Práctica 9 : SmallTalk

Tomás Felipe Melli

June 30, 2025

Índice

1 Objetos y Mensajes

1.1 Ejercicio 5

Nos piden identificar mensajes, el objeto receptor y los colaboradores.

1. `10 numberOfDigitsInBase: 2`. El selector es `numberOfDigitsInBase:` que recibe el objeto `10` y el colaborador externo es `2`

2. `10 factorial`. El selector es `factorial` y el objeto que recibe el mensaje es el `10`, no hay colaboradores externos.

3. `20 + 3 * 5`. En este caso el selector es el `+` y lo recibe el objeto `20`, como colaborador externo el 3. Luego, se resuelve y `(20 + 3)` recibe el mensaje `*` que tiene como colaborador externo `5`

4. `20 +(3 * 5)`. En este escenario, el `3` recibe el mensaje `*` y como colaborador externo `5` que luego será colaborador externo del mensaje `+` donde el `20` es receptor.

5. `December first, 1985`. El objeto receptor es `December` que recibe el mensaje `first` sin colaboradores externos. A continuación, la coma `,` que es un mensaje de concatenación de colecciones lo recibe el resultado de lo anterior y se le concatena `1985`.

6. `1 = 2 ifTrue: ['What?!']`. El mensaje `=` recibe el objeto `1` con el colaborador externo `2`. Luego, el resultado, que es una instancia de la clase Boolean, recibe el mensaje `ifTrue:` con un `closure` como colaborador externo.

7. `1@1 insideTriangle: 0@0 with: 2@0 with: 0@2`. En este caso, el objeto `1@1` que es una instancia de la clase `Point`, recibe el mensaje `insideTriangle: with: with:` que recibe dos colaboradores externos, el punto 2@0 y 0@2.

8. `'Hello world' indexOf: $o startingAt: 6`. El string recibe el mensaje `indexOf: startingAt:` con dos colaboradores externos, el `caracter` $o y el 6.

9. `(Ordered Collection with: 1) add: 25; add: 35; yourself`. En primer lugar, la clase `Ordered Collection` recibe el mensaje `with:` con el colaborador externo `1` donde se instancia la Ordered Collection. Ese objeto instanciado, a continuación, recibe en `cascada` dos mensajes `add:` con dos números, 25 y 35 como colaboradores externos, y finalmente el mensaje `yourself`.

10. `Object subclass: #algo instanceVariableNames: 'algo' classVariableNames: '' poolDictionaries: '' category: 'categoria'`. La clase `Object` recibe este mensaje de creación de instancia `subclass: instanceVariableNames: classVariableNames: poolDictionaries: category:` con los colaboradores que vemos ahí.

2 Bloques, métodos y colecciones

Para cada una de las siguientes expresiones, indicar qué valor devuelve o explicar por qué se produce un error.

1. `[:x | x + 1] value: 2`. El bloque reemplaza `x := 2` y devuelve `3`.

2. `[|x| x := 10. x + 12] value`. Esto cumple con la sintaxis de un bloque sin parámetros. Evaluado da `22`.

3. `[:x :y | |z| z := x + y ] value: 1 value: 2`. Este bloque toma dos argumentos y evaluado da `3`.

4. `[:x :y | x + 1] value: 1`. El bloque no recibe todos los argumentos que debería y lanza el error `BlockClosure>>vnumArgsE`

5. `[:x | [:y | x + 1]] value: 2`. Devuelve el closure `[:y | 3]`

6. `[[:x | x + 1]] value`. Este closure sin argumentos devuelve el closure anidado.

7. `[:x :y :z | x + y + z] valueWithArguments: (1 2 3)`. Evalúa el closure con una collección de argumentos. Devuelve `6`.

8. `[ |z| z := 10. [:x | x + z]] value value: 10`. En este caso se evalua el closure externo, y al anidado, se le pasa 10 como argumento para evaluar.

2.1 Ejercicio 12

Mostrar ejemplos de los siguientes mensajes que se pueden enviar a colecciones.

1. `collect: aBlock`. En cuis nos dice que `"Evaluate aBlock with each of the receiver's elements as the argument. Collect the resulting values into a collection that is like the receiver. Answer the new collection. "`. Un ejemplo puede ser :

```
coleccion := OrderedCollection with: 1 an OrderedCollection(1) .
coleccion collect: [:a | a + 1] an OrderedCollection(2) .
```

2. `select:`. En cuis nos dice que `"Evaluate aBlock with each of the receiver's elements as the argument. Collect into a new collection like the receiver, only those elements for which aBlock evaluates to true. Answer the new collection."`

```
coleccion := OrderedCollection with: 1 with:2 with: 3. an OrderedCollection(1 2 3) .
coleccion select:[:a | a > 2] an OrderedCollection(3) .
```

3. `inject: into:`. En cuis nos dice, `inject: thisValue into: binaryBlock "Accumulate a running value associated with evaluating the argument,binaryBlock, with the current value and the receiver as block arguments. The initial value is the value of the argument, thisValue.For instance, to sum a collection, use: collect inject: 0 into: [:subTotal :next | subTotal + next].".`

```
coleccion := OrderedCollection with: 1 with:2 with: 3. an OrderedCollection(1 2 3) .
coleccion inject: 0 into:[:subTotal :a | a + subTotal ]  6 .
```

4. `reduce: (o fold:)`. En cuis, `fold: aTwoArgBlock "Evaluate the block with the first two elements of the receiver, then with the result of the first evaluation and the next element, and so on. Answer the result of the final evaluation. If the receiver is empty, raise an error. If the receiver has a single element, answer that element." ('if' 'it' 'is' 'to' 'be' 'it' 'is' 'up' 'to' 'me') fold: [:a :b | a, ' ', b]`.

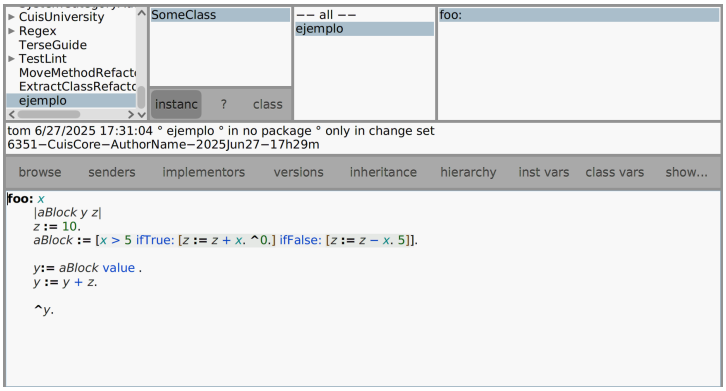
```
coleccion := OrderedCollection with: 1 with:2 with: 3. an OrderedCollection(1 2 3) .
coleccion fold: [:a :b | a + b] 6 .
```

5. `reduceRight:`. En cuis, no lo tengo.

6. `do:`. En cuis, `"Evaluate aBlock with each of the receiver's elements as the argument."`

```
coleccion := OrderedCollection with: 1 with:2 with: 3. an OrderedCollection(1 2 3) .
res := OrderedCollection new.
coleccion do:[:a |  res add: (a+1)].
res. an OrderedCollection(2 3 4) .
```

2.2 Ejercicio 13




```
ejemplo := SomeClass new a SomeClass .

ejemplo foo:4 11 .

Message selector: #foo argument:5 foo (5) .

ejemplo foo:10 0 .
```

2.3 Ejercicio 14

1. #curry

```
BlockClosure>>curry
    ^ [:x | [:y | self value: x value: y ]]

| curried new result |
curried := [:x :res | x + res] curry.
new := curried value: 10.
result := new value: 2.
```

2. #flip

```
BlockClosure>>flip
    ^ [ :a :b | self value: b value: a ]

| bloque invertido |
bloque := [ :x :y | x - y ].
invertido := bloque flip.
invertido value: 3 value: 10.
```

3. #repetirVeces

```
Integer>>repetirVeces: unBloque
    1 to: self do: [:i | unBloque value ]
```

3 Method Dispatch, self y super

3.1 Ejercicio 16

1. Verdad.En Smalltalk, todo es un objeto. Las clases son instancias de metaclases que son a su vez, objetos.
2. Verdadero. En caso de no encontrarlo allí, buscará en su super clase, ya que podría heredarlo de allí.
3. Falso. Si el mensaje lo recibe la clase, es un método de clase, en este caso, de creación de instancia.
4. Falso. La variable de instancia es un atributo de la instancia particular de cierta clase.
5. Falso. Las variables de clase sólo son accesibles para el objeto clase, pero no por las instancias.
6. Verdadero. Cuando miramos el código del método sabemos que `self` se refiere al receptor de ese mensaje.
7. Falso. `super` nos indica que la búsqueda del método (o sea, cuál implementación usar) ocurre desde la super clase del objeto actual hacia arriba en la jerarquía.
8. Falso. Un método de clase puede acceder a las variables de clase, pero no a las de instancia. No necesariamente devuelve una instancia de la clase receptora: puede devolver cualquier objeto.
9. Verdadero. En Smalltalk, las clases son objetos, y sus métodos de clase son en realidad métodos de instancia del objeto clase. Lo mismo con las variables de clase.

3.2 Ejercicio 17

Suponiendo que anObject es una instancia de la clase OneClass que tiene definido el método de instancia aMessage. Al ejecutar la siguiente expresión: anObject aMessage

1. Al objeto que recibe `aMessage`, en este caso, `anObject`.
2. Análogo a lo anterior, `super` queda ligado a `self` ya que lo que sucede es que el mensaje se envía a `self` pero modificando el `methodLookup`.
3. `super == self`. Da `true`.