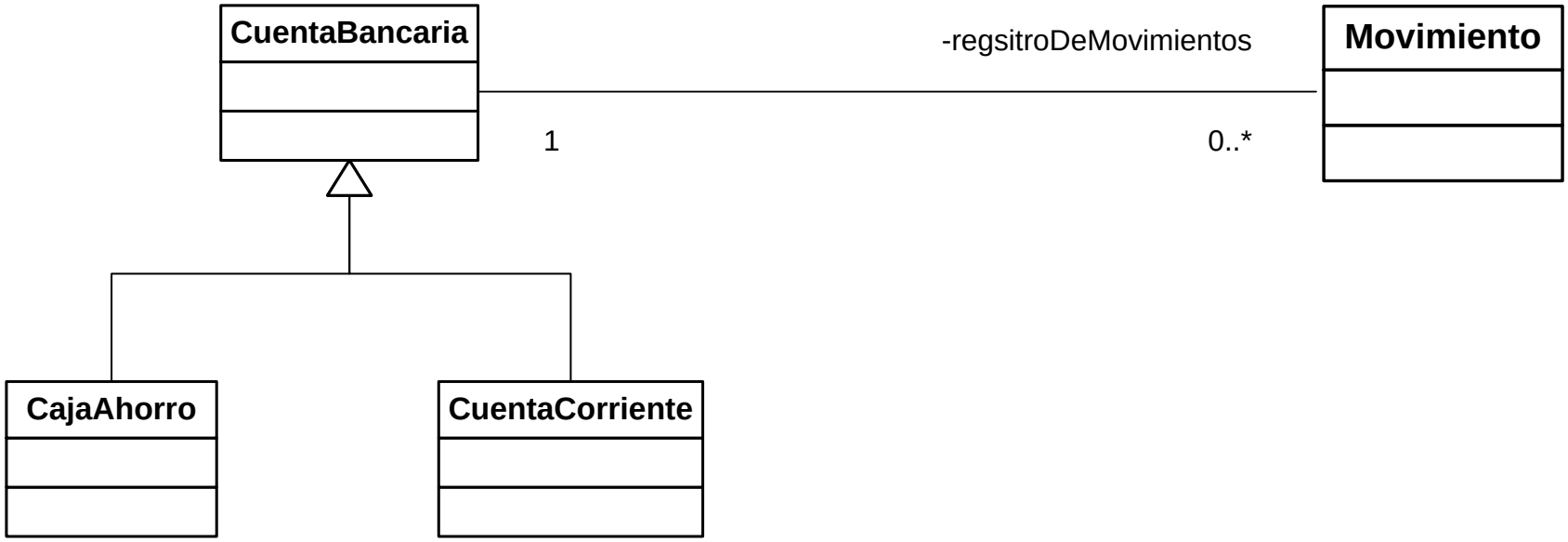
Uso de Colecciones y polimorfismo

Crear registro de movimientos

Registro de Movimientos

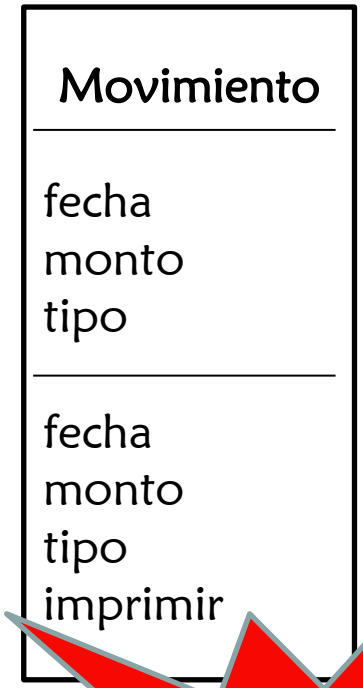
- Registra cada **movimiento** que ocurrió en una cuenta bancaria
- Un **movimiento** representa una transacción en una cuenta bancaria
  - Pueden ser por extracción o depósito
- Mensualmente se usa para generar el resumen de cuenta

Regitro de Movimientos



Movimiento

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



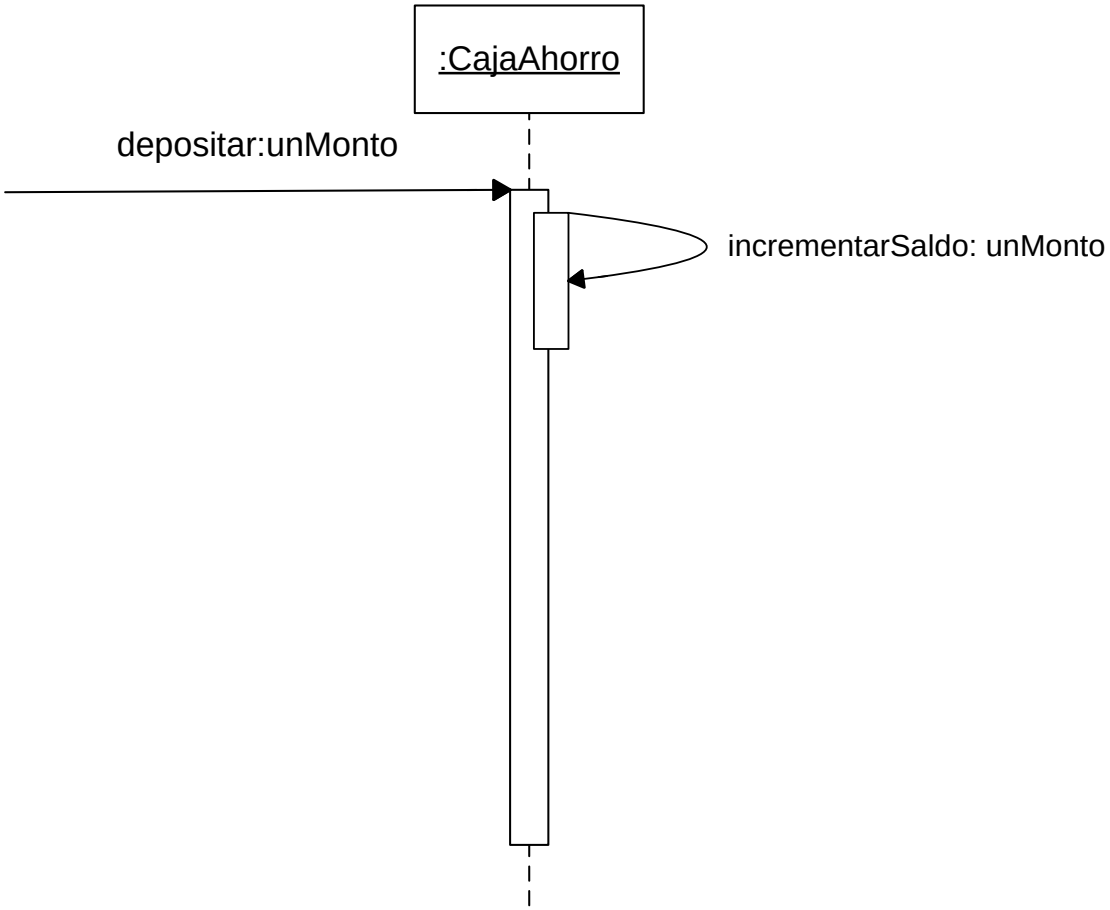
Esta bien tener la v.i. **tipo**?

Lo resolvemos mas adelante..

Cuando se crea?

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

Diagrama Secuencia del #depositar:

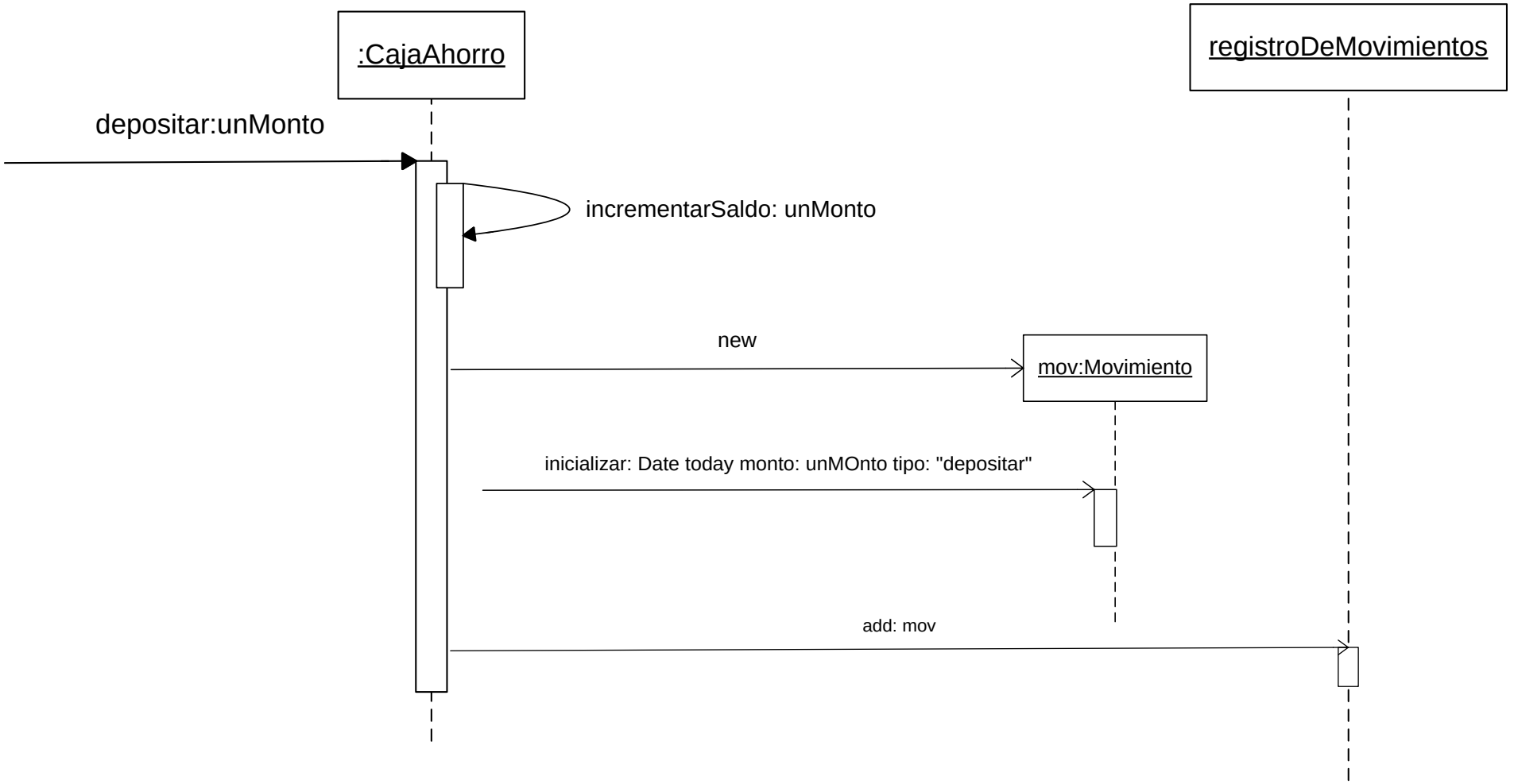


Luego debemos registrar el movimiento …

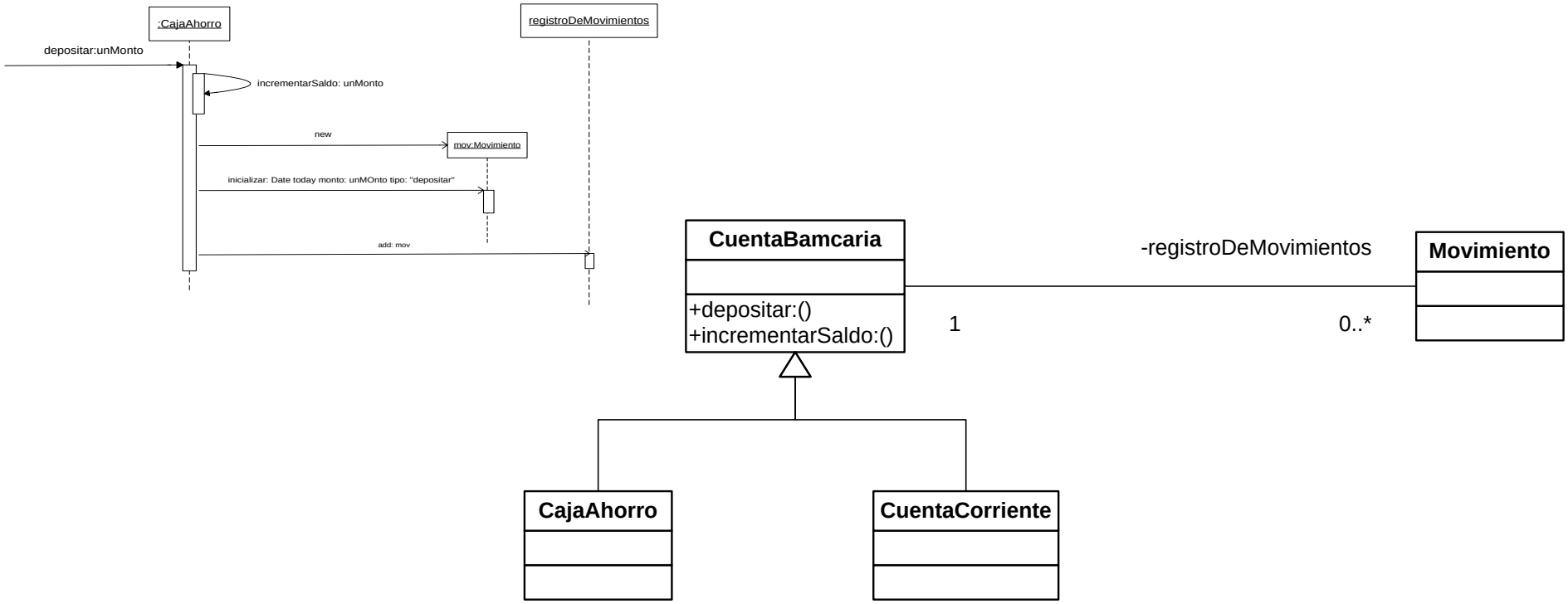
registar el Movimiento

- 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.
```

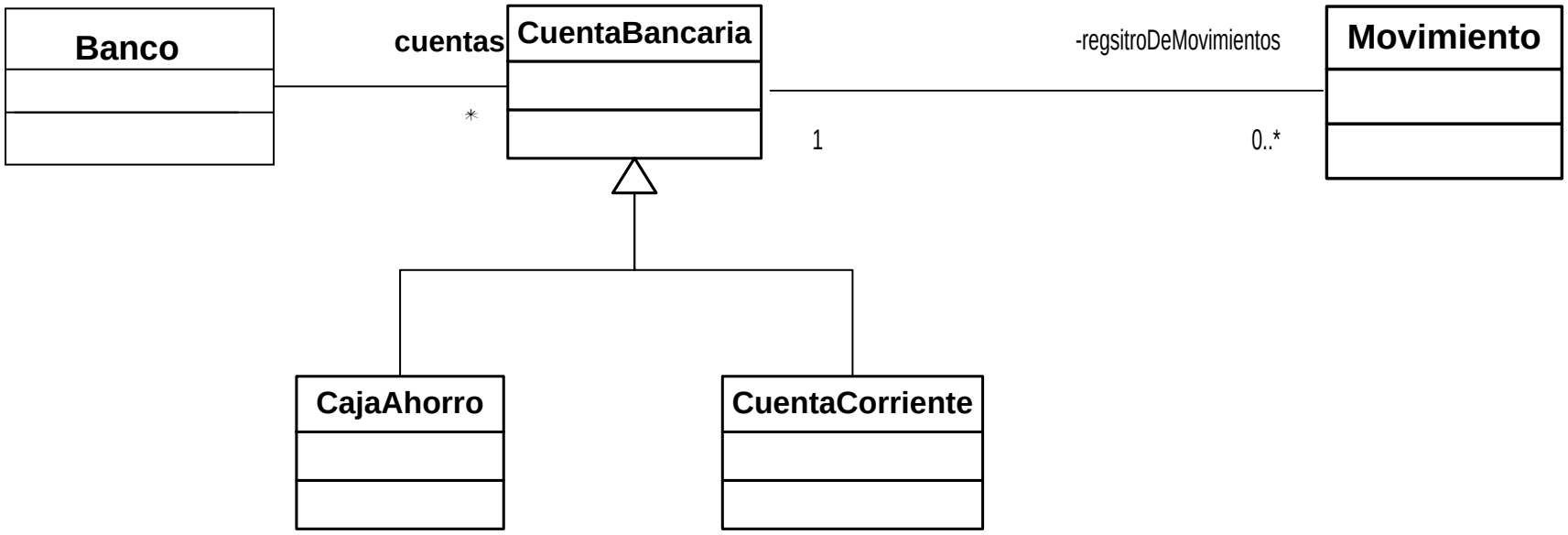
Un buen programador Smalltalk ....

- 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 ???

Crear el registroDeMovimientos para Cuentas Bancarias

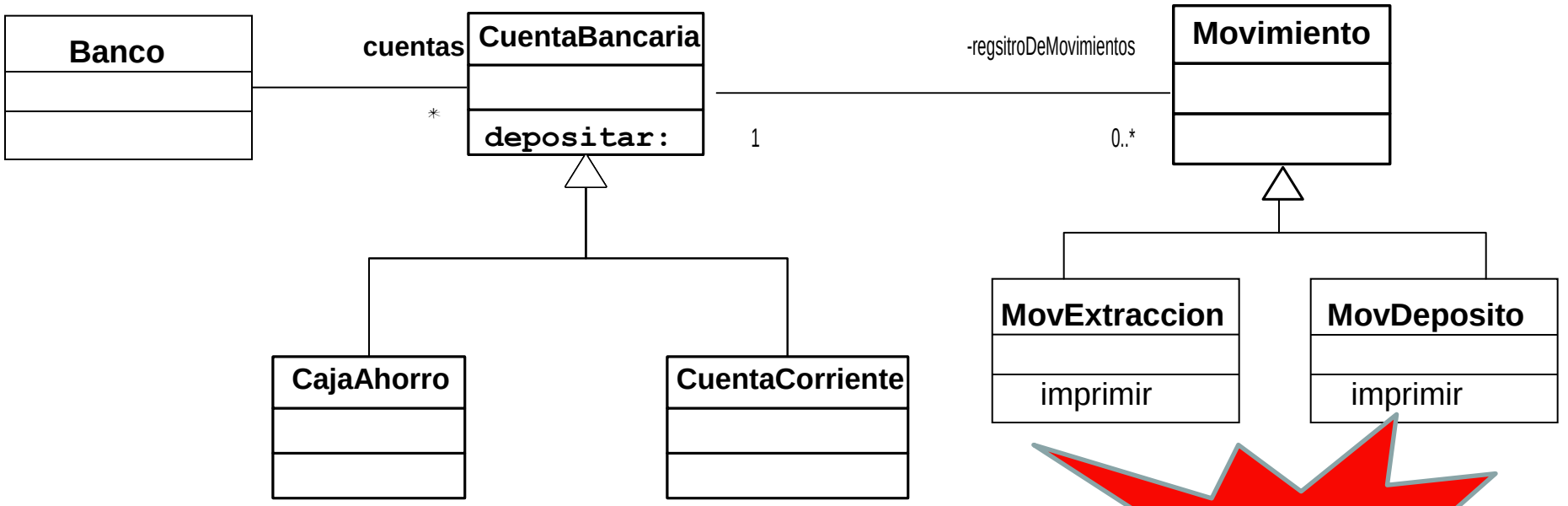



```
CuentaBancaria>>inicializar:unNumero para:unaPersona
        self numeroCuenta: unNumero.
        self titular: unaPersona.
        self saldo: 0.
        registroDeMovimientos:= OrderedCollection new.
```

Registrar Movimientos de Extracción y Depósito

- Supongamos que queremos distinguir la forma en que se imprime cada tipo de movimiento
- Crear subclases de la clase **Movimiento**
  - Cada subclase de **Movimiento** se imprime a su manera
  - Cada objeto **CuentaBancaria** conoce una colección de movimientos heterogénea.
  - En cada transacción se crea el movimiento correspondiente
  - El movimiento se agrega al **registroDeMovimientos**

Registrar Movimientos de Extracción y Depósito



No usamos v.i. tipo.

```
CuentaBancaria>>depositar: unMonto

        self incrementarSaldo: unMonto.

        self registroDeMovimientos add: (MovDeposito new inicializar:Date
                                                today monto:unMonto)
```

Emitir resumen de todas las cuentas del banco

```
Banco>> emitirResumen
self cuentas do:[:c| c emitirResumen]
```

```
CuentaBancaria>>emitirResumen
self registroDeMovimientos do:[:m| m imprimir]
```

- De qué clase son los objetos de la colección cuentas?
- De que clase son los objetos de la colección registroDeMovimientos?
- Donde está presente el polimorfismo?

Retornar el último movimiento realizado en una cuenta

```
CuentaBancaria>>ultimoMovimiento
^(self registroDeMovimientos asSortedCollection:[:m1 :m2
|m1 fecha > m2 fecha]) first
```

Reuso de Código
Herencia vs. Composición

Herencia de Clases

- Herencia total: debo conocer todo el código que se hereda -> Reutilización de Caja Blanca
- Usualmente debemos redefinir
- Los cambios en la superclase se propagan automáticamente a las subclases
- Herencia de Estructura vs. Herencia de comportamiento
- Es útil para extender la funcionalidad del dominio de aplicación

Composición de Objetos

- Los objetos se componen en forma Dinámica -> Reutilización de Caja Negra
- Los objetos pueden reutilizarse a través de su interfaz (sin conocer el código)
- A través de las relaciones de composición se pueden delegar responsabilidades entre los objetos

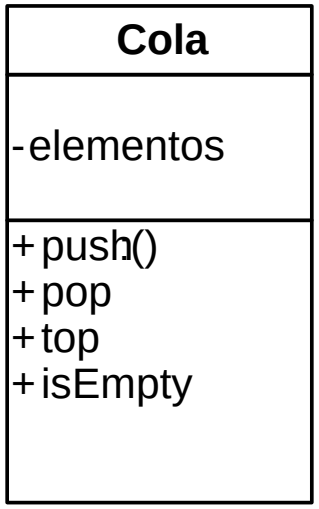
Un simple ejemplo: Clase Cola

- Cola es una estructura de datos con comportamiento específico.
- Implementaría Cola como subclase de OrderedCollection?
- Tiene sentido heredar todo el comportamiento de OrderedCollection?
- Habria que anular o redefinir demasiado comportamiento?
- Cola se compone de una OrderedCollection para mantener sus elementos?

Un simple ejemplo: Clase Cola

Cola como subclase de Object

```
Cola>>initialize

elementos := OrderedCollection new.

Cola>> push: unObjeto
       elementos addLast: unObjeto

Cola>> pop
      ^ elementos removeFirst

Cola>> top
       ^ elementos first

Cola>> isEmpty
       ^ elementos isEmpty
```



Un simple ejemplo: ColaDoble y Pila

- Una ColaDoble es una secuencia de elementos a la que se puede agregar y sacar elementos por ambos extremos.
- Como implementaría la clase ColaDoble usando la clase Cola?
- Como implementaría la clase Pila usando ColaDoble?

- Ejercicio para la casa….