Departamento de Informática | Universidad de Valladolid

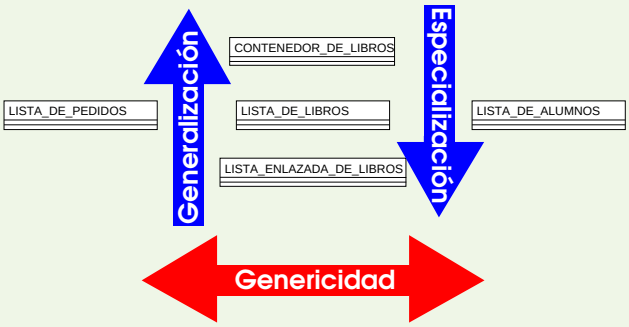
Programación II (I.T.I. de Gestión)

Genericidad y Herencia

Félix Prieto
Esperanza Manso

Curso 2009/10

Dimensiones de la «generalidad»



Alternativas para conseguir genericidad

- Existen TAD que almacenan a otros
  - *PILA COLA VECTOR LISTA*
- ¿Incluimos el tipo del contenido en la definición del contenedor?
  - Java[1] o SmallTalk no lo hacen, pero no pueden hacer un chequeo fuerte de tipos
  - Eiffel lo hace, pero al instanciar el contenedor debemos proporcionar un tipo explícito para el contenido

[1] Hasta la versión 1.5

Una clase genérica

```
class PILA[G]
feature
 contador: INTEGER
   -- Número de elementos
 vacia: BOOLEAN is
   -- ¿Está vacia?
  do ... end
 llena: BOOLEAN is
   -- ¿Está llena?
  do ... end
 cima: G is
   -- Elemento de la cima
  do ... end
 mete( x:G) is
   -- Introduce 'x' por la cima
  do ... end
 saca is
   -- Quita la cima
   do ... end
end -- Clase PILA
```

- Utilizamos un *parámetro genérico*, `G` en este caso
- Para instanciar la clase debemos proporcionar como *parámetro actual* un tipo, por ejemplo `a:PILA[INTEGER]`
- De una clase genérica se derivan tantos tipos como parámetros actuales podamos proporcionar.
- El parámetro puede aparecer en cualquier declaración de entidades (incluyendo retorno de función) que se realice en el texto de la clase.
- Debemos reconsiderar las reglas que controlan la sustitución de *argumentos formales* por *argumentos actuales* en el paso de mensajes

Genericidad y mensajes (I)

Regla 1 Sea `C` una clase no genérica y `f(a:T):U is ...` la definición de un método que aparece en ella. La llamada `x.f(d)` que aparece en un método de la clase `B` es correcta si:

- `x` es de tipo `C` (o compatible por herencia).
- `f` está exportada a `B` (explícita o implícitamente).
- `d` es de tipo `T` (o compatible por herencia).

En este caso el resultado de `f` es de tipo `U`.

Genericidad y mensajes (II)

Regla 2 Sea `C` una clase genérica con parámetro formal `G` y `h(a:G):G is ...` la definición de un método que aparece en ella. La llamada `y.h(e)` que aparece en un método de la clase `B` es correcta si existe un tipo `V`, parámetro actual de `C` tal que :

- `y` es de tipo `C[V]` (o compatible por herencia).
- `h` está exportada a `B` (explícita o implícitamente).
- `e` es de tipo `V` (o compatible por herencia).

En este caso el resultado de `h` es de tipo `V`.

Restricciones asociadas a la genericidad

Regla 3 Si `x` es una entidad cuyo tipo `G` es parámetro formal de una clase genérica las operaciones permitidas sobre `x` son:

- Uos en el lado derecho o izquierdo de una conexión cuando el otro lado también es de tipo `G`
- Uso en el lado derecho o izquierdo de un operador `=` o `/=` cuando el otro lado es también de tipo `G`
- Uso como parámetro actual para una rutina correspondiendo a un parámetro formal declarado de tipo `G` o `ANY`.
- Uso como objetivo de una característica de `ANY`.

Introducción a la herencia

- La herencia permite implementar los mecanismos de generalización y especialización
- Proporciona la posibilidad de agrupar las pautas comunes a subgrupos
- Permite relajar el control de los tipos, manteniendo a pesar de ello un chequeo fuerte de tipos
- Todo ello proporciona la posibilidad de combinar polimorfismo y ligadura dinámica

Una clase sencilla

```
class POLIGONO
 create ...
feature -- Acceso
 contador: INTEGER
   -- Número de vértices
 perimetro: DOUBLE is
  do ... end
 muestra is
  do ... end
feature -- Modificación
 rota (centro:PUNTO; angulo: DOUBLE) is
  do ... end
 traslada (a,b:DOUBLE) is
  do ... end
feature {NONE}-- Implementación
 vertices: LINKED_LIST[PUNTO]
end -- Class POLIGONO
```


Una clase construida mediante herencia

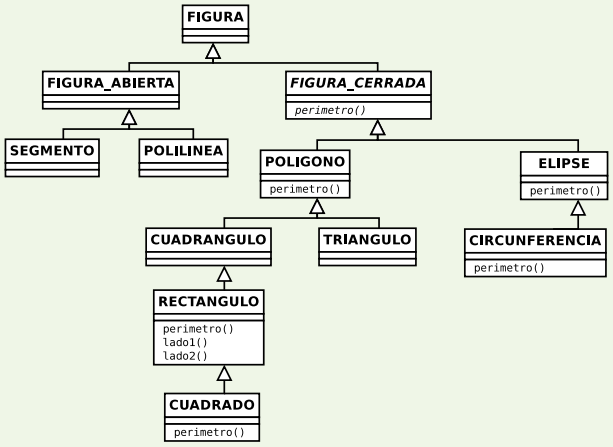
```
class RECTANGULO
inherit POLIGONO -- Tenemos todas sus características
 redefine perimetro -- Aportamos un nuevo perímetro
 end
create make
feature -- La creación es también nueva
 make(centro:PUNTO;s1,s2,angulo: DOUBLE) is
feature
 lado1, lado2 : DOUBLE
 diagonal : DOUBLE -- Como lados y diagonal
 perimetro: DOUBLE is
 do
  Result:= lado1*2 + lado2*2
 end
```


Primeras nociones

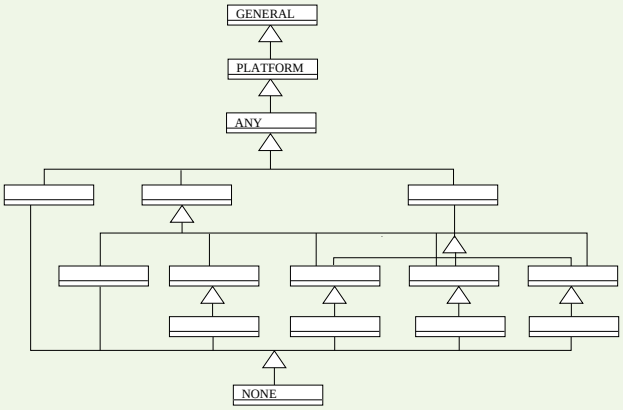
- **Definición 1** Llamaremos clase *descendiente* de C a:
  - La propia clase C
  - Recursivamente cualquier heredera de una descendiente de C

  Diremos que B es *descendiente propio* de C si es descendiente de ella y ambas son distintas.
  Llamaremos *antecesor* o *ancestro* de C a cualquier clase de la que C sea descendiente.
- **Regla 4** Un método que tiene la condición de método de creación en un ancestro no mantiene su condición especial en los descendientes.

Jerarquía de herencia de las figuras



Jerarquía SmartEiffel (simplificada)



El concepto de polimorfismo

- **Definición 2** Si polimorfismo es la habilidad de tener varias formas, para nosotros polimorfismo va a ser la habilidad de las referencias para estar conectadas a objetos de tipos distintos en tiempo de ejecución, aunque siempre con las restricciones que se deducen de la herencia.
- Si definimos
  `p:POLIGONO; r:RECTANGULO; t: TRIANGULO`
  - `p:=r` `p:=t` son conexiones correctas
  - `r:=p` `r:=t` son conexiones erróneas
- Eiffel dispone del «intento de asignación» (`?=`), pero sólo hay que utilizarlo en casos de auténtica necesidad

Límites del polimorfismo

- **Regla 5** En una llamada `x.f` donde el tipo de x está basado en una clase C debe ocurrir que `f` sea característica de uno de los ancestros de C
- **Definición 3** Un tipo U es conforme a un tipo T si y solo si la clase base de U es descendiente de la clase base de T y, en el caso de tipos genéricos formalmente derivados cada parámetro actual de U es conforme a cada parámetro actual de T.
- **Regla 6** Una conexión de fuente x y objetivo y es válida si el tipo de x es conforme con el tipo de y.

Ligadura dinámica o tardía

Cuando un objeto está referenciado desde una entidad polimorfa sólo puede recibir a través de ella mensajes conocidos por la clase base de la entidad, sin embargo la versión del método que se ejecutará será la definida en la clase base del objeto.
La ligadura dinámica permite al programador ignorar la versión del método que será realmente ejecutada.

```
metodo(p:POLIGONO) is
  do -- ¿'p' es un rectángulo o un triángulo?
    print(p.perimetro)
  end
```

Genericidad restringida

```
class VECTOR[G] inherit
  ARRAY[G]
create make
feature {ANY}
  infix "+" (otro: like current): like current is
  -- Suma incorrecta. Requiere (G->NUMERIC)
  local   i:INTEGER
  do
    create Result.make(lower,upper)
    from i:=lower until i> upper loop
      Result.put(item(i)+otro.item(i),i)
      i:=i+1
    end
  end
...
end -- class VECTOR(G)
```