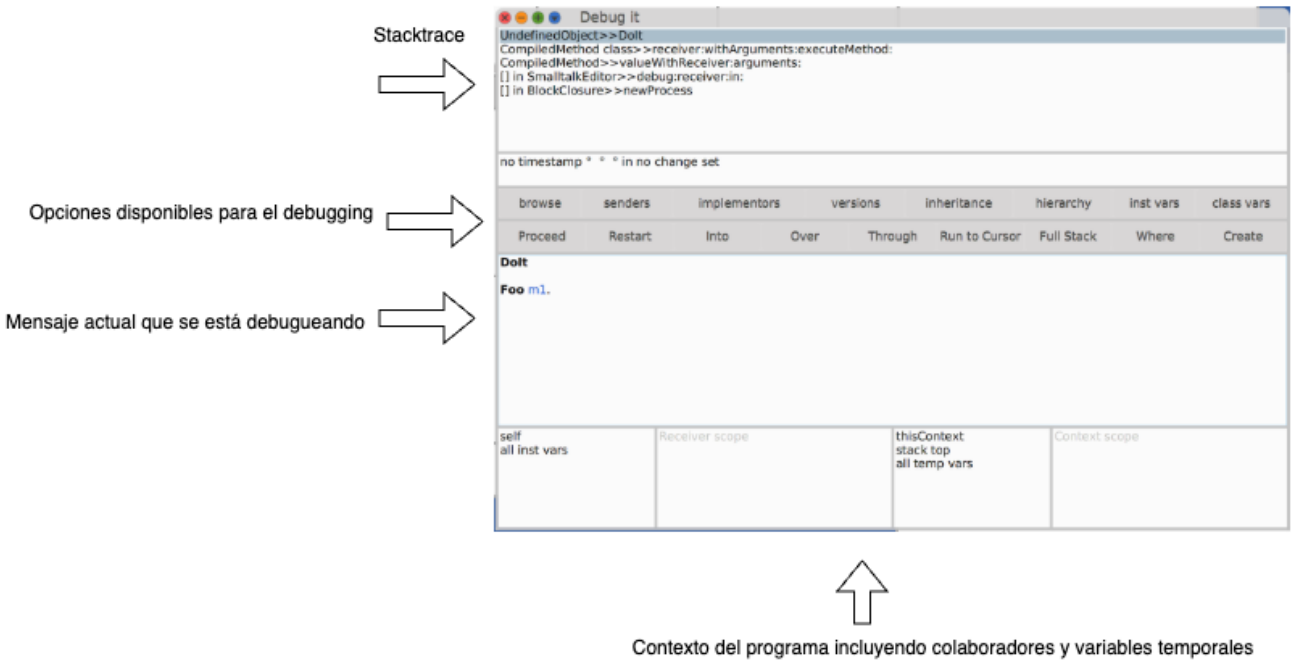
Parte 1 - Guía de ejercicios I

0. 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`:

- **¿Cuál es la diferencia entre las acciones Into, Over y Through (en el menú del debugger)?**
    - `Into` permite entrar en la implementación de un mensaje y continuar el debugging a partir de ese punto.
    - `Over` evalúa por completo el conjunto de colaboraciones dentro de ese mensaje y pasa al siguiente.
    - `Through` es similar a `Into`, pero se usa en bloques en lugar de en mensajes. Si no hay un bloque, su comportamiento es como `Over`.
- **¿Qué sucede al hacer clic en Restart?**
    - Se resetea el contexto actual del programa. En el caso de estar en un bloque, se reinicia el bloque. Si se está en la implementación de un mensaje en una clase, se reinicia al principio de la implementación.
    - Es importante destacar que cuando se aplica un `Restart`, el estado de los colaboradores internos se mantiene igual y no son reiniciados a su valor anterior.
- **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`?**

Queda ubicado en `b := 24` y el valor de `aVar` es `43`.

1. Colecciones

1.1 Tipos de Colecciones

a. Array (fixed-length collection)

```
x := #(5 4 3 2).
```

- **Cambiar el elemento en la primera posición con el valor 42.**

```
x at: 1 put: 42.
```

- **¿Qué pasa si queremos agregar un elemento en la posición 5?** Se generará un error, ya que el array tiene un tamaño fijo de 4 elementos.

b. OrderedCollections

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

- **Agregar elemento 42.**

```
x add: 42.
```

- **Agregar elemento 2.**

```
x add: 2.
```

- **¿Cuántos elementos tiene la colección?**

```
x size.
```

- **¿Cuántas veces aparece el 2?**

```
x occurrencesOf: 2.
```

c. Sets

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

- **Agregar elemento 42.**

```
x add: 42.
```

- **Agregar elemento 2.**

```
x add: 2.
```

- **¿Cuántos elementos tiene la colección?**

```
x size.
```

- **¿Cuántas veces aparece el 2?**

1, ya que es un `Set` y no permite duplicados.

d. Dictionary

```
x := Dictionary new.
```

- **Agregar la key `#e` con el valor `42`.**

```
x at: #e put: 42.
```

- **¿Cuántos elementos tiene la colección?**

```
x size.
```

- **Listar las keys.**

```
x keys.
```

- **Listar los valores.**

```
x values.
```

- **Obtener el valor de la key #a.**

```
x at: #a ifAbsent: [nil].
```

- **Obtener el valor de la key #z, devolviendo 24 si no existe.**

```
x at: #z ifAbsent: [24].
```

1.2 Conversión de colecciones

e. Convertir el Array del punto a en una OrderedCollection y en un Set.

```
x asOrderedCollection.
x asSet
```

f. Convertir el Set del punto c en Array

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

g. ¿Qué retorna convertir el Dictionary en Array?

Retorna un Array con los values.

1.3, 1.4, 1.5, 1.6, 1.7 consisten en hacer pruebas

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

```
| elements odds oddIndex |

elements:= #(2 3 4 5 6).
odds := OrderedCollection new.
oddIndex := true.

elements do: [:element |
    oddIndex ifTrue: [odds add: element].
    oddIndex := oddIndex not].
odds
```

Te permite desligarte de conocer el indice y el tamaño de la OrderedCollection y tan solo te preocupas por la paridad del mismo.

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?

```
|elements odds |

elements:= #(2 3 4 5 6).
oddIndex := false.

odds := elements select: [:element |
    oddIndex := oddIndex not.
    ].
odds.
```

La ventaja es que agrega de forma automatica los elementos al Array. En esta implementación tan solo se especifica que condicion utilizar para agregar

1.10 y 1.11 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)

```
| elements index double |
elements:= #(1 2 5 6 9).
double := OrderedCollection new.
 index := 1.

[index <= elements size] whileTrue: [ double add: (elements at: index)* 2.
index := index +1].

double
```

```
|elements double|

elements:= #(2 3 4 5 6).
double := OrderedCollection new.

elements do: [:element | double add: (2* element)].
double
```

Se acumula en otro objeto auxiliar.

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

```
|elements|
elements:= #(2 3 4 5 6).
elements collect: [:element | 2* element].
```

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

Se considera que ya pueden resolver la implementacion del do y while de forma autónoma.

```
|elements|
elements:= #(1 5 3 6 9).
elements findFirst: [:element | element even].
```

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

Se devuelve 0.

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)

```
|elements|
elements:= #(1 5 3 7 9).
firstIndex := elements findFirst: [:element | element even].
firstIndex = 0 ifTrue: [self error: 'No hay pares']
```

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.

Se considera que ya pueden resolver la implementacion del do y while de forma autónoma.

```
|elements|
elements:= #(1 2 3).
elements sum
```

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)

Recibe 2 colaboradores, el valor inicial y un bloque para saber como construir el siguiente elemento en base a el valor actual y el siguiente elemento en la colección.

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

```
string := 'No hay pares'.
string select: [:character | character  isVowel ].
```

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

Que tienen muchos mensajes en común con la misma semántica.

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

Yo no los conocía!

2. Bloques (Closures)

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

Nos devuelve el valor definido.

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

Se puede acceder a la misma y modificarla.

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

```
sumar := [:parameter1 :parameter2 | parameter1  + parameter2].
sumar value: 2 value: 4.
```

3. Símbolos

4. Medidas

4.2 Evalúe estas colaboraciones. ¿Qué resultado esperaba? ¿Cuál obtuvo?

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

Se esperaba obtener `10 pesos + 10 dólares`, pero se genera un error `MessageNotUnderstood` en `BaseUnit` con el mensaje `multiplyCompoundMeasure`.

Si se agregan paréntesis:

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

El resultado es el esperado.

4.4 ¿Qué es `peso`? Inspecciónelo:

```
peso inspect.
```

Es una `BaseUnit`.

4.5 ¿Qué es `10 * peso`? Evalúe:

```
(10 * peso) inspect.
```

Es una `SimpleMeasure`, ya no es una unidad.

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

El `amount` de un número es ese mismo número y su unidad es `NullUnit`.

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

Ambos devuelven una `CompoundMeasure`, ya que mezclan unidades distintas.

4.8 Cree `peso` nuevamente. ¿Qué representa `$$`?

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

`$$` representa el símbolo usado para la unidad `peso`.

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

```
| meter centimeter |
meter := BaseUnit named: 'meter'.
centimeter := ProportionalDerivedUnit baseUnit: meter conversionFactor:
1/100 named: 'centimeter'.
(10 * meter) + (500 * centimeter).
```

El resultado es `15 meters`.

4.10 Resuelva el problema del cohete Ariane

```
| metro pulgada|
metros := BaseUnit nameForOne: 'metro' nameForMany: 'metros'.
pulgadas := ProportionalDerivedUnit baseUnit: metros conversionFactor:
1/39 named: 'pulgada'.
```

```
sesentaPulgadas := 60 * pulgadas.
diezMetros := 10 * metros.

diezMetros + sesentaPulgadas
```

4.11 Construir los Celsius y Fahrenheit en función de los Kelvin

```
kelvin := BaseUnit nameForOne: 'kelvin' nameForMany: 'kelvin'.

celcius := ( NotProportionalDerivedUnit
            baseUnit: kelvin
            conversionBlock:  [:kelvins | kelvins + (5463/20)]
            reciprocalConversionBlock: [ :celsius | celsius - (5463/20) ]
            named: 'celcius').

fahrenheit := ( NotProportionalDerivedUnit
            baseUnit: kelvin
            conversionBlock:  [:kelvins | (kelvins - 32) * (5/9) + 273]
            reciprocalConversionBlock: [ :farenheit | farenheit -
(5463/20) ]
            named: 'fahrenheit').

(30 * kelvin) + (20 * celcius) + (10 * fahrenheit).
```

5. Fechas

5.2 Obtener el dia dentro de una semana:

```
FixedGregorianDate today next: 7 * day.
```

5.3 Sumar 24 segundos a la fecha de hoy:

```
FixedGregorianDate today next: 24 * second.
```

Genera un error, ya que `FixedGregorianDate` la unidad minima que se puede alterar es el día.

5.4 Sumar un día en segundos:

```
FixedGregorianDate today next: 86400 * second.
```

El resultado es la fecha de mañana.

5.5 Evaluar por separado estas colaboraciones: ¿Qué ocurre?

```
2024 isLeap.
```

Genera un error, ya que `Integer` no entiende el mensaje `isLeap`. Se debe evaluar en un `Date`:

```
(FixedGregorianDate year: 2024) isLeap.
```

5.6 Corrija la siguiente expresión para que funcione:

```
TimeOfDay now next: 3600 * second
```