UNIVERSIDAD
Popular del cesar

Ingeniería de Sistemas

ESPECIALIZACION EN INGENIERIA DE SOFTWARE
MODULO PATRONES DE DISEÑO DE SOFTWARE

EL DOCENTE

JAIRO FRANCISCO SEOANES LEON

jairoseoanes@unicesar.edu.co
(300) 600 06 70

Educación formal

- ✓ **Ingeniero de sistemas**, Universidad Popular del Cesar sede Valledupar, Feb 2002 – Jun 2009.
- ✓ **MsC en Ingeniería de Sistemas y Computación**, Universidad Nacional de Colombia, Bogotá, Feb 2011 – Mar 2015
- ✓ **PhD Ciencia, Tecnologia e innovación, Urbe, Venezuela, Mayo 2024**

Formación complementaria

- ✓ **AWS Academy Graduate -** AWS Academy Cloud Foundations, 2022
  https://www.credly.com/go/p3Uwht36
- ✓ **Associate Cloud Engineer Path -** Google Cloud Academy, 2022
  https://www.cloudskillsboost.google/public_profiles/c7e7936c-3e37-4bad-b822-74d40c49d0db
- ✓ **Fundamentos De Programación Con Énfasis En Cloud Computing –** AWS Academy y Misión Tic 2022
- ✓ **Google Cloud Computing Foundations –** Google Academy, 2022
- ✓ **Aplicación de cloud: retos y oportunidades de mejora para las empresas de software gestionando la computación en la nube –** Fedesoft, 2023
- ✓ **Desarrollo De Aplicaciones Web En Angular, Para El Nivel Frontend** – Universidad EAFIT, 2023
- ✓ **Microsoft Scrum Foundations –** Intelligent Training - MinTic , 2023

Experiencia profesional

- ✓ **Docente Universitario**, Universidad Popular del Cesar sede Valledupar, marzo del 2013.
- ✓ **Técnico de Sistemas Grado 11**, Rama judicial Seccional Cesar, SRPA Valledupar, Junio del 2009

UNIVERSIDAD Popular del cesar

MODULO DE PATRONES DE DISEÑO DE SOFTWARE

Unidad 1

Introducción a UML Y Diseño O.O

Unidad 2

Introducción A Los Patrones de Diseño

Unidad 3

Patrones de Diseño Creacionales

Unidad 4

Patrones de Diseño Estructurales

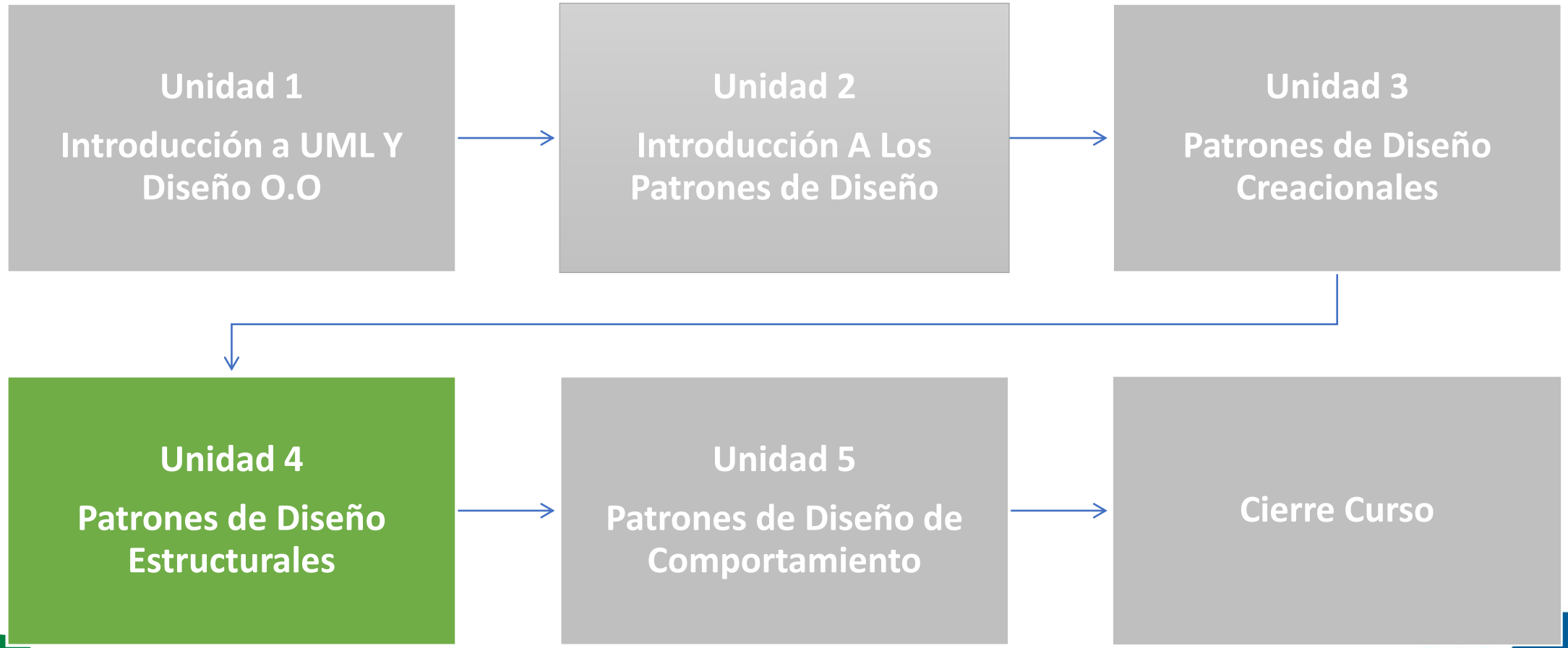
Unidad 5

Patrones de Diseño de Comportamiento

Cierre Curso

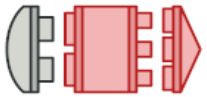
UNIVERSIDAD Popular del cesar

MODULO DE PATRONES DE DISEÑO DE SOFTWARE



UNIVERSIDAD Popular del cesar

Unidad 4. Patrones estructurales



Adapter

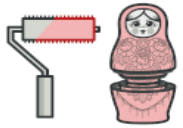
Permite la colaboración entre objetos con interfaces incompatibles.

Bridge

Permite dividir una clase grande o un grupo de clases estrechamente relacionadas, en dos jerarquías separadas (abstracción e implementación) que pueden desarrollarse independientemente la una de la otra.

Composite

Permite componer objetos en estructuras de árbol y trabajar con esas estructuras como si fueran objetos individuales.

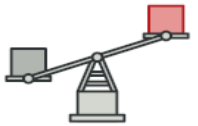


Decorator

Permite añadir funcionalidades a objetos colocando estos objetos dentro de objetos encapsuladores especiales que contienen estas funcionalidades.

Facade

Proporciona una interfaz simplificada a una biblioteca, un framework o cualquier otro grupo complejo de clases.



Flyweight

Permite mantener más objetos dentro de la cantidad disponible de memoria RAM compartiendo las partes comunes del estado entre varios objetos en lugar de mantener toda la información en cada objeto.

Proxy

Permite proporcionar un sustituto o marcador de posición para otro objeto. Un proxy controla el acceso al objeto original, permitiéndote hacer algo antes o después de que la solicitud llegue al objeto original.

UNIVERSIDAD Popular del cesar

Patrones Estructurales - Generales

- ✓ Describen como los objetos y clases se pueden combinar para formar estructuras mas grandes y complejas ayudando a estructurar los objetos, las relaciones y la herencia entre clases, para mantener su flexibilidad y eficiencia.

- ✓ En vez de combinar interfaces o implementaciones (estructuras estáticas), los patrones estructurales de objetos describen formas de componer objetos para obtener nueva funcionalidad.

- ✓ La flexibilidad añadida de la composición de objetos viene dada por la capacidad de cambiar la composición en tiempo de ejecución, lo que es imposible con la composición de clases estática.

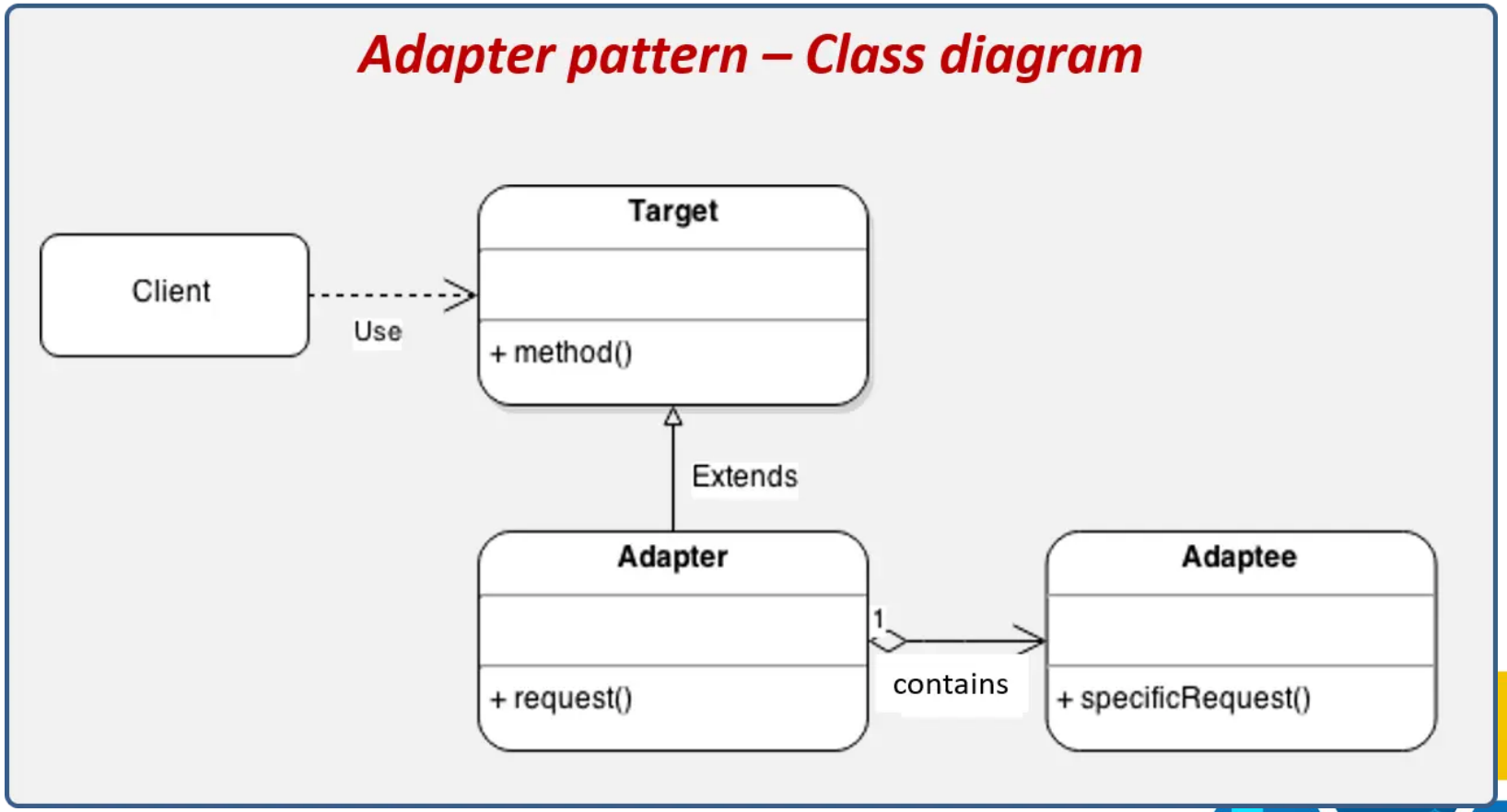
UNIVERSIDAD Popular del cesar

Patrones Estructurales – Adapter

Adapter es un patrón de diseño estructural que permite la colaboración entre objetos con interfaces incompatibles.

Útil cuando tenemos interfaces de software incompatibles, pero con funcionalidad similar,

Se crean clases que convierten una entrada o salida en otra para poder interactuar con una clase



UNIVERSIDAD Popular del cesar

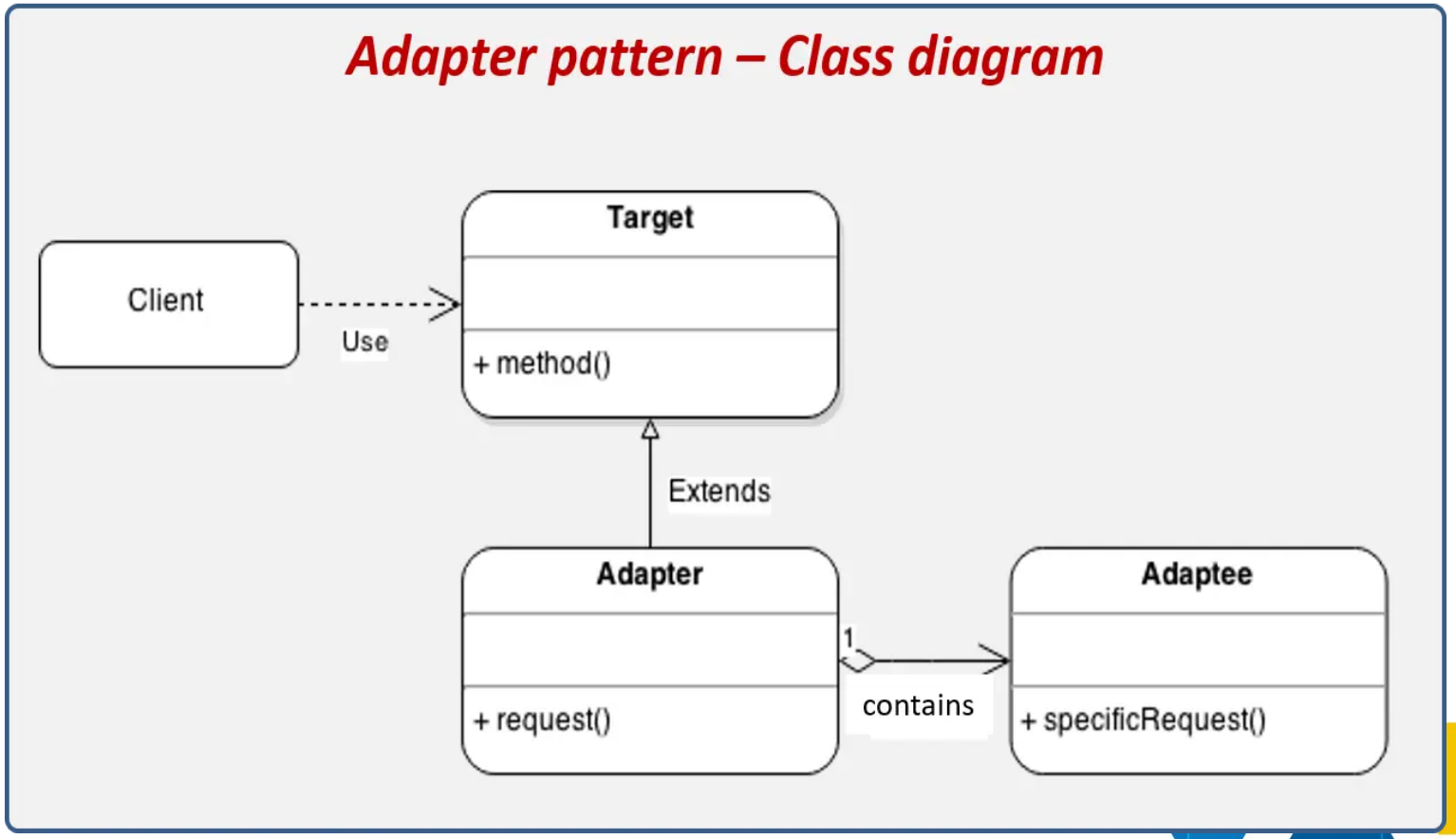
Patrones estructurales – Adapter

Client: Actor que interactúa con el Adapter.

Target: Interface que nos permitirá homogenizar la forma de trabajar con las interfaces incompatibles, esta interface es utilizada para crear los Adapter.

Adapter: Representa la implementación del Target, el cual tiene la responsabilidad de mediar entre el Client y el Adaptee. Oculta la forma de comunicarse con el Adaptee.

Adaptee: Representa la clase con interface incompatible.



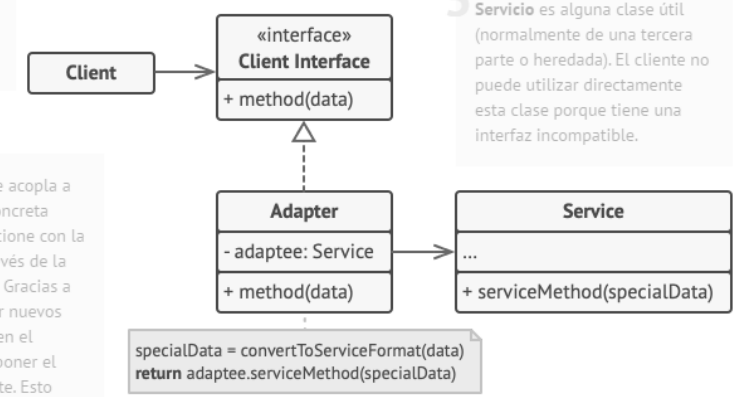
UNIVERSIDAD Popular del cesar

Patrones estructurales – Adapter

2 La **Interfaz con el Cliente** describe un protocolo que otras clases deben seguir para poder colaborar con el código cliente.

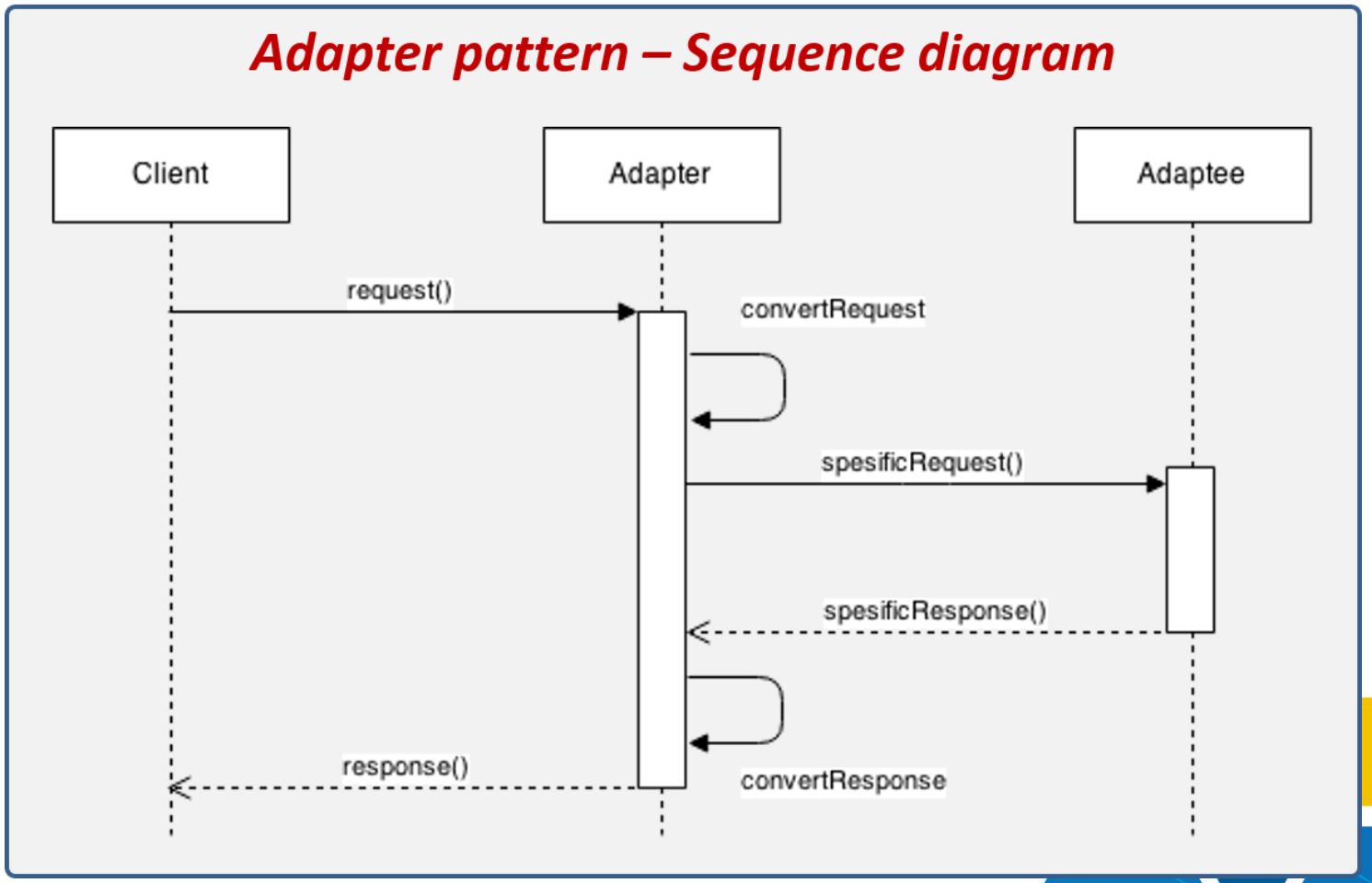
1 La clase **Cliente** contiene la lógica de negocio existente del programa.

3 **Servicio** es alguna clase útil (normalmente de una tercera parte o heredada). El cliente no puede utilizar directamente esta clase porque tiene una interfaz incompatible.



5 El código cliente no se acopla a la clase adaptadora concreta siempre y cuando funcione con la clase adaptadora a través de la interfaz con el cliente. Gracias a esto, puedes introducir nuevos tipos de adaptadores en el programa sin descomponer el código cliente existente. Esto puede resultar útil cuando la interfaz de la clase de servicio se cambia o sustituye, ya que puedes crear una nueva clase adaptadora sin cambiar el código cliente.

4 La clase **Adaptadora** es capaz de trabajar tanto con la clase cliente como con la clase de servicio: implementa la interfaz con el cliente, mientras envuelve el objeto de la clase de servicio. La clase adaptadora recibe llamadas del cliente a través de la interfaz de cliente y las traduce en llamadas al objeto envuelto de la clase de servicio, pero en un formato que pueda comprender.

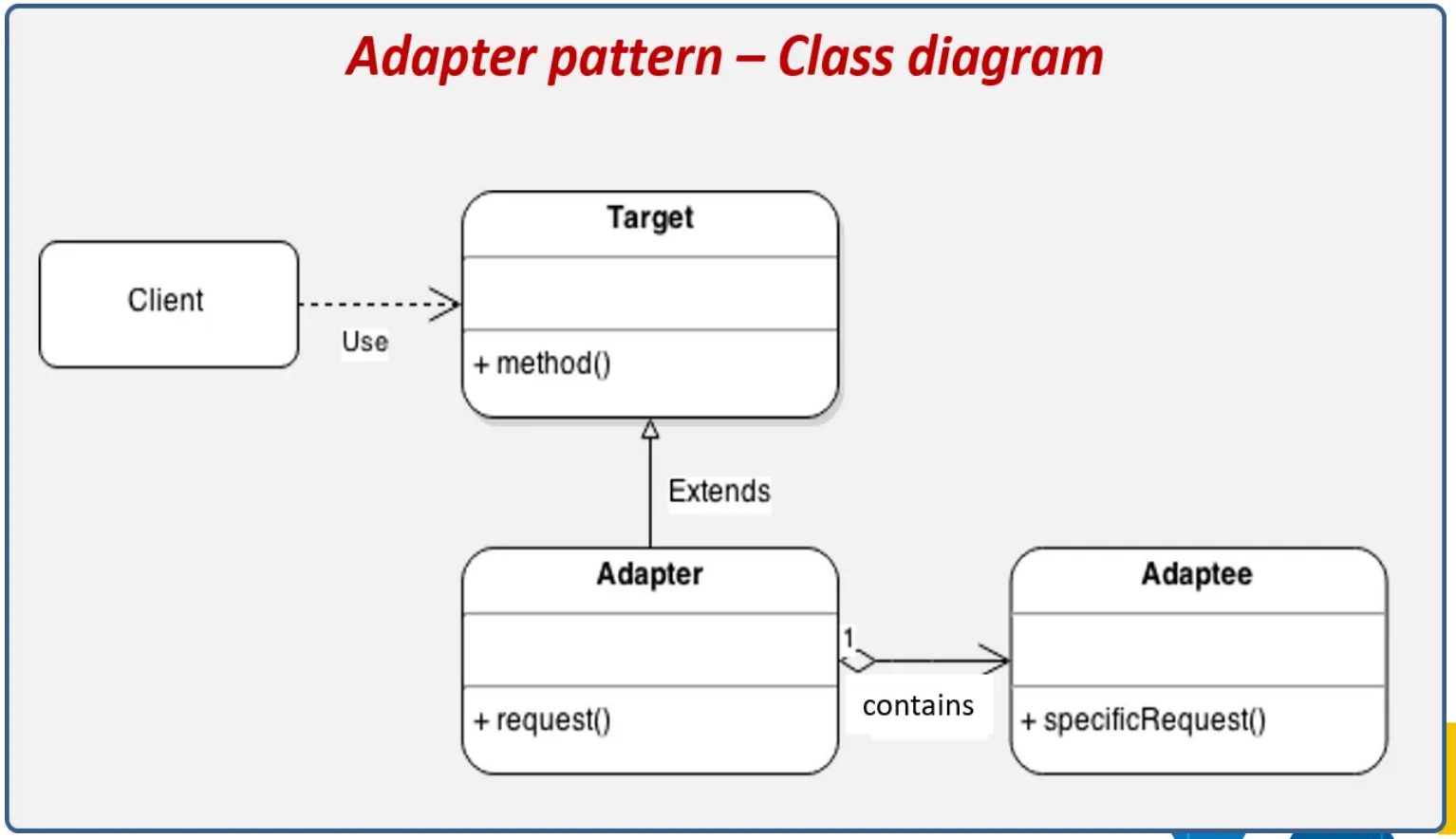


UNIVERSIDAD Popular del cesar

Patrones estructurales – Adapter

Cuando implementarlo:

- Utiliza la clase adaptadora cuando quieras usar una clase existente, pero cuya interfaz no sea compatible con el resto del código.
- Utiliza el patrón cuando quieras reutilizar varias subclases existentes que carezcan de alguna funcionalidad común que no pueda añadirse a la superclase.
- Cuando se quiera homogeneizar la forma de trabajar con interfaces incompatibles mediante una interface común



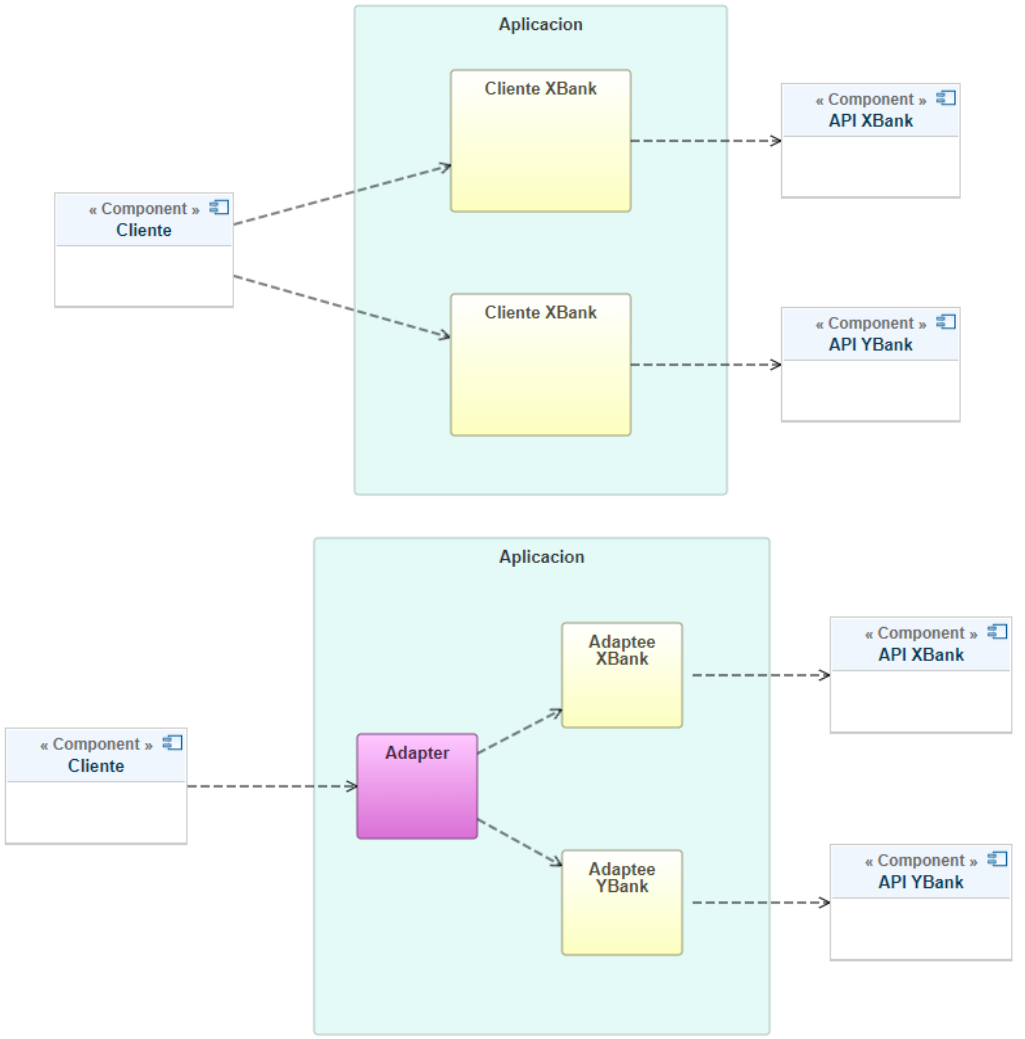
UNIVERSIDAD Popular del cesar

Patrones estructurales – Adapter

Ejemplo: Sistema bancario para prestamos personales.

Nuestro sistema utiliza las apis proporcionadas por los bancos para comunicarse con ellos, con el fin de validad si pueden prestar la cantidad solicitada por el cliente. La aplicación tendrá que validar con dos bancos (XBank y YBank) para verificar la información e informar al cliente si es posible prestarle el efectivo.

Cada banco proporciona sus propias apis para comunicarse con ellos, cada una en esencia requiere la misma información para hacer la solicitud, sin embargo, el nombre de clases, variables y tipos de datos son distintos entre cada api.



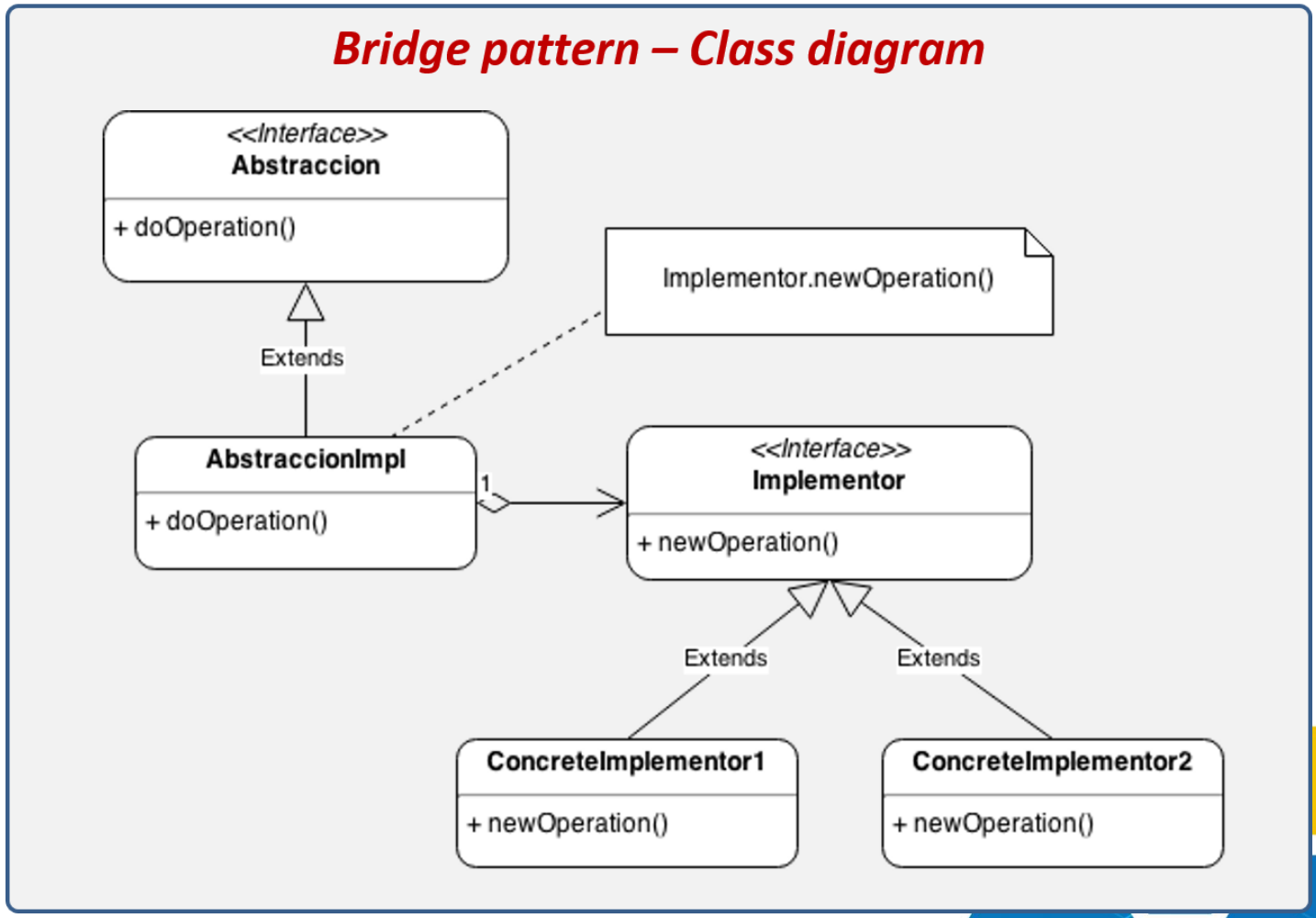
UNIVERSIDAD Popular del cesar

Patrones Estructurales – Bridge

Permite dividir una clase grande, o un grupo de clases estrechamente relacionadas, en dos jerarquías separadas (abstracción e implementación) que pueden desarrollarse independientemente la una de la otra.

Se desacopla una abstracción de su implementación para que puedan variar independientemente.

Muy útil cuando dos piezas de software están relacionadas directamente, sin embargo, existe una probabilidad de que una de ellas cambien y esto puede llevar a la necesidad de modificar la pieza del otro lado.



UNIVERSIDAD Popular del cesar

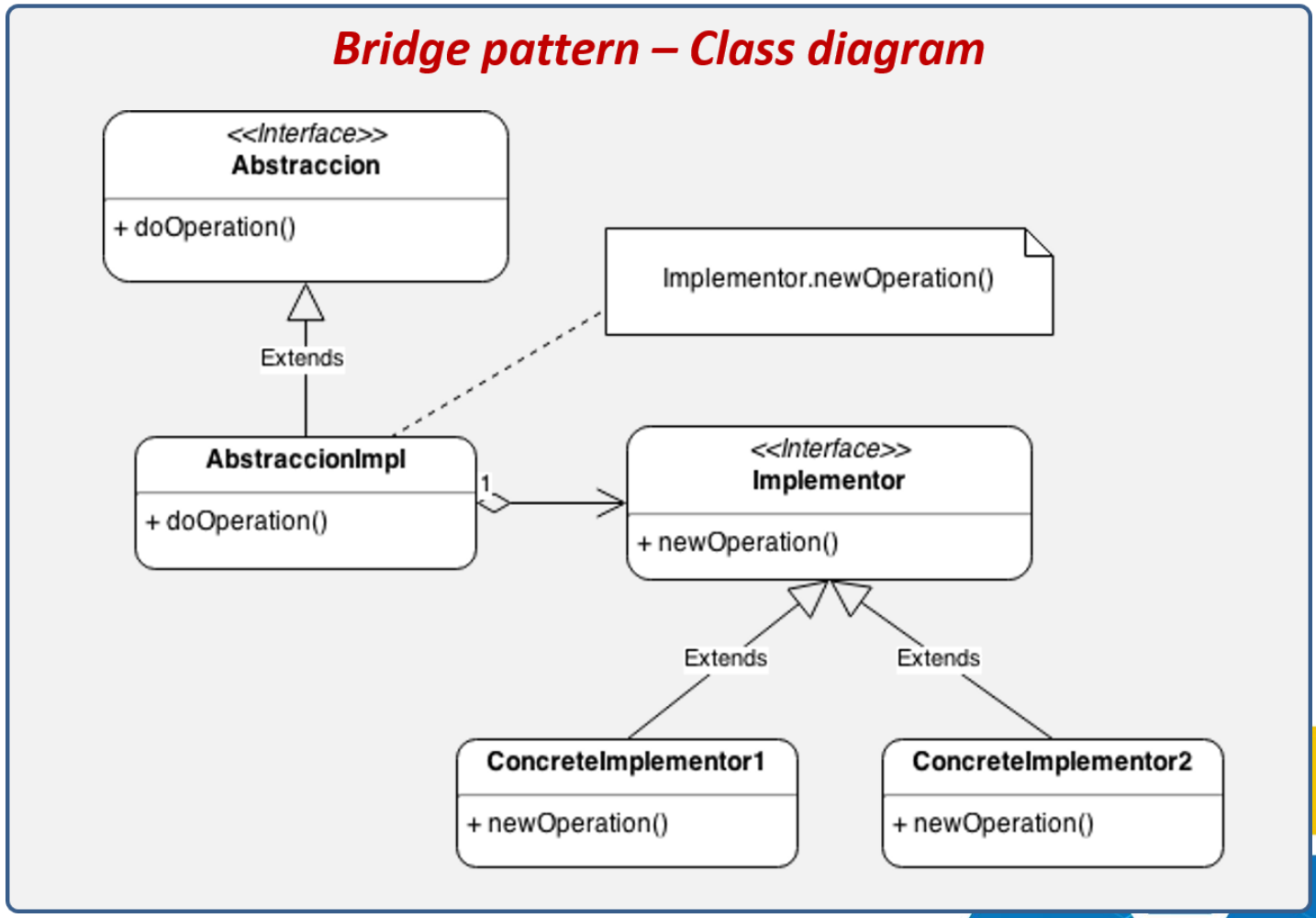
Patrones Estructurales – Bridge

Abstraction: Interface que define la estructura de la clase adaptadora o bridge.

AbstractionImpl: Clase utilizada como puente para desacoplar a Abstraction de Implementor, la clase hereda de Abstraction.

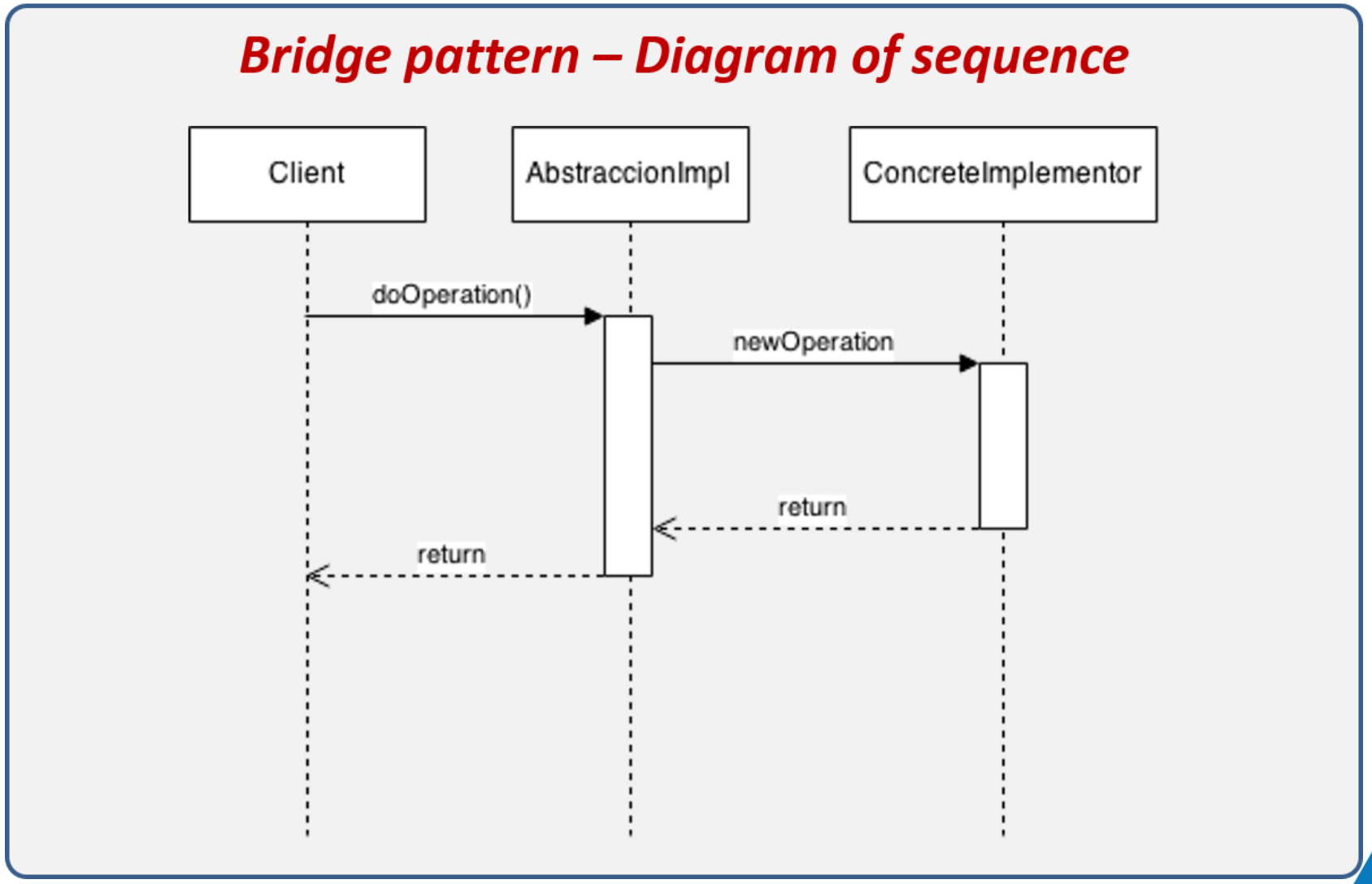
Implementor: Define una estructura de clase común para todas los ConcreteImplementor. Esta interface no es estrictamente requerida para implementar el patrón.

ConcreteImplementor: Conjunto de clases que heredan de Implementor y que son propensas a cambiar, es por esta razón que se opta por implementar el patrón de diseño Bridge.



UNIVERSIDAD Popular del cesar

Patrones Estructurales – Bridge

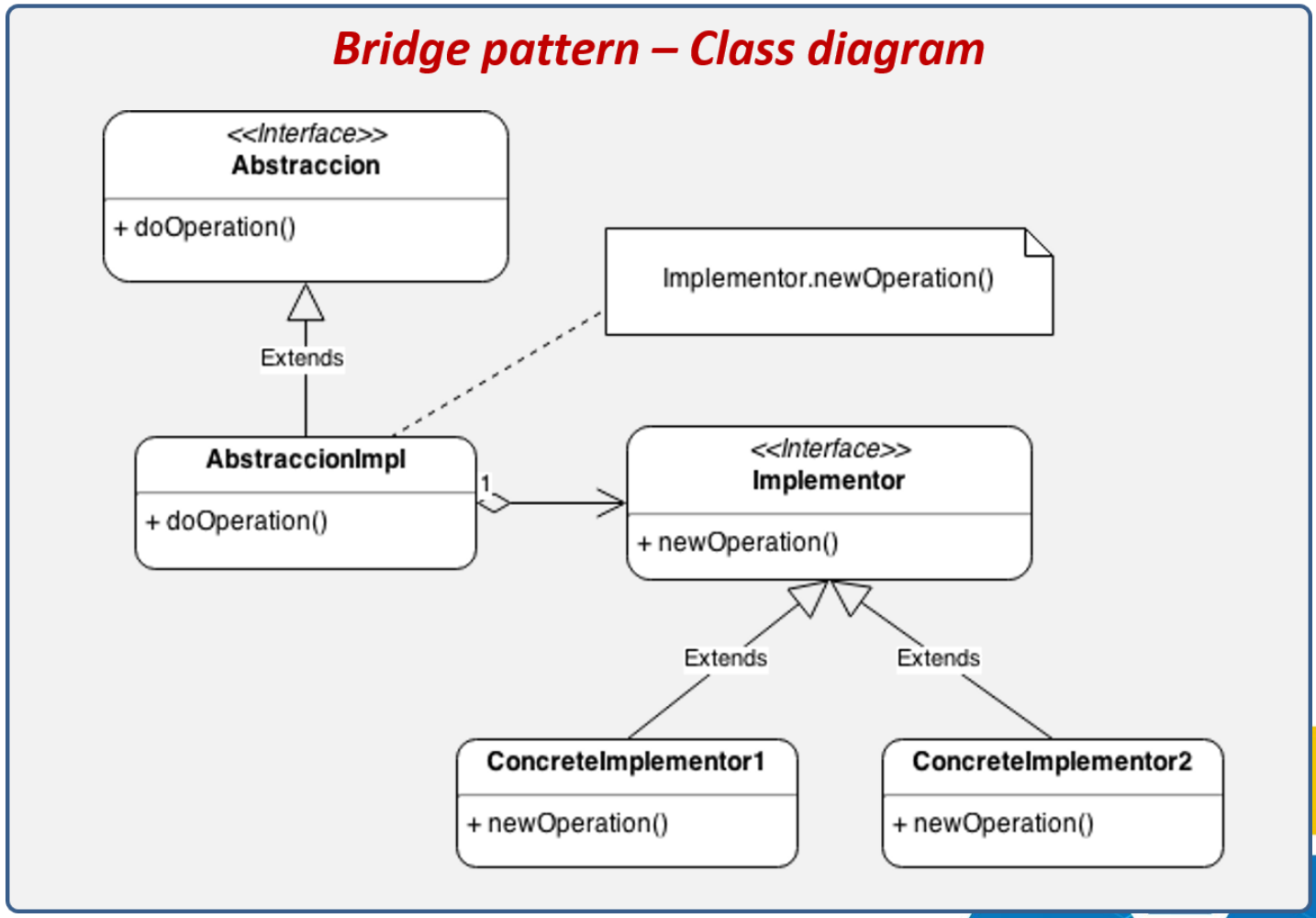


UNIVERSIDAD Popular del cesar

Patrones Estructurales – Bridge

Cuando implementarlo:

- Utiliza el patrón Bridge cuando quieras dividir y organizar una clase monolítica que tenga muchas variantes de una sola funcionalidad (por ejemplo, si la clase puede trabajar con diversos servidores de bases de datos).
- Utiliza el patrón cuando necesites extender una clase en varias dimensiones ortogonales (independientes).
- Utiliza el patrón Bridge cuando necesites poder cambiar implementaciones durante el tiempo de ejecución.
- Cuando existen clases fuertemente relacionadas entre si, pero que alguna de las dos es propensa a cambiar



UNIVERSIDAD Popular del cesar

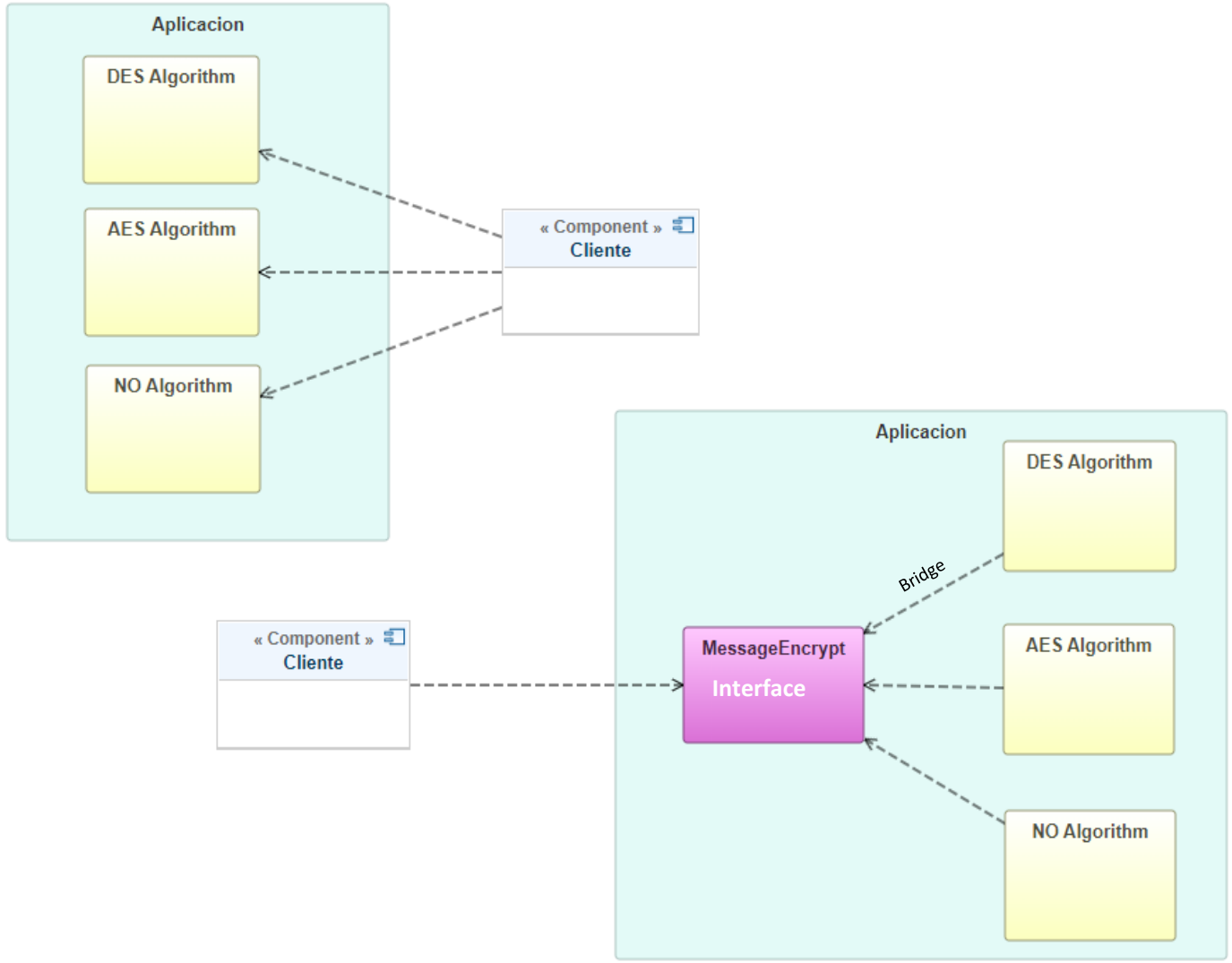
Patrones Estructurales – Bridge

Ejemplo: Programa que se comunica con otra aplicación

Dentro de los requerimientos se hace énfasis en la confidencialidad de la información que viaja, por lo cual se requiere que se encripte la información antes de enviarla.

Sin embargo, los usuarios de la aplicación destino no han definido exactamente cual será el método de encriptamiento con el cual se deben enviar los mensajes.

Por lo cual, han pedido un componente que se versátil, que permita cambiar el método de encriptamiento sin que eso afecte el componente desarrollado.



UNIVERSIDAD Popular del cesar

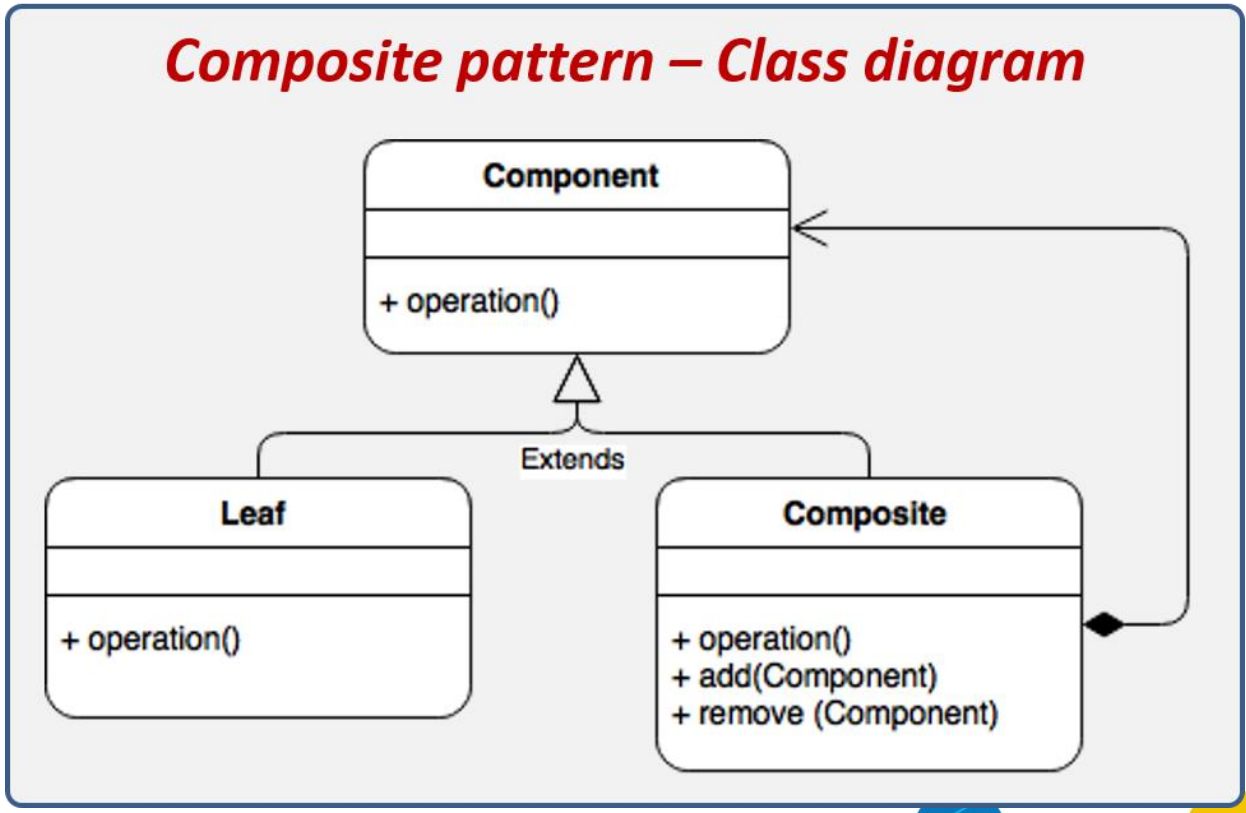
Patrones Estructurales – Composite

Composite es un patrón de diseño estructural que te permite componer objetos en estructuras de árbol y trabajar con esas estructuras como si fueran objetos individuales.

Compone objetos en estructuras de árbol para representar jerarquías de parte-todo. Permite que los clientes traten de manera uniforme a los objetos individuales y a los compuestos

El uso del patrón Composite sólo tiene sentido cuando el modelo central de tu aplicación puede representarse en forma de árbol.

Muy útil cuando queremos manipular un conjunto de elementos muy parecidos entre si de una forma uniforme



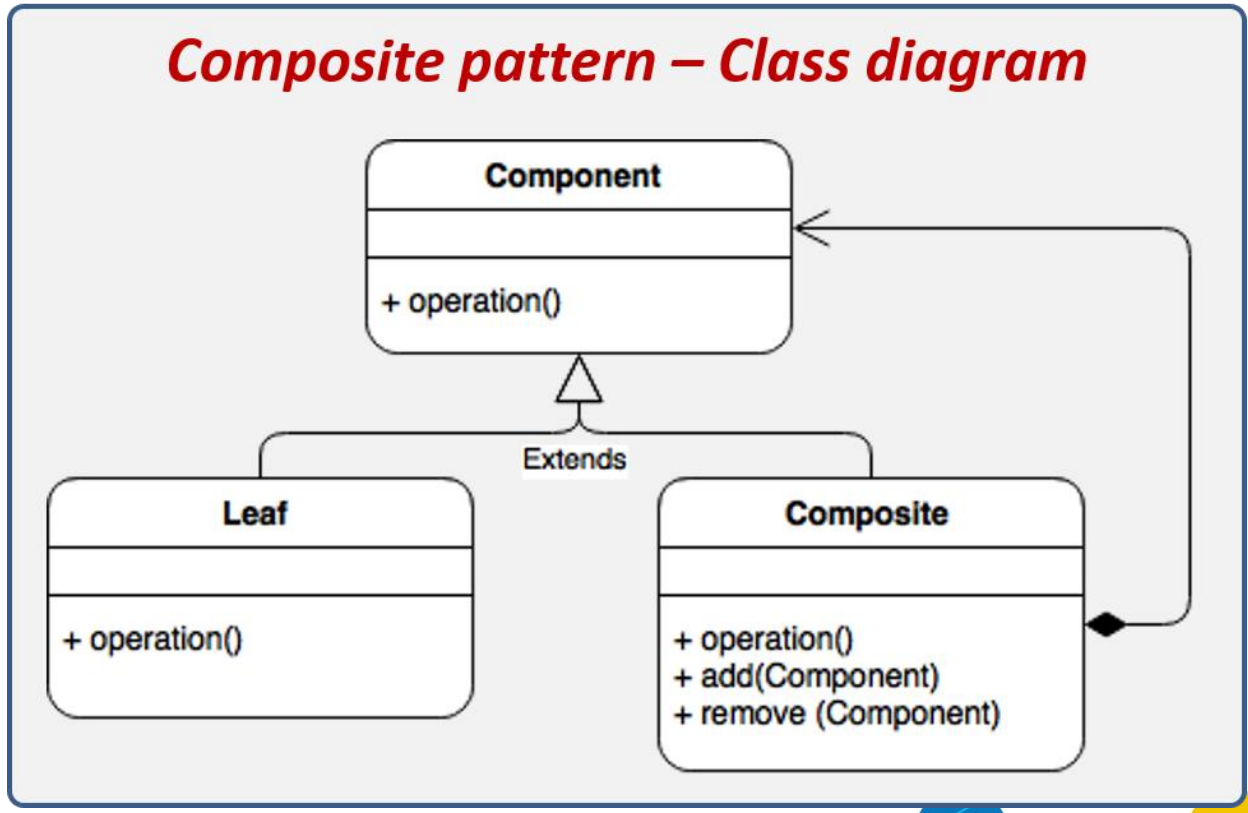
UNIVERSIDAD Popular del cesar

Patrones Estructurales – Composite

Component: Generalmente es una interface o clase abstracta que tiene las operaciones mínimas que serán utilizadas, este componente deberá ser extendido por los otros dos componentes Leaf y Composite.

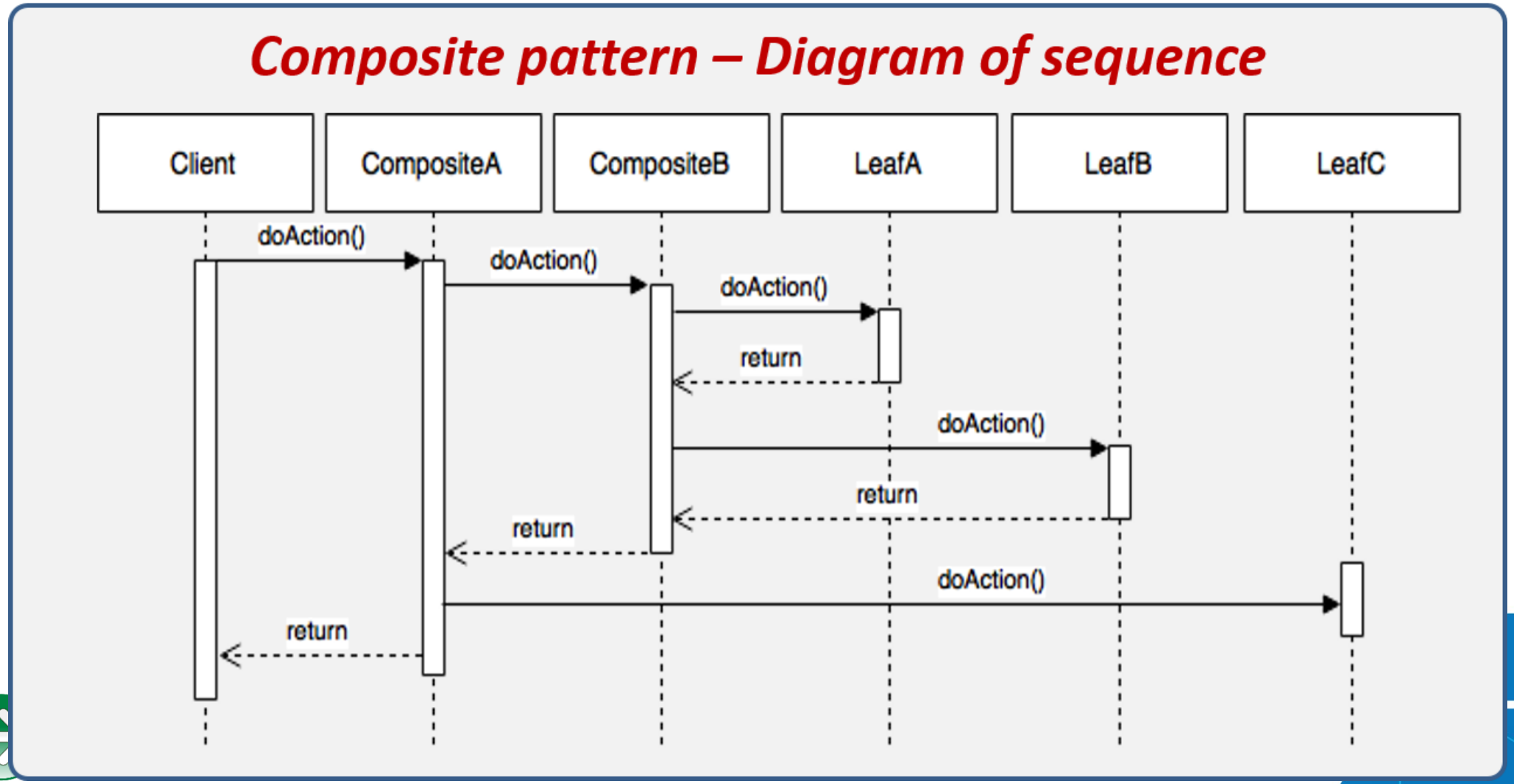
Leaf: El leaf u hoja, representa la parte más simple o pequeña de toda la estructura y ésta hereda de Component. Leaf recibe su nombre de la teoría de árboles, donde se le nombra así a todo nodo que no tiene descendencia, en este caso son clases simple que no están compuestas de otras.

Composite: Este componente es el que le da vida al patrón de diseño ya que este objeto está conformado por un conjunto de Component y Leaf. En teoría de árboles este componente representaría una rama.



UNIVERSIDAD Popular del cesar

Patrones Estructurales – Composite

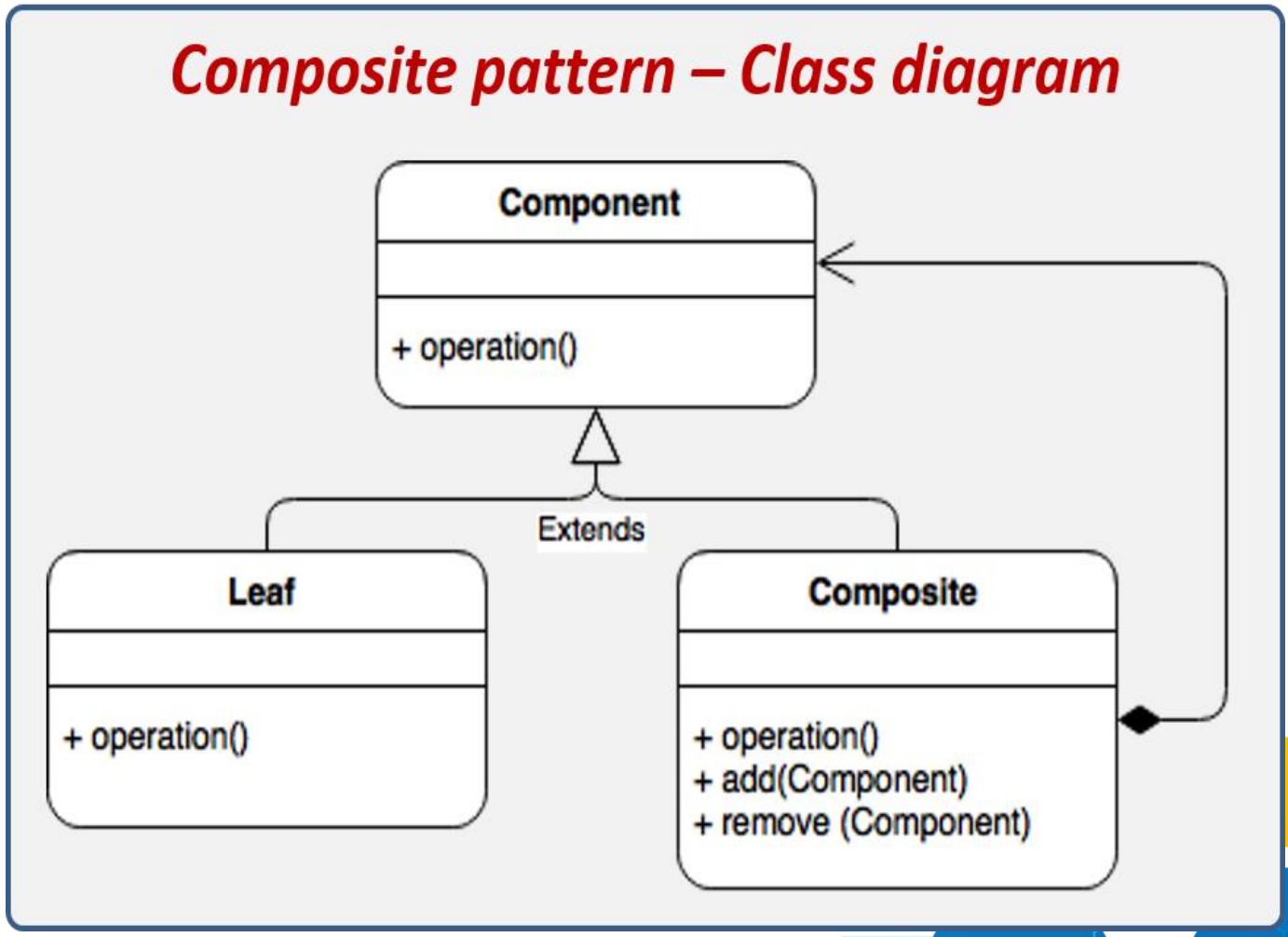


UNIVERSIDAD Popular del cesar

Patrones Estructurales – Composite

Cuando implementarlo:

- Utiliza el patrón Composite cuando tengas que implementar una estructura de objetos con forma de árbol.
- Cuando detectamos estructuras complejas que podrían dividirse en elementos mas pequeño, muy relacionados entre si, en funcionamiento y estructura
- Utiliza el patrón cuando quieras que el código cliente trate elementos simples y complejos de la misma forma.



UNIVERSIDAD Popular del cesar

Patrones Estructurales – Composite

Ejemplo: sistema punto de venta

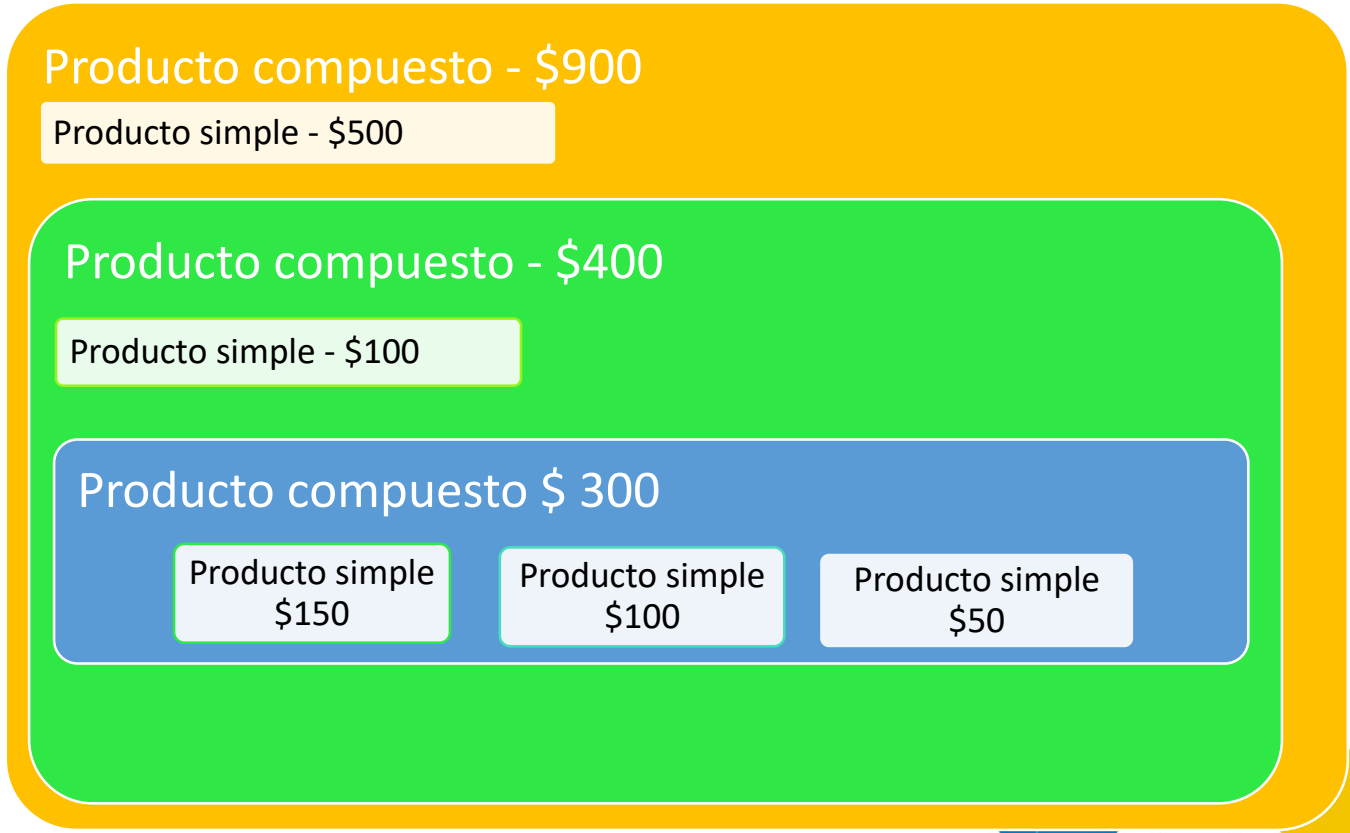
Se mantiene un sistema de punto de ventas, en el cual se le puede vender al cliente una seria de productos que pueden ser simples o paquetes.

El sistema debe permitir crear ordenes de ventas que estarán compuesta por uno o muchos productos.

El precio de los productos deberá ser calculado mediante la suma del precio de todos los productos internos.

Un paquete es creado mediante una serie de productos simples y compuestos.

El precio de un paquete es la suma de todos los productos simples que contenga.



UNIVERSIDAD Popular del cesar

Patrones Estructurales – Composite

Ejemplo: sistema punto de venta

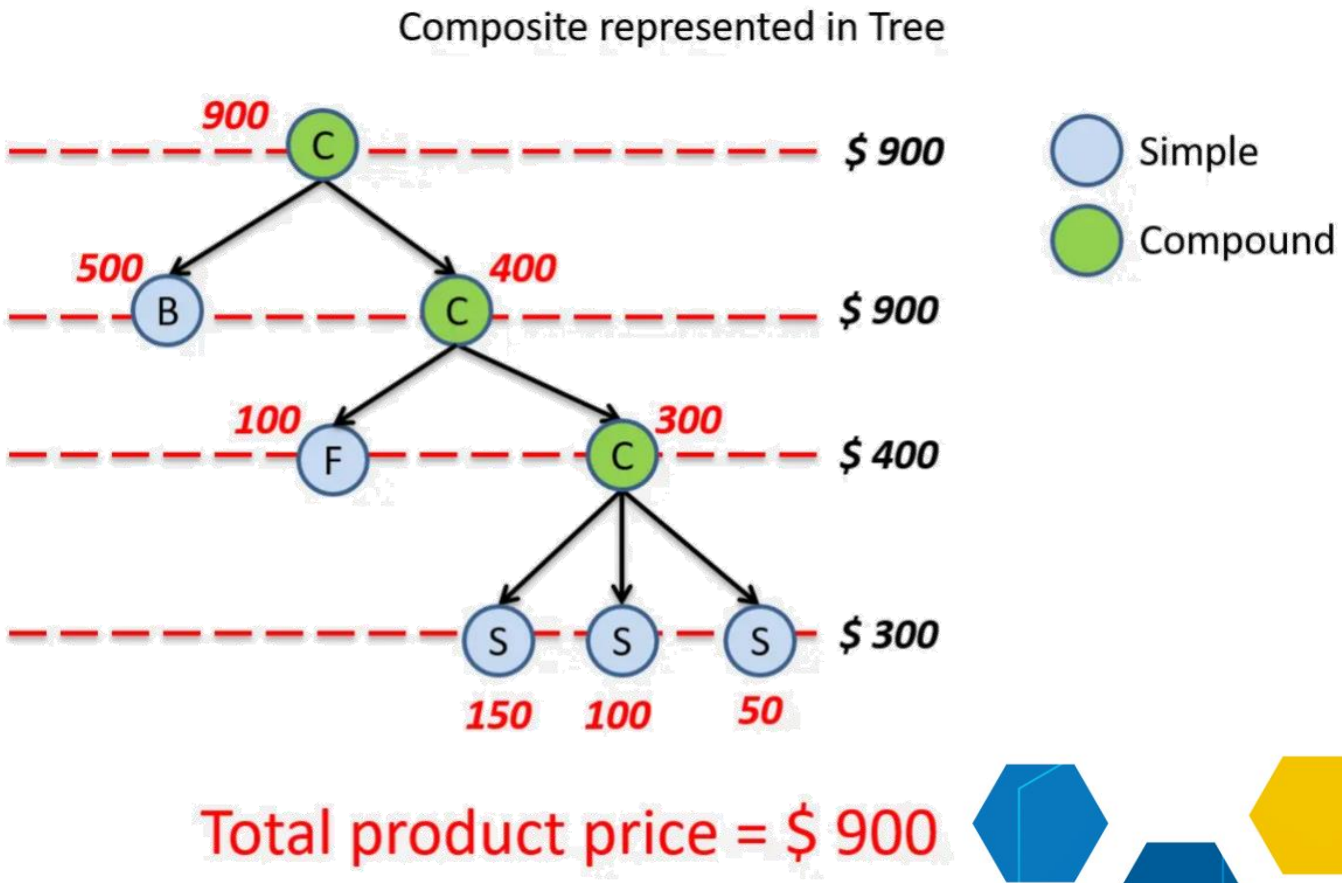
Se mantiene un sistema de punto de ventas, en el cual se le puede vender al cliente una seria de productos que pueden ser simples o paquetes.

El sistema debe permitir crear ordenes de ventas que estarán compuesta por uno o muchos productos.

El precio de los productos deberá ser calculado mediante la suma del precio de todos los productos internos.

Un paquete es creado mediante una serie de productos simples y compuestos.

El precio de un paquete es la suma de todos los productos simples que contenga.



UNIVERSIDAD Popular del cesar

Patrones Estructurales – Composite

Ejemplo: sistema punto de venta

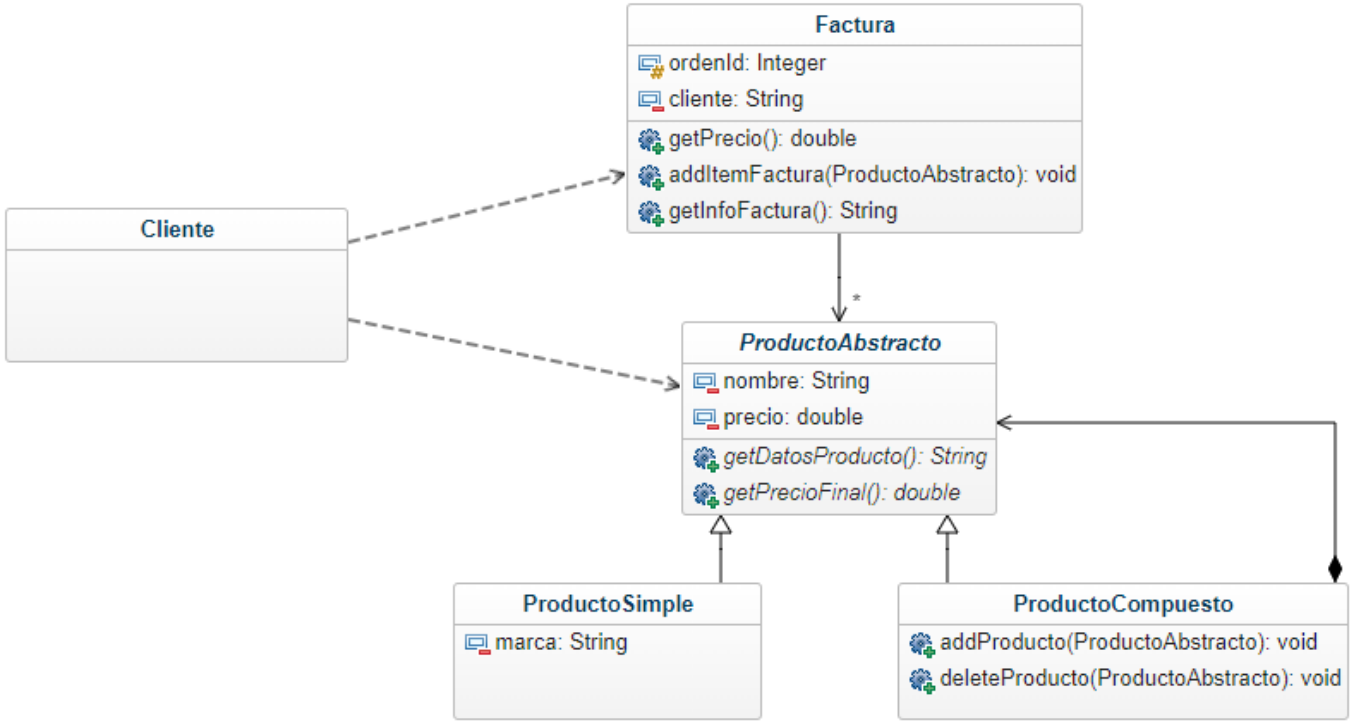
Se mantiene un sistema de punto de ventas, en el cual se le puede vender al cliente una seria de productos que pueden ser simples o paquetes.

El sistema debe permitir crear ordenes de ventas que estarán compuesta por uno o muchos productos.

El precio de los productos deberá ser calculado mediante la suma del precio de todos los productos internos.

Un paquete es creado mediante una serie de productos simples y compuestos.

El precio de un paquete es la suma de todos los productos simples que contenga.



UNIVERSIDAD Popular del cesar

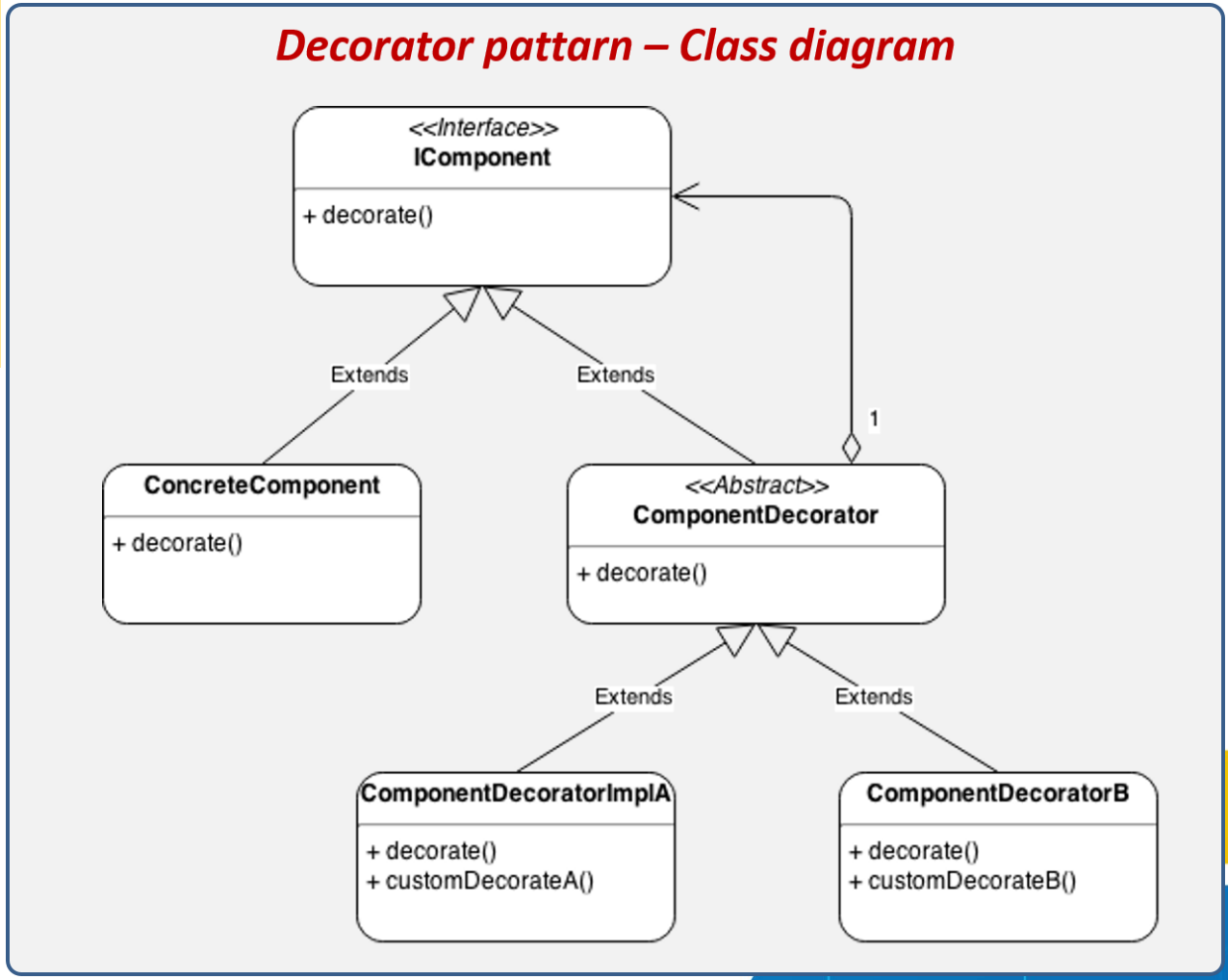
Patrones Estructurales – Decorator (Wrapper)

Decorator es un patrón de diseño estructural que te permite añadir funcionalidades a objetos colocándolos dentro de objetos encapsuladores especiales que contienen estas funcionalidades.

Asigna responsabilidades adicionales a un objeto dinámicamente, proporcionando una alternativa flexible a la herencia para extender la funcionalidad.

Permite agregarle nuevas funcionalidades a un objeto existente sin alterar su estructura.

Diseñado para solucionar problemas donde la jeraquia con subclasificación no puede ser aplicada, o se requiere de un gran impacto en todas las clases de la jeraquia para lograr el comportamiento deseado.



UNIVERSIDAD Popular del cesar

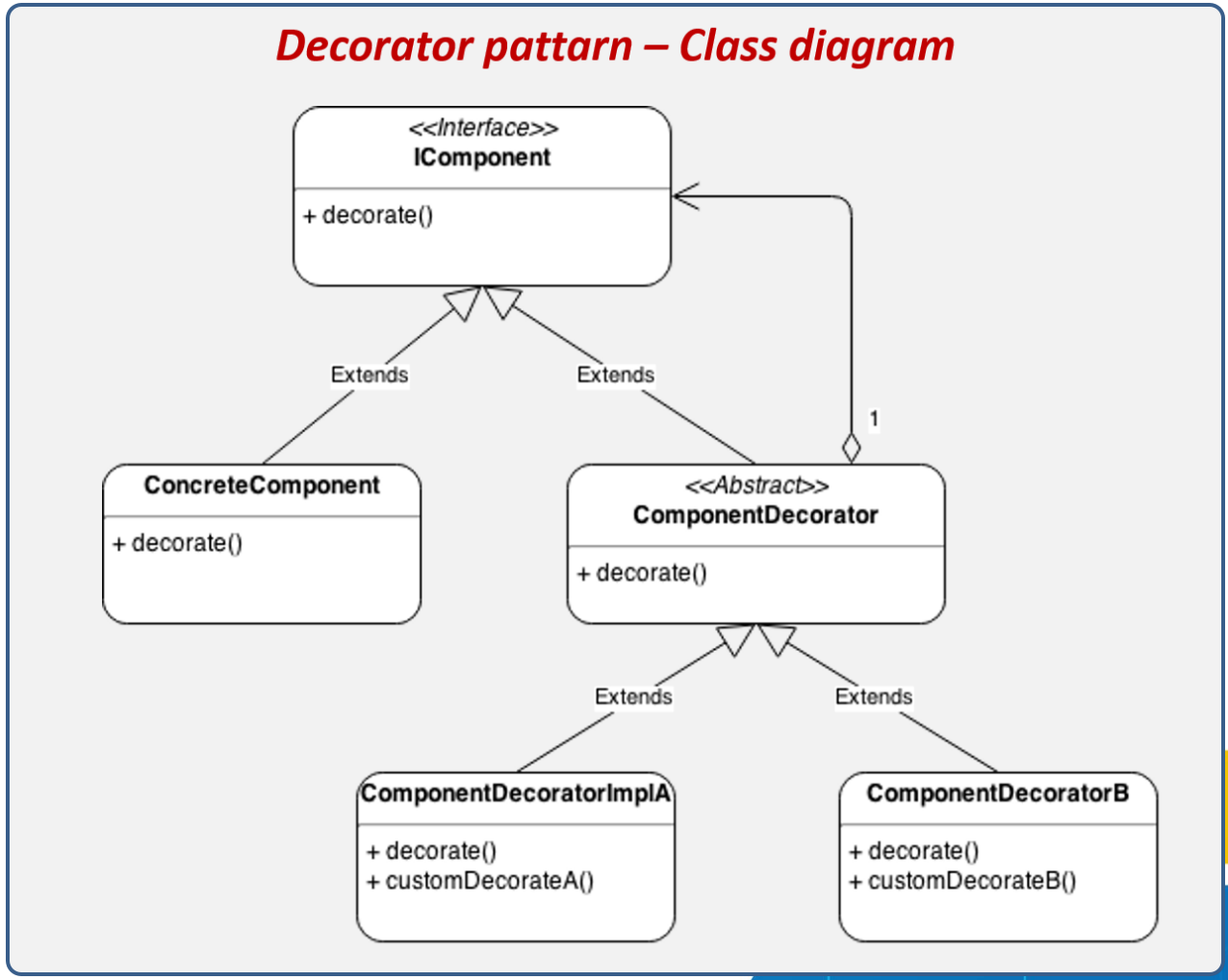
Patrones Estructurales – Decorator (Wrapper)

IComponent: Interface que define la estructura minina del componente o componentes que pueden ser decorados.

ConcreteComponent: Implementación de IComponent y define un objeto concreto que puede ser decorado.

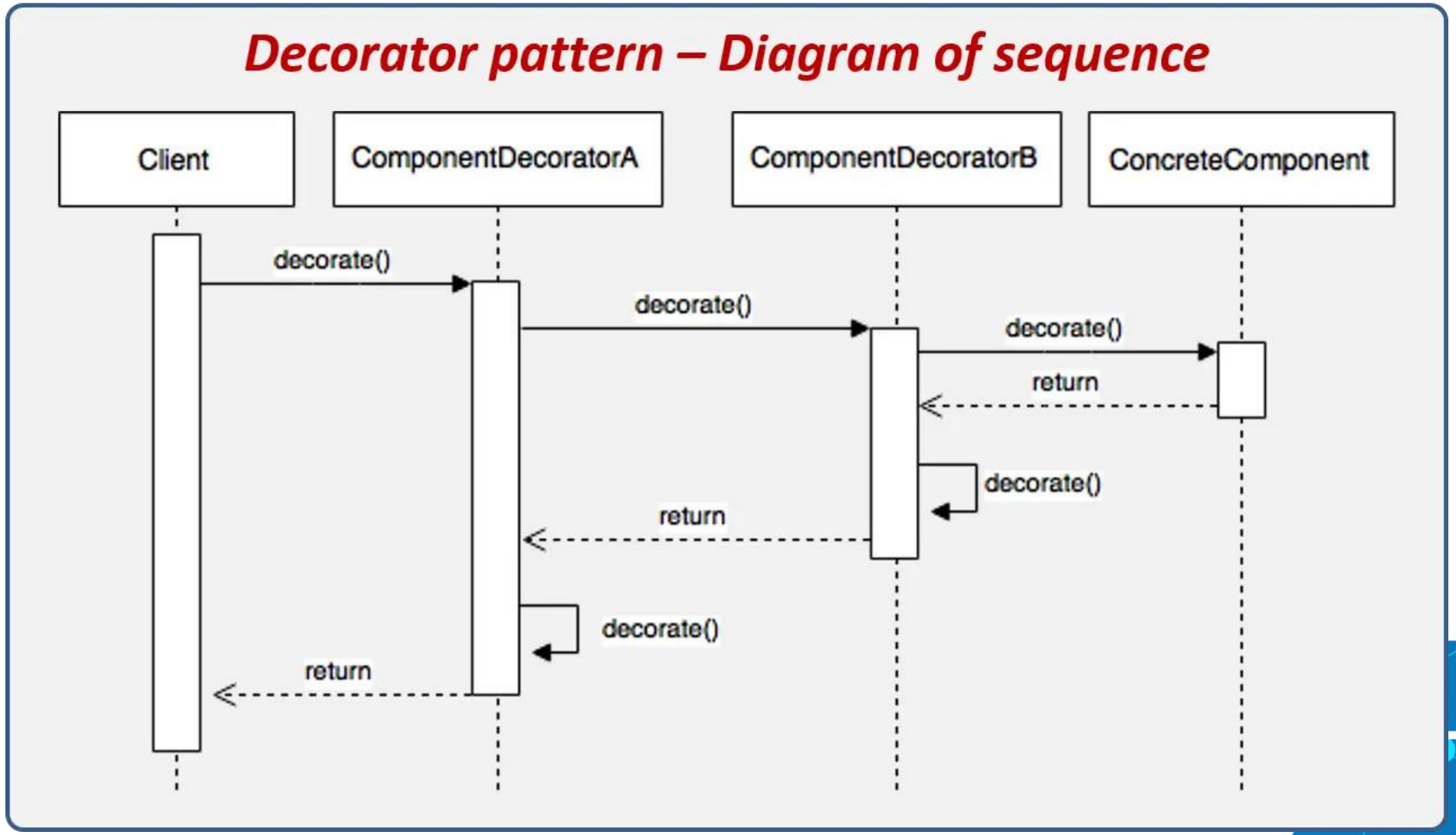
ComponentDecorator: Por lo general es una clase abstracta que define la estructura mínima de un Decorador, el cual mínimamente deben de heredar de IComponent y contener alguna subclase de IComponent al cual decorarán.

ComponentDecoratorImpl: Representan todos los decoradores concretos que heredan de ComponentDecorator.



UNIVERSIDAD Popular del cesar

Patrones Estructurales – Decorator (Wrapper)



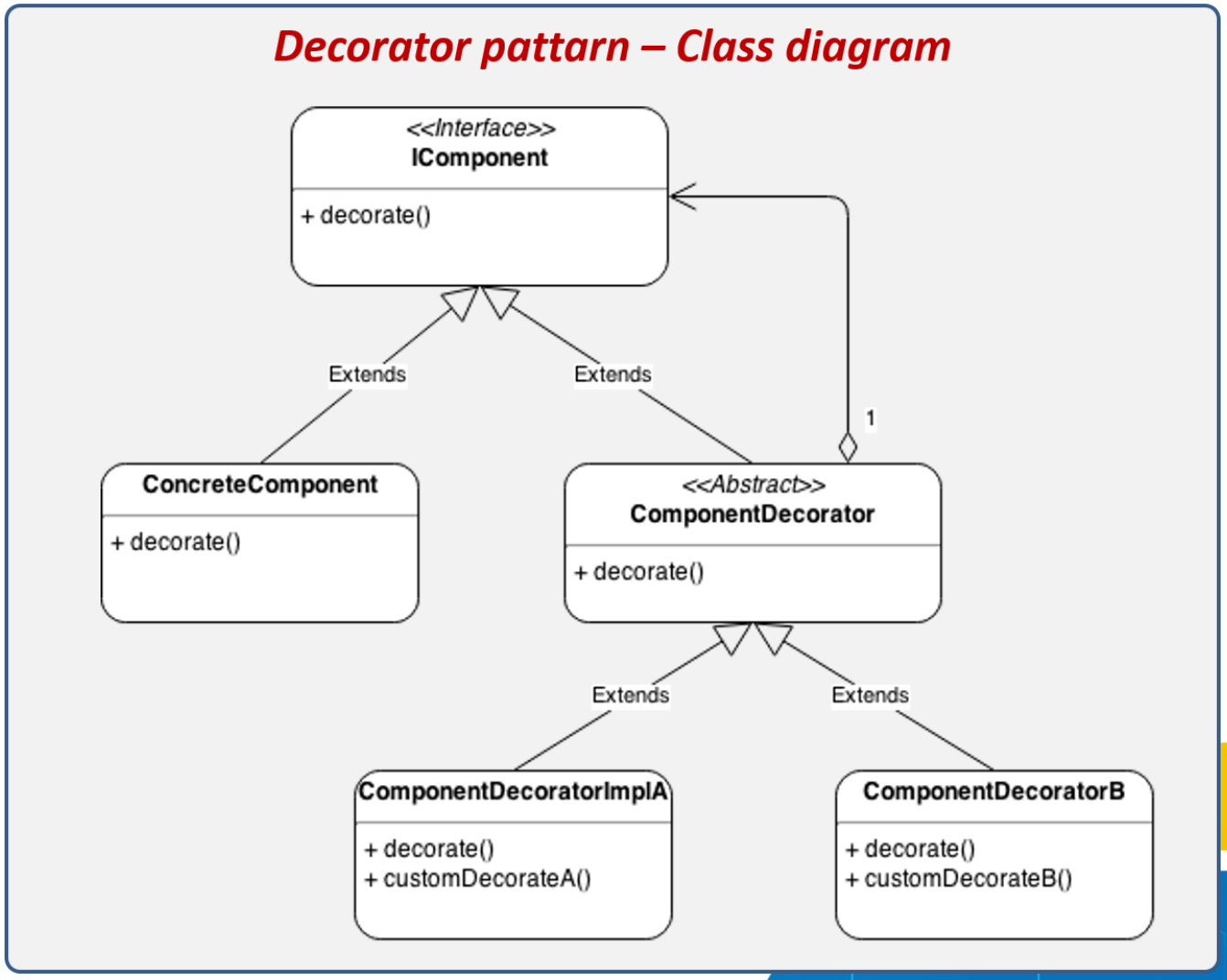
UNIVERSIDAD Popular del cesar

Patrones Estructurales – Decorator

Cuando implementarlo:

- Cuando necesitamos agregar dinámicamente una nueva funcionalidad sobre un objeto y esta funcionalidad de agrega sobre capas de funcionalidades previas.

- Utiliza el patrón cuando resulte extraño o no sea posible extender el comportamiento de un objeto utilizando la herencia.



UNIVERSIDAD Popular del cesar

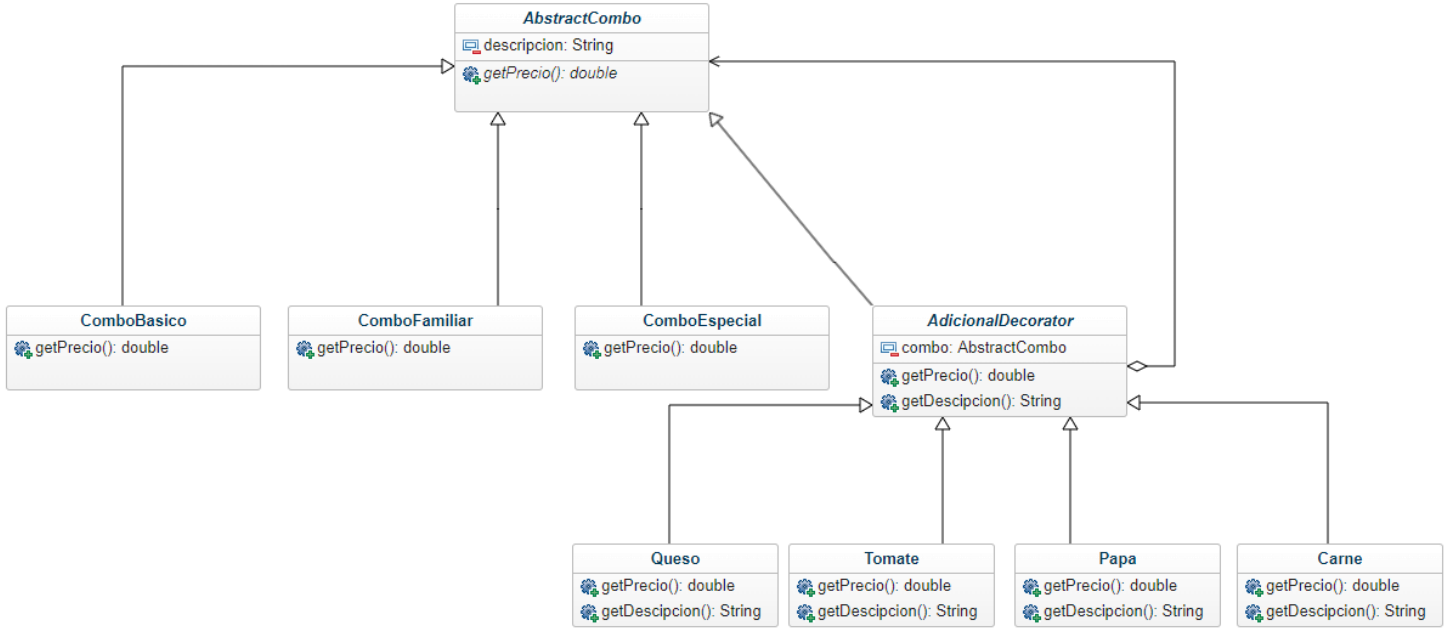
Patrones Estructurales – Decorator

Ejemplo: un restaurante

Un restaurante de comidas rápidas ofrece 3 tipos de combos (Combo Básico, Combo Familiar, Combo Especial) cada combo tiene características diferentes en cuanto a cantidad, porciones, salsas entre otros.

El restaurante también ofrece la posibilidad de aumentar el pedido mediante diferentes porciones adicionales (Tomate, Papas, Carne, Queso).

Se desea crear un sistema de pedidos que permita al usuario seleccionar el combo deseado, así como armar su propio pedido con las porciones adicionales, el sistema deberá informar sobre el pedido del usuario y el valor total del mismo.



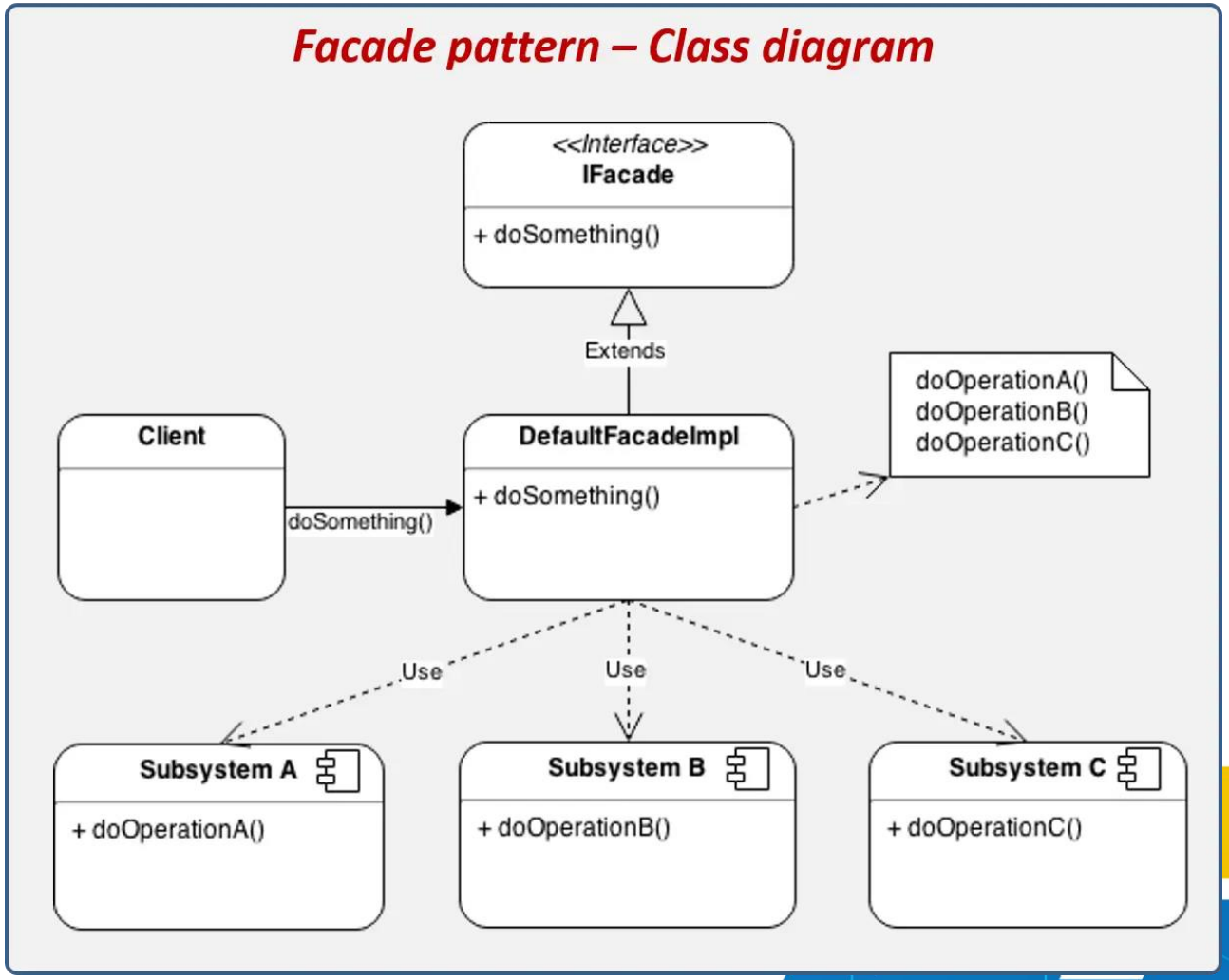
UNIVERSIDAD Popular del cesar

Patrones Estructurales – Facade

Facade es un patrón de diseño estructural que proporciona una interfaz simplificada a una biblioteca, un framework o cualquier otro grupo complejo de clases.

Es una buena estrategia cuando se requiere interactuar con varios subsistemas para realizar una operación concreta ya que necesita tener el conocimiento técnico y funcional para saber que operaciones de cada subsistema tenemos que ejecutar y en que orden.

Permite ocultar la complejidad de interactuar con un conjunto de subsistemas proporcionando una interface de alto nivel, que se encarga de realizar la comunicación con todos los subsistemas necesarios.



UNIVERSIDAD Popular del cesar

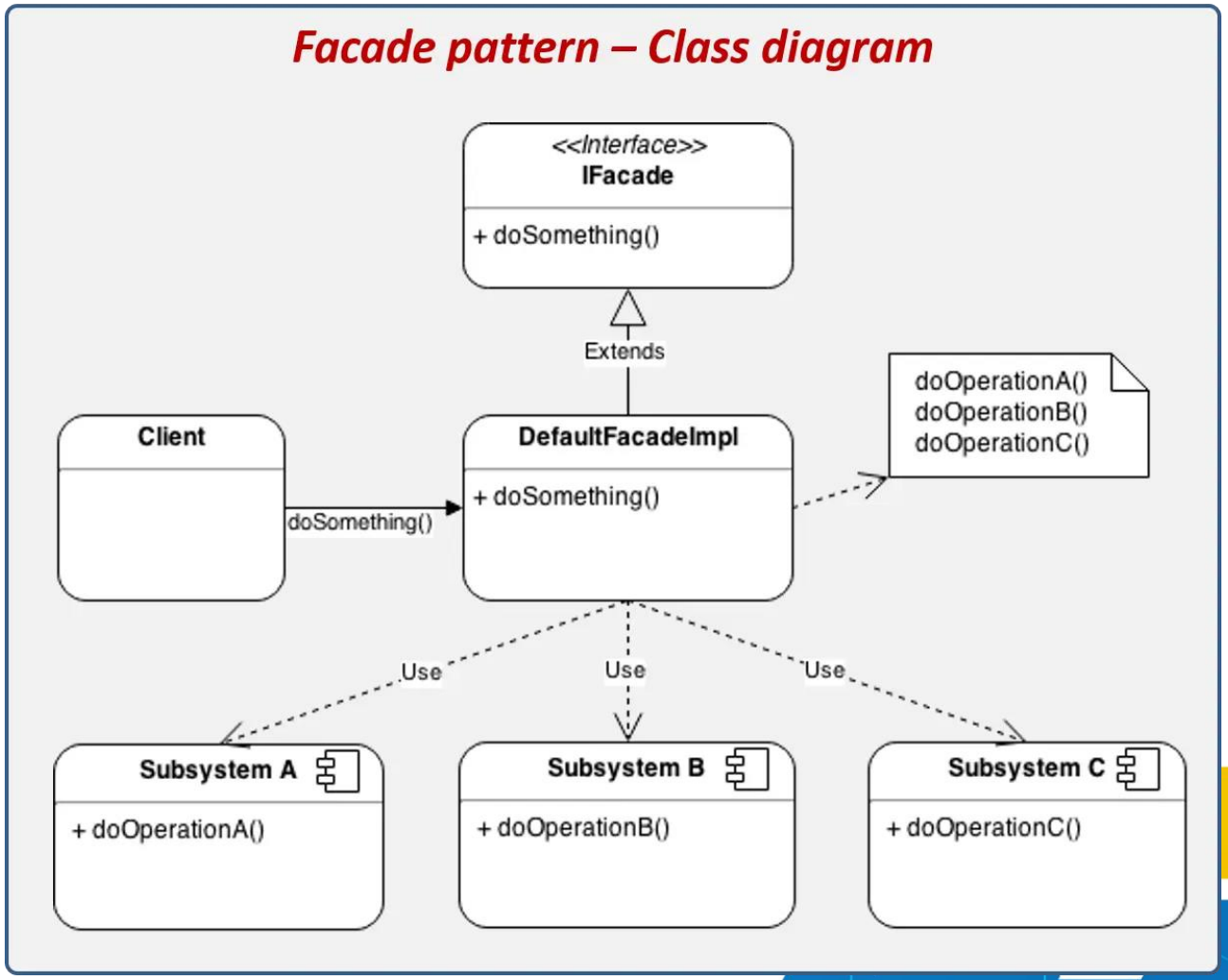
Patrones Estructurales – Facade

IFacade: Proporciona una interface de alto nivel que oculta la complejidad de interactuar con varios sistemas para realizar una operación.

Client : Sistema o evento que interactúa con la fachada.

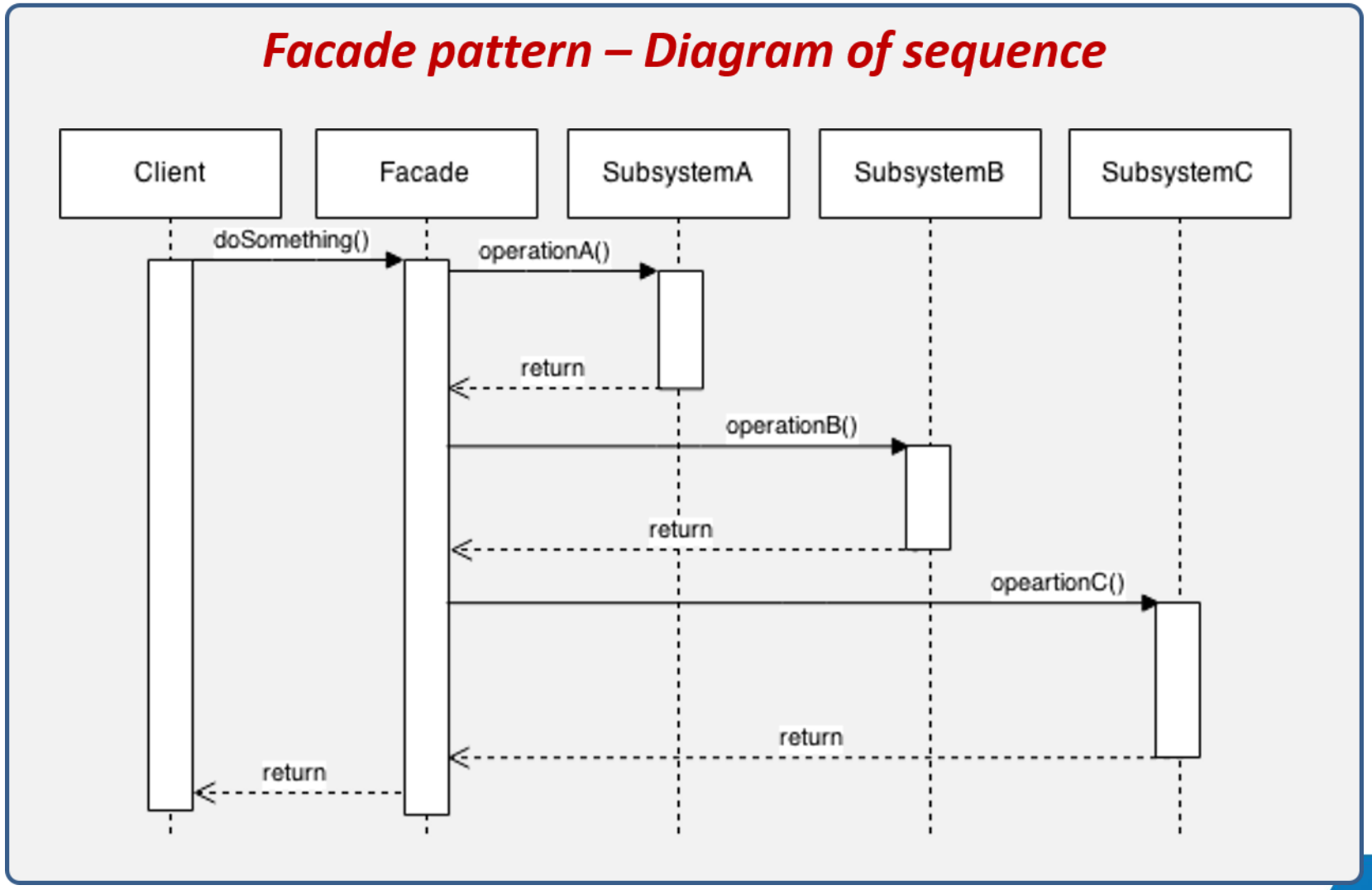
DefaultFacadeImpl: Representa la implementación de IFacade y se encarga de comunicarse con todos los subsistemas.

Subsystems: Representa módulos o subsistemas que exponen interfaces para comunicarse con ellos.



UNIVERSIDAD Popular del cesar

Patrones Estructurales – Facade

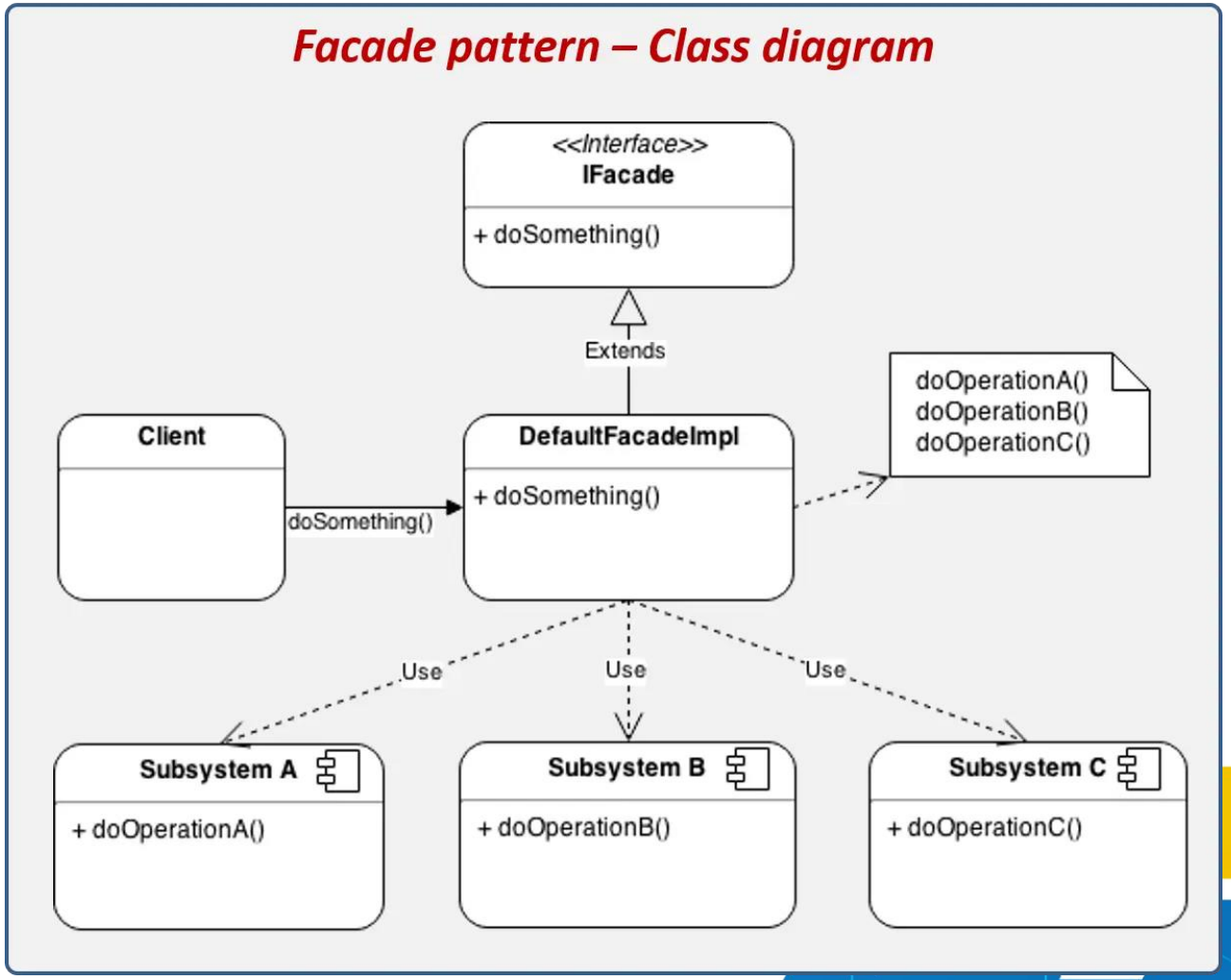


UNIVERSIDAD Popular del cesar

Patrones Estructurales – Facade

Cuando implementarlo:

- Cuando interactuar con un conjunto de subsistemas es complicado, debido a que es necesario conocer los objetos necesarios para tener una interacción reciproca con cada sistema
- Cuando queremos construir interfaces de alto nivel para nuestros usuarios
- Utiliza el patrón Facade cuando necesites una interfaz limitada pero directa a un subsistema complejo.
- Utiliza el patrón Facade cuando quieras estructurar un subsistema en capas.



UNIVERSIDAD Popular del cesar

Patrones Estructurales – Facade

Ejemplo: Portal de pagos web

- Una empresa tiene un portal web desde el cual brinda varios servicios a sus cliente y desea agregar la funcionalidad de pagos en línea.
- Esta funcionalidad no solo implica hacer el cobro, si no, que tenemos que afectar varios subsistemas para que el pago sea aplicado correctamente y el cliente sea notificado que el pago se realizó exitosamente.
- La empresa ya cuenta con algunos subsistemas con los cuales será necesario interactuar.

- **BankSystem:** sistema de bancos que permite realizar los cargos a los clientes por medio de los datos de tarjeta debido o crédito.
- **BillingSystem:** sistema de facturación que tiene el detalle del saldo del cliente
- **CRMSystem:** sistema que gestiona la información del cliente
- **EmailSystem:** sistema para envío de correos electrónicos

Pasos para realizar un pago:

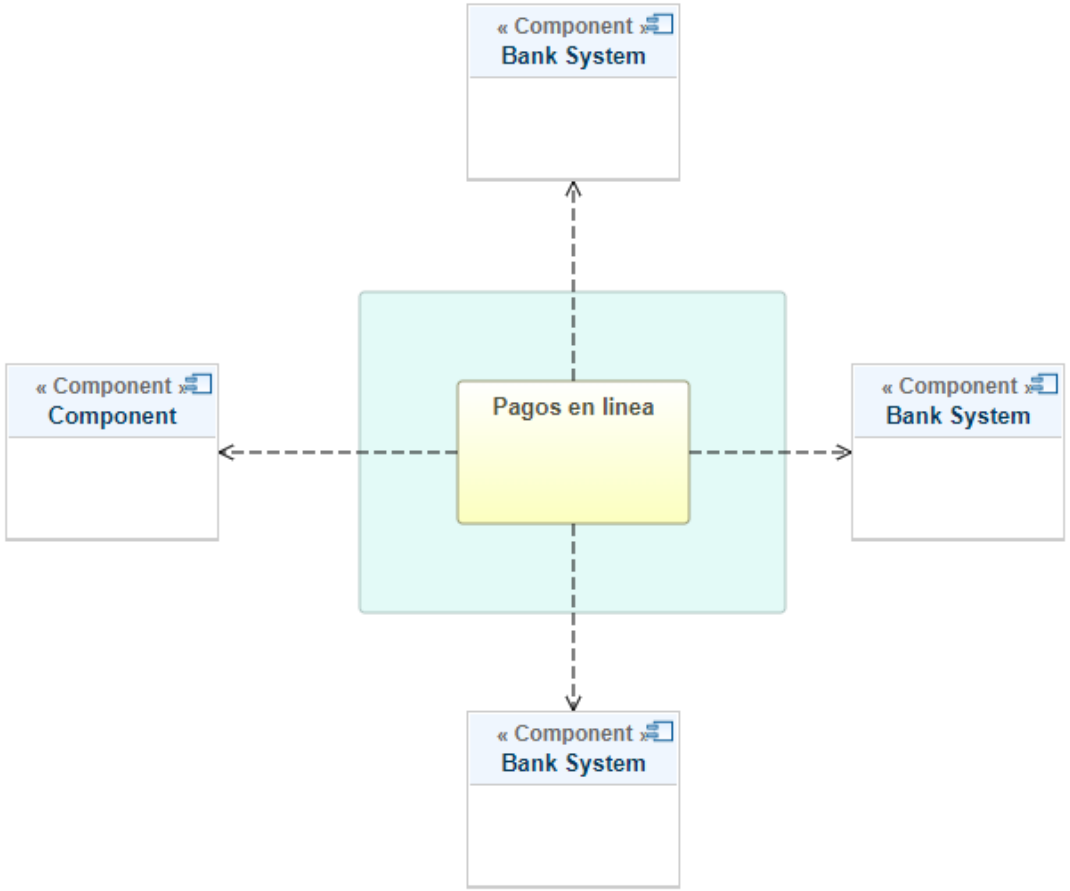
1. Validación de bines validos en el BankSystem
2. Validar y aplicar pago en BillingSystem
3. Actualizar status del cliente en CRMSystem
4. Notificar al cliente del pago realizado

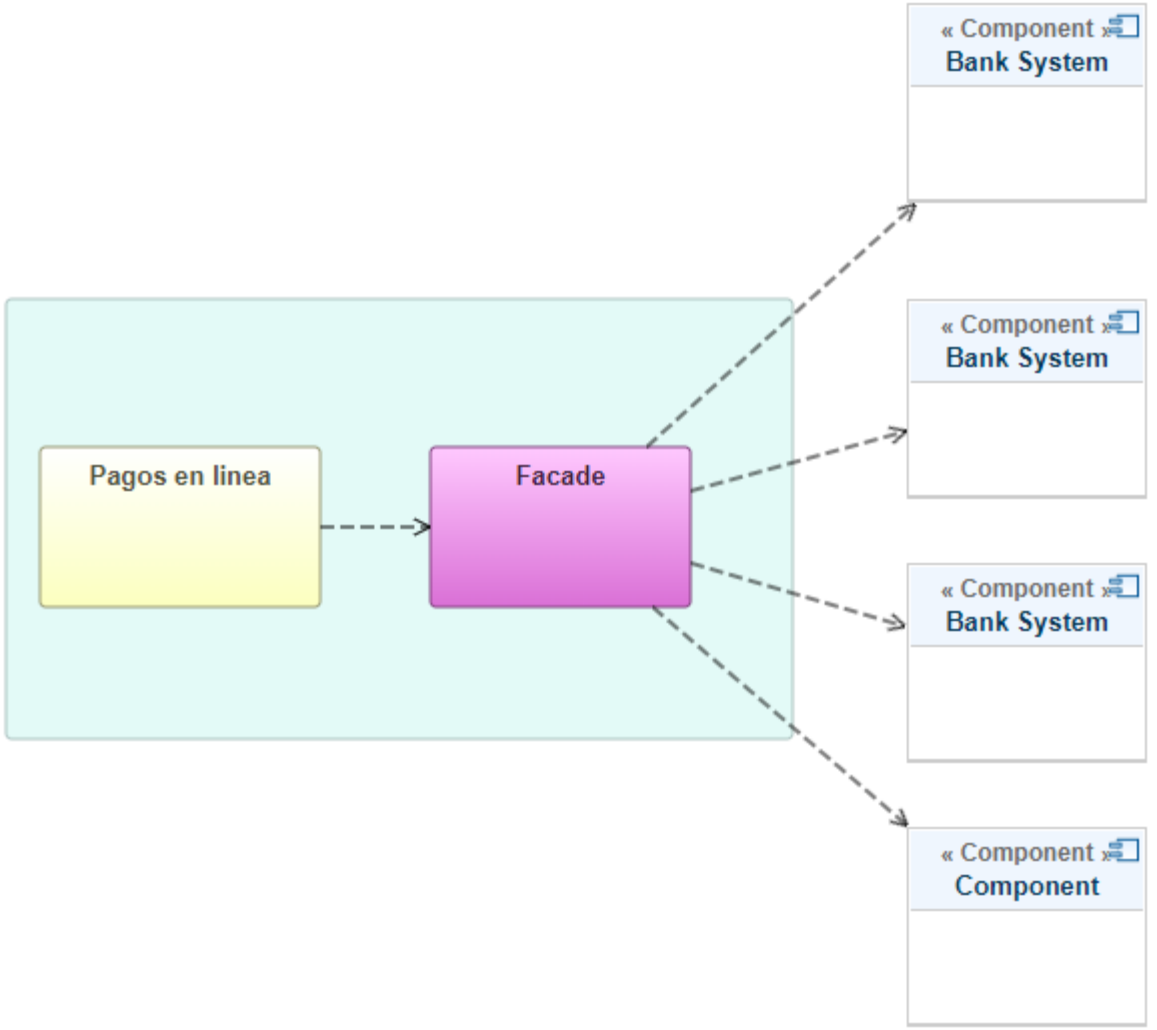
El cliente debe existir y su status no debe ser ‘Baja’ para aplicar el pago.

El status del cliente se cambia a activo, cuando el saldo después del pago en menor o igual a $50.

UNIVERSIDAD Popular del cesar

Patrones Estructurales – Facade





UNIVERSIDAD Popular del cesar

Patrones Estructurales – Flyweight

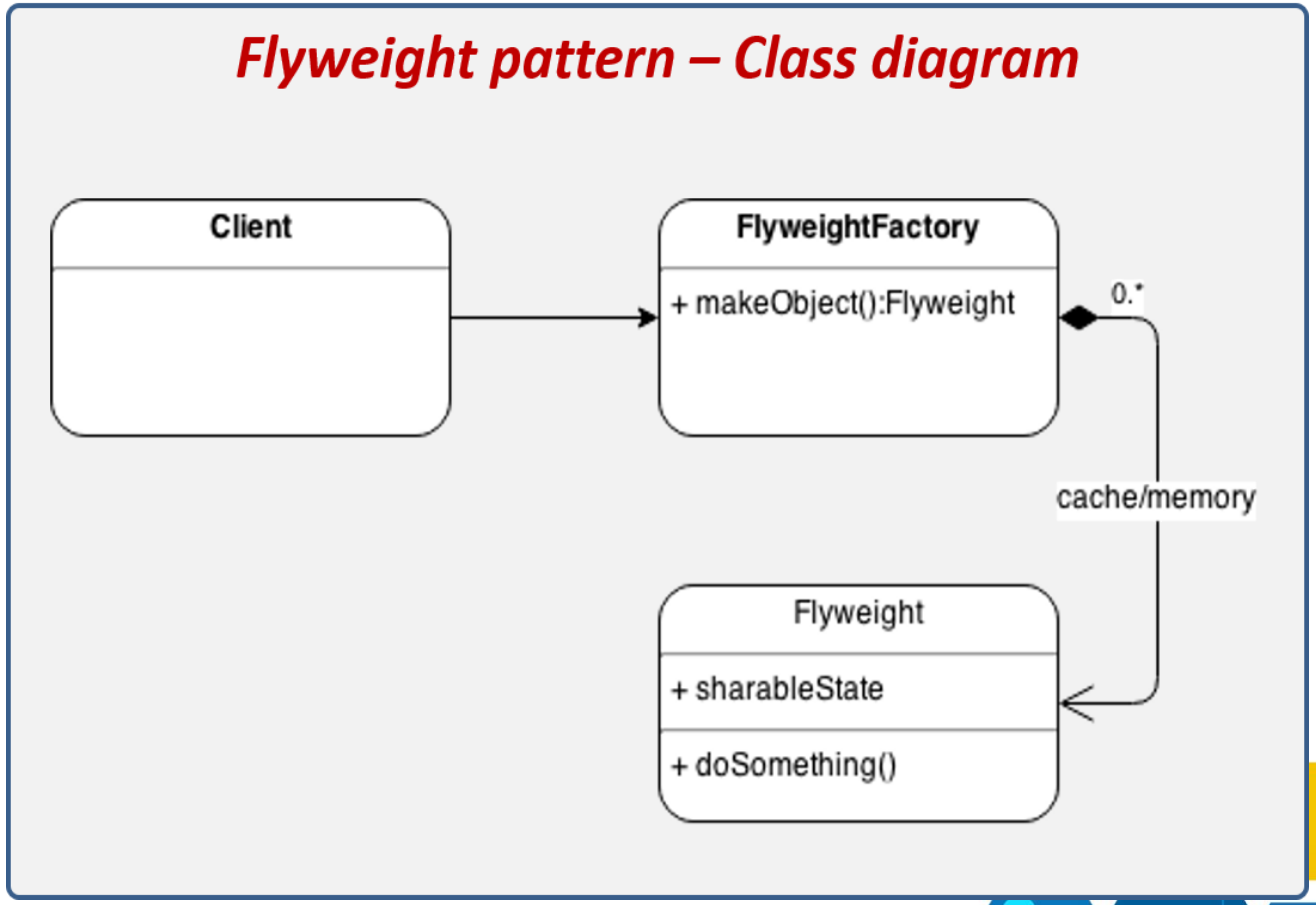
Flyweight te permite crear objetos ligeros, compartiendo las partes comunes del estado entre varios objetos, en lugar de mantener toda la información en cada objeto.

Centra su atención en la creación de objetos ligeros, compartiendo las partes reutilizables con otros objetos.

Permite un manejo eficiente de memoria, reduce la cantidad de memoria requerida por la aplicación.

Es útil cuando la optimización de recursos es primordial, este patrón elimina la redundancia de objetos con estados idénticos.

Divide los objetos en dos partes: estado **intrínseco** y estado **extrínseco**. Utiliza cache para almacenar todas las instancias creadas



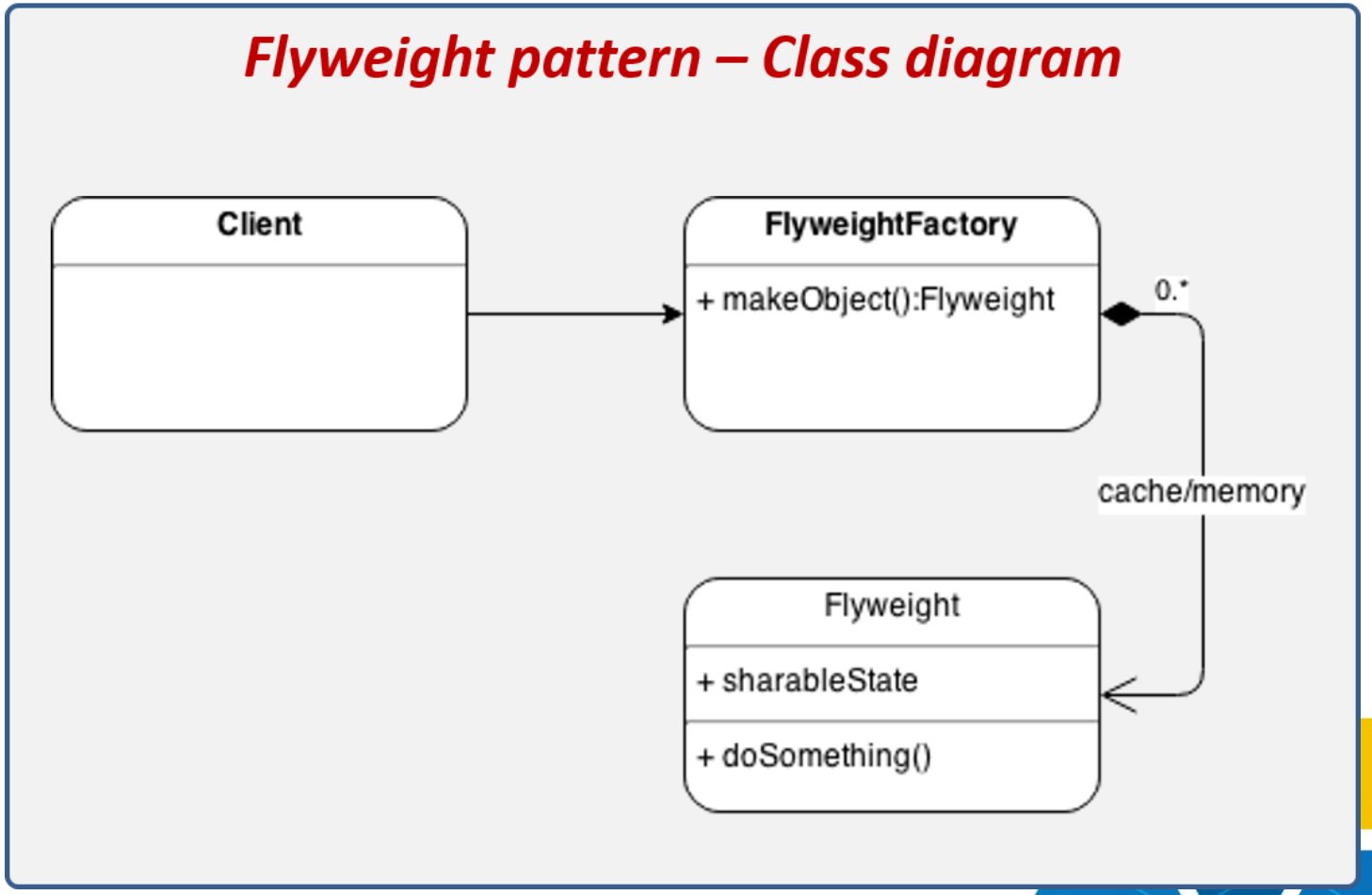
UNIVERSIDAD Popular del cesar

Patrones Estructurales – Flyweight

Client: Objeto que dispara la ejecución.

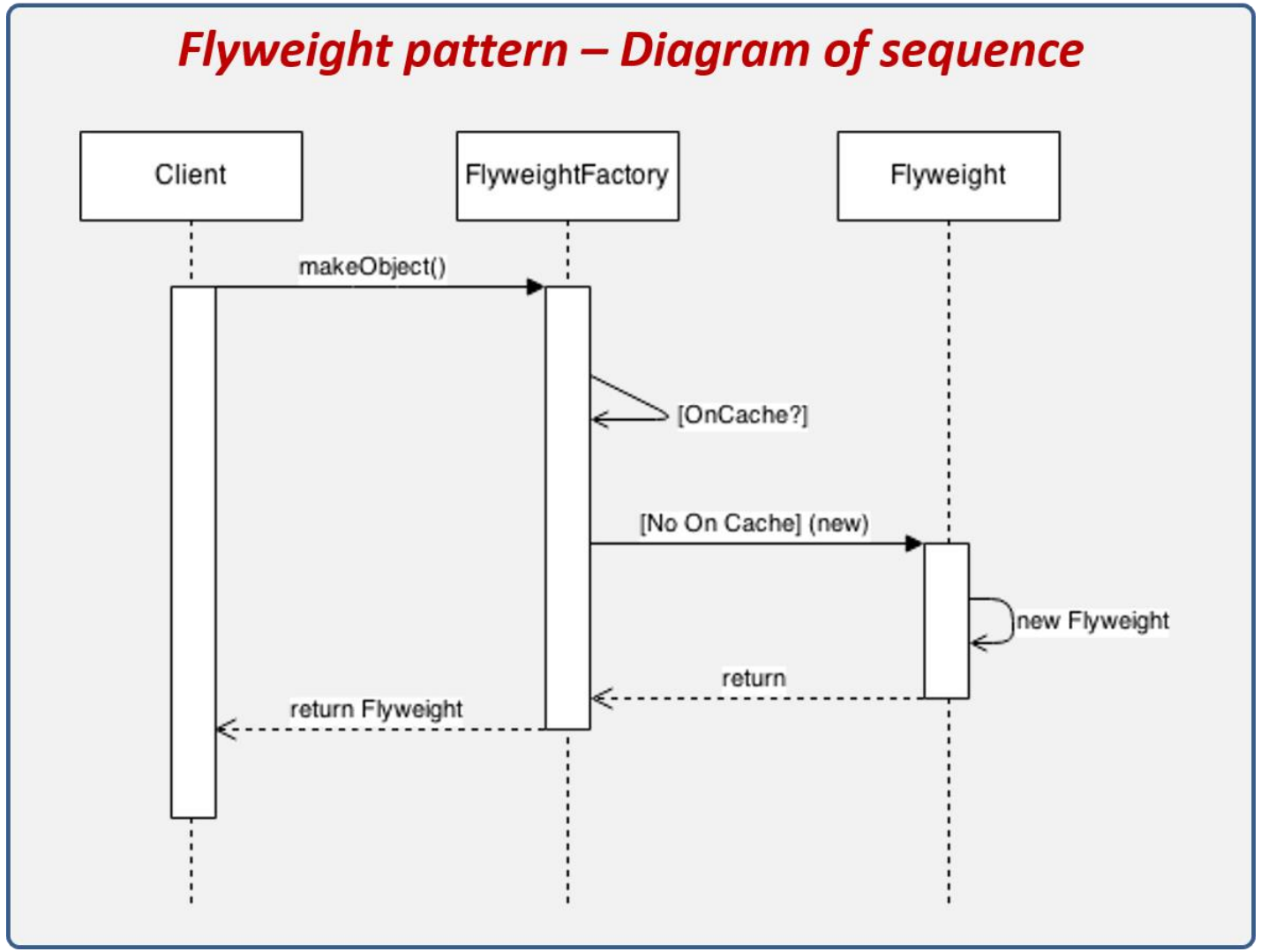
FlyweightFactory: Fábrica que utilizaremos para crear los objetos Flyweight u objetos ligeros.

Flyweight: Corresponde a los objetos que deseamos reutilizar con el fin de que nuestros objetos sean más ligeros.



UNIVERSIDAD Popular del cesar

Patrones Estructurales – Flyweight

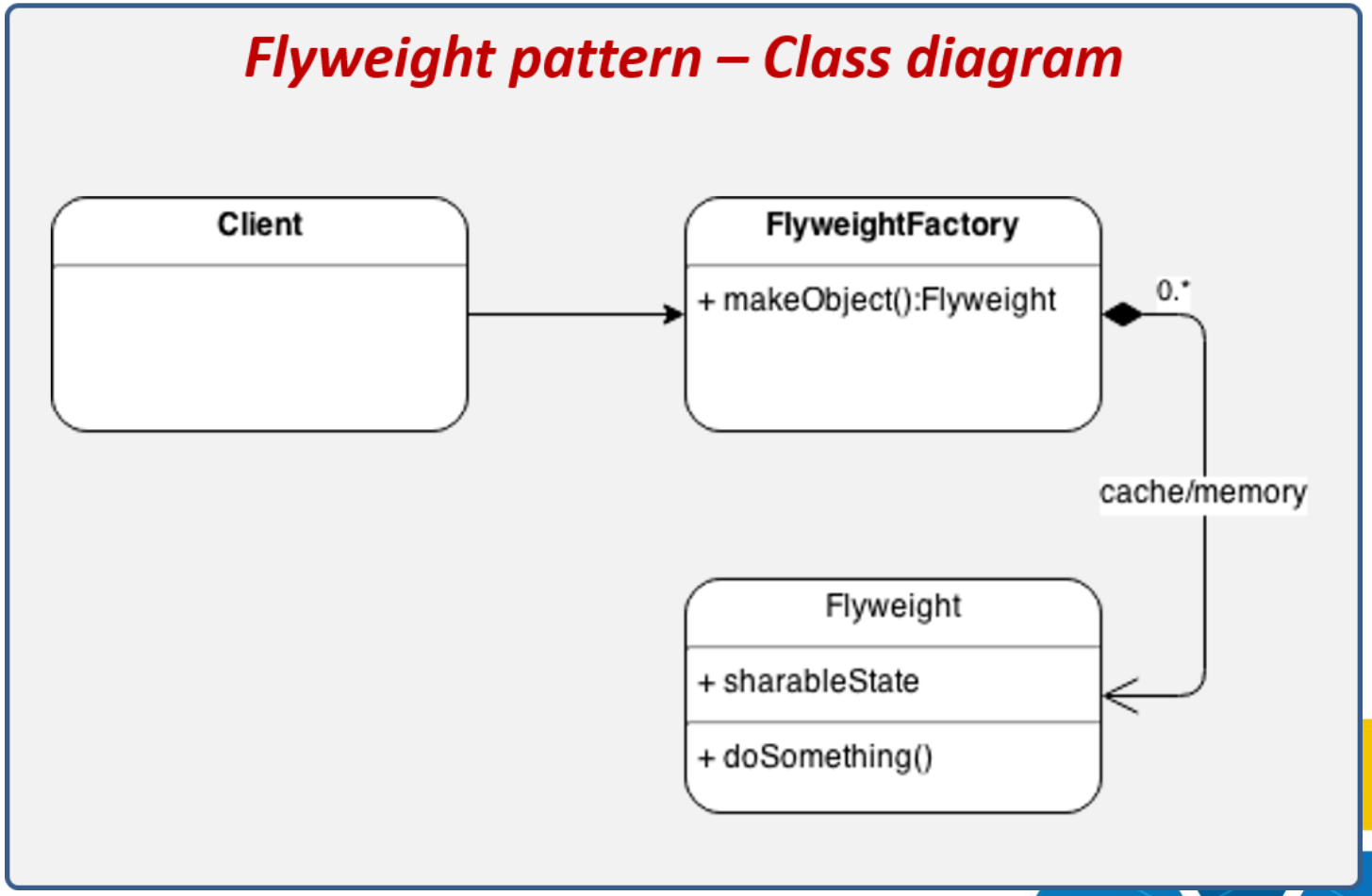


UNIVERSIDAD Popular del cesar

Patrones Estructurales – Flyweight

Cuando implementarlo:

- Cuando observamos que varios de los objetos que creamos contienen la misma información y vemos que podemos tener objetos compartidos.
- Cuando es necesario optimizar el uso de memoria que utilizan nuestros objetos.
- Utiliza el patrón Flyweight únicamente cuando tu programa deba soportar una enorme cantidad de objetos que apenas quepan en la RAM disponible.



UNIVERSIDAD Popular del cesar

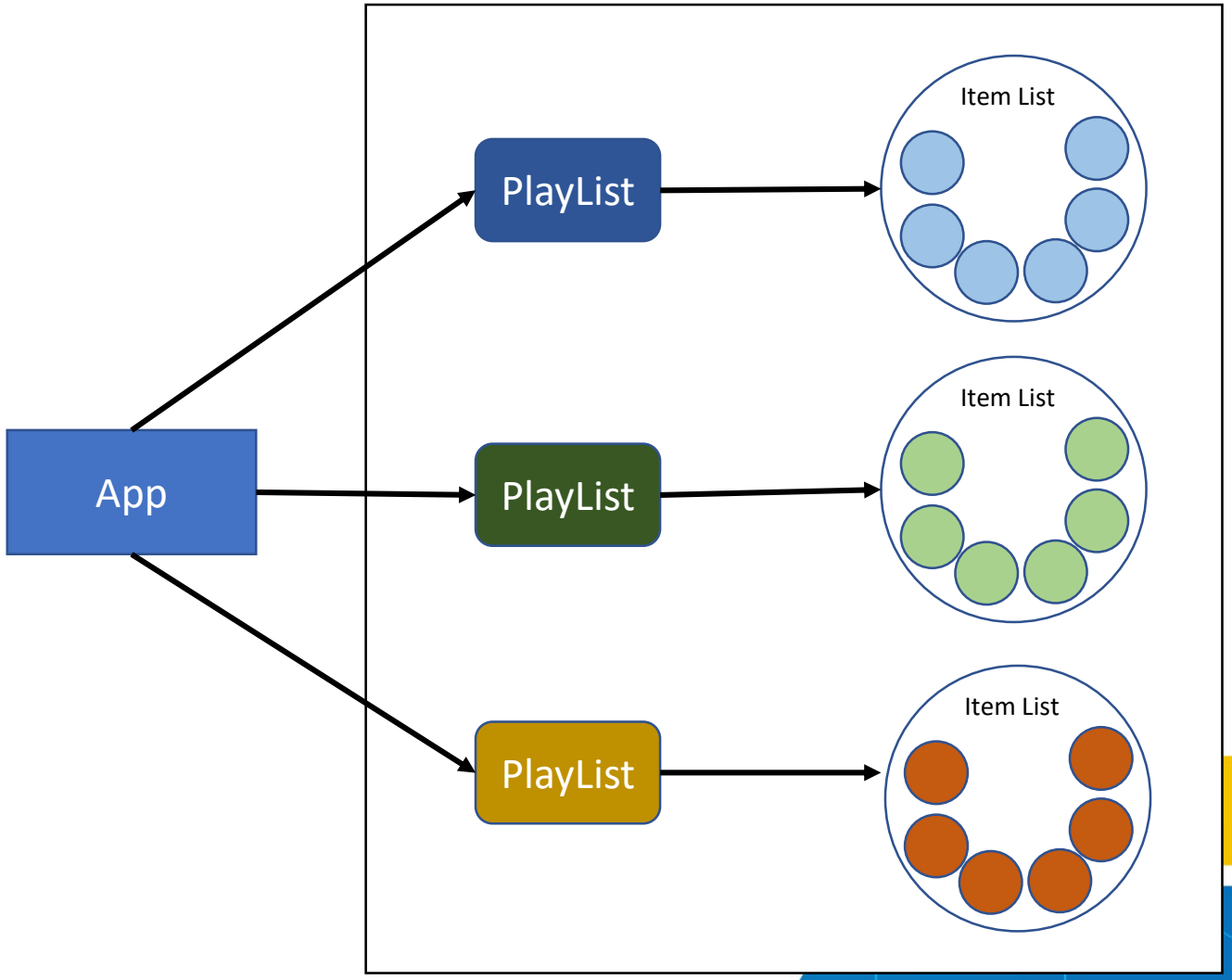
Patrones Estructurales – Flyweight

Ejemplo: app para reproducción de música

Se requiere una aplicación de música, donde los usuarios puedan crear sus propias listas de reproducción.

Un usuario puede crear cuantas listas desee y en ellas puede agregar cualquier numero de canciones. Inclusive una misma canción puede estar en varias listas.

El problemas con los hits del momento.



UNIVERSIDAD Popular del cesar

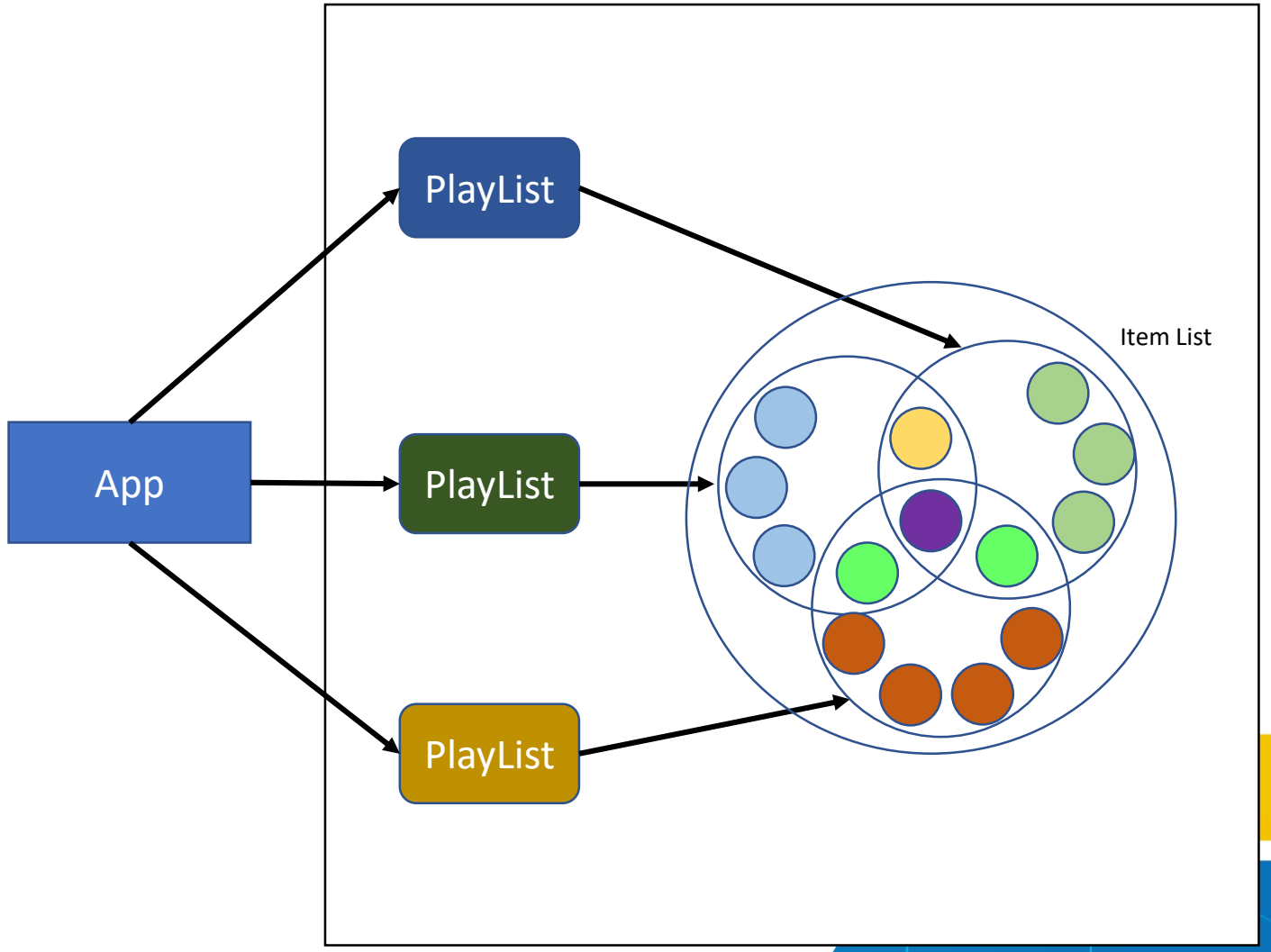
Patrones Estructurales – Flyweight

Ejemplo: app para reproducción de música

Se requiere una aplicación de música, donde los usuarios puedan crear sus propias listas de reproducción.

Un usuario puede crear cuantas listas desee y en ellas puede agregar cualquier numero de canciones. Inclusive una misma canción puede estar en varias listas.

El problemas con los hits del momento.



UNIVERSIDAD Popular del cesar

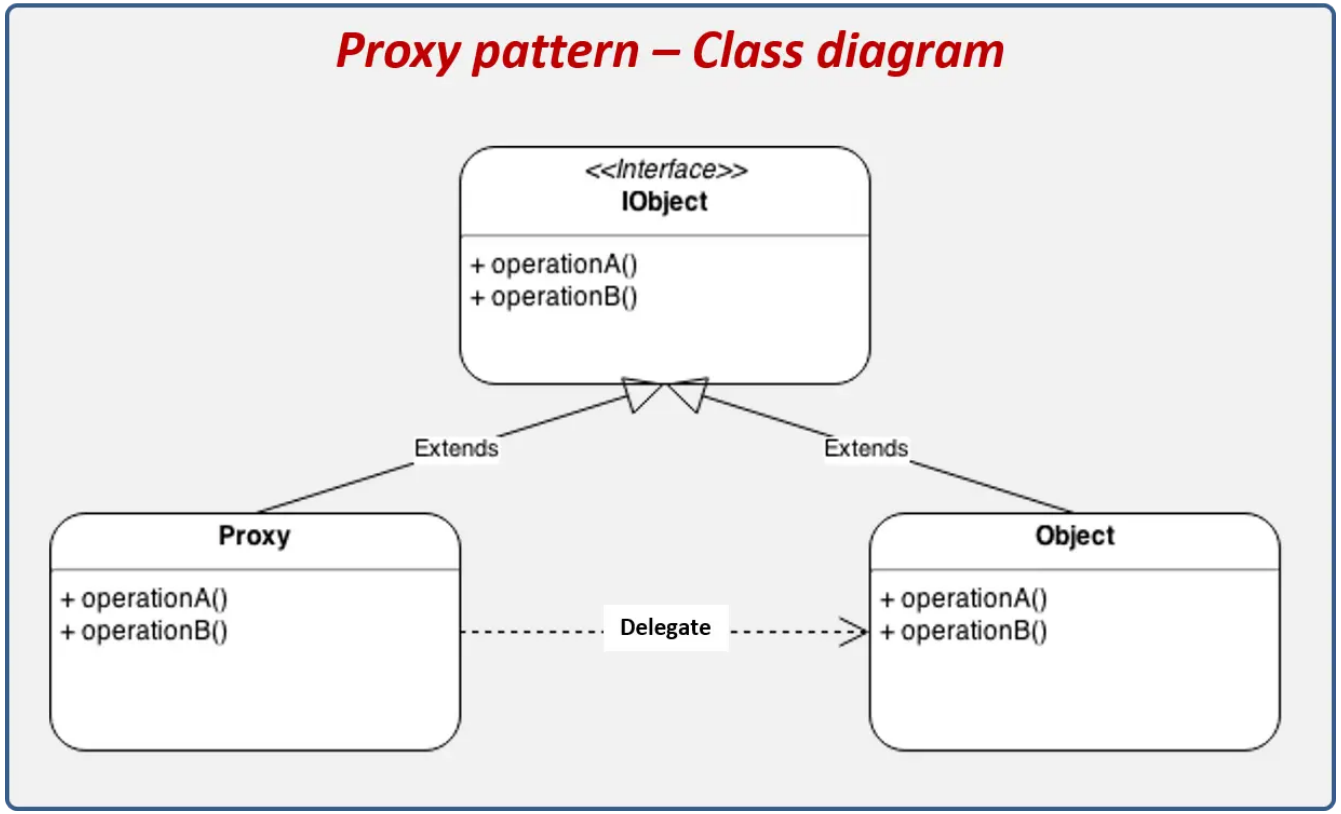
Patrones Estructurales – Proxy

Proxy te permite proporcionar un sustituto o marcador de posición para otro objeto. Un proxy controla el acceso al objeto original, permitiéndote hacer algo antes o después de que la solicitud llegue al objeto original.

Permite ejecutar ciertas acciones antes y después de realizar la acción deseada por el usuario (**mediación**).

El cliente ignora completamente que una mediación se esta llevando a cabo, debido a que el cliente recibe un objeto idéntico en estructura al esperado y no es consciente de la interacción con el proxy.

El proxy actúa como una mascara para encapsular la lógica requerida antes o después de ejecutar un objeto.



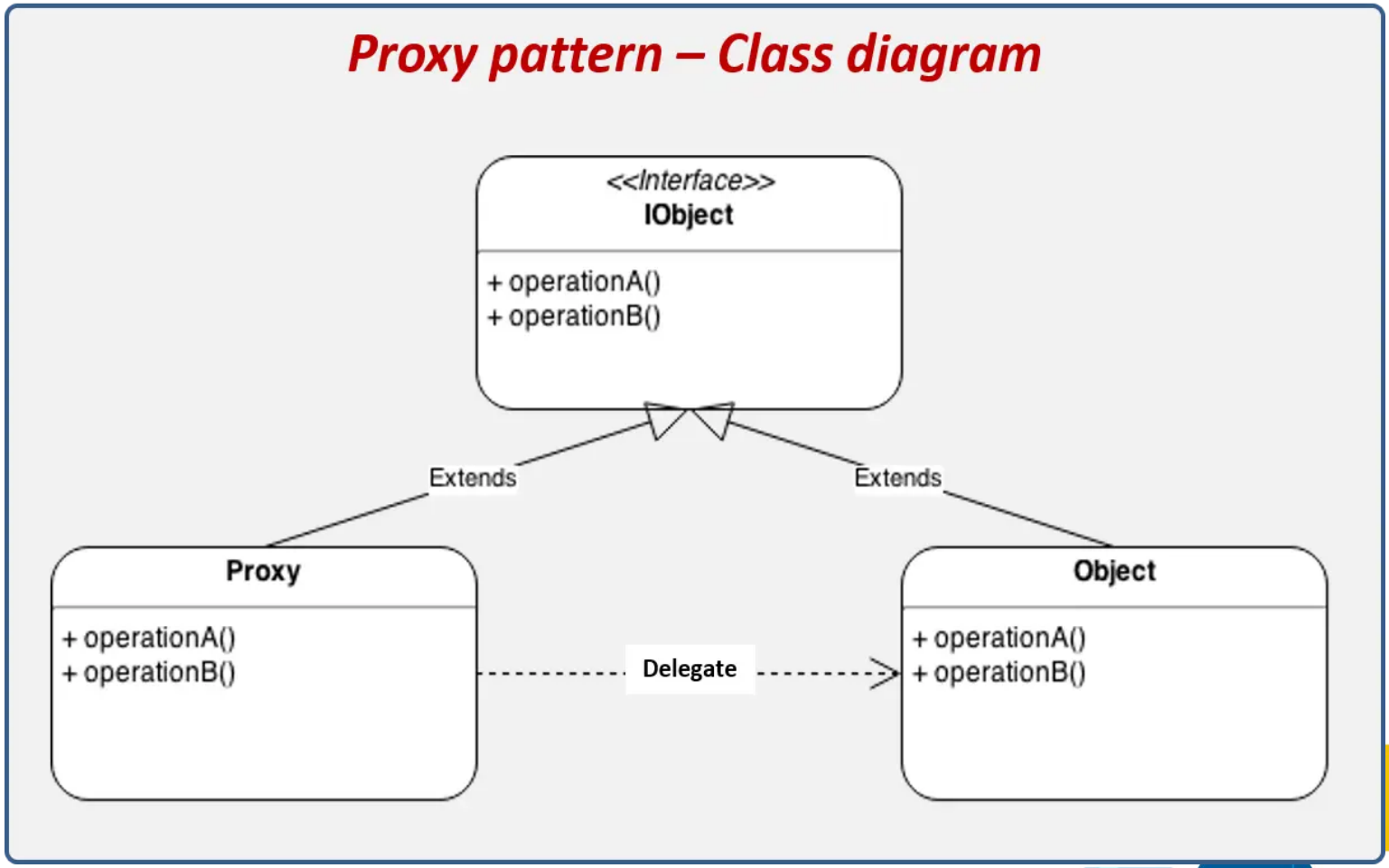
UNIVERSIDAD Popular del cesar

Patrones Estructurales – Proxy

IObject: Representa la interface común entre el Objet y el Proxy.

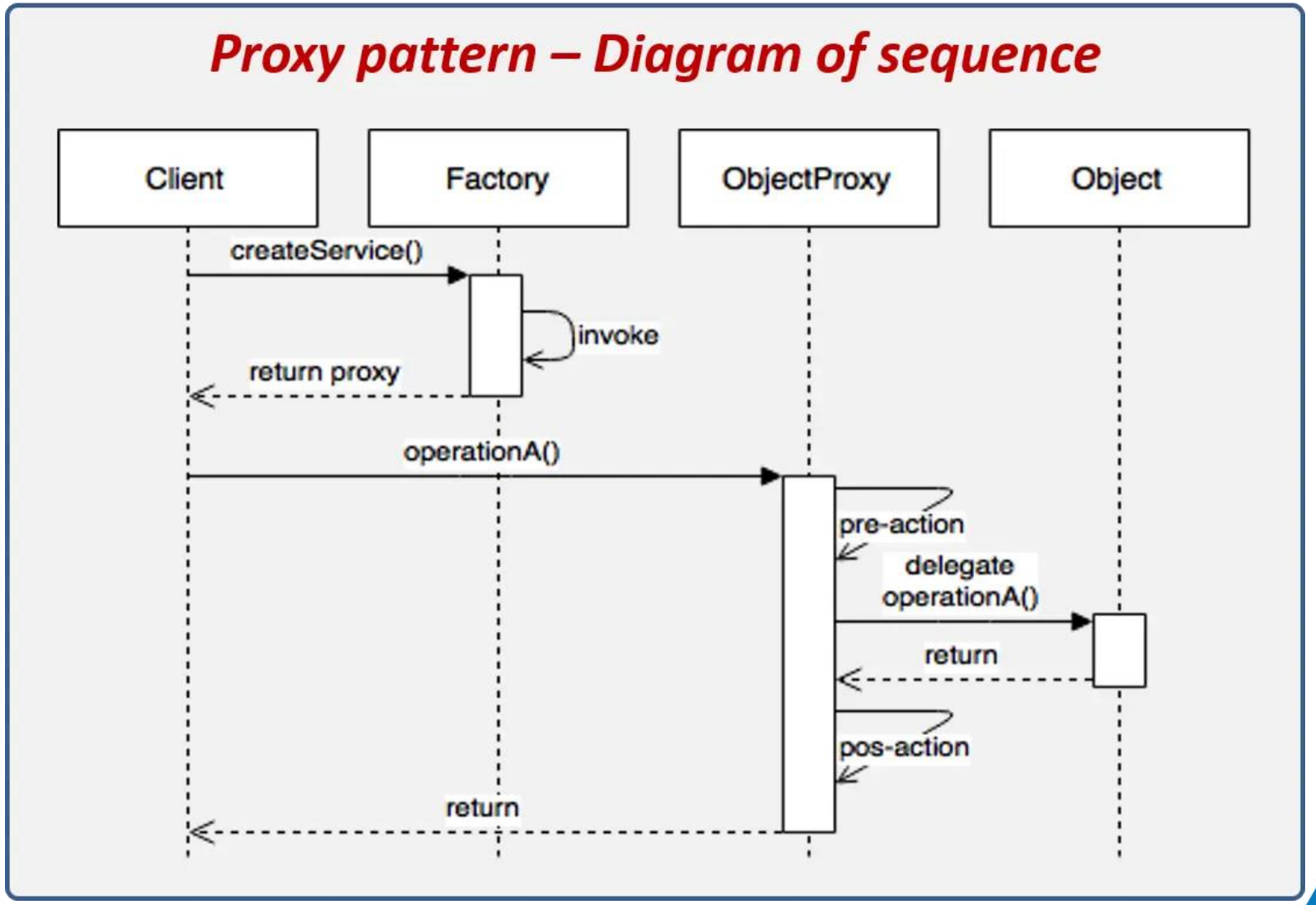
Object: Representa el objeto al que el cliente quiere ejecutar.

Proxy: Clase que implementa IObject y delega la responsabilidad al Object, sin embargo, puede realizar una acción antes y después de llamar al Object.



UNIVERSIDAD Popular del cesar

Patrones Estructurales – Proxy

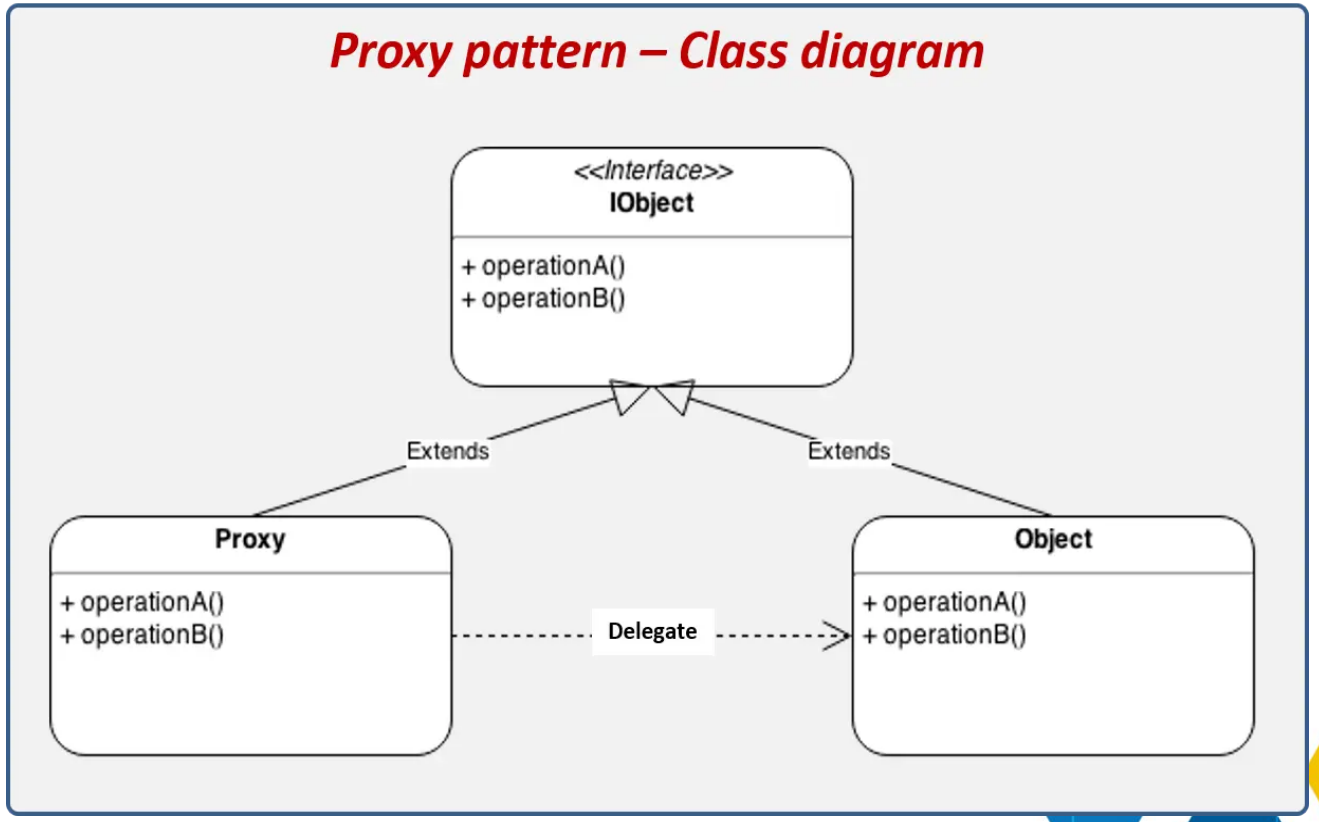


UNIVERSIDAD Popular del cesar

Patrones Estructurales – Proxy

Cuando implementarlo:

- Cuando se requiere controlar la forma en que se ejecuta un objeto sin afectar al consumidor, agregando acciones previas y posteriores a la ejecución de la operación solicitada.
- Es cuando quieres que únicamente clientes específicos sean capaces de utilizar el objeto de servicio
- Ejecución local de un servicio remoto (proxy remoto). Es cuando el objeto de servicio se ubica en un servidor remoto
- Resultados de solicitudes en caché (proxy de caché).



UNIVERSIDAD Popular del cesar

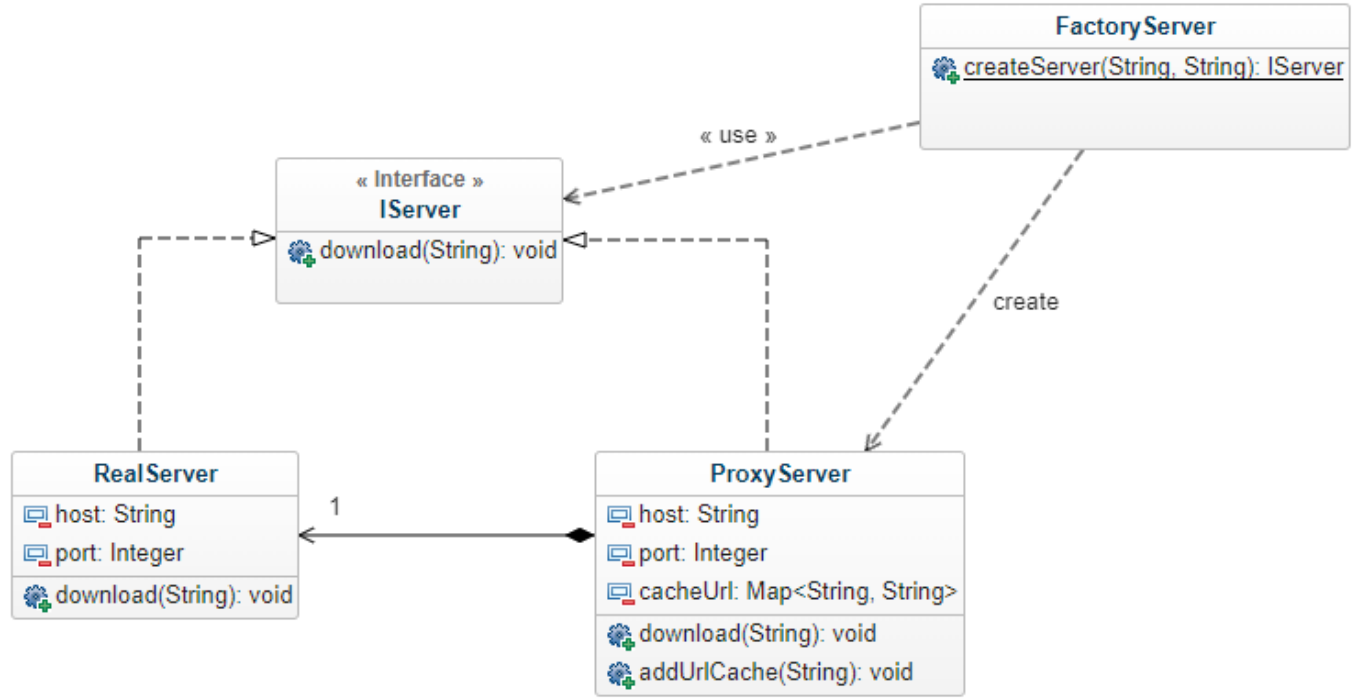
Patrones Estructurales – Proxy

Ejemplo: sistema de descargas

Supóngase que se requiere implementar un sistema de descarga de archivos digitales desde diferentes servidores.

Los materiales que se pretenden descargar son de acceso publico, sin embargo, algunos de ellos se encuentran restringidos en el país del cliente, por lo cual, no debe permitir su descarga.

Se debe tener en cuenta que es frecuente la descarga del mismo material, por ello, la aplicación debe hacer un manejo eficiente de dichas descargas.



UNIVERSIDAD Popular del cesar

Patrón de diseño – Recursos

Libros:

- ***"Patrones de diseño. Elementos de software orientado a objetos reutilizables"***
  Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides

- ***"Sumérgete en los patrones de diseño"***
  Alexander Shvets

- ***"Introducción a los patrones de diseño. Un enfoque practico"***
  Oscar Blancharte

- ***"Patrones de diseño en java. Los 23 modelos de diseño: descripción y soluciones ilustradas "***
  Lauren Debrauwer

UNIVERSIDAD
Popular del cesar