JML (Java Modeling Language)

- Es un lenguaje para escribir contratos de especifican el comportamiento esperado de los programas.
- Fundamentalmente permite definir:
  - @requires (precondición)
  - @ensures (poscondición)
  - @signals (efecto de las excepciones)
  - @invariant (invariantes que deben respetar los objetos)

JML

- @requires: precondiciones, es decir, propiedades que debe satisfacer el estado en el que se entra a la ejecución del método. Por ejemplo: $x \geq 0$ para cómputo de la raíz cuadrada, o $l != null$ para una lista que se va a iterar.

- @ensures: postcondiciones, descripción del estado final en función del estado inicial. Por ejemplo: $\result * \result = \old(x)$.

- @signals: propiedad que debe valer cuando un tipo de excepción es lanzado. Por ejemplo: “@signals (Exception e) false” indica que nunca se debiera lanzar una excepción. “@signals (RunTimeException e) x == null” indica que si se lanza una RunTimeException, la variable x debe tener el valor null.

- @invariant: propiedad que deben cumplir los objetos de la clase para ser considerados válidos. Debe ser válida en el estado inicial, y ser válida al acabar la ejecución de los métodos. Ejemplo: \forall Node n; \reach(head, Node, next).has(n) implies !\reach(n.next, Node, next).has(n) (no hay ciclos)

JML: Ejemplo

```
public class BinTree {

    /*@
    @ invariant (\forall Node n;
    @     \reach(root, Node, left + right).has(n) == true;
    @     \reach(n.right, Node, right + left).has(n) == false &&
    @     \reach(n.left, Node, left + right).has(n) == false);
    @
    @ invariant (\forall Node n;
    @     \reach(root, Node, left + right).has(n) == true;
    @     (\forall Node m; \reach(n.left, Node, left + right).has(m) == true; m.key <= n.key) &&
    @     (\forall Node m; \reach(n.right, Node, left + right).has(m) == true; m.key > n.key));
    @
    @ invariant size == \reach(root, Node, left + right).int_size();
    @
    @ invariant (\forall Node n;
    @       \reach(root, Node, left + right).has(n) == true;
    @       (n.left != null ==> n.left.parent == n) && (n.right != null ==> n.right.parent == n));
    @
    @ invariant root != null ==> root.parent == null;
    @*/
    public /*@nullable@*/ Node root;

    public int size;

    public BinTree() {
    }
```

```
/*@
  @ requires true;
  @
  @ ensures (\result == true) <==> (\exists Node n;
  @                 \reach(root, Node, left+right).has(n) == true;
  @                 n.key == k);
  @
  @ ensures (\forall Node n;
  @                 \reach(root, Node, left+right).has(n);
  @                 \old(\reach(root, Node, left+right)).has(n));
  @
  @ ensures (\forall Node n;
  @                 \old(\reach(root, Node, left+right)).has(n);
  @                 \reach(root, Node, left+right).has(n));
  @
  @ signals (RuntimeException e) false;
  @*/
  public boolean contains( int k ) {
      Node current = root;
     //@decreasing \reach(current, Node, left+right).int_size();
      while (current != null) {
         if (current.key > k) {
             current = current.left;
          } else {
              if (k > current.key) {
                  current = current.right;
              } else {
                  return true;
              }
          }
      }
      return false;
  }
```

TACO (Translation of Annotated Code)

Demo

[*mfrias@itba.edu.ar*](mailto:mfrias@itba.edu.ar)

Cómo funciona TACO?

- Para entenderlo necesitamos introducir tres conceptos:
    - Corrección parcial y total, y
    - Precondición más débil,
    - La relación entre ambos.

Corrección Parcial

- Dado un programa P con precondición $\alpha(s)$ y poscondición $\beta(s, s')$, se dice que P es *parcialmente correcto* con respecto a su contrato si:
  - para todo estado inicial s que satisface α, si el programa P termina al ejecutarlo en el estado s, lo hace en un estado s' que satisface $\beta(s,s')$.

Corrección Total

- Dado un programa P con precondición $\alpha(s)$ y poscondición $\beta(s, s')$, se dice que P es *totalmente correcto* con respecto a su contrato si:

    - para todo estado inicial s que satisface α, el programa P termina al ejecutarlo en el estado s y lo hace en un estado s' que satisface $\beta(s,s')$.

Corrección Parcial versus Total: Ejemplo

```
\*@
  @ requires x > 0;
  @ ensures \result = x*x;
  @*\
  public static int square(int x) {
     while (true){
        x++;
     }
  }
```

Cómo se puede verificar si un programa es parcialmente correcto?

- Definición: Dada una poscondición $\beta(s,s')$ y un programa $P(s)$, la *precondición más débil* es una fórmula $\alpha(s)$ que describe a todos los estados a tales que si ejecutamos $P(a)$ se llega a un estado a' que satisface $\beta(a,a')$.

- Se usa más débil en el sentido de más laxa, que admite mayor cantidad de estados.

- Notación: $wp(P, \beta)$.

Cómputo de la Precondición más Débil (Dijkstra, 1975)

- Vamos a ver un algoritmo para computar la precondición más débil.
- $wp(x := t, \alpha) = \alpha[x/t]$
- $wp(P1;P2, \alpha) = wp(P1, wp(P2, \alpha))$
- $wp(if C then P1 else P2, \alpha) = (C => wp(P1,\alpha)) && (!C => wp(P2.\alpha)$
- $wp(if C then P1, \alpha) = (C => wp(P1,\alpha)) && (!C => wp(skip, \alpha) = (C => wp(P1,\alpha)) && (!C => \alpha)
- $wp(assert \beta, \alpha) = \beta && \alpha$.
- $wp(while C do P1 od, \alpha) = ???$