Introducción al Lenguaje de Modelado UML

Prof. Roxana Giandini

LIFIA - Facultad de Informática - UNLP

Diagramas de Interacción

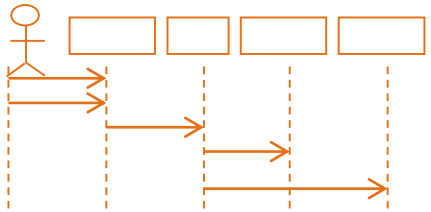
Diagramas de Interacción

Definición:

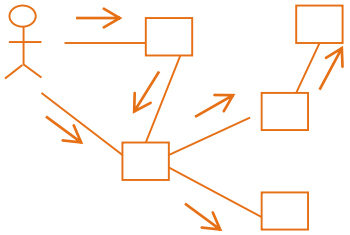
Un diagrama de interacción describe una interacción, que consta de un conjunto de objetos y sus relaciones, incluyendo los mensajes que se pueden enviar, para realizar un comportamiento.

Diagramas de Interacción: tipos

- **Diagramas de Secuencia:**



- **Diagramas de Colaboración:**



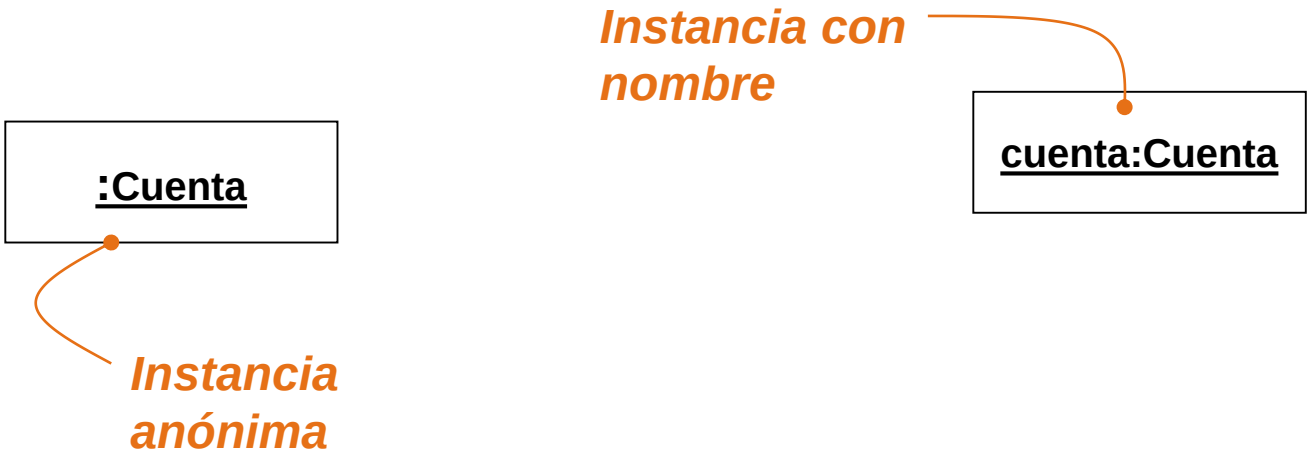
Diagramas de Interacción: contenido

- Objetos
- Enlaces
- Mensajes
  - **Pueden contener:**
    - notas y restricciones.

Diagramas de Interacción: objetos

- Los objetos que participan en una interacción son o bien elementos concretos o bien elementos prototípicos.

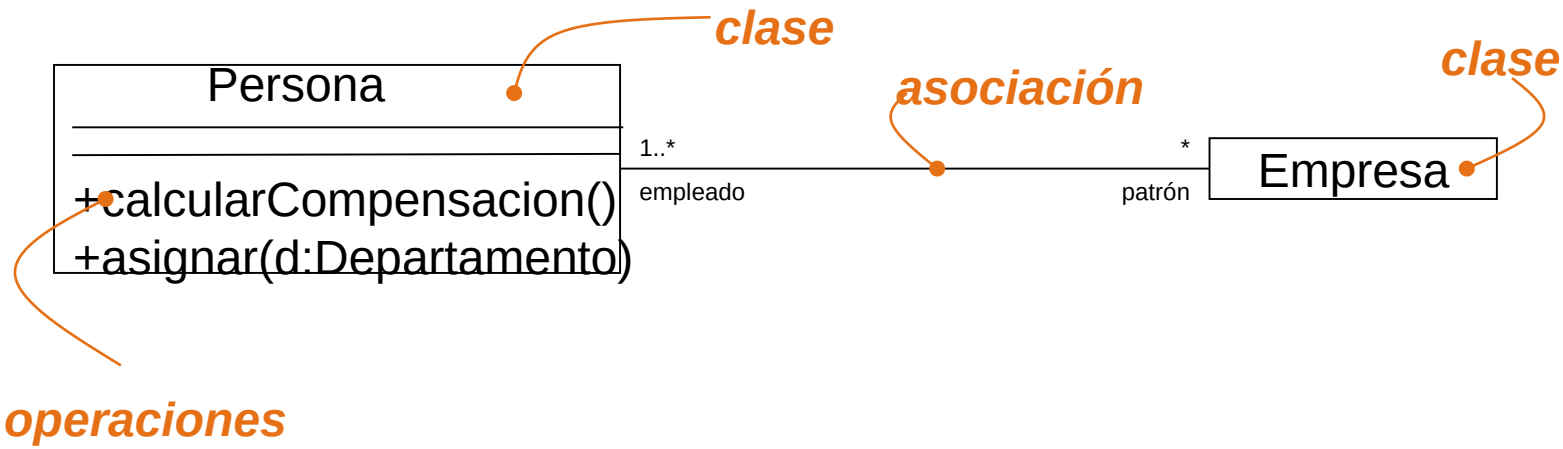
- **Ejemplos**

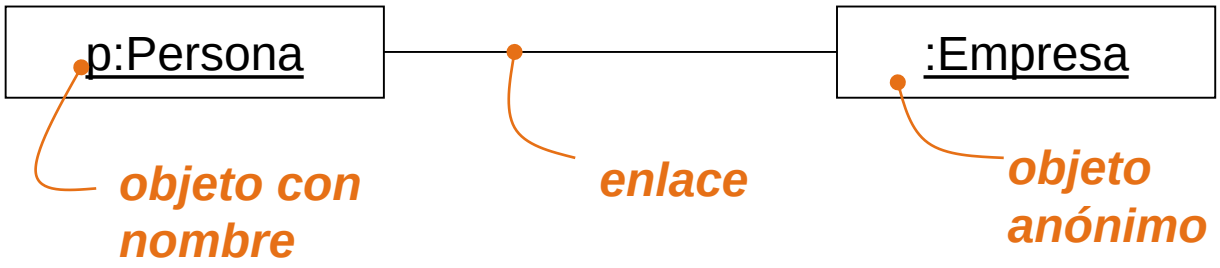


Diagramas de Interacción: enlaces

- Un enlace es una conexión semántica entre objetos.

- **Ejemplos**





Diagramas de Interacción

Diagramas de Secuencia

Diagrama de Secuencia

- **Definición:**

  Un diagrama de secuencia destaca la ordenación temporal de los mensajes.

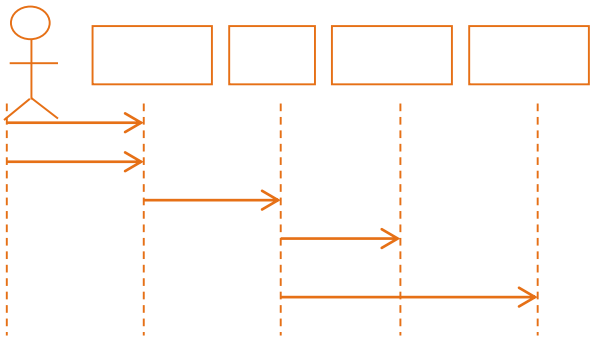
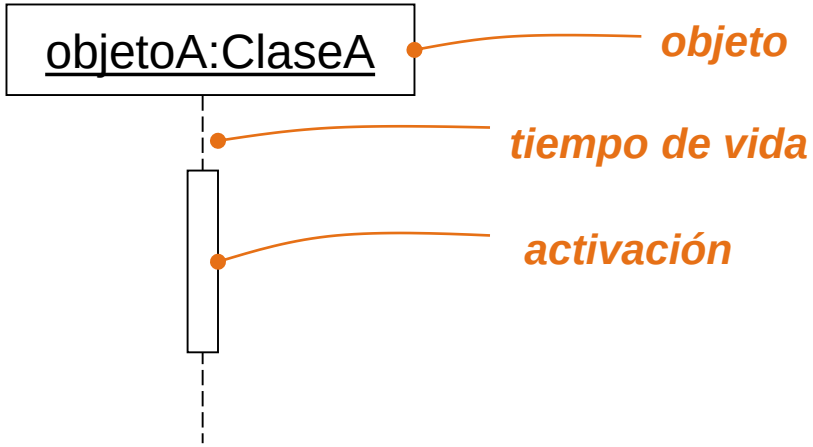


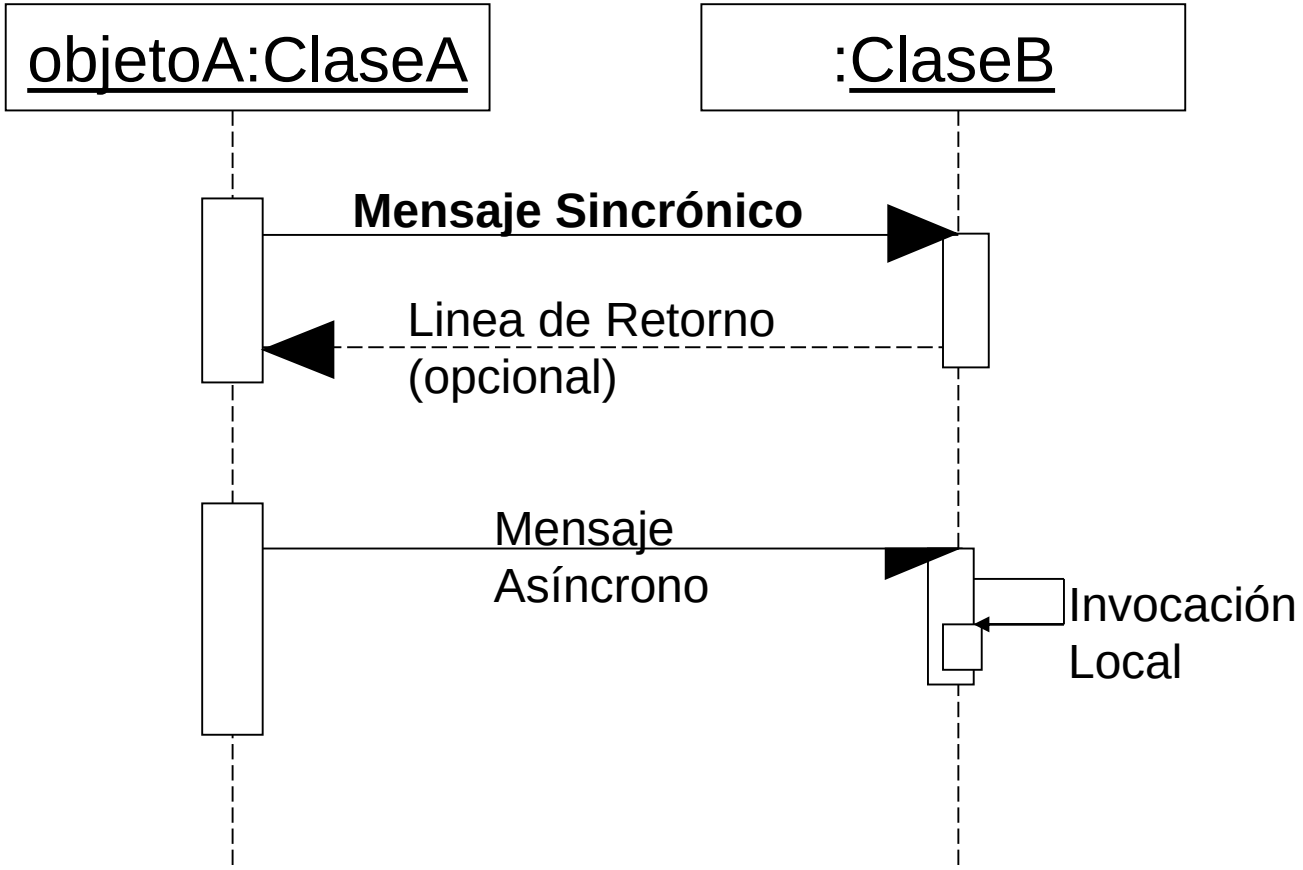
Diagrama de Secuencia

- Cada objeto cuenta con una *línea de vida*, que muestra el tiempo de vida del mismo.
- La *activación* de un objeto representa la ejecución de una operación que realiza el mismo, a través del envío de un *mensaje*.

- **Notación:**



Mensajes


objetoA:ClaseA
:ClaseB
Mensaje Sincrónico
Linea de Retorno
(opcional)
Mensaje
Asíncrono
Invocación
Local

Mensaje sincrónico

- **Sintaxis:**

  [expresión iteración] [valor de retorno :=] nombre del mensaje (parámetros)

- Puede indicarse el fin de su ejecución con una línea punteada del objeto receptor al emisor: Línea Retorno/ Resultado Esta línea es OPCIONAL (en caso que se especifique correctamente la Activación del objeto Receptor).

- El Resultado o Valor de Retorno también puede especificarse a izquierda del nombre del mensaje, en una asignación (ver Sintaxis), en caso que No se haya especificado la línea Retorno.

Diagrama de Secuencia: construcción

- **Primer Paso:** Se colocan los objetos que participan en la interacción en la parte superior del diagrama.

- **Ejemplo**

:Cliente

a:AyudaPlanificación

Diagrama de Secuencia: construcción

- **Segundo Paso:** se colocan los mensajes que estos objetos envían y reciben, en orden de sucesión en el tiempo, desde arriba hasta abajo.
- **Ejemplo**

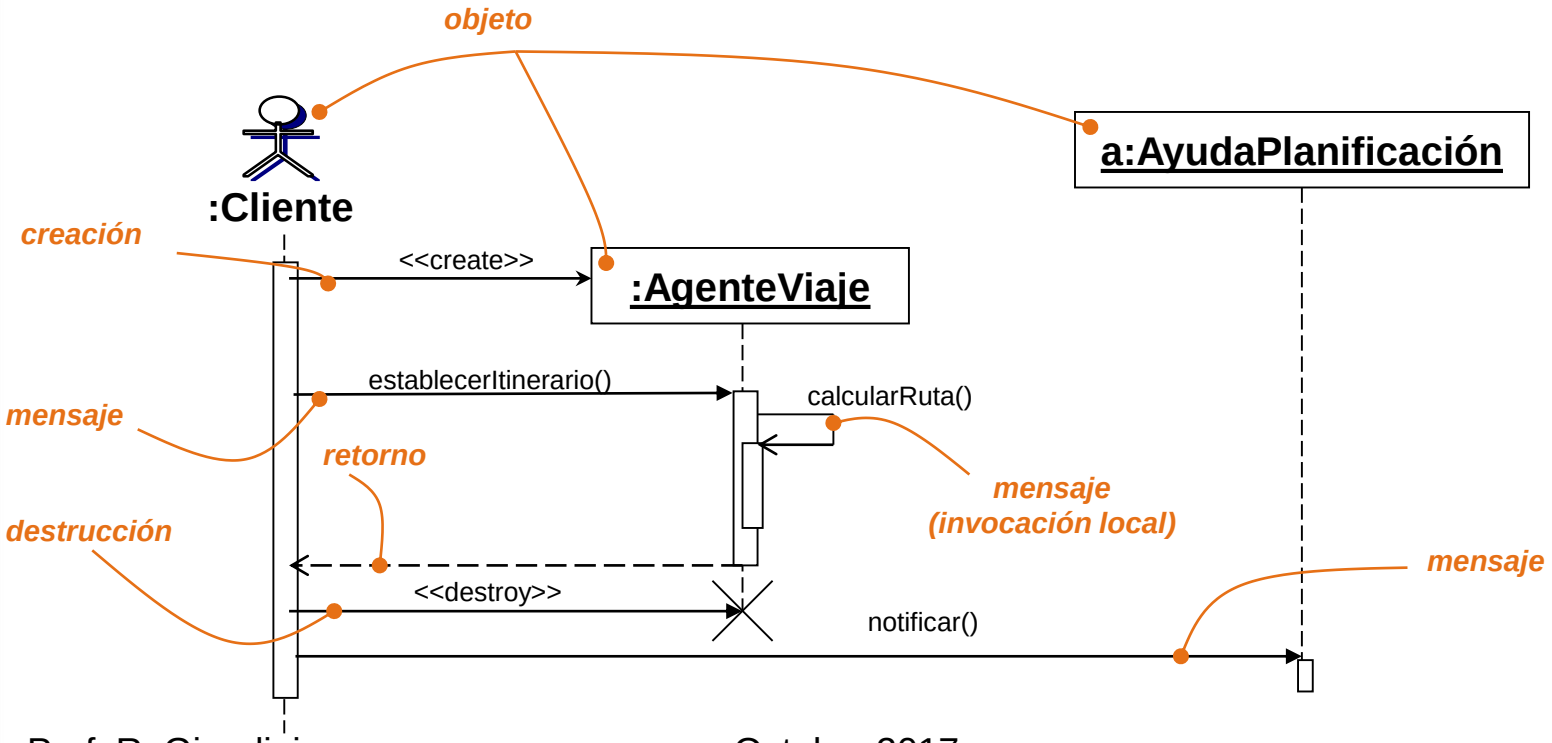


Diagrama de Secuencia: notación

- Se encierra en un rectángulo y se le agrega una etiqueta con **sd** seguido del nombre.

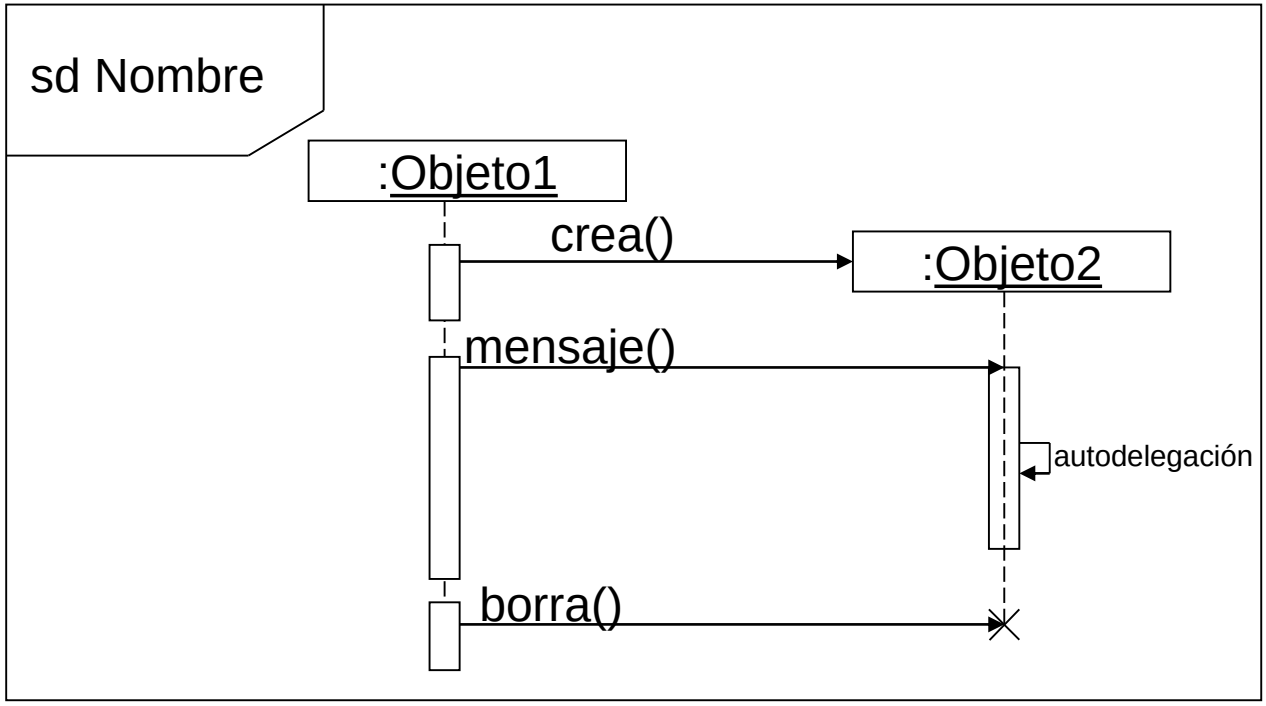


Diagrama de Secuencia: fragmentos (*frames*)

- **Definición:**

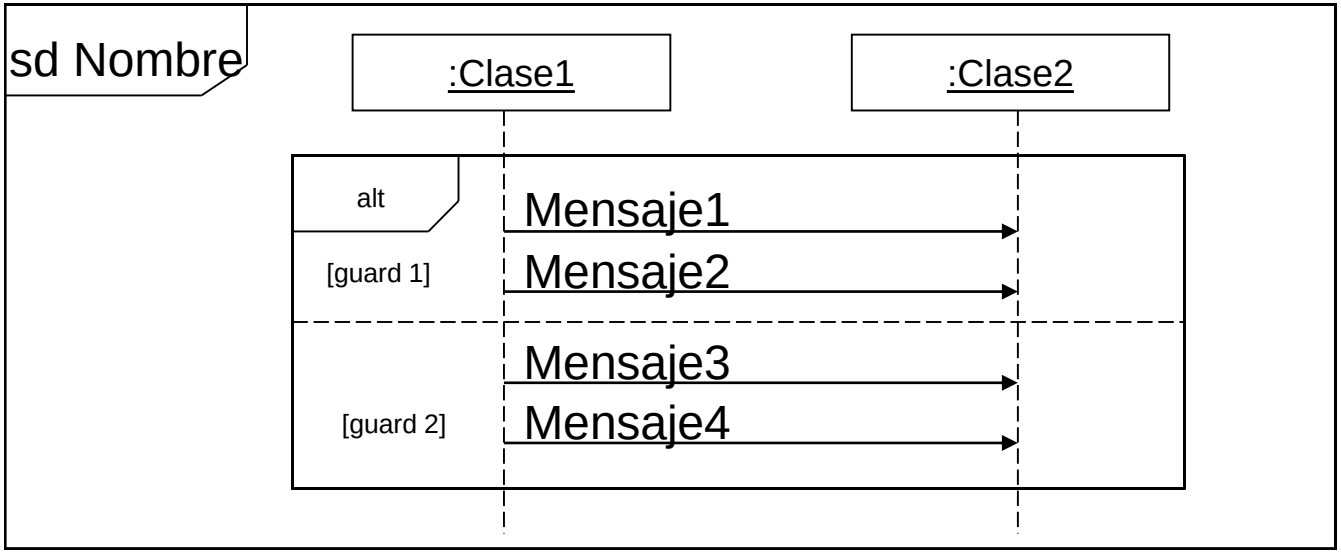
  mecanismo a través del cual se puede realizar la especificación de bloques repetitivos, opcionales, alternativos, entre otros.

- **Operadores más utilizados:**

  - opt alt loop

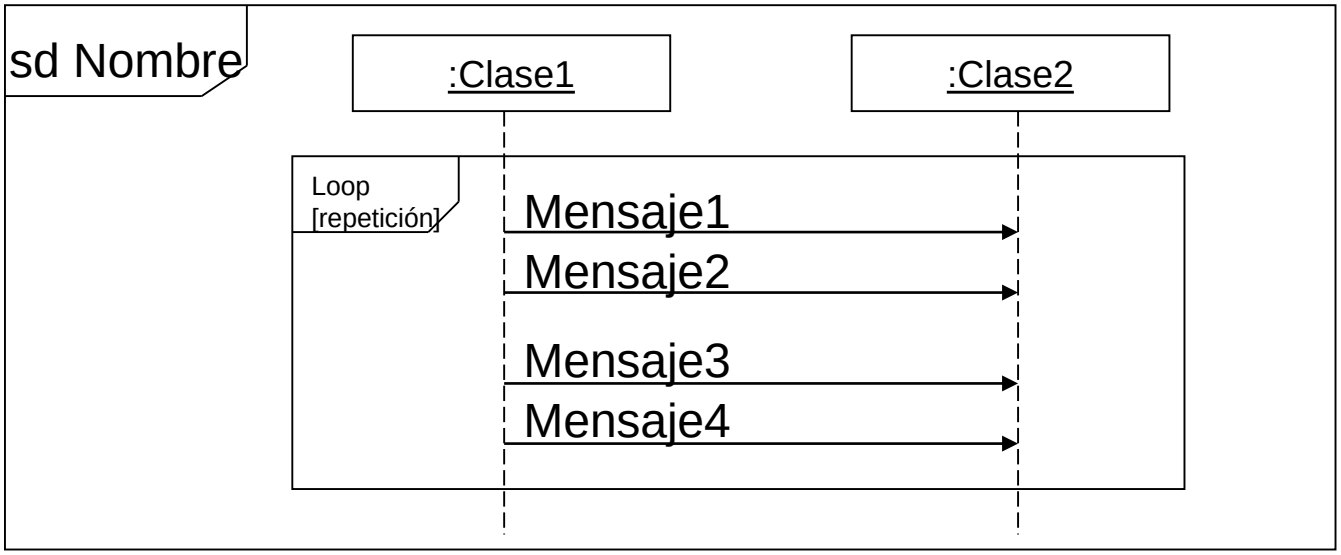
Fragmento: alternativa

- **Notación**: se encierra en un rectángulo (*frame*), se le agrega una etiqueta con el operador **alt** y se colocan las guardas.

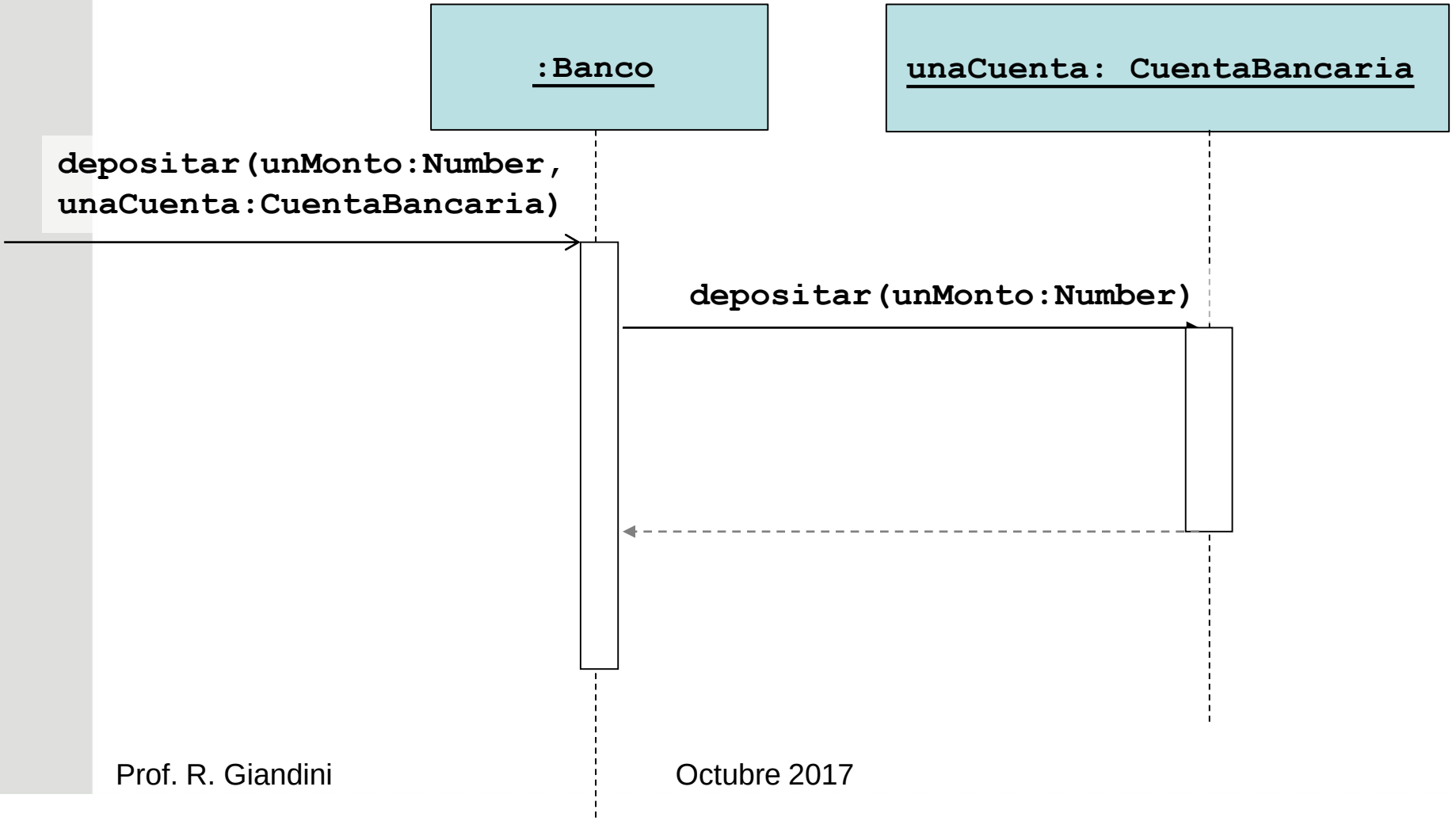


Fragmento: bucle

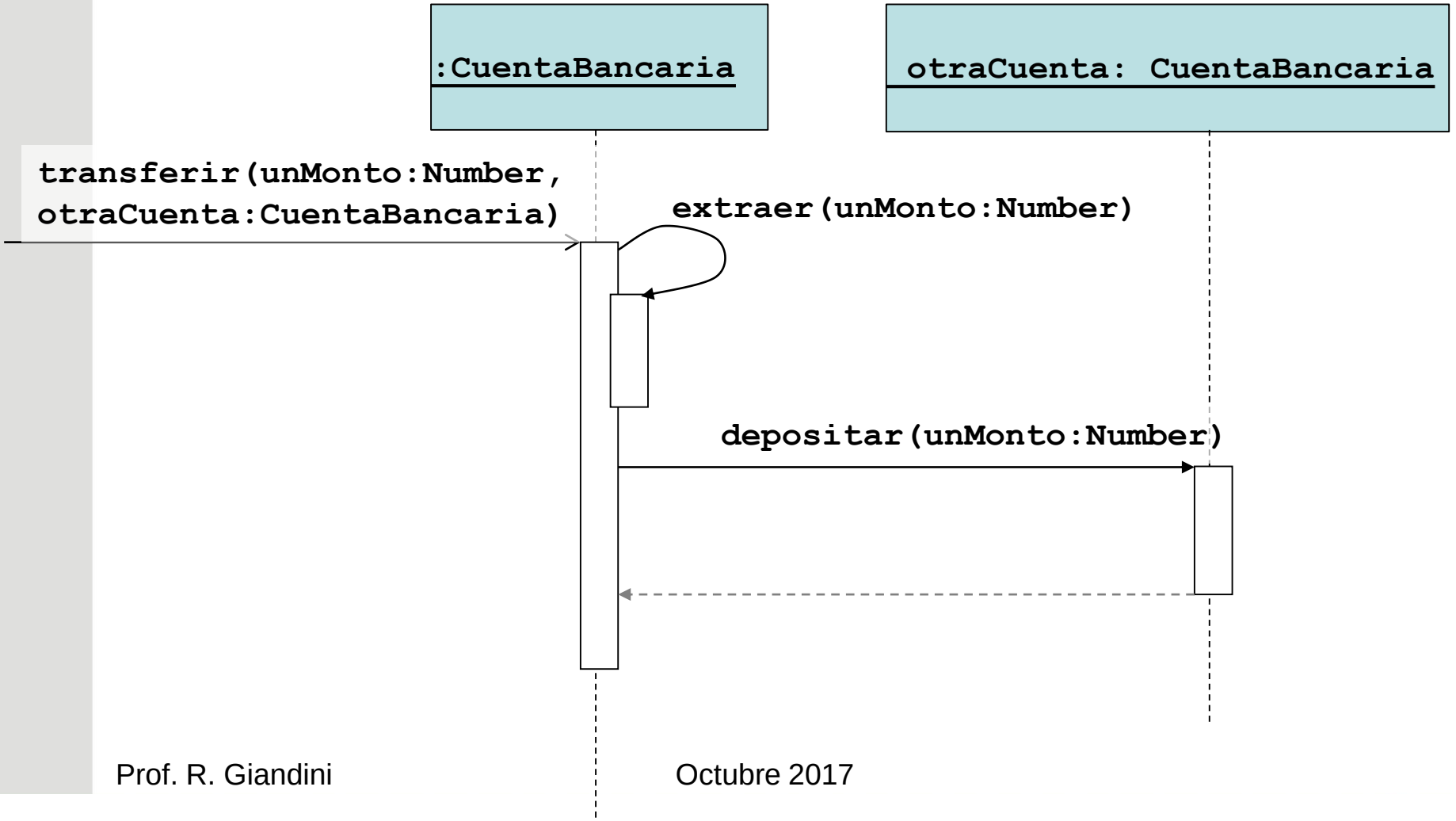
- **Notación**: se encierra en un rectángulo (*frame*), se le agrega una etiqueta con el operador **loop** y la cantidad de iteraciones (opcional).



Ejemplo: Depósito bancario

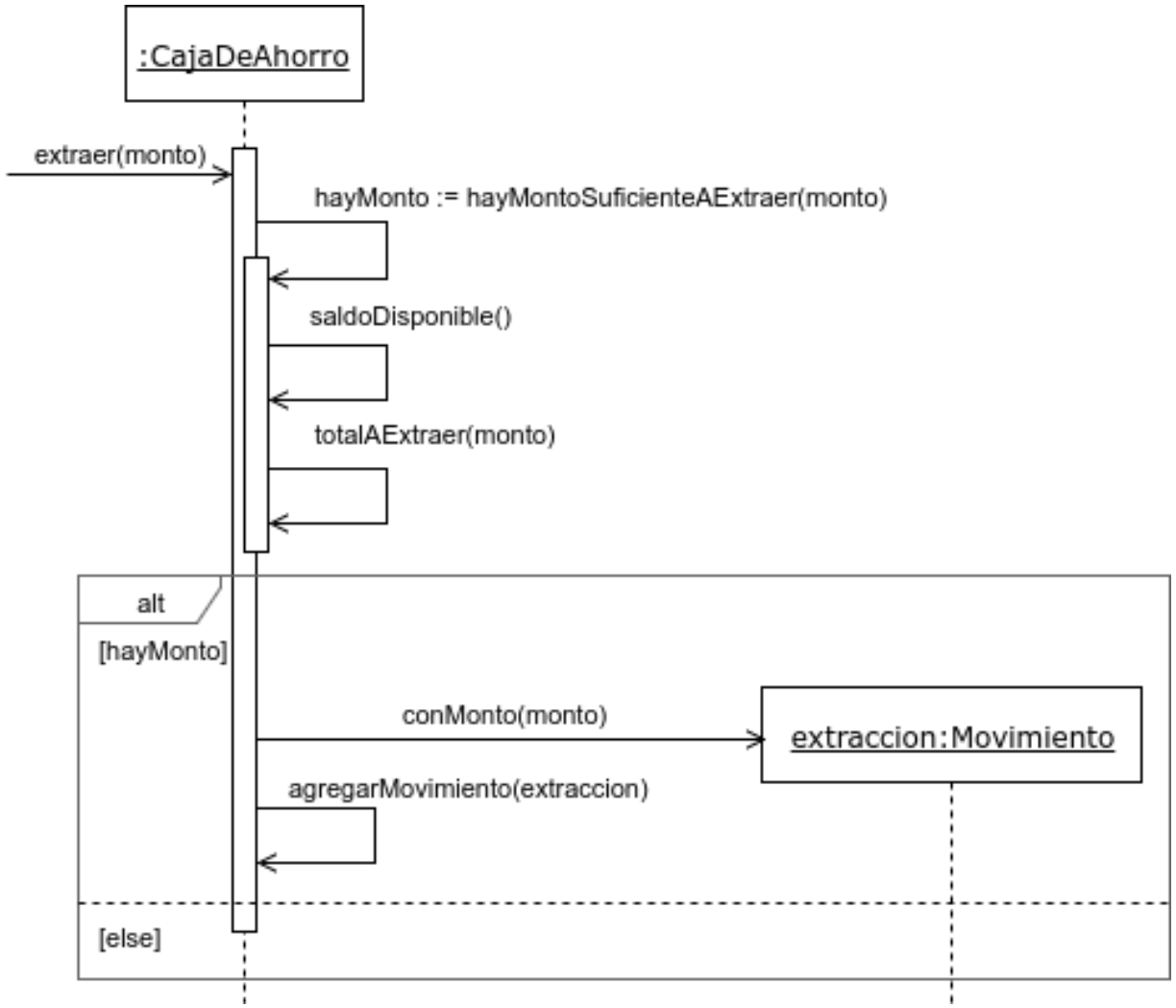


Ejemplo: Transferencia bancaria



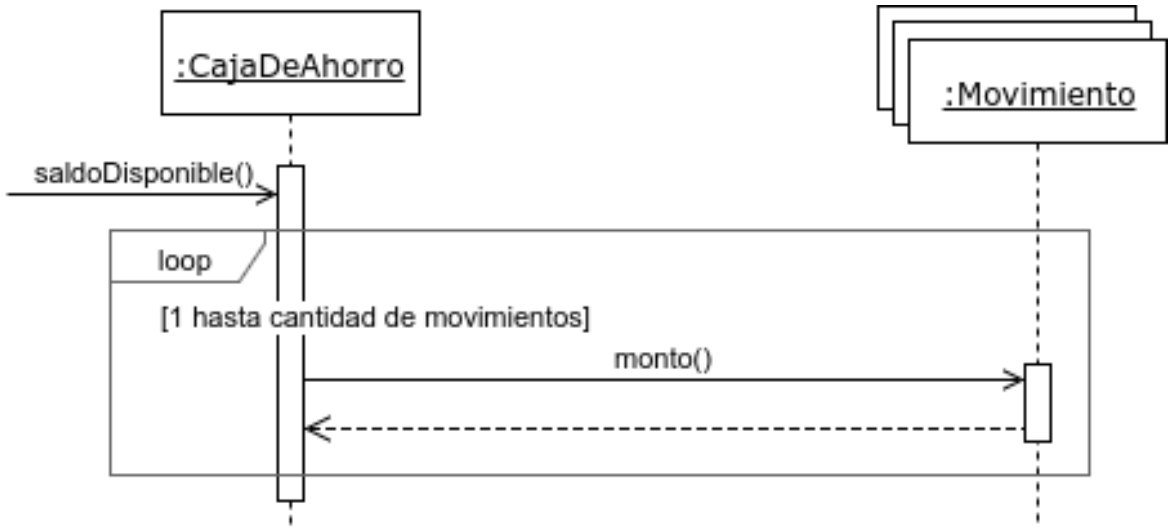
Otro Ejemplo: Extracción con registro de movimiento

El diagrama describe siempre un **escenario de uso**, no es necesario que represente todas las posibilidades



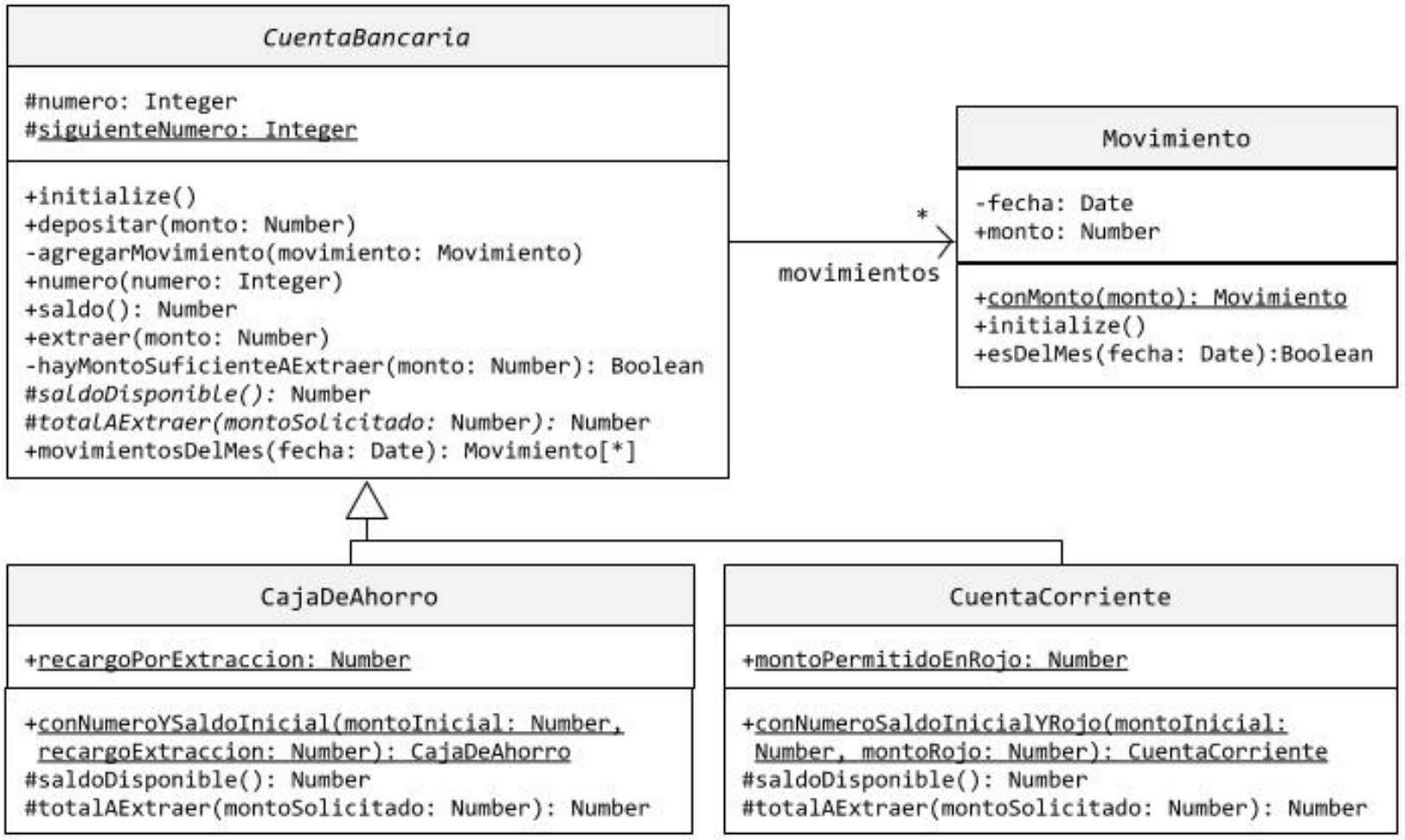
Otro Ejemplo: Calculo del saldo disponible

Considerando los *movimientos* de la CuentaBancaria



Qué es el registro de movimientos de una CuentaBancaria???

Diagrama de clases – Ejemplo ya presentado



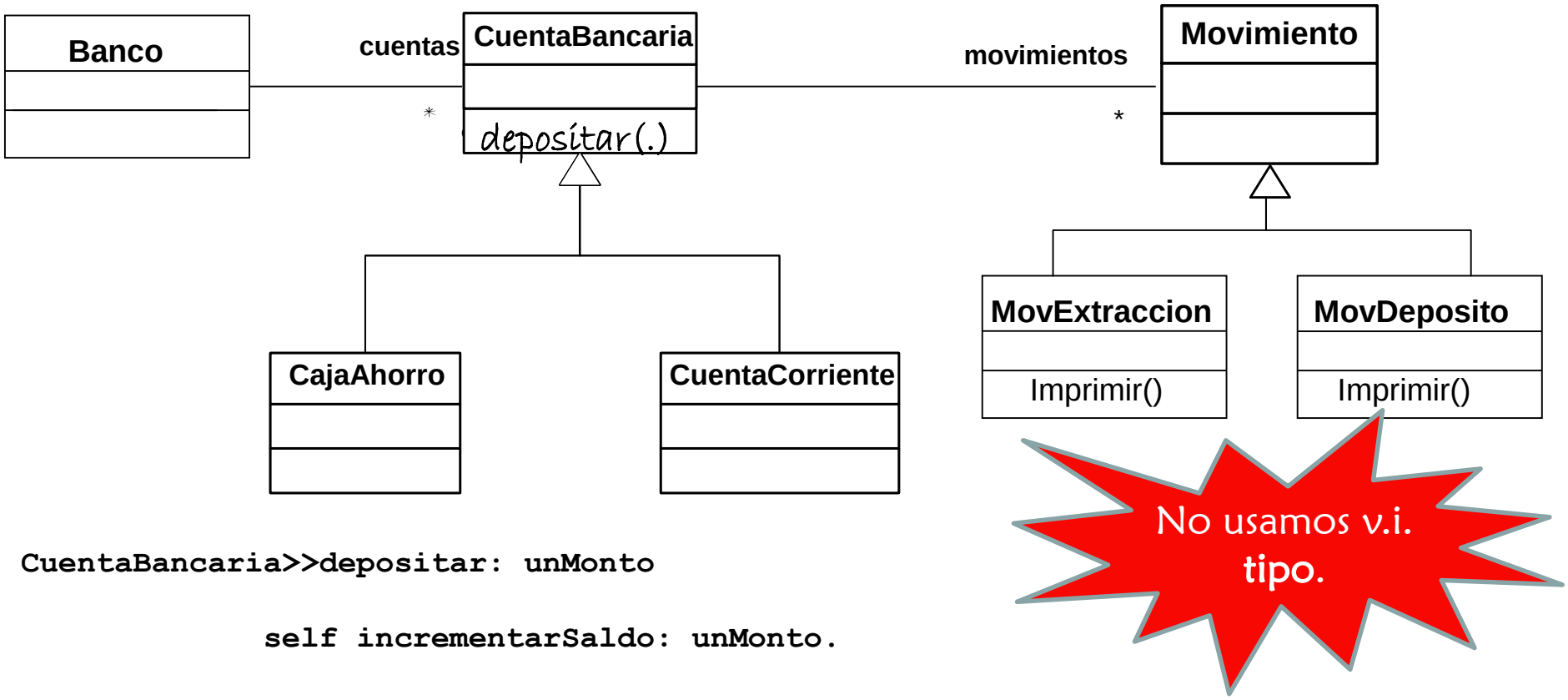
Crear el registroDeMovimientos para Cuentas Bancarias

```
CuentaBancaria>>initialize:unNumero

        self numeroCuenta: unNumero.
        ….
        movimientos:= 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 a la colección de **movimientos**

Registrar Movimientos de Extracción y Depósito




```
CuentaBancaria>>depositar: unMonto

        self incrementarSaldo: unMonto.

        self movimientos 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 movimientos do:[:m| m imprimir]
```

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

Retornar el último movimiento realizado en una cuenta

```
CuentaBancaria>>ultimoMovimiento
^(self movimientos 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

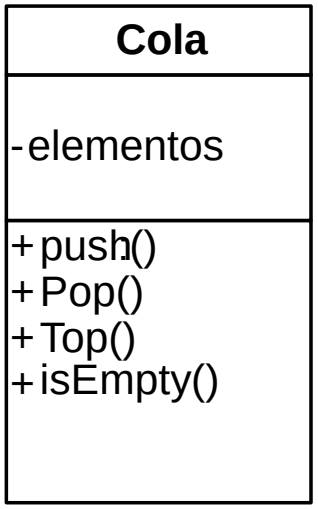
Un simple ejemplo: Clase Cola

- 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….