

Colecciones

Referencias a Objetos

- Variables de instancia.
- Variables globales.
- Parámetros.
- Hasta ahora siempre referenciando a una instancia

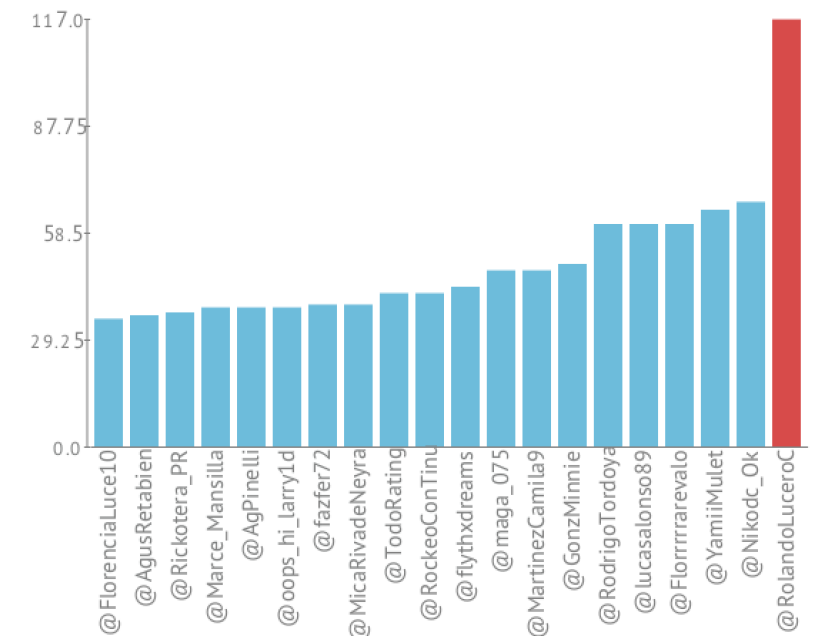
Redes Sociales

- Muchos usuarios
- Muchos posts
- Ej Twitter
 - Usuarios, Tweets, hashtags, seguidores



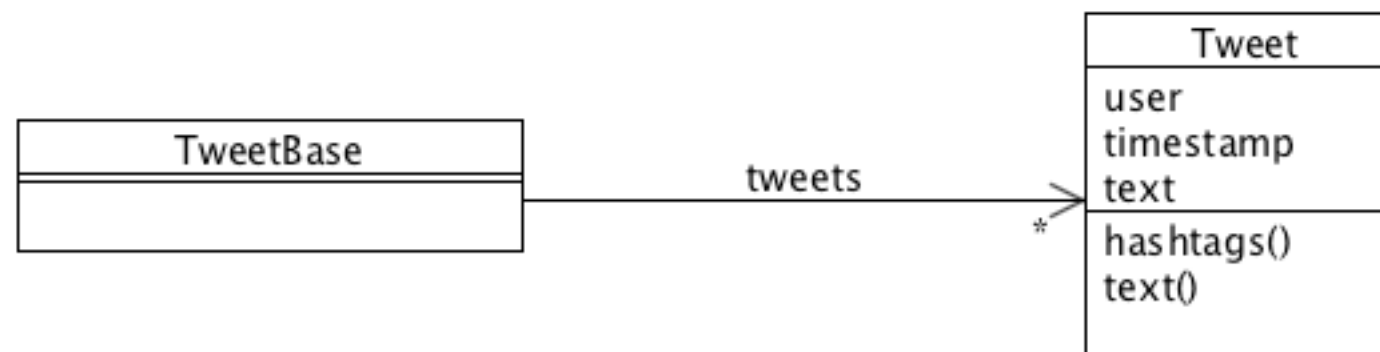
Contexto:

- Queremos hacer un programa que permita procesar datos de Twitter.
- Necesitamos operar sobre muchos tweets.
- Necesitamos almacenarlos, filtrarlos, manipularlos, recolectar datos etc.



Requerimientos

- Debemos tener un objeto que nos permita operar sobre muchos tweets
- No sabemos cuantos.
- Datos repetidos y datos únicos.
- Ordenamiento.

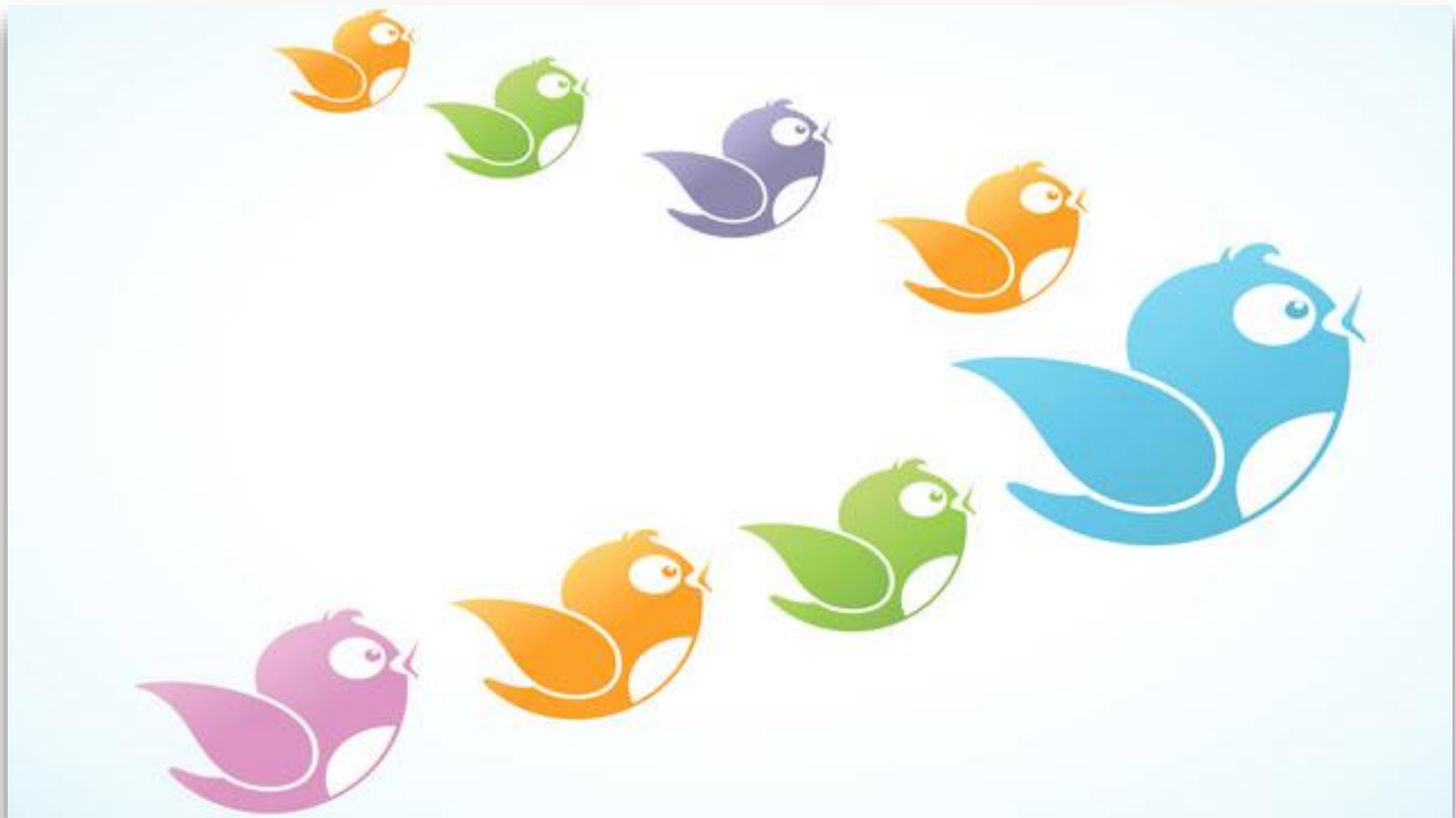


Colecciones

- Las colecciones en Smalltalk, son objetos especializados en referenciar a muchos otros objetos.
- Variantes de acuerdo a:
 - Si queremos o no duplicados
 - Si sabemos cuantos objetos son.
 - Cómo queremos recuperar los objetos referenciados.
 - etc.



Colecciones - Responsabilidades



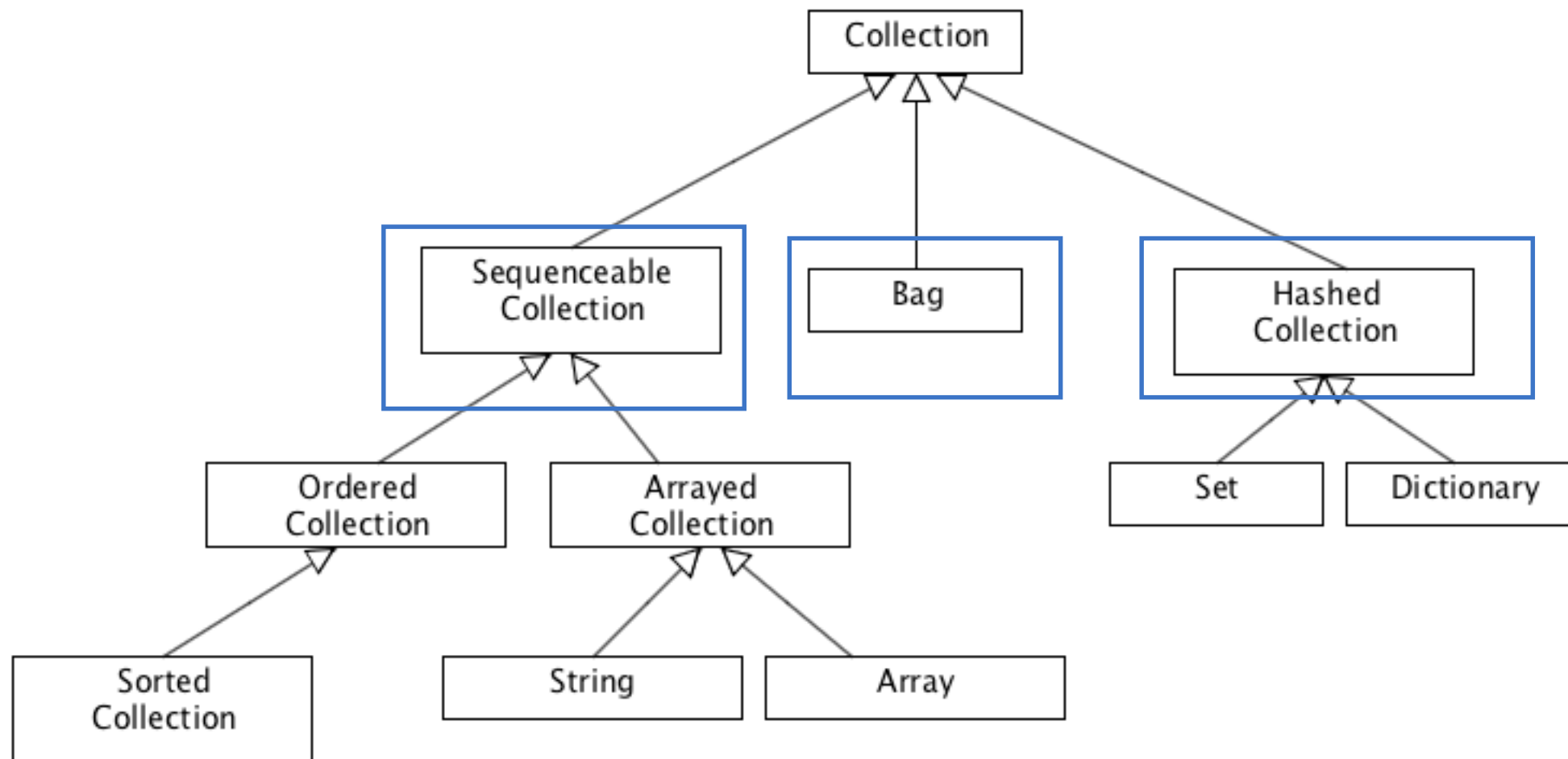
regar, remove, buscar, seleccionar, iterar sobre los elementos, e

Collection - Protocollo

- #new, #with:, #withAll:
- #add:
- #remove:
- #size
- #isEmpty
- #select: aBlock
- #collect: aBlock
- #detect: aBlock



Jerarquía(*) de Colecciones en Smalltalk



(*) Estas son sólo algunas... las que usaremos

Instanciación

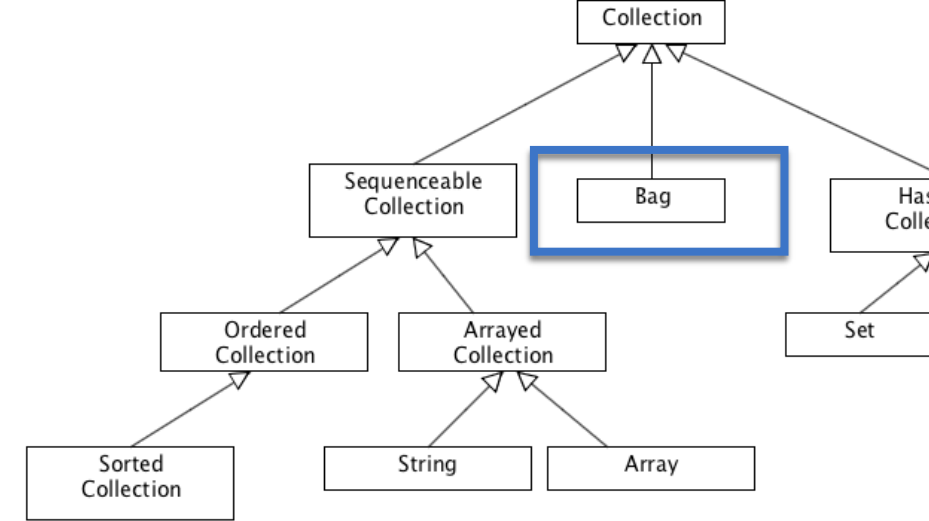
```
myColl := OrderedCollection new.
```

```
myColl := Array new: 30.
```

```
myColl := Set with:'red'  
              with:'blue'  
              with:'yellow'.
```

- ST es dinámicamente tipado, podemos poner obj. de diferentes clases en una colección.

Bag



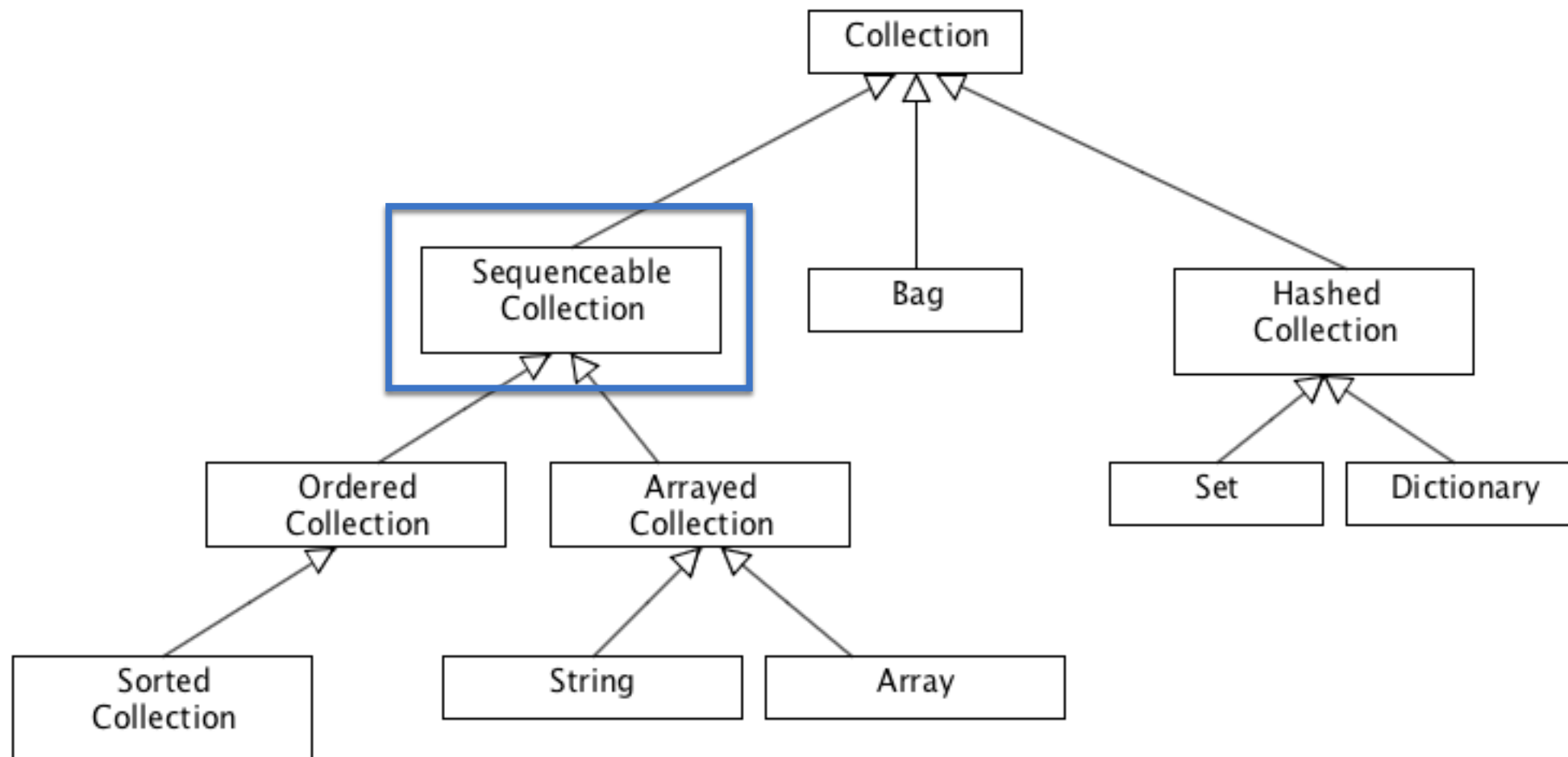
- No indexada,
- No tiene noción de orden. No entiende: `#first`, `#last`, `#at:`. etc
- Acepta repeticiones. `#add:withOccurrences`:
- Ejemplo de uso: Queremos saber cuantas veces fueron mencionados los hashtags utilizados.
- Podemos poner todos en un bag y contar las ocurrencias

ni cualquier
mensaje que
implique orden

```
myBag := Bag new.  
tweets do: [ :t |  
    myBag addAll: t hashtags.  
].  
myBag occurrencesOf: '#vamoschino'
```

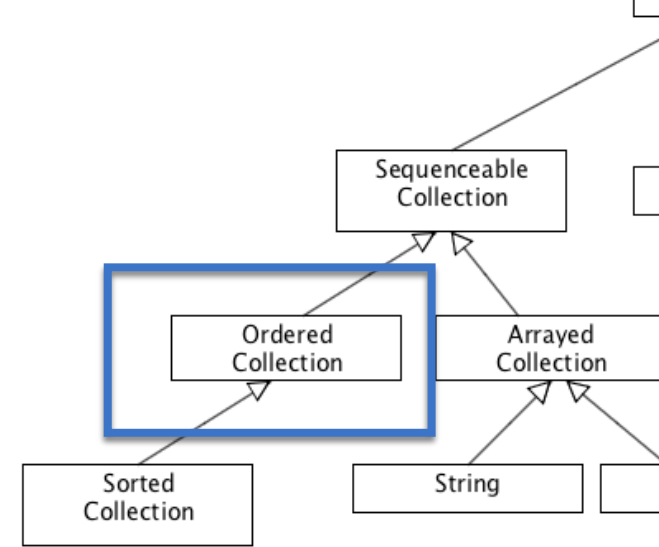


Sequenceable Collection



Orden bien definido para los elementos
#first, #last, #addFirst:, #removeFirst:, #at:

OrderedCollection



- Colección indexada (#at:).
- Orden de los elementos almacenados. Posición
- Tamaño variable (crece automáticamente)

```
(tweets at:1) = tweets first.
```

```
tweets last.
```

```
tweets addFirst:anotherTweet.
```

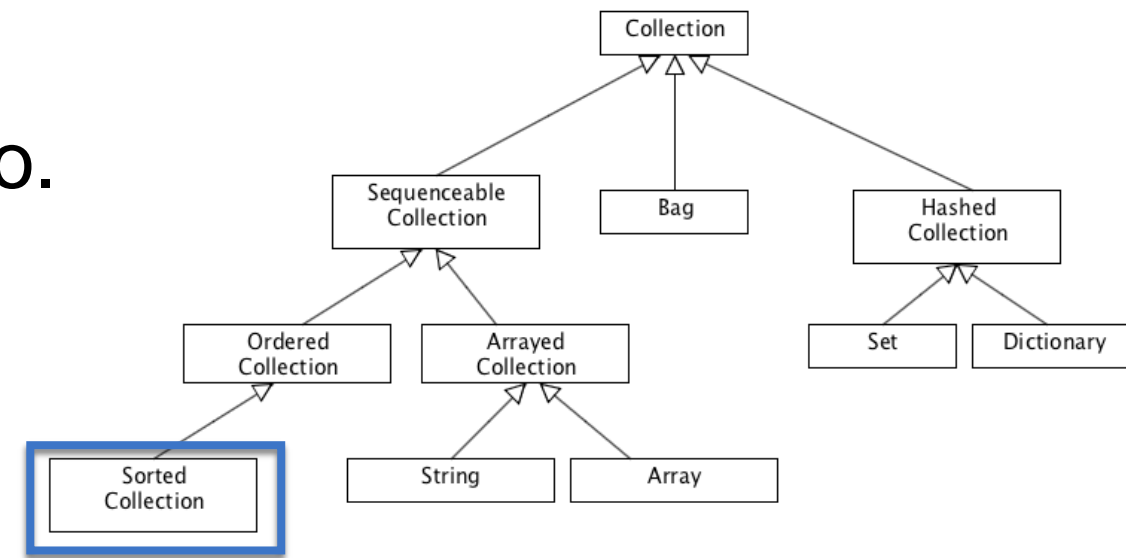
```
tweets removeLast.
```



SortedCollection

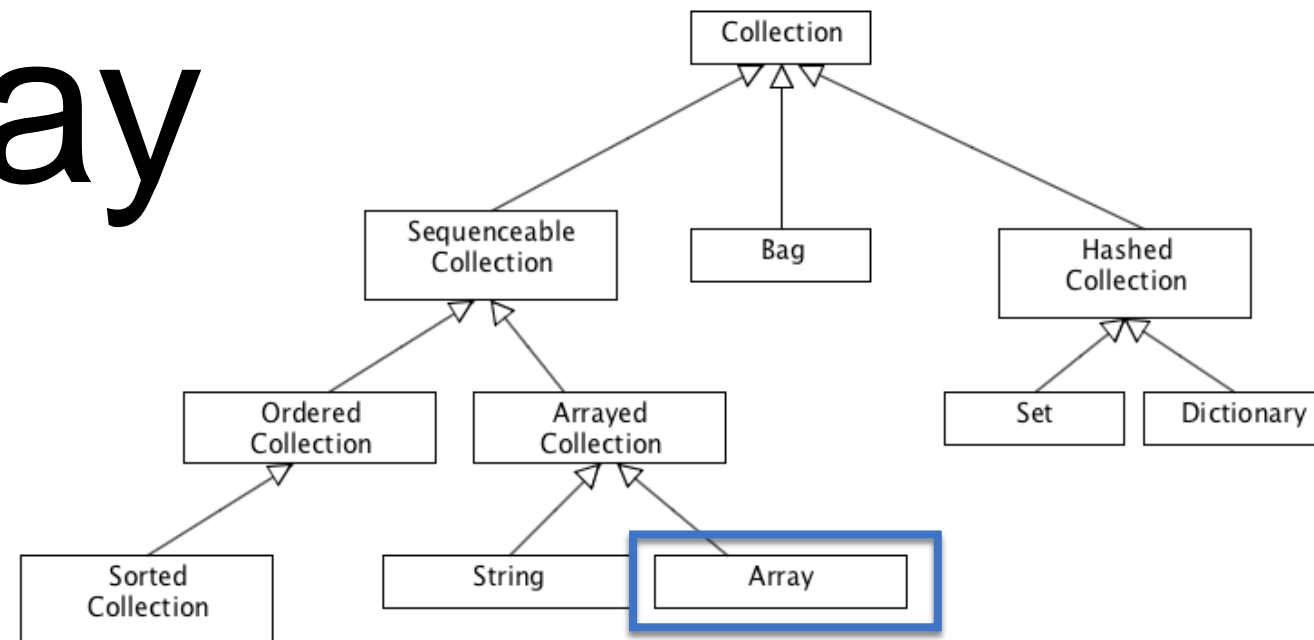
- Elementos ordenados por un criterio.
- Relación de orden
- sort block:
 - default: [:a :b | a<=b]
 - parametrizado.
- Ej. queremos los tweets ordenados por fecha:

```
tweets asSortedCollection sortBlock: [:tw1 :tw2 | tw1  
timestamp < tw2 timestamp ]
```

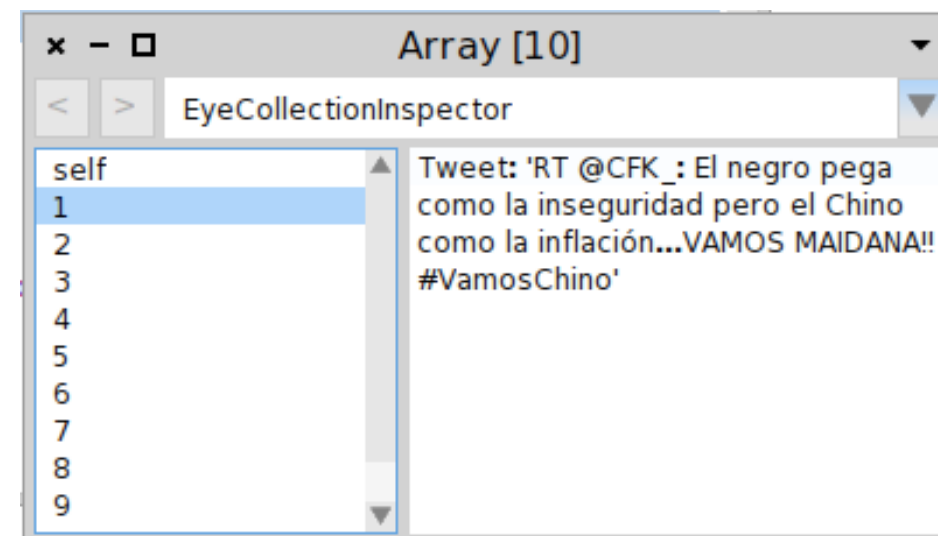


Array

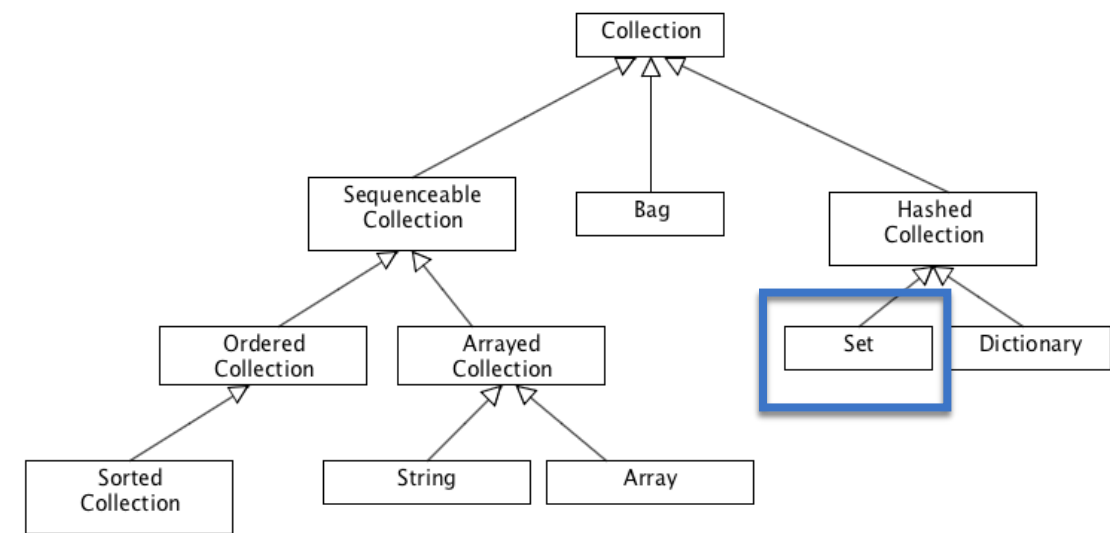
- Tamaño predefinido
- Acceso indexado por un integer.
- No es completamente polimórfica con Collection
- No entienden #add: #remove: etc.



```
myArray:=Array new:10.  
1 to: 10 do[:i |  
    myArray at:i put: tweets atRandom  
].  
myArray.
```



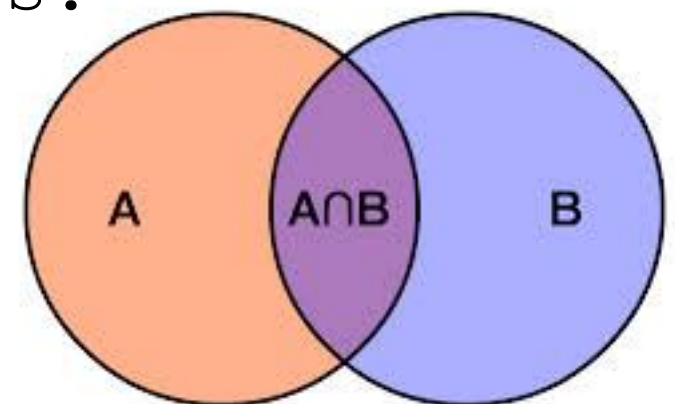
Set



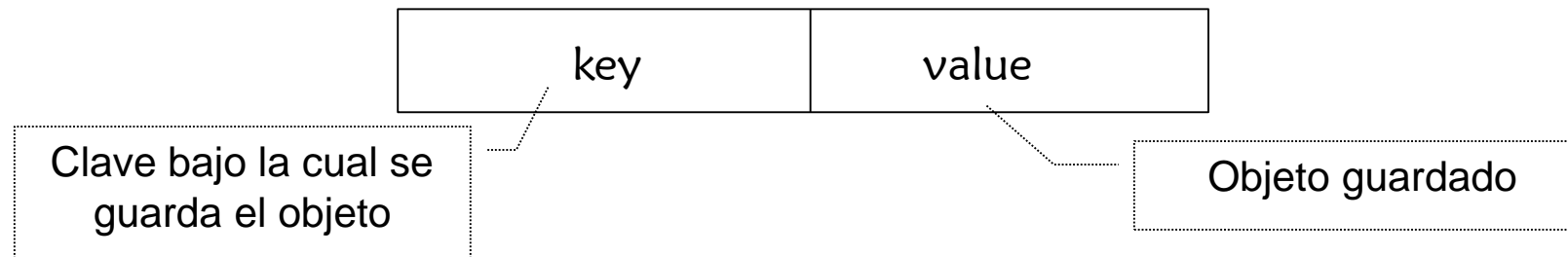
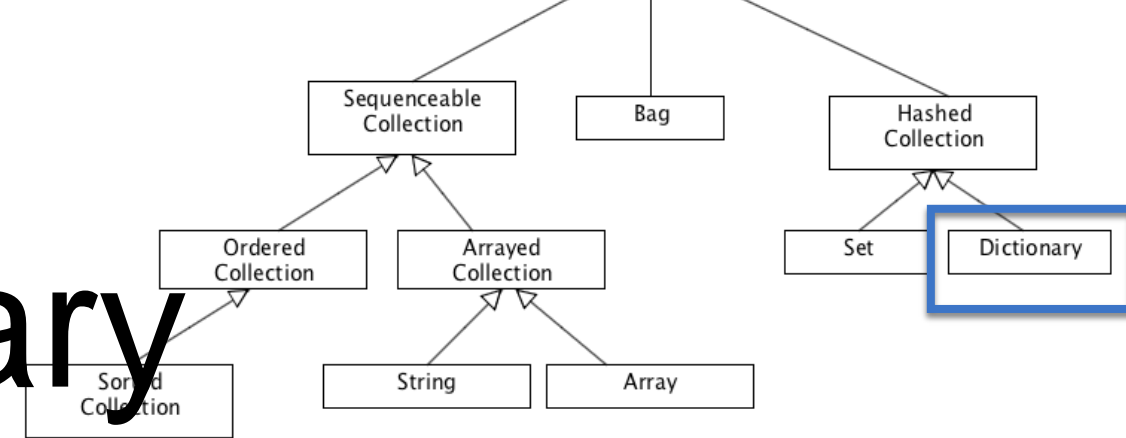
- No existe noción de orden. No entiende: `#first`, `#last`, `#at:`, etc .
- Sin duplicados.
- Depende fuertemente de la definición de `=` (y de `#hash`)
- Ej. queremos todos los hashtags que aparecen en los tweets, sin repeticiones.

ni cualquier mensaje que implique orden

```
mySet := Set new.  
tweets do: [ :t |  
    mySet addAll: t hashtags.  
].  
mySet
```



Dictionary



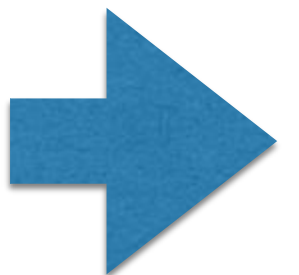
- Pares *clave->valor* (*Associations*)
- Muchos mensajes (`#add:`, `#remove:`, etc) trabajan con las *associations*
- `#at:` y `#at: put:` Variantes: `#at:ifAbsent:`

```
| result |
result := Dictionary new.
tweets
  do: [ :tweet |
    | hhmm |
    hhmm := tweet hhmm.
    result at: hhmm ifAbsentPut: 0.
    result at: hhmm put: (result at: hhmm) + 1 ].
^ result
```



Prox. Clase:Iteradores

- #do:
- #select: y #reject:
- #collect:
- #inject:into:
- #detect:



Iteradores- #do:

- Collection>>do: aBlock
- bloque - 1 param - acciones a realizar por cada uno de los elementos.
- ejecuta el bloque para cada uno de los elementos
- Ej: imprimir contenido de los tweets

```
tweets do:[:tweet | Transcript show:tweet text; cr.]
```



Iteradores- #select: /reject:

- Collection>>select: aBlock
- bloque booleano - condición a satisfacer
- retorna una colección con aquellos elementos que satisfacen la condición.
- Ej: quiero los tweets que tienen mas de 2 hashtags

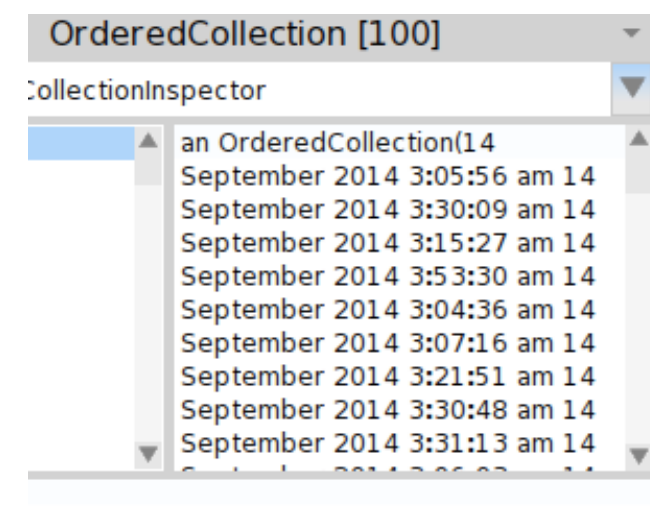


```
tweets select:[:tweet | tweet hashtags size > 2 ]
```

Iteradores- #collect:

- Collection>>collect: aBlock
- bloque
- retorna una colección con el resultado de haber evaluado aBlock para cada elemento.
- Ej: quiero el listado de los timestamps de todos los tweets

```
tweets collect:[:tweet | tweet timestamp ] .
```



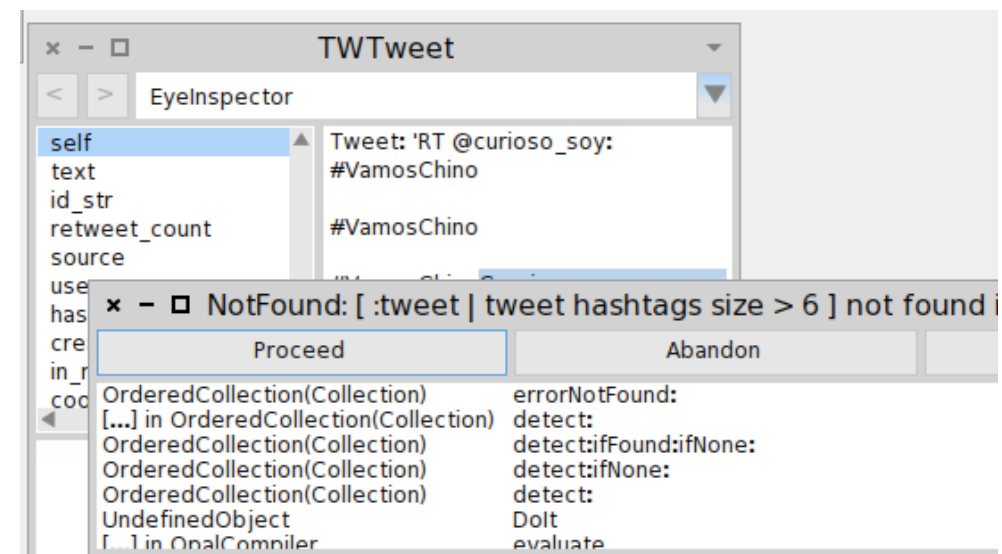
Iteradores- #detect:

- Collection>>detect: aBlock
- bloque booleano - condición a satisfacer
- retorna el primer elemento que satisface la condición, si no existe uno falla!
- Ej: quiero un tweet con 4 hashtags

tweets detect: [:tweet | tweet hashtags size > 4].



detect:
ifNone:



Resumen

- Protocolo de Collection
- Tipos de colecciones y características que las diferencian (escenarios de uso).

Referencias/Bibliografía

- Smalltalk Blue Book: Smalltalk 80 - The language and its implementation. Goldberg y Robson
- Pharo By Example. Ducasse.
- Programando con Smalltalk. Gomez Deck