




# TESTING E VERIFICA DEL SOFTWARE

## ESERCITAZIONE 3 - JML

(27-03-2015, 6-4-2016, 4-4-2017, 13-4-2023)

### INDICAZIONI PER UTILIZZARE jml:

- Scrivi il tuo progetto Java come al solito
- Una volta scritti i contratti, premi tasto destro sul progetto>>Enable JML
- L'icona del progetto deve cambiare da  a 

1. premi  per verificare la sintassi dei tuoi contratti.
2. premi "RAC" per compilare.
3. scrivi un main in cui provi i metodi della tua classe ed eseguillo.
4. Per ogni esercizio fai almeno alcune chiamate di metodi in cui:
  - a **Violi una precondition** passando degli input non corretti
  - b **Violi una post condizione** (modificando il codice e introducendo un errore che causa una violazione della post)
  - c (opzionale) **Violi un invariante se ne hai inserito uno.**

### Esercizio 0

Traduci i contratti nei commenti in java, poi scrivi un main in cui provi i metodi di Account:

```
public class Account{

    private /*@ spec_public @*/ int balance;

    //post: l'account è creato con valore di balance 100
    public Account(){
balance=100;
    }

    //pre: qta maggiore di 0;
    //pre: balance maggiore-uguale di qta;
    //post: il valore di balance è stato modificato correttamente
    //post: balance non può diventare negativo
    void preleva(int qta){
        balance-=qta;
    }
}
```

public

```

        //pre: qta maggiore di 0;
        //post: balance è stato modificato correttamente
public
void deposita(int qta){
    balance+=qta;
}
}

```

### Esercizio 1

Scrivi i contratti per la classe BoundedStack (stack con max n elementi)

```

public class BoundedStack{

    private /*@ spec_public*/Object[] elems;
    private /*@ spec_public*/int size=0;

    // invarianti:
    // size >= 0 e limitato dalla size di elems
    // elems non è nullo
    // tutti gli elementi da 0 a size (escluso) sono non nulli ma
    // quelli da size a length (escluso) sono NULLI (non ancora inseriti)

    // 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
    // post: restituisce l'ultimo oggetto
    public Object top(){
        return elems[size-1];
    }
}

```

Prova a scrivere un main in cui utilizzi i metodi di BoundedStack

### Esercizio 2

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

Adesso devi pensare sia ai contratti che all'implementazione dei metodi.

```

public class BagOfInt {
    private/*@ spec_public @*/int[] a;
    private/*@ spec_public @*/int n; //la lunghezza del vettore

    /*inizializza il vettore a e la lunghezza n*/
    public BagOfInt(int[] input) {

    }

    /* ritorna le occorrenze del numero i nel vettore a[]*/
    public int occurrences(int i) {

    }

    /* ritorna il valore minimo contenuto in a[]*/
    public int extractMin() {

    }
}

```

Scrivi un main in cui provi i metodi di BagOfInt.

### Esercizio 3:

Implementare la classe Somma, la quale nel metodo public /\*@pure@\*/ static int sum(int a[]) , va a sommare i valori contenuti nel vettore a[] mediante un ciclo. Definire i seguenti contratti:

- Pre-condizione: `a[]` non nullo;
- Post-condizione: il risultato deve avere lo stesso valore della somma dei valori in `a[]`;

Utilizzare inoltre nel metodo la clausola `@loop_invariant`. Gli invarianti devono essere quindi veri prima e dopo il ciclo. Testare infine con una classe `Main.class`

#### **Esercizio 4 (Tema d'esame 27/1/2015):**

Riprendi l'esercizio 4 dell'esercitazione 1 e scrivi i contratti JML. Cerca di scrivere sia le precondizioni, che le postcondizioni dei metodi.

Cerca di scrivere anche invarianti. Cerca di spostare nelle precondizioni alcuni controlli che facevi all'inizio dei metodi (ad esempio relativi agli indici).

Aggiungi anche questi contatti:

- precondizione al metodo `insert`: esiste un prodotto con meno di 100 unità
- postcondizioni al metodo `insert`:
  - la media dei prodotti a magazzino è minore o uguale a 100;
  - la quantità dei prodotti diversi da `productIndex` è rimasta invariata.

Prova i contratti JML con una classe `main` in cui chiami i diversi metodi.

Prova anche a modificare il codice e controlla che i contratti siano violati.

#### **Esercizio 5 (tema d'esame giugno 22)**

Implementa il tutto in una classe `LightArray`.

`LightArray` ha un array di 10 luci ognuna della quale può essere in stato accesa o spenta.

Inizialmente sono tutte accese.

Posso cambiare lo stato di una lampadina singola (`toggle`) (*attenzione: cosa accade alle lampadine di cui non faccio il toggle*).

Posso decidere di spegnerle tutte (`allOff`) e questa operazione ha priorità rispetto al cambio di stato di una lampadina (non serve per JML).

Scrivi il codice in Java con in contratti opportuni (la classe `LightArray` ha uno costruttore, il metodo `allOff` che spegne tutte le lampadine e `toggle(i)` che cambia lo stato alla lampadina `i`-esima).

Cerca di scrivere sia le precondizioni, che le postcondizioni dei metodi.

Cerca di scrivere anche invarianti.

Testa i contratti JML con una classe `main` in cui chiami i diversi metodi.

Prova anche a modificare il codice e controlla che i contratti siano violati.  
Documenta bene le violazioni e le loro cause in commenti e nel file di documento.  
Verifica (esercitazione successiva).

Copia il progetto precedente, toglì tutte le cose statiche (anche il main) non usare gli enumerativi. Dimostra con key che i contratti siano rispettati.

### Tema d'esame del 2025

Ho un silos di grano che ha  $n$  ( $>0$  e  $< 20$ ) bidoni cilindrici che possono contenere fino a 12 quintali di grano (usiamo un **intero** per indicare la quantità di grano).

Una **gru** può versare del grano (da 1 fino a 5 quintali) in un singolo cilindro a scelta dell'operatore, mentre un **nastro** trasportatore può prelevare 2 quintali di grano da tutti i contenitori (se un cilindro ha meno di due quintali, si svuota – cioè ad esempio se voglio prelevare 2 da tutti i cilindri ma uno di essi ha solo 1 quintale, esso viene semplicemente svuotato).

La gru e il nastro non funzionano mai tutte e due insieme, però almeno uno dei due funziona sempre.

Il sistema per sicurezza non permette mai di avere più di 10 quintali in ogni bidone (anche se ce ne starebbero 12) (la politica per garantire questo quando la gru riempie i cilindri è già implementata si spera in modo corretto).

Inizialmente i bidoni sono tutti con 1 quintale di grano.

Scrivi i contratti opportuni (la classe Silos ha uno **costruttore**, il metodo **gru** mette il grano (data la posizione del cilindro e il grano da mettere) e **nastro** che toglie 2 quintali di grano a tutti i silos.

Cerca di scrivere sia le precondizioni, che le postcondizioni dei metodi.

Cerca di scrivere anche invarianti.

Testa i contratti JML con una classe main in cui chiami i diversi metodi.

Prova anche a modificare il codice e controlla che i contratti siano violati.  
Documenta bene le violazioni e le loro cause in commenti.