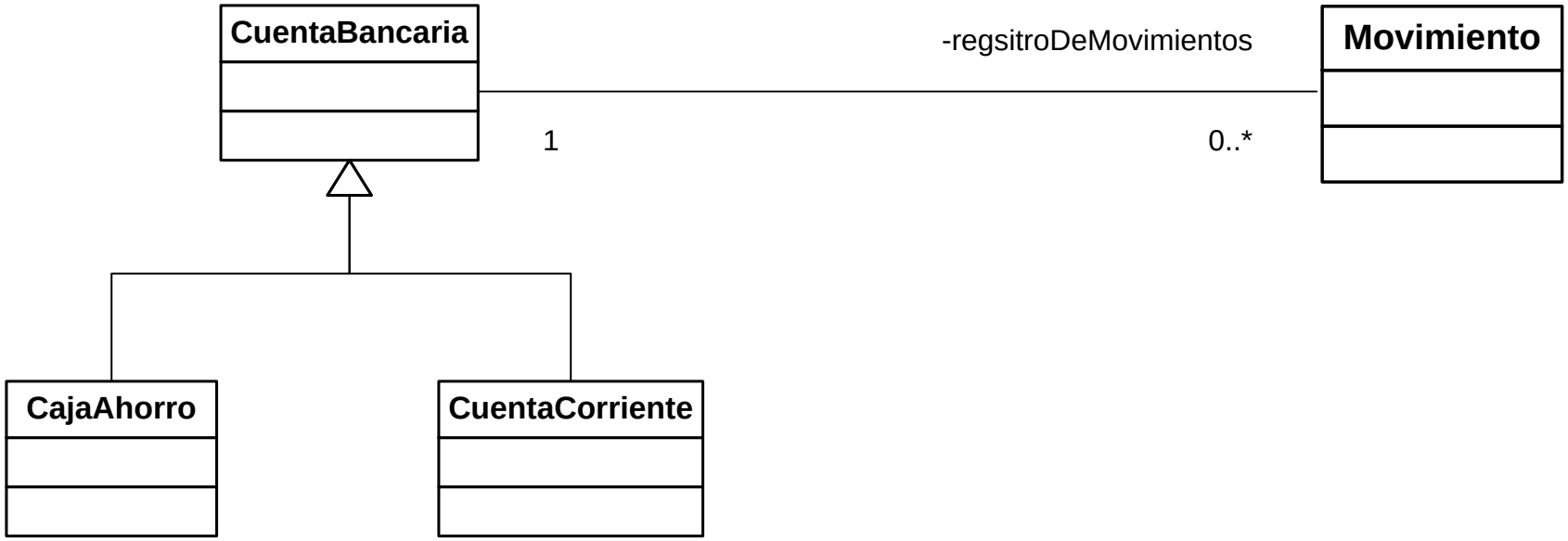
Crear registro de movimientos

Registro de Movimientos

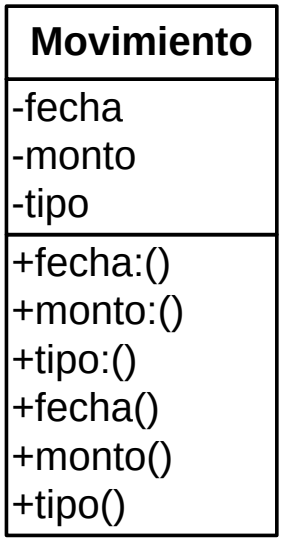
- Registra cada movimiento que ocurrió en una cuenta bancaria
- Un movimiento representa una transacción en una cuenta bancaria
  - Pueden ser por extracción o depósito
- Mensualmente se usa para generar el resumen de cuenta

Regitro de Movimientos



Movimiento

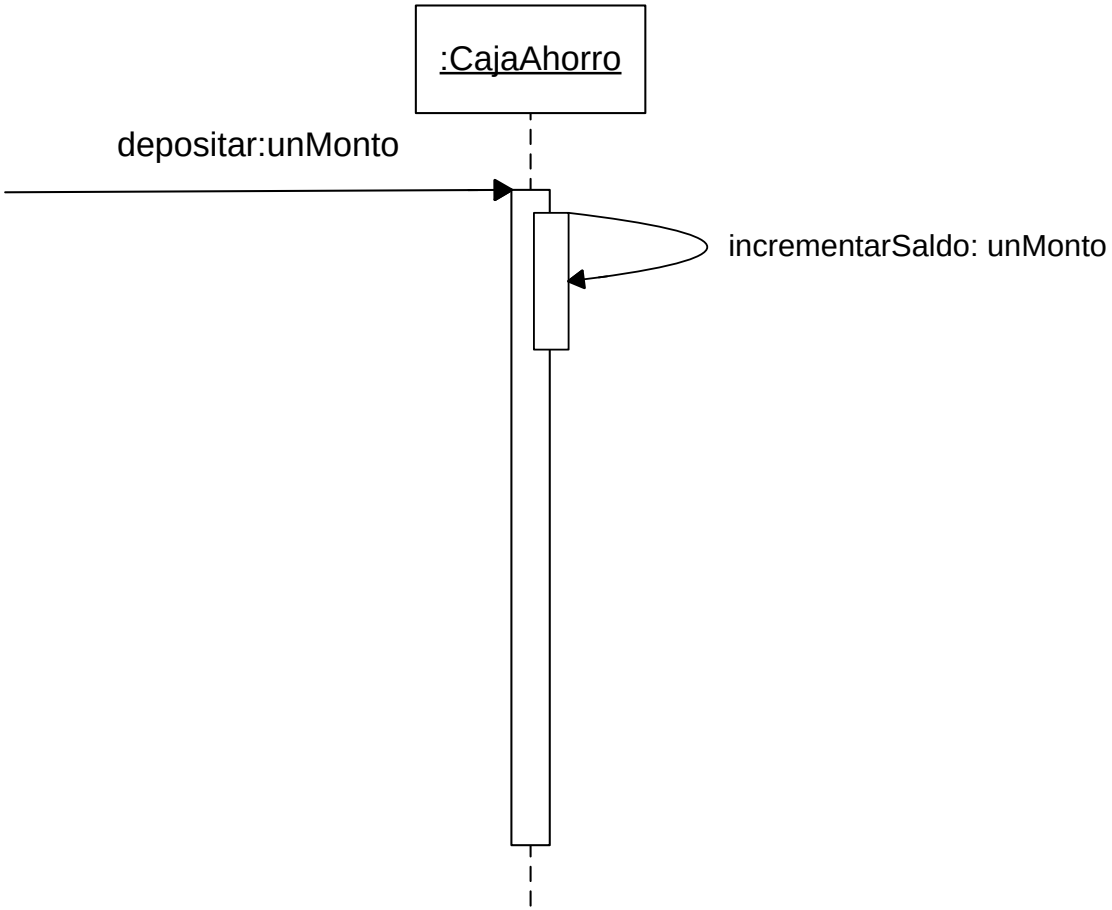
- Registra
  - La fecha que ocurrió la transacción
  - El tipo de transacción
  - El monto involucrado en la transacción



Cuando se crea?

- Cuando se ejecuta una transacción
  - `depositar:`
  - `extraer:`
- En el caso del **`depositar:`** debemos:
  - modificar el saldo
  - registrar el movimiento

Diagrama Secuencia: depositar:

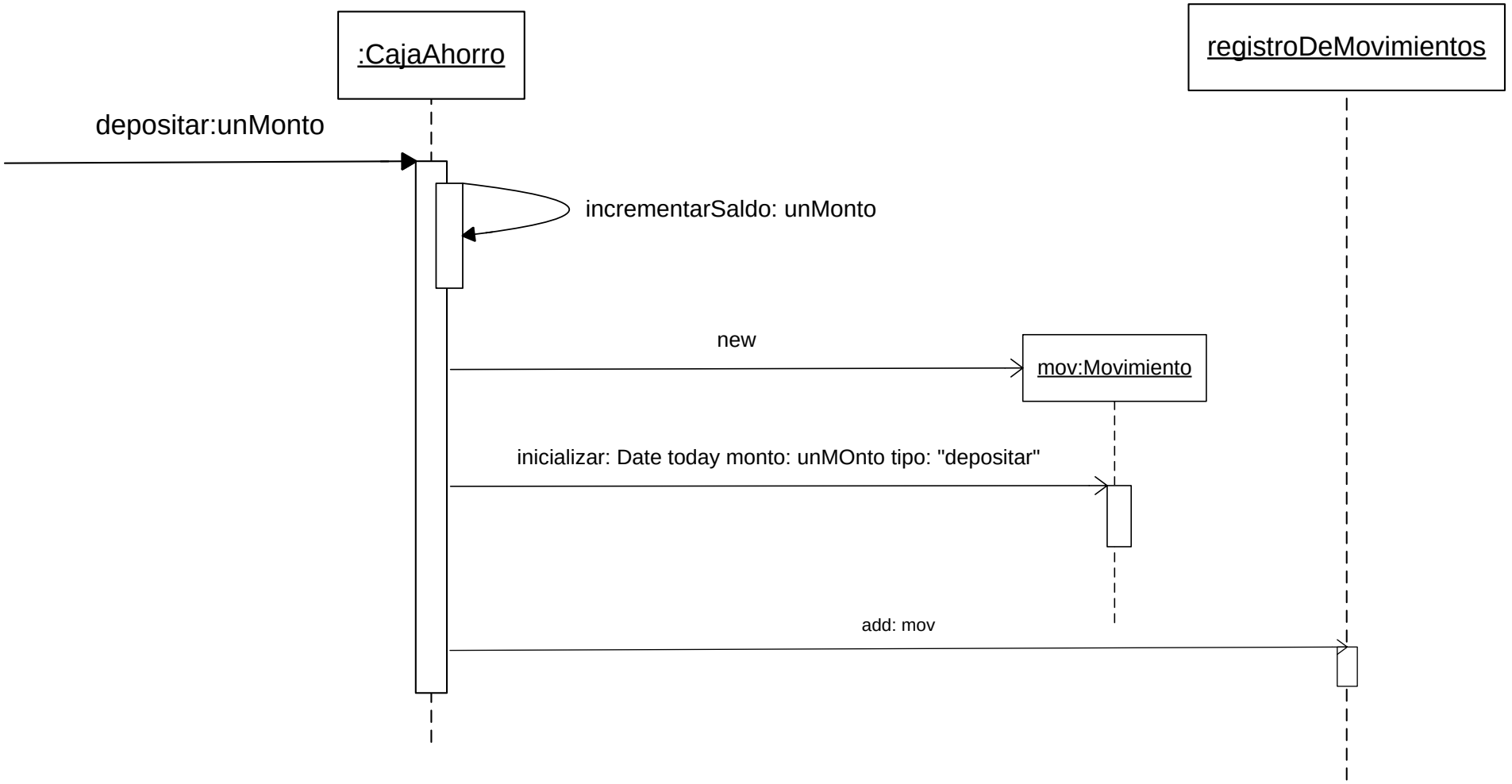


Luego debemos registrar el movimiento …

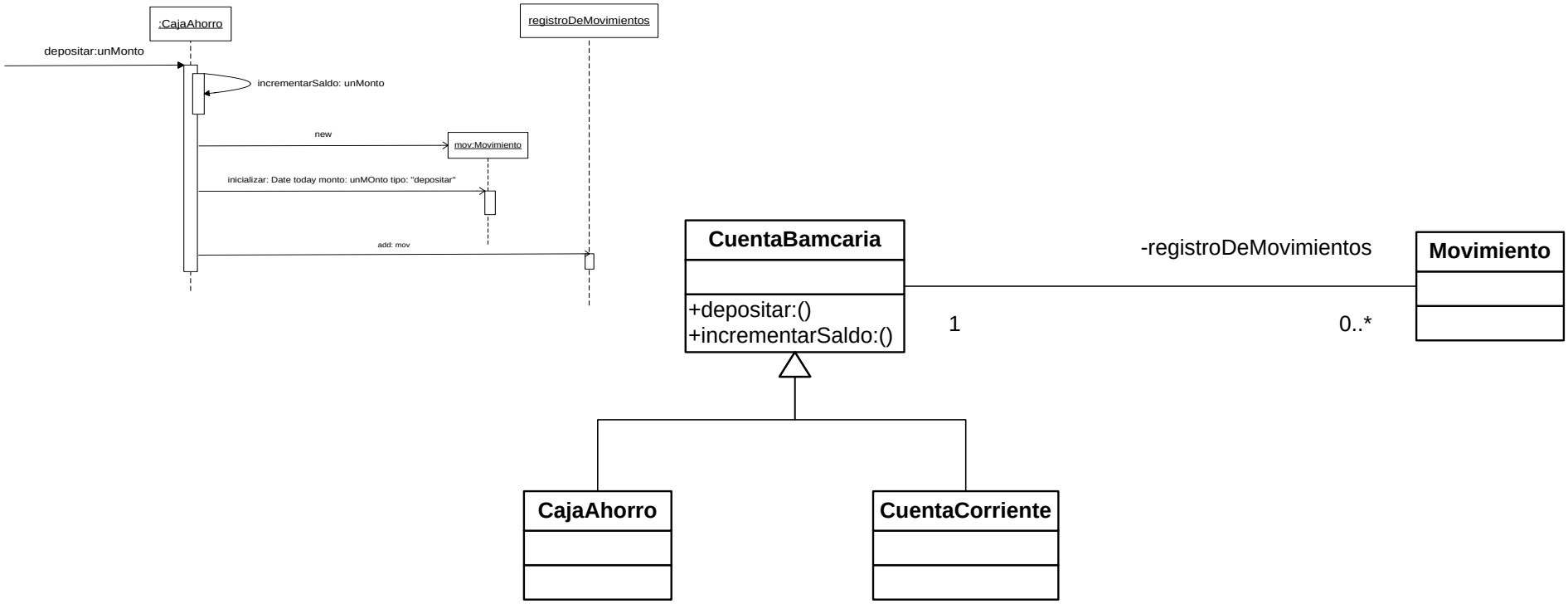
registar el Movimiento

- Implica:
  - crearlo
  - inicializarlo
    - la fecha actual
      - `Date today`
    - el monto
      - `unMonto`
    - y el tipo de operación
      - `"deposito"`
  - agregarlo al registro de movimientos

Diagrama Secuencia: depositar:



registar el Movimiento




```
CuentaBancaria>>depositar: unMonto

self incrementarSaldo: unMonto.
|mov|
mov := Movimiento new inicializar:Date today monto:unMonto tipo:"depositar"
registroDeMovimientos add: mov.
```

Un buen programador Smalltalk ....

- Evitaría la variable temporal

```
CuentaBancaria>>depositar: unMonto

self incrementarSaldo: unMonto.

registroDeMovimientos add: (Movimiento new inicializar:Date today
                                          monto:unMonto
                                          tipo:"depositar")
```

Qué es el registro de movimientos ???

Colecciones Smalltalk

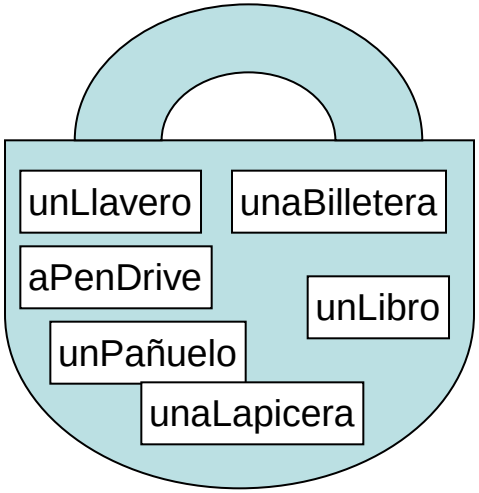
Lifia

Contenido

- ¿Qué es una colección SmallTalk?
- Propiedades de las colecciones
  - Comportamiento de una colección
  - Heterogeneidad
- Ejemplos de uso
- Jerarquía de Collection
    - Bag y Set
    - Array
    - OrderedCollection
      - SorteredCollection
    - Dictionary
- Iteradores - recorriendo colecciones
    - do:
    - select:, detect:, collect:, reject:

¿Qué es una colección ST?

- Es un objeto que "contiene" un grupo de otros objetos
- Los objetos contenidos se llaman "elementos" de la colección



`miBolso`

Comportamiento básico

- Clase Collection
- Responsabilidades:
  - Contener los elementos
    - Testear y consultar la colección y su contenido
      - `miBolso size`
      - `miBolso includes: unLibro`
      - `miBolso isEmpty`
    - Permitir agregar y eliminar elementos
      - `miBolso add: otroLibro`
      - `miBolso remove: unaLapicera`

Heterogeneidad

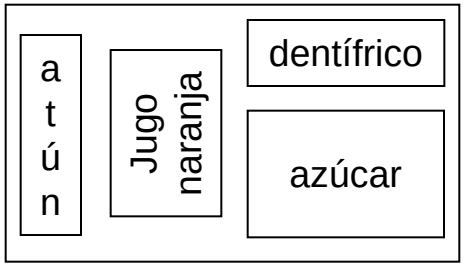
- Una colección puede contener objetos de cualquier clase

- No se declara de que clase son sus elementos
  - ST es no tipado

  - `col ← Collection`
  - `col add: anObject`

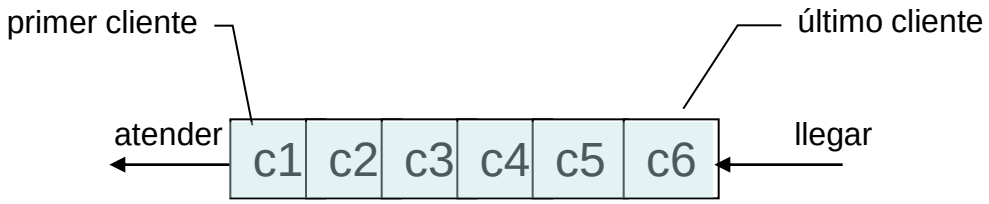
- ejemplo **`miBolso`** es heterogeneo
- Excepciones: **`String`**, que es una colección de **`Characters`**

Ejemplos de Uso

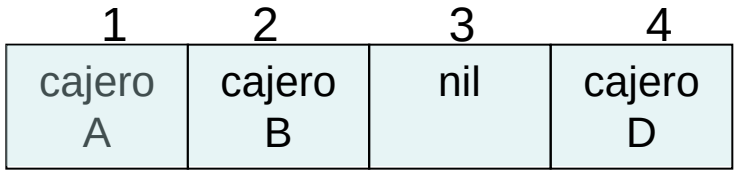


ticket de compras

Agregar productos
Calcular el total



Clientes en la cola de una caja
(colaClientes)



Asignación de cajeros
(cajasSupermercado)

Cambiar el cajero de una caja en particular

azúcar – 3

dentífrico – 3,40

atun – 5.60

jugoNaranja –2,70

Lista de Precios
(listaPrecios)

Conocer el precio de un producto

Jerarquía de la clase Collection

```
Object
    Collection
      Bag
      SequenceableCollection
            ArrayedCollection
              Array
              CharacterArray
                    String
                          Symbol
            OrderedCollection
              SortedCollection
      Set
       Dictionary
```

Instanciación de colecciones

```
myColl := OrderedCollection new.

myColl := Array new: 30.

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

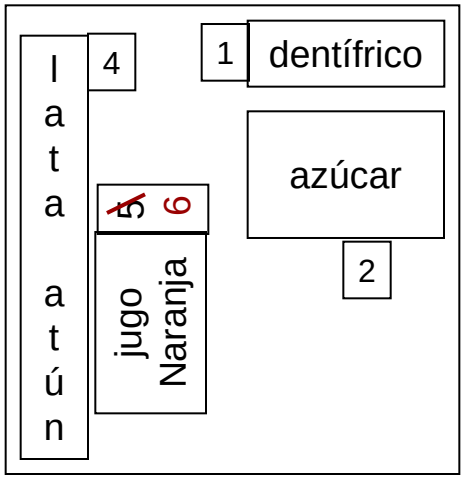
- Luego, a través del **add:** algunas pueden crecer

Eligiendo el tipo de una colección?

- ¿se necesita tener los elementos ordenados?
- ¿en caso afirmativo, cómo se determina el orden?
- ¿los elementos necesitan ser accedidos por una clave?
    - ¿cómo es la clave?
- ¿habrá elementos repetidos?

Bag

- Es una colección desordenada y sin acceso por clave de elementos
- No secuenciable
- Permite elemento repetidos



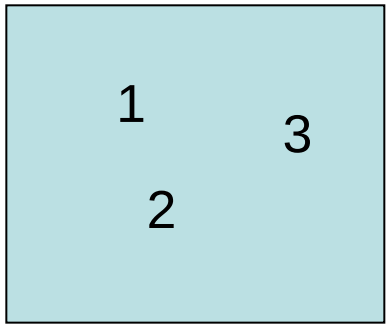

```
shoppingCart:= Bag new.
shoppingCart add: ….

shoppingCart add:  jugoNaranja

shoppingCart occurencesOf: jugoNaranja  → 6
```

Set

- Es como el Bag, pero sin elementos repetidos




```
mySet:= Set new add:3; add:2; add:1.

mySet add: 3

mySet occurencesOf: 3  → 1
```

Array

- Es una *secuencia* de elementos accesibles por un índice entero

1	2	3	4
cajero A	cajero B	nil	cajero D

```
cajasSupermercado at: 2
```
→ cajeroB

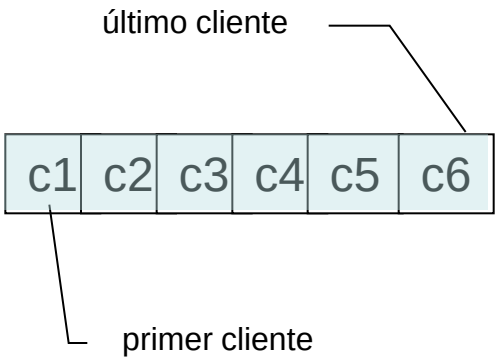
El `add:` esta cancelado

```
cajasSupermercado at: 3 put: cajeroX
```

cajero A	cajero B	cajero X	cajero D

OrderedCollection

- Define una *secuencia* de elementos ordenados
- todo elemento tiene su anterior o siguiente, excepto los extremos




```
"llega un cliente"
        colaClientes addlast: unCliente

"atender"
        colaClientes removefirst
```

Otros métodos:

```
addFirst:, add: after:, first,
after:, before:
```

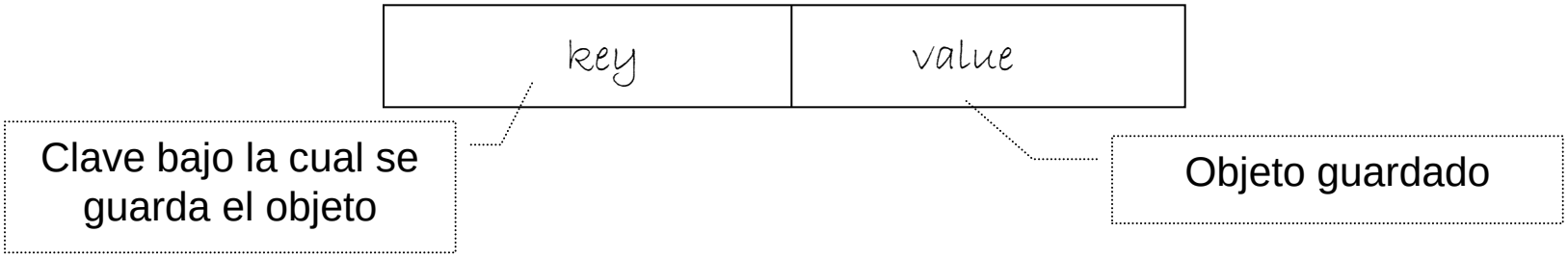
SorteredCollection

- Es una OrderedCollection, pero
- los elementos están ordenados por alguna característica de los elementos


```
listaNumeros:=SortedCollection sortBlock: [:nro1 :nro2| nro1 < nro2]
```

Dictionary

- Son como los arrays pero el índice es un objeto cualquiera
- Es un conjunto de pares:



azúcar – 3
dentífrico – 3,40
atun – 5.60
jugoNaranja –2,70

```
listaPrecios at: jugoNaranja      → 2,70

listaPrecios at: jugoNaranja put:2,65
```

Iteradores

- Las colecciones también son responsables de recorrerse a sí mismas
- Existen mensajes, llamados iteradores, que evalúan un bloque por cada elemento
- El más conocido es el do:
  - `col do: aBlock`

do:

```
col do: aBlock
```

para cada `elemento` perteneciente a la colección

```
aBlock value: elemento
```

Por ejemplo: aumentar el precio de los elementos de la lista de precios

```
listaPrecios do: [:prod| prod price: prod price*0,001]
```

Más iteradores

- `#(1 5 2 89 34 53) select: [:element| element > 28]`

- `#(1 5 2 89 34 53) reject: [:element| element > 28]`

- `#(1 5 2 89 34 53) collect: [:element| element > 28]`

- `#(1 5 2 89 34 53) detect: [:element| element > 28]`

- ```
  #(1 5 2 89 34 53) detect: [:element| (element rem: 3)=0]
                            ifNone: [Dialog warn: 'No such element']
  ```

- ```
  |sum|
  sum := #(1 5 2 89 34 53) inject:0
                           into:[:element :tempSum|  tempSum + element].
  ```