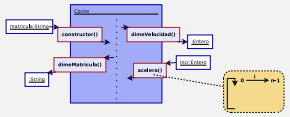
Programación



Resumen de Notación para Diseño, Algoritmos y Protocolos

UNIVERSIDAD POLITECNICA DE VALENCIA

Escola Politècnica Superior de Gandia

DSIIC

Departament de Sistemes Informàtics i Computació

Notación para el Diseño de la Arquitectura de un Programa

Nombres de tipos: empiezan por mayúscula. Nombres de variable: empiezan por minúscula. Nombres de función empiezan por minúscula y tienen un verbo.

Declaración de parámetro: `nombre:Tipo`. Si el nombre no importa, indicar sólo el tipo.

Tipos simples

- número natural: $\mathbb{N}$, número entero $\mathbb{Z}$, número real: $\mathbb{R}$.
- una letra: **Carácter, ASCII**.
- un texto (varias letras): **Texto**.
  (En realidad, Texto = [Caracter]).
- enumerados:
  `NuevoTipo = {VALOR1, VALOR2, ..., VALORN}`
  Ejemplo:
  `Color = {ROJO, AZUL, VERDE}`
- booleano (verdadero o falso): **VoF, Booleano**.
  (En realidad, VoF = {FALSO, VERDADERO}).

Vector = Lista = Array = []

Se sabe el tamaño y se puede acceder mediante índice a cada casilla. No distinguimos entre vector, lista o array (si no es así, indicarlo mediante comentario).

`[Tipo]`

o

`Vector<Tipo>`

Ejemplo: vector o lista o array de enteros:

`numeros:[ℤ]`
`numeros:Lista<ℤ>`
`numeros:Vector<ℤ>`
`numeros:Array<ℤ>`

(todos sinónimos, en principio).

Tuplas = tipos estructurados

`NuevoTipo = (variable1:Tipo1, variable2:Tipo2, ...)`

o

NuevoTipo
`variable1:Tipo1`
`variable2:Tipo2`
`variable3:Tipo3`

Ejemplo:
`Coordenada = (latitud:ℝ, longitud:ℝ)`

o

Coordenada
`latitud:ℝ`
`longitud:ℝ`

Funciones

Sólo se indica qué recibe y qué devuelve una función, *nunca* el mecanismo usado para transmitir la información.

En una línea

TipoEntrada1, ... → funcion() → TipoSalida1, ...

En varias líneas

`TipoEntrada1 | ...` →
`funcion()`
`TipoSalida1 | ...` ←

Función que puede disparar una excepción.

`TipoEntrada1 | ...` →
`funcion()`
`TipoSalida1 | TipoSalida2` | `Error` ←

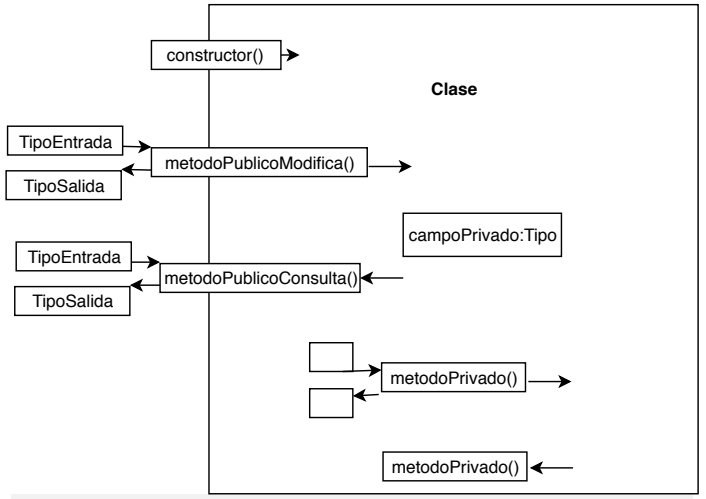
habiendo definido Error como tipo enumerado.

Ejemplo

`dividendo:ℕ | divisor:ℕ` →
`dividir()`
`cociente:ℕ | resto:ℕ` | `Error` ←

Clases

Privado: totalmente contenido. Público: sobresale de la clase. Métodos: diseñados como las funciones añadiendo una flecha hacia la clase para los "mutadores", y de la clase al método para los "accesores".



- La clase A *tiene* la clase B: `A` ◇⟼ `B`
- La clase A hereda de la clase B: `A` ⟶ `B`
- ◇ La clase A implementa el interfaz B: `A` ⋯→ `B`

Ejemplos[1]

- $\mathbb{Z} \rightarrow$ duplicar() $\rightarrow \mathbb{Z}$

Todos las siguientes son posibles implementaciones del **mismo** diseño anterior.

```c
// C
int duplicar( int n ) {
    return 2*n;
}
```

```c
// C
void duplicar( int n, int * pRes ) {
     (*pRes) = 2*n;
}
```

```c
// C
void duplicar( int * pn ) {
     (*pn) = 2 * (*pn);
}
```

```java
// Java
 public static void duplicar( int n, int[] array ) {
     array[0] = 2*n;
 }
```

```java
// Java
 public static void duplicar( int n, int[] array, int pos ) {
     array[pos] = 2*n;
 }
```

```javascript
// JavaScript
function duplicar( n, array ) {
     array[0] = 2*n
}
```

```javascript
// JavaScript
function duplicar( n, callback ) {
     callback( 2*n )
}
```

```javascript
// JavaScript
function duplicar( n ) {
 var prom = new Promise( function( resolver, rechazar ) {
  resolver( 2*n )
 })
 return prom
} // ()
```

```php
<?php
  echo 2 * intval( $_GET[ "valor" ] );
?>
```

- $\mathbb{R}, \mathbb{R} \rightarrow$ ordenar() $\rightarrow \mathbb{R}, \mathbb{R}$

Posibles implementaciones

```c
// C
void ordenar( int a, int b, int * pMin, int * pMay ) {
     if( a < b ) {
         (*pMin) = a;
         (*pMay) = b;
         return;
     }
     (*pMin) = b;
     (*pMay) = a;
}
```

```c
// C
void ordenar( int * pa ) {
     int aux;
     if( pa[0] > pa[1] ) {
         aux = pa[0];
         pa[0] = pa[1];
         pa[1] = aux;
     }
}
```

```javascript
// JavaScript
function ordenar( a, b, callback ) {
     if( a < b ) {
         callback( a, b )
         return
     }
     callback( b, a )
}
```

```java
// Java
 public static void ordenar( int pos, int[] array ) {
     int aux;
     if( pa[pos] > pa[pos+1] ) {
         aux = pa[pos];
         pa[pos] = pa[pos+1];
         pa[pos+1] = aux;
     }
 }
```

- (x:$\mathbb{R}$, y:$\mathbb{R}$) $\rightarrow$ intercambiar() $\rightarrow$ (x:$\mathbb{R}$, y:$\mathbb{R}$)

A diferencia el caso anterior ($\mathbb{R}, \mathbb{R} \rightarrow$ f() $\rightarrow \mathbb{R}, \mathbb{R}$), ahora se entiende que los dos $\mathbb{R}$ están vinculados formando una tupla. Por tanto, sería preferible disponer de un tipo

Coordenada = (x:$\mathbb{R}$, y:$\mathbb{R}$)

Coordenada $\rightarrow$ intercambiar() $\rightarrow$ Coordenada

Posibles implementaciones

```c
// C
typedef struct {
  double x;
  double y;
} Coordenada;

Coordenada intercambiar( Coordenada c ) {
  Coordenada res;
  res.x = c.y;
  res.y = c.x;
  return res;
}
```

```c
// C
void intercambiar( Coordenada * pc ) {
  double aux = (*pc).x;
  (*pc).x = (*pc).y;
  (*pc).y = aux;
}
```

[1] En ningún caso debe entenderse que los ejemplos mostrados para cada lenguaje son la forma en que debe o conviene implementarse el correspondiente diseño. Unicamente ilustran las múltiples posibilidades de implementación de cada diseño.

```java
// Java
class  Coordenada {
 public double x; public double y;
 public Coordenada( double x, double y ) {
  this.x = x; this.y = y;
 }
}
```

```java
// Java
public static Coordenada intercambiar1( Coordenada c ) {
 return new Coordenada( c.y, c.x );
}
```

```java
// Java
public static void intercambiar2( Coordenada[] pc ) {
  double aux = pc[0].x;
  pc[0].x = pc[0].y;
  pc[0].y = aux;
}
```

```javascript
// JavaScript
function intercambiar( co ) {
 return {x : co.y, y : co.x} // JSON
}
```

```javascript
// JavaScript
function intercambiar( co, callback ) {
 callback( {x : co.y, y : co.x} )
}
```

- $[\mathbb{R}]$, $(\mathbb{R} \to \text{VoF}) \to \text{filtrar}() \to [\mathbb{R}]$

```c
// C
typedef  int FunCondicion (double);

void filtrar( double * o, int to, FunCondicion cond,
              double * d, int * td) {
   (*td) = 0;
   for( int i=0; i<=to-1; i++ ) {
      if ( cond( o[i] ) ) {
         d[ (*td) ] = o[i];
         (*td)++;
      }
   }
}
```

```cpp
// C++11
void filtrar( vector<double> & o,  function<bool(double)> cond,
              vector<double> & d ) {
   d.clear();
   for( auto & x : o ) {
      if ( cond( x ) ) {
         d.push_back( x );
      }
   }
}
```

```java
// Java
interface Condicion { boolean condicion( double x ); }

public static void filtrar( List<Double> o, Condicion c,
                            ArrayList<Double> d ) {
   d.clear();
   for( double x : o ) {
      if( c.condicion( x ) ) {
         d.add( x );
      }
   }
} // ()
```

```javascript
// JavaScript
function filtrar( o, cond, d ) {
 d.length = 0
 for( var x of o ) {
  if( cond( x ) ) {
   d.push( x )
  }
 }
} // ()
```

El anterior y el siguiente, en realidad son

$[T]$, $(T \to \text{VoF}) \to \text{filtrar}() \to [T]$

para cualquier tipo T.

```javascript
function filtrar( o, cond, callback ) {
 res = []
 for( var x of o ) {
  if( cond( x ) ) {
   res.push( x )
  }
 }
  callback( res )
} // ()
```

Notación para Algoritmos

Definir los parámetros de entrada y de salida del algoritmo, (se puede usar la notación para el diseño de funciones). Indicar las condiciones que deben cumplir los parámetros.

- Asignación

variable ← valor

- Condiciones

si condicion

acciones cuando la condición es verdadera

si no

acciones cuando la condición es falsa

◇

si condicion

acciones cuando la condición es verdadera

- Repeticiones

condicion

acciones

significa `while( condicion )`

◇

$a \xrightarrow{i} b$

acciones

significa `for (i=a; i<=b; i++)`

◇

$a \xrightarrow{i(-2)} b$

acciones

significa `for (i=a; i>=b; i=i-2)`

◇

$a \xrightarrow{i} b \wedge$ condicion

acciones

significa `for (i=a; i<=b && condicion ; i++)`

◇

$\forall e \in l$

acciones

significa `for( var a of l )`, (para cada elemento de un array).

Ejemplos

- Algoritmos de ordenación

a:$[\mathbb{R}]$ → ordenar() → a':$[\mathbb{R}]$

N es el tamaño de a. La ordenación se realiza sobre el mismo array ($a = a'$). Al terminar, $(\forall i | 0 \leq i \leq N-2 : a[i] <= a[i+1])$.

◇ Algoritmo de la burbuja

$N-2 \xrightarrow{i\ (-1)} 0$

$0 \xrightarrow{j} i$

si $a[j] < a[j+1]$

intercambiar casillas j y $j+1$

◇ Algoritmo de inserción

$1 \xrightarrow{i} N-1$

$i \xrightarrow{j\ (-1)} 1 \wedge a[j-1] > a[j]$

intercambiar casillas $j-1$ y j

Notación para Protocolos

Definir los mensajes que puede enviar cada parte: mensajes del cliente, mensajes del servidor. Ejemplo:

```
LOGIN <usuario:Texto>
<contraseña:Texto>
```

En este mensaje se ve que:

- Tiene dos líneas
- Hay una parte obligatoria fija (en mayúscula): LOGIN
- Hay dos partes obligatorias que deben reemplazarse con un valor.

Por tanto, un ejemplo del anterior mensaje podría ser:

```
LOGIN pepe
1234
```

Para simplifcar los mensajes de error, utilizar el siguiente tipo de mensaje.

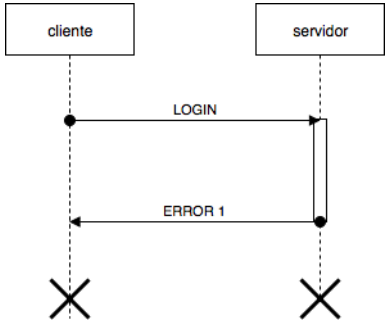
```
ERROR <numero de error:ℕ>
```

Habrá, eso sí, que definir qué significa cada número de error. Por ejemplo,

```
ERROR 1
```

podría significar (en el caso anterior) “login incorrecto”.

Finalmente, para describir las interacciones del protocolo utiliza el siguiente formato:



8 abril 2019