

# Linguaggi di Programmazione per la Sicurezza

## Esame del 24 febbraio 2011 - (Prova di laboratorio)

### 1 Knight's tour

Nel gioco degli scacchi, il cavallo può muoversi solo eseguendo un movimento ad **L**, cioè muovendosi prima in orizzontale (a sinistra o destra) di 1/2 celle e poi in verticale (verso l'alto o verso il basso) di 2/1 celle. Nella tabella 1 sono mostrate le mosse che il cavallo posizionato in c2 può fare in una scacchiera  $5 \times 5$ : può andare in b4, d4, a3, e3, a1, e1.


	a	b	c	d	e
5					
4		×		×	
3	×				×
2					
1	×				×

Table 1: Possibili mosse di un cavallo posizionato in c2

Il problema *Knight's tour* prevede di far eseguire al cavallo un *tour* completo della scacchiera. Il cavallo, all'inizio, deve essere posizionato in una cella della scacchiera. Successivamente, bisogna far eseguire una serie di mosse al cavallo, seguendo le seguenti regole:

- il cavallo deve essere mosso seguendo le regole degli scacchi (movimento a **L**);
- quando il cavallo visita una cella, questa deve essere marchiata come *visitata*;
- una cella non può essere visitata due volte.

Il problema prevede che il cavallo esegua un *tour completo*, cioè che visiti tutte le celle della scacchiera. Il cavallo deve eseguire un *open tour*, cioè la cella iniziale e la cella finale del tour devono essere diverse.

#### 1.1 Classe Java [pt. 2]

Scrivere una classe *KnightTour* che rappresenti una scacchiera  $5 \times 5$  di booleani: la scacchiera deve memorizzare se una cella è già stata visitata o meno. La classe deve permettere anche di memorizzare le coordinate in cui si trova attualmente il cavallo. Inoltre la classe deve disporre di un metodo che permetta di muovere il cavallo nella scacchiera.

All'inizio le coordinate del cavallo sono (0,0) e la scacchiera indica che solo la cella (0,0) è stata visitata.

**Metodo move** La classe deve disporre di un metodo *move(int h, int k)* che permetta di spostare in posizione (h,k) il cavallo e che memorizzi che la cella (h,k) della scacchiera è stata visitata. Non bisogna fare alcun controllo sul fatto che h, k siano indici validi e che la mossa sia corretta (rispetti le regole della mossa del cavallo).

**Metodo isTourCompleted** La classe deve disporre di un metodo booleano *isTourCompleted()* che dica se il cavallo ha eseguito un tour completo della scacchiera.

## 2 Junit

- In JUnit, per il metodo *move*, scrivere:
  - un caso di test in cui si mostri che una mossa valida del cavallo (una mossa ad **L** corretta e tale per cui  $(h, k)$  identifichino una cella non ancora visitata) avviene in modo corretto: il cavallo viene spostato in  $(h, k)$  e la cella  $(h, k)$  viene marcata come *visitata*; [pt. 1]
  - un caso di test in cui si mostri che una mossa non valida (non è una mossa ad **L**) viene eseguita. Il caso di test si aspetta che, se si prova ad eseguire una mossa non valida, la mossa non dovrebbe essere eseguita: invece il metodo esegue la mossa. [pt. 1]
- scrivere in JUnit un caso di test che evidenzi che sulla scacchiera iniziale (quella costruita dal costruttore) non è stato eseguito un tour completo; [pt. 1]
- scrivere in JUnit un caso di test che evidenzi che è possibile eseguire una serie di 24 mosse che portano all'esecuzione di un tour completo; nella tabella 2 sono mostrati i parametri che bisogna usare nelle 24 chiamate a *move*. [pt. 1]

<b>h</b>	2	4	3	4	2	0	1	2	4	3	1	0	1	3	4	2	0	1	3	4	3	2	0	1
<b>k</b>	1	0	2	4	3	4	2	4	3	1	0	2	4	3	1	0	1	3	4	2	0	2	3	1

Table 2: Mosse per ottenere un tour completo a partire dalla configurazione impostata dal costruttore

## 3 JML

Scrivere in JML le seguenti postcondizioni al costruttore:

- la cella  $(0, 0)$  è marcata come *visitata*; [pt. 0.25]
- tutte le celle della scacchiera tranne  $(0, 0)$  sono marcate come *non visitate*. [pt. 1.25]

Scrivere in JML le seguenti precondizioni al metodo *move(int h, int k)*:

- gli indici  $h$  e  $k$  sono indici validi della scacchiera; [pt. 0.5]
- la cella identificata da  $(h, k)$  non è già stata visitata; [pt. 0.5]
- la mossa dalla posizione corrente del cavallo ad  $(h, k)$  è una mossa ad **L** corretta. [pt. 1.5]

**Totale punti = 2 + 4 + 4 = 10**

package esami;

```
public class KnightTour_20110224 {
    /*@ spec_public @*/ boolean[][] visited;
    /*@ spec_public @*/ int x, y;//posizione del cavallo

    //@ ensures x == 0 && y == 0;

    //la cella (0, 0) e' marcata come visitata
    //@ ensures visited[0][0];

    //tutte le celle della scacchiera tranne (0, 0) sono marcate come non visitate
    //versione 1
    //@ ensures (\forallall int i, j; i >= 0 && i < 5 && j >= 0 && j < 5; (i != 0 || j != 0) ==> !visited[i][j]);
    //versione 2
    //@ ensures (\forallall int i, j; i >= 0 && i < 5 && j >= 0 && j < 5 && (i != 0 || j != 0); !visited[i][j]);
    KnightTour_20110224() {
        visited = new boolean[5][5];
        x = 0;
        y = 0;
        visited[x][y] = true;
    }

    //gli indici "h" e "k" sono indici validi della scacchiera
    //@ requires h >= 0 && h < 5 && k >= 0 && k < 5;
    //la mossa dalla posizione corrente del cavallo ad (h, k) e' una mossa ad L corretta
    //@ requires (Math.abs(h - x) == 2 && Math.abs(k - y) == 1) || Math.abs(h - x) == 1 && Math.abs(k - y) ==
2;
    //la cella identificata da (h, k) non e' gia' stata visitata
    //@ requires !visited[h][k];
    //@ ensures visited[h][k];
    public void move(int h, int k) {
        visited[h][k] = true;
        x = h;
        y = k;
    }

    public boolean isTourCompleted() {
        for(int i = 0; i < 4; i++) {
            for(int j = 0; j < 4; j++) {
                if(!visited[i][j]) {
                    return false;
                }
            }
        }
        return true;
    }

    public static void main(String[] args) {
        KnightTour_20110224 kt = new KnightTour_20110224();
        //kt.move(0, 1);//viola la preconditione sulla mossa corretta

        //kt.move(5, 2);//viola la preconditione sugli indici validi

        //kt.move(1, 2);
        //kt.move(0, 0);//viola la preconditione sulla cella gia' visitata
    }
}
```

```
//tour completo
kt.move(2, 1);
kt.move(4, 0);
kt.move(3, 2);
kt.move(4, 4);
kt.move(2, 3);
kt.move(0, 4);
kt.move(1, 2);
kt.move(2, 4);
kt.move(4, 3);
kt.move(3, 1);
kt.move(1, 0);
kt.move(0, 2);
kt.move(1, 4);
kt.move(3, 3);
kt.move(4, 1);
kt.move(2, 0);
kt.move(0, 1);
kt.move(1, 3);
kt.move(3, 4);
kt.move(4, 2);
kt.move(3, 0);
kt.move(2, 2);
kt.move(0, 3);
kt.move(1, 1);
    }
}
```

```
package esami;
```

```
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;
```

```
import org.junit.Test;
```

```
public class Test_KnightTour_20110224 {
```

```
    //Un caso di test in cui si mostri che una mossa valida del cavallo (una mossa ad L corretta e tale
    //per cui (h, k) identifichino una cella non ancora visitata) avviene in modo corretto: il cavallo viene
    //spostato in (h, k) e la cella (h, k) viene marcata come visitata.
```

```
    @Test
```

```
    public void testMove1() {
        KnightTour_20110224 kt = new KnightTour_20110224();
        kt.move(1, 2);
        assertEquals(1, kt.x);
        assertEquals(2, kt.y);
        assertEquals(true, kt.visited[1][2]);
    }
```

```
    //Un caso di test in cui si mostri che una mossa non valida (non e' una mossa ad L) viene eseguita. Il
    //caso di test si aspetta che, se si prova ad eseguire una mossa errata, la mossa non dovrebbe essere
    //eseguita: invece il metodo esegue la mossa.
```

```
    @Test
```

```
    public void testMove2() {
        KnightTour_20110224 kt = new KnightTour_20110224();
        kt.move(1, 2);
        kt.move(0, 0);
        assertEquals(1, kt.x);
        assertEquals(2, kt.y);
    }
```

```
    //Un caso di test che evidenzi che sulla scacchiera iniziale (quella costruita dal costruttore) non e'
    //stato eseguito un tour completo.
```

```
    @Test
```

```
    public void testIsTourCompleted1() {
        KnightTour_20110224 kt = new KnightTour_20110224();
        assertFalse(kt.isTourCompleted());
    }
```

```
    //Un caso di test che evidenzi che e' possibile eseguire una serie di 24 mosse che portano all'esecuzione
    //di un tour completo.
```

```
    @Test
```

```
    public void testIsTourCompleted2() {
        KnightTour_20110224 kt = new KnightTour_20110224();
        kt.move(2, 1);
        kt.move(4, 0);
        kt.move(3, 2);
        kt.move(4, 4);
        kt.move(2, 3);
        kt.move(0, 4);
        kt.move(1, 2);
        kt.move(2, 4);
        kt.move(4, 3);
        kt.move(3, 1);
    }
```

```
        kt.move(1, 0);
        kt.move(0, 2);
        kt.move(1, 4);
        kt.move(3, 3);
        kt.move(4, 1);
        kt.move(2, 0);
        kt.move(0, 1);
        kt.move(1, 3);
        kt.move(3, 4);
        kt.move(4, 2);
        kt.move(3, 0);
        kt.move(2, 2);
        kt.move(0, 3);
        kt.move(1, 1);
        assertTrue(kt.isTourCompleted());
    }
}
```