Esercizi JML + KEY - laboratorio

1. BoundedStack of Objects

Aggiunti in contratti per uno stack (bounded) di Object.

```
public class BoundedStack {

  private Object[] elems;
  private int size = 0;

  // invarianti:
  // size >= 0 e limitato dalla size di elems
  // elems non è nullo, tutti gli elementi da 0 a size sono non nulli
  // ma tutti gli elementi da size a length di elems sono nulli

  // pre: n > 0
  // post: elems ha n elementi
  public BoundedStack(int n) {
    elems = new Object[n];
  }

  // pre: non è pieno,
  // post: size viene incrementato e l'oggetto viene inserito
correttamente
  public void push(Object x) {
    elems[size] = x;
    size++;
  }

  // pre: non è vuoto
  // post size decrementato, oggetto tolto (ma gli altri rimangono uguali
  public void pop() {
    size--;
    elems[size] = null;
  }

  // pre: non è vuoto
  // restituisce l'ultimo oggetto
  public Object top() {
    return elems[size-1];
  }
}
```

Scrivi la classe, comincia a scrivere i contratti e un main in cui chiami la classe e vedi mano a mano che i contratti vengono violati, correggi/completa il codice sia del main (con commenti) sia della classe.

2. Bag of int

Scrivi l'implementazione e i contratti per questa classe che rappresenta un insieme (con possibili ripetizioni) di interi:

```
public class BagOfInt {
      private int[] a;
      private int n;

      /** Initialize to contain input's elements. */
      public BagOfInt(int[] input);

      /** Return the multiplicity of i. */
      public int occurrences(int i);

      /** Return and delete the minimum element. */
      public int extractMin();
}
```

Note:

- occurrences è pure e usa nel contratto (\num_of int ...)
- extractMin ha un contratto del tipo:

```
ensures \result == (\min int i ....);
```

```
ensures (\forall int j; 0 <= j && j < \old(n); in termini di occurrences
```

Scrivi il main che dovrà essere più o meno:

```
class BagOfIntMain {

  public static void main(String [] argv) {
    int[] mine
       = new int[] {0, 10, 20, 30, 40, 10};
    BagOfInt b = new BagOfInt(mine);
    System.out.println(
       "b.occurrences(10) == "
        + b.occurrences(10));
    //@ assert b.occurrences(10) == 2;
    //@ assert b.occurrences(5) == 0;
    int em1 = b.extractMin();
    //@ assert em1 == 0;
    int em2 = b.extractMin();
    //@ assert em2 == 10;
    int em3 = b.extractMin();
    //@ assert em3 == 10;
  }
}
```

3. Filosofi a Cena (vedi PDF)

Con anche la soluzione

4. Queue

In this assignment, we are going to specify an implementation of a **Queue**. We do not actually implement it.

Consider the following half-done class description of a Queue:

```
public class Queue {

    Object[] arr;
    int size;
    int first;
    int next;

    Queue( int max ) {
        // ...
    }

    public int size() {
        // ...
    }

    public void enqueue( Object x ) {
        // ...
    }

    public Object dequeue() {
        // ...
    }
}
```

Give a JML specification of the Queue. It should consist of: class invariant(s), and pre- and post-conditions for each method.

JML + KEY

5. maxPositiveIntegers

Il metodo maxPositiveIntegers(int a, int b) ritorna il massimo di due numeri interi forniti in input.

- definire una precondizione che controlli che i parametri di input non siano negativi;
- definire una postcondizione che controlli che il risultato ritornato sia il massimo tra i due parametri;
- se il metodo maxPositiveIntegersWrong(int a, int b) contiene un'implementazione errata di maxPositiveIntegers: può essere usato per controllare un'eventuale violazione della postcondizione.

Scrivi contratti e prova con JML e poi la correttezza con KEY

6. Account

Scrivi una classe Account con i seguenti metodi (tipo quello fatto a lezione)

- costruttore Account, crea un account con un minimo di balance (dato come costante)
- preleva
- deposita

Scrivi i contratti (pre e post e invarianti) e prova a dimostrare la correttezza usando Key per eclipse.

7. Contatore

La classe Counter implementa un contatore:

  - all'inizio il contatore vale 0;
  - i metodi incr() e decr() incrementano e decrementano il contatore;
  - il getter getCounter() ritorna il valore del contatore.
- annotare in modo opportuno il campo privato counter anche possa essere utilizzato in specifiche JML pubbliche;
- annotare in modo opportuno il metodo getCounter() anche possa essere utilizzato in specifiche JML;
- denire postcondizioni ai metodi incr() e decr() che controllino che le operazioni siano eseguite correttamente.
- Provare con KEY

8. Somma N numeri

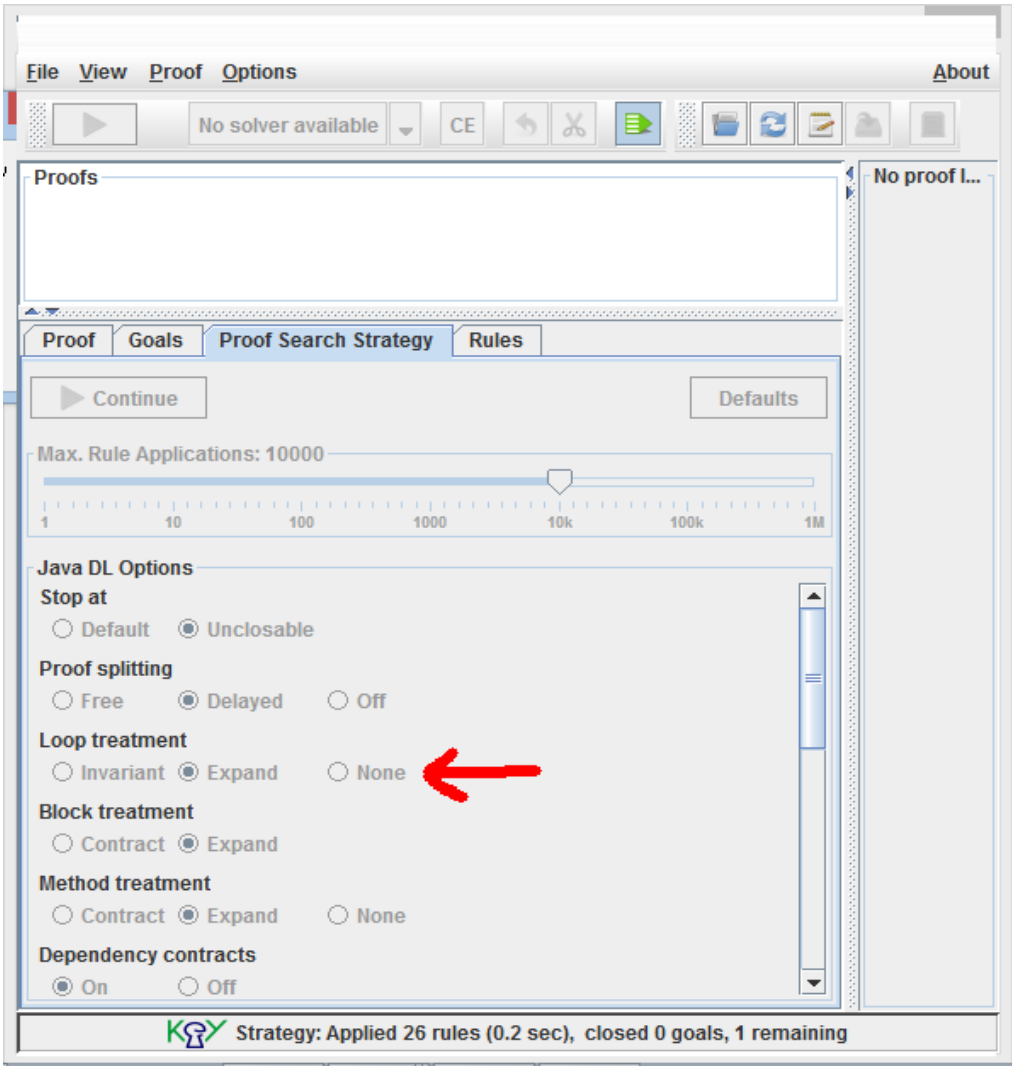
Il metodo sumN(int n) ritorna la somma dei primi n numeri naturali:

- scrivere le pre e post condizioni
- scrivere il codice in cui calcoli la somma **senza** usare cicli come = (n * ....)/2
  - prova con qualche esempio se funziona (JML runtime check) e prova la dimostrazione
- scrivi il codice con un ciclo

- prova con qualche esempio se funziona (JML runtime check) e prova la dimostrazione

ATTENZIONE

per provare gli inavrianti con monkey/key, devi attivare il trattamento dei loop invariant medinate inavarianti.



9. Somma

Il seguente codice calcola la somma di un array. Completa la postcondizione e il loop invariant e prova la correttezza.

```
//@ requires a != null;
//@ ensures ....
static int sum(int[] a) {
     int sum = 0; int i = 0;
     //@ loop_invariant sum == ****
     while (i < a.length) {
          sum += a[i];
          i++;
     }
```

```
        //@ assert i == a.length;
        return sum;
    }
```