Politecnico di Milano
Facoltà di Ingegneria dell'Informazione
Dipartimento di Elettronica e Informazione

Esercitazioni di Ingegneria del Software

09: JML – Specifiche Totali

0. Divisione

- Si consideri il metodo:

```
//@ requires in >= 0;
//@ ensures Math.abs(\result*\result-in)<0.0001;
public static float sqrt (float in) { ... }
```

- E i seguenti usi:
  - float zero = sqrt (0);
  - float uno = sqrt (1);
  - float due = sqrt (4);
  - float tre = sqrt (9);
  - float boh = sqrt (-1);
    - Visto che non stiamo rispettando le precondizioni, il comportamento è impredicibile

0. Divisione

- Come possiamo creare sistemi Robusti?
  - Il comportamento è predicibile anche nei casi in cui i patti non sono stati rispettati
- Si possono:
  - togliere le precondizioni
    - //@ requires true
  - evidenziare i casi problematici
    - //@ ensures `in >= 0 && (* altre postcondizioni *)`
  - sollevando le opportune eccezioni
    - //@ signals (NegativeException e) `in < 0;`

0. Divisione: operativamente

- Specifica con pre e post-condizioni:

```
//@ requires in >= 0;
//@ ensures Math.abs(\result*\result-in)<0.0001;
public static float sqrt (float in) { ... }
```

- Specifica totale:

```
//@ requires true;
//@ ensures in >= 0 && Math.abs(\result*\result-in)<0.0001;
//@ signals (NegativeException e) in < 0;
public static float sqrt (float in) { ... }
```

1. getInterval parziale

```
//@ requires times != null
//@ && (\forall int i; 0 <= i < times.length -1;
//@         times[i] < times[i+1]);


//@ ensures (\exists int i; 0 <= i < times.length -1;
//@ times[i]==\result.low && times[i+1]==\result.high)
//@ && timePoint >= \result.low
//@ && timePoint < \result.high;

public Interval getInterval(float[] times,float timePoint)
```


1. getInterval: totale

```
//@ requires true;

//@ ensures times != null
//@ && (\forall int i; 0 <= i < times.length -1;
//@       times[i] < times[i+1])
//@ && (\exists int i; 0 <= i < times.length -1;
//@ times[i]==\result.low && times[i+1]==\result.high)
//@ && timePoint >= \result.low
//@ && timePoint < \result.high;
//@ signals (NullPointerException npe) times == null;
//@ signals (UnorderedArrayException uae)
//@    ! (\forall int i; 0 <= i < times.length -1;
//@                         times[i] < times[i+1]);

public Interval getInterval(float[] times,float timePoint)
     throws NullPointerException, UnorderedArrayException;
```

Queste condizioni sono prese dalla requires e viene negata dalle signals; in questo modo garantiamo che le eccezioni vengano lanciate *se e solo se* le precondizioni non sono verificate.

2. subString parziale

```
//@ requires text != null && chunk != null &&
//@  chunk.length <= text.lenght;
//@ ensures \result == true <==>
//@   (\exists int i; 0<= i<= text.length - chunk.length;
//@     (\forall int j; 0 <= j < chunk.length;
//@      text[i+j] == chunk[j]));
public static boolean subString (char[] text,char[] chunk)
```


2. subString totale

```
//@ requires true;
//@ ensures text != null && chunk != null &&
//@  chunk.length <= text.lenght &&
//@ \result == true <==>
//@   (\exists int i; 0<= i<= text.length - chunk.length;
//@     (\forall int j; 0 <= j < chunk.length;
//@      text[i+j] == chunk[j]));
//@ signals (NullPointerException npe)
//@         text == null || chunk == null;
//@ signals (InvertedLengthException ile)
//@         chunk.length > text.length;
public static boolean subString (char[] text,char[] chunk)
   throws ...
```

4. isPermutation parziale

```
/*@ requires x != null && y != null &&
  @(\forall int i; 0 <= i < x.length -1;
      (\forall int j; i <= j < x.length; x[i] != x[j]))
  @ && (* same for y *);
  @ ensures (\result == true) <==>
  @    (x.length == y.length) &&
  @(\forall int i; 0 <= i < x.length;
  @   (\exists int j; 0 <= j < y.length; x[i] == y[j]));
  @*/
public static boolean isPermutation(int x[], int y[])
```


4. isPermutation totale

```
/*@ requires true
  @ ensures x != null && y != null &&
  @(\forall int i; 0 <= i < x.length -1;
      (\forall int j; i <= j < x.length; x[i] != x[j]))
  @ && (* same for y *) &&
  @ (\result == true) <==>
  @    (x.length == y.length) &&
  @(\forall int i; 0 <= i < x.length;
  @   (\exists int j; 0 <= j < y.length; x[i] == y[j]));
  @ signals (NullPointerException npe)
  @x == null || y == null;
  @ signals (DuplicateException de)
  @(\exists int i; 0 <= i < x.length -1;
  @   (\exists int j; i < j < x.length; x[i] == x[j]));
  @*/
public static boolean isPermutation(int x[], int y[]) …
```