

# Progettazione di software sicuro

## Esame del 4 aprile 2012 - (Prova di laboratorio)

### 1 NaturalSet [pt. 2]

Scrivere una classe *NaturalSet* che rappresenti un insieme di numeri naturali. La classe rappresenta l'insieme usando un array di interi: le celle libere dell'array contengono il valore -1.

Il costruttore della classe deve inizializzare l'array usando come dimensione il valore che viene passato come parametro al costruttore. All'inizio l'insieme è vuoto.

**Metodo remove** La classe deve disporre di un metodo *remove(int value)* che permetta di rimuovere un elemento dall'insieme; il metodo è mostrato nel Codice 1. Se il valore *value* è presente nell'insieme, il metodo lo rimuove e lo ritorna, altrimenti viene ritornato il valore -1.

```
public int remove(int value) {  
    int i = 0;  
    while(i < set.length) {  
        if(set[i] == value) {  
            set[i] = -1;  
            return value;  
        }  
        i++;  
    }  
    return -1;  
}
```

Codice 1: Metodo *remove*

**Metodo add** La classe deve disporre di un metodo booleano *add(int value)* che permetta di aggiungere un elemento all'insieme. Se il valore *value* non è già presente nell'insieme, deve essere inserito e il metodo deve ritornare *true*; altrimenti, se il valore è già presente nell'insieme, non deve essere inserito e il metodo deve ritornare

*false*<sup>1</sup>.

Il metodo, invece, non deve controllare che il valore che viene inserito sia positivo.

## 2 Junit

In JUnit, scrivere per il metodo *add* nella classe *NaturalSetTest*:

- un caso di test che mostri che, se si prova ad aggiungere due volte lo stesso elemento, la seconda volta l'elemento non viene aggiunto (basandosi sul valore ritornato dal metodo); [pt. 1]
- un caso di test che mostri che, se si prova ad aggiungere un numero negativo, il numero viene aggiunto (basandosi sul valore ritornato dal metodo); il caso di test si aspetta che il codice sia corretto e quindi che, se si aggiunge un numero negativo, il numero non dovrebbe essere aggiunto: il metodo, invece, aggiunge l'elemento; [pt. 1]
- un caso di test che mostri che, in un insieme inizializzato con 10 posizioni libere, si possono inserire 10 elementi distinti. [pt. 1]

## 3 Copertura

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

## 4 JML

Scrivere in JML la seguente preconditione al costruttore:

- la dimensione massima dell'insieme deve essere compresa tra 1 e 100. [pt. 0.25]

Scrivere in JML la seguente postcondizione al costruttore:

- l'insieme non contiene elementi. [pt. 0.5]

Scrivere in JML la seguente preconditione al metodo *add(int value)*:

- il valore *value* è positivo. [pt. 0.25]

Scrivere in JML le seguenti postcondizioni al metodo *add(int value)*:

- nell'insieme non esistono due elementi uguali; [pt. 1.5]
- il numero di elementi dell'insieme dopo l'esecuzione del metodo è uguale o al massimo maggiore di un'unità al numero di elementi dell'insieme prima dell'esecuzione del metodo. [pt. 1.5]

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

---

<sup>1</sup>Per l'implementazione bisogna notare che non è possibile sapere a priori quale cella è libera, perché, a causa dell'esecuzione del metodo *remove*, l'array potrebbe alternare celle occupate (con valore maggiore od uguale a zero) a celle libere (con valore uguale a -1).

```
package esami;
```

```
public class NaturalSet_20120404 {
    private /*@ spec_public @*/ int[] set;

    //la dimensione massima dell'insieme deve essere compresa tra 1 e 100
    //@ requires size >= 1 && size <= 100;
    //l'insieme non contiene elementi
    //@ ensures (\forall int i; i >= 0 && i < size; set[i] == -1);
    public NaturalSet_20120404(int size) {
        set = new int[size];
        for(int i = 0; i < size; i++) {
            set[i] = -1;
        }
    }

    //il valore "value" e' positivo
    //@requires value >= 0;
    //nell'insieme non esistono due elementi uguali
    //@ ensures !(\exists int i, j; i >=0 && i < set.length && j >=0 && j < set.length && i!=j; set[i] == set[j] &&
set[i] != -1);
    //il numero di elementi dell'insieme dopo l'esecuzione del metodo e' uguale
    //o al massimo maggiore di un'unita' al numero di elementi dell'insieme prima
    //dell'esecuzione del metodo
    /*@ ensures ((\num_of int i; i >=0 && i < set.length; set[i] > -1) -
    @      (\num_of int i; i >=0 && i < set.length; \old(set[i] > -1))) == 1
    @      ||
    @      ((\num_of int i; i >=0 && i < set.length; set[i] > -1) -
    @      (\num_of int i; i >=0 && i < set.length; \old(set[i] > -1))) == 0;
    @*/
    public boolean add(int value) {
        int index = -1;
        for(int i = 0; i < set.length; i++) {
            if(set[i] == value) {
                return false;
            }
            else if(index == -1 && set[i] == -1) {
                index = i;
            }
        }
        if(index == -1) {
            return false;
        }
        else {
            set[index] = value;
            return true;
        }
    }

    public int remove(int value) {
        int i = 0;
        while(i < set.length) {
            if(set[i] == value) {
                set[i] = -1;
                return value;
            }
            i++;
        }
    }
}
```

```
    }  
    return -1;  
}  
  
public static void main(String[] args) {  
    //viola la preconditione "la dimensione massima dell'insieme deve essere compresa tra 1 e 100"  
    //NaturalSet_20120404 set = new NaturalSet_20120404(200);  
  
    NaturalSet_20120404 set1 = new NaturalSet_20120404(10);  
    set1.add(6);  
    set1.add(-5); //viola la preconditione "il valore "value" e' positivo"  
}  
}
```

```
package esami;
```

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

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

```
public class NaturalSetTest_20120404 {
```

```
    //un caso di test che mostri che, se si prova ad aggiungere due
    //volte lo stesso elemento, la seconda volta l'elemento non viene
    //aggiunto (basandosi sul valore ritornato dal metodo)
```

```
    @Test
```

```
    public void testAdd1() {
        NaturalSet_20120404 set = new NaturalSet_20120404(10);
        assertTrue(set.add(3));
        assertFalse(set.add(3));
    }
```

```
    //un caso di test che mostri che, se si prova ad aggiungere un numero negativo,
    //il numero viene aggiunto (basandosi sul valore ritornato dal metodo);
    //il caso di test si aspetta che il codice sia corretto e quindi che, se si
    //aggiunge un numero negativo, il numero non dovrebbe essere aggiunto: il metodo,
    //invece, aggiunge l'elemento
```

```
    @Test
```

```
    public void testAdd2() {
        NaturalSet_20120404 set = new NaturalSet_20120404(10);
        assertFalse(set.add(-3));
    }
```

```
    //un caso di test che mostri che in un insieme inizializzato con 10 posizioni
    //libere si possono inserire 10 elementi distinti
```

```
    @Test
```

```
    public void testAdd3() {
        NaturalSet_20120404 set = new NaturalSet_20120404(10);
        for(int i = 0; i < 10; i++) {
            assertTrue(set.add(i));
        }
    }
}
```

```
package esami;
```

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

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

```
//Scrivere in JUnit una test suite che soddisfi la copertura delle decisioni
```

```
//per il metodo "remove"; mettere i test case nella classe
```

```
//"RemoveCoperturaDecisioni" e commentare in modo opportuno i singoli test case.
```

```
public class RemoveCoperturaDecisioni_20120404 {
```

```
    //copre:
```

```
    // - "i < set.length" a true
```

```
    // - "set[i] == value" a true
```

```
    @Test
```

```
    public void testRemoveCoperturaDecisioni1() {
```

```
        NaturalSet_20120404 set = new NaturalSet_20120404(10);
```

```
        set.add(2);
```

```
        assertEquals(2, set.remove(2));
```

```
    }
```

```
    //copre:
```

```
    // - "i < set.length" a false
```

```
    // - "set[i] == value" a false
```

```
    @Test
```

```
    public void testRemoveCoperturaDecisioni2() {
```

```
        NaturalSet_20120404 set = new NaturalSet_20120404(10);
```

```
        assertEquals(-1, set.remove(1));
```

```
    }
```

```
}
```