

Esercizi su Array

(Fondamenti di Informatica – Emilio Di Giacomo)

Esercizio 1 Ogni oggetto della classe **BattagliaNavale** rappresenta uno schema per il gioco della battaglia navale. La classe ha una variabile di istanza **schema** (di tipo **int[][]**) e una variabile di classe **idNave**. La variabile **schema** rappresenta la griglia su cui sono disposte le navi. Ogni nave è identificata da un codice identificativo (*id*) che è un numero intero positivo. La variabile di classe **idNave** rappresenta il codice identificativo della prossima nave che verrà inserita. Ogni nave inoltre occupa un certo numero di celle consecutive. Il numero di tali celle è la *lunghezza* della nave. Per semplicità si assume che le navi siano tutte disposte in orizzontale (ogni nave pertanto occupa una serie di celle consecutive sulla stessa riga). La griglia di gioco viene gestita secondo le seguenti regole:

- Una cella il cui valore è 0 non è occupata da navi.
- Una nave con $id\ k > 0$ occupa una serie di celle consecutive di una certa riga. Ciascuna di tali celle contiene il valore k se non è mai stata colpita, e il valore $-k$ in caso contrario.

Una nave che è stata colpita su tutte le sue celle è *affondata*.

La tabella 1 mostra un esempio di schema in cui sono disposte 3 navi identificate dai codici 1, 2, e 3 di lunghezza 3, 2 e 2, rispettivamente. La tabella 2 mostra lo stesso schema dopo che la nave 1 e 2 sono state colpite ciascuna una volta (le celle colpite sono evidenziate in grassetto).

0	0	0	0	0	0
0	1	1	1	0	0
0	0	0	0	0	0
0	0	0	0	0	0
3	3	0	2	2	0
0	0	0	0	0	0

Tabella 1

0	0	0	0	0	0
0	1	-1	1	0	0
0	0	0	0	0	0
0	0	0	0	0	0
3	3	0	2	-2	0
0	0	0	0	0	0

Tabella 2

La classe **BattagliaNavale** è dotata dei seguenti metodi:

- **public BattagliaNavale(int n)**: costruttore della classe. Crea uno schema di dimensione $n \times n$
- **public boolean aggiungiNave(int i, int j, int l)**: aggiunge una nave allo schema. I parametri *i* e *j* sono gli indici della prima cella occupata dalla nave, mentre *l* è la lunghezza della nave. La nave aggiunta avrà come id il valore di **idNave** (che dovrà opportunamente essere aggiornato). Il metodo restituisce **true** se la nave è stata aggiunta correttamente, **false** se non è possibile aggiungere la nave. I motivi perché l'aggiunta di una nave potrebbe non essere possibile sono:
 - almeno una delle celle che la nave dovrebbe occupare è già occupata da un'altra nave;
 - la nave esce fuori dai margini dello schema.
- **public int colpo(int i, int j)**: spara un colpo sulla cella (*i,j*). Qualora il colpo vada a segno (cioè colpisca una nave) lo schema viene aggiornato conseguentemente. Il metodo restituisce uno dei seguenti valori:
 - 0 se il colpo è andato a vuoto
 - 1 se il colpo è andato a segno ma la nave non è stata affondata

- 2 se il colpo è andato a segno e la nave è stata affondata
- **public String toString():** restituisce una rappresentazione testuale dello schema di gioco.

Si scriva la classe **BattagliaNavale** ed una classe **ProvaBattagliaNavale** che contiene il solo metodo **main** e che esegue le seguenti azioni:

- Crea uno schema di battaglia navale chiedendone la dimensione all'utente.
- Chiede ripetutamente all'utente se vuole inserire una nave. Ogni volta che l'utente risponde affermativamente gli viene chiesta la posizione e la lunghezza della nave e gli viene comunicato se la nave è stata inserita o meno.
- Quando l'utente ha terminato di inserire le navi gli viene mostrato lo schema creato;
- Chiede ripetutamente all'utente se vuole tentare di colpire una nave. Ogni volta che l'utente risponde affermativamente gli viene chiesta la cella da colpire e gli viene comunicato l'esito del colpo ("Mancato!", "Colpito!", oppure "Colpito e affondato!").

Esercizio 2 – Un *sistema lineare* è un insieme di m equazioni lineari in n incognite; un sistema lineare può essere genericamente scritto come segue:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \dots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m \end{cases}$$

I termini a_{ij} sono numeri reali chiamati coefficienti, i termini b_i sono numeri reali e sono detti termini noti, i termini x_j sono le incognite del sistema. Un sistema lineare può essere descritto mediante una matrice A ed un vettore b ; la matrice A , detta matrice dei coefficienti, è la matrice

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

Il vettore b , detto vettore dei termini noti, è il vettore:

$$b = \begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_m \end{pmatrix}$$

Una soluzione di un'equazione del sistema è un vettore di numeri reali $(\overline{x_1}, \overline{x_2}, \dots, \overline{x_n})$ che sostituiti alle incognite rendono vera l'equazione. Una soluzione del sistema è un vettore di numeri reali $(\overline{x_1}, \overline{x_2}, \dots, \overline{x_n})$ che sostituiti alle incognite rendono vere tutte le equazioni del sistema.

Ogni oggetto della classe **Sistema** rappresenta un sistema lineare di m equazioni lineari in n incognite. La classe ha il seguente scheletro:

```
public class Sistema {
```

```
    private double[][] A; // matrice dei coefficienti
```

```
private double[] b;    // vettore dei termini noti
```

```
/* crea un sistema lineare la cui matrice dei coefficienti è A e il cui vettore dei termini
noti è b. Si può assumere che A e b abbiano dimensioni coerenti (cioè il numero di
elementi di b sia pari al numero di righe di A) */
```

```
public Sistema(double[][] A, double[] b){...}
```

```
/* Restituisce true se il vettore x passato come parametro è una soluzione della
equazione i-esima del sistema. Si può assumere che la dimensione di x sia corretta
(cioè sia pari al numero di colonne della matrice dei coefficienti) */
```

```
public boolean verificaEquazione(int i, double[] x){...}
```

```
/* Restituisce true se il vettore x passato come parametro è una soluzione del
sistema. Si può assumere che la dimensione di x sia corretta (cioè sia pari al
numero di colonne della matrice dei coefficienti) */
```

```
public boolean verificaSistema(double[] x){...}
```

```
/* Restituisce sotto forma di stringa l'equazione i-esima del sistema */
```

```
public String stampaEquazione(int i){...}
```

```
/* Restituisce sotto forma di stringa il sistema */
```

```
public String toString(){...}
```

```
}
```

Si scriva la classe **Sistema** ed una classe **ProvaSistema** che contiene il solo metodo **main** e che esegue le seguenti azioni:

- fa inserire all'utente un sistema lineare chiedendogli di fornire sia la matrice dei coefficienti A sia il vettore dei termini noti b (è necessario garantire che A e b abbiano dimensioni coerenti);
- crea un oggetto Sistema s che rappresenta il sistema lineare inserito dall'utente;
- visualizza il sistema creato;
- fa inserire all'utente un vettore x che rappresenta una possibile soluzione del sistema (è necessario garantire che la dimensione del vettore inserito sia corretta);
- Dice all'utente se x è effettivamente una soluzione del sistema, oppure no.

Esercizio 3 – Ogni oggetto della classe **Tartaglia** rappresenta un triangolo di Tartaglia con un certo numero di righe. Si ricorda che il triangolo di Tartaglia è una disposizione geometrica a forma di triangolo dei coefficienti binomiali, ossia dei coefficienti dello sviluppo del binomio $(a+b)$ elevato ad una qualsiasi potenza n . Il triangolo di Tartaglia può essere così definito:

- l' i -esima riga ha $i+1$ elementi ($i=0,1,2,\dots$)
- l'elemento t_{ij} in posizione j -esima sulla riga i -esima ($0 \leq j \leq i$) è definito come segue:

$$t_{ij} = \begin{cases} 1 & \text{se } j=0 \text{ oppure } j=i \\ t_{i-1,j-1} + t_{i-1,j} & \text{altrimenti} \end{cases}$$

Le prime cinque righe del triangolo di Tartaglia sono le seguenti:

```
    1
   1 1
  1 2 1
 1 3 3 1
1 4 6 4 1
```

Si scriva la classe **Tartaglia** che ha il seguente scheletro:

```
class Tartaglia{

    private int[][] triangolo;

    /* costruttore. Costruisce un triangolo di Tartaglia con n righe */
    public Tartaglia(int n){...}

    /* restituisce la riga i-esima del triangolo rappresentato */
    public int[] getRiga(int i){...}

    /* restituisce una descrizione del triangolo rappresentato
       sotto forma di stringa */
    public String toString(){..}

}
```

Si scriva poi una classe di test **ProvaTartaglia** che testa la precedente.