

Linguaggi di Programmazione per la Sicurezza

Esame del 9 febbraio 2011 - (Prova di laboratorio)

1 Gioco dell'otto

Il gioco dell'otto¹ consiste di una griglia quadrata, divisa in 3 righe e 3 colonne, su cui sono posizionate in modo casuale 8 tessere, numerate a partire da 1. Le tessere possono scorrere in orizzontale o verticale, ma il loro spostamento è limitato dall'esistenza di un singolo spazio vuoto. Scopo del gioco è riordinare le tessere: la tessera con il numero 1 in alto a sinistra, e quelle con gli altri numeri a seguire in modo ordinato da sinistra a destra e dall'alto in basso, con la cella vuota alla fine, come mostrato in tabella 1.

1	2	3
4	5	6
7	8	0

Table 1: Configurazione finale

1.1 Classe Java [pt. 2]

Scrivere una classe *GiocoDello8* che rappresenti una griglia 3×3 ; tale classe deve permettere di muovere le tessere nella griglia. Le tessere sono rappresentate da numeri nell'intervallo $[1, 8]$ e la cella vuota da 0.

All'inizio la griglia deve contenere le tessere nell'ordine mostrato in tabella 2.

2	3	6
1	5	0
4	7	8

Table 2: Configurazione iniziale

Metodo move La classe deve disporre di un metodo *move(int i, int j, int h, int k)* che permetta di spostare in posizione (h, k) la tessera in posizione (i, j) . Non bisogna fare alcun controllo sul fatto che i, j, h, k siano indici validi e che la mossa sia fattibile.

Metodo isSolved La classe deve disporre di un metodo booleano *isSolved()* che dica se il rompicapo è stato risolto, cioè se i numeri sono disposti in ordine crescente (a partire da 1) da sinistra a destra e dall'alto in basso, e se l'ultima cella è vuota (come in tabella 1).

¹Il gioco dell'otto è una versione modificata del gioco del 15 (http://en.wikipedia.org/wiki/Fifteen_puzzle).

2 JUnit

- In JUnit, per il metodo *move*, scrivere:
 - un caso di test in cui si mostri che lo spostamento di una tessera da (i, j) ad (h, k) (scelti da voi in modo che identifichino celle contigue, che (i, j) contenga una tessera, e che (h, k) sia vuota) avviene in modo corretto; **[pt. 1]**
 - un caso di test in cui si mostri che lo spostamento di una tessera da una cella piena (i, j) a una cella piena (h, k) viene eseguito portando la griglia in una configurazione errata (scompare la tessera presente nella cella di destinazione). Il caso di test si aspetta che, se si prova a spostare una tessera in una cella piena, lo spostamento non dovrebbe essere eseguito: invece il metodo esegue lo spostamento. **[pt. 1]**
- scrivere in JUnit un caso di test che evidenzi che la griglia impostata dal costruttore non contiene la soluzione del rompicapo; **[pt. 1]**
- scrivere in JUnit un caso di test che evidenzi che è possibile eseguire una serie di 7 mosse che portano alla soluzione del rompicapo; nella tabella 3 sono mostrati i parametri che bisogna usare nelle 7 chiamate a *move*. **[pt. 1]**

i	j	h	k
0	2	1	2
0	1	0	2
0	0	0	1
1	0	0	0
2	0	1	0
2	1	2	0
2	2	2	1

Table 3: Mosse per ottenere la soluzione del rompicapo a partire dalla configurazione impostata dal costruttore

3 JML

Scrivere in JML la seguente postcondizione al costruttore:

- per ogni tessera numerata con un numero dell'intervallo $[1,8]$ c'è una cella che la contiene; inoltre c'è una cella vuota (nota che la cella vuota è identificata con 0). **[pt. 1.5]**

Scrivere in JML le seguenti precondizioni al metodo *move*:

- gli indici i, j, h e k sono indici validi della griglia; **[pt. 0.5]**
- la cella identificata da (i, j) contiene una tessera; **[pt. 0.5]**
- la cella identificata da (h, k) è la cella vuota; **[pt. 0.5]**
- le celle identificate dagli indici (i, j) e (h, k) sono contigue. **[pt. 1]**

Totale punti = 2 + 4 + 4 = 10

```

public class GiocoDello8_20110209 {
    /*@ spec_public @*/ private int[][] board;

    //Scrivere in JML la seguente postcondizione al costruttore:
    //per ogni tessera dell'intervallo [1,8] c'e' una cella che la contiene;
    //inoltre c'e' una cella vuota (nota che la cella vuota e' identificata con 0)

    /*@ ensures (\forall int k; k >= 0 && k <= 8;
        @           (\exists int i, j; i >= 0 && i <= 2 && j >= 0 && j <= 2;
        @           board[i][j] == k)
        @           );
    @*/

    //oppure in modo esplicito
    //@ ensures (\exists int i, j; i >= 0 && i <= 2 && j >= 0 && j <= 2; board[i][j] == 0);
    //@ ensures (\exists int i, j; i >= 0 && i <= 2 && j >= 0 && j <= 2; board[i][j] == 1);
    //@ ensures (\exists int i, j; i >= 0 && i <= 2 && j >= 0 && j <= 2; board[i][j] == 2);
    //@ ensures (\exists int i, j; i >= 0 && i <= 2 && j >= 0 && j <= 2; board[i][j] == 3);
    //@ ensures (\exists int i, j; i >= 0 && i <= 2 && j >= 0 && j <= 2; board[i][j] == 4);
    //@ ensures (\exists int i, j; i >= 0 && i <= 2 && j >= 0 && j <= 2; board[i][j] == 5);
    //@ ensures (\exists int i, j; i >= 0 && i <= 2 && j >= 0 && j <= 2; board[i][j] == 6);
    //@ ensures (\exists int i, j; i >= 0 && i <= 2 && j >= 0 && j <= 2; board[i][j] == 7);
    //@ ensures (\exists int i, j; i >= 0 && i <= 2 && j >= 0 && j <= 2; board[i][j] == 8);
    public GiocoDello8_20110209() {
        board = new int[3][3];
        board[0][0] = 2;
        board[0][1] = 3;
        board[0][2] = 6;
        board[1][0] = 1;
        board[1][1] = 5;
        board[1][2] = 0;
        board[2][0] = 4;
        board[2][1] = 7;
        board[2][2] = 8;
    }

    //Scrivere in JML le seguenti precondizioni al metodo "move":
    //gli indici i, j, h e k sono indici validi
    //@ requires i >= 0 && i <= 2 && j >= 0 && j <= 2;
    //la cella identificata da (i, j) identifica una tessera
    //@ requires board[i][j] != 0;
    //la cella identificata da (h, k) e' la cella vuota
    //@ requires board[h][k] == 0;
    //@ requires h >= 0 && h <= 2 && k >= 0 && k <= 2;
    //le celle identificate dagli indici (i, j) e (h, k) sono contigue
    //@ requires (Math.abs(i - h) == 1 && (j == k)) || (Math.abs(j - k) == 1 && (i == h));
    public void move(int i, int j, int h, int k) {
        board[h][k] = board[i][j];
        board[i][j] = 0;
    }

    public boolean isSolved() {
        for(int i = 0; i < 3; i++) {
            for(int j = 0; j < 3; j++) {
                if((i*3 + j + 1)%9 != board[i][j]) {
                    return false;
                }
            }
        }
    }
}

```

```

    }
    return true;
}

public int get(int i, int j) {
    return board[i][j];
}

/**
 * Versione alternativa
 */
public boolean isSolved_v2() {
    int k = 1;
    for(int i = 0; i < 3; i++) {
        for(int j = 0; j < 3; j++) {
            if(i==2 && j==2) {
                if(board[i][j]!=0) {
                    return false;
                }
            }
            else if(board[i][j]!=k) {
                return false;
            }
            k++;
        }
    }
    return true;
}

public static void main(String[] args) {
    GiocoDello8_20110209 q = new GiocoDello8_20110209();
    //q.move(3, 0, 0, 1);//viola le preconditione sugli indici validi
    //q.move(1, 2, 1, 2);//viola le preconditione che richiede che (i, j) identifichino una tessera
    //q.move(2, 0, 2, 1);//viola le preconditione che richiede che (h, k) identifichino la cella vuota
    q.move(0, 0, 1, 2);//viola le preconditione che richiede che (h, k) identifichino la cella vuota
}
}

```

```

import static org.junit.Assert.*;

import org.junit.Test;

public class Test_GiocoDello8_20110209 {
    //un caso di test in cui si mostri che lo spostamento di una tessera da (i, j) a (h, k)
    //(scelti da voi in modo che identifichino celle contigue, che (i, j) contenga una tessera, e che (h, k) sia vuota)
    //avviene in modo corretto.
    @Test
    public void testMove1() {
        GiocoDello8_20110209 g = new GiocoDello8_20110209();
        assertEquals(6, g.get(0, 2));
        assertEquals(0, g.get(1, 2));
        g.move(0, 2, 1, 2);
        assertEquals(0, g.get(0, 2));
        assertEquals(6, g.get(1, 2));
    }

    //un caso di test in cui si mostri che lo spostamento di una tessera da una cella piena (i, j) a una
    //cella piena (h, k) viene eseguito portando la griglia in una configurazione errata (scompare la tessera
    //presente nella cella di destinazione). Il caso di test si aspetta che, se si prova a spostare una tessera in
    //una cella piena, lo spostamento non dovrebbe essere eseguito: invece il metodo esegue lo spostamento.
    @Test
    public void testMove2() {
        GiocoDello8_20110209 g = new GiocoDello8_20110209();
        assertEquals(2, g.get(0, 0));
        assertEquals(3, g.get(0, 1));
        g.move(0, 0, 0, 1);
        assertEquals(2, g.get(0, 0));
        assertEquals(3, g.get(0, 1));
    }

    //scrivere in JUnit un caso di test che evidenzi che la griglia impostata dal
    //costruttore non contiene la soluzione del rompicapo
    @Test
    public void testIsSolved1() {
        GiocoDello8_20110209 g = new GiocoDello8_20110209();
        assertFalse(g.isSolved());
    }

    //scrivere in JUnit un caso di test che evidenzi che e' possibile eseguire una serie di
    //7 mosse che portano alla soluzione del rompicapo
    @Test
    public void testIsSolved2() {
        GiocoDello8_20110209 g = new GiocoDello8_20110209();
        g.move(0, 2, 1, 2);
        g.move(0, 1, 0, 2);
        g.move(0, 0, 0, 1);
        g.move(1, 0, 0, 0);
        g.move(2, 0, 1, 0);
        g.move(2, 1, 2, 0);
        g.move(2, 2, 2, 1);
        assertTrue(g.isSolved());
    }
}

```