

Specifica di Astrazioni Procedurali con JML

Esercizio 1

`public static boolean sottoStringa (char[] testo, char[] parola)`

Es: `testo = [abbcdddeff]` e `parola = [bcdde]` -> risultato è true

Es: `testo = [abbcdddeff]` e `parola = [baadde]` -> risultato è false

- a. Scrivere le pre-condizioni sui due array in modo che l'operazione sia sensata.
- b. Togliere le pre-condizioni del passo precedente e rendere il metodo totale.

Soluzione 1a

```
//@ requires testo != null && parola != null;
```

```
//@ assignable \nothing;
```

```
//@ ensures (\result == true) <==>
```

```
    (\exists int i; 0<=i<=testo.length - parola.length;
```

```
    (\forall int j; 0<=j<=parola.length-1; parola[j] == testo[i+j] ));
```

```
static boolean sottoStringa (char[] testo, char[] parola)
```

Soluzione 1

//@ requires **true;** – *metodo totale*

//@ assignable \nothing;

//@ ensures **testo != null && parola != null &&**

(\result == true) <==>

(\exists int i; 0 <= i <= testo.length - parola.length;

(\forall int j; 0 <= j <= parola.length - 1; parola[j] == testo[i+j]));

– *vecchie precondizioni nella postcondizione relativa al comportamento normale*

//@ signals (NullPointerException npe) testo==null || parola==null;

– *precondizioni negate come postcondizioni del caso eccezionale*

static boolean sottoStringa (char[] testo, char[] parola) **throws**

NullPointerException

Esercizio 2

- Data una classe *Interval*
 - rappresenta un intervallo
 - ha solo due metodi pubblici puri che restituiscono float (osservatori):
 - float `getLowerBound()`, float `getUpperBound()`
- Vogliamo specificare
`public static Interval getInterval (float[] times, float timePoint)`
 - *times* non nullo
 - *times* contiene valori ordinati in maniera strettamente crescenti (punti dell'asse dei tempi)
 - restituisce un oggetto di tipo *Interval* che corrisponde a un intervallo temporale avente come estremi due punti contigui di *times*.
 - *timePoint* deve essere maggiore o uguale all'estremo minore e strettamente minore all'estremo maggiore
 - a. Identificare pre-post condizioni e clausola assignable
 - b. Rendere il metodo totale e cambiare la specifica

Esercizio 2

- public static Interval getInterval (float[] times, float timePoint)
 - *times* non nullo: **precondizione**
 - *times* contiene valori ordinati in maniera strettamente crescente (punti dell'asse dei tempi): **precondizione**
 - **restituisce** un oggetto di tipo *Interval* che corrisponde a un intervallo temporale avente come estremi due punti contigui di *times*:
sicuramente postcondizione, ma vincola anche qualcosa sulla lunghezza minima possibile di times (precondizione).
 - *timePoint* deve essere maggiore o uguale all'estremo minore e strettamente minore all'estremo maggiore: ***questa frase predica sull'intervallo restituito, quindi sicuramente determina la postcondizione; tuttavia, determina anche quali sono i valori validi di timePoint rispetto a times***
 - ***Assignable: il metodo non ha side effect di alcun tipo (times non viene modificato)***

Soluzione 2a - *metodo parziale*

```
//@ requires times != null && times.length >= 2 &&  
    (\forall int i; 0 <= i <= times.length - 2;  
        times[i] < times[i + 1]) &&  
    timePoint >= times[0] && timePoint < times[times.length - 1]  
  
//@ assignable \nothing;  
  
//@ ensures  (\exists int i; 0 <= i <= times.length - 2;  
    \result.getLowerBound() == times[i] &&  
    \result.getUpperBound() == times[i + 1]) &&  
    timePoint >= \result.getLowerBound() &&  
    timePoint < \result.getUpperBound();  
  
static Interval getInterval (float[] times, float timePoint)
```


Soluzione 2b - *metodo totale*

```
//@ requires true;  
//@ assignable \nothing;  
    //@ ensures times != null && (\forallall int i; 0<=i<=times.length-2;  
                                times[i] < times[i+1]) &&  
    times.length >=2 && timePoint >= times[0] &&  
    timePoint < times[times.length-1] &&  
    (\exists int i; 0<=i<=times.length-2;  
     \result.getLowerBound() == times[i] &&  
     \result.getUpperBound() == times[i+1]) ) &&  
    timePoint >= \result.getLowerBound() &&  
    timePoint < \result.getUpperBound();  
  
//@ signals (NullPointerException npe) times == null;  
//@ signals (ArrayNotOrdered anoex) !(\forallall int i; 0<=i<=times.length-2;  
                                     times[i] < times[i+1]);  
  
//@ signals (IllegalArgumentException ipex) times.length < 2 || timePoint < times[0] ||  
    timePoint >= times[times.length-1];  
  
static Interval getInterval (float[] times, float timePoint) throws  
    ArrayNotOrdered, NullPointerException, IllegalArgumentException
```


Esercizio 3

- Si supponga che il tipo di dato astratto Set sia definito in una classe separata e che fornisca il seguente metodo puro:
 - `boolean /*@ pure */ contains(int n)`
- Dato un array di valori numerici interi, il metodo produce l'insieme di valori (tramite un oggetto di tipo Set) che ricadono all'interno di un intervallo chiuso $[a,b]$
- a) assumere che l'array non abbia elementi duplicati;
b) Specificare che nel caso l'array contenga elementi duplicati venga lanciata un'eccezione.
- Ci sono ragioni per preferire a) rispetto a b)?

```
public static Set interval(int[] x, int a, int b)
```

Soluzione 3a

```
//@ requires x != null && a < b &&  
    (\forall int i; 0 <= i <= length-2;  
    !(exists int j; i < j <= x.length-1; x[i] == x[j])) ;  
  
//@ ensures (\forall int i ; 0 <= i <= x.length-1;  
    (a <= x[i] <= b) ==> (\result).contains(x[i])) &&  
    (\forall int n; (\result).contains(n);  
    (\exists int i; 0 <= i <= x.length-1;  
    x[i] == n));
```

```
public static Set interval(int[] x, int a, int b)
```

Soluzione 3b

```
//@ requires x != null && a < b;
```

```
//@ ensures (\forallall int i; 0 <= i <= length-2;
```

```
!(exists int j; i < j <= x.length-1; x[i] == x[j])) &&
```

```
(\forallall int i ; 0 <= i <= x.length -1; (a <= x[i] <= b) ==> (\result).contains(x[i])) &&
```

```
(\forallall int n; (\result).contains(n) ; (\exists int i; 0 <= i <= x.length-1; x[i] == n)) ;
```

```
//@ signals (DuplicateElementException deex)
```

```
(\exists int i; 0 <= i <= x.length-1;
```

```
(\exists int j; 0 <= j <= x.length -1 && i != j; x[i]==x[j])) ;
```

```
public static Set interval(int[] x, int a, int b) throws DuplicateElementException
```

Esercizio

- `public static void highLowNums (int [] nums, int [] highs, int n)`
 - l'array `nums` contiene interi tutti diversi tra di loro
 - l'array `highs` è lungo esattamente `n`.
 - Il metodo trova gli `n` numeri interi più grandi di `nums` e li inserisce in ordine decrescente nell'array `highs`; l'array `nums` non viene modificato.

Soluzione

```
\\@ requires nums != null && highs != null &&
    highs.length == n && nums.length >= n &&
    (\\forall int i; 0 <= i <= nums.length-2;
        (\\forall int j; i < j <= nums.length-1; nums[i] != numes[j]));

\\@ ensures nums.length==\\old(nums.length) &&
    (\\forall int i; 0 <= i <= nums.length-1;
        nums[i] == \\old(nums[i])) &&
    (\\forall int i; 0 <= i <= highs.length-1;
        (\\exists int j; 0 <= j <= nums.length-1; highs[i] == nums[j] )) && (\\forall
int i; 0 <= i <= highs.length-2; highs[i] > highs[i+1]) &&
    (\\forall int i; 0 <= i <= highs.length-1;
        (\\numof int j; 0 <= j <= nums.length-1;
            highs[i] < nums[j]) <=n);
```

```
void highLowNums (int [] nums, int [] highs, int n)
```

Esercizio

- E' data una classe immutabile Risultato che rappresenta il risultato di un esame.
 - `public TipoRisultato tipo()`
 - `public int voto() throws EsameException`
 - `public boolean lode() throws EsameException`
 - TipoRisultato e' una enum con tre valori:
 - RECUPERO, RIMANDATO, SUFF

Esercizio

- `public static Risultato calcola(int P1, int P2, int L);`
 - P1 primo compitino, P2 secondo compitino, L laboratorio
 - Ogni prova intermedia assegna fino a 13 punti. Lo studente deve prendere almeno 6 punti per passarlo (se no fa il recupero di entrambe)
 - Il laboratorio dà fino a 4 punti. Lo studente deve prenderne almeno 2. Non c'è recupero ma bisogna ripetere l'intero corso con annullamento delle prove in itinere
 - Per superare l'esame P1 e P2 devono totalizzare almeno 16 punti e il risultato complessivo deve essere almeno 18, pena il recupero.
 - Se il voto supera 31, lo studente ha la lode

Soluzione

```
//@ requires true;
//@ ensures
    0 <= P1<=13 && 0 <= P2 <= 13 && 0 <= L <= 4 &&
    (\result.tipo() == RIMANDATO) <==> ( L < 2 ) &&
    (\result.tipo() == RECUPERO) <==> ( L >= 2 &&
        ( P1 < 6 || P2 < 6 || P1 + P2 <
16) ) && (\result.tipo() == SUFF && \result.voto() == L +
P1 + P2 && !\result.lode()) <==> ( L + P1 + P2 < 31 &&
L>=2 && P1 >= 6 && P2 >= 6 && P1 + P2 >= 16) &&
    (\result.tipo() == SUFF && \result.lode()) <==> ( L +
P1 + P2 >= 31) ;

//@ signals (IncorrectGradeException igex)
    P1 < 0 || P1 > 13 || P2 < 0 || P2 > 13 || L < 0 || L > 4;

static Risultato calcolaVoto (int P1, int P2, int L) throws
    IncorrectGradeException
```


Specifica di Risultato

- Come specificare una classe immutabile?
 - Creatori + Osservatori + Produttori.
 - Ciascun osservatore garantisce qualcosa sul risultato degli altri.
 - Per esempio, se lode() allora voto() == 30
 - Se tipoRisultato() != SUFF, lode() e voto() lanciano eccezione EsameException

Soluzione: Metodi

```
public /*@ pure */ class Risultato {  
  
    public Risultato(TipoRisultato tipo);  
    public Risultato(int voto, boolean lode);  
  
    public TipoRisultato tipo()  
  
    /*@ requires true;  
    /*@ ensures tipo() == SUFF && 18 <= \result <= 30;  
    /*@ signals (EsameException e) tipo != SUFF;  
    public int voto();  
  
    /*@ requires true;  
    /*@ ensures \result ==> (tipo() == SUFF && voto()18 ==  
    30);
```

Soluzione: Costruttori

```
public /*@ pure */ class Risultato {  
  
    /*@ requires tipo == RIMANDATO || tipo == RECUPERO;  
    /*@ ensures tipo() == tipo;  
    public Risultato(TipoRisultato tipo)  
  
    /*@ requires 18 <= voto <= 30 && (lode ==> voto == 30);  
    /*@ ensures tipo() == SUFF && voto() == voto &&  
        lode() == lode;  
    public Risultato(int voto, boolean lode);
```

Soluzione: Osservatori

```
public /*@ pure */ class Risultato {  
  
    /*@ requires true;  
    /*@ ensures true;  
    public TipoRisultato tipo()  
  
    /*@ requires true;  
    /*@ signals (EsameException e) tipo() != SUFF;  
    public int voto();  
  
    /*@ requires true;  
    /*@ signals (EsameException e) tipo() != SUFF;  
    public boolean lode();  
  
}
```