Facultad de Informática

UNLP

11:00 hs
Aula a confirmar
Presencial

03 OCT

Conferencia del Dr. Danny Dig

AI Programming Assistant

Many are asking will Generative AI and Large Language Models (LLMs) put programmers out of business? Will it displace my education and research and make it irrelevant? This talk provides a fresh perspective on how Generative AI is not displacing programmers, but it is a thinking partner. We provide examples of how GenAI assists with the 9 hardest things software developers have to do. A top concern remains the trustworthiness of the solutions provided by GenAI. While many solutions resemble the ones produced by expert developers, LLMs are known to produce hallucinations, i.e., solutions that seem plausible at first, but are deeply flawed. To help software developers trust LLM solutions, we developed novel research that synergistically combines the creative potential of LLMs with the safety of static and dynamic analysis from program transformation systems. Our current results show that our approach is effective: it safely automates code changes and is several times better than previous state of the art that relies solely on static analysis. Moreover, our approach produces results that expert developers trust: we submitted patches generated by our LLM-powered tools to famous open-source projects whose developers accepted most of our contributions. This shows the usefulness of our novel approach and ushers us into a new era when LLMs become effective AI assistants for developers. We hope to inspire you with ideas on how LLMs can help you and your education and research go even further.

Bio

Danny Dig is an associate professor of Computer Science at the University of Colorado where he leads research in GenAI for Software Engineering. He is back at the university, after a long industry sabbatical with JetBrains, the company that develops leading IDEs such as IntelliJ IDEA and PyCharm.
Danny is the founder and the executive director for the Center on Pervasive Personalized Intelligence (https://ppicenter.org), an industry-university consortium under the US National Science Foundation. Together with partners from industry, government, and academia, they advance the science, practice, and education on Intelligent Systems.
Danny's research group was recognized with more than a dozen distinguished awards at the flagship academic conferences in Software Engineering. Together with his grad students, they released dozens of software systems that are shipping with the official release of the popular development environments which are used daily by millions of developers.
Danny equips transformational leaders in academia, businesses, and government so they can reach their full potential.

Para asistir, registrarse en https://forms.gle/2J9Cg5LwLXzGDwib7

Lifia
FACULTAD DE INFORMÁTICA
UNIVERSIDAD NACIONAL DE LA PLATA

Introducción al Análisis y Diseño Orientado a Objetos (parte 2)

Diego Torres – diego.torres@lifia.info.unlp.edu.ar

2do parte de diseño
Contratos de las operaciones

Contratos

- Son una forma de describir el comportamiento en un sistema en forma detallada.
- Describen pre (antes) y post (después) condiciones.

Secciones en un contrato

- Operación: se detalle el nombre de la operación y los parámetros
- Pre-condiciones: Suposiciones relevantes sobre el estado del sistema o de los objetos del Modelo del Dominio, antes de la ejecución de la operación
- Post-condiciones: el estado del sistema o de los objetos del Modelo del Dominio, después de que se complete la ejecución de la operación.
  - Describen cambios que deben ocurrir. Por ejemplo: Creación de elementos, creación de asociaciones, destrucción de elementos o asociaciones.
  - Son declarativas: afirmaciones que deben ser verdaderas luego de la ejecución de la operación.

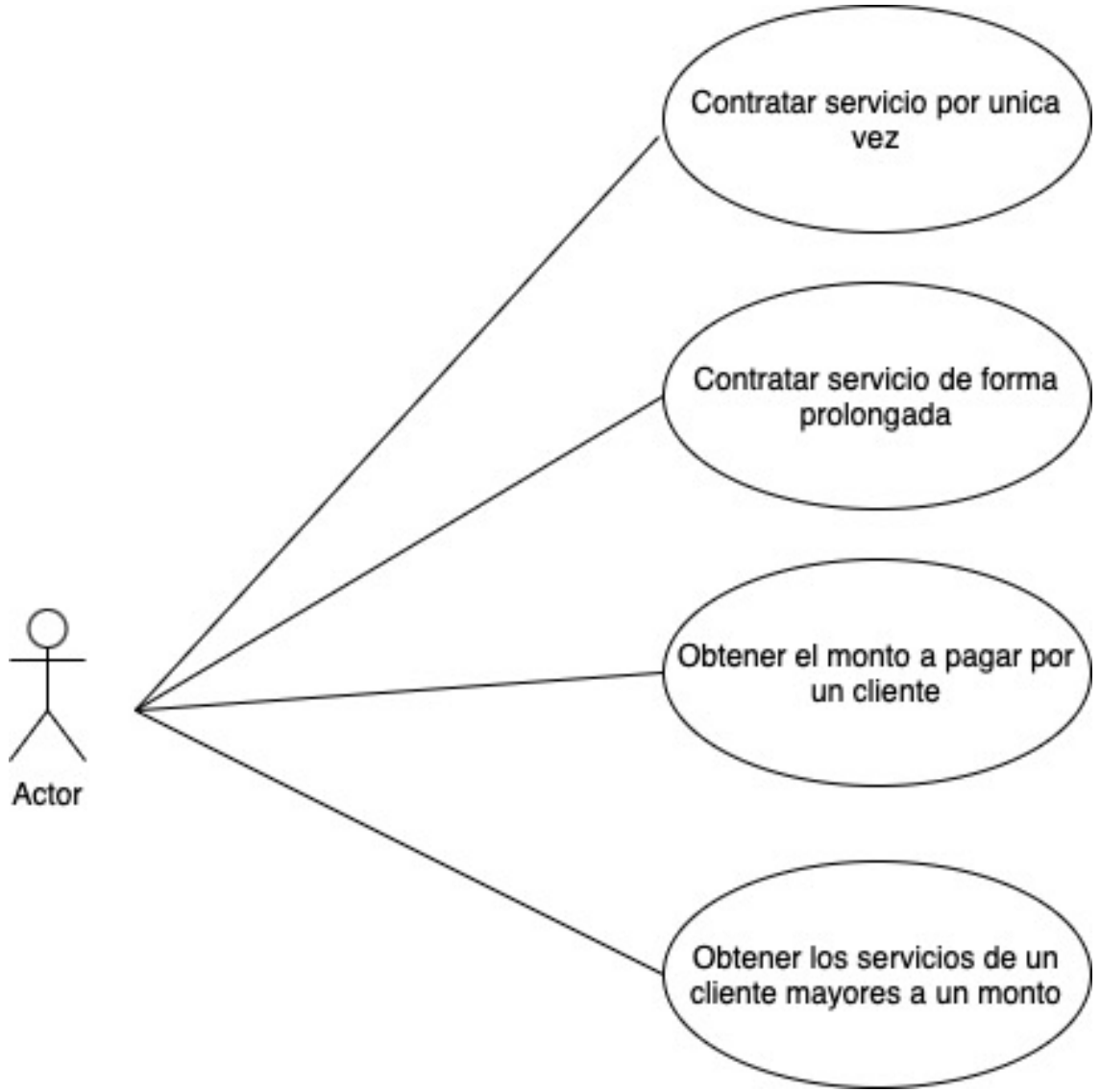
Pre-condiciones

- Suposiciones relevantes sobre el estado del modelo antes de la ejecución de la operación.
- No se validarán dentro de la operación, sino que se asumirán válidas.
- Son suposiciones no triviales.
- Ejemplo:
  - La cantidad de días de contratación de un servicio prolongado es mayor a 1.

Post-condiciones

- Describen cambios en los objetos del modelo de dominio.
  - Modificación del valor de atributos.
  - Creación o eliminación de instancias.
  - Creación o ruptura de asociaciones.
- Son declarativas.
- Ejemplo
  - El cliente posee una nueva contratación.

Casos de uso


Contratar servicio por unica vez
Contratar servicio de forma prolongada
Obtener el monto a pagar por un cliente
Obtener los servicios de un cliente mayores a un monto
Actor

Ejemplo

- Operación: contratar servicio por única vez (c: Cliente, fecha: Date, s: Servicio)
- Pre-condición:
  - la fecha es una fecha válida para el contexto.
- Post-condición
  - el cliente posee una contratación por única vez para el servicio s y la fecha indicada.

Contratos – Test de Unidad

- Los contratos son una antesala a los test de unidad.
- Definen los requerimientos en términos de pre y post condiciones.
- Las pre-condiciones dan una idea del fixture del test.
- Las post-condiciones dan una idea de las verificaciones del test (los assert).

Test de Unidad lo verán
la próxima clase con el profesor
Alejandro Fernández

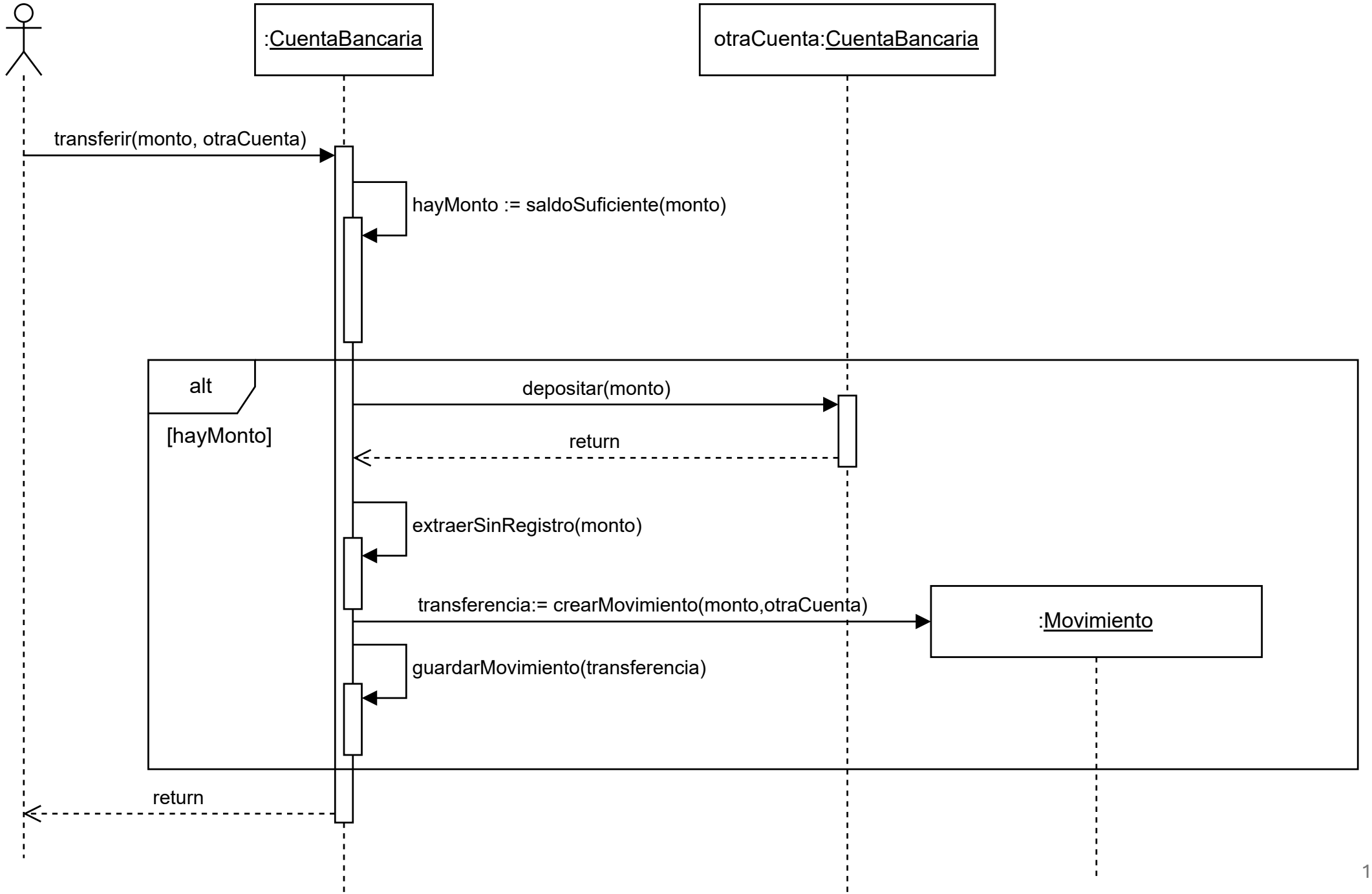
Del análisis al diseño

- Crear diagramas de interacción que muestran cómo los objetos se comunican con el objetivo de cumplir con los requerimientos capturados en la etapa de análisis.
- A partir de los diagramas de interacción, diseñar diagramas de clases representando las clases que serán implementadas.

Crear diagramas de interacción requiere la aplicación de Principios o Heurísticas para la Asignación de Responsabilidades.

Diagrama UML de Secuencia

En 10 minutos


:CuentaBancaria
otraCuenta:CuentaBancaria
transferir(monto, otraCuenta)
hayMonto := saldoSuficiente(monto)
alt
[hayMonto]
depositar(monto)
return
extraerSinRegistro(monto)
transferencia:= crearMovimiento(monto,otraCuenta)
:Movimiento
guardarMovimiento(transferencia)
return

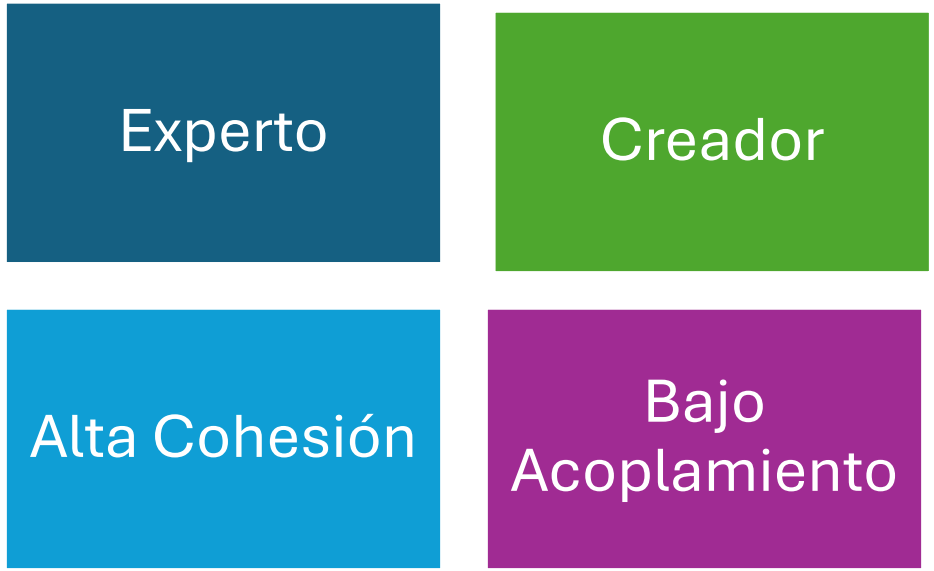
Heurísticas para Asignación de Responsabilidades (HAR)

Responsabilidades de los objetos

- Hacer
  - Hacer algo por si mismo.
  - Iniciar una acción en otros objetos.
  - Controlar o coordinar actividades de otros objetos.
- Conocer (para hacer)
  - Conocer sus datos privados encapsulados.
  - Conocer sus objetos relacionados.
  - Conocer cosas derivables o calculables.

Heurísticas para asignar responsabilidades

- La habilidad para asignar responsabilidades es extremadamente importante para el diseño orientado a objetos.
- La asignación de responsabilidades generalmente sucede durante la creación de los diagramas de secuencia.



Experto

- Asignar una responsabilidad al experto en información (la clase que tiene la información necesaria para realizar la responsabilidad).
- Expresa la intuición de que los objetos hacen cosas relacionadas con la información que tienen.
- Para cumplir con su responsabilidad, un objeto puede requerir de información que se encuentra dispersa en diferentes clases expertos en información "parcial".

Ejemplo

- En el ejercicio de los servicios de limpieza y parquizaciones.

- ¿Quién tiene la responsabilidad de responder el monto a pagar de todos los servicios contratados?
  - El Cliente

¿Cuáles serían los colaboradores para que Cliente complete la responsabilidad?

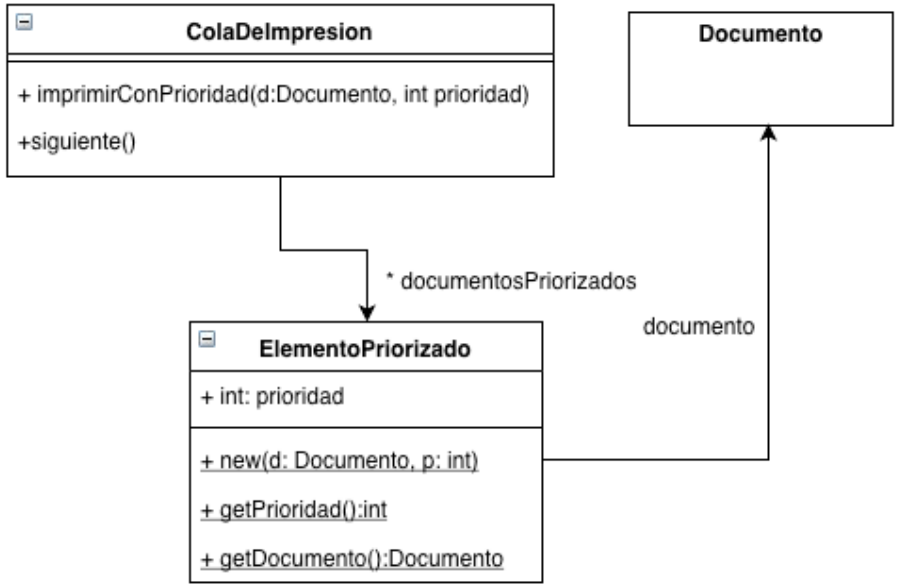
Creador

Asignar a la clase B la responsabilidad de crear una instancia de la clase A si:

- **B usa a objetos A en forma exclusiva.**
  - **Ningún otro sabe de esos objetos.**
- B contiene objetos A (agregación, composición).
  - **¡Es una relación fuerte de composición, no es simplemente que los conoce !**
- B tiene los datos para inicializar objetos A.
  - **¡No necesita que se los pasen por parámetro!**

Ejemplo

- Una Cola de Impresión crea Posicionadores



Creador

Asignar a la clase B la responsabilidad de crear una instancia de la clase A si:

- **B usa a objetos A en forma exclusiva.**
  - **Ningún otro sabe de esos objetos.**
- B contiene objetos A (composición).
  - **¡Es una relación fuerte de composición, no es simplemente que los conoce !**
- B tiene los datos para inicializar objetos A.
  - **¡No necesita que se los pasen por parámetro!**

Ejemplo

- Una clase instancia su colección de elementos.

```
public class Presupuesto {
    private List<Articulo> articulos;

    public Presupuesto(){
        this.articulos = new ArrayList<Articulo>();
    }
```

Creador

Asignar a la clase B la responsabilidad de crear una instancia de la clase A si:

- **B usa a objetos A en forma exclusiva.**
  - **Ningún otro sabe de esos objetos.**
- B contiene objetos A (agregación, composición).
  - **¡Es una relación fuerte de composición, no es simplemente que los conoce !**
- B tiene los datos para inicializar objetos A.
  - **¡No necesita que se los pasen por parámetro!**

Ejemplo

```
public void enviarEmail() {
    String cuerpo = "Estimado "+
        cliente.getNombreyApellido()+
        "su envio de "+items.size()+
        "ya se está enviando";
    Email email = new Email(cliente.getDireccionEMail(),
            this.emailFrom(),"Aviso de envío", cuerpo);
    this.getServidor().enviar(email);
}
```

Bajo Acoplamiento

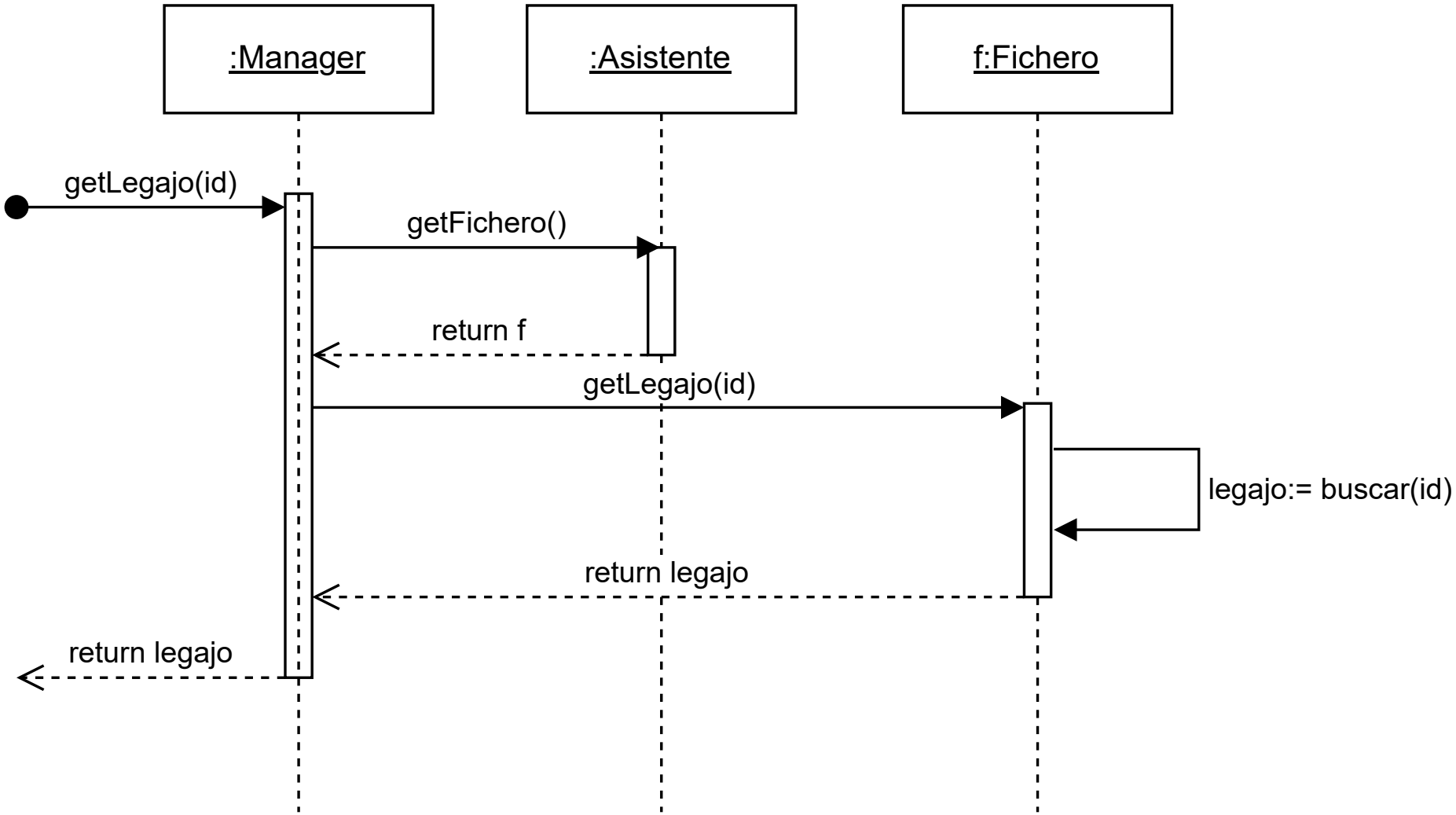
- El acoplamiento es una medida de dependencia de un objeto con otros. Es bajo si mantiene pocas relaciones con otros objetos.
- El alto acoplamiento dificulta el entendimiento y complica la propagación de cambios en el diseño.
- No se puede considerar de manera aislada a otras heurísticas, sino que debe incluirse como principio de diseño que influye en la elección de la asignación de responsabilidad.
- Asignar responsabilidad de manera que el acoplamiento se mantenga lo más bajo posible.

Alta Cohesión

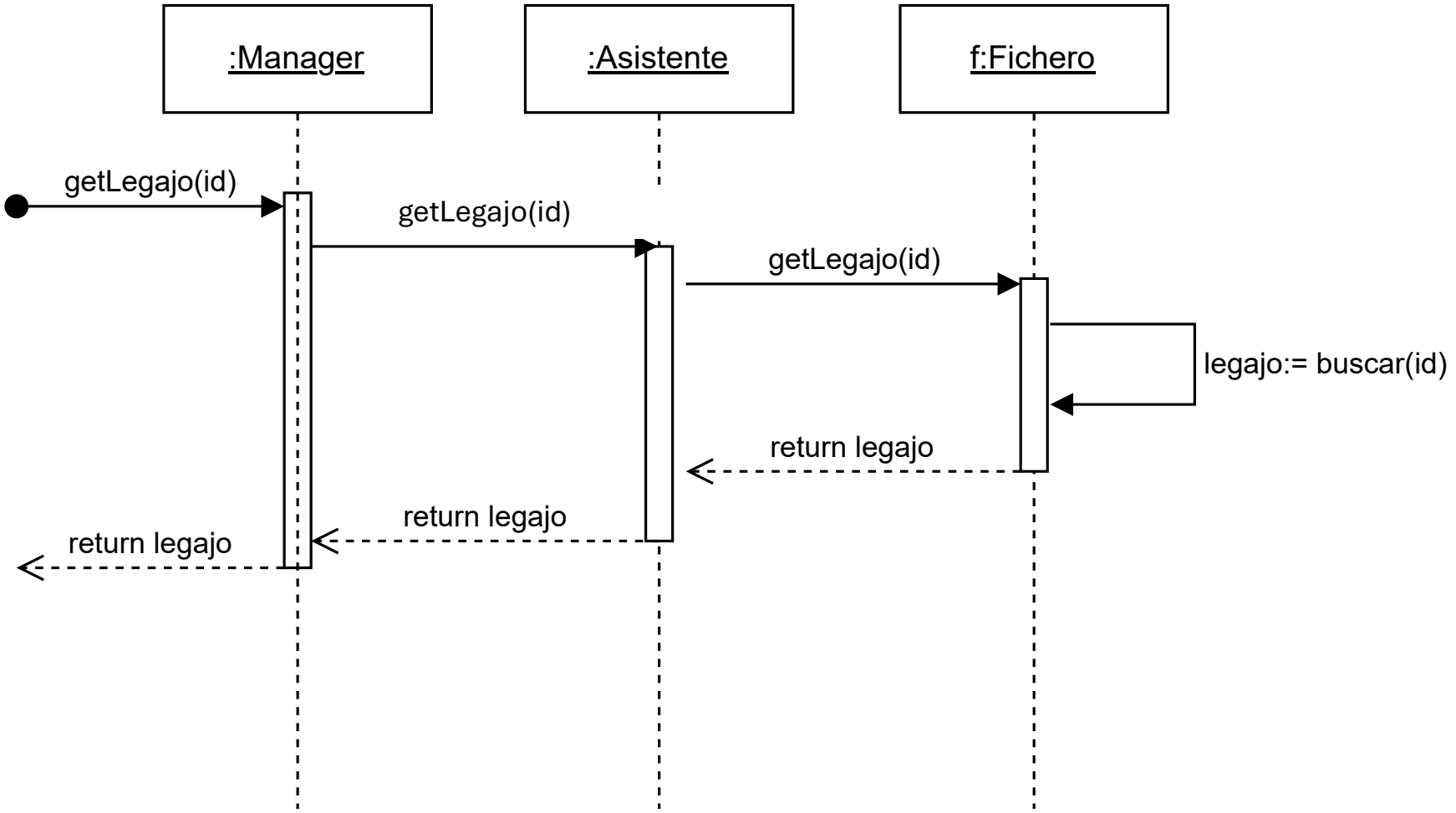
- Asignar responsabilidades de manera que la cohesión se mantenga lo más fuerte posible.
- La cohesión es una medida de la fuerza con la que se relacionan las responsabilidades de un objeto, y la cantidad de ellas.
- **Ventaja:** clases más fáciles de mantener, entender y reutilizar.
- El nivel de cohesión no se puede considerar de manera aislada a otras responsabilidades y otras heurísticas, como Experto y Bajo Acoplamiento.

Manager y ficheros

Acoplamiento / Cohesión



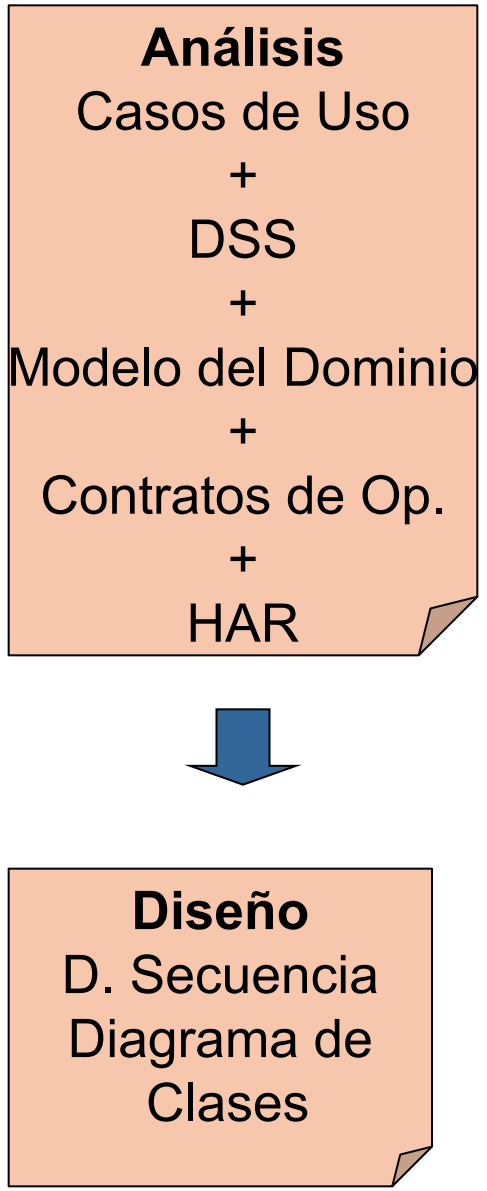
Acoplamiento / Cohesión



Modelo de Diseño

Del Análisis al Diseño

- Los **casos de uso** sugieren los eventos del sistema que se muestran en los **diagramas de secuencia** del sistema.
- En los contratos de las operaciones, utilizando conceptos del Modelo del Dominio, se describen los efectos que dichos eventos (operaciones) producen en el sistema.
- Los eventos del sistema representan los mensajes que dan inicio a Diagramas de Secuencia del Diseño, mostrando las interacciones entre los objetos del sistema.
- Los objetos con sus métodos y relaciones se muestran en el Diagrama de Clases del Diseño (basado en el Modelo del Dominio).



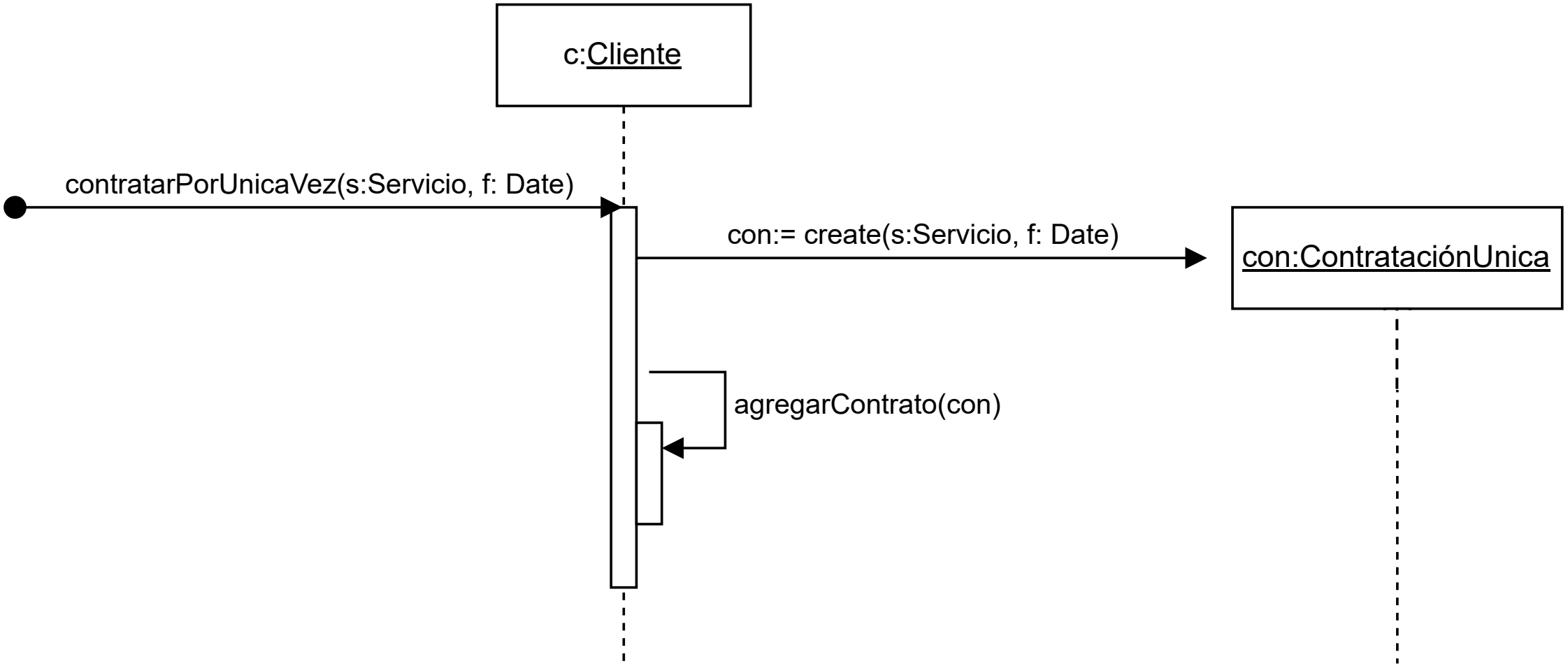
Diagramas de Secuencia

- Cree un **diagrama de secuencia** por cada operación asociada al caso de uso.
- Si el diagrama queda complejo, sepárelo en diagramas menos complejos (uno por cada escenario).
- Use el **contrato de la operación** como punto de partida; piense en objetos que colaboran para cumplir la tarea (la mayoría de estos objetos están definidos en el modelo del dominio).
- Aplique las **Heurísticas para Asignación de Responsabilidades** (HAR) para obtener un mejor diseño.

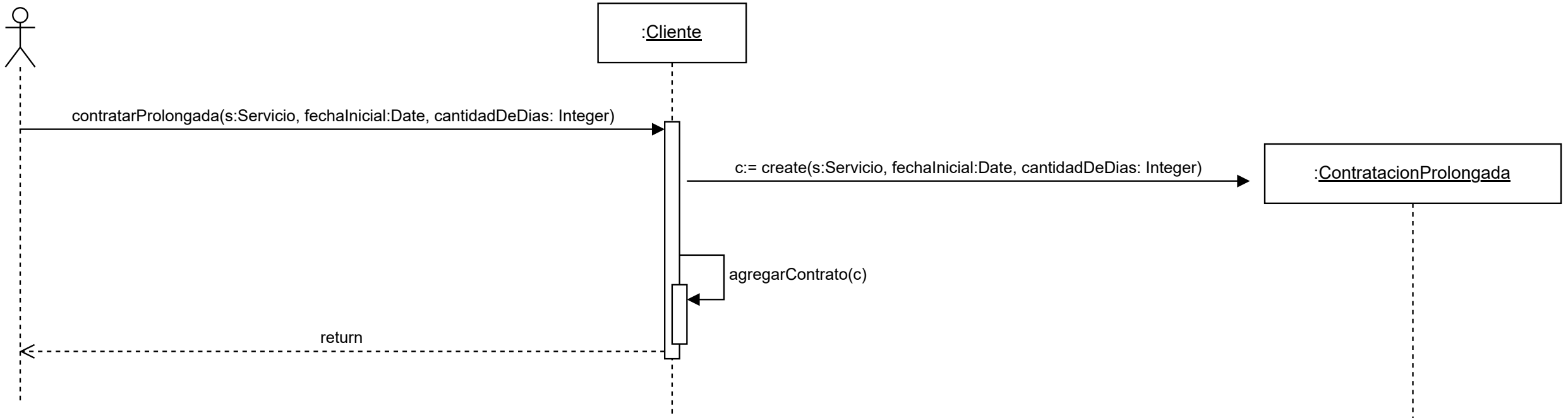
Ejemplo

- Operación: contratar servicio por única vez (c: Cliente, fecha: Date, s: Servicio)
- Pre-condición:
  - la fecha es una fecha válida para el contexto.
- Post-condición
  - el cliente posee una contratación por única vez para el servicio s y la fecha indicada.

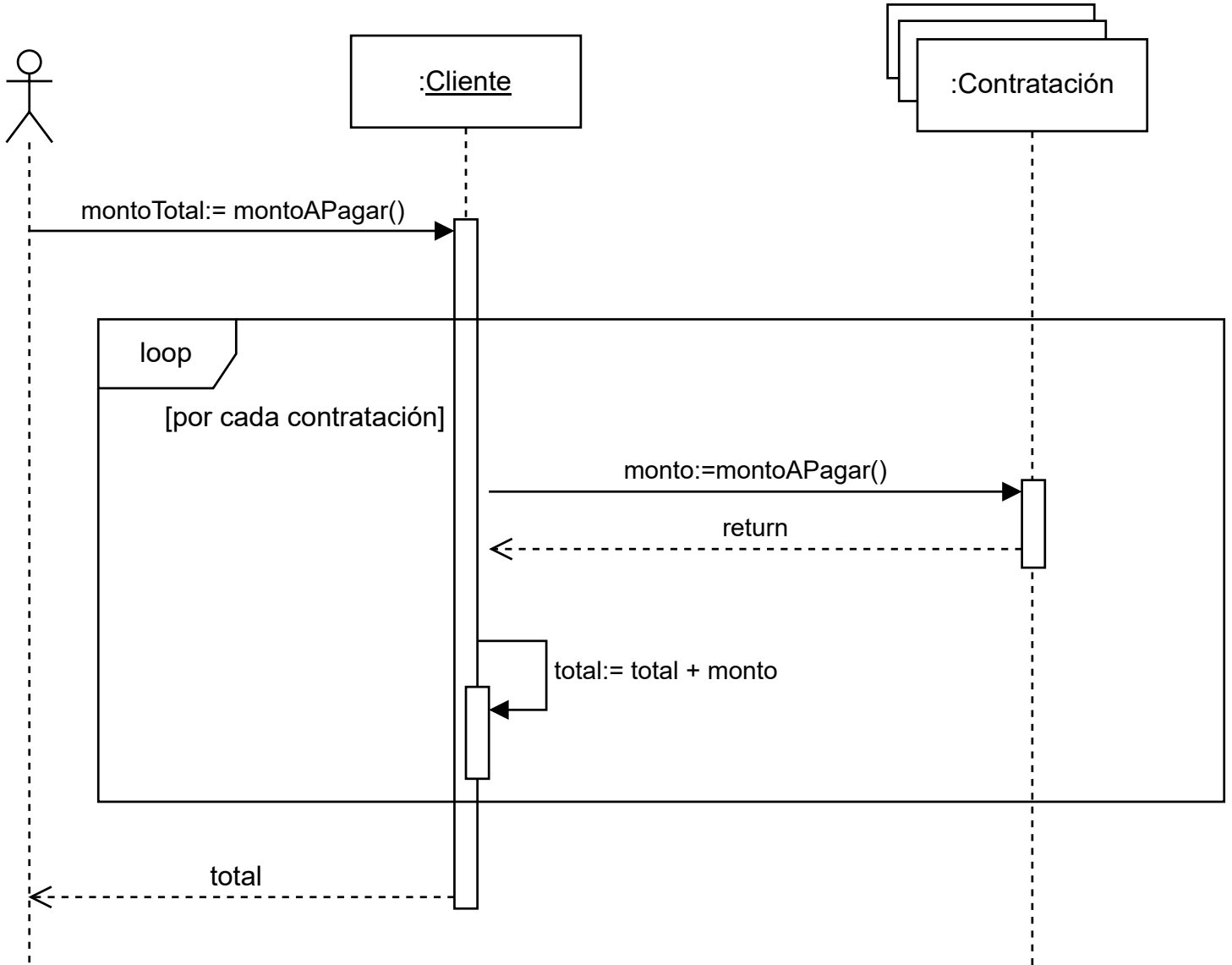
Secuencia


c:Cliente
contratarPorUnicaVez(s:Servicio, f: Date)
con:= create(s:Servicio, f: Date)
con:ContrataciónUnica
agregarContrato(con)

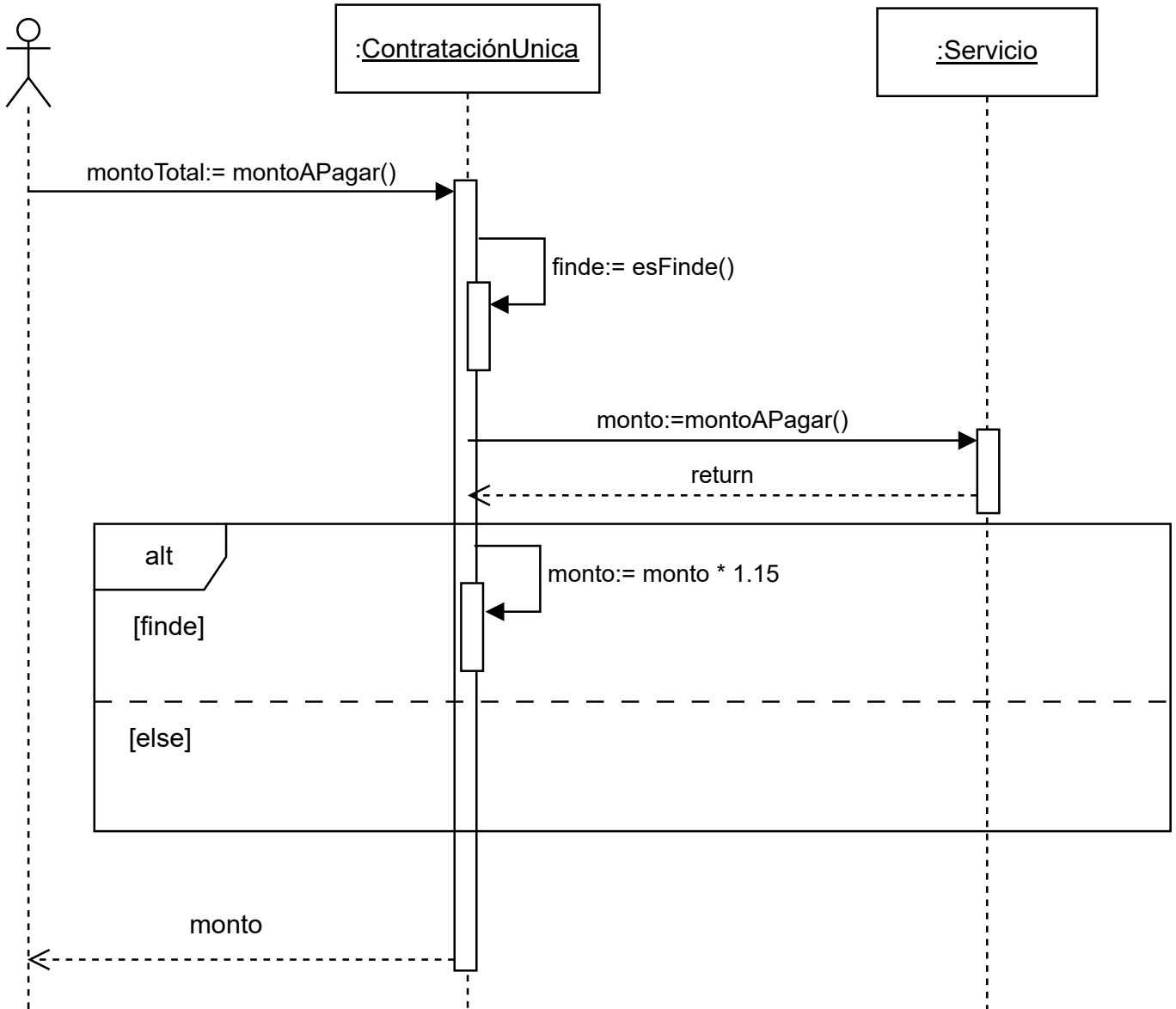
Secuencia


:Cliente
contratarProlongada(s:Servicio, fechaInicial:Date, cantidadDeDias: Integer)
c:= create(s:Servicio, fechaInicial:Date, cantidadDeDias: Integer)
:ContratacionProlongada
agregarContrato(c)
return

Secuencia


:Cliente
:Contratación
montoTotal:= montoAPagar()
loop
[por cada contratación]
monto:=montoAPagar()
return
total:= total + monto
total

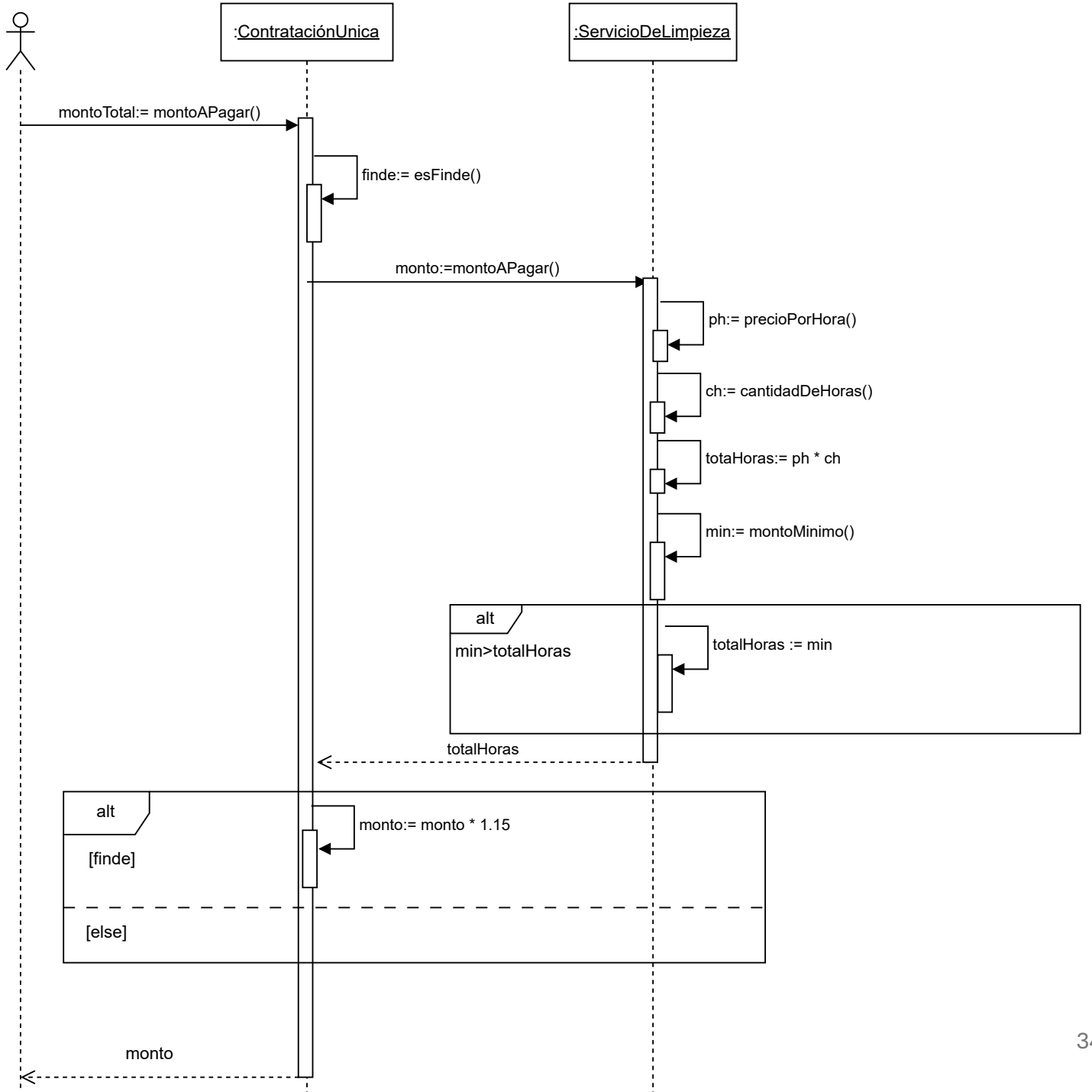
Secuencia


:ContrataciónUnica
:Servicio
montoTotal:= montoAPagar()
finde:= esFinde()
monto:=montoAPagar()
return
alt
[finde]
monto:= monto * 1.15
[else]
monto

Secuencia

Aquí detallamos el caso específico del tipo de Servicio. En este caso para un Servicio de Limpieza

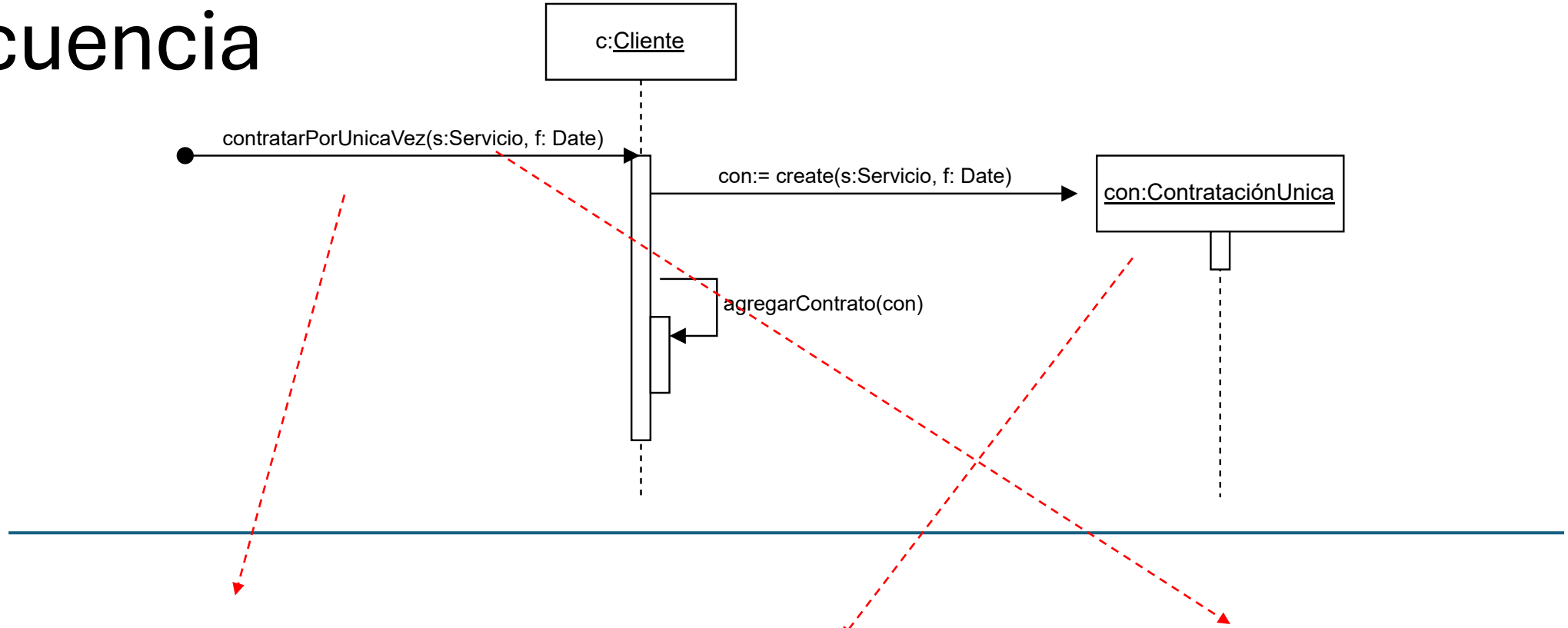
Debe describirse qué es lo que sucede con un Servicio de Parquización



Creación de los Diagramas de Clases

- Identificar las clases que participan en los diagramas de interacción y en el Modelo del Dominio o Conceptual.
- Graficarlas en un diagrama de clases.
- Colocar los atributos presentes en el Modelo Conceptual.
- Agregar nombres de métodos analizando los diagramas de interacción.
- Agregar tipos y visibilidad de atributos y métodos.
- Agregar las asociaciones necesarias.
- Agregar roles, navegabilidad, nombre y multiplicidad a las asociaciones.

Secuencia



Cliente
+ nombreYApellido: String + direccion: String
+ contratarPorUnicaVez(s:Servicio, f:Date)

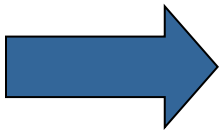
ContrataciónUnica
+ fechaDeContratación: Date
+ new(s:Servicio, f:Date)

Servicio
+ ??
+ ??

Construyendo el Diagrama de Clases

Transformación de los diseños en código

- Mapear los artefactos de diseño a código orientado a objetos :

Clases		Clases
Atributos		Variables simples
Asociaciones (roles)		Variables de referencia
Métodos		Métodos
Multiobjetos		Collections

Extendiendo el modelo conceptual

Nuevos conceptos pueden ser identificados y agregados al modelo.

Supongamos, para un pago se indica ahora que:

- Sólo permite hacer el pago con tarjeta de crédito (no hay lugar físico)

- debe poder realizarse el pago con la tarjeta registrada en el cliente.

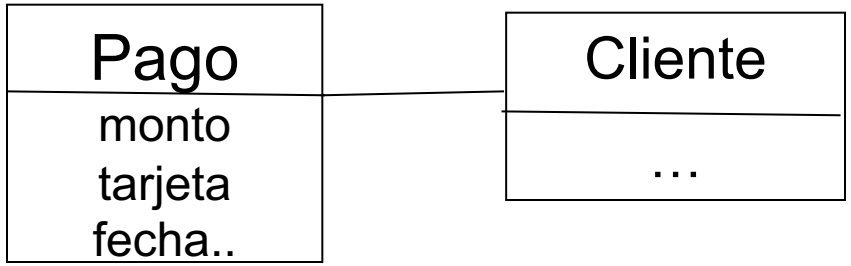
Entonces:

-Podemos pensar en un objeto que modele un pago y posea el atributo **tarjeta**

Extendiendo el modelo conceptual

Solo se permite hacer el pago con tarjeta de crédito (no hay lugar físico)

-Agregamos el atributo **tarjeta** al Pago, para registrar el tipo de tarjeta con que se realiza el pago



Ahora, supongamos que se consideran las siguientes particularidades:

- Si el pago se hace con Tarjeta Clásica, el único beneficio es pagar el precio en 3 cuotas sin interés.
- una bonificación del 3% en el Pago a los clientes que pagan con tarjeta de Crédito Oro
- una bonificación del 5% si el monto supera los 35.000 pesos, a los que pagan con tarjeta Platino.

Extendiendo el Modelo Conceptual

Descubriendo nuevas clases para evitar preguntar por el tipo o valor de un atributo.

Para resolver el planteo que depende del atributo **tarjeta** de la Clase **Pago**, debemos consultar por su valor (**uso de if , sentencias Case..):**

if tarjeta = `oro`…….

if tarjeta = `platino`…….

if tarjeta = `clásica`…..

¿Es esta una buena práctica en Orientación a Objetos?

¿Cómo lo resolveríamos usando conceptos Orientados a Objetos?

Extendiendo el Modelo Conceptual

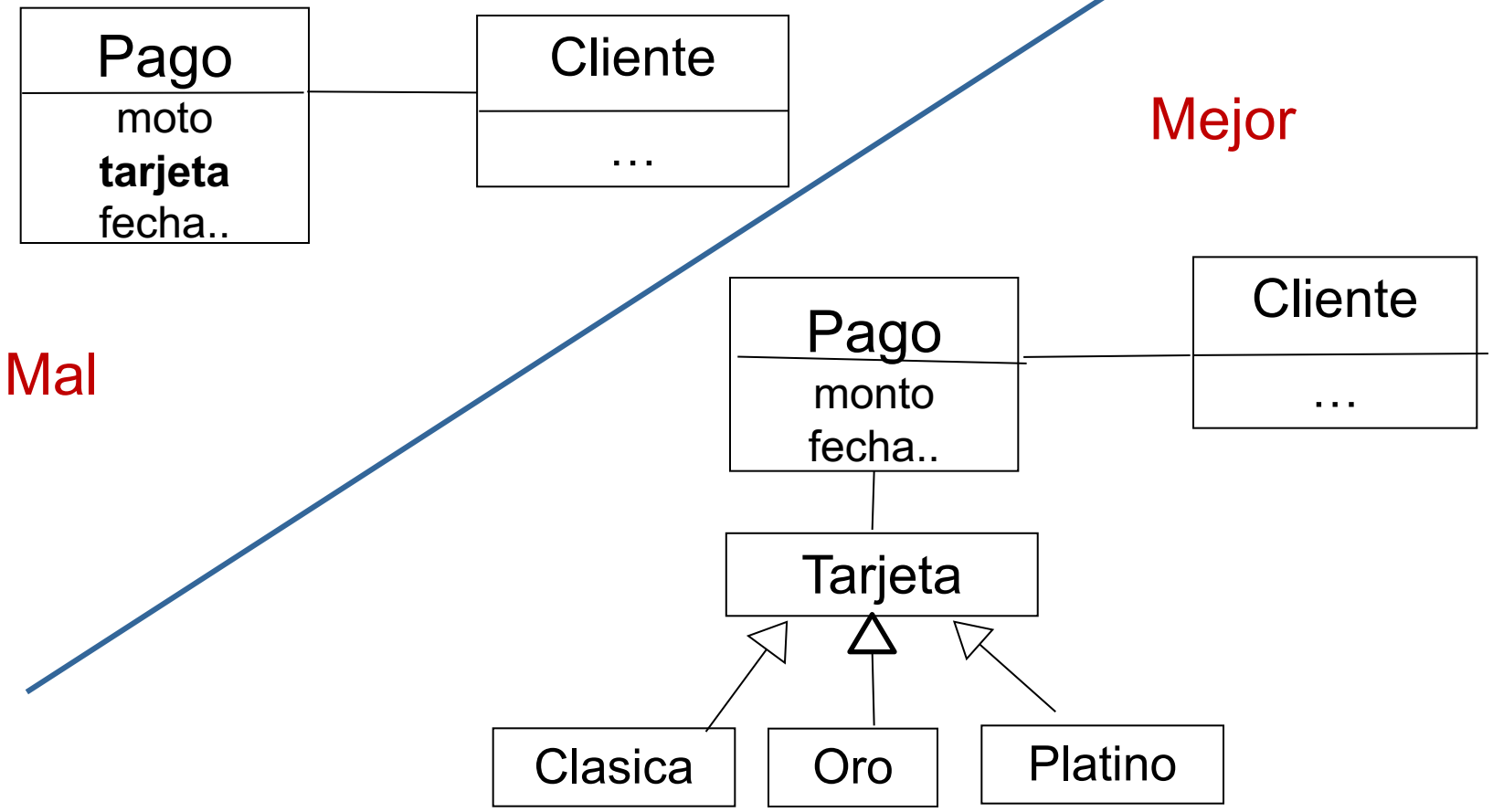
Descubriendo nuevas clases para evitar preguntar por el tipo o valor de un atributo. Polimorfismo

Debemos poder delegar en el **tipo de tarjeta** el cálculo de la **bonificación** que corresponde a cada tipo.

Es decir, debemos agregar una **nueva clase y subclases** que resuelvan, aplicando **polimorfismo**, cada cálculo de la bonificación.

Extendiendo el Modelo Conceptual

Descubriendo nuevas clases para evitar preguntar por el tipo o valor de un atributo. Polimorfismo

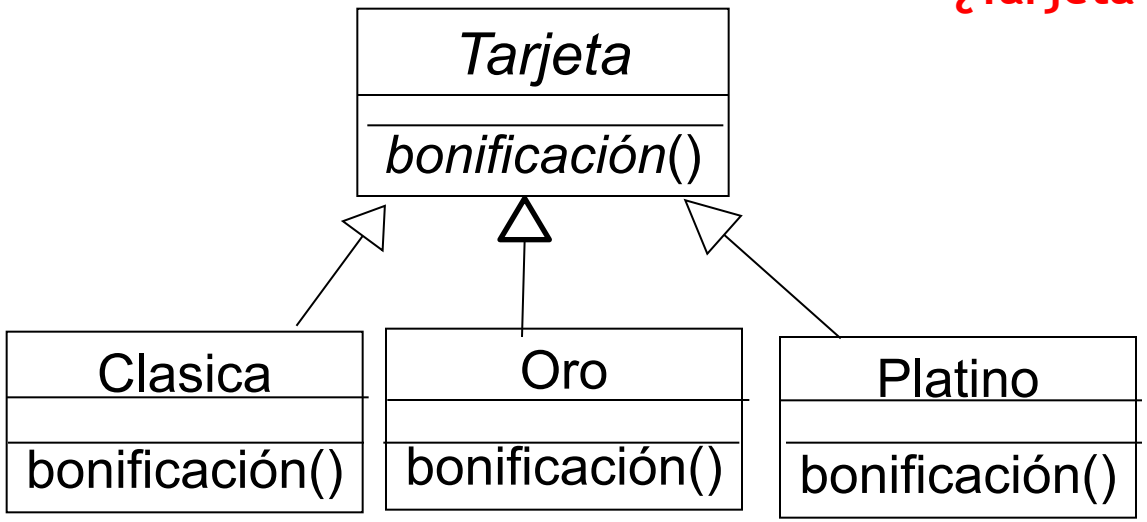


Extendiendo el Modelo Conceptual

Descubriendo nuevas clases para evitar preguntar por el tipo o valor de un atributo.

¿Dónde aplico Polimorfismo?

¿Tarjeta Clase o Interfaz?



Descubriendo jerarquías

¿Debería clasificar objetos por su estado, o clasificar estados (cambia el objeto en tiempo de ejecución)?

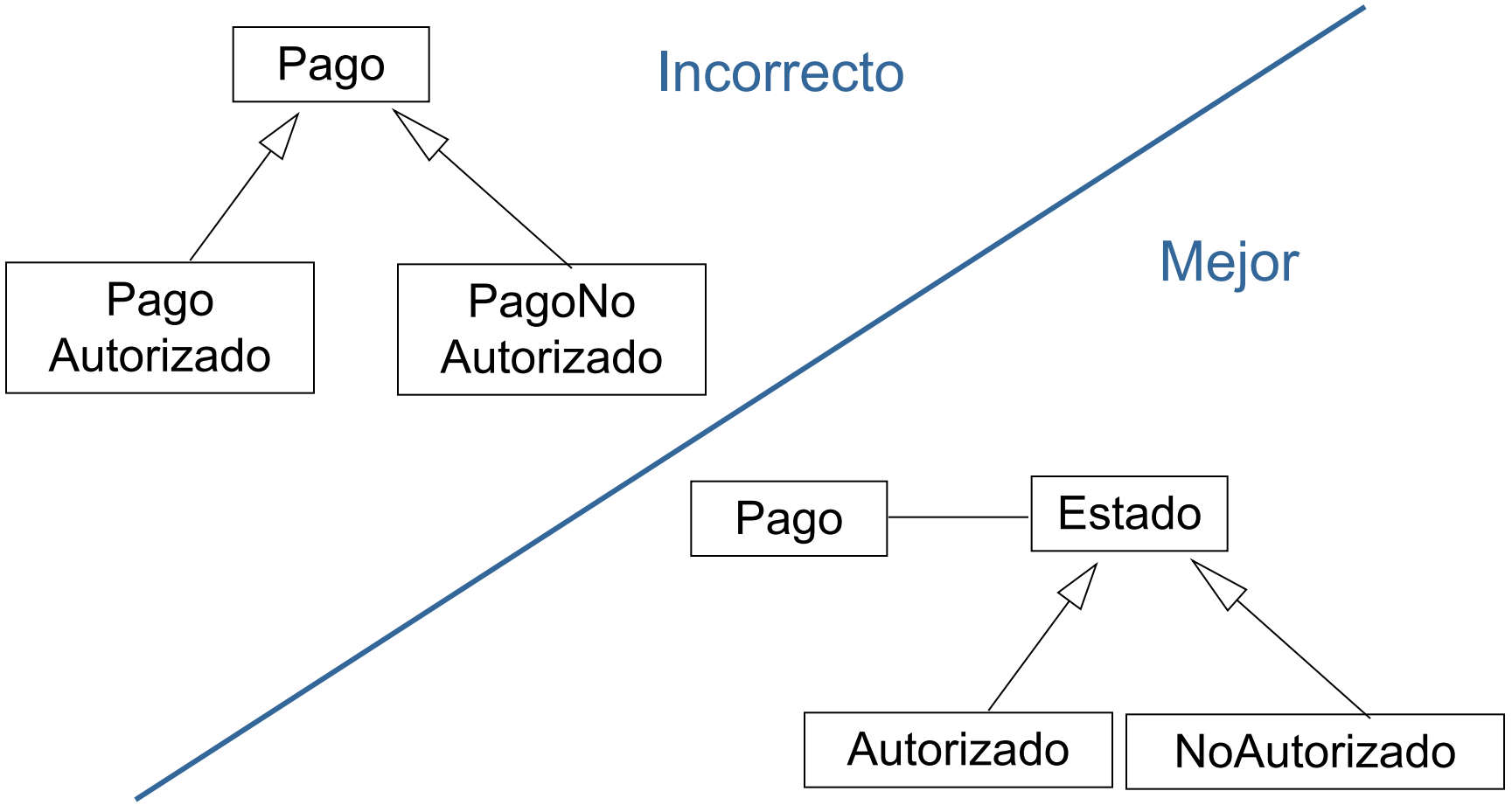


Diagrama de clases – Requiere seguir iterando