Guía de Ejercicios

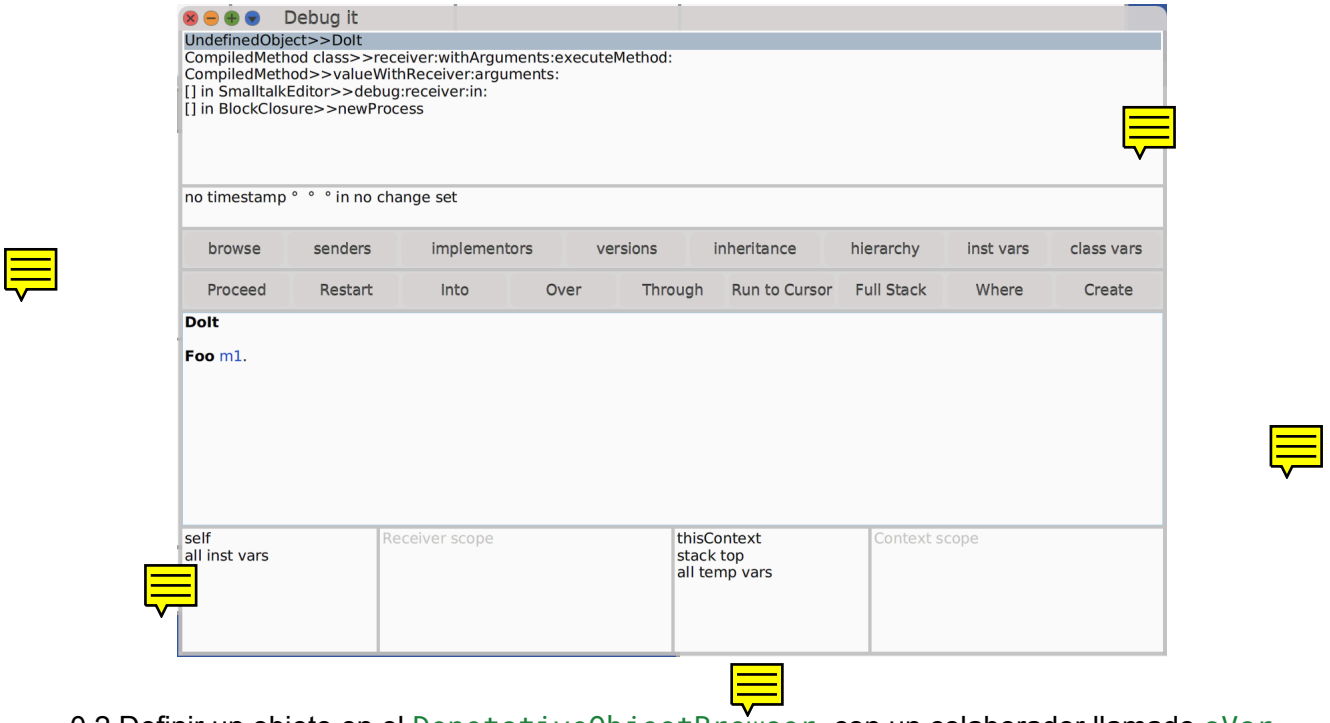
Ingeniería de Software I

Parte 1. Introducción al paradigma, el lenguaje y sus herramientas

Sección 2. Introducción al lenguaje Smalltalk

0[1]. Debugger

0.1 Identificar y nombrar las diferentes partes y secciones del debugger:



0.2 Definir un objeto en el `DenotativeObjectBrowser`, con un colaborador llamado `aVar` y con dos métodos `m1` y `m2`:

```
m1
    |b|

    b := 42.
    aVar := 1.

    ^ self m2: b
```

[1] Como buenos computadores, al primer ejercicio le pusimos el número 0. ¡Lógico! Pregunta: ¿Las colecciones de Smalltalk también se numeran así? 🙂

```
m2: anotherValue

    |b|

    b := 24.
    aVar < 3 ifTrue: [ aVar := aVar + anotherValue].

    ^ aVar + b
```

Luego en el **workspace** enviar el mensaje `m1` al objeto creado utilizando el debugger (*Debug it* en el menú):

```
anObject m1. “Debug it here!”
```

- ¿Cuál es la diferencia entre las acciones `Into`, `Over` y `Through` (en el menú del debugger)?
- ¿Qué sucede al hacer clic en `Restart`?
- Recorrer el código con `Into` hasta la última línea del método `m2`. Luego hacer `Restart`. ¿Dónde queda ubicado el debugger? ¿Cuál es el valor de `aVar`?

Pueden ver la guia del uso de la imagen y debugger aca:

https://www.youtube.com/playlist?list=PLMkq_h36PcLCQiU5gxNmedxdXbl10JYeV

1. Colecciones

1.1 Acerca de algunas colecciones muy utilizadas

a. Array (fixed length collection)

x := Array with: 5 with: 4 with: 3 with: 2.

Sintaxis reducida para crear arrays:

x := #(5 4 3 2).

Para resolver en el Workspace:

- Crear un array usando alguna de las sintaxis anteriores.
- Cambiar el elemento en la primera posición con el valor 42.
- ¿Qué pasa si queremos agregar un elemento en la posición 5?

b. Ordered Collections

x := OrderedCollection with: 4 with: 3 with: 2 with: 1.

Resolver en el Workspace:

- Agregar elemento 42.
- Agregar elemento 2.
- ¿Cuántos elementos tiene la colección?
- ¿Cuántas veces aparece el 2?

c. Sets

x := Set with: 4 with: 3 with: 2 with: 1.

Resolver en el Workspace:

- Agregar elemento 42.
- Agregar elemento 2.
- ¿Cuántos elementos tiene la colección?
- ¿Cuántas veces aparece el 2?

d. Dictionary

x := Dictionary new.
x add: #a->4; add: #b->3; add: #c->1; add: #d->2; yourself.

Resolver en el Workspace:

- Agregar la *key* #e con el *value* 42.
- ¿Cuántos elementos tiene la colección?
- Listar las keys.
- Listar los values
- Obtener el value del key #a.
- Obtener el value del key #z (en caso de no encontrarlo retornar 24)

1.2 Conversión de colecciones

e. Convertir el Array del punto a en una OrderedCollection y en un Set.
f. Convertir el Set del punto c en Array
g. ¿Qué retorna convertir el Dictionary en Array?

1.3 Crear una secuencia de colaboraciones para encontrar los elementos impares en un arreglo.

```
| elements index odds |

elements:= #(1 2 5 6 9).

odds := OrderedCollection new.
index := 1.

[index <= elements size]
whileTrue: [
        ((elements at: index) odd) ifTrue: [odds add: (elements at: index)].
        index := index +1.
        ].
^odds
```

Descripción del código:

| elements index odds | <= son tres colaboradores temporales

elements:= #(1 2 5 6 9) *es una forma de construir una referencia a un arreglo con esos 5 elementos*

odds := OrderedCollection new. es la forma de asignar una colección ordenada vacía a la referencia odds

index := 1. es la asignación inicial del valor 1 a la referencia de index que va a iterar en el ciclo

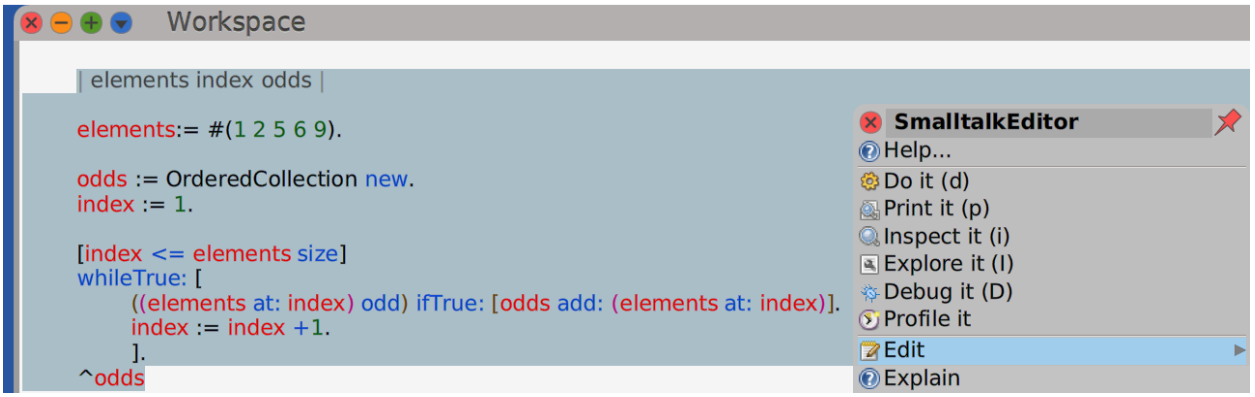
[index <= elements size] Esta es la guarda que se evalúa en cada iteración. Mientras sea verdadera (el índice es menor o igual que el tamaño de la coleccion de entrada) se evaluará el bloque colaborador del *whileTrue*:

```
((elements at: index) odd) ifTrue: [odds add: (elements at: index)].
        index := index +1.
```

Este es el cuerpo del ciclo

^odds este retorno es la última colaboración donde pueden inspeccionar lo obtenido.

1.4 Evaluarlo dentro de un nuevo workspace (encontrar la forma de abrir uno nuevo)



1.5 Usar las opciones. *Do it, print It, Inspect It , Explore It, Debug It*. Comprobar que es lo que hace cada una.

Es muy importante aprender a debugguear ciclos y entender el rol que juegan los dos bloques en un *whileTrue:* (de un lado un bloque con la condición, del otro un bloque con la acción a realizar mientras la condición sea verdadera)

1.6 Cambiar los elementos de la colección elements para comprobar que las colaboraciones funcionan

1.7 Enumerar los problemas que tiene ese algoritmo según lo visto en la carrera.

1.8 Convertir el script de 1.1 sin usar *#whileTrue*, utilizando el mensaje *#do*:, ¿qué ventaja tiene la nueva versión?

1.9 Volver a convertir el algoritmo sin cambiar su comportamiento pero usando el mensaje *#select:* en lugar de #do ¿qué ventaja tiene la nueva versión?

1.10 Crear una secuencia similar a la de 1.1 pero que obtenga el doble de cada elemento de la colección. Por ejemplo elements = #(1 2 5) debería retornar #(2 4 10)

1.11 Reescribir el algoritmo utilizando *while* y luego utilizando *do* ¿Donde se acumulan los resultados?

1.12 Encontrar luego un mensaje mejor en colecciones y dejar el algoritmo más compacto. ¿Qué retorna el nuevo mensaje?

1.13 Crear una nueva secuencia de colaboraciones para encontrar el primer número par, utilizando otro mensaje de colecciones. Como siempre primero con while: luego con do: y luego con un mensaje específico. Ejemplo: dado *#(1 2 5 6 9)* debería retornar 2

1.14 Utilizar la secuencia de colaboraciones con una colección sin pares. Por ejemplo *#(1 5 9)*. ¿Qué ocurre?

1.15 Modificar la secuencia para generar un error en caso de no contener pares utilizando *self error: 'No hay pares'.* Evaluarlo en una colección con pares (retorna el primero) y sin pares (se genera un error con el mensaje específico)

1.16 Sumar los números de una colección utilizando primero *while*, luego *do* y luego un mensaje de sumar colecciones. Hay un mensaje específico para la suma y otro para acumular elementos llamado inject:into: Solucionarlo utilizando ambos.

#(1 5 9) retorna 15 (que es la suma)

1.17 ¿Cuántos colaboradores recibe *inject:into:* ? Pruebe debuggearlo con el menú o poniendo *self halt.* antes de las colaboraciones (esto detendrá la ejecución y abrirá el debugger)

1.18 Crear una nueva secuencia para extraer únicamente las vocales en el orden que aparecen en un *string*.

Implementar un extractor de vocales preservando el orden de un *string* usando #select

Ejemplo, para 'abcdefguijp' debe retornar *'aei'*

1.19 ¿Qué observa con respecto a los *strings* y otras colecciones?

1.20 ¿Conocía estos mensajes de colecciones de materias anteriores? ¿Cómo se llamaban?

2. Bloques (Closures)

a. ¿Cuál es la definición de Blocks que se encuentra en el libro *Smalltalk-80 The Language and its Implementation*.?
b. ¿Qué valor retorna un Block cuando se evalúa (con `value`)?

c. Evaluar en el Workspace lo siguiente:

```
| x |
x := [ y := 1. z := 2. ].
x value.
```

i. ¿Qué sucede si queremos acceder a una variable definida en el bloque desde fuera del bloque?

```
| x |
x := [ y := 1. z := 2. ].
x value.
y.
```

¿Qué sucede al acceder a una variable definida fuera del bloque desde dentro del bloque?

```
| x y |
x := [ y := 1. z := 2. ].
x value.
y.
```

ii. Dé un ejemplo de un bloque con dos parámetros y su evaluación.

3. Símbolos

d. ¿Cuál es la definición de Symbol que se encuentra en el libro *Smalltalk-80 The Language and its Implementation*.?

e. Evalúe en el Workspace:

i. `

```
| x  y |
x := #pepe
y := #pepe
x = y.
```

ii. ¿Cuál es el resultado de concatenar símbolos?

```
#Hello , #World, #!
```

4. Medidas

4.1 Sobre la importancia de las medidas en nuestra profesión y sobre las responsabilidades de los desarrolladores de software en la industria:

- http://www-users.math.umn.edu/~arnold/disasters/ariane.html
- https://motherboard.vice.com/en_us/article/qkvzb5/the-time-nasa-lost-a-mars-orbiter-because-of-a-metric-system-mixup
- https://en.wikipedia.org/wiki/Gimli_Glider

¿Qué tienen en común los casos mencionados en las notas anteriores?

4.2 Evalúe estas colaboraciones

```
10 * peso +  10 * dollar
```

¿Qué resultado esperaba? ¿Cuál Obtuvo?

4.3 Evalúe estas colaboraciones anotando previamente que resultado cree va a obtener

```
 10 * peso +( 10 * dollar)
```

```
 10 * peso +( 10 * dollar) - (2 * dollar)
```

```
 10 * peso +( 10 * dollar) - (2 * dollar) - (8 * dollar)
```

4.4 ¿Qué es peso? inspecciónelo:

peso inspect

4.5 ¿qué es 10 * peso? evalúe:

```
(10 * peso) amount
(10 * peso) unit
```

4.6 Y ¿qué son los números en este contexto?¿Qué unidad llevan?

```
1 amount
1 unit
```

4.7 ¿Cuánto es (10 * peso) + 1 y 1 + (10 * peso) ?

```
 (10 * peso)  * 5
 (10 * peso)  * 5
 (10 * peso)  *  (5 * peso)
```

4.8 Cree el peso nuevamente:

```
peso := BaseUnit nameForOne:  'peso' nameForMany:  'pesos' sign: $$
```

¿Qué representa $$? (Evaluar la expresión ayuda a entenderlo)

4.9 Calcular: (10 * metros) + (500 * centimetros)

Sugerencia:

- Crear los centímetros en función de los metros como una ProportionalDerivedUnit (ver los mensajes de creación de instancia)

4.10 Resuelva el problema del cohete Arianne sobre el cual leyó

```
metros := BaseUnit nameForOne:  'metro' nameForMany:  'metros'.
diezMetros := 10 * metros.

pulgadas :=  BaseUnit nameForOne:  'pulgada' nameForMany:  'pulgadas'.
sesentaPulgadas := 60 * pulgadas.

diezMetros + sesentaPulgadas
```

Sugerencia:

- Cambiar la creación de las pulgadas en función de los metros (como una ProportionalDerivedUnit)

4.11 Definir los bitcoins
Crear un método que retorne el importe en pesos al dia 19/04/2024

4.12 Sumar 1 btc + 10000 pesos

Sugerencia:

- Usar un *MeasureConverter* basándose en una *ConversionTable*

4.13 Construir los Celsius y Fahrenheit en función de los Kelvin para poder realizar en un método

```
(30 * kelvin) + (20 * celsius) + (10 * fahrenheit)
```

Sugerencia:

- Las conversiones ya no son directas

5. Fechas

5.1 Tiempo tradicional.

El tiempo basado en un punto fijo arbitrario y una distancia en segundos

Evaluar:

```
DateAndTime fromSeconds: 0.
```

(DateAndTime fromSeconds: 0) + (Duration days: 1).

Time now.

Time hour: 1 minute: 2 second: 4.

Time now + (Duration hours: 1) “FALLA porque no es un timespan”

(DateAndTime fromSeconds: 0) + (Duration days: 1).

Date today.

Date newDay: 1 month: 2 year: 3 .

Estas clases están en *Kernel-Chronology*

5.2 Inspeccionar usando los objetos del paquete *Aconcagua* que ya viene preinstalado en *CuisUniversity*

FixedGregorianDate today.

FixedGregorianDate today next next.

GregorianDateTime now.

GregorianDateTime now next.

GregorianDateTime now next distanceFrom: GregorianDateTime now.

(GregorianDateTime now next distanceFrom: GregorianDateTime now) convertTo: second / millisecond.

TimeOfDay now.

TimeOfDay now next: (30 * hour).

FixedGregorianDate today year.

FixedGregorianDate today month.

FixedGregorianDate today monthOfYear.

5.2 Obtener el dia lunes de la semana que viene (sumar 7 días) a partir de la fecha de hoy: (*FixedGregorianDate today)*

5.3 Sumarle 24 segundos a la fecha de hoy y ver que pasa.

5.4 Sumarle la cantidad de segundos que hay en un día a la fecha de hoy y ver que pasa.

5.5 Evaluar por separado estas colaboraciones:

a) 2024 isLeap

b) (April, 2024) year isLeap

¿Qué ocurre?

5.6 Corrija la siguiente expresión para que funcione:

TimeOfDay now next: 3600

6. Material Complementario

Para profundizar el conocimiento del ambiente y sus herramientas pueden ver esta serie de videos:

https://www.youtube.com/watch?v=blj7itWxk2Y&list=PLMkq_h36PcLCtLKrrdOKKFV2r267VFH_t&index=1