Progettazione di software sicuro online
Esame del 13 marzo 2015 – (Prova di laboratorio)

1 Rubik2D [pt. 2]

Scrivere una classe `Rubik2D` per rappresentare una versione semplificata del cubo di Rubik in cui solo una faccia del cubo viene rappresentata.

All'inizio, la configurazione della faccia è quella mostrata in Fig. 1.

1	0	1
2	1	2
0	2	0

Figure 1: Configurazione iniziale

Metodo `gira` La classe dispone di un metodo booleano `gira` che, dati un indice di colonna `idCol`, sposta verso l'alto la colonna (in modo ciclico); il metodo è mostrato in Codice 1.

```
public boolean gira(int idCol) {
    if(idCol >= 0 && idCol < 3) {
        int primo = faccia[0][idCol];
        for(int i = 0; i < 2; i++) {
            faccia[i][idCol] = faccia[i + 1][idCol];
        }
        faccia[2][idCol] = primo;
        return true;
    }
    return false;
}
```

Codice 1: Metodo `gira(int idCol)`

Il metodo ritorna *true* se `idCol` indica una colonna valida e quindi lo spostamento viene fatto, *false* altrimenti.

Pe esempio, eseguendo il metodo `gira(2)` della configurazione iniziale, si ottiene la configurazione mostrata in Fig. 2.

1	0	2
2	1	0
0	2	1

Figure 2: Configurazione ottenibile eseguendo `gira(2)` a partire dalla configurazione iniziale

Metodo `isSolved` La classe deve disporre di un metodo booleano `isSolved()` che dica se la faccia è *risolta*, cioè se su ogni riga c'è lo stesso numero. Un esempio di faccia risolta viene mostrata in Fig. 3.

2	2	2
0	0	0
1	1	1

Figure 3: Esempio di faccia risolta

2 Junit

In JUnit, scrivere i seguenti casi di test nella classe `Rubik2DTest`.

- Per il metodo `gira(int idCol)`, scrivere:
  - un caso di test che mostri che è possibile eseguire lo spostamento su una colonna (basandosi sul valore ritornato dal metodo; **[pt. 1]**
  - un caso di test che mostri che, se si seleziona una colonna inesistente, nessuno spostamento viene eseguito (basandosi sul valore ritornato dal metodo). **[pt. 1]**
- Scrivere un caso di test che mostri che è possibile risolvere la faccia. **[pt. 1]**

3 Copertura

Scrivere in JUnit una test suite per il metodo `gira` che soddisfi la copertura delle condizioni. Creare una classe `GiraCoperturaCondizioni` e commentare in modo opportuno i singoli casi di test. **[pt. 1]**

4 JML

Scrivere in JML la seguente postcondizione al costruttore:

- il valore di tutte le celle della faccia sono comprese tra 0 e 2. **[pt. 1]**

Scrivere in JML le seguenti postcondizioni al metodo `gira(int idCol)`:

- ci sono tre celle con valore uguale a 0; **[pt. 1]**
- la somma di tutte le celle è 9; **[pt. 1]**
- le celle che non appartengono alla colonna `idCol` non hanno cambiato valore rispetto a prima dell'esecuzione del metodo. **[pt. 1]**

Totale punti = 2 + 3 + 1 + 4 = 10