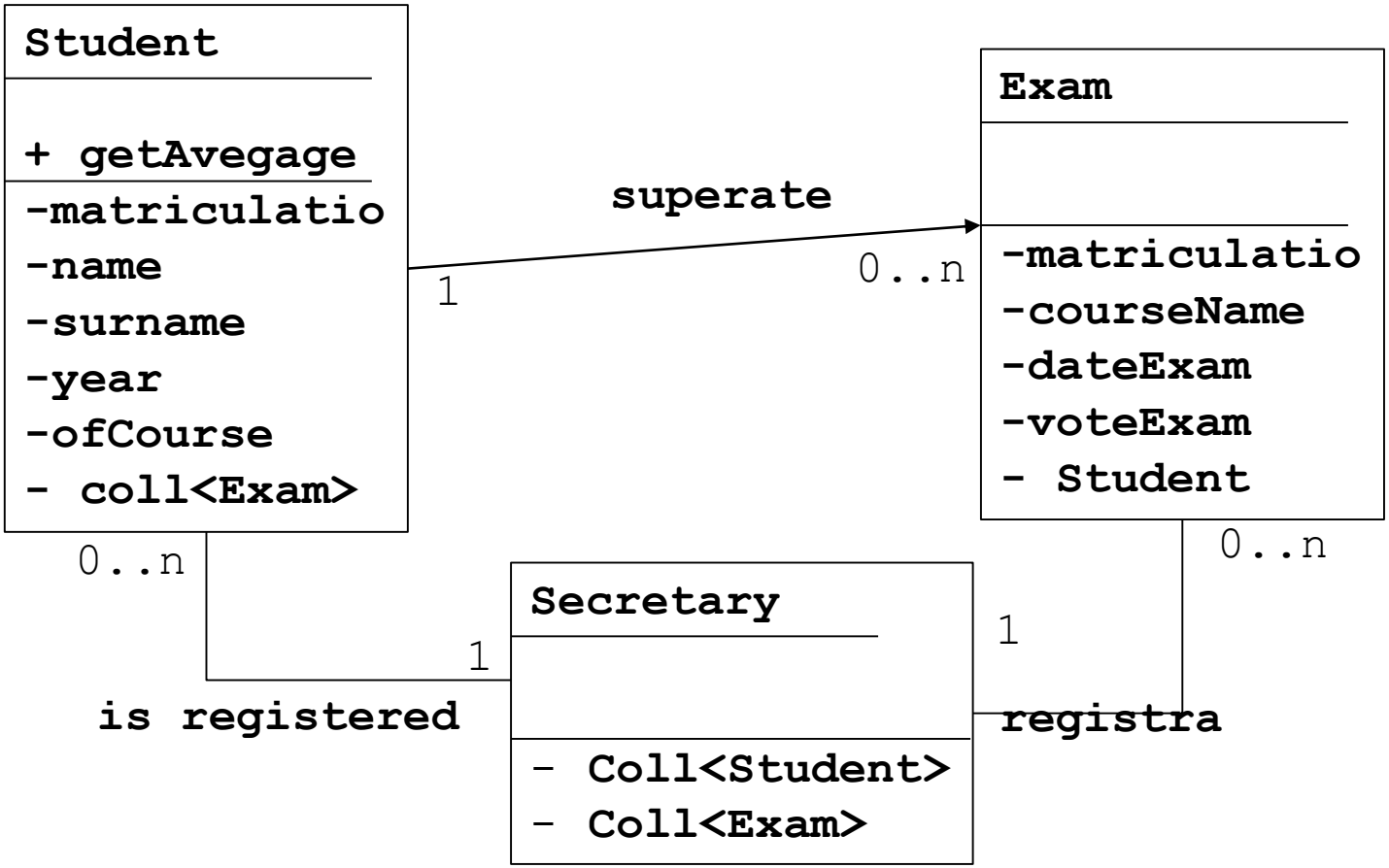
Esercizio

- Un file "Studenti.dati" contiene un elenco di studenti iscritti ed è organizzato come segue
  - Matricola Nome Cognome AnnoIscrizione FuoriCorso
  - Esempio

    1511627 Paolo Bianchi 3 true
    1234563 Mario Rossi 2 false
    12427459 Maria Verdi 3 false
- Il file "Esami.dati" contiene gli esami che sono stati superati da ciascuno studente. Esso è organizzato come segue:
  - Matricola Esame Data Voto
  - Esempio:

    1511627 Programmazione 12-10-2008 25
    12427459 Fisica 09-11-2007 23
    1511627 Fisica 20-09-2007 27
    …

Scrivere un programma che dato il nome e cognome di uno studente, restituisca l'elenco degli esami che egli ha sostenuto


Student
+ getAvegage
-matriculatio
-name
-surname
-year
-ofCourse
- coll<Exam>
superate
1
0..n
Exam
-matriculatio
-courseName
-dateExam
-voteExam
- Student
0..n
1
registra
0..n
Secretary
- Coll<Student>
- Coll<Exam>
1
is registered

Esercizio

- Una classe collezione che chiameremo Set
- Determinazione del comportamento
  - Set — Costruttore
  - contains — test di appartenenza
  - isEmpty — verifica dell'insieme vuoto null
  - addElement — aggiunge un elemento a un Set
  - copy — fa una copia di un Set
  - size — restituisce il numero di elementi di un Set
  - elements — restituisce una collezione per l'attraversamento
  - union — genera l'unione di due Set
  - intersection — genera l'intersezione di due Set
  - print — stampa l'insieme

Array bidimensionali

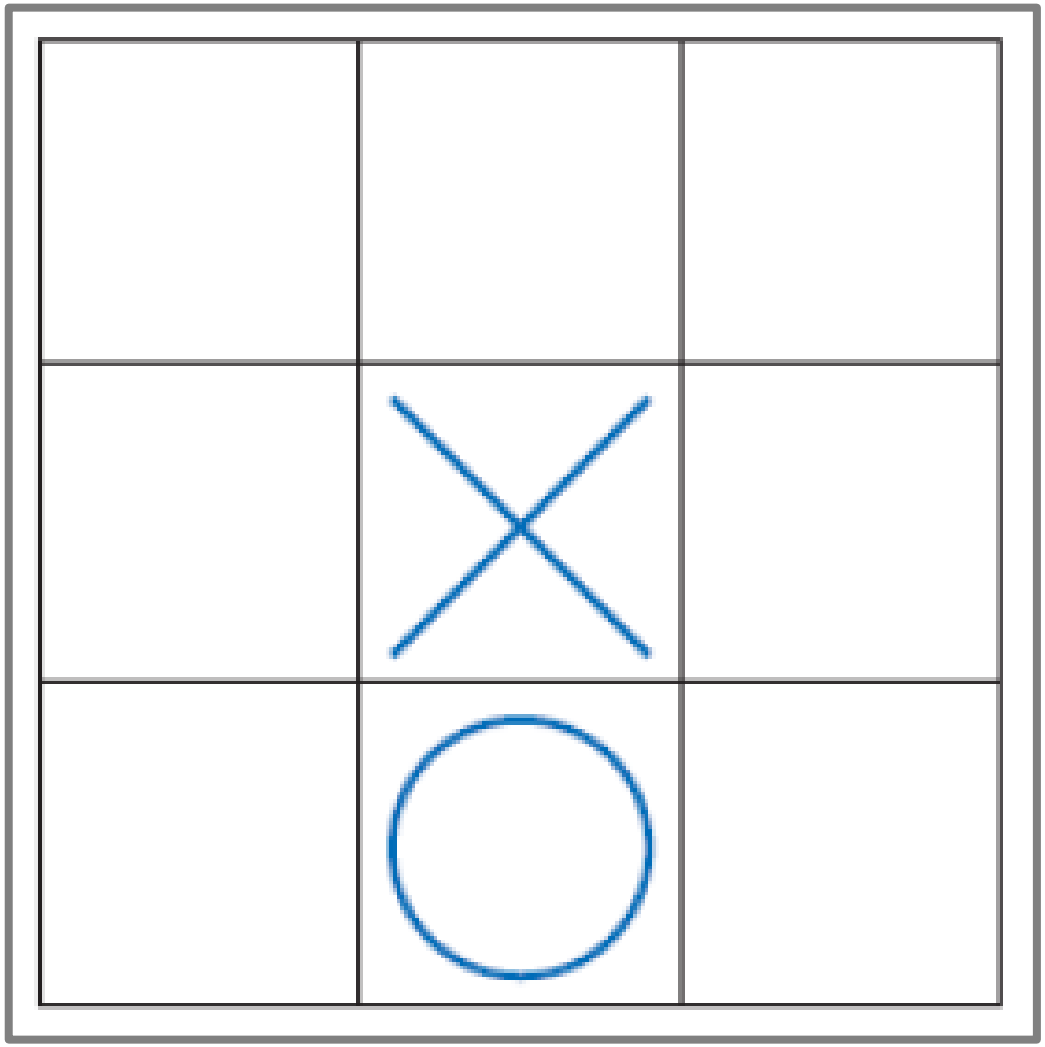
- Si specifica il numero di righe e di colonne in fase di creazione:

```
final int ROWS = 3;
final int COLUMNS = 3;
String[][] board = new String[ROWS][COLUMNS];
```

- L'accesso avviene mediante una coppia di indici `a[i][j]`

```
board[i][j] = "x";
```

Esempio



Attraversamento

- Tipicamente si ricorre a due cicli innestati:

```
for (int i = 0; i < ROWS; i++)
   for (int j = 0; j < COLUMNS; j++)
      board[i][j] = " ";
```

File TicTacToe.java

```
01: /**
02:    A 3 x 3 tic-tac-toe board.
03: */
04: public class TicTacToe
05: {
06:    /**
07:       Constructs an empty board.
08:    */
09:    public TicTacToe()
10:    {
11:       board = new String[ROWS][COLUMNS];
12:       // Fill with spaces
13:       for (int i = 0; i < ROWS; i++)
14:          for (int j = 0; j < COLUMNS; j++)
15:             board[i][j] = " ";
16:    }
17:
```

File TicTacToe.java

```
18:    /**  Sets a field in the board.
19:         The field must be unoccupied.
20:         @param i the row index
21:         @param j the column index
22:         @param player the player ("x" or "o")
23:    */
24:    public void set(int i, int j, String player)
25:    {
26:       if (board[i][j].equals(" "))
27:          board[i][j] = player;
28:    }
29:
30:    /** Creates a string representation of the board,
31:         such as:
32:       |x  o|
33:       |  x |
34:       |   o|
35:       @return the string representation
36:    */
```

File TicTacToe.java

```
37:     public String toString()
38:     {
39:        String r = "";
40:        for (int i = 0; i < ROWS; i++)
41:        {
42:           r = r + "|";
43:           for (int j = 0; j < COLUMNS; j++)
44:              r = r + board[i][j];
45:           r = r + "|\n";
46:        }
47:        return r;
48:     }
49:
50:     private String[][] board;
51:     private static final int ROWS = 3;
52:     private static final int COLUMNS = 3;
53: }
```

File TicTacToeTester.java

```
01: import java.util.Scanner;
02:
03: /**
04:    This program tests the TicTacToe class by prompting
05:    the user to set positions on the board and printing
06:    out the result.
07: */
08: public class TicTacToeTester
09: {
10:    public static void main(String[] args)
11:    {
12:       Scanner in = new Scanner(System.in);
13:       String player = "x";
14:       TicTacToe game = new TicTacToe();
15:       boolean done = false;
16:       while (!done)
17:       {
```

File TicTacToeTester.java

```
18:             System.out.print(game);
19:             System.out.print(
20:               "Row for " + player + " (-1 to exit): ");
21:             int row = in.nextInt();
22:             if (row < 0) done = true;
23:             else
24:             {
25:               System.out.print("Column for "+player+": ");
26:               int column = in.nextInt();
27:               game.set(row, column, player);
28:               if (player.equals("x"))
29:                   player = "o";
30:                else
31:                   player = "x";
32:             }
33:         }
34:     }
35: }
```

Output

```
|  |
|  |
|  |
Row for x (-1 to exit): 1

Column for x: 2

|   |

| x|

|

Row for o (-1 to exit): 0

Column for o: 0

|o  |

| x|

|   |

Row for x (-1 to exit): -1
```

Esercizio

- Implementare la classe Matrice di Double
- Prodotto riga colonna
- Somma
- Trasposta
- Uguaglianza
- Inversa