

001

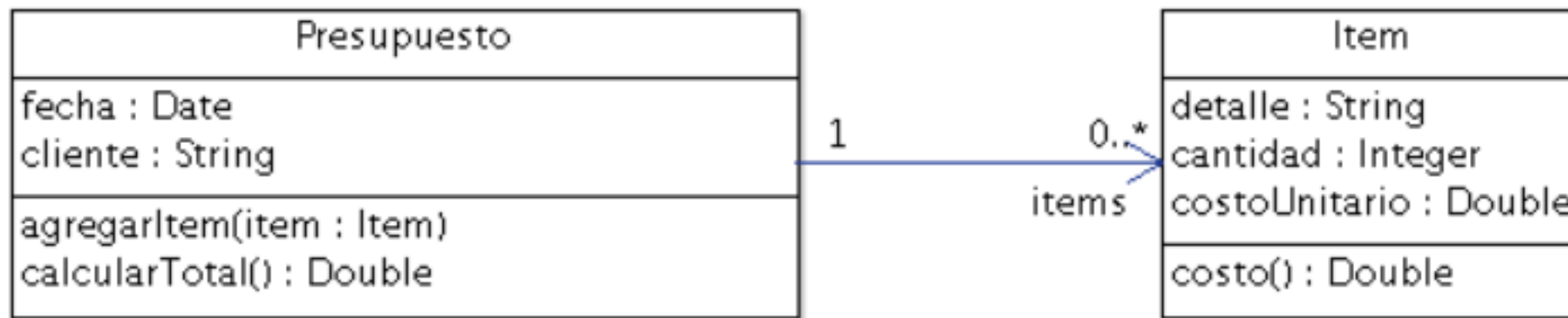
Explicación práctica

Semana: 10/9

# Referencias entre objetos

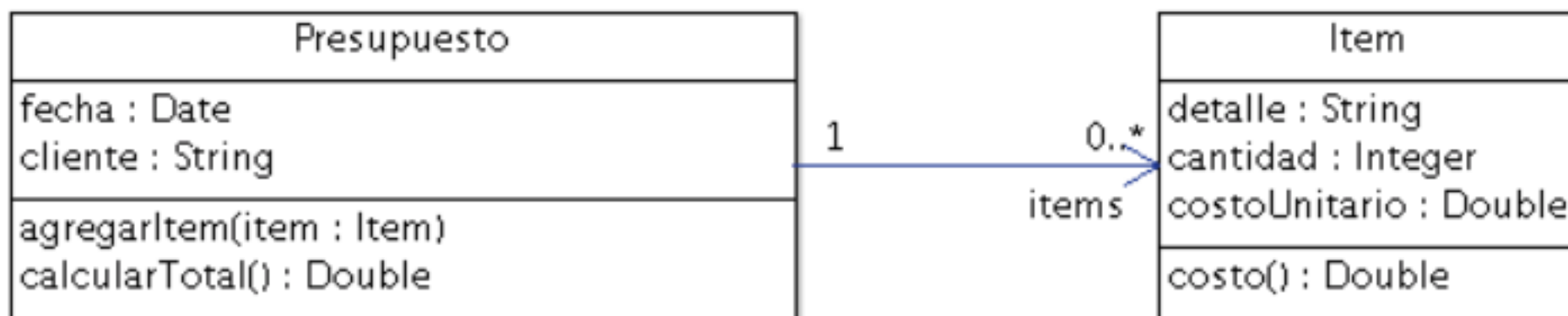
0..\* - 1..\*

- Presupuesto e ítems...



# Colecciones

- En Smalltalk, las variables solo pueden tener una referencia a un objeto
- Presupuesto tendrá una variable de instancia **“items”**
- ¿qué objeto asignamos a la v.i. items?



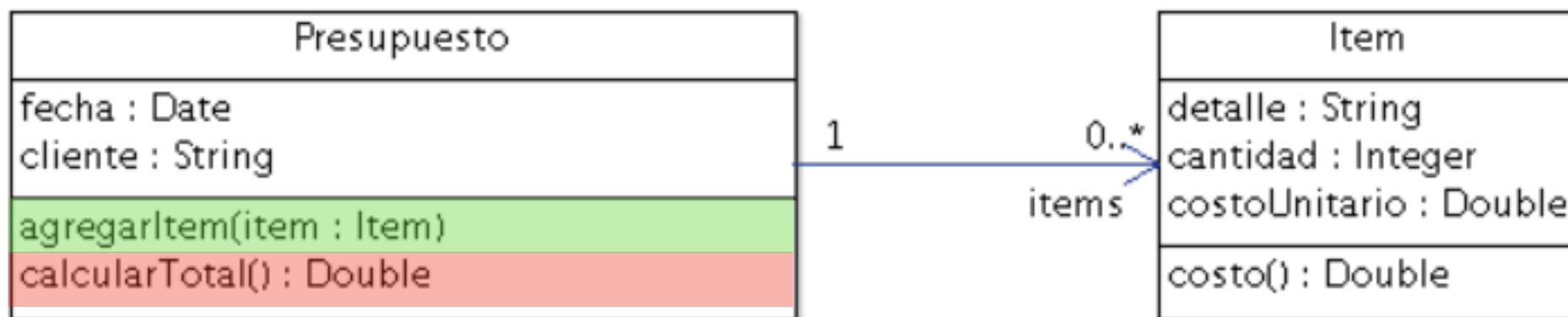


# COLLECTION



# Colecciones

- Presupuesto tendrá una variable de instancia “**items**”



```
#Presupuesto>>agregarItem: item
```

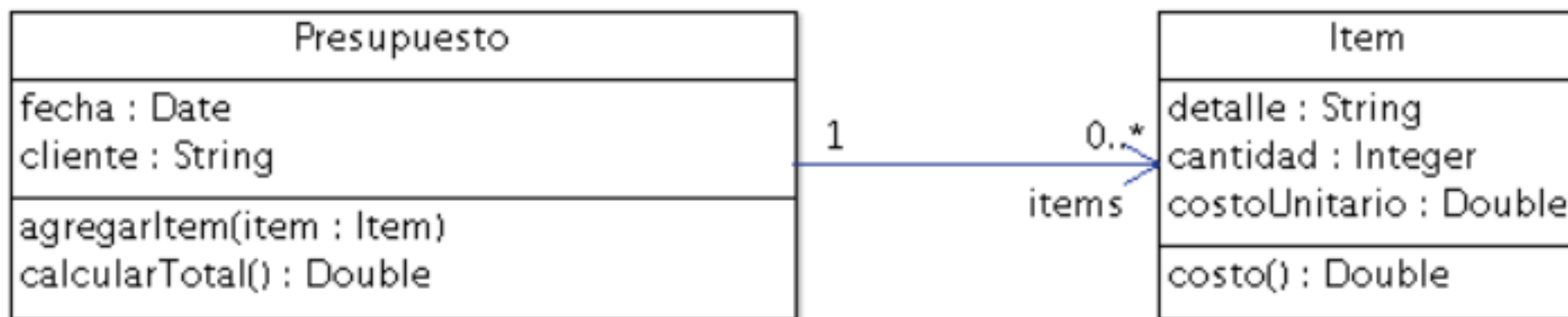
```
items add: item.
```

```
#Presupuesto>>initialize
```

```
items := OrderedCollection new.
```

# Colecciones

- Presupuesto tendrá una variable de instancia “**items**”



```
#Presupuesto>>calcularTotal
```

```
| total |
```

```
total := 0.
```

```
items do: [ :item | total := item costo + total].
```

```
^total.
```

**ITERADOR**  
#do:







Diferentes problemas. Diferentes herramientas

# Colecciones

- Supongamos que queremos agregar nuevo comportamiento a Presupuesto:
  - Retornar la cantidad total de productos.

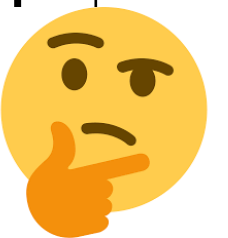
```
#Presupuesto>>cantidadDeProductos
```

```
| cantidad |
```

```
cantidad := 0.
```

```
items do: [ :item | cantidad := cantidad + item cantidad ]
```

```
^cantidad.
```





# Colecciones

- Supongamos que queremos agregar nuevo comportamiento a Presupuesto:
  - Obtener todos los ítems con costo mayor a un número X.

```
#Presupuesto>>itemsConCostoMayorA: unCosto
```

```
| itemsARetornar |
```

```
itemsARetornar := OrderedCollection new.
```

```
items do: [ :item | item costo > unCosto ifTrue:[
```

```
                itemsARetornar add: item.
```

```
            ]
```

```
].
```

```
^itemsARetornar.
```



# Colecciones

- Supongamos que queremos agregar nuevo comportamiento a Presupuesto:
  - Obtener todos los ítems que tengan mas de X cantidad de productos

```
#Presupuesto>>itemsConCantidadMayorA: cantidad
```

```
| itemsARetornar |
```

```
itemsARetornar := OrderedCollection new.
```

```
items do: [ :item | item cantidad > cantidad ifTrue:[
```

```
                itemsARetornar add: item.
```

```
            ]
```

```
].
```

```
^itemsARetornar.
```



# Colecciones

```
#Presupuesto>>cantidadDeProductos
```

```
| cantidad |
```

```
cantidad := 0.
```

```
items do: [ :item | cantidad := cantidad + item cantidad ].
```

```
^cantidad.
```

```
#Presupuesto>>calcularTotal
```

```
| total |
```

```
total := 0.
```

```
items do: [ :item | total := item costo + total].
```

```
^total.
```

# Colecciones

```
#Presupuesto>>cantidadDeProductos
```

```
  | cantidad |
```

```
  cantidad := 0.
```

```
  items do: [ :item | cantidad := cantidad + item cantidad ].
```

```
  ^cantidad.
```

```
#Presupuesto>>calcularTotal
```

```
  | total |
```

```
  total := 0.
```

```
  items do: [ :item | total := item costo + total].
```

```
  ^total.
```

# Colecciones

```
#Presupuesto>>itemsConCantidadMayorA: cantidad
```

```
| itemsARetornar |
```

```
itemsARetornar := OrderedCollection new.
```

```
items do: [ :item | item cantidad > cantidad ifTrue:[
```

```
    itemsARetornar add: item.
```

```
]
```

```
].
```

```
^itemsARetornar.
```

```
#Presupuesto>>itemsConCostoMayorA: unCosto
```

```
| itemsARetornar |
```

```
itemsARetornar := OrderedCollection new.
```

```
items do: [ :item | item costo > unCosto ifTrue:[
```

```
    itemsARetornar add: item.
```

```
]
```

```
].
```

```
^itemsARetornar.
```

# Colecciones

```
#Presupuesto>>itemsConCantidadMayorA: cantidad
```

```
| itemsARetornar |
```

```
itemsARetornar := OrderedCollection new.
```

```
items do: [ :item | item cantidad > cantidad ifTrue:[
```

```
    itemsARetornar add: item.
```

```
]
```

```
].
```

```
^itemsARetornar.
```

```
#Presupuesto>>itemsConCostoMayorA: unCosto
```

```
| itemsARetornar |
```

```
itemsARetornar := OrderedCollection new.
```

```
items do: [ :item | item costo > unCosto ifTrue:[
```

```
    itemsARetornar add: item.
```

```
]
```

```
].
```

```
^itemsARetornar.
```

# Colecciones - iteradores

- Además de **#do**:
  - las colecciones entienden otros mensajes que nos ofrecen comportamiento para iterarlas con cierto objetivo:
    - **#select**:
    - **#reject**:
    - **#inject: into**:
    - **#collect**:
    - **#detect**:
    - **#contains**:



# Colecciones - iteradores

- #select: / #reject:
  - retornan una colección con aquellos elementos que satisfacen una condición

# Colecciones - iteradores

```
#Presupuesto>>itemsConCantidadMayorA: cantidad
```

```
| itemsARetornar |
```

```
itemsARetornar := OrderedCollection new.
```

```
items do: [ :item | item cantidad > cantidad ifTrue:[
```

```
                itemsARetornar add: item.
```

```
            ]
```

```
        ].
```

```
^itemsARetornar.
```

```
#Presupuesto>>cantidadDeProductos: cantidad
```

```
^items select: [ :item | item cantidad > cantidad]
```



# Colecciones - iteradores

```
#Presupuesto>>itemsConCostoMayorA: unCosto
```

```
| itemsARetornar |
```

```
itemsARetornar := OrderedCollection new.
```

```
items do: [ :item | item costo > unCosto ifTrue:[
```

```
                    itemsARetornar add: item.
```

```
                ]
```

```
        ].
```

```
^itemsARetornar.
```

```
#Presupuesto>>itemsConCostoMayorA: unCosto
```

```
^items reject: [ :item | item costo <= unCosto]
```



# Colecciones - iteradores

- #inject: into:
  - retorna el resultado de un computo que comienza utilizando un valor inicial

# Colecciones - iteradores

- #inject: into:

```
#Presupuesto>>calcularTotal
```

```
| total |
```

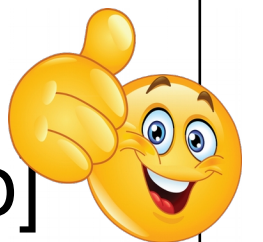
```
total := 0.
```

```
items do: [ :item | total := item costo + total].
```

```
^total.
```

```
#Presupuesto>>calcularTotal
```

```
^items inject: 0 into: [ :total :item | total + item costo]
```



# Colecciones - iteradores

- `#collect:`
  - retorna el una colección que tiene como objetos el resultado de un computo para cada elemento de la colección
  - Presupuesto: Obtener todas las descripciones (detalles) de los items.

```
#Presupuesto>>detalles
```

```
^items collect: [ :item | item detalle ]
```



# Colecciones - iteradores

- #detect:
  - retorna el primer elemento que cumple con una condición, y falla si no existe
  - Presupuesto: Obtener un ítem cuyo costo sea mayor a un número X.

```
#Presupuesto>>itemCostoMayorA: costo
```

```
^items detect: [ :item | item costo > costo ]
```

```
#Presupuesto>>itemCostoMayorA: costo
```

```
^items detect: [ :item | item costo > costo ] ifNone: [...]
```





# Colecciones - iteradores

- #contains:
  - retorna verdadero si al menos un elemento cumple con una condición, y falso en caso contrario
  - Presupuesto: ¿tiene ítems cuya cantidad sea mayor a X?.

```
#Presupuesto>>tieneItemConCantidadMayorA: cantidad
```

```
^items contains: [ :item | item cantidad > cantidad ]
```



# Colecciones - iteradores

```
#Presupuesto>>tieneItemConCantidadMayorA: cantidad  
^items contains: [ :item | item cantidad > cantidad ]
```

```
#Presupuesto>>tieneItemConCantidadMayorA: cantidad  
^items contains: [ :item | item cantidad > cantidad ]
```

Objeto

Mensaje  
de  
palabra  
clave

Parametro  
si es un parámetro... ¿qué es?

```
#Presupuesto>>itemsConCostoMayorA: unCosto  
^items reject: [ :item | item costo <= unCosto]
```

```
#Presupuesto>>calcularTotal  
^items inject: 0 into: [ :acumulado item costo ]
```

BlockClosure

```
#Presupuesto>>cantidadDeProductos: cantidad  
^items select: [ :item | item cantidad > cantidad]
```

# BlockClosure

- Son objetos que permiten evaluar programáticamente expresiones para obtener el valor.
  - #value
  - #value:
  - #value:value:
- Ejemplos:
  - [3+4] value
  - [:p1| p1 + 4] value: 3
  - [:p1 :p2| p1 + p2] value: 3 value:4

# BlockClosure

```
p1 := Producto new precioPorKilo: 10; peso: 20.  
bloque := [ :p | p precioPorKilo > 100].  
bloque value: p1.
```

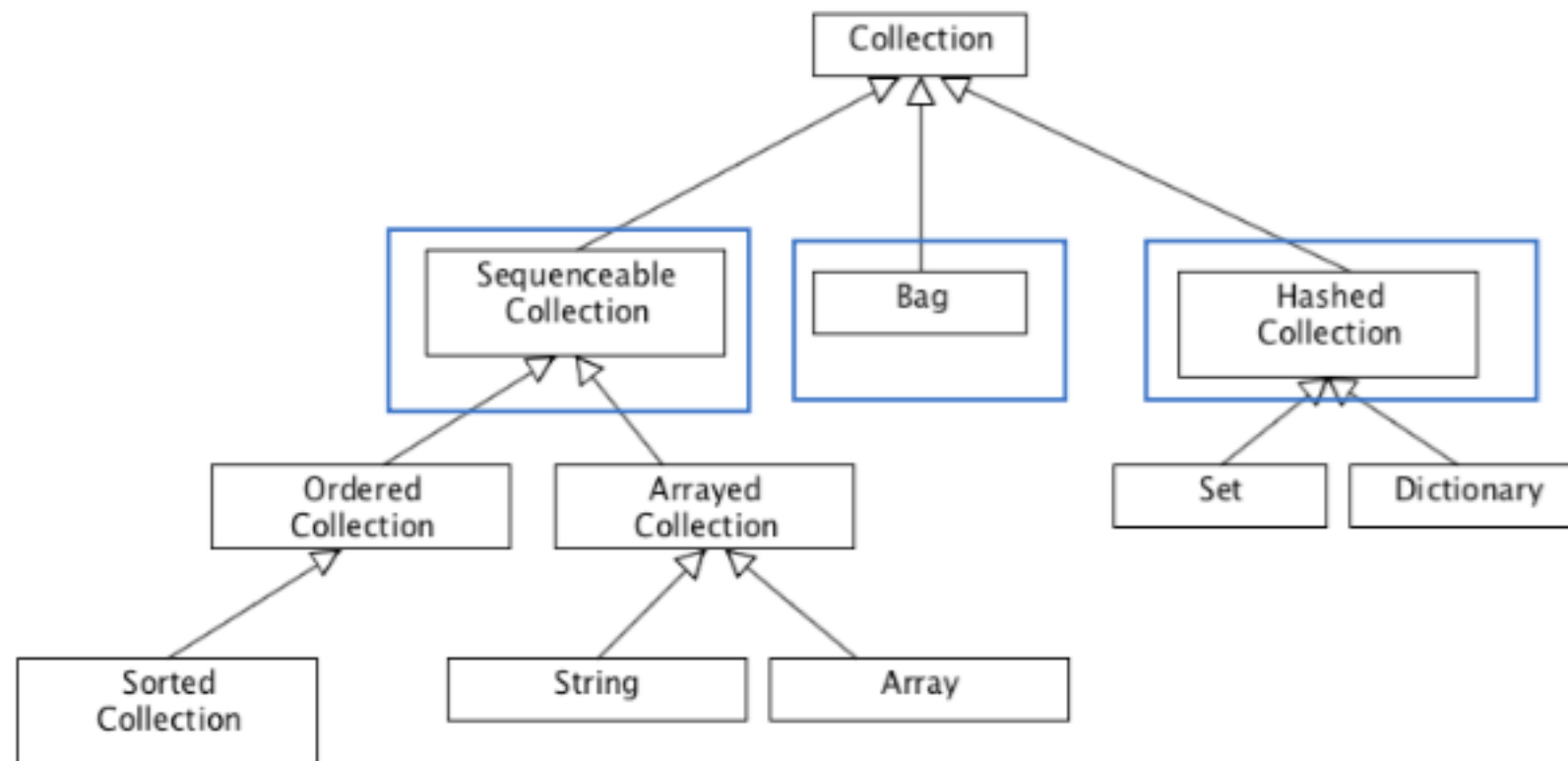
¿Qué retorna **#value**: cuando se lo enviamos a **bloque**?

```
p1 := Producto new precioPorKilo: 10; peso: 20.  
p2 := Producto new precioPorKilo: 90; peso: 1.  
bloque := [ :instancia1 :instancia2 | instancia1 precio + instancia2 precio].  
bloque value: p1 value: p2.      <— #value: value:
```

**¿Cómo es la colaboración entre una OrderedCollection y BlockClosure cuando enviamos un mensaje que itera la colección?**

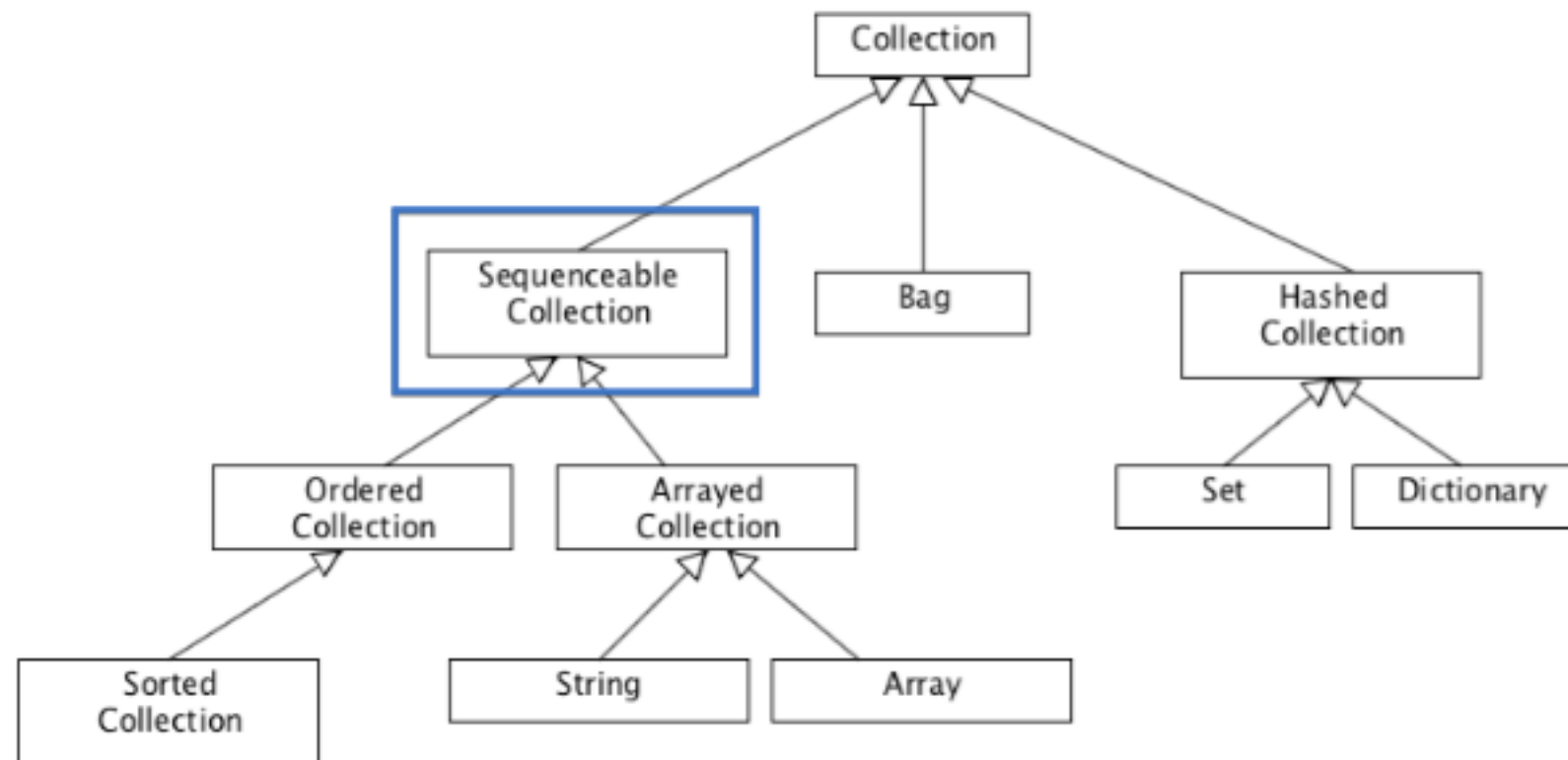
```
#Presupuesto>>cantidadDeProductos: cantidad  
^items select: [ :item | item cantidad > cantidad]
```

# Colecciones... Mas alla de OrderedCollection



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

# Sequenceable Collection



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

# SortedCollection

- Ordena la colección según un criterio específico

```
producto1 := Producto new peso: 1; precioPorKilo: 130.  
producto2 := Producto new peso: 1.5; precioPorKilo: 100.  
producto3 := Producto new peso: 2; precioPorKilo: 80.
```

```
collection := SortedCollection sortBlock: [ :producto1 :producto2 |  
                                           producto1 peso > producto2 peso].
```

```
collection add: producto1; add: producto2; add: producto3.
```

```
collection first. ¿Qué producto retorna?
```



# Array

- Colección indexada y de tamaño predefinido...
- ¿qué pasa con #add:? ¿#remove:?

```
collection := Array new:10.  
collection at:1 put: producto3.
```

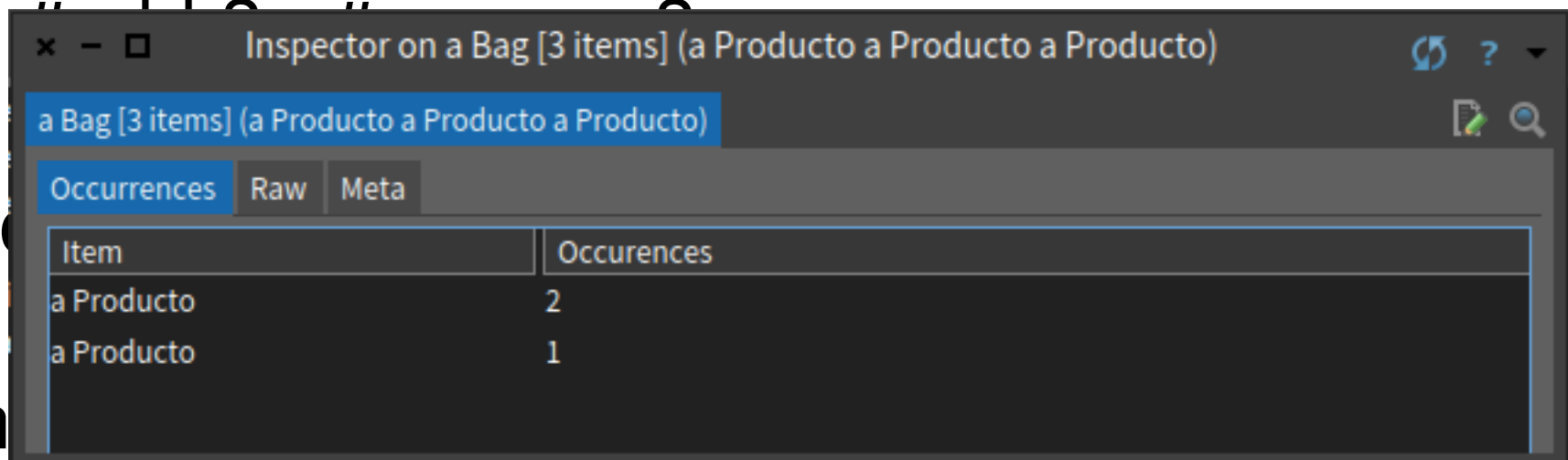
# Set

- Lógica de conjunto
- Colección **no** indexada
- Sin orden... por ejemplo, no entiende #first
  - ¿Cuál es el comportamiento de #add:?
  - ¿Cuántas veces se referencia a producto1 en el siguiente código?

```
producto1 := Producto new peso: 1; precioPorKilo: 130.  
producto2 := Producto new peso: 1.5; precioPorKilo: 100.  
producto3 := Producto new peso: 2; precioPorKilo: 80.  
  
collection := Set with: producto1.  
collection add: producto2; add: producto3; add: producto1.
```

# Bag

- Colección **no** indexada
- Sin orden... por ejemplo, no entiende #first
- ¿qué pasa con "#add: a"?
- Acepta repetición
  - #add: with



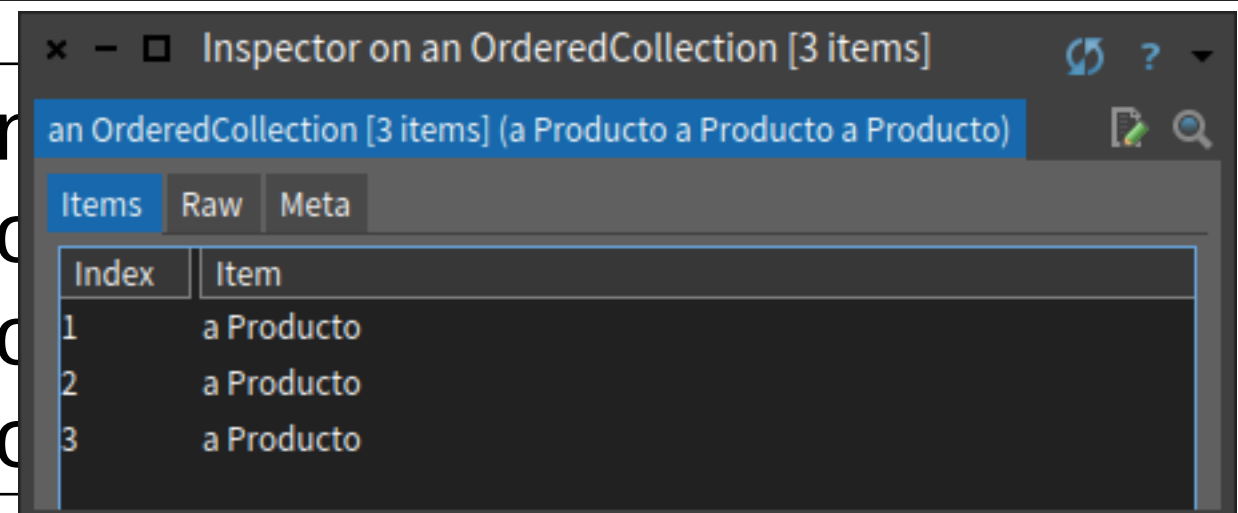
Inspector on a Bag [3 items] (a Producto a Producto a Producto)

a Bag [3 items] (a Producto a Producto a Producto)

Occurrences Raw Meta

Item	Occurrences
a Producto	2
a Producto	1

```
collection := Bag new
collection add: producto
collection add: producto
collection occurrenceOf: producto
```



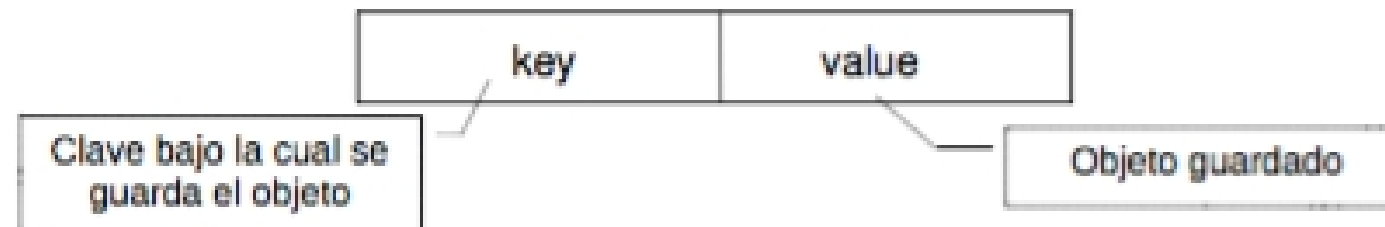
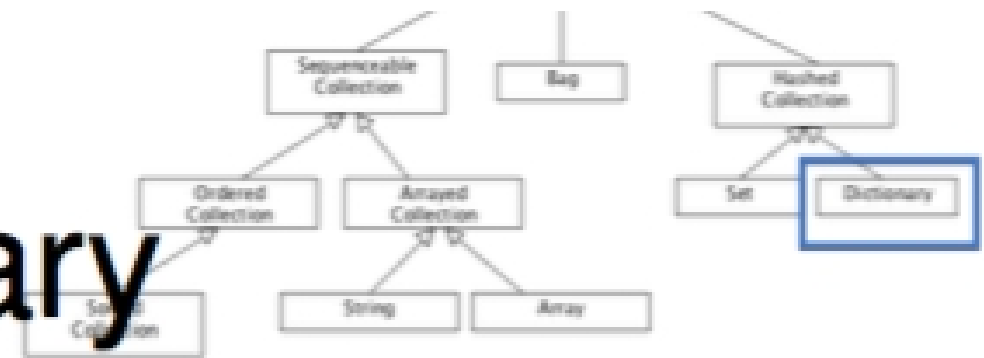
Inspector on an OrderedCollection [3 items]

an OrderedCollection [3 items] (a Producto a Producto a Producto)

Items Raw Meta

Index	Item
1	a Producto
2	a Producto
3	a Producto

# 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
```

