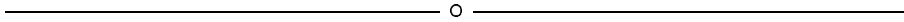


Traduciendo a Java

Procedimientos y Funciones

En este capítulo se muestra cómo traducir las definiciones de procedimientos y funciones, y las llamadas a los mismos. Se señalan algunas limitaciones de JAVA en cuanto a la clasificación entrada/salida de los parámetros de procedimientos y funciones.



0 Procedimientos

El esquema de traducción para la definición de procedimientos es el siguiente:

<pre>“proc <nombre> (<parametros PL>)” { Pre: <precondicion> } { Post: <postcondicion> } [<cuerpo PL>]</pre>	\rightsquigarrow <pre>“private static void <nombre> (<parametros Java>)” /* Pre: <precondicion> */ /* Post: <postcondicion> */ { <cuerpo Java> }</pre>
---	--

Analicemos las distintas partes de este esquema de traducción. La palabra clave “**proc**”, por procedimiento, se traduce a “**private static void**”. Primero, usamos **private** porque asumimos que el procedimiento que estamos definiendo es para nuestro uso interno y no para ser usado por otros programas (en cuyo caso debería ser **public**). Esto es, el procedimiento resuelve uno de los subproblemas en los que decidimos dividir nuestro problema principal, pero no es de interés para quien “desde afuera” quiera usar nuestro programa. A tal persona sólo le interesa la solución del problema completo, a la que puede acceder a través del programa principal, que es público (ver **public** en el encabezado de **main**). Segundo, es **static** porque en terminología de JAVA un procedimiento es un “método estático” (por el momento no necesitamos entender qué significa esto). Tercero, es **void** pues es un procedimiento y no una función. Una función debe devolver un valor y, por lo tanto, su definición debe indicar cuál es el tipo de tal valor. Como un procedimiento no devuelve ningún valor, se dice que “el tipo del valor a devolver está vacante” (vacante \approx **void**).

Con respecto a los parámetros o argumentos del procedimiento, en la declaración de éstos se coloca el tipo antes del nombre, tal como antes vimos que debe hacerse en las declaraciones de variables. Los parámetros se separan por comas (“,”) y cada uno va acompañado de su tipo (sin poder abreviar que varios parámetros sean del mismo tipo). Con respecto a definir cuáles parámetros son de **entrada**, cuáles de **salida** y cuáles de **entrada-salida**, JAVA no provee mucha flexibilidad. Todo parámetro que sea de un tipo básico, como los tipos **int**, **double**, **boolean** y **char**, se asume que es de **entrada**; y todo parámetro cuyo tipo sea una “clase” se asume que es de **entrada-salida**. Por ahora, sólo nos interesa saber que una “clase” es un tipo compuesto, que puede haber sido definido por un programador o puede estar provisto por las librerías estándar de JAVA. En JAVA se utiliza la convención de dar a las clases nombres que empiecen en mayúscula, mientras que los tipos básicos provistos por el lenguaje, como los cuatro antes enumerados, tienen nombres que empiezan en minúscula; esta regla nos puede servir por el momento para diferenciar tipos básicos de tipos “clases”.

Veamos un ejemplo. Si tenemos la siguiente definición de procedimiento en el pseudo-lenguaje:

```

proc dibujarRectangulo (entrada-salida mt : MaquinaTrazados; entrada xsi, ysi, xid, yid : entero)
  { Pre: ... }
  { Post: ... }
[
  ...
]

```

definición en la que los tipos de los parámetros corresponden perfectamente a la clasificación entrada/salida que por defecto utiliza JAVA, ya que *MaquinaTrazados* es una clase y *entero* es un tipo básico, su traducción sería:

```

private static void dibujarRectangulo (MaquinaTrazados mt,
                                       int xsi, int ysi, int xid, int yid)
{
  /* Pre:   ... */
  /* Post:  ... */
  {
    ...
  }
}

```

Cuando la clasificación entrada/salida de nuestros parámetros no coincida con la que por defecto aplica JAVA, no podremos traducir el procedimiento. En el caso de tener un único parámetro de **salida** que sea de un tipo básico, podríamos resolver el problema convirtiendo el procedimiento a función y luego traduciendo la función a JAVA.

Con respecto a los parámetros de **entrada**, la guía de teoría indica que, si éstos son mencionados en la post-condición del procedimiento, éstos no deben ser modificados en el cuerpo del mismo. De hecho, es saludable que, en cualquier caso, los parámetros de **entrada** nunca sean modificados en un procedimiento, se mencionen éstos o no en la postcondición. De acuerdo a esta convención, nos interesa declarar en JAVA a tales parámetros como **final**, tal como antes hacíamos con las constantes (ver traducción de **const**), para indicar que éstos no pueden cambiar de valor en el cuerpo. En nuestro ejemplo anterior, el encabezado del procedimiento cambiaría y quedaría como se indica a continuación:

```

private static void dibujarRectangulo (MaquinaTrazados mt,
                                       final int xsi, final int ysi,
                                       final int xid, final int yid)
{
  /* Pre:   ... */
  /* Post:  ... */
  {
    ...
  }
}

```

En la documentación del procedimiento, tal como hicimos en el programa principal, estamos manejando a la especificación pre/post-condición como comentarios. Posteriormente aprenderemos a utilizar herramientas que manejan a la pre- y la post-condición de una manera más sofisticada.

No hemos indicado aún dónde se colocan estas definiciones de procedimientos. Éstas deben ir dentro de la clase principal, al mismo nivel en que se coloca el programa principal **main** (puede ser antes de éste o después de éste). Por ejemplo:

```

class ProblemaCompleto
{
  public static void main (String[] args)
  {
    ...
  }
}

```

```

}

private static void dibujarRectangulo ( ... )
    /* Pre:   ... */
    /* Post:  ... */
{
    ...
}
}

```

Por último, las llamadas a procedimientos se escriben en JAVA exactamente igual a como se escriben en el pseudo-lenguaje.

1 Funciones

La traducción de funciones a JAVA es casi igual a la de procedimientos. La única diferencia está en el “tipo” `void` que se asocia a los procedimientos; en el caso de una función, este tipo debe ser el tipo del valor devuelto por ésta. El esquema de traducción quedaría así:

$$\begin{array}{l}
 \text{“} \langle \text{tipo PL} \rangle \text{ func } \langle \text{nombre} \rangle (\langle \text{params PL} \rangle) \text{”} \rightsquigarrow \text{“private static } \langle \text{tipo Java} \rangle \langle \text{nombre} \rangle (\langle \text{params Java} \rangle) \text{”} . \\
 \{ \text{Pre: } \langle \text{precondicion} \rangle \} \qquad \qquad \qquad \text{/* Pre: } \langle \text{precondicion} \rangle \text{ */} \\
 \{ \text{Post: } \langle \text{postcondicion} \rangle \} \qquad \qquad \qquad \text{/* Post: } \langle \text{postcondicion} \rangle \text{ */} \\
 [\qquad \qquad \qquad \qquad \qquad \qquad \qquad \qquad \{ \\
 \langle \text{cuerpo PL} \rangle \qquad \qquad \qquad \qquad \qquad \qquad \langle \text{cuerpo Java} \rangle \\
] \qquad \qquad \qquad \qquad \qquad \qquad \qquad \qquad \}
 \end{array}$$

La instrucción para especificar el valor a ser devuelto por una función en JAVA es `return`. Tenemos entonces el siguiente esquema de traducción:

$$\text{“devolver } (\langle \text{expresion PL} \rangle) \text{”} \rightsquigarrow \text{“return } \langle \text{expresion Java} \rangle ; \text{”} .$$

Note que la instrucción `return` en JAVA debe terminar en “;” por ser una instrucción simple.