

Progettazione di software sicuro

Esame del 16 luglio 2012 - (Prova di laboratorio)

1 Sudoku [pt. 2]

Scrivere una classe *Sudoku* che permetta di giocare ad una versione semplificata del sudoku in cui la scacchiera è di dimensione 4×4 . Una cella si considera vuota se contiene il numero 0. Il gioco è risolto se:

- ogni riga contiene tutti i numeri dell'intervallo $[1, 4]$ senza ripetizioni;
- ogni colonna contiene tutti i numeri dell'intervallo $[1, 4]$ senza ripetizioni;
- le quattro sotto-scacchiere 2×2 contengono tutti i numeri dell'intervallo $[1, 4]$ senza ripetizioni.

Tabella 1 mostra un esempio di sudoku non risolto, mentre tabella 2 mostra un esempio di sudoku risolto.

1	2	0	0
0	0	1	0
2	0	4	0
0	0	0	3

Table 1: Sudoku non risolto

1	2	3	4
3	4	1	2
2	3	4	1
4	1	2	3

Table 2: Sudoku risolto

Il costruttore della classe deve inizializzare la scacchiera in modo che il sudoku sia vuoto.

Metodo add La classe deve disporre di un metodo booleano *add(int r, int c, int value)* che permetta di aggiungere un numero nella scacchiera; il metodo ritorna *true* se l'inserimento viene eseguito, *false* altrimenti. Il metodo è mostrato nel Codice 1.

```
public boolean add(int r, int c, int value) {  
    if(r >= 0 && r <= 3 && c >= 0 && c <= 3 && value >= 1 && value <= 4) {  
        board[r][c] = value;  
        return true;  
    }  
    else {  
        return false;  
    }  
}
```

Codice 1: Metodo *add(int r, int c, int value)*

Metodo `isMaybeSolved` La classe deve anche disporre di un metodo booleano `isMaybeSolved` che dica se la scacchiera soddisfa una condizione necessaria (ma non sufficiente) affinché possa essere considerata risolta:

- ogni riga deve sommare a 10;
- ogni colonna deve sommare a 10;
- le quattro sotto-scacchiere 2×2 devono sommare tutte a 10.

2 Junit

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

- Per il metodo `add`, scrivere:
 - un caso di test che mostri che, se i parametri attuali sono corretti, il valore viene aggiunto alla scacchiera (basandosi sul valore ritornato dal metodo); **[pt. 1]**
 - un caso di test in cui si mostri che si può aggiungere un numero alla scacchiera che non soddisfa le regole del sudoku (ad esempio un numero già contenuto nella riga). Il caso di test si aspetta che, se si prova a mettere un numero in una riga che contiene già quel numero in una posizione diversa, l'aggiunta non dovrebbe essere eseguita: invece il metodo esegue l'aggiunta. **[pt. 1]**
- Scrivere un caso di test che mostri che, se si inseriscono i numeri come mostrato dalla tabella 2, il metodo `isMaybeSolved` dice che il sudoku potrebbe contenere la soluzione. **[pt. 1]**

3 Copertura

Scrivere in JUnit una test suite che soddisfi la copertura delle condizioni per il metodo `add`; mettere i test case nella classe `AddCoperturaCondizioni` e commentare in modo opportuno i singoli test case. **[pt. 1]**

4 JML

Scrivere in JML la seguente preconditione al metodo `add(int r, int c, int value)`:

- tutti gli elementi sulla riga r sono diversi da $value$. **[pt. 1]**

Scrivere in JML le seguenti postcondizioni al metodo `add(int r, int c, int value)`:

- il numero di celle vuote è minore od uguale a 16; **[pt. 1.5]**
- il metodo ha ritornato `true` se e solo se la decisione della regola **if** contenuta nel metodo valuta a `true`. **[pt. 1.5]**

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