

# **Reuso de Código**

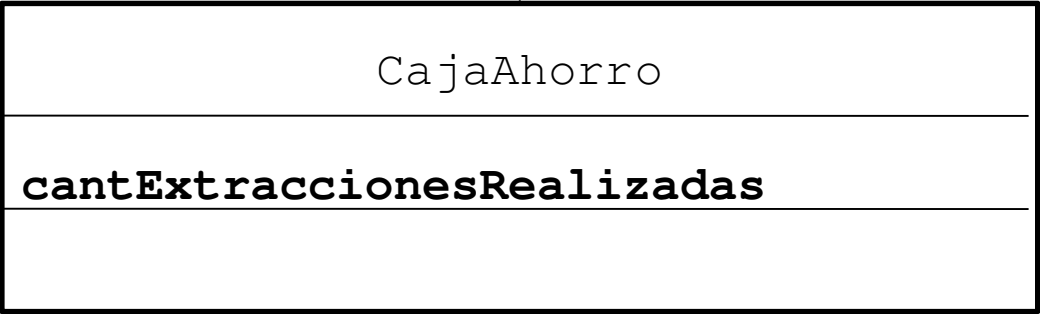
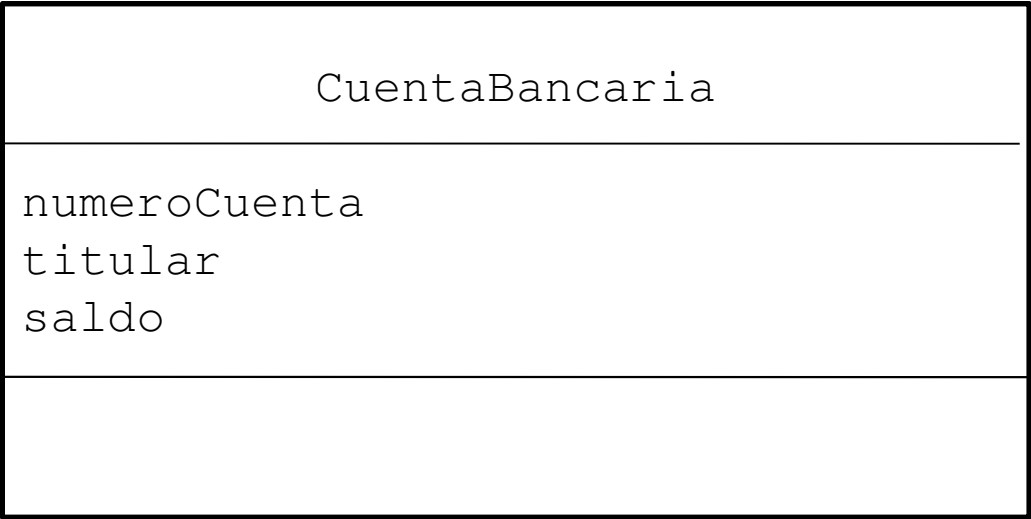
## **Abstracción de comportamiento en método de la superclase**



## #extraer: en la clase Cuenta Bancaria

- Tener en cuenta:
  - chequear que haya saldo suficiente
  - Las cajas de ahorro cuentan las cantidades de extracciones
  - Las cuentas corrientes tienen un monto en descubierto permitido.





Orientad

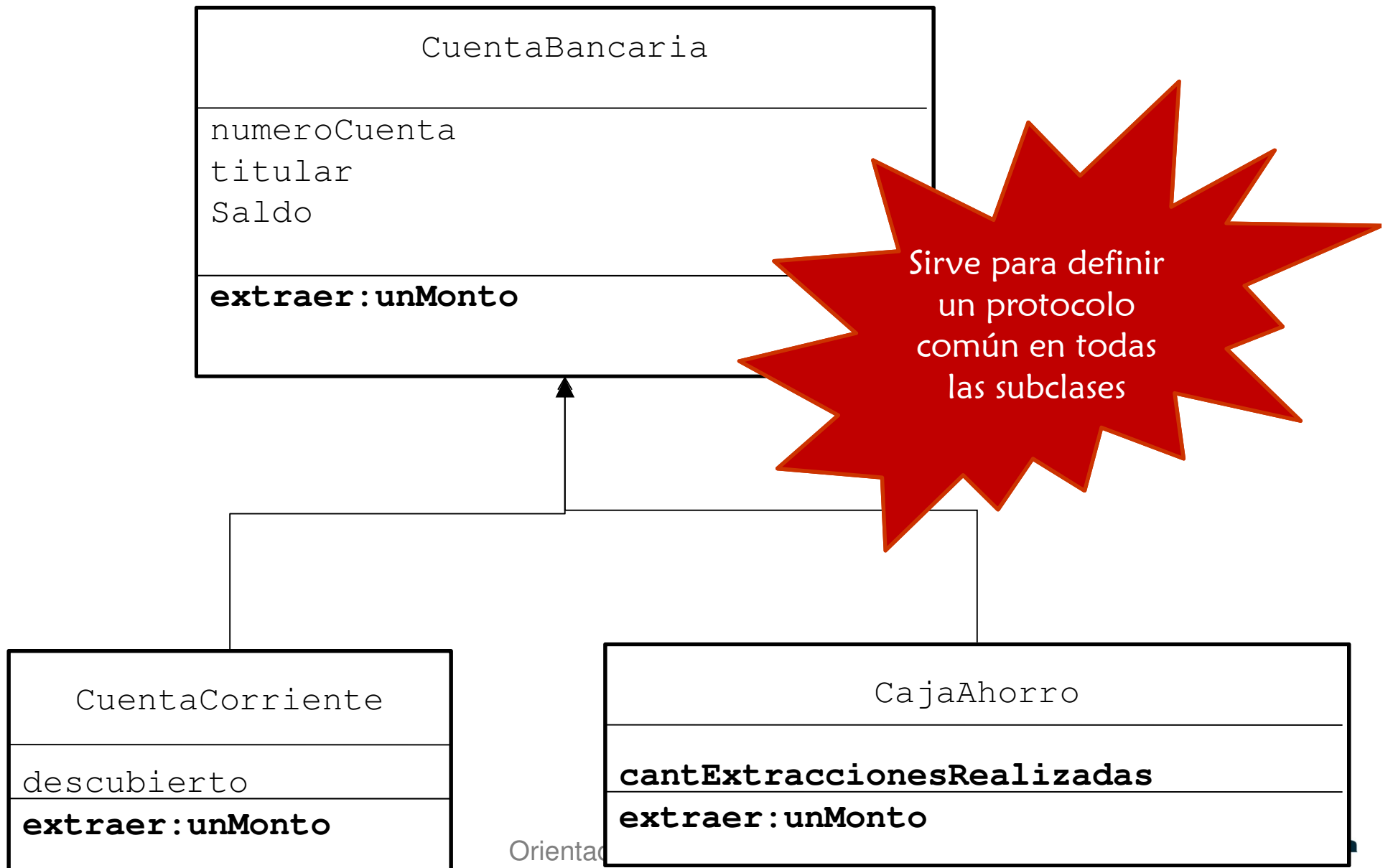


## Ejemplo: Implementación de #extraer:

*Opción 1:* El método en la superclase puede estar **vacío**, indicando que debe implementarse en las subclases.



## Opción 1: #extraer vacío en la superclase



## Opción 1: #extraer vacío en la superclase

**CuentaBancaria>> extraer: unMonto**

^self subclassResponsibility

**CajaAhorro>> extraer: unMonto**

self *hayFondos*: unMonto

if True: [ self *incrementarExtr.*

self *decrementarSaldo*: unMonto]

**CuentaCorriente>> extraer: unMonto**

self *hayFondos*: unMonto

if True: [ self *decrementarSaldo*: unMonto]

**CajaAhorro>> hayFondos: unMonto**

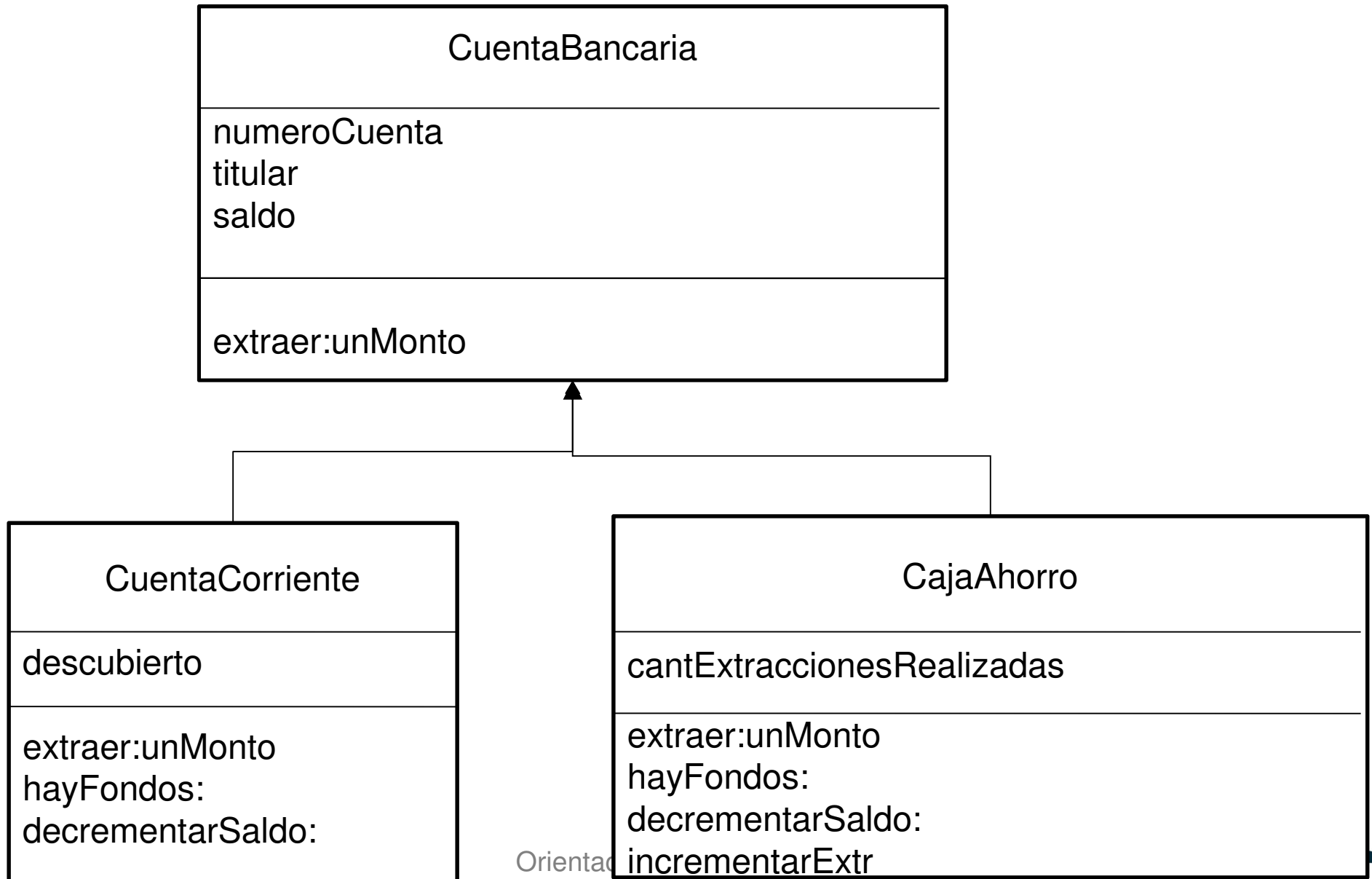
^saldo >= unMonto

**CuentaCorriente>> hayFondos: unMonto**

^saldo + descubierto >= unMonto



## Opción 1: #extraer vacío en la superclase



# Ejemplo: Implementación de #extraer:

Opción 2: Podemos reusar código en común usando *super*





Teníamos.....

**CuentaBancaria>> extraer: unMonto**

^self subClassResponsability

**CajaAhorro>> extraer: unMonto**

self *hayFondos*: unMonto

if True: [ self *incrementarExtr.*

self *decrementarSaldo*: unMonto

**CuentaCorriente>> extraer: unMonto**

self *hayFondos*: unMonto

if True: [ self *decrementarSaldo*: unMonto]

**CajaAhorro>> *decrementarSaldo* : unMonto**

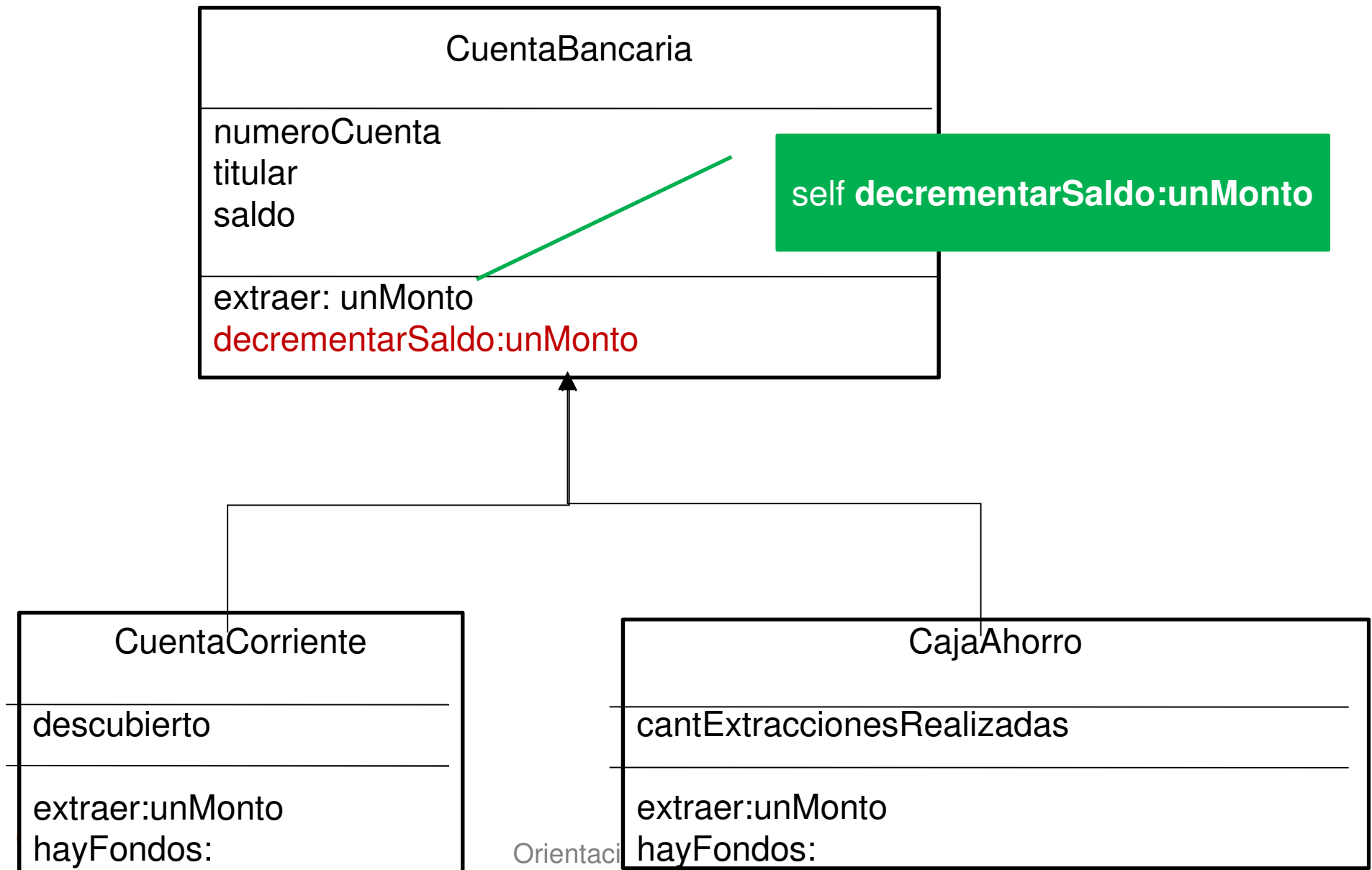
^saldo:= saldo - unMonto

**CuentaCorriente>> *decrementarSaldo* : unMonto**

^saldo:= saldo - unMonto



Opción 2: #extraer redefinido en las subclases (uso de super)



## Opción 2: #extraer redefinido en las subclases (uso de super)

**CuentaBancaria>> extraer: unMonto**  
self decrementarSaldo:unMonto

**CajaAhorro>> extraer: unMonto**  
self hayFondos: unMonto  
if True:[ self incrementarExtr.  
super extraer:unMonto]

**CuentaCorriente>> extraer: unMonto**  
self hayFondos: unMonto  
if True:[super extraer:unMonto]

**CajaAhorro>> hayFondos: unMonto**  
^saldo >= unMonto

**CuentaCorriente>> hayFondos: unMonto**  
^saldo + descubierto>= unMonto



## Ejemplo: Implementación de #extraer:

Opción 3: podemos abstraer en la superclase “la forma o esqueleto” del método e implementar el comportamiento específico en las subclases (uso de *self*)



Teníamos....

**CuentaBancaria>> extraer: unMonto**  
self decrementarSaldo:unMonto

**CajaAhorro>> extraer: unMonto**  
self hayFondos: unMonto  
if True: [self incrementarExtr.  
super extraer:unMonto]

**CuentaCorriente>> extraer: unMonto**  
self hayFondos: unMonto  
if True: [super extraer:unMonto]

Ejecución  
de la  
extracción

**CajaAhorro>> hayFondos: unMonto**  
^saldo >= unMonto

**CuentaCorriente>> hayFondos: unMonto**  
^saldo >= unMonto + descubierto



# Definimos el # ***ejecutarExtraccion:***

**CuentaBancaria>> extraer: unMonto**  
self decrementarSaldo:unMonto

**CajaAhorro>> extraer: unMonto**  
self hayFondos: unMonto  
if True:[ self ***ejecutarExtraccion:*** unMonto]

**CuentaCorriente>> extraer: unMonto**  
self hayFondos: unMonto  
if True:[self ***ejecutarExtraccion:*** unMonto]

Ambos  
metodos  
son  
iguales

**CajaAhorro>> ***ejecutarExtraccion:*** unMonto**

**CuentaCorriente>> ***ejecutarExtraccion:*** unMonto**



Implementamos el #extraer: en la superclase

**CuentaBancaria>> extraer: unMonto**

self hayFondos: unMonto

if True: [ self ejecutarExtraccion: unMonto]

**CajaAhorro>> ejecutarExtraccion: unMonto**

self incrementarExtr.

self decrementarSaldo: unMonto

**CuentaCorriente>> ejecutarExtraccion: unMonto**

self decrementarSaldo: unMonto

**CajaAhorro>> hayFondos: unMonto**

^saldo >= unMonto

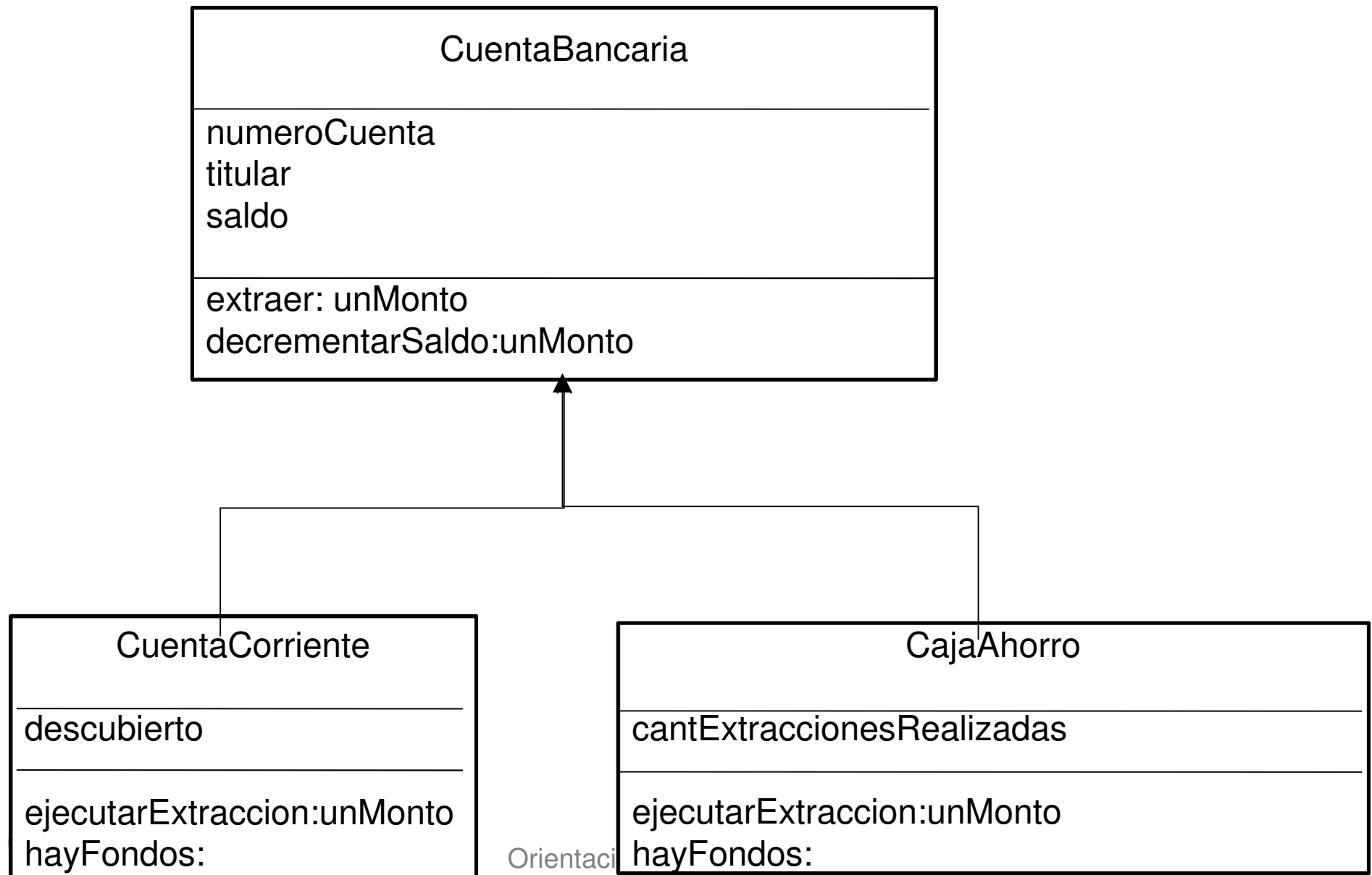
**CuentaCorriente>> hayFondos: unMonto**

^saldo + descubierto >= unMonto

No hace falta  
implementar el  
#extraer: en las  
subclases



Opción 2: #extraer redefinido en las subclases (uso de super)





# Otros ejemplos de la Opción 3: #between: and:

Magnitude|Kernel

- ExactFloatPrintPolicy
- FloatPrintPolicy
- InexactFloatPrintPolicy
- Magnitude**
- Number
- Float
- BoxedFloat64

-- all --

- comparing
- hash
- testing

between:and:

compareWith:ifLesser:ifEqual:ifGrea

hash

max

**between: min and: max**

"Answer whether the receiver is less than or equal to the argument, max, and greater than or equal to the argument, min."

```
^self >= min and: [self <= max]
```

Scoped Variables

Magnitude|Kernel

- Number
- Float
- BoxedFloat64
- SmallFloat64
- Fraction**
- ScaledDecimal
- Integer

-- all --

- arithmetic
- comparing
- converting
- mathematical fi
- printing
- private
- self evaluating
- testing

<= aNumber

```
aNumber isFraction ifTrue:  
  [^ numerator * aNumber denominator <= (aNumber nume  
denominator)].  
^ aNumber adaptToFraction: self andCompare: #<=
```

Scoped Variables

Magnitude|Kernel

- Number
- Float
- BoxedFloat64
- SmallFloat64
- Fraction
- ScaledDecimal
- Integer**

-- all --

- accessing
- arithmetic
- benchmarks
- bit manipulation
- comparing
- converting
- converting-arra
- enumerating

<= aNumber

```
aNumber isInteger ifTrue:  
  [self negative == aNumber negative  
    ifTrue: [self negative  
      ifTrue: [^ (self digitCompare: aNumber) >= 0]  
      ifFalse: [^ (self digitCompare: aNumber) <= 0]]  
    ifFalse: [^ self negative]].  
^ aNumber adaptToInteger: self andCompare: #<=
```

# Otros ejemplos de la Opción 3: #select:

The screenshot shows the Scala REPL interface with the `Collection>>#select:` command. The left pane shows the package hierarchy with `Collections-Abstract` selected. The right pane shows a list of methods including `select`. The main text area displays the documentation for `select`:

```
select: aBlock
  "Evaluate aBlock with each of the receiver's elements as the argument.
  Collect into a new collection like the receiver, only those elements for
  which aBlock evaluates to true. Answer the new collection."

  | newCollection |
  newCollection := self copyEmpty.
  self do: [ :each |
    (aBlock value: each)
    ifTrue: [ newCollection add: each ]].
  ^newCollection
```

The screenshot shows the Scala REPL interface with the `Collection>>#do:` command. The left pane shows the package hierarchy with `Collections-Abstract` selected. The right pane shows a list of methods including `do`. The main text area displays the documentation for `do`:

```
do: aBlock
  "Evaluate aBlock with each of the receiver's elements as the argument."

  self subclassResponsibility
```



# Otros ejemplos de la Opción 3: #select:

The screenshot shows the Smalltalk IDE interface. On the left, the 'Collections-Sequenceable' class is selected in the browser. The middle pane shows the class hierarchy with 'ManifestCollectionsSequenceable' as the superclass. The right pane shows the 'do:' method. The main editor displays the following code:

```
do: aBlock
    "Override the superclass for performance reasons."

    firstIndex to: lastIndex do: [ :index |
        aBlock value: (array at: index) ]
```

The screenshot shows the Smalltalk IDE interface. On the left, the 'Collections-Unordered' class is selected in the browser. The middle pane shows the class hierarchy with 'Bag' as the superclass. The right pane shows the 'do:' method. The main editor displays the following code:

```
do: aBlock
    "Evaluate aBlock with each of the receiver's elements as the argument."

    contents associationsDo: [:assoc | assoc value timesRepeat: [aBlock value: assoc key]]
```

