Ejercicio 1

La clase Dictionary se utiliza para representar un conjunto de asociaciones clave-valor. Cree un diccionario como se detalla a continuación y evalúe las expresiones que se listan debajo. Inspeccione el resultado y al objeto receptor luego de recibir cada mensaje y explique qué hace:

```
| dict |
dict := Dictionary new.
dict at: 'Uno' put: 1.
dict at: 'Dos' put: 2.0.
dict at: 'TresYMedio' put: 1/2 + 3.
```

- dict keys.
- dict values.
- dict at: 'Uno'.
- dict includesKey: 'Uno'.
- dict includesKey: 'Cuatro'.
- dict at: 'Cuatro'.
- dict at: 'Cuatro' ifAbsent: [25].
- dict at: 'Uno' put: 83.
- dict keyAtValue: 2.0.
- dict select: [:element | element >= 3].
- dict do: [:element | Transcript show: element; cr].
- dict keysAndValuesDo: [:key :value | Transcript show: 'Hay un ' , value printString , ' en ' , key; cr].

Ejercicio 2

Para cada uno de los siguientes mensajes de colecciones indique qué hace y escriba un mensaje de test que ejercite dicho comportamiento sobre la clase OrderedCollection:

#add: newObject	#addAll: aCollection	#addFirst: newObject	#addLast: newObject
#select: selectBlock	#reject: rejectBlock	#allSatisfy: aBlock	#anySatisfy: aBlock
#count: aBlock	#detect: aBlock	#detect: aBlock ifNone: exceptionBlock	
#groupedBy: aBlock	#intersection: aCollection	#union: aCollection	#removeAll: aCollection
#isEmpty	#includes: anObject	#removeAllSuchThat: aBlock	
#includesAll: aCollection	#includesAny: aCollection	#inject: thisValue into: binaryBlock	

Ejercicio 3

Modifique la implementación del mensaje Collection>>sum para que retorne 0 en el caso de la colección vacía. Escriba los test de unidad correspondientes para verificar el correcto funcionamiento de su implementación.

Ejercicio 4

Evalúe cada una de las expresiones que se indican a continuación, documente su resultado e indique qué realizan.

- [1 + 2].
- [1 + 2] value.
- [:number | number + 8].
- [:number | number + 8] value.
- [:number | number + 8] value: 5.

- | squared |
  squared := [:number | number ** 2].
  squared value: 5.
  squared value: 7.
- | diff base |
  base := 5.
  diff := [:number | base - number].
  diff value: 8.
  diff value: 1.
  base := 77.
  diff value: 8.
  diff value: 1.

Ejercicio 5

Utilizando el browser de clases investigue la implementación de los mensajes que se detallan a continuación y explique cómo funcionan las expresiones detalladas en la columna izquierda.

'('Helio' includes: $e) ifTrue: [Transcript show: 'Sipi'] ifFalse: [Transcript show: 'Nopo'].	Boolean>>ifTrue:ifFalse: True>>ifTrue:ifFalse: False>>ifTrue:ifFalse:
5 timesRepeat: [Transcript show: 'Hola Transcript'; cr]	Integer>>timesRepeat:
| i | i := 1. [i <= 10] whileTrue: [Transcript show: 'Hola - ' , i printString; cr. i := i + 1.]	BlockClosure>>whileTrue:
1 to: 10 do: [:i | Transcript show: 'Oh - ' , i printString ; cr]	Number>>to:do:

Ejercicio 6 (Entregar - Fecha tope: 20/10/2015)

Las pilas son colecciones de objetos que manejan sus elementos utilizando el método LIFO (last-in, first-out), al tiempo que las pilas trabajan con el método FIFO (first-in, first-out). Usted debe implementar las clases necesarias para representar pilas y colas (junto con sus tests de unidad). Ambas clases deben comprender el siguiente protocolo:

- #push: anObject. Agrega un objeto.
- #pop. Retorna un elemento y lo elimina de la colección.
- #isEmpty. Indica si está vacia.
- #do: aBlock. Evalúa un bloque con cada elemento.
- #select: aBlock. Retorna una nueva pila (o cola dependiendo del objeto receptor) con aquellos los elementos que cumplen una determinada condición.

Ejercicio 7

Implemente una clase (y sus tests de unidad correspondientes) para representar una matriz bidimensional de objetos. El tamaño de dicha matriz (cantidad de filas y columnas) debe especificarse en la creación de la misma y se mantiene fijo a partir de ese momento. Los mensajes que debe implementar son:

- #columns “Retorna la cantidad de columnas”.
- #rows “Retorna la cantidad de filas”.
- #at: row at: column put: anObject “Coloca un objeto en la fila y columna indicada”.
- #at: row at: column “Retorna el objeto de la fila y columna indicada”.
- #= otherMatrix “Retorna true si ambas matrices son iguales, false en caso contrario”.
- #transpose “Retorna una nueva matriz que es el receptor transpuesto”.
- #do: aBlock “Evalúa un bloque por cada elemento de la matriz”.
- #collect: aBlock “Retorna una nueva matriz del mismo tamaño cuyos elementos son el resultado de aplicar el bloque al elemento de la matriz receptora en cada posición”.
- #withIndicesDo: aBlock “Evalúa un bloque de tres argumentos por cada elemento de la matriz. Dicho bloque tiene la forma [:row :column :object | ….]”.

Ejercicio 8

Dada su implementación de matrices del Ejercicio 7 encuentre la forma de obtener la matriz transpuesta sin tener que duplicar la estructura de almacenamiento o recorrer todos los elementos.