Colecciones

Referencias a Objetos

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

Redes Sociales

- Muchos usuarios

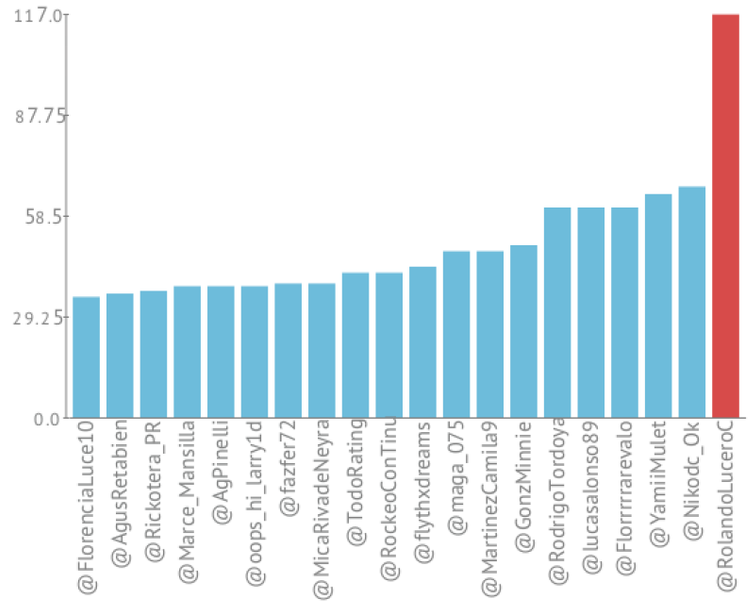
- Muchos posts
- Ej Twitter
  - Usuarios, Tweets, hashtags,seguidores

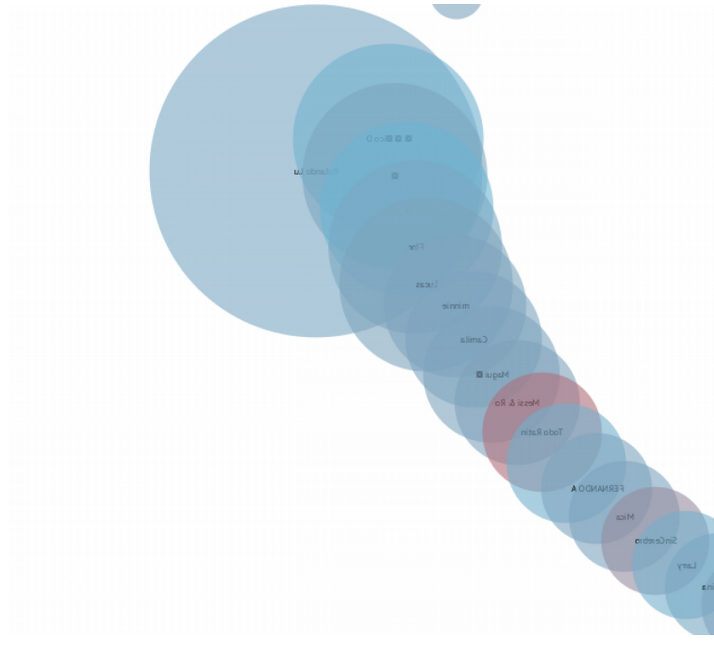
Contexto:

- Queremos hacer un programa que permita procesar datos de Twitter.

- Necesitamos operar sobre muchos tweets.

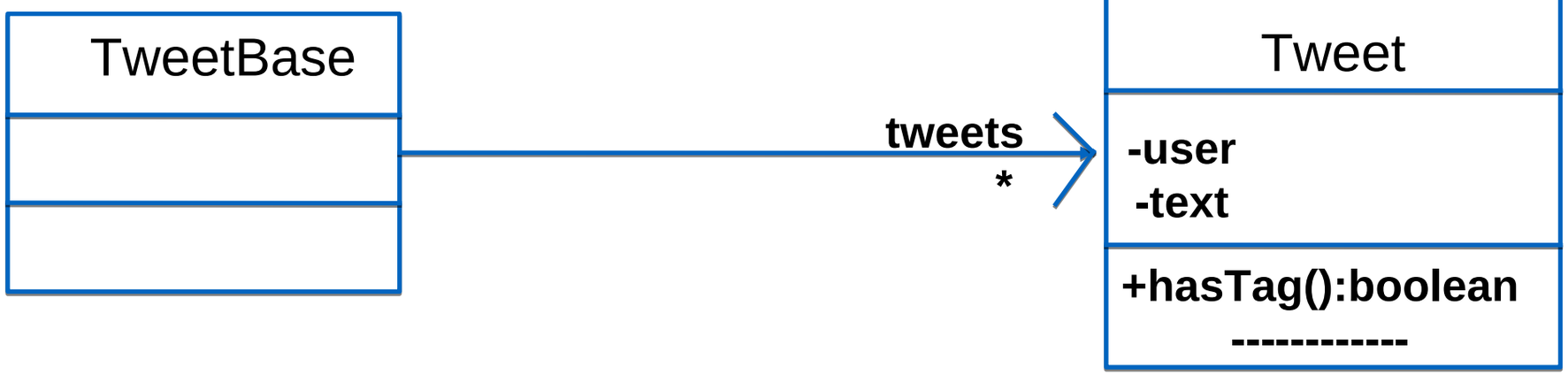
- Necesitamos almacenarlos, filtrarlos, manipularlos, recolectar datos etc.





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 compuestos que referencian 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.

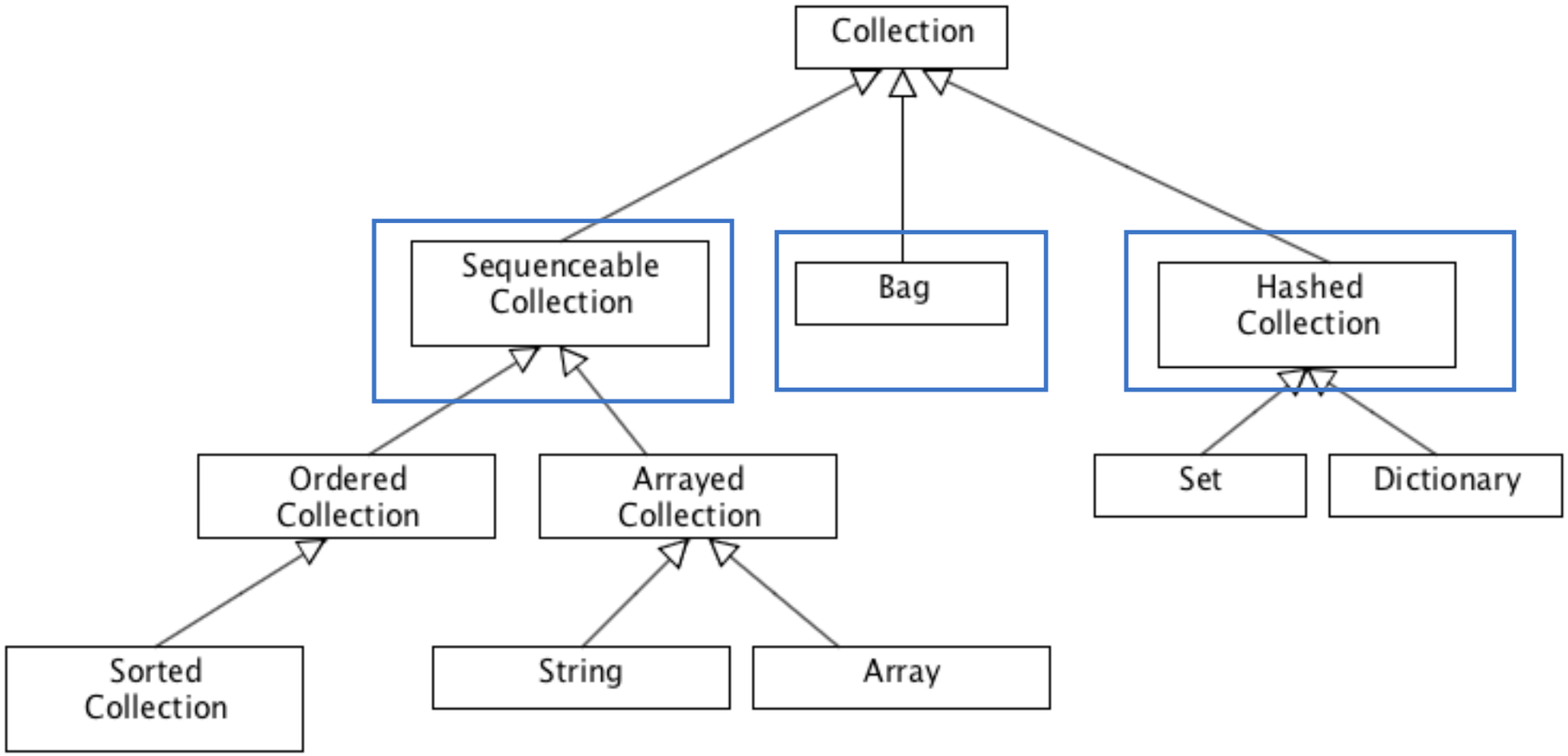
Colecciones - Responsabilidades

agregar, remover, buscar, seleccionar, iterar sobre los elementos, etc.

Collection - Protocolo

- #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

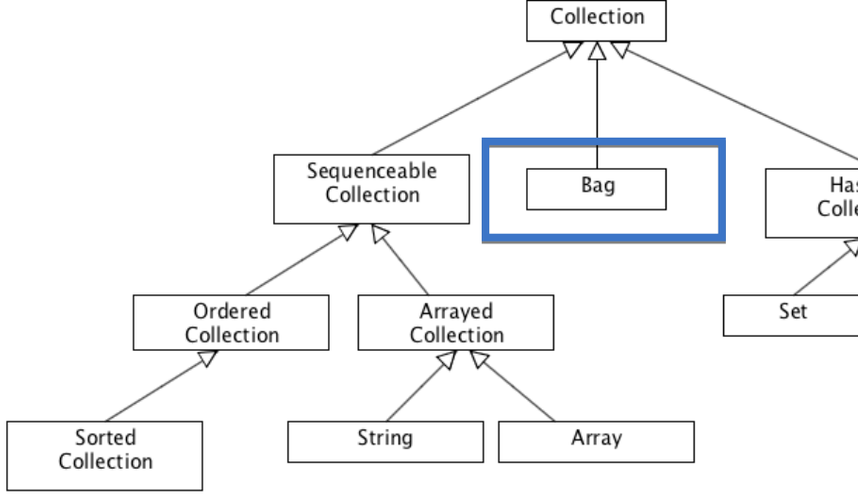
```
myColl := OrderedCollection new.

myColl := Array new: 30.

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

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

Bag

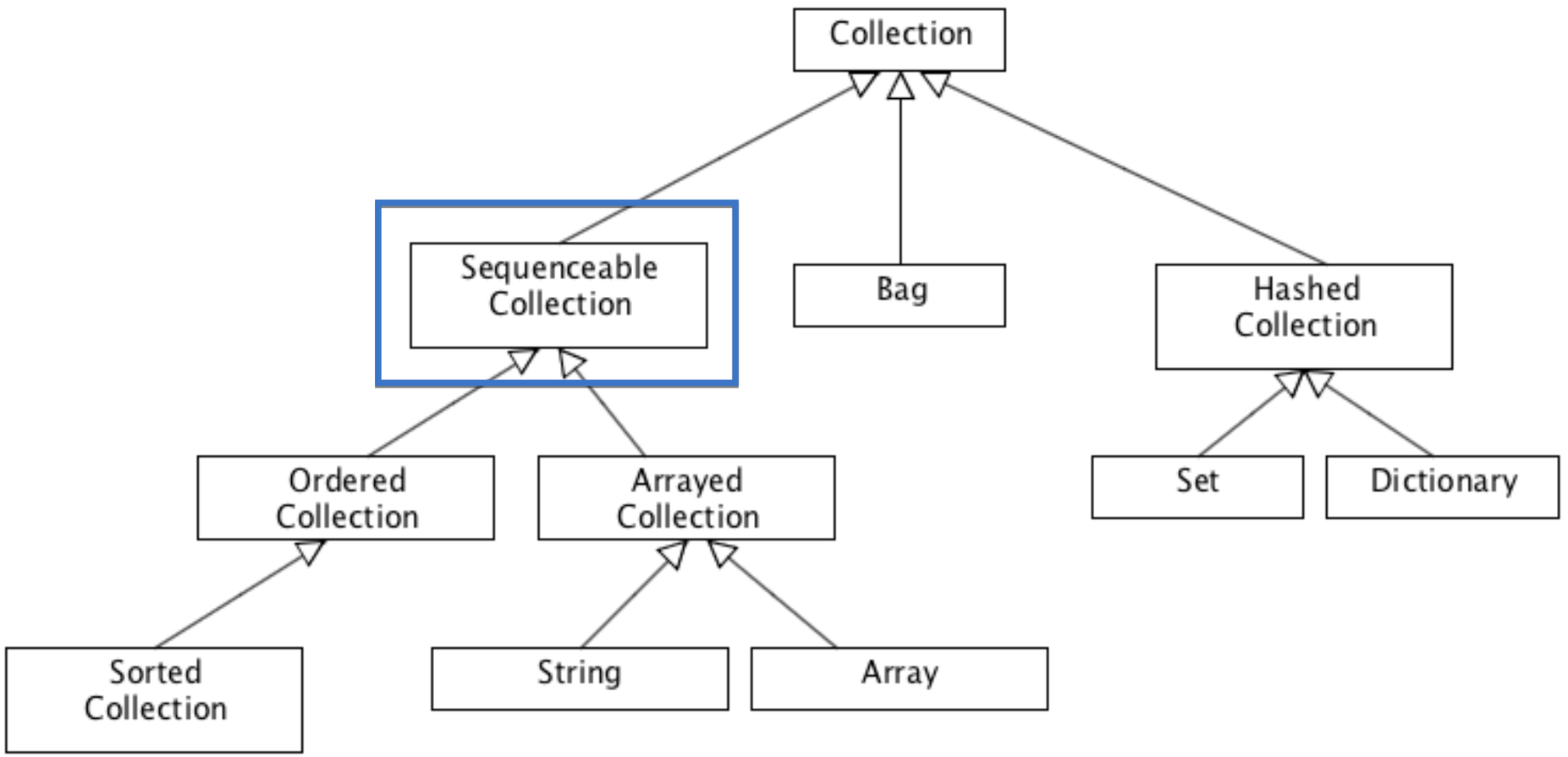


- No indexada,
- No tiene noción de orden. No entiende: #first, #last, #at:. etc
- Acepta repeticiones. #add:withOcurrences:
- Ejemplo de uso: Queremos saber cuantas veces fueron mencionados los hashtags utilizados en una colección llamada **tweets** (por ejemplo una OrderedCollection)
- Podemos poner 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'
```

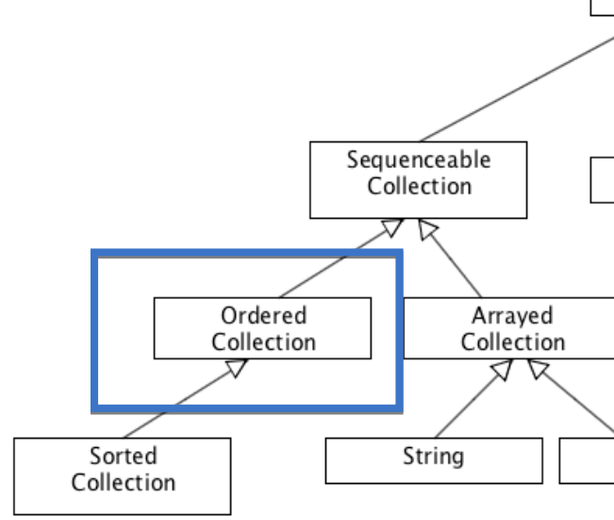
Sequenceable Collection



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)

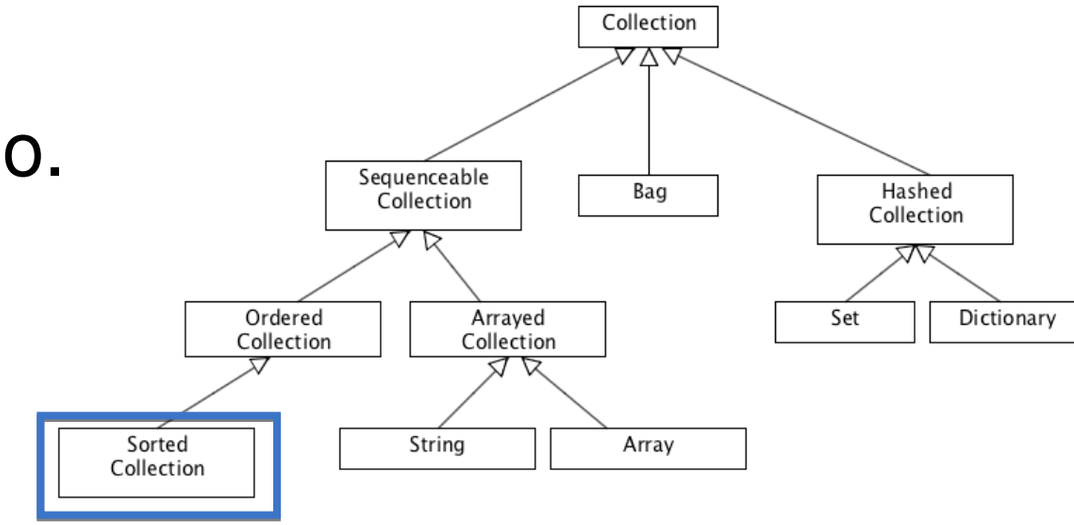
```
(tweets at:1) = tweets first.

tweets last.

tweets addFirst:anotherTweet.

tweets removeLast.
```

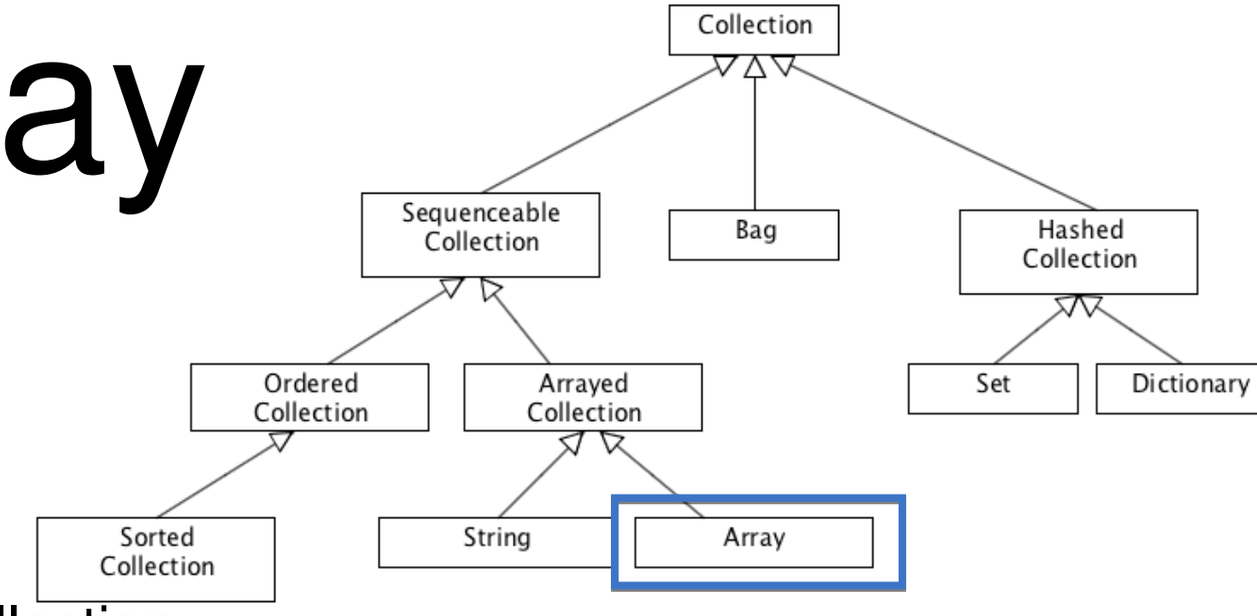
SortedCollection



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

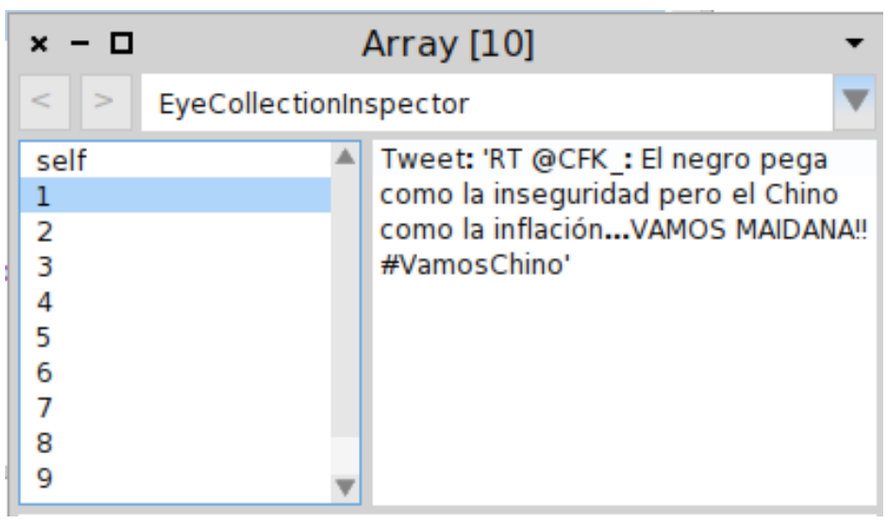
```
tweets asSortedCollection sortBlock: [:tw1 :tw2 | tw1
        timestamp < tw2 timestamp ]
```

Array

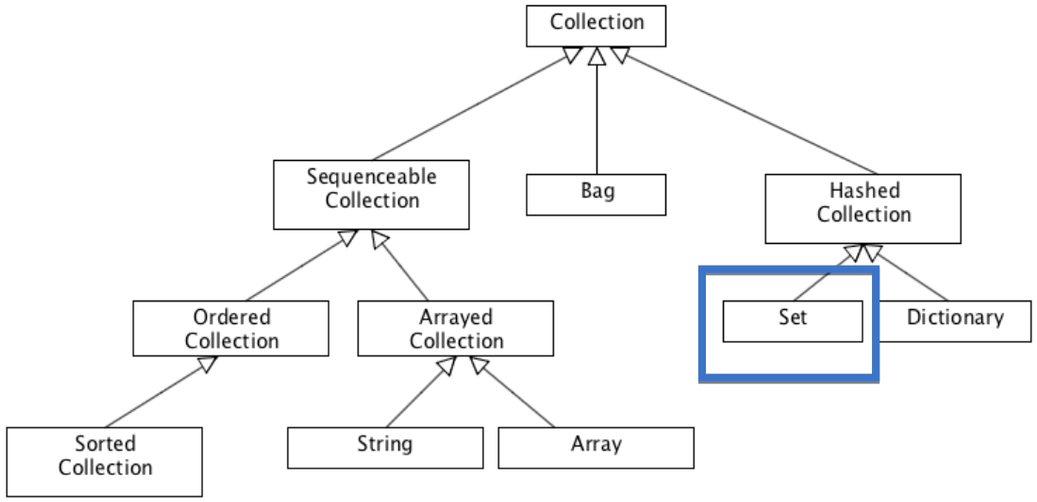


- Tamaño predefinido
- Acceso indexado por un integer.
- No es completamente polimórfica con Collection
- No entienden #add: #remove: etc.

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



Set

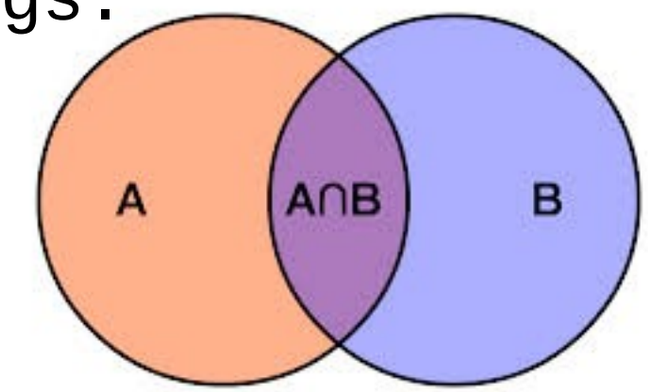


- No existe noción de orden. No entiende: #first, #last, #at:, etc .
- Sin duplicados.
- Depende fuertemente de la definición de **=** (y de #hash)

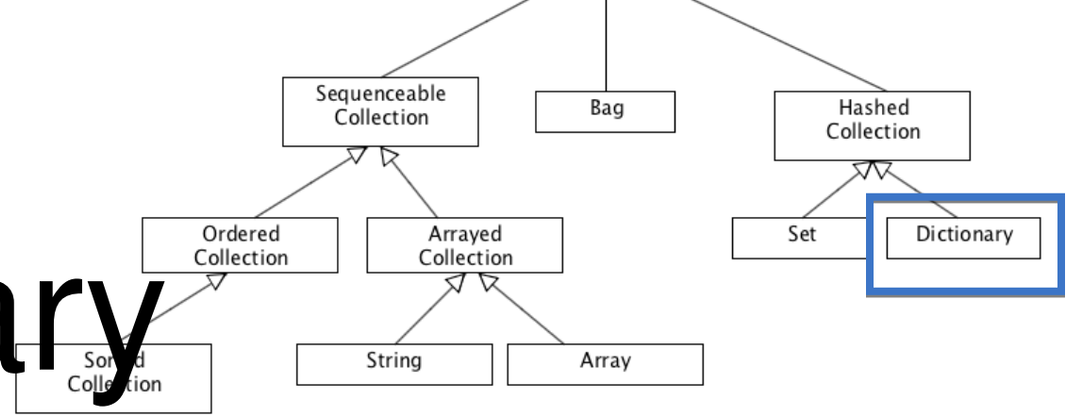
ni cualquier mensaje que implique orden

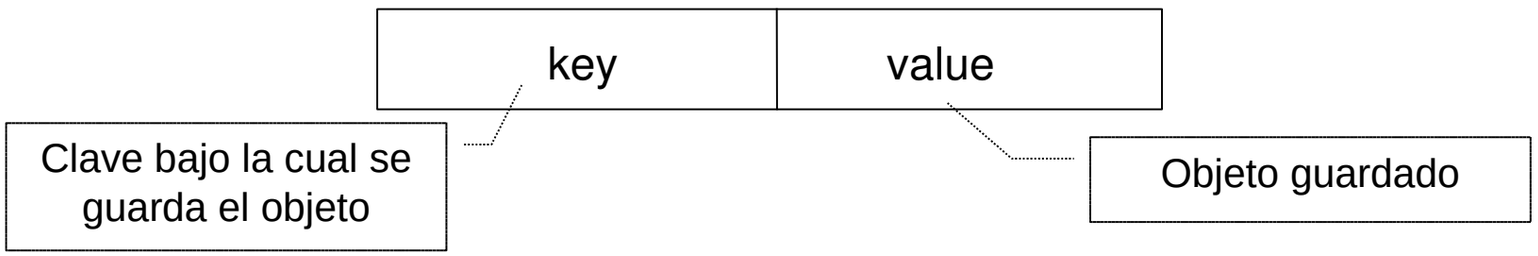
- Ej. queremos todos los hashtags que aparecen en los tweets, sin repeticiones.

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



Dictionary



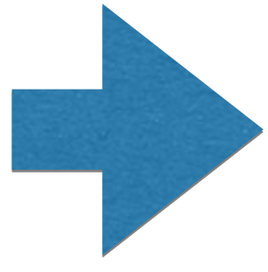


- Pares *clave->valor (Associations)*
- Muchos mensajes (#add:, #remove:, etc) trabajan con las *associations*
- #at: y #at: put: Variantes: #at:ifAbsent:

```
| 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

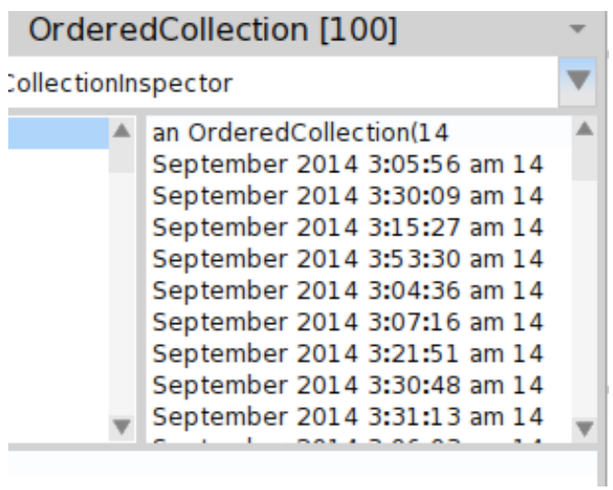
```
tweets select:[:tweet | tweet hashtags size > 2 ]
```

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- #inject:into:

- Collection>>**inject:** *aValue* **into:** *aBlock*
- Bloque con dos parametros que realiza una operación entre aValue y cada elemento de la colección.
- Retorna: el resultado del computo.
- Ej: conocer el numero total de hashtags en todos los tweets

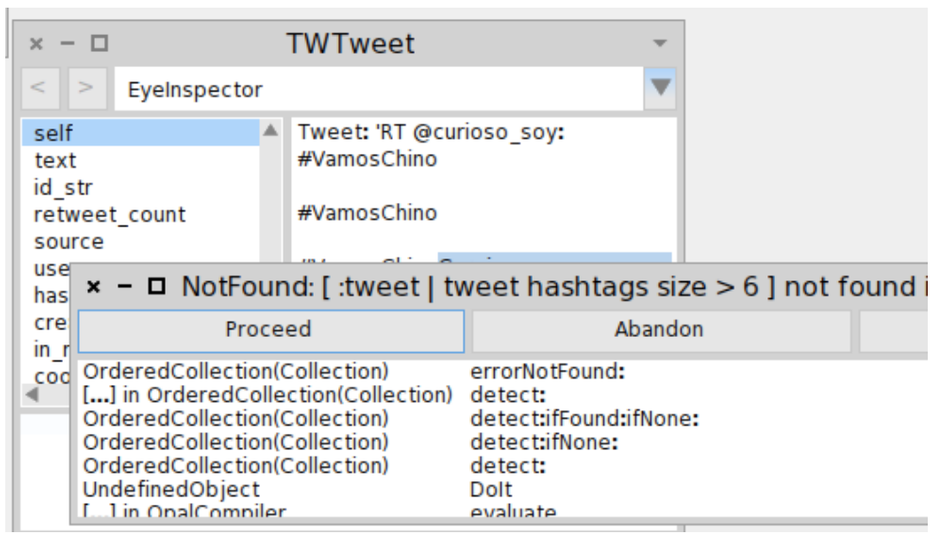
```
tweets inject:0 into:[:sum :tweet | sum + tweet hashtags size ].
```

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:



Escenarios de Uso

Quiero saber cual es el numero que mas veces salio en la quiniela en los ultimos diez años.

Necesito mantener las transacciones de una cta bancaria ordenadas cronologicamente.

Un video juego tiene un “cache” de 10 atajos de usuario.

Una applicación web desea “cruzar” los contactos de un usuario en dos redes sociales diferentes.

Un mercado electronico (bolsa) guarda para cada instrumento (acciones) todas los pedidos de compra/venta .

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

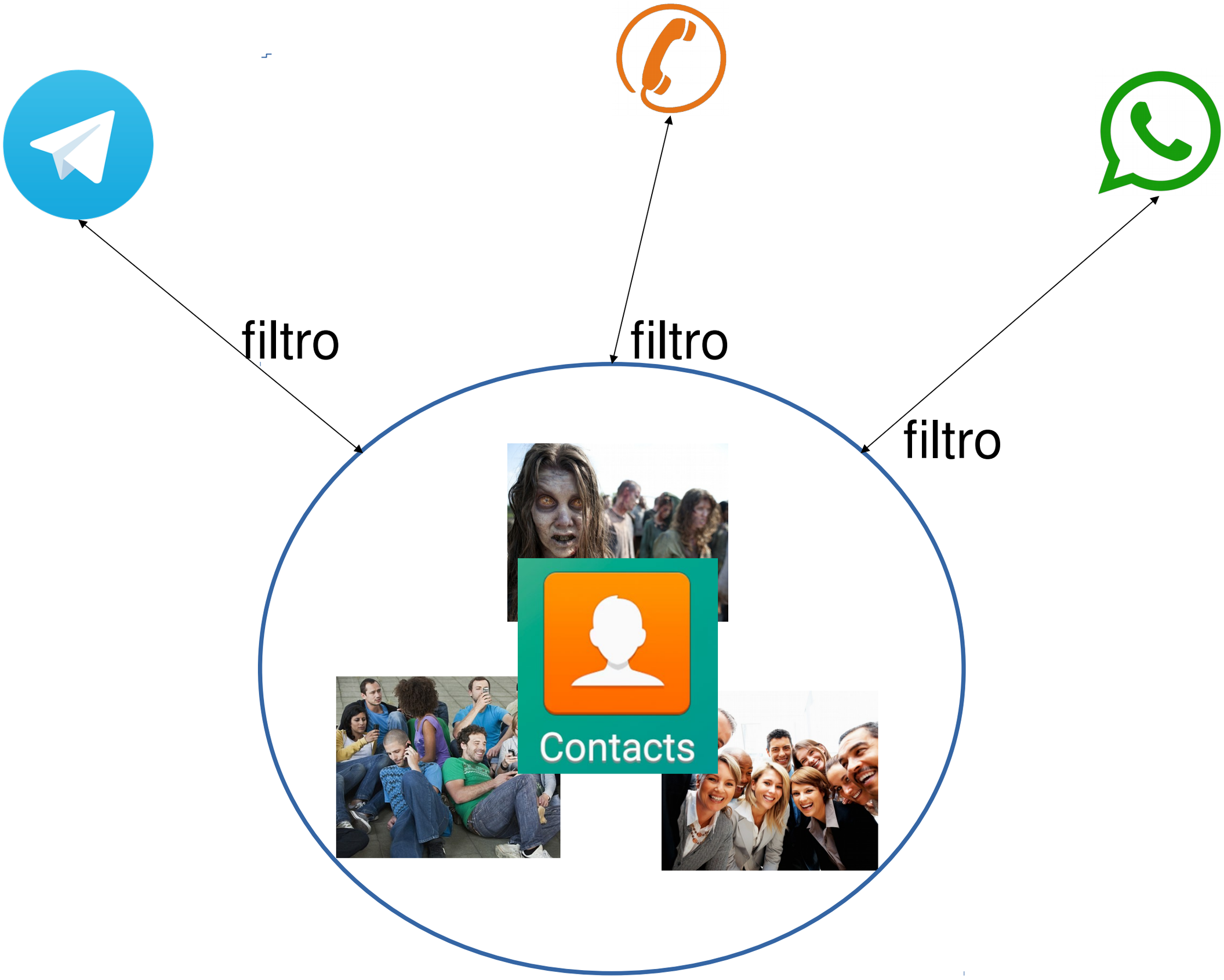
- Pharo By Example. Ducasse.

http://pharo.gforge.inria.fr/PBE1/PBE1ch10.html

- Programando con Smalltalk. Gomez Deck

Si tuviera amigos muy diferentes, los pondria en colecciones diferentes o en la misma coleccion? Por qué?

Si tuviera amigos de la facu y otro de amigos del laburo, los pondria en colecciones diferentes o en la misma coleccion? Por qué?


filtro
filtro
filtro
Contacts

Colecciones y jerarquias

Escenarios

Una clase tiene una variable donde guarda una coleccion de objetos que vienen de una jerarquia

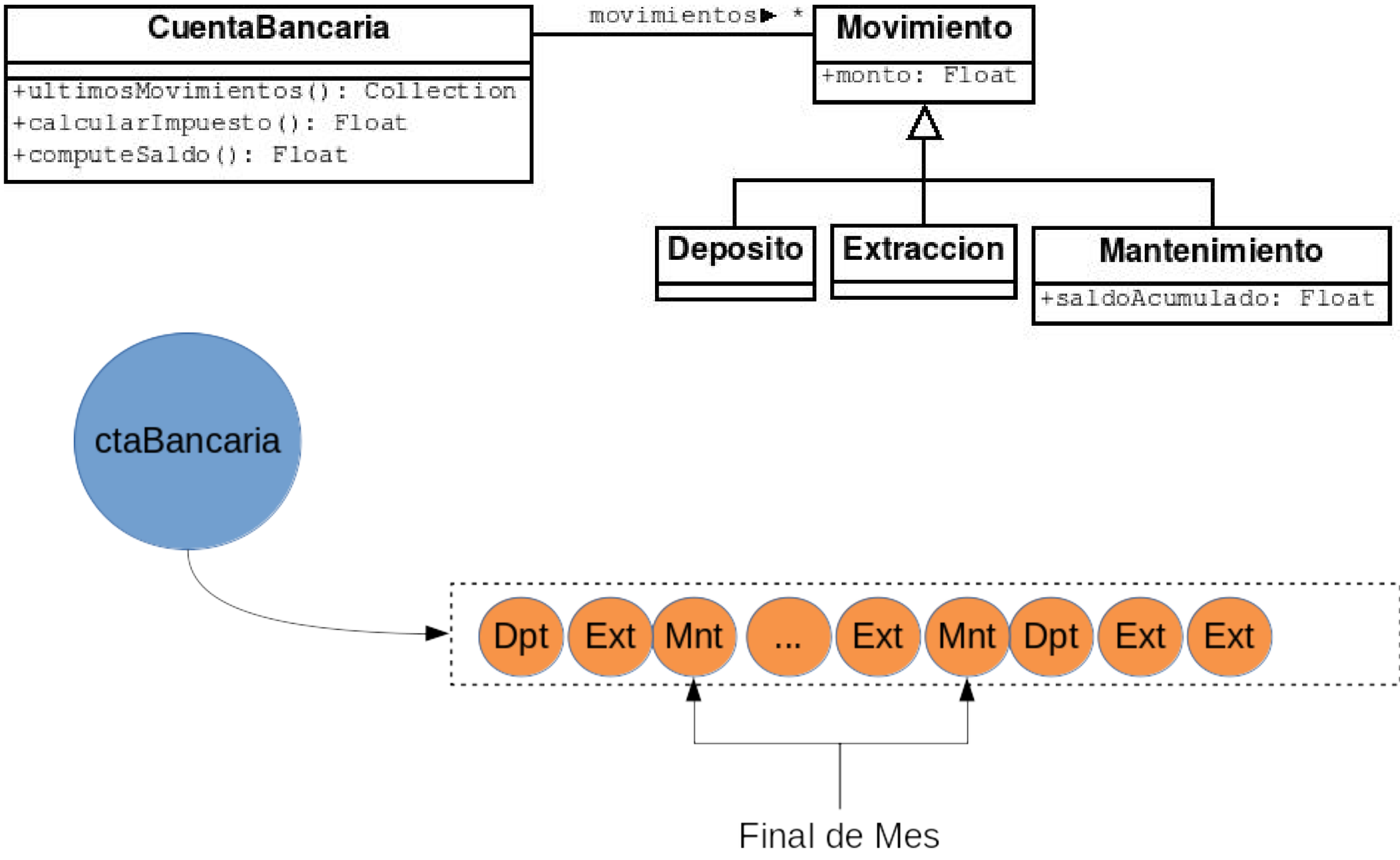
FAQ

1) Guardamos todos esos objetos en una variable o en multiples variables?

2) Si tenemos que discriminar (filtrar) solo un tipo de esos objetos preguntamos por la clase?

Colecciones y Jerarquías

- CuentaBancaria tiene Movimientos
- → Depositos, Extracciones y Mantenimiento
- Depositos y Extracciones desc. Impuestos (imputables)
- Mantenimiento (mensual) no desc Impuestos
- Extraccion y Mantenimiento decrementan saldo

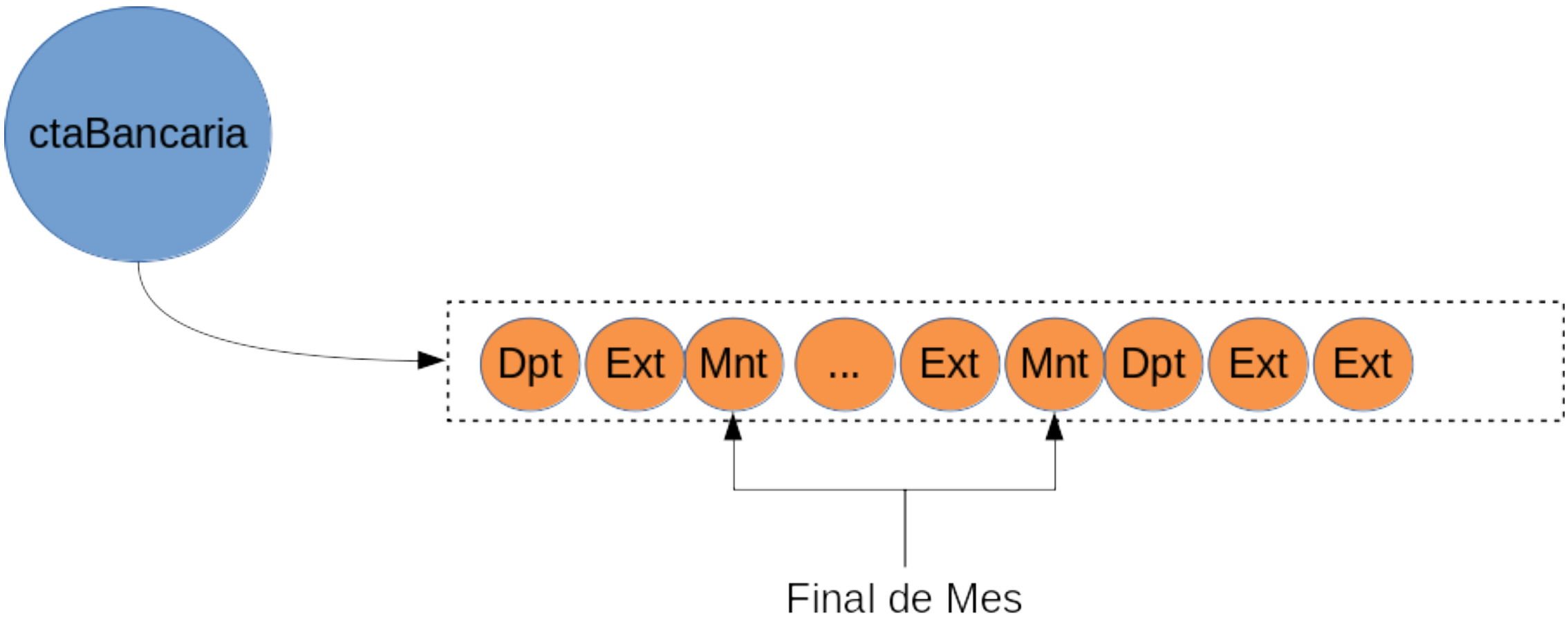


Los movimientos posteriores al último Mnt son los “ultimos movimientos”

Los “ultimos movimientos” de este mes serán viejos el mes que viene :-(

Codigo: computeSaldo

```
CuentaBancaria>>computeSaldo
 |s|
 s:= self saldoUltimoMantenimiento.
 ^self ultimosMovimientos inject: s into:[:s: e| s+ e monto ]
```



Código: calcularImpuesto

```
CuentaBancaria>>calcularImpuesto
 ^self movimientosImputables inject:0 into:[:s: e| s+(e monto *0.1)]
```

```
CuentaBancaria>>movimientosImputables
 ^self ultimosMovimientos select:[:each| each esImputable]
```

Tanto la “**ultimoMovimientos**” como “**movimientosImputables**” son colecciones que usa CuentaBancaria pero no son variables, son calculadas

#esImputable es un “filtro” que implementan los Movimientos retorna true o false segun corresponda.

→ **Extraccion y Depositos retornan true**

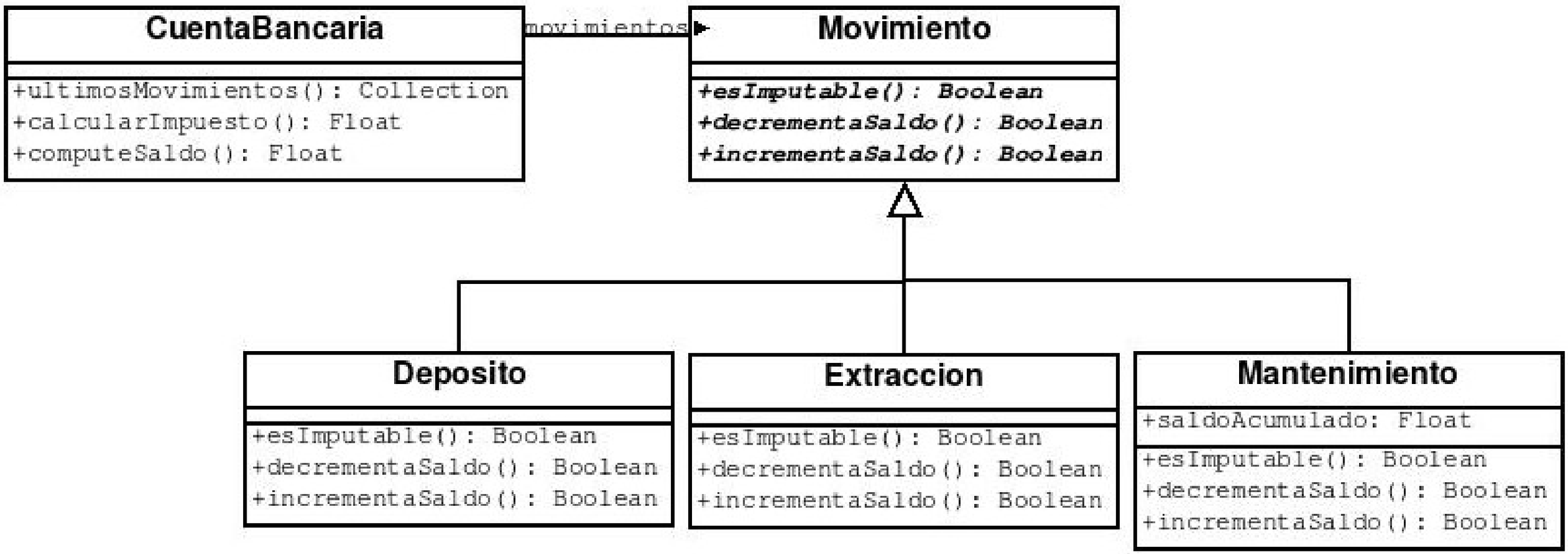
→ **Mantenimiento retorna false**

```
CuentaBancaria>>ultimosMovimientos
 “retorna los movimientos posteriores al ultimo mantenimiento”
```

CuentaBancaria>>ultimosMovimientos
"retorna los movimientos posteriores al ultimo mantenimiento"

1) "Detectar" el ultimo Mantenimiento (varias maneras de hacerlo)
2) "seleccionar aquellos movimientos posteriores al último movimiento

Modelo Resultante



Colecciones y jerarquias

Escenario

Una clase tiene una variable donde guarda una coleccion de objetos que vienen de una jerarquia

FAQ

1) Guardamos todos esos objetos en una variable o en multiples variables?

EN UNA VARIABLE

2) Si tenemos que discriminar solo un tipo de esos objetos preguntamos por la clase?

NUNCA PREGUNTE POR LA CLASE. IMPLEMENTE **METODOS FILTRO** SEGúN LOS ROLES DE LOS OBJETOS EN EL PROBLEMA

Resumiendo

- Todos los Movimientos en una colección

- Colecciones “tematicas” o vistas se obtienen por medio de envio de mensajes

- No se pregunta por clases de la jerarquia

- Se distinguen “roles” que juegan los objetos en el dominio, por ejemplo: si un movimiento es imputable o no.

- La clase continente de la coleccion (CuentaBancaria) implementa las vistas en metodos separados.

- Solo en caso de performance se podría convertir una vista en una variable de instancia. Como ultimo recurso

Bonus Track

Piense en los siguientes problemas:

1) Bucket Chain Pump

2) Mi lista de reproducción es Spotify se reinicia cuando llega al final

3) Una de las politicas de scheduling de procesos es “round robin”

CircularQueue!

Deberia entender:

- **#push:**
- **#top**
- **#remove:**

Es una coleccion o tiene una coleccion?

Qué tipo de colección ?