Programación III

Clase VIII

Andrés Fortier

La pregunta de la Rhodesia

- Defina qué es un constructor y para que se utiliza.

Repaso - Constructores

- Son mensajes **de clase** relacionados a la construcción de objetos bien formados.
- Podemos crear constructores que tomen como parámetros los colaboradores básicos del nuevo objeto
  - Así, creamos objetos bien formados.
  - Un objeto bien formado se encuentra en estado consistente desde su creación.

Repaso - Ejemplo

```
Fecha class >> dia: unDia mes: unMes año: unAño

    | fecha |

    fecha := self new.
    fecha dia: unDia.
    fecha mes: unMes.
    fecha año: unAño.
    ^fecha.
```

```
| hoy |

hoy := Fecha dia: 25 mes: 8 año: 2014.
hoy dia. "printIt => 25"
```

Repaso - Más de un constructor

```
CuentaBancaria class >> titular: unaPersona saldo: unSaldo
    | cuenta |

    cuenta := self new.
    cuenta titular: unaPersona.
    cuenta saldo: unSaldo.
    ^cuenta.
```

```
CuentaBancaria class >> titular: unaPersona

    ^self titular: unaPersona saldo: 0.
```

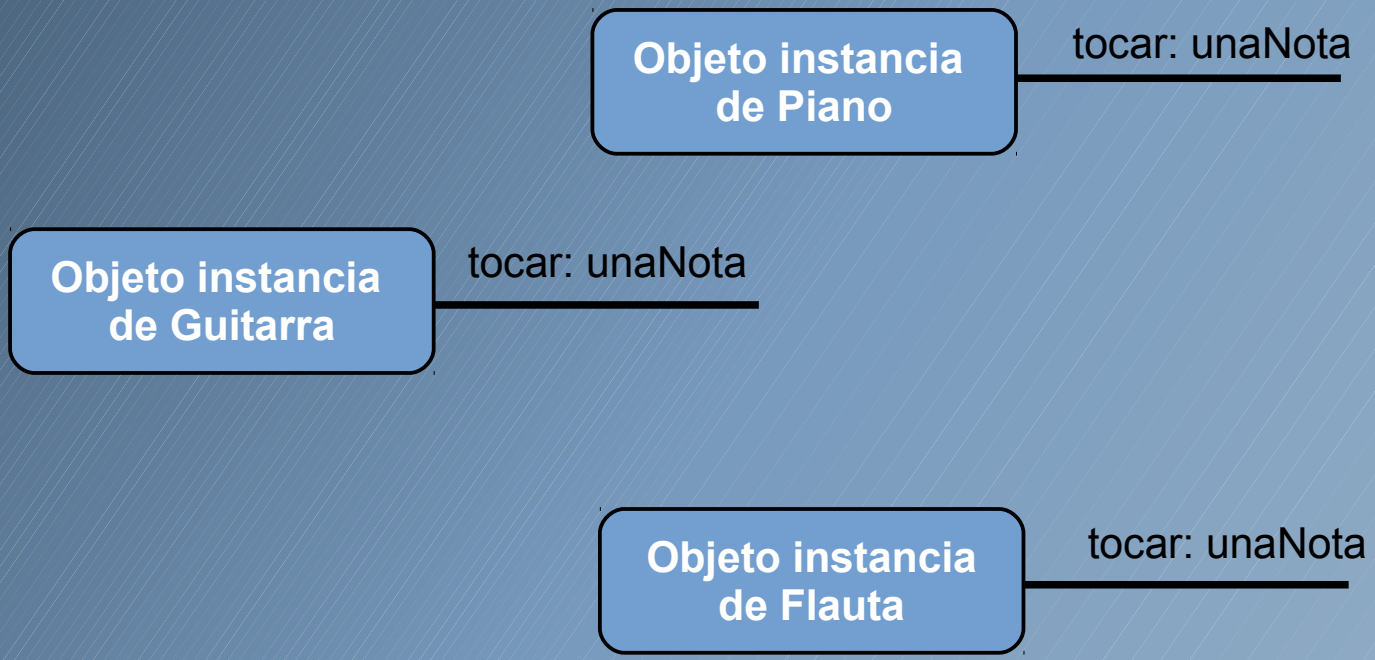
```
| pepe juan cuentaPepe cuentaJuan |

pepe := …
juan := …
cuentaPepe := CuentaBancaria titular: pepe saldo: 1000.
cuentaJuan := CuentaBancaria titular: juan.
```

Polimorfismo

- Dos o más objetos son polimórficos con respecto a un mensaje si pueden entender ese mensaje, aún cuando cada uno lo haga de un modo diferente.
  - Mismo mensaje puede ser enviado a diferentes objetos.
  - Distintos receptores reaccionan diferente (diferentes métodos).

Polimorfismo

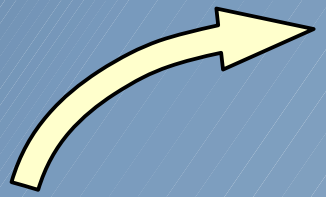


Polimorfismo

```
interpretar: unaPartitura con: unInstrumento

    por cada nota en unaPartitura
        unInstrumento tocar: nota
```

interpretar: unaPartitura con: unInstrumento

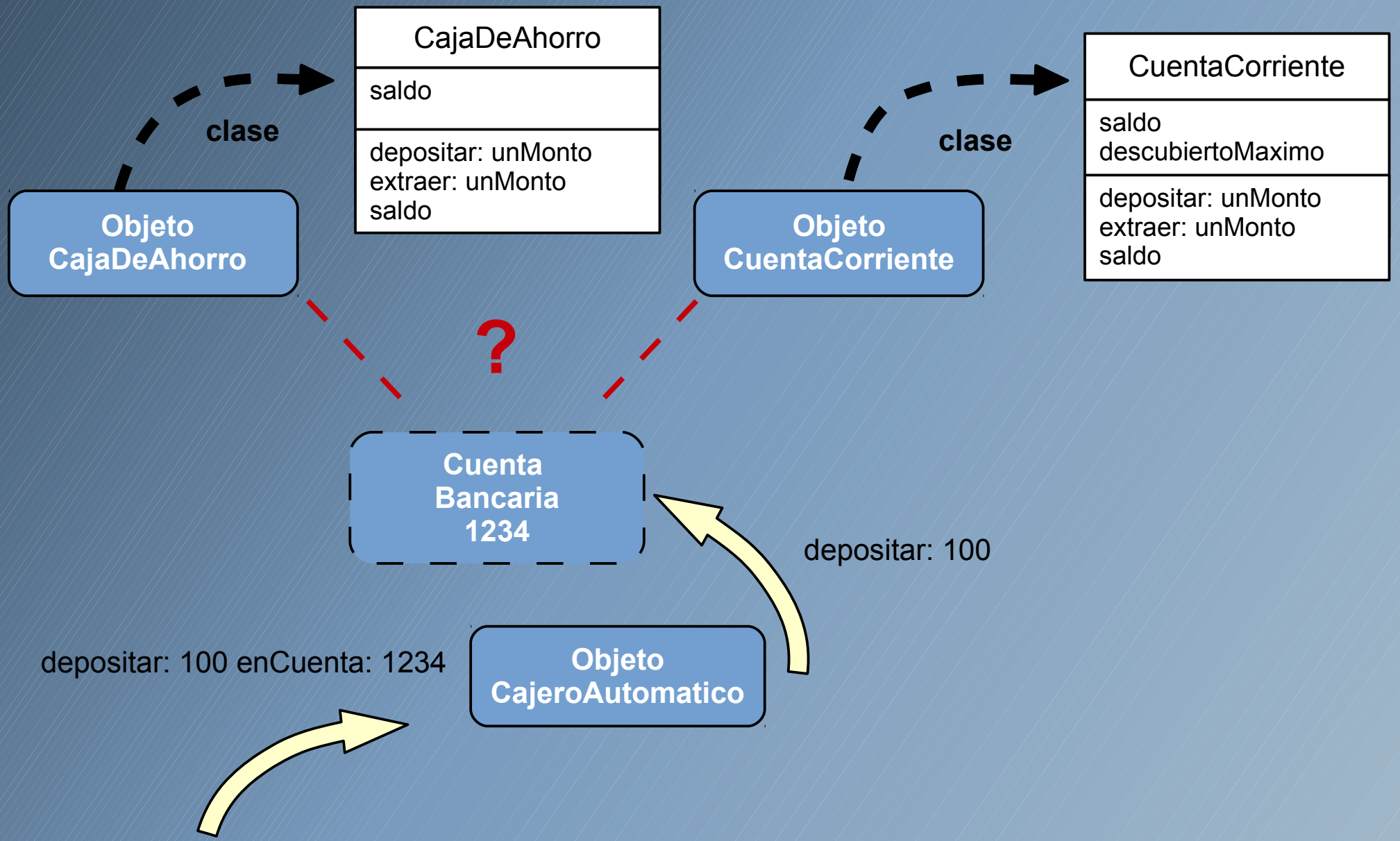


Objeto instancia de InterpreteDePartituras

Cuentas bancarias

- Un banco tiene dos clases de cuentas bancarias:
  - Cajas de ahorro.
  - Cuentas corrientes.
- Ambas cuentas permiten depositar dinero.
- Un usuario accede a su cuenta por medio de un cajero automático.

Cuentas bancarias



Polimorfismo

- Supongamos la siguiente implementación:

```
Cajero >> depositar: unMonto en: unNumeroDeCuenta

   | cuenta |

   cuenta := banco obtenerCuenta: unNumeroDeCuenta.
   cuenta esCajaDeAhorro
      ifTrue:[self depositarEnCajaDeAhorro: unMonto]
      IfFalse:[self depositarEnCuentaCorriente: unMonto].
```

- ¿Pueden identificar algún problema?
- ¿Qué pasa si el día de mañana agregamos una nueva cuenta (ej. SuperCuenta)?

Polimorfismo

- Deberíamos modificar el código

```
Cajero >> depositar: unMonto en: unNumeroDeCuenta

   | cuenta |

   cuenta := banco obtenerCuenta: unNumeroDeCuenta.
   cuenta esCajaDeAhorro
      ifTrue:[self depositarEnCajaDeAhorro: unMonto]
      IfFalse:[cuenta esCuentaCorriente
         ifTrue:[self depositarEnCuentaCorriente: unMonto]
         IfFalse:[self depositarEnSuperCuenta: unMonto]].
```

Polimorfismo

- Si utilizamos objetos polimórficos debemos crear una nueva clase que implemente el mensaje #depositar: unMonto.
- El código del cajero queda inalterado.

```
Cajero >> depositar: unMonto en: unNumeroDeCuenta

   | cuenta |

   cuenta := banco obtenerCuenta: unNumeroDeCuenta.
   cuenta depositar: unMonto.
```

Ventajas del uso de polimorfismo

- Código genérico.
- Objetos desacoplados.
- Objetos intercambiables.
- **Programar por protocolo, no por implementación.**

Volvamos a nuestras cuentas bancarias

CajaDeAhorro
saldo
depositar: unMonto extraer: unMonto saldo

CuentaCorriente
saldo descubiertoMaximo
depositar: unMonto extraer: unMonto saldo

```
depositar: unMonto

    saldo := saldo + unMonto.
```

```
depositar: unMonto

    saldo := saldo + unMonto.
```

```
saldo

    ^saldo.
```

```
saldo

    ^saldo.
```

```
extraer: unMonto

    saldo >= unMonto
        ifTrue:[saldo := saldo - unMonto]
```

```
extraer: unMonto

    saldo + descubiertoMaximo >= unMonto
        ifTrue:[saldo := saldo - unMonto]
```

Comportamiento común

- Ambas clases comparten cosas:
  - Representan conceptualmente una cuenta bancaria.
  - La variable de instancia saldo.
  - La implementación de los métodos #saldo y #depositar: unMonto.

Subclasificación

- Se reúne el comportamiento y la estructura común en una clase, la cual cumplirá el rol de superclase.
- Se conforma una *jerarquía de clases*.
- Luego otras clases pueden cumplir el rol de subclases, heredando ese comportamiento y estructura en común.

Cuentas bancarias

CajaDeAhorro
saldo
depositar: unMonto extraer: unMonto saldo

CuentaCorriente
saldo descubiertoMaximo
depositar: unMonto extraer: unMonto saldo

```
depositar: unMonto

	saldo := saldo + unMonto.
```

```
depositar: unMonto

	saldo := saldo + unMonto.
```

```
saldo

	^saldo.
```

```
saldo

	^saldo.
```

```
extraer: unMonto

	saldo >= unMonto
		ifTrue:[saldo := saldo - unMonto]
```

```
extraer: unMonto

	saldo + descubiertoMaximo >= unMonto
		ifTrue:[saldo := saldo - unMonto]
```

Cuentas bancarias

```
depositar: unMonto

    saldo := saldo + unMonto.
```

CajaDeAhorro
saldo
depositar: unMonto extraer: unMonto saldo

CuentaCorriente
saldo descubiertoMaximo
depositar: unMonto extraer: unMonto saldo

```
saldo

    ^saldo.
```

```
saldo

    ^saldo.
```

```
extraer: unMonto

    saldo >= unMonto
        ifTrue:[saldo := saldo - unMonto]
```

```
extraer: unMonto

    saldo + descubiertoMaximo >= unMonto
        ifTrue:[saldo := saldo - unMonto]
```

Cuentas bancarias

```
depositar: unMonto

        saldo := saldo + unMonto.
```

```
saldo

        ^saldo.
```

CajaDeAhorro
saldo
depositar: unMonto extraer: unMonto saldo

CuentaCorriente
saldo descubiertoMaximo
depositar: unMonto extraer: unMonto saldo

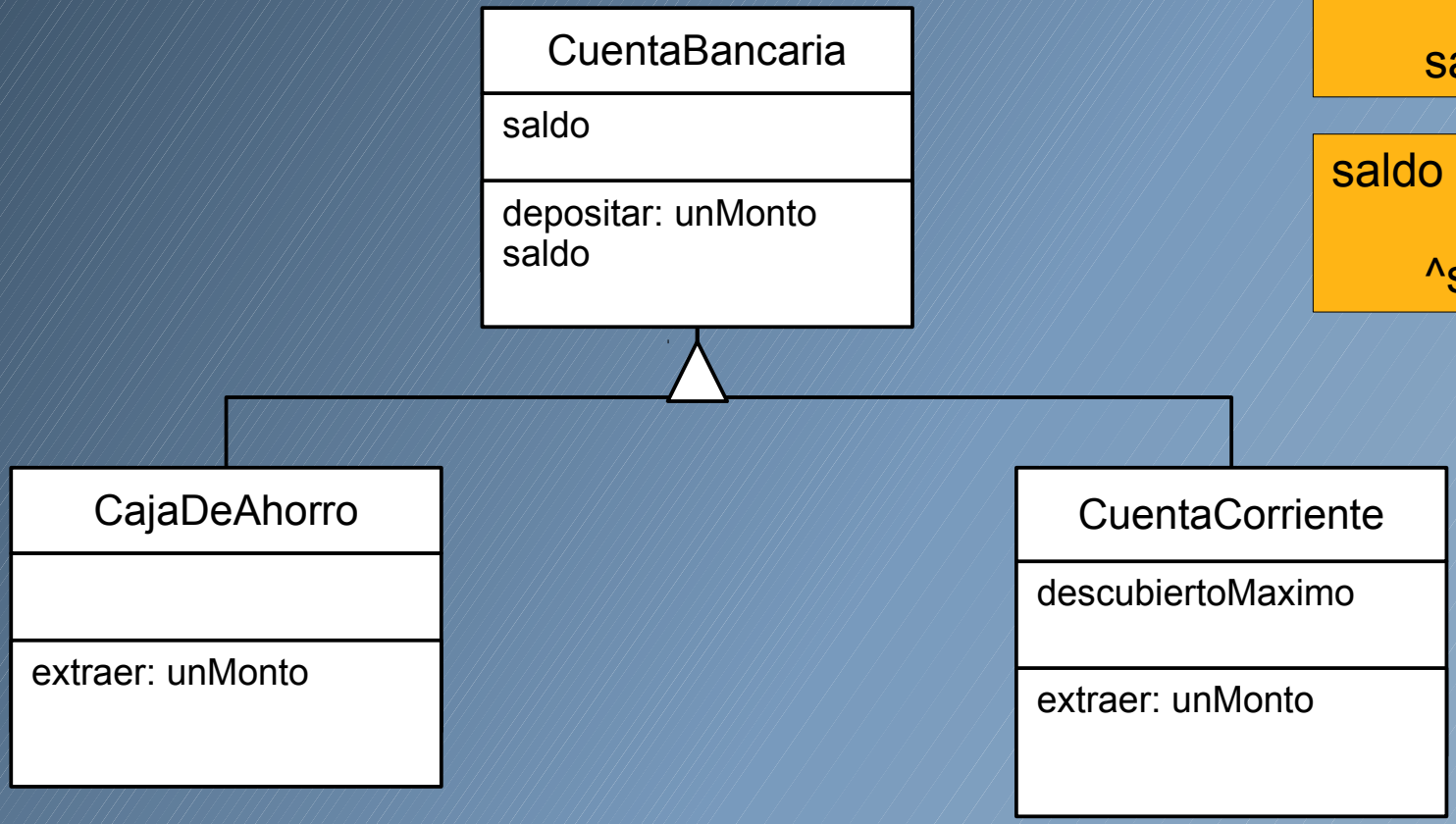
```
extraer: unMonto

        saldo >= unMonto
                ifTrue:[saldo := saldo - unMonto]
```

```
extraer: unMonto

        saldo + descubiertoMaximo >= unMonto
                ifTrue:[saldo := saldo - unMonto]
```

Cuentas bancarias




```
depositar: unMonto

    saldo := saldo + unMonto.
```

```
saldo

    ^saldo.
```

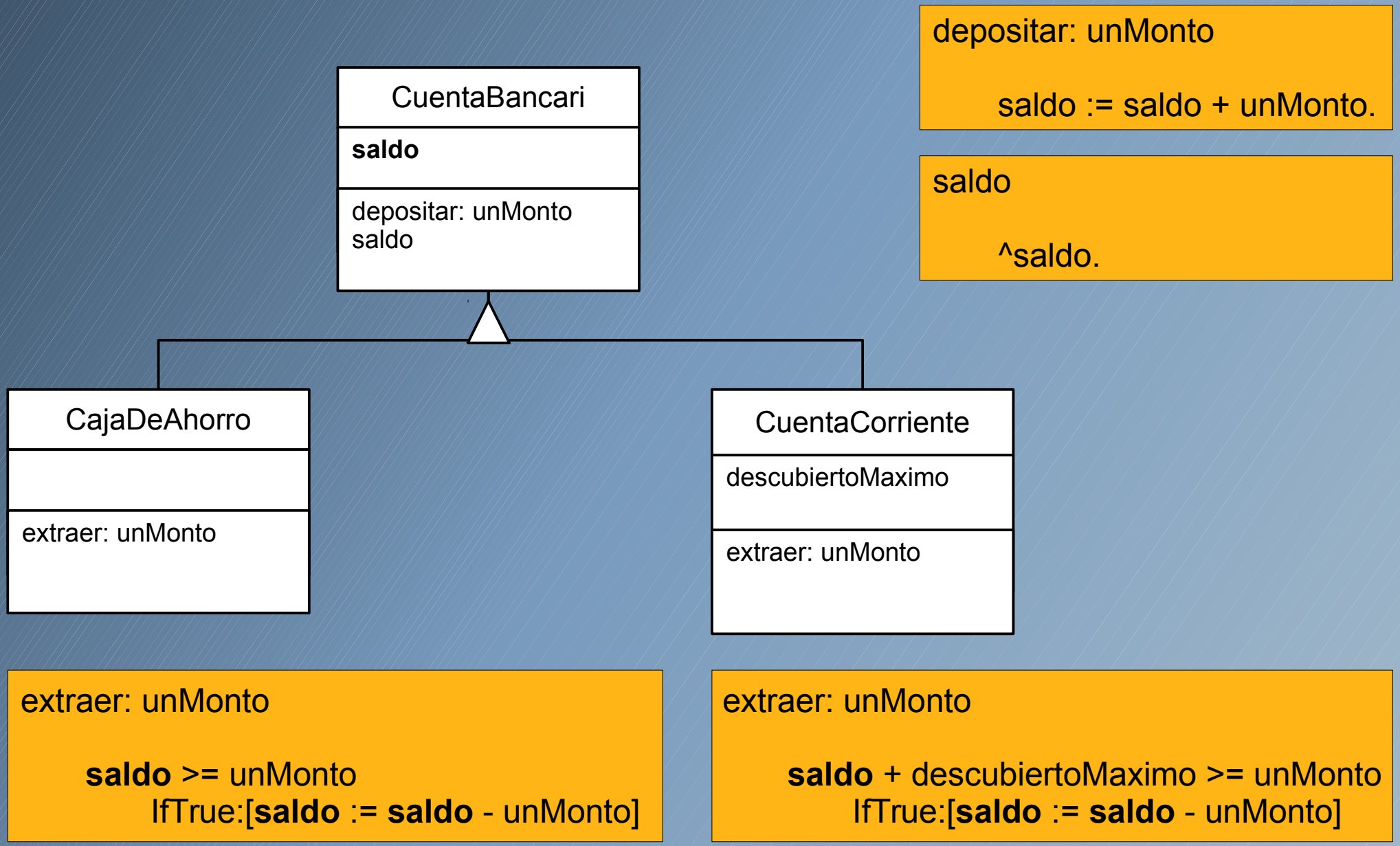
```
extraer: unMonto

    saldo >= unMonto
        ifTrue:[saldo := saldo - unMonto]
```

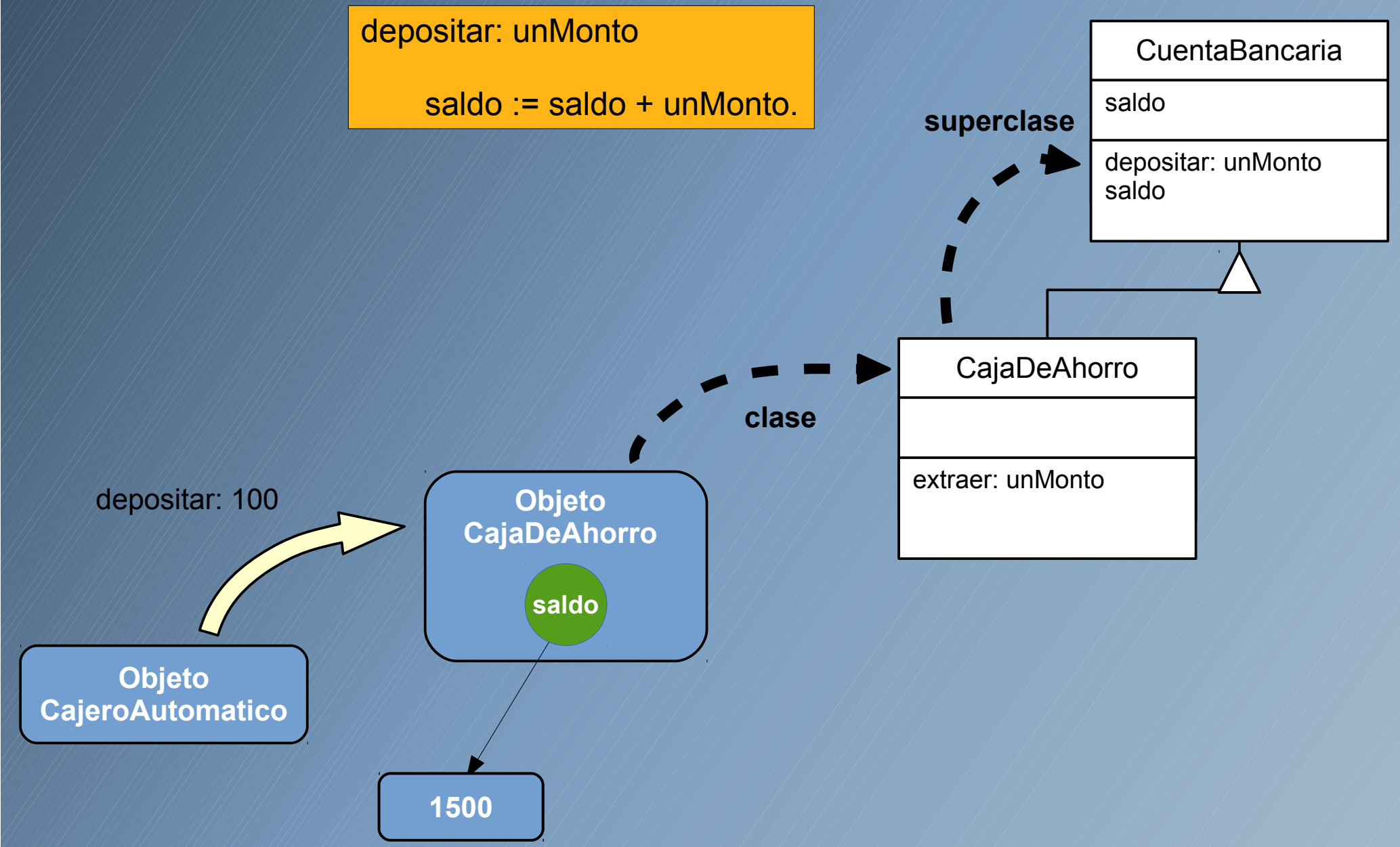
```
extraer: unMonto

    saldo + descubiertoMaximo >= unMonto
        ifTrue:[saldo := saldo - unMonto]
```

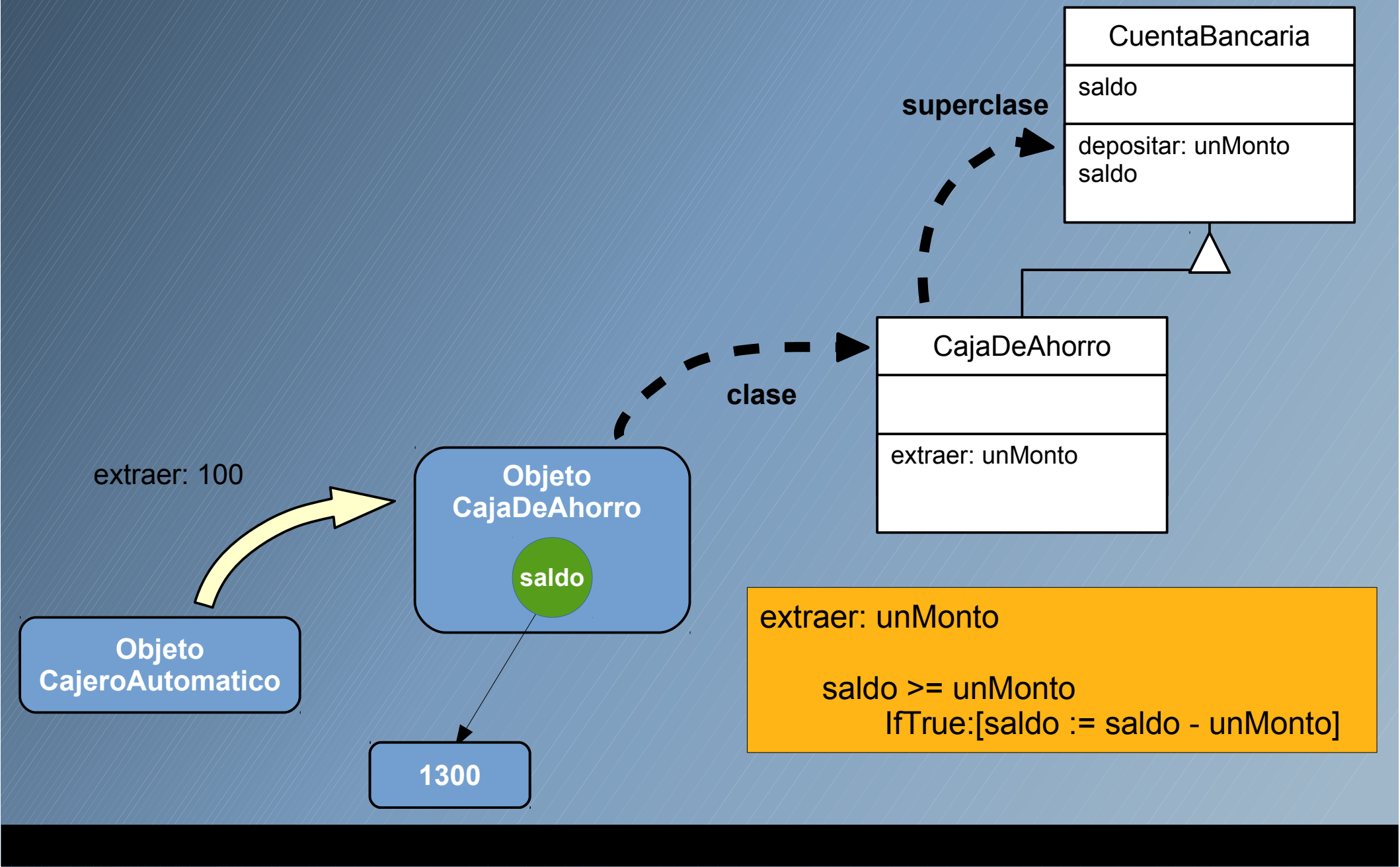
Cuentas bancarias – herencia de v.i.



Envío de mensaje



Envío de mensaje



Herencia de comportamiento

- Ambas cuentas bancarias heredan los métodos
  - #saldo.
  - #depositar: unMonto.
- Cada cuenta implementa el mensaje #extraer: unMonto.
- ¿Cuál es el protocolo de una caja de ahorro?
- ¿Y el de una cuenta corriente?
- Decimos que ambos objetos son polimórficos en su protocolo.