Programozási nyelvek és paradigmák

Generikus programozás Eiffelben

Kozsik Tamás (2020)

Generikus osztály

```
class ARRAY [G]
...
feature -- Measurement
    lower, upper: INTEGER
    count, capacity: INTEGER
feature -- Initialization
    make_empty
    make_filled ( a_default_value: G;
                  min_index, max_index: INTEGER )
    make ( min_index, max_index: INTEGER )
        obsolete
feature -- Access
    item alias "[]" ( i: INTEGER ): G assign put
feature -- Element change
    put ( v: G; i: INTEGER )
```

Osztály és típus

- Osztály: amit definiálunk
  - Lehet generikus is
- Típus: amivel típusozunk
  - Lehet egy osztály
  - Lehet egy felparaméterezett generikus osztály
  - stb.

3-operandusú bracket-operátor, balérték is

```
class MATRIX      -- You can use this like: m[1,1] := 1.0
create
   make
feature    -- interface

   rows, cols: INTEGER

   item alias "[]" ( i, j: INTEGER ) : REAL assign put
      require 1 <= i; i <= rows; 1 <= j; j <= cols

   put( val: REAL; i, j: INTEGER )
      require 1 <= i; i <= rows; 1 <= j; j <= cols
      ensure val ~ item(i,j)

invariant
   rows > 0 and cols > 0
end -- class MATRIX
```

Mátrix ábrázolása

```
feature {NONE}
   data: attached ARRAY[REAL]
   make( nr_rows, nr_cols: INTEGER )
      require nr_rows > 0; nr_cols > 0
      do
         rows := nr_rows
         cols := nr_cols
         create data.make_filled(0.0, 1, rows*cols)
      ensure rows = nr_rows; cols = nr_cols
      end -- make
invariant
   rows > 0 and cols > 0
   data.lower = 1; data.upper = rows*cols
```

Mátrix műveletei

```
feature

   item alias "[]" ( i, j: INTEGER ) : REAL assign put
      do
         Result := data[(i-1)*cols+j]
      end -- item

   put( val: REAL; i, j: INTEGER )
      do
         data[(i-1)*cols+j] := val
      end -- put
```

Saját generikus osztály – vázlat

```
class STACK[T]
create make
feature
   size: INTEGER
   capacity: INTEGER
   push( element: attached T )
      require size < capacity
      ensure top = element ; size = old size + 1
   top: attached T
      require size /= 0
      ensure size = old size
   pop
      require size /= 0
      ensure size = old size - 1
invariant
   0 <= size ; size <= capacity
end -- class STACK
```

Saját generikus osztály – belső ábrázolás (obsolate)

```
class STACK[T]
...
feature {NONE}
   data: attached ARRAY[attached T]

   make( capacity_: INTEGER )
      require capacity_ > 0
      do create data.make(1,capacity_)  -- size := 0
      ensure
         size = 0 ; capacity = capacity_
      end -- make

invariant
   data.lower = 1 ; data.upper = capacity
   0 <= size ; size <= capacity
end -- class STACK
```

Saját generikus osztály – műveletek

```
push( element: T )
   require size < capacity
   do size := size + 1 ; data.put(element,size)
   ensure top = element ; size = old size + 1
   end -- push

top: T
   require size /= 0
   do Result := data.item(size)
   ensure size = old size
   end -- top

pop
   require size /= 0
   do size := size - 1
   ensure size = old size - 1
   end -- pop
```

Még egy példa generikus osztályra

```
class MAYBE[T]
create
   nothing, just
feature
   has: BOOLEAN
   item: attached T
      require has_item: has
      ...
feature {NONE}
   value: detachable T
   nothing
      ...
   just( v: attached T )
      ...
invariant
   has implies attached value
end
```

Még egy példa generikus osztályra: implementáció

```
feature {NONE} nothing
   do
      has := False
   end

feature {NONE} just( v: attached T )
   do
      has := True
      value := v
   end

feature item: attached T
   require has_item: has
   do
      check attached value as attached_value then
         Result := attached_value
      end
   end
```