



Arquitectura y Patrones para Aplicaciones Web

Patrones de Diseño

[https://github.com/miw-upm/apaw \(Teoría\)](https://github.com/miw-upm/apaw)

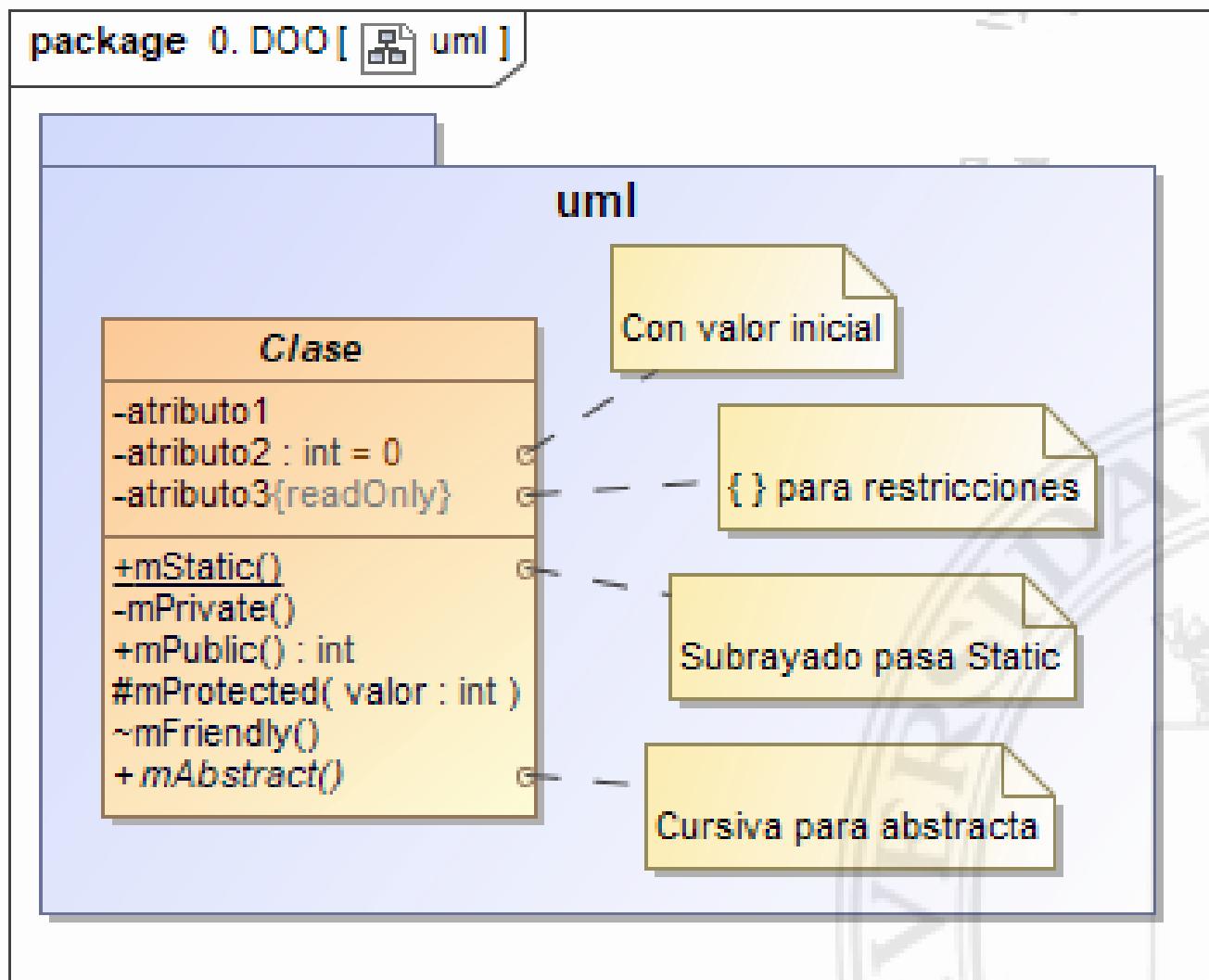
<https://github.com/miw-upm/apaw-ep-themes>

<https://github.com/miw-upm/apaw-microservice-themes-theme>

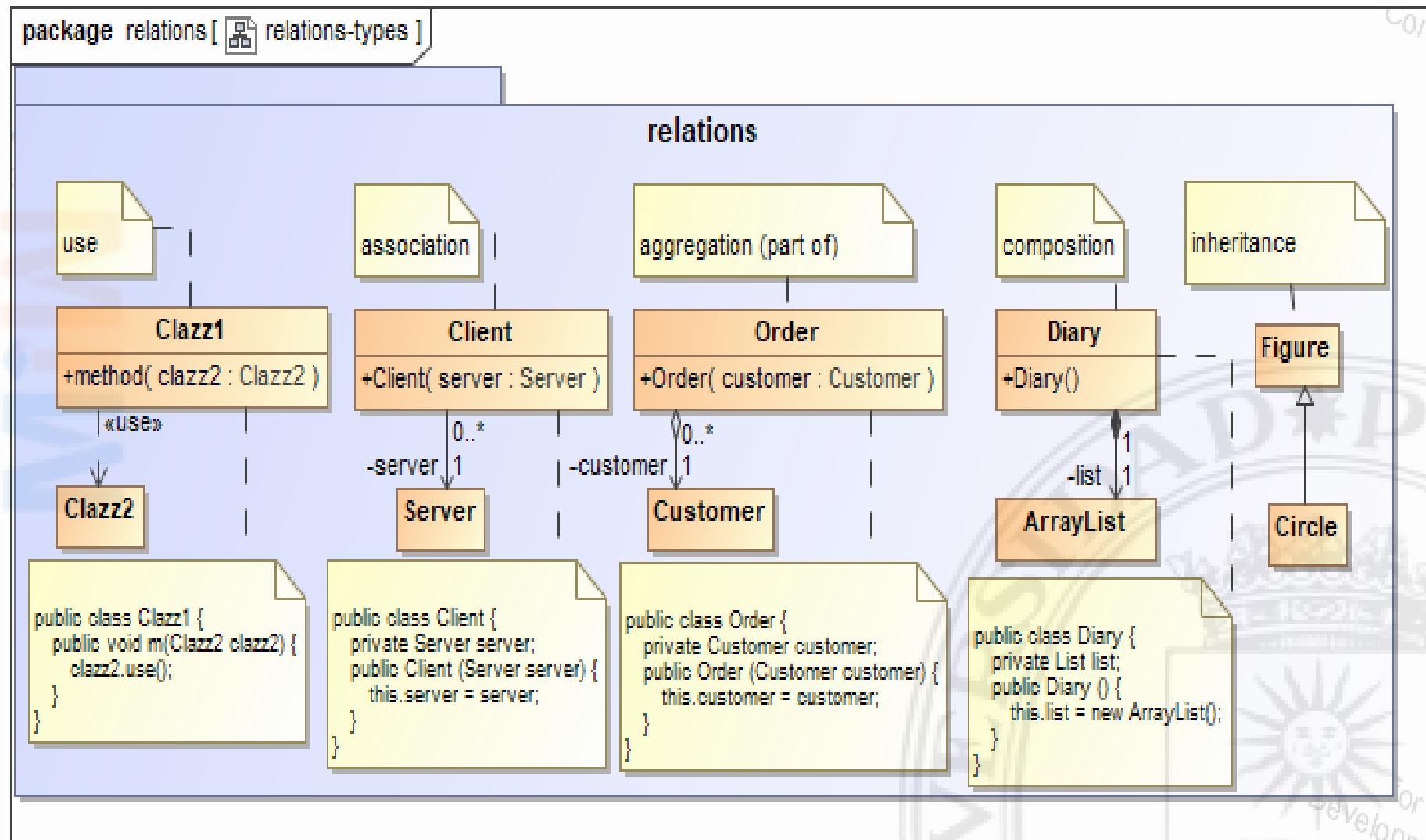
<https://github.com/miw-upm/apaw-microservice-themes-user>

<https://github.com/miw-upm/apaw-microservice-themes-suggestion>

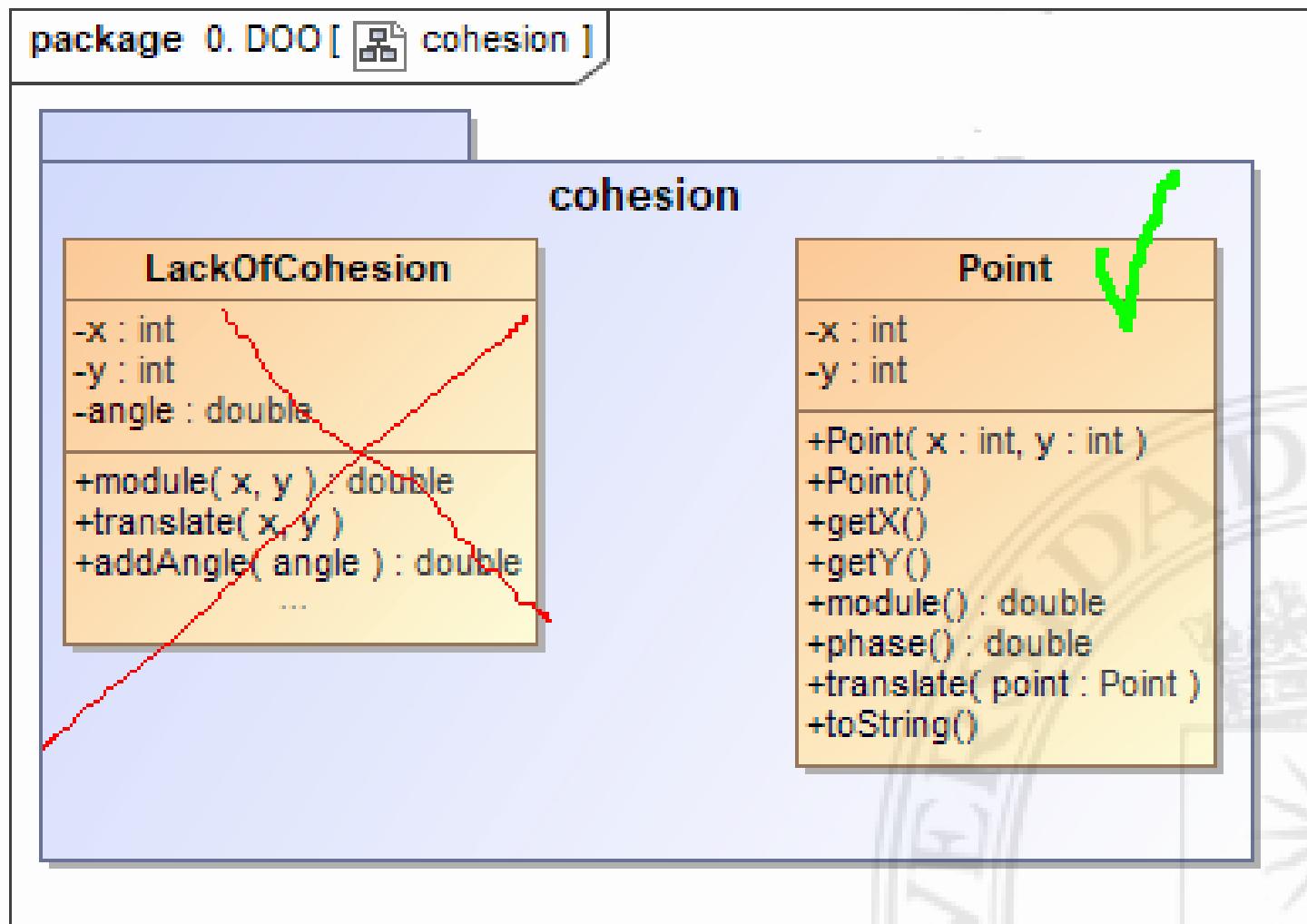
UML: Clase



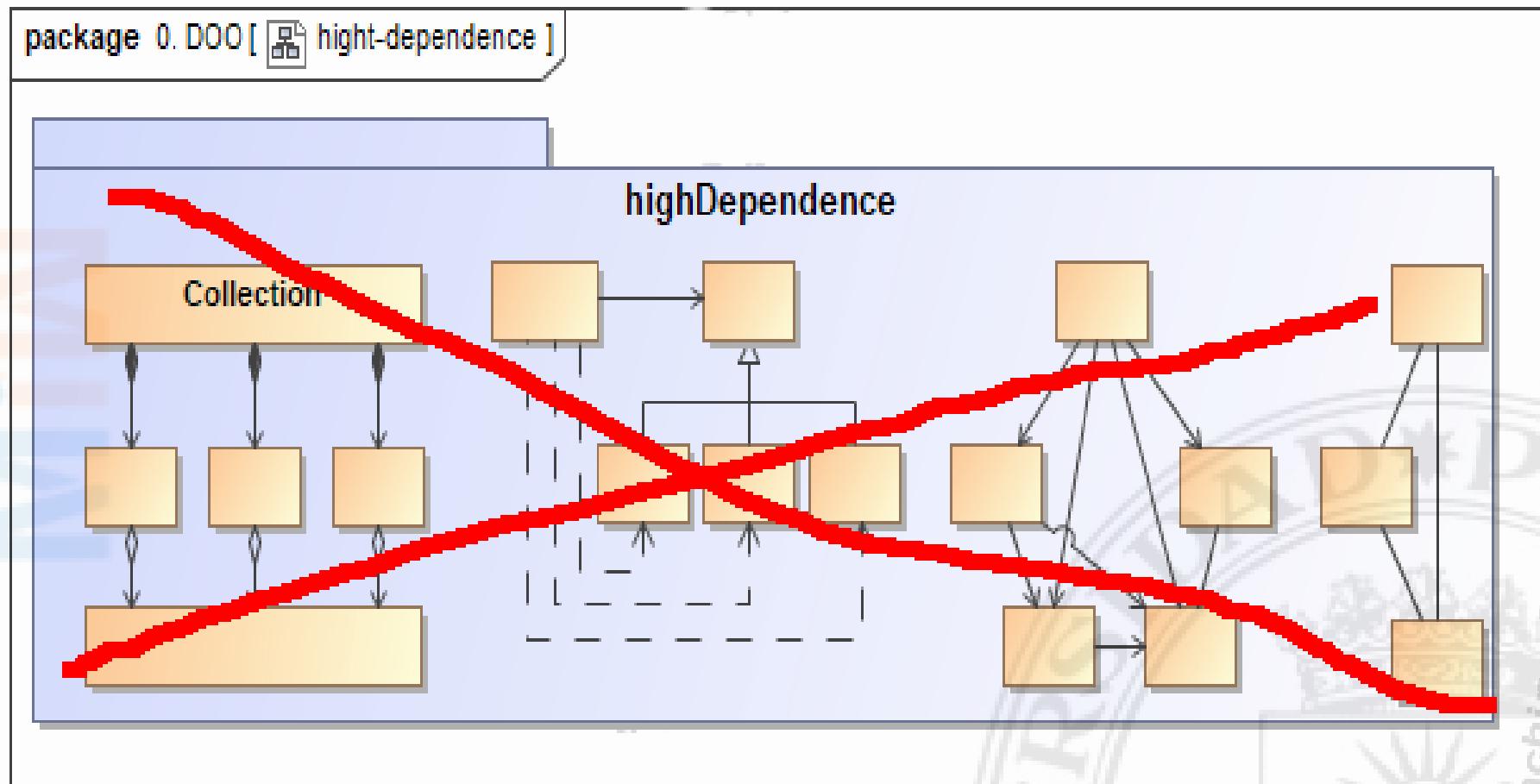
UML: Relaciones



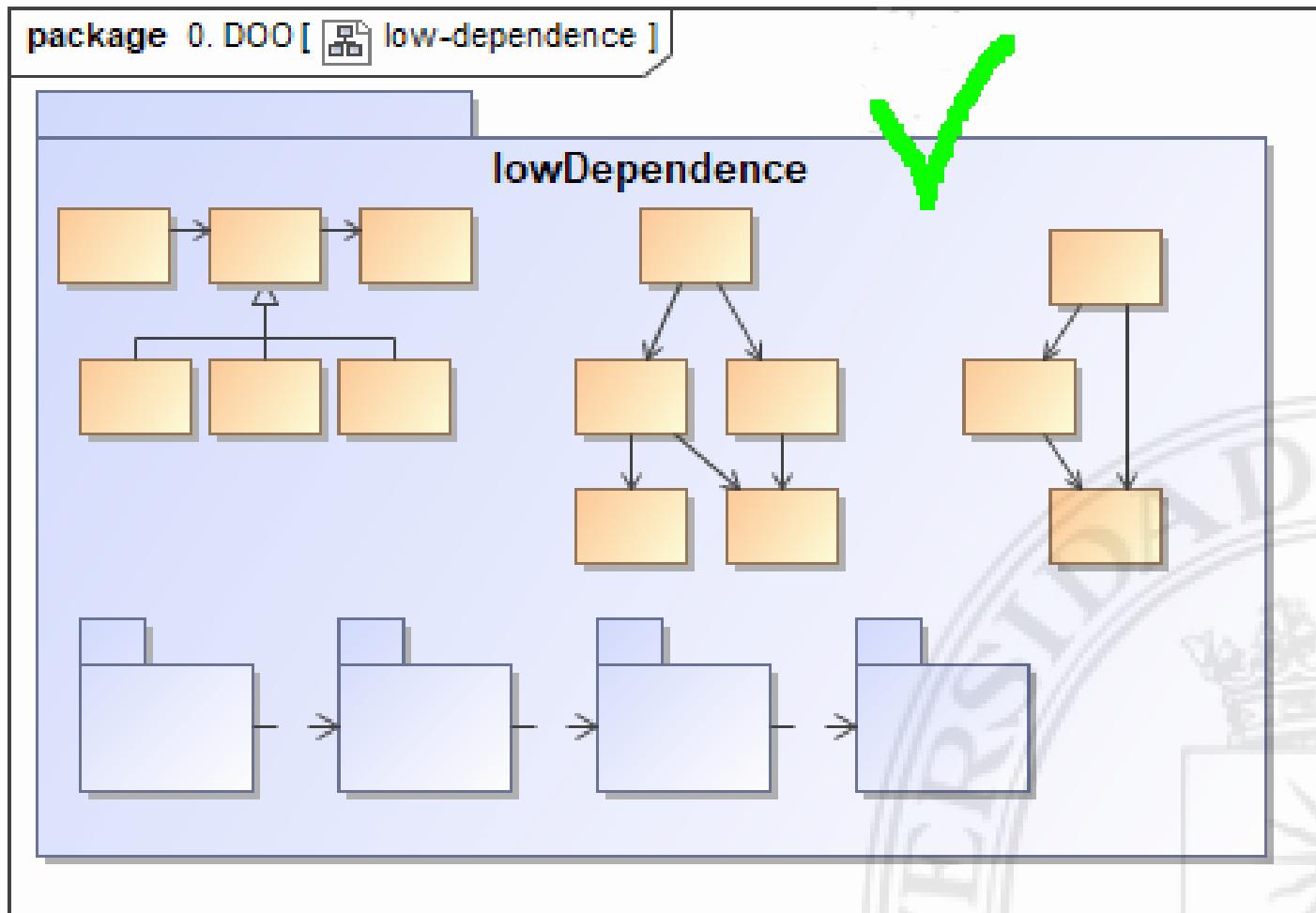
Alta Cohesión – Única responsabilidad



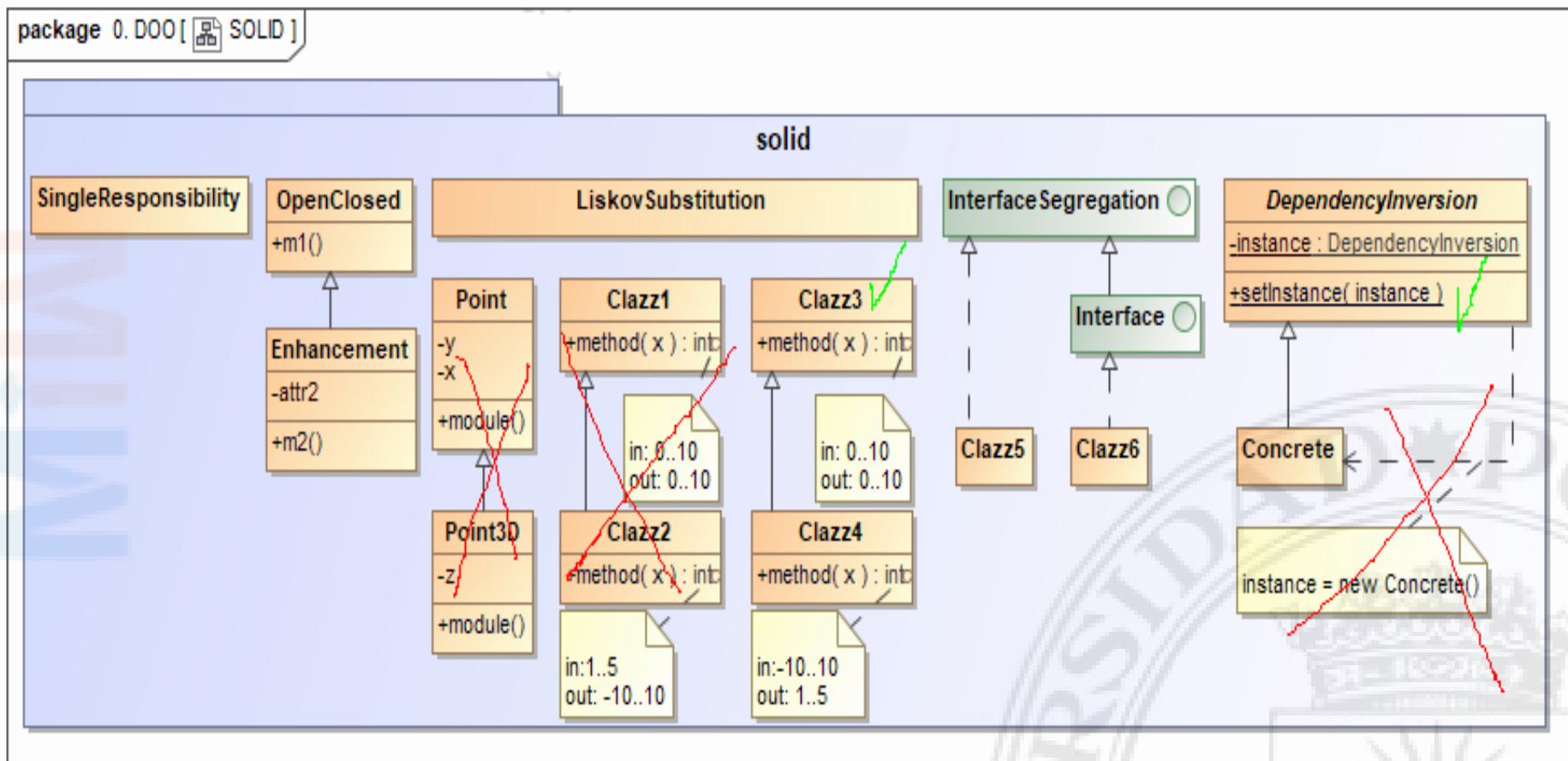
Acoplamiento bajo



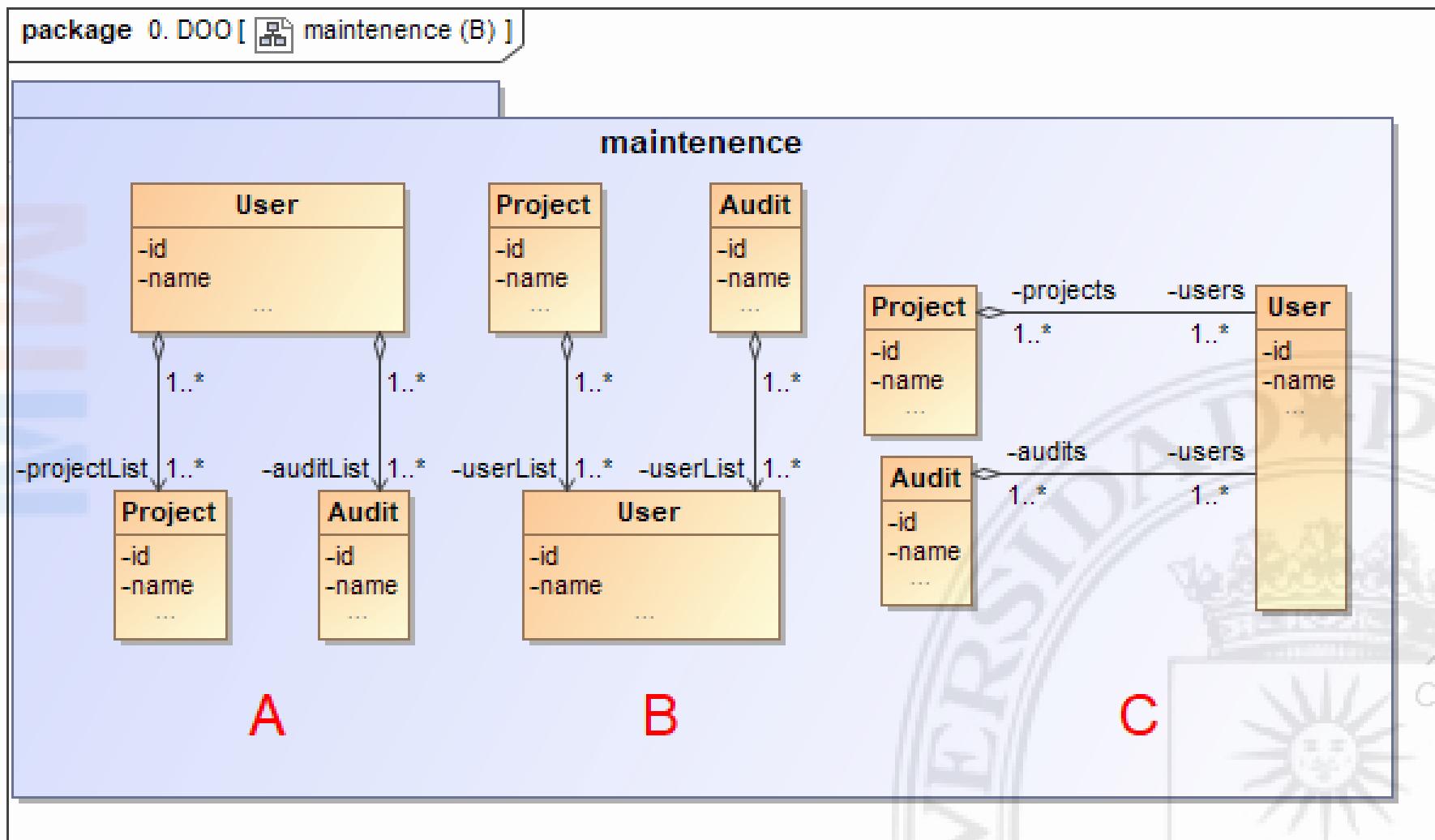
Acoplamiento bajo



Principios SOLID



✍ Test



Introducción

- **Antipatrones (Anti-patterns)**
 - Cut & paste
 - Magic numbers
 - Blob
 - Lava flow
 - Spaghetti code
 - Blind faith
 - Poltergeist
 - Golden Hammer
 - Programming by permutation...
 - Sequential coupling
 - Checking type instead of interface
 - Busy spin
 - ...



Calidad del código

- Formateo
 - Herramienta del IDE
 - Líneas en blanco
 - Sin comentarios
 - Nombres de clases, métodos, atributos, parámetros y variables
- Sencillez del código
 - Estructuras anidadas: <3
 - Complejidad ciclomática: <9-12 ≈ <4 bifurcaciones
- Métricas
 - Paquete: <20 clases
 - Clases: <500-200 líneas, <20 métodos
 - Métodos: <3-4 parámetros, <15 líneas
- Eliminar redundancias (*copy & paste*)
- Eliminar código muerto
- Tratamiento de errores

Introducción

- Qué es un patrón: “es una solución a un problema que ocurre de forma repetida en nuestro entorno”
- Partes esenciales
 - *Nombre del patrón.* Enriquece el vocabulario y permite comunicarse mejor entre diseñadores
 - *Problema.* Explica el problema y su contexto
 - *Solución.* Describe el diseño de la solución, sus relaciones y su responsabilidades
 - *Consecuencias.* Describe las ventajas e inconvenientes de aplicar el patrón
- Referencia. *Patrones de Diseño, GoF (Gang Of Four), E. Gamma, R. Helm, R. Johnson y J. Vlissides*

Catálogo de patrones. Propósito: Creación

- Ámbito: *Objeto*
 - *Singleton*. Se garantiza que una clase sólo tenga una instancia y se proporciona un acceso global a ella
 - *Builder*. Separa la construcción de objeto complejo de su representación, permitiendo diferentes construcciones
 - *Abstract Factory*. Proporciona un interface para crear familias de objetos relacionadas
 - *Prototype*. Especifica y crea objetos por medio de un prototipo mediante la clonación
- Ámbito: *Clase*
 - *Factory Method*. Define una abstracción para crear objetos, y son las subclases que deciden la clase concreta a instanciar

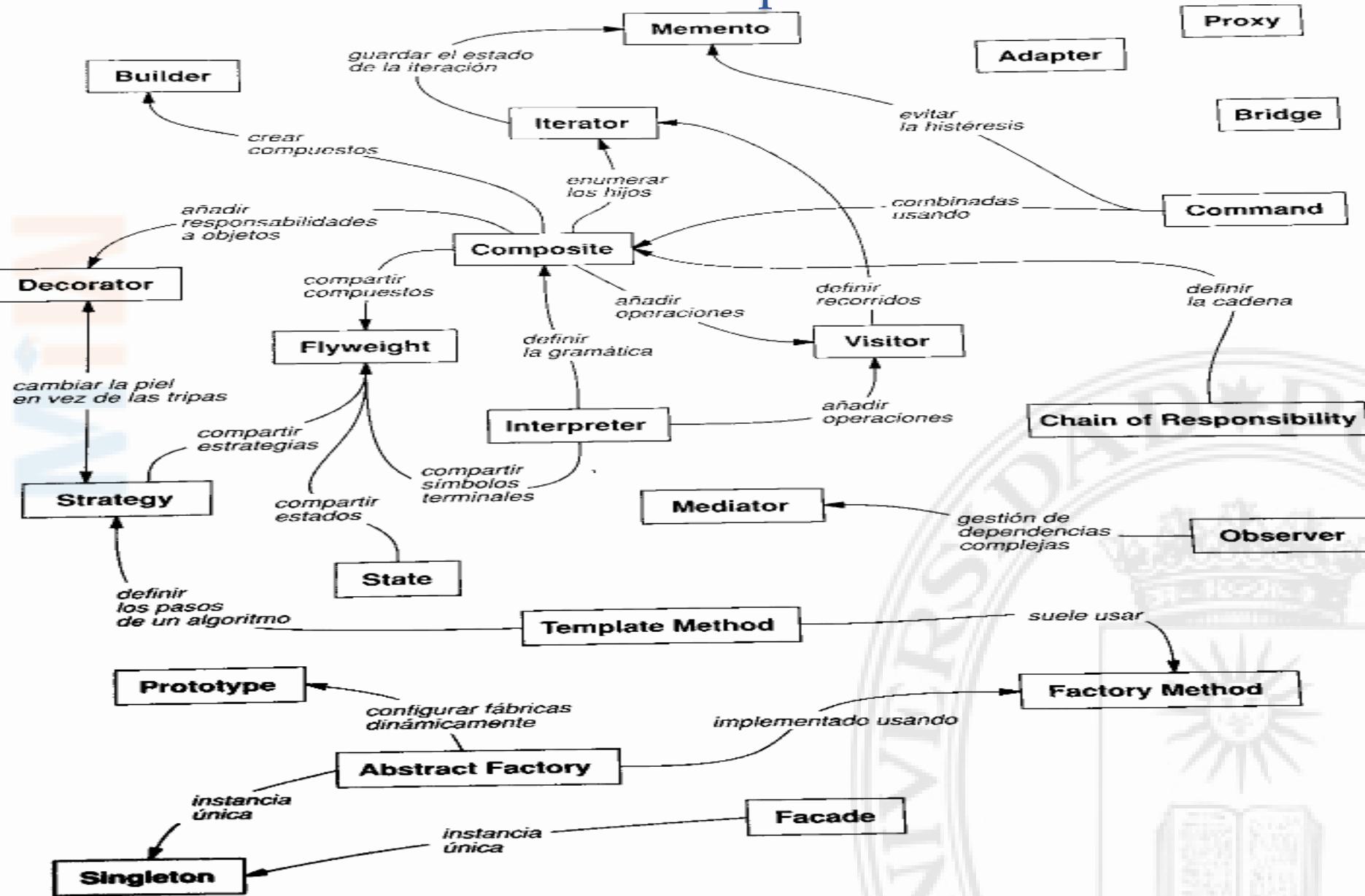
Catálogo de patrones. Propósito: Estructural

- Ámbito: *Objeto*
 - *Composite*. Permite estructuras en árbol tratando por igual a las hojas que a los elementos compuestos
 - *Decorator*. Asigna responsabilidades de forma dinámica a objetos, proporcionando una alternativa flexible a la herencia
 - *Facade*. Proporciona un interface unificado para un conjunto de interfaces de un subsistema
 - *Flyweight*. Compartir objetos de grado fino de forma eficiente
 - *Proxy*. Se proporciona un sustituto o representante para controlar el acceso a un objeto
 - *Bridge*. Desacopla una abstracción de su implementación, permitiendo modificaciones independientes de ambas
- Ámbito: *Clase y Objeto*
 - *Adapter*. Convierte una interface de una clase en otra que es la que esperan los clientes

Catálogo de patrones. Propósito: Comportamiento

- Ámbito: *Objeto*
 - **Observer.** Se define una dependencia entre uno a muchos, del tal manera, que cuando cambie avise a todos los objetos dependientes
 - **Command.** Se desacopla el objeto que invoca a la operación asociada, mediante un objeto. Ello permite realizar ordenes compuestas (patrón *composite*) o llevar una cola y deshacer operaciones
 - **State.** Permite que un objeto cambie su comportamiento cada vez que cambie su estado interno
 - **Visitor.** Definir un conjunto de operaciones sobre una estructura de datos de forma independiente
 - **Memento.** Externaliza el estado interno de un objeto sin violar la encapsulación
 - **Iterator.** Proporciona un modo de acceder secuencialmente a los elementos de un objeto sin exponer su representación interna
 - **Strategy.** Define un conjunto de algoritmo haciéndolos intercambiables dinámicamente
 - **Mediator.** Define un objeto que encapsula como interactúan un conjunto de ellos
- Ámbito: *Clase*
 - **Interpreter.** Define una representación de la gramática de un lenguaje con un intérprete
 - **Template Method.** Define una operación en el esqueleto de un algoritmo, delegando en las subclases los detalles

Relación entre patrones

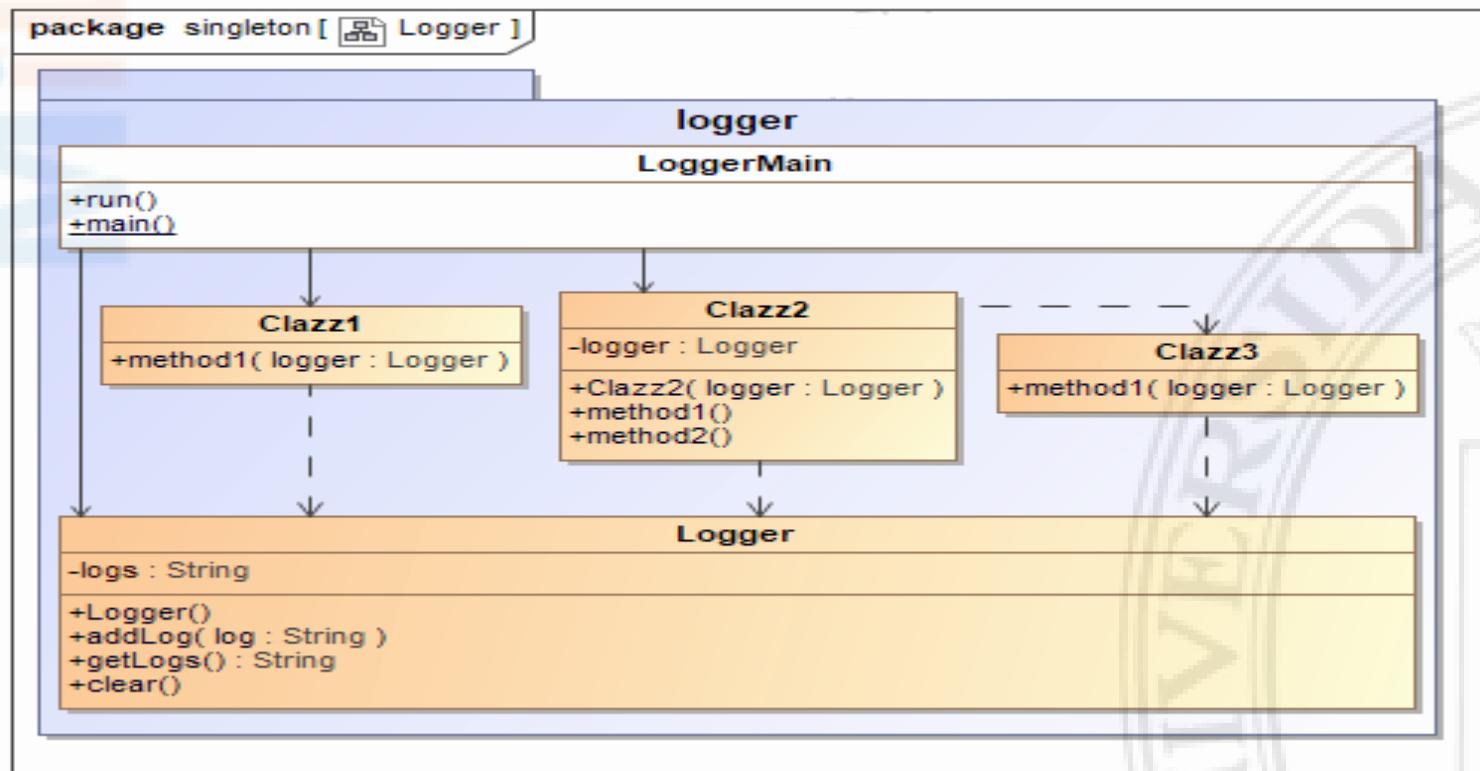


Singleton (Único)

Motivación

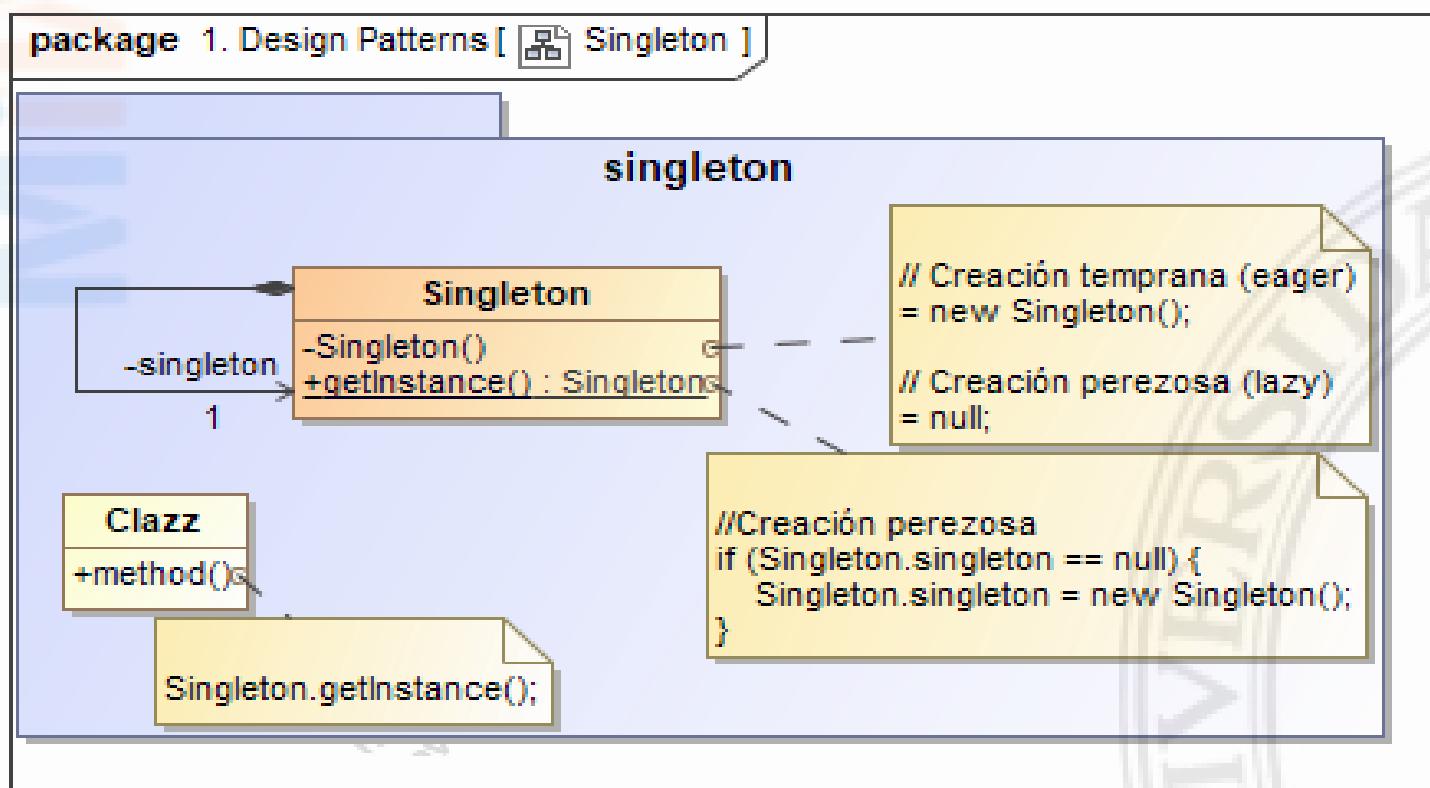
- En una aplicación se necesita que exista un solo objeto de una clase, Gestor de Ficheros, Cola de Impresión, Gestor de Registros, Gestor de BD..., y además, sea accesible desde el resto de clases de la aplicación

Fuentes: paquete singleton



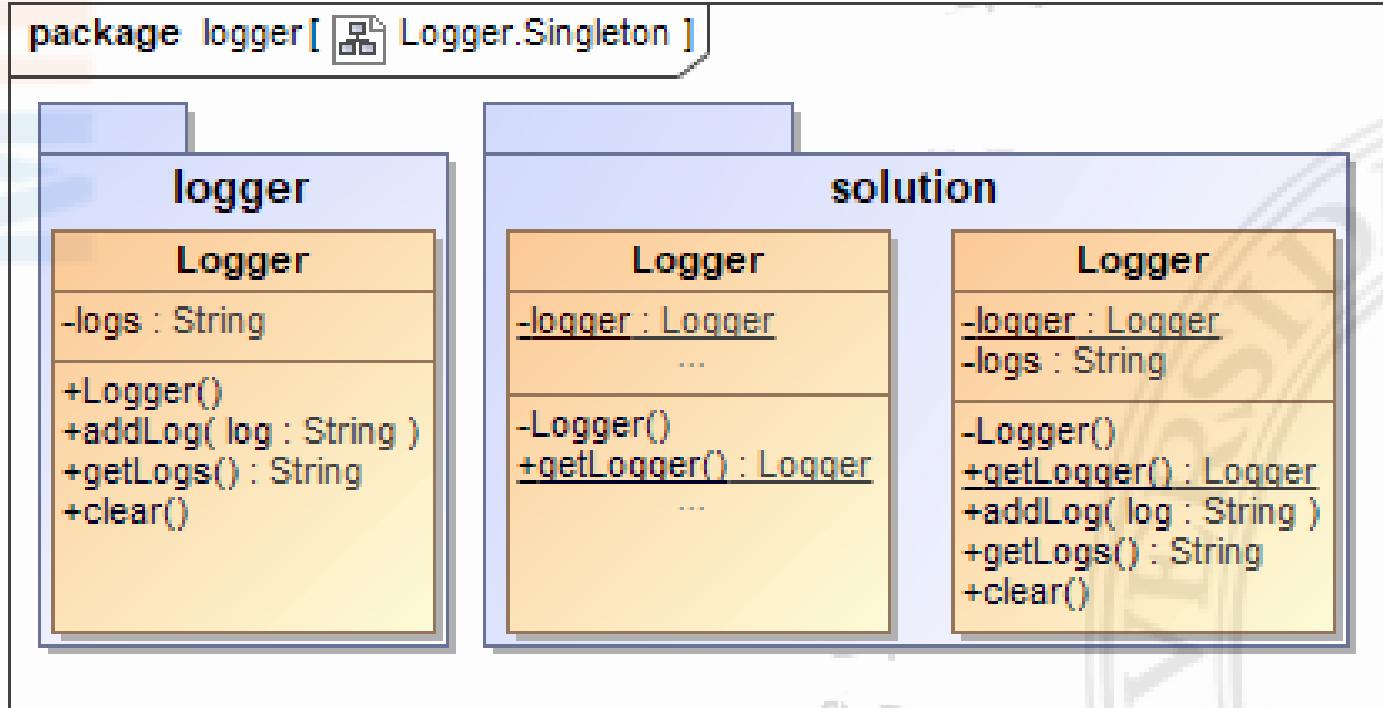
Singleton (Único)

- Propósito
 - Se garantiza que una clase sólo tenga una instancia y se proporciona un acceso global a ella
- Estructura



Logger

- Roles:
 - Singleton: Logger
 - + Atributo privado estático de la Clase
 - + Constructor privado
 - + Método *get* estático y público, para devolver el atributo



✍ Factory

- Aplicar el patrón Singleton a *ReferencesFactory*, con creación temprana
 - Refactorizar *ReferencesFactory* respetando la funcionalidad original

```
package 2.PD [ ReferencesFactory ]
```

ReferencesFactory

```
-references : Map<String,Integer>
-reference : int

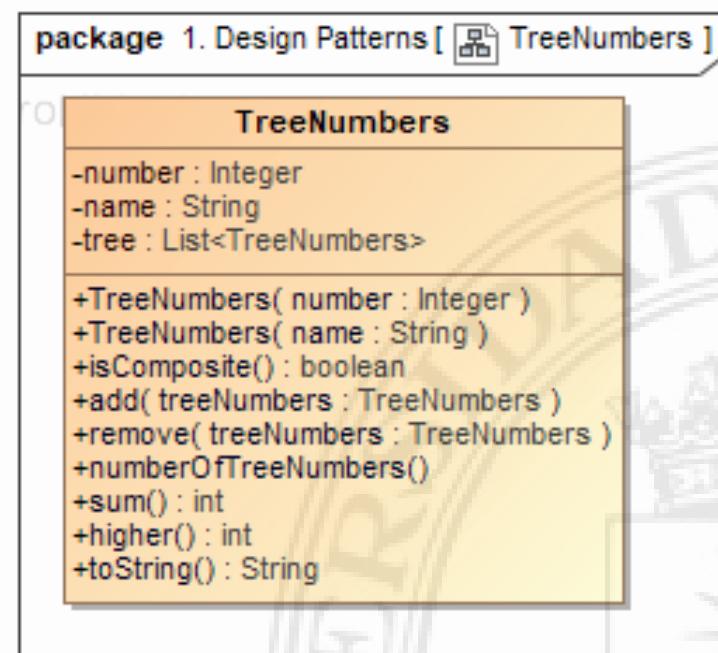
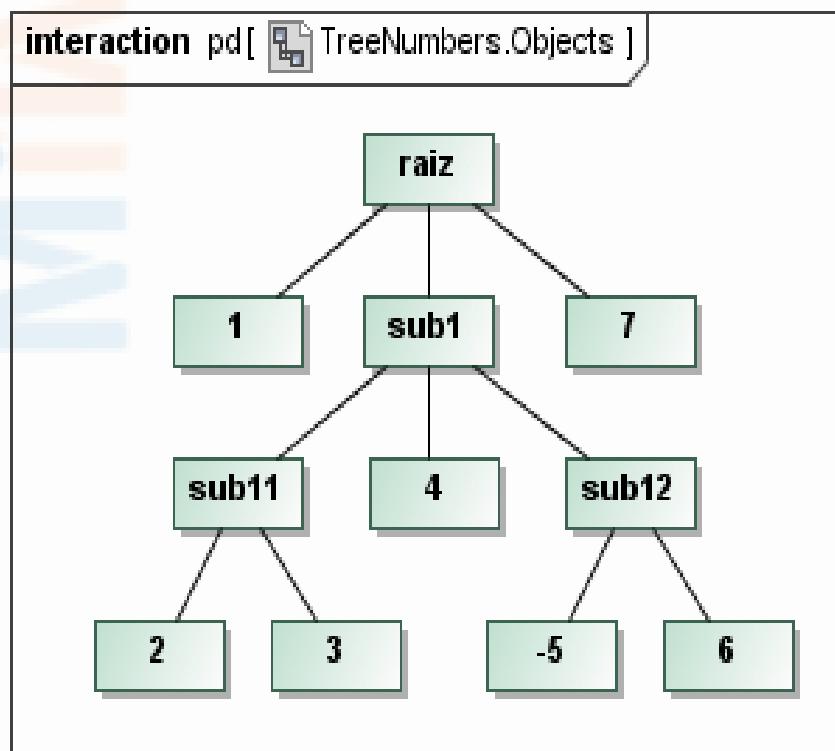
+ReferencesFactory()
+getReference( key : String ) : int
+removeReference( key : String )
```

Composite (Compuesto)

- Motivación
 - En las aplicaciones gráficas, se permiten realizar dibujos por la agrupación de elementos simples y otros elementos agrupados
- Fuentes: paquete composite
- Árbol de números
 - Se quiere construir una estructura de árbol con valores numéricos. Existen nodos con valores numéricos (hojas) y nodos que contienen otros nodos (compuestos)
 - Si se intenta añadir a un nodo hoja, se debe lanzar la excepción: *UnsupportedOperationException*. Si se intenta borrar nodos a una hoja no se hace nada y no genera excepción
 - Si se intenta añadir o borrar con parámetro *null*, no hace nada y no genera excepción
 - Se deben crear las operaciones *numberOfNodes*, *sum* y *higher*, se opera sobre si mismo y todos los nodos que dependen de él

Composite (Compuesto)

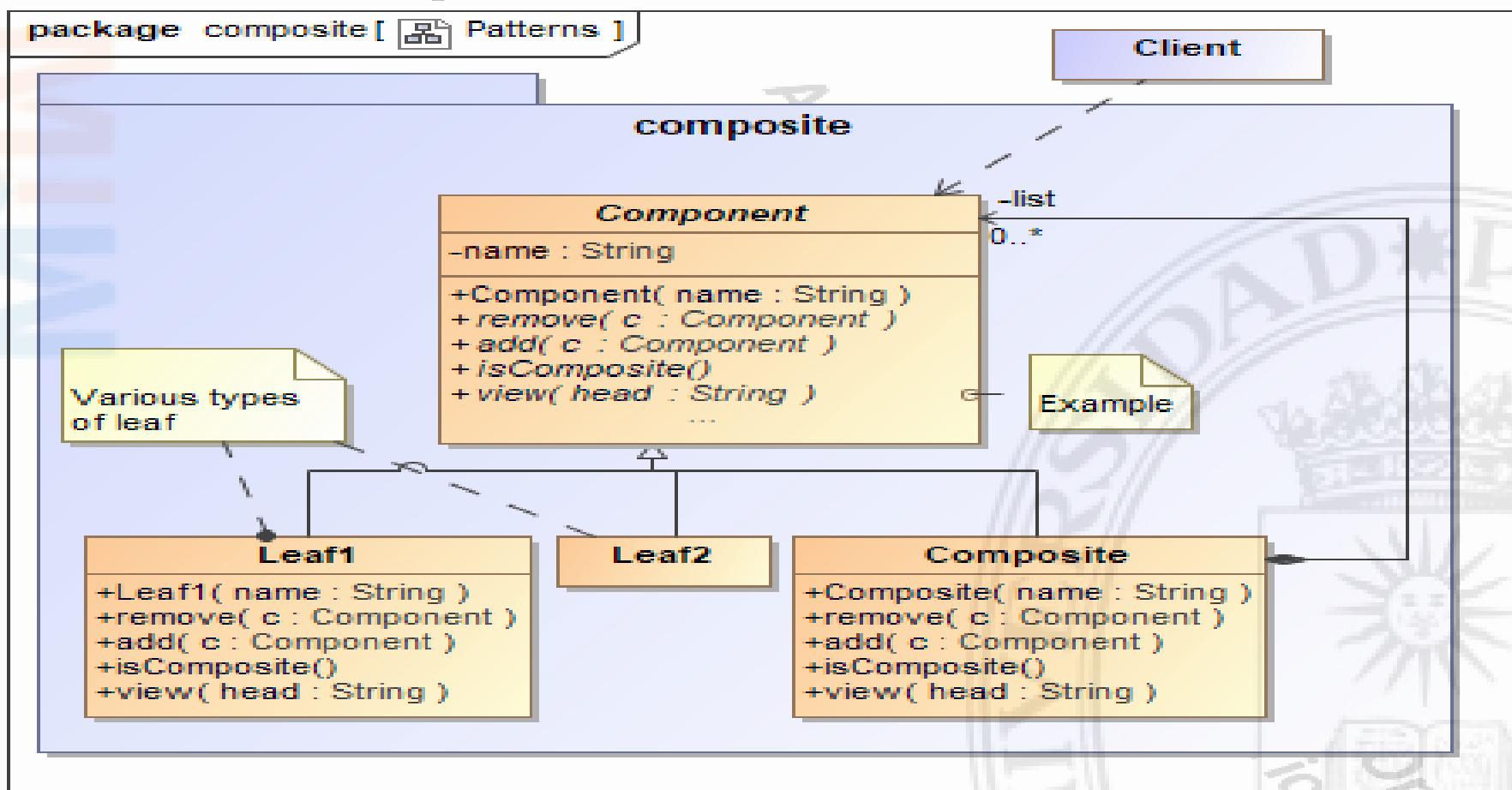
- Clase *TreeNumbers*, Clase *TreeNumbersTest*
- ¿Tiene alta cohesión? ¿Cumple los principios del DOO?
 - **SOLID** (*Single responsibility*), Anti-Pattern(*Spaghetti code*)
- ¿Y si se quiere añadir un nuevo tipo de hoja ☹?



Composite (Compuesto)

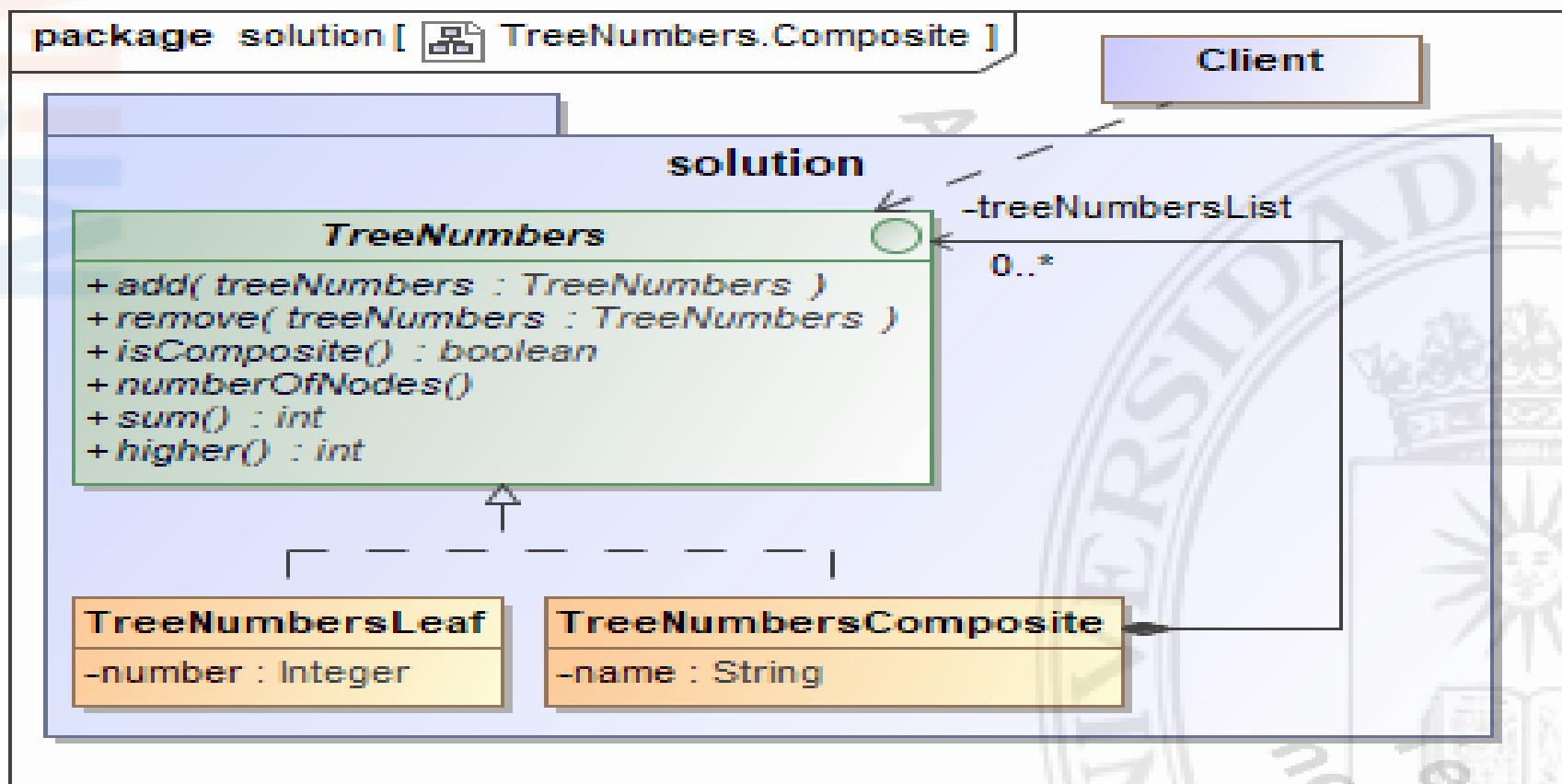
■ Propósito

- Permite estructuras en árbol tratando por igual a las hojas que a los elementos compuestos



TreeNumbers

- Roles:
 - Component: TreeNumbers
 - Composite: TreeNumbersComposite
 - Leaf: TreeNumbersLeaf

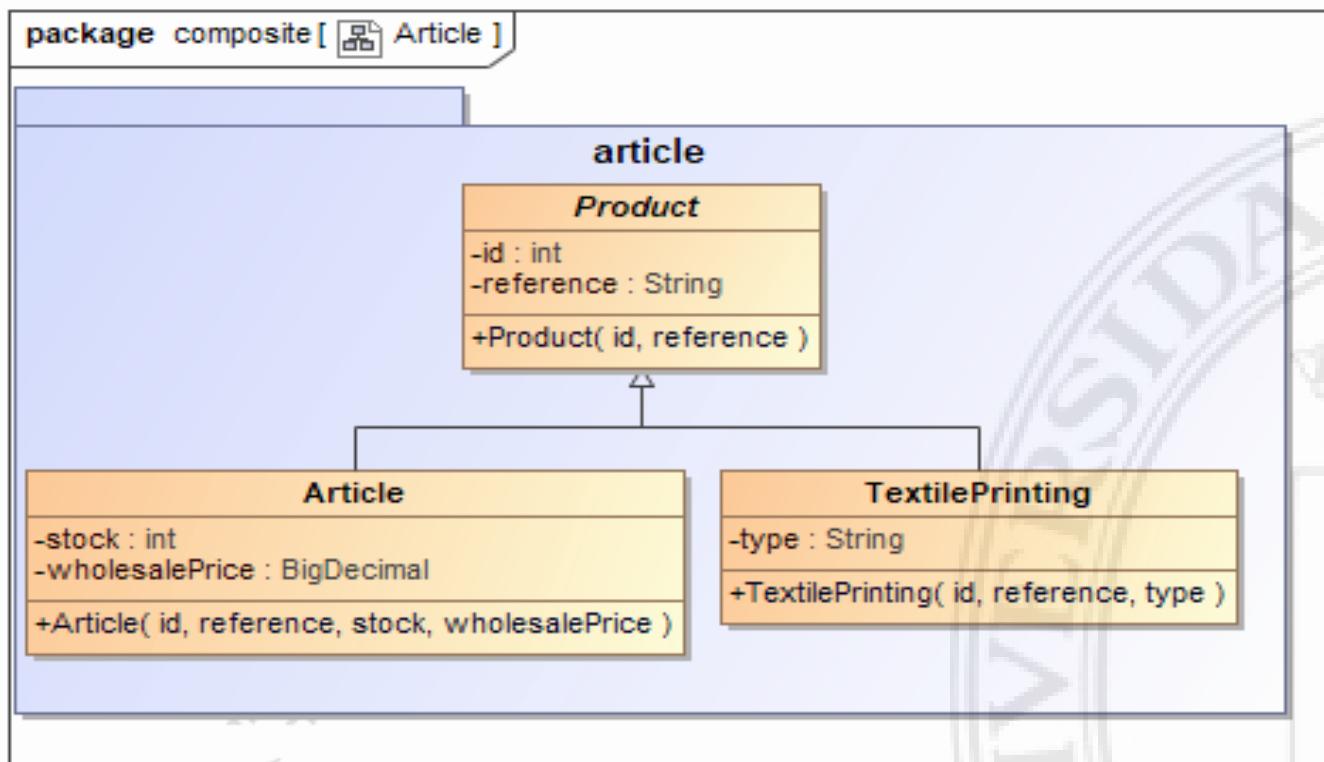


✍ Expression

- Se quiere construir un editor de expresiones matemáticas con valores enteros. Especificar el diagrama de clases que permita representar las expresiones
- Una expresión válida estará formada o bien por un número, o bien por la suma, resta, división o multiplicación de dos expresiones
- Ejemplos de expresiones válidas:
 - 5
 - $(1+(8*3))$
 - $((7+3) * (1+5))$

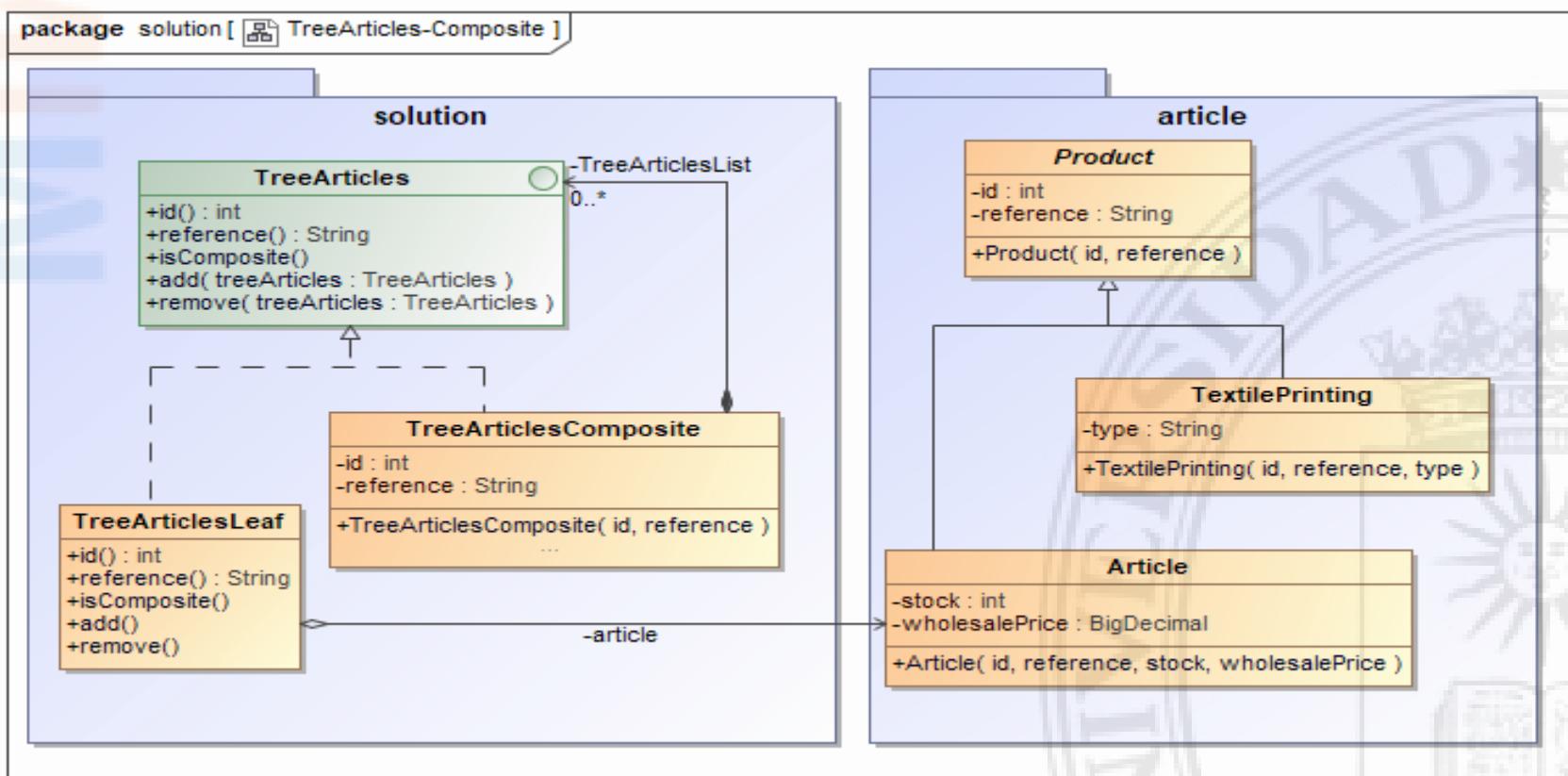
✍ Article

- Se parte de un código en producción, y **no se quieren alterar las clases existentes**
- Aplicar el patrón compuesto para realizar agrupaciones en árbol de solo artículos



✍ Article

- Se parte de un código en producción, y **no se quieren alterar las clases existentes**
- Aplicar el patrón compuesto para realizar agrupaciones en árbol de solo artículos



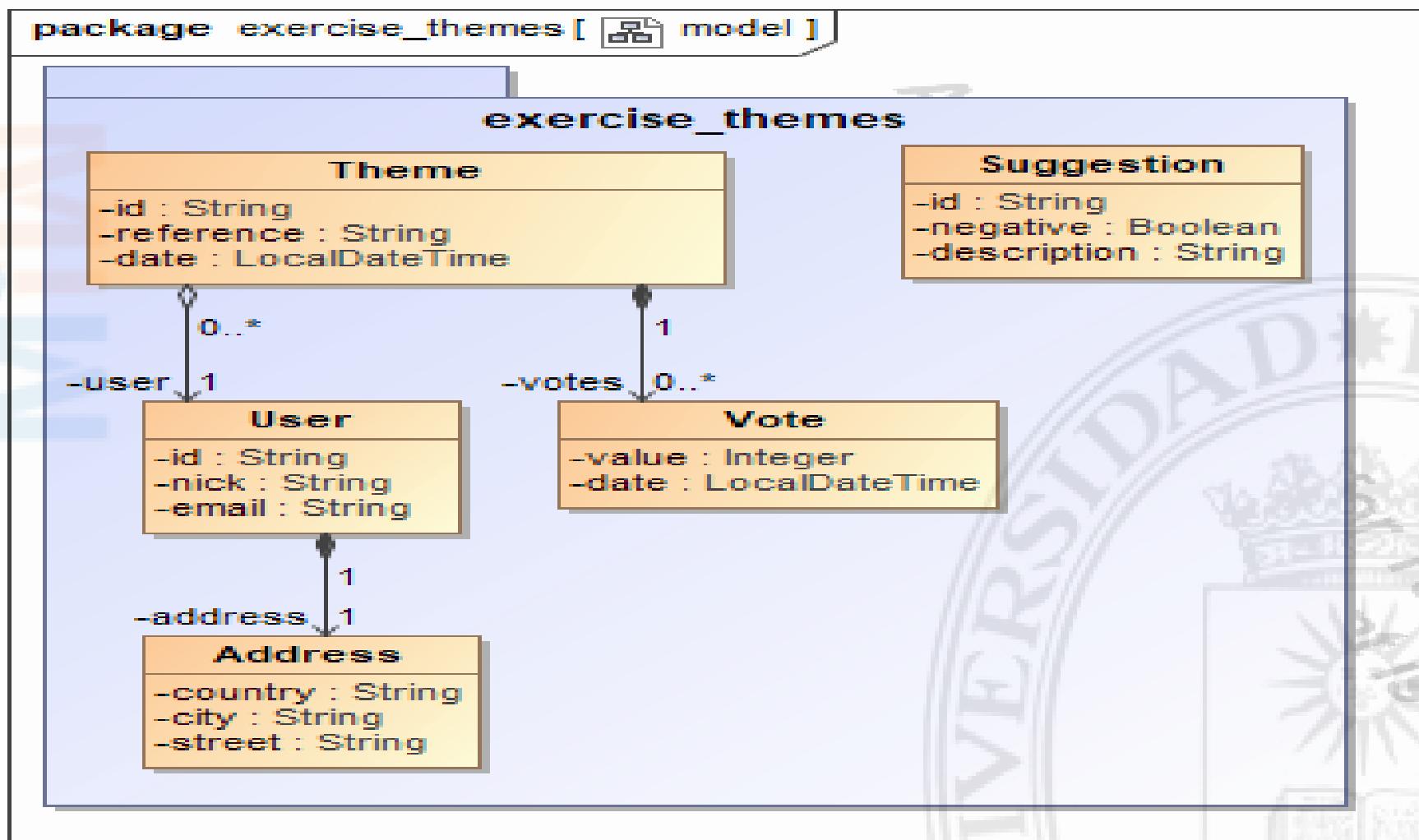
✍ Práctica de APAW

1. Establecer el esquema UML entre las entidades

- Cinco entidades, con un mínimo de 2 atributos, y si se guarda en BD, otro atributo *Id* de tipo *String*.
- Un mínimo de 15 atributos entre todas las entidades (incluidos *id*).
- Debe existir alguna entidad con relación de *n..1*.
- Debe existir alguna entidad con relación de *1..n*.
- Debe existir alguna entidad con relación *1..1* o *n..n*.
- Alguna entidad no debe tener relaciones
- Algún atributo con:
 - *LocalDateTime*
 - *Boolean*
 - *Integer*

✍ Práctica de APAW

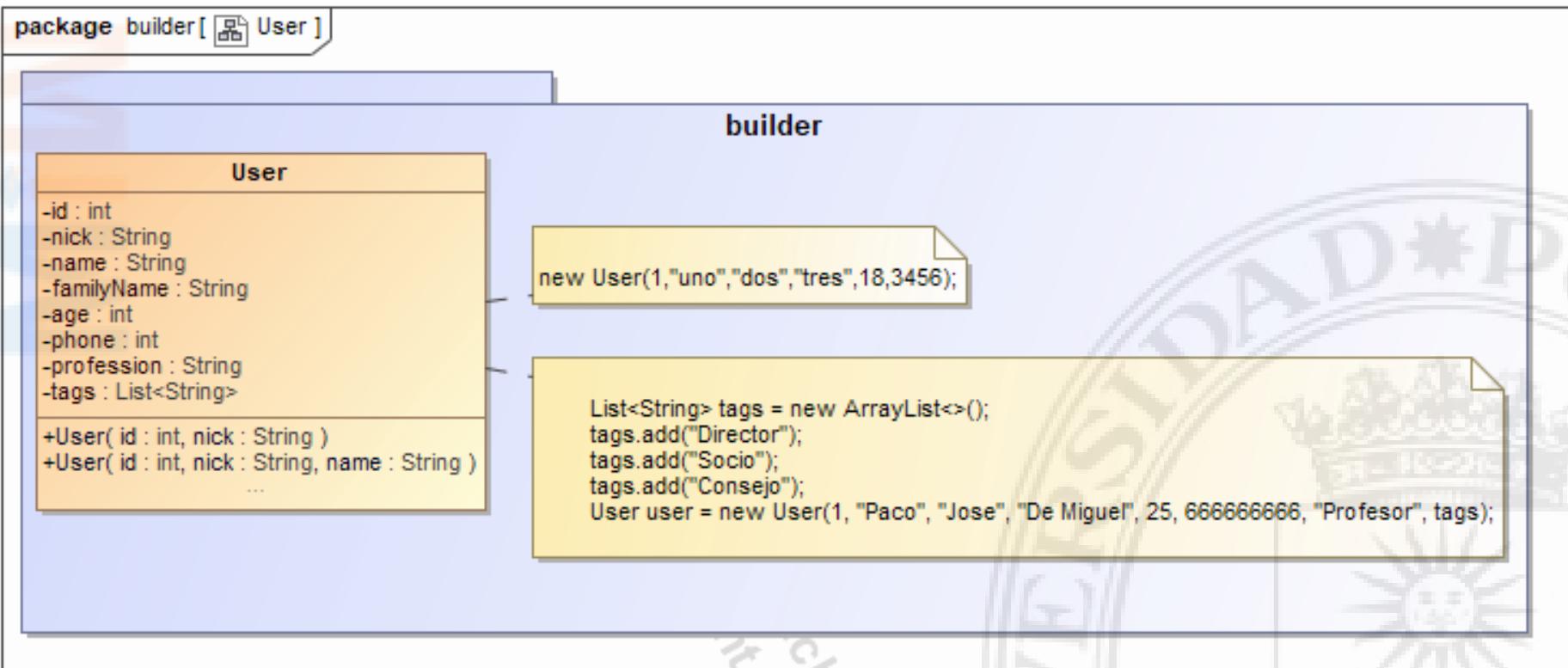
- Práctica a modo de ejemplo: *Themes*



Builder (Constructor)

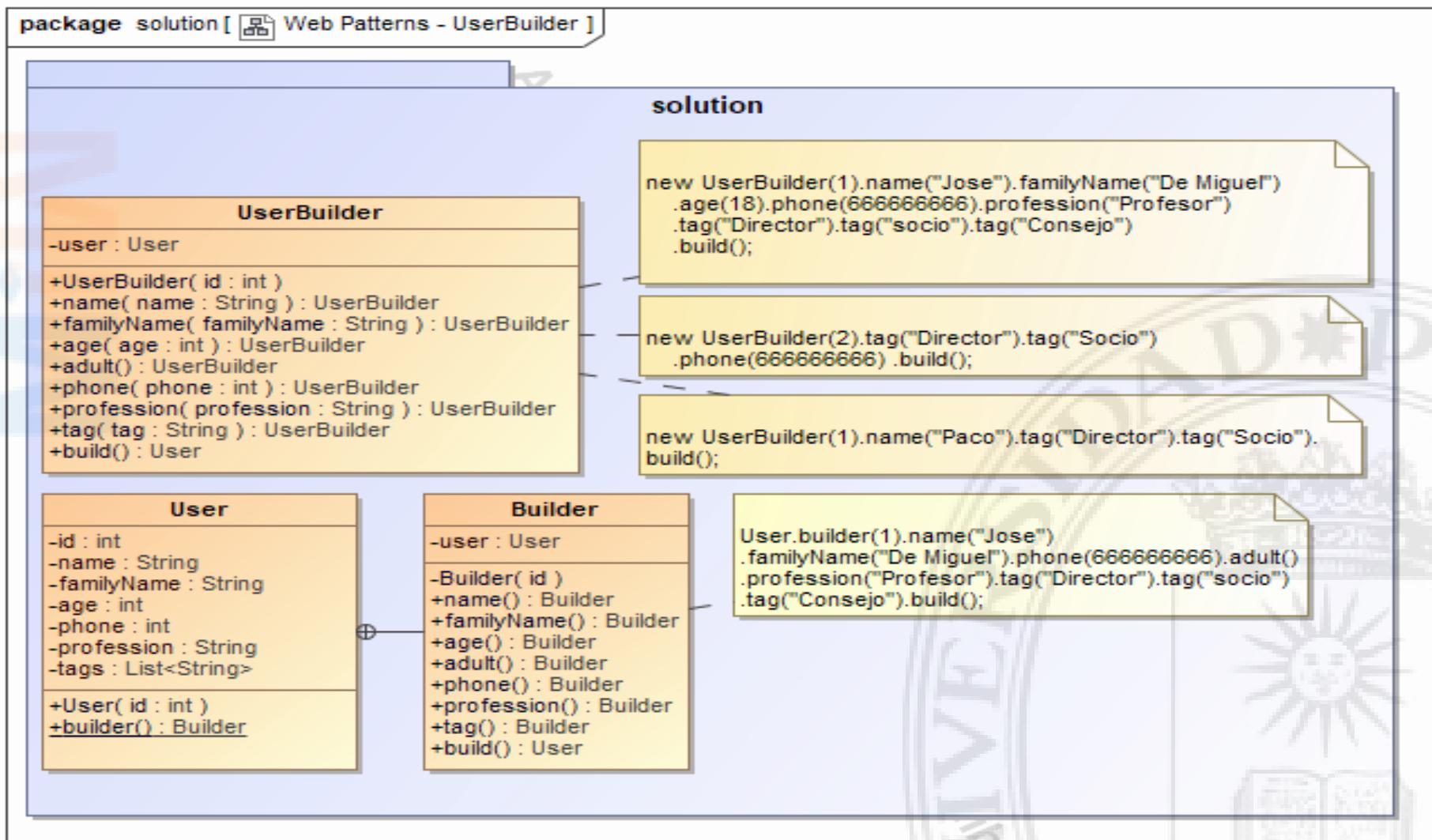
Motivación

- Este patrón permite tener una política general para la creación de objetos, centralizándolo en una clase constructora



Builder. user

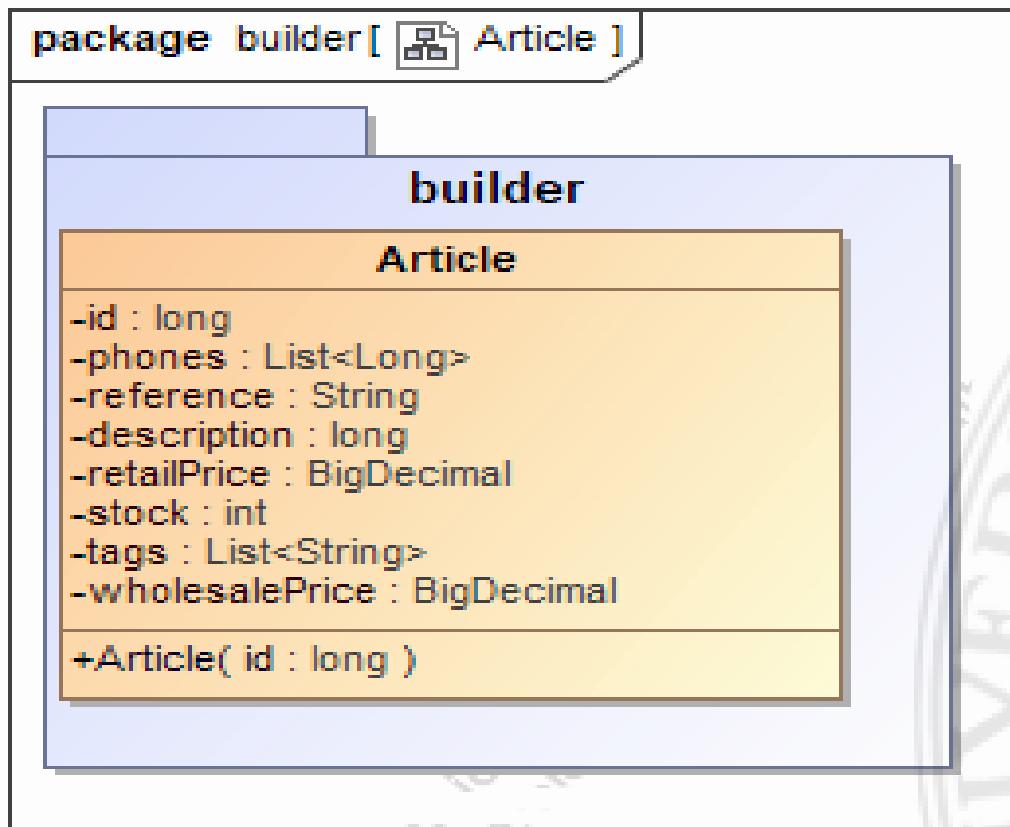
UserBuilder



✍ Article

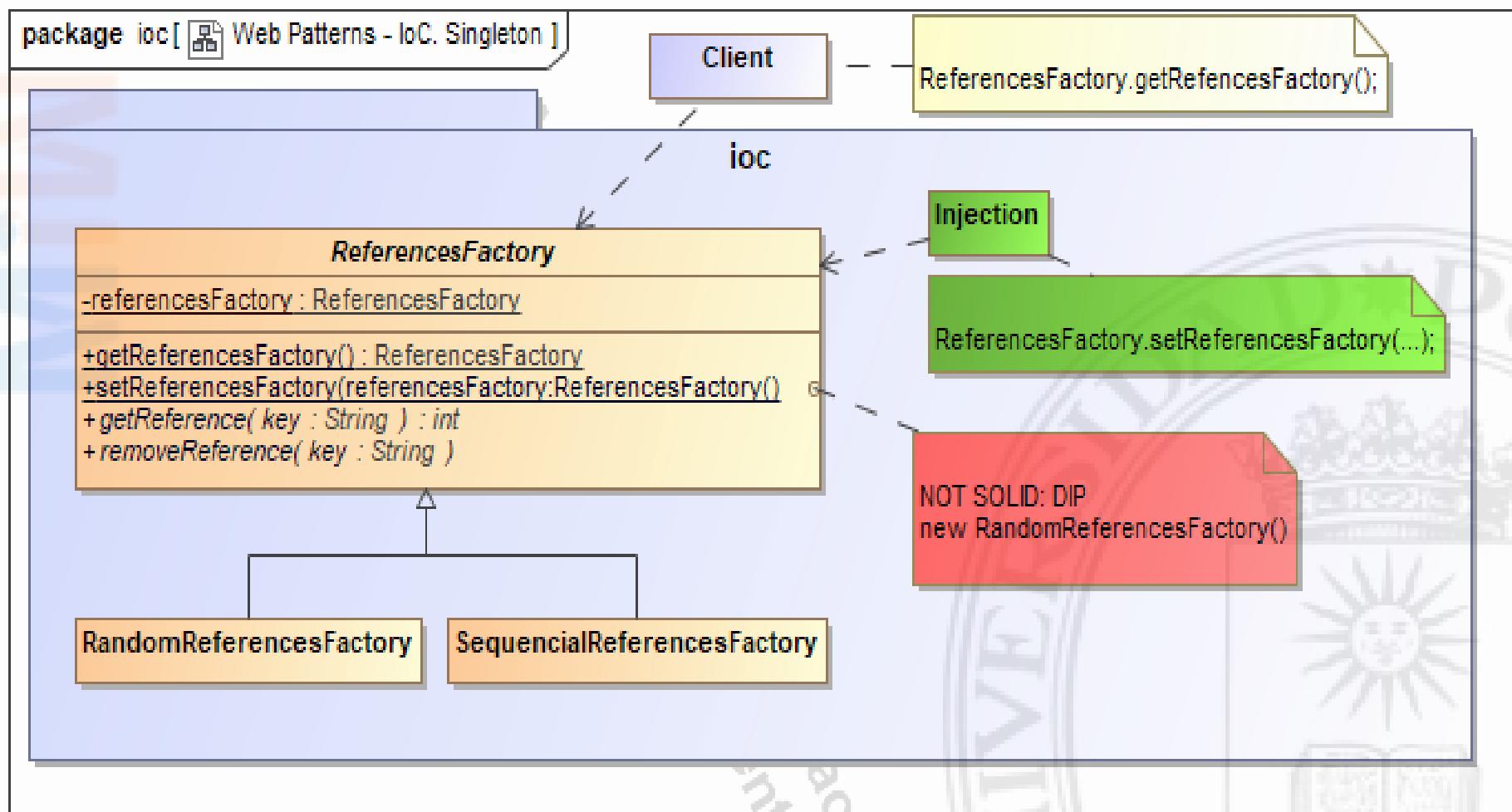
- Aplicar el patrón *Builder* a *Article*

- *Article.builder(id).build();*
- *Article.builder(id).tag("tag1").build();*
- *Article.builder(id).phone(555555555).reference("referencia").tag("tag1").tag("tag2").build();*



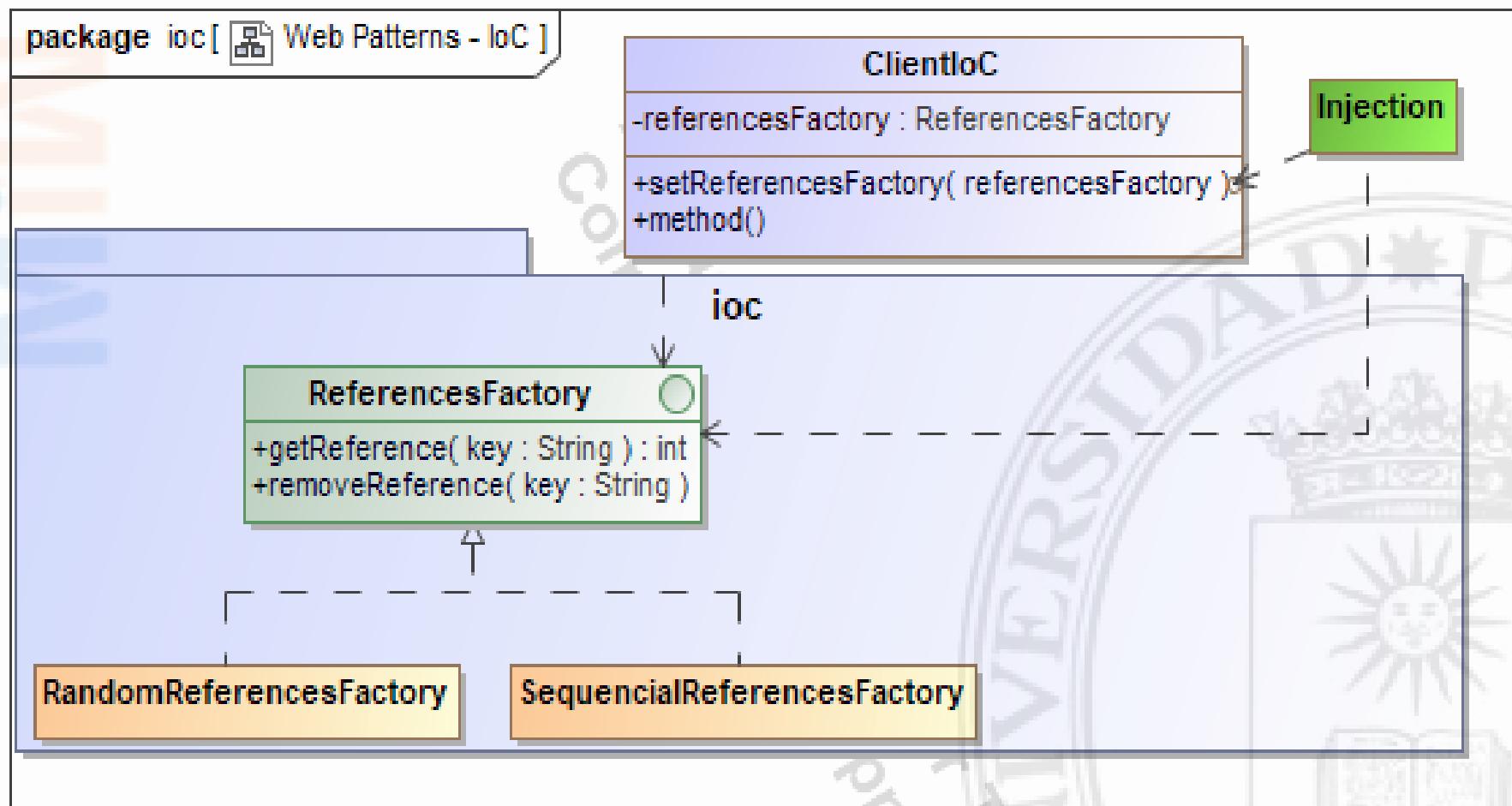
Inversion of control

- En una aplicación se necesita que exista un solo objeto de una clase *ReferencesFactory* (*abstract class*)



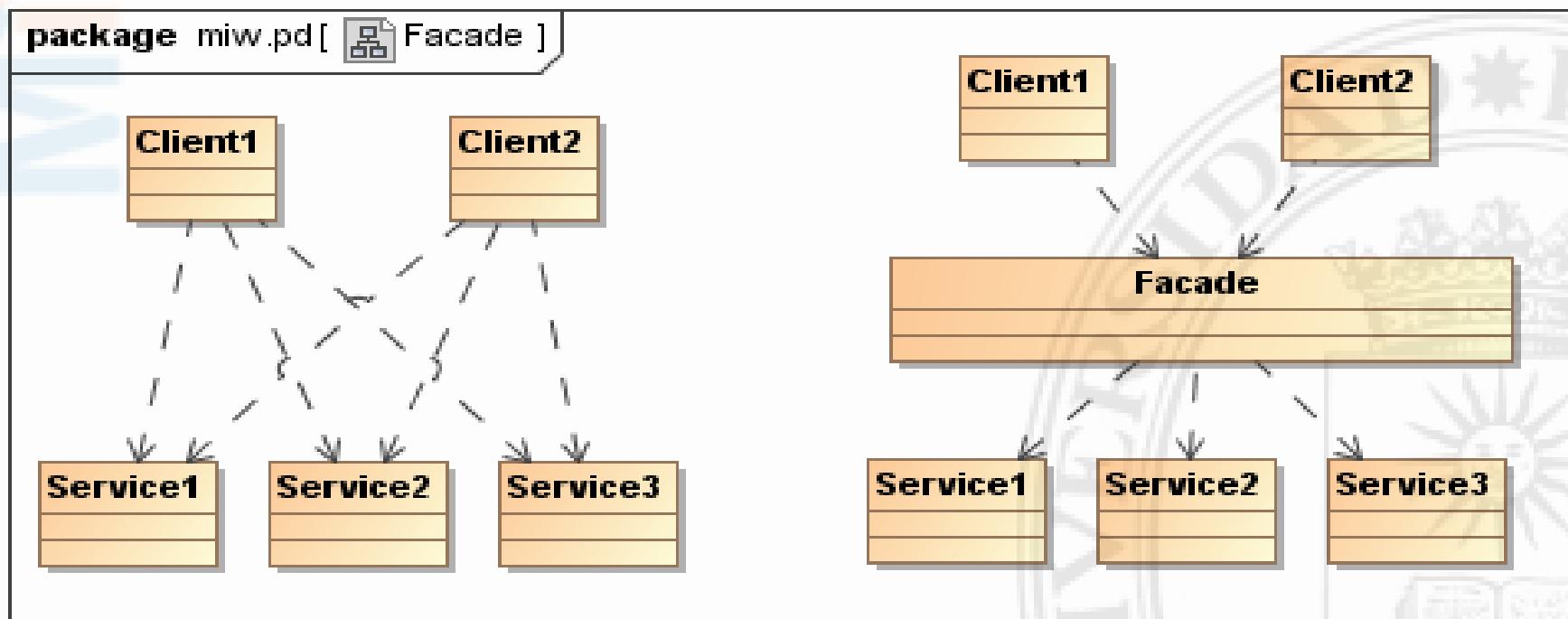
Inversion of control

- En una aplicación se necesita que exista un solo objeto de tipo *ReferencesFactory* (*interface*)



Facade (Fachada)

- Motivación
 - Dado un subsistema complejo, se quiere ofrecer una interface única y simplificada que ayude a dar servicios generales
 - Cuando se quiera estructurar varios subsistemas en capas, y se requiere simplificar el punto de entrada en cada nivel
- Propósito
 - Proporciona un interface unificado para un conjunto de interfaces de un subsistema
- *En cualquier librería de terceros, se debe aplicar una Fachada*



Abstract Factory (Fábrica abstracta)

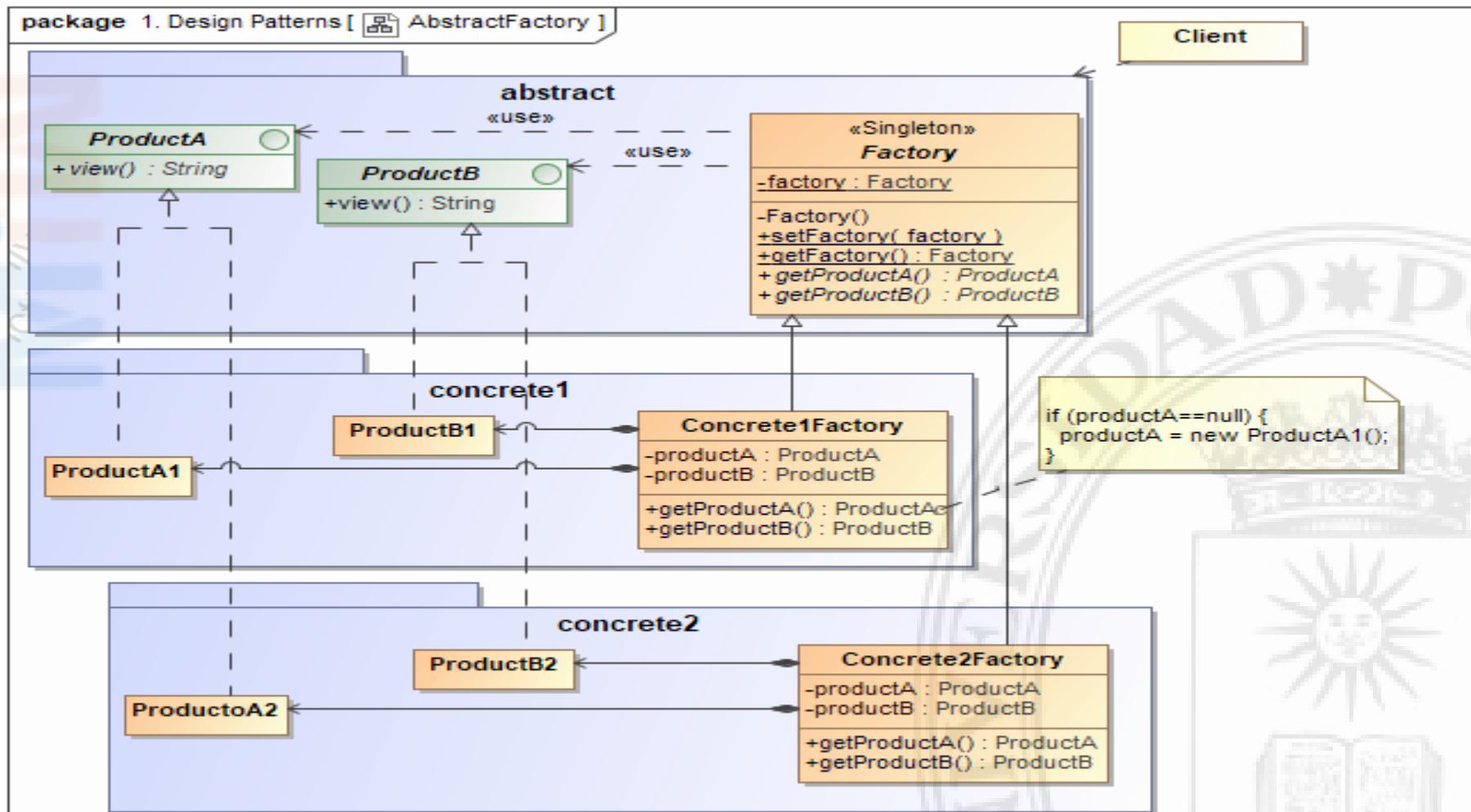
- También conocido
 - Kit
- Motivación
 - Se permiten múltiples interface de usuario
- Fuentes: paquete abstractFactory



Abstract Factory. Implementación

■ Propósito

- Proporciona un interface para crear familias de objetos relacionadas

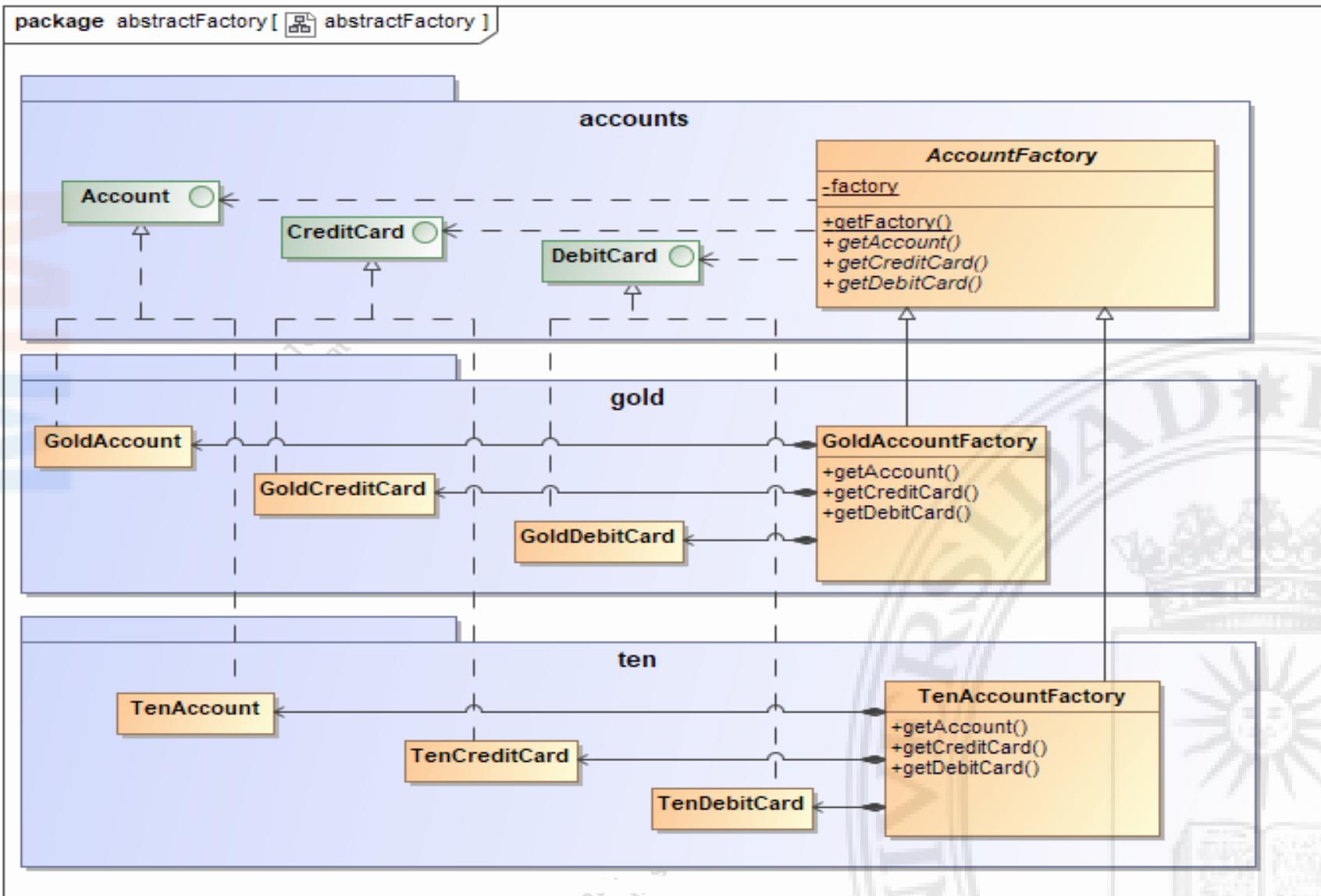


✍ Cuentas

- Fabricas: *Joven*, *10* y *Oro*
- Productos: *Cuenta*, *Tarjeta débito* y *Tarjeta crédito*
- Métodos de productos: *show(): String*
- *Total: 16 entre clases e interfaces*

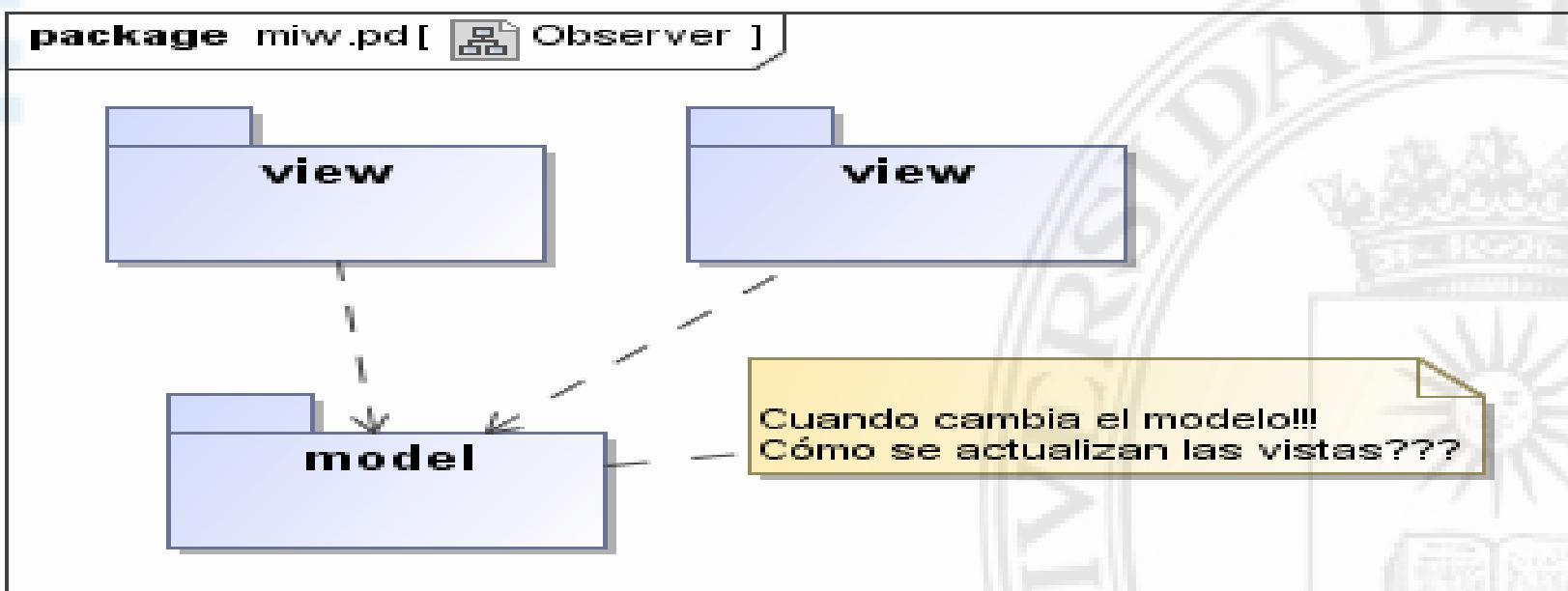
Tipo Cuenta	Cuenta	Tarjeta débito	Tarjeta crédito
Joven	1%	Gratuita	Gratuita Max.600
10	1'5%	Gratuita	10 € Max. 2000€
Oro	2%	5 €	20€ Max. 4000€

✍ Cuentas



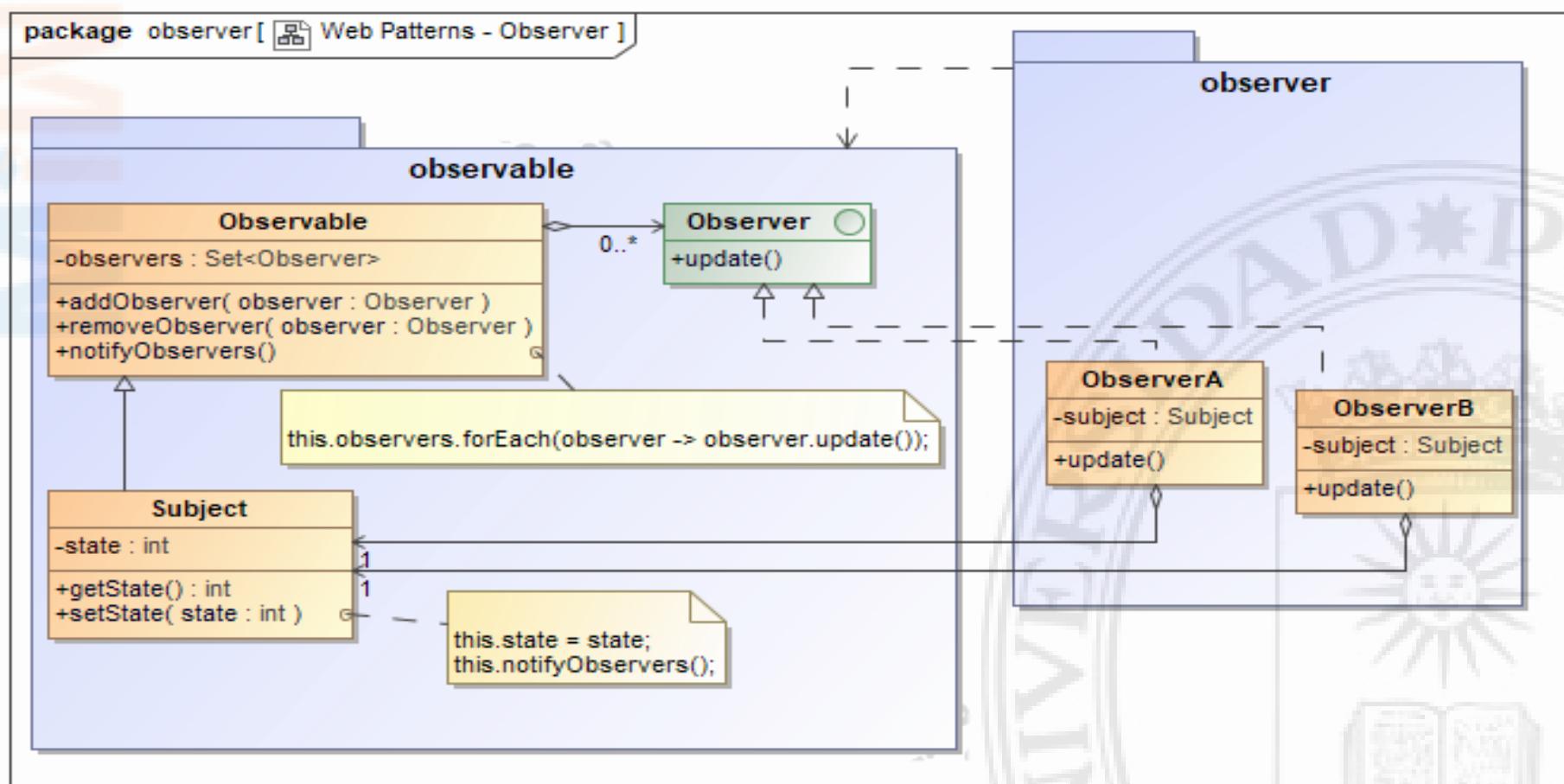
Observer (Observador)

- También conocido
 - Dependents (Dependiente), Publish-Subscribe (Publicar-Suscribir)
- Motivación
 - Muchas veces se separa los datos en si de su representación (MVC), pudiendo tener varias representaciones de un mismo dato
- Fuentes: paquete observer



Observer (Observador)

- Propósito
 - Se define una dependencia entre uno a muchos, del tal manera, que cuando cambie avise a todos los objetos dependientes



Lambda

```
c Lambda.java x
11  public class Lambda {
12      public Consumer<String> functionConsumer() { // accept(T)
13          return msg -> LogManager.getLogger(this.getClass()).info( s: "Consumer: " + msg);
14      }
15      public Consumer<String> functionConsumer2() { // accept(T)
16          return (String msg) -> LogManager.getLogger(this.getClass()).info( S: "Consumer: " + msg);
17      }
18      public Function<String, Integer> function() { // apply(T) :R
19          return msg -> Integer.parseInt(msg);
20      }
21      public Function<String, Integer> function2() { // apply(T) :R
22          return msg -> {
23              msg.concat("...Concat");
24              return Integer.parseInt(msg);
25          };
26      }
27      public Function<String, Integer> function3() { // apply(T) :R
28          return Integer::new;
29      }
30      public Predicate<String> functionPredicate() { // test(T) :boolean
31          return "two)::equals;
32      }
33      public Predicate<String> functionPredicate2() { // test(T) :boolean
34          return msg -> "two".equals(msg);
35      }
36      public Supplier<String> functionSupplier() { // get(): T
37          return () -> {
38              String prefix = "...";
39              return prefix.concat("-");
40          };
41      }
42  }
```

Lambda

```
class LambdaTest {  
    @Test  
    void testFunctionConsumer() {  
        Stream.of("one", "two", "three").forEach(new Lambda().functionConsumer());  
        Stream.of("one", "two", "three").forEach(msg -> LogManager.getLogger(this.getClass()).info( $: "Consumer: " + msg));  
    }  
    @Test  
    void testFunction() {  
        List<Integer> list1 = Stream.of("1", "2", "3").map(new Lambda().function()).collect(Collectors.toList());  
        List<Integer> list2 = Stream.of("1", "2", "3").map(Integer::new).collect(Collectors.toList());  
        LogManager.getLogger(this.getClass()).info( $: "Function: " + list1 + ", " + list2);  
    }  
    @Test  
    void testPredicate() {  
        List<String> list1 = Stream.of("one", "two", "three")  
            .filter(new Lambda().functionPredicate()).collect(Collectors.toList());  
        List<String> list2 = Stream.of("one", "two", "three")  
            .filter("two"::equals).collect(Collectors.toList());  
        LogManager.getLogger(this.getClass()).info( $: "Predicate: " + list1 + ", " + list2);  
    }  
    @Test  
    void testSupplier() {  
        List<String> list1 = Stream.generate(new Lambda().functionSupplier()).limit(3).collect(Collectors.toList());  
        List<String> list2 = Stream.generate(() -> {  
            String prefix = "...";  
            return prefix.concat("-");  
        }).limit(3).collect(Collectors.toList());  
        LogManager.getLogger(this.getClass()).info( $: "Supplier: " + list1 + ", " + list2);  
    }  
}
```

Publisher-Subscribe

- Proyecto: *Reactor*
 - <https://projectreactor.io/docs/core/release/reference>
- Publisher
 - Un **Mono<T>** es un publicador
 - Emite de forma asíncrona 0 o 1 elemento y termina con una señal *onComplete*.
 - Termina con una señal *onError*.
 - Un **Flex<T>** es un publicador
 - Emite una secuencia asíncrona de 0 a N elementos y termina con una señal *onComplete*.
 - Termina con una señal *onError*.
- Subscribe
 - `publisher.subscribe(`
 - **Consumer<? super T> consumer,**
 - **Consumer<? super Throwable> errorConsumer,**
 - **Runnable completeConsumer**
 - **Consumer<? super Subscription> subscriptionConsumer);**

Publisher-Subscribe

■ Creación

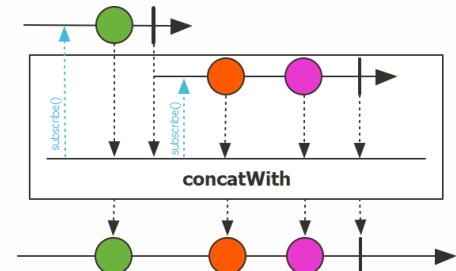
- `Mono.empty();`
- `Mono.just("one"); //se ejecuta directamente, sin retraso`
- `Mono.error(new RuntimeException("mono-error"));`
- `Mono.delay(Duration.ofSeconds(2)).map(value -> "One");`
- `Flux.just("one", "two", "three");`
- `Flux.interval(Duration.ofMillis(200)).map(value -> "A" + value).take(8);`

■ Subscripción

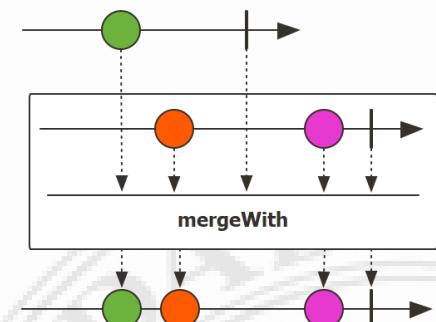
- `publisher.subscribe(onNext, onError, onCompleted, onSubscribe);`
- **Asíncrona**
 - `s1;`
 - `publisher.subscribe(msg -> System.out.println(msg)); // asynchronous`
 - `s3;`
- **Síncrono**
 - `s1;`
 - `String msg = publisher.block(); // synchronous`
 - `s3;`

Publisher-Subscribe

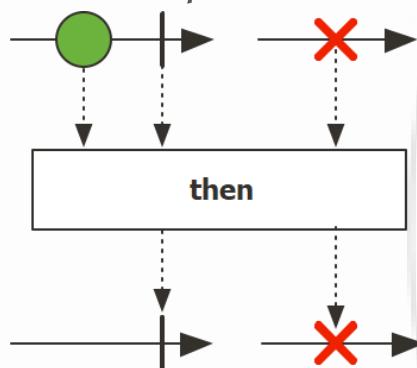
- Concatenación
 - `publisher=publisher1.concatWith(publisher2);`



- Unión (se entremezcla)
 - `publisher=publisher1.mergeWith(publisher2);`



- Sincronización
 - `publisher=Mono.when(publisher1, publisher2, publisher3); //(...)`



Publisher-Subscribe



```
ReactiveProgrammingTest.java x
10  class ReactiveProgrammingTest {
11      @Test
12      void testMonoEmpty() {
13          new ReactiveProgramming().monoEmpty().subscribe(
14              msg -> LogManager.getLogger(this.getClass()).info("Consumer: " + msg), //onNext
15              throwable -> LogManager.getLogger(this.getClass()).info("Error: " + throwable.getMessage()),
16              () -> LogManager.getLogger(this.getClass()).info("Completed") //onComplete
17          );
18      }
19      @Test
20      void testMonoOne() {
21          LogManager.getLogger(this.getClass()).info("Before Mono.just...");
22          new ReactiveProgramming().monoOne().subscribe(
23              msg -> LogManager.getLogger(this.getClass()).info("Consumer: " + msg)
24          );
25          LogManager.getLogger(this.getClass()).info("... After Mono.just");
26      }
27      @Test
28      void testMonoError() {
29          new ReactiveProgramming().monoError().subscribe(
30              msg -> LogManager.getLogger(this.getClass()).info("Consumer: " + msg),
31              throwable -> LogManager.getLogger(this.getClass()).info("Error: " + throwable.getMessage()),
32              () -> LogManager.getLogger(this.getClass()).info("Completed")
33          );
34      }
35      @Test
36      void testMonoDelayBlock() {
37          LogManager.getLogger(this.getClass()).info("Before Mono.delay...");
38          String msg = new ReactiveProgramming().monoDelay().block();
39          LogManager.getLogger(this.getClass()).info("... After Mono.delay. msg:" + msg);
40      }
}
```

Publisher-Subscribe

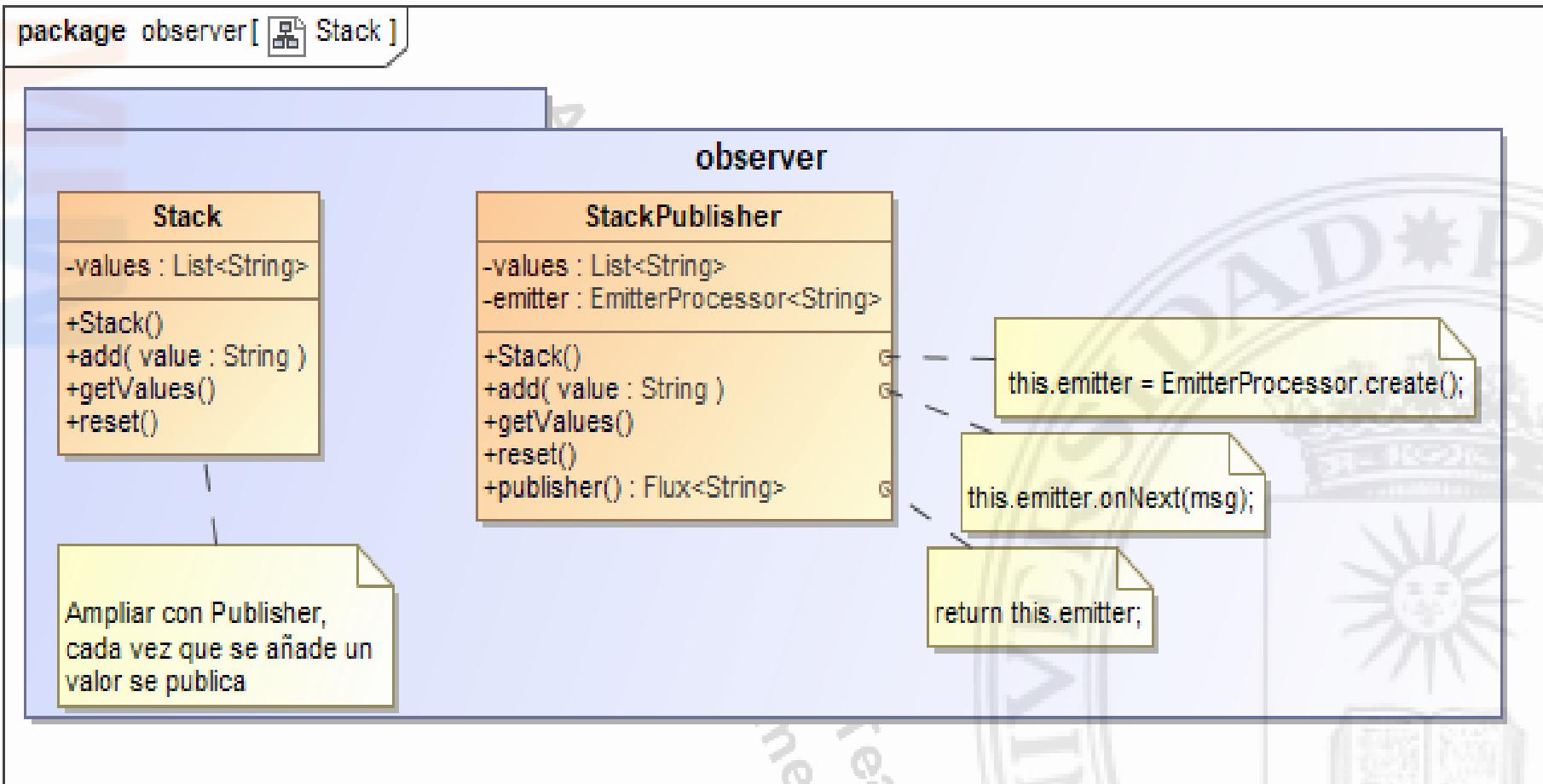


The screenshot shows a Java code editor with a file named `ReactiveProgrammingTest.java`. The code contains several test methods using the `StepVerifier` class to verify the behavior of reactive publishers.

```
43     void testMonoDelay() {
44         LogManager.getLogger(this.getClass()).info("Before Mono.delay...");
45         Mono<String> publisher = new ReactiveProgramming().monoDelay();
46         publisher.subscribe(msg -> LogManager.getLogger(this.getClass()).info("Consumer: " + msg));
47         LogManager.getLogger(this.getClass()).info("... After Mono.delay");
48         publisher.block();
49     }
50
51     @Test
52     void testMonoDelayStepVerifier() {
53         StepVerifier
54             .create(new ReactiveProgramming().monoDelay()) FirstStep<String>
55             .expectNext("One") Step<String>
56             .expectComplete() StepVerifier
57             .verify();
58     }
59
60     @Test
61     void testFluxDemoStepVerifier() {
62         StepVerifier
63             .create(new ReactiveProgramming().fluxDemo()) FirstStep<String>
64             .expectNext(t: "A0", t1: "A1", t2: "A2") Step<String>
65             .expectNextMatches(name -> name.startsWith("A")) //A3
66             .expectNextCount(4) //A4..7
67             .expectComplete() StepVerifier
68             .verify();
69
70     @Test
71     void testFluxDemoLimitByFlow() {
72         StepVerifier
73             .create(new ReactiveProgramming().fluxLimitByFlow()) FirstStep<String>
74             .expectNext(t: "B0", t1: "B1", t2: "B2") Step<String>
75             .expectComplete() StepVerifier
76             .verify();
77 }
```

Publisher

- Propósito
 - Se define una dependencia entre uno a muchos, del tal manera, que cuando cambie avise a todos los objetos dependientes

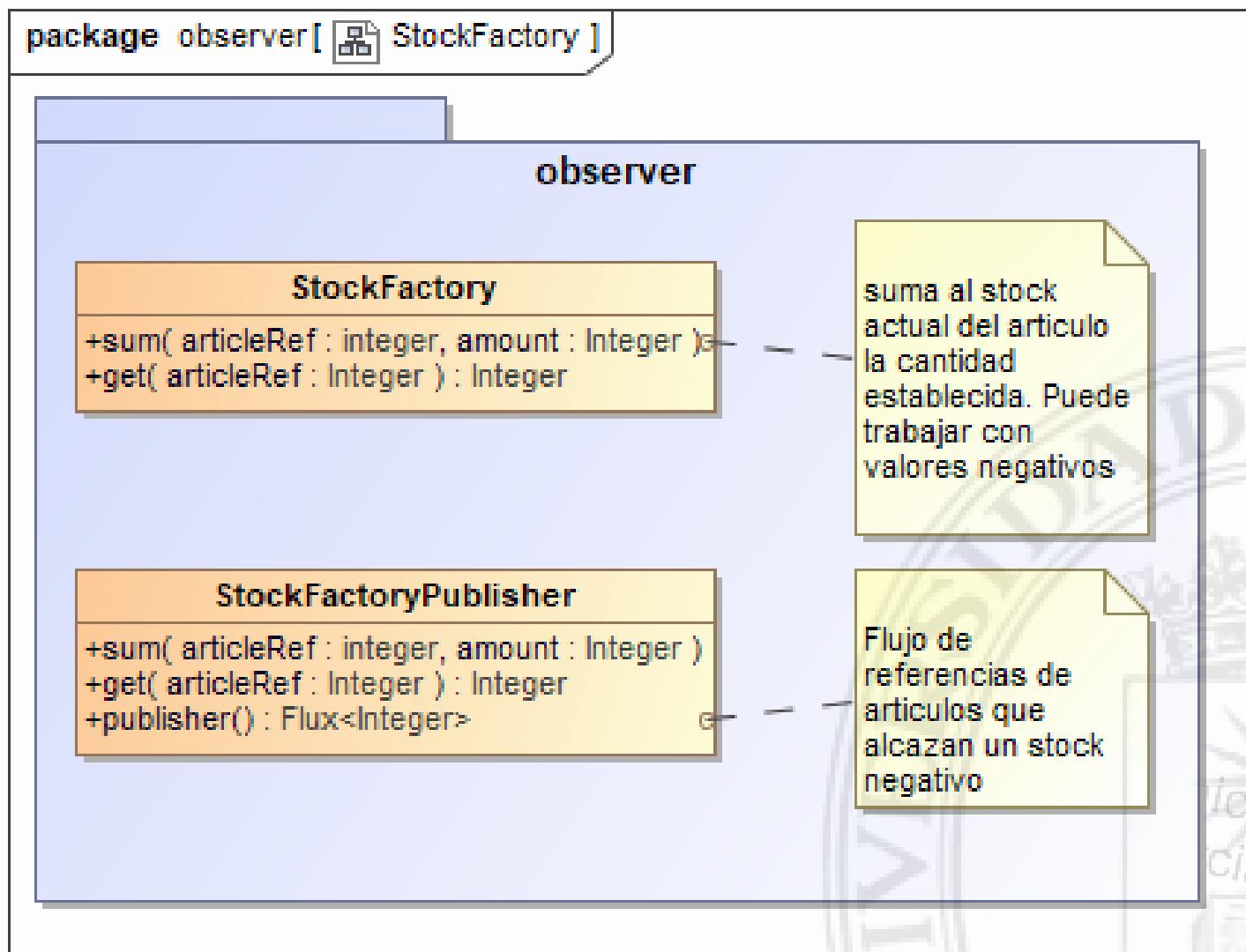


Publisher

```
c StackPublisherTest.java ×

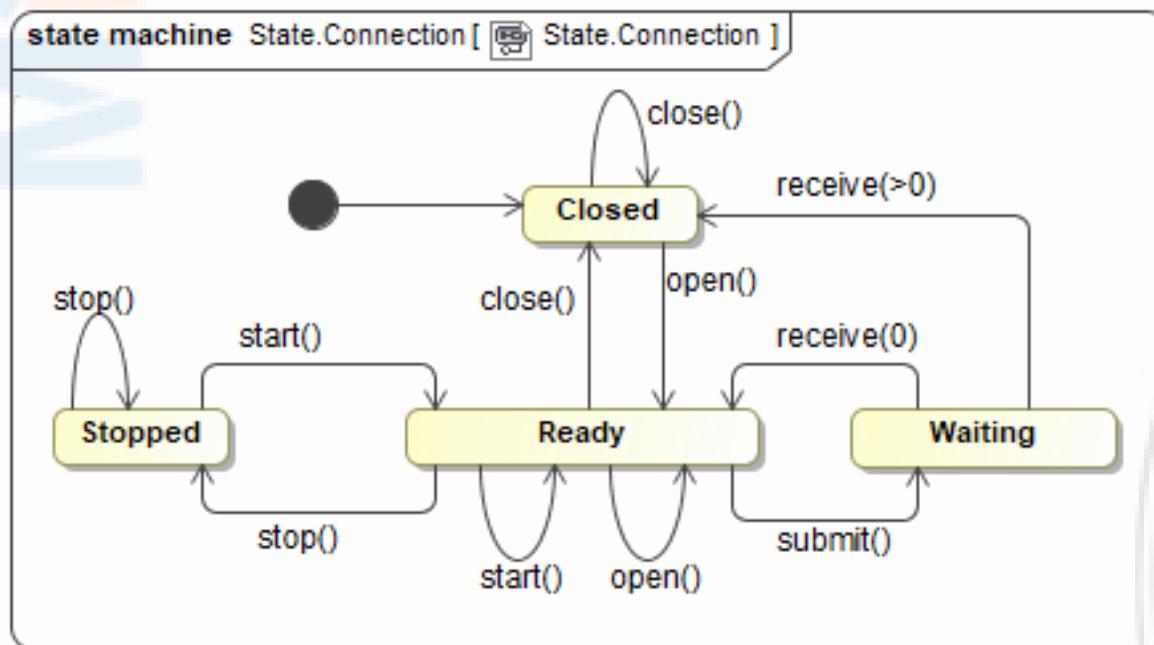
8  class StackPublisherTest {
9
10
11     @Test
12     void testStack() {
13         StackPublisher stackPublisher = new StackPublisher();
14         stackPublisher.add("one");
15         stackPublisher.add("two");
16         stackPublisher.add("three");
17         assertEquals( expected: 3, stackPublisher.getValues().size());
18         stackPublisher.reset();
19         assertEquals( expected: 0, stackPublisher.getValues().size());
20         stackPublisher.add("four");
21     }
22
23     @Test
24     void testStackPublisher() {
25         StackPublisher stackPublisher = new StackPublisher();
26         StepVerifier
27             .create(stackPublisher.publisher()) FirstStep<String>
28             .then(() -> stackPublisher.add("One")) Step<String>
29             .expectNext("One") Step<String>
30             .then(() -> stackPublisher.add("Two")) Step<String>
31             .expectNext("Two") Step<String>
32             .thenCancel() StepVerifier
33             .verify();
34     }
}
```

✍ Publisher



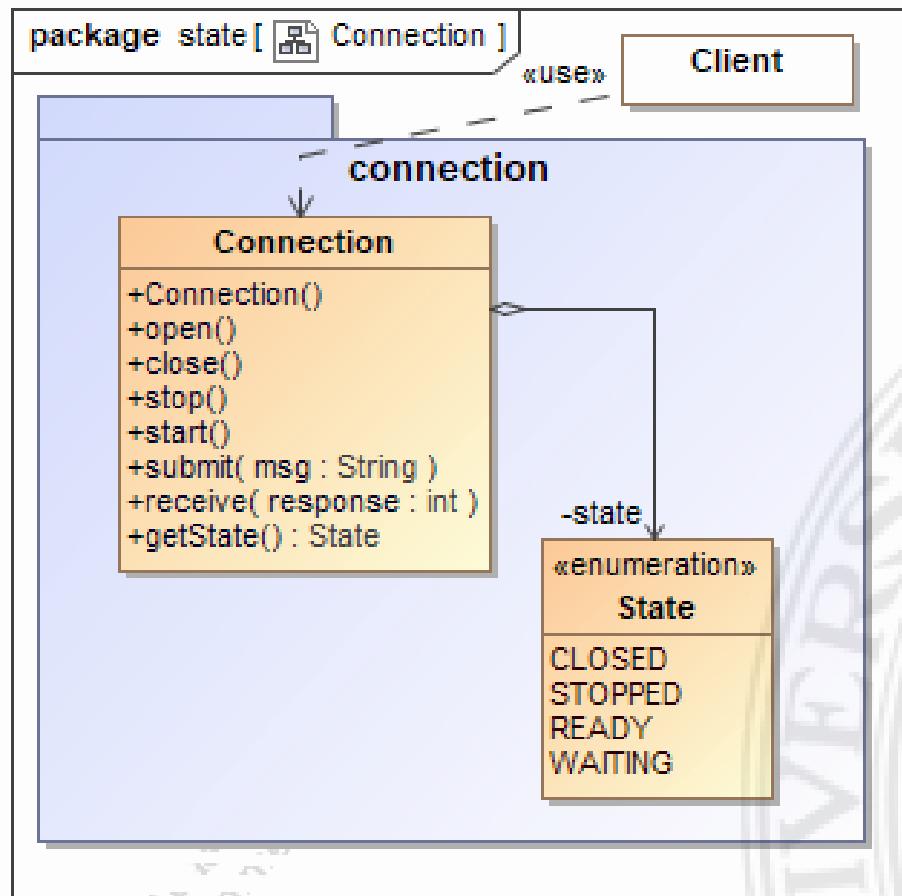
State (Estado)

- También conocido
 - Objects for States (Estados como Objetos)
- Fuentes: paquete state, paquete test state
- Motivación
 - Partiendo de una clase que representa una conexión TCP, la acción enviar debe tener diferentes respuestas dependiendo del estado de la conexión. Ello nos lleva al antipatrón: código espagueti



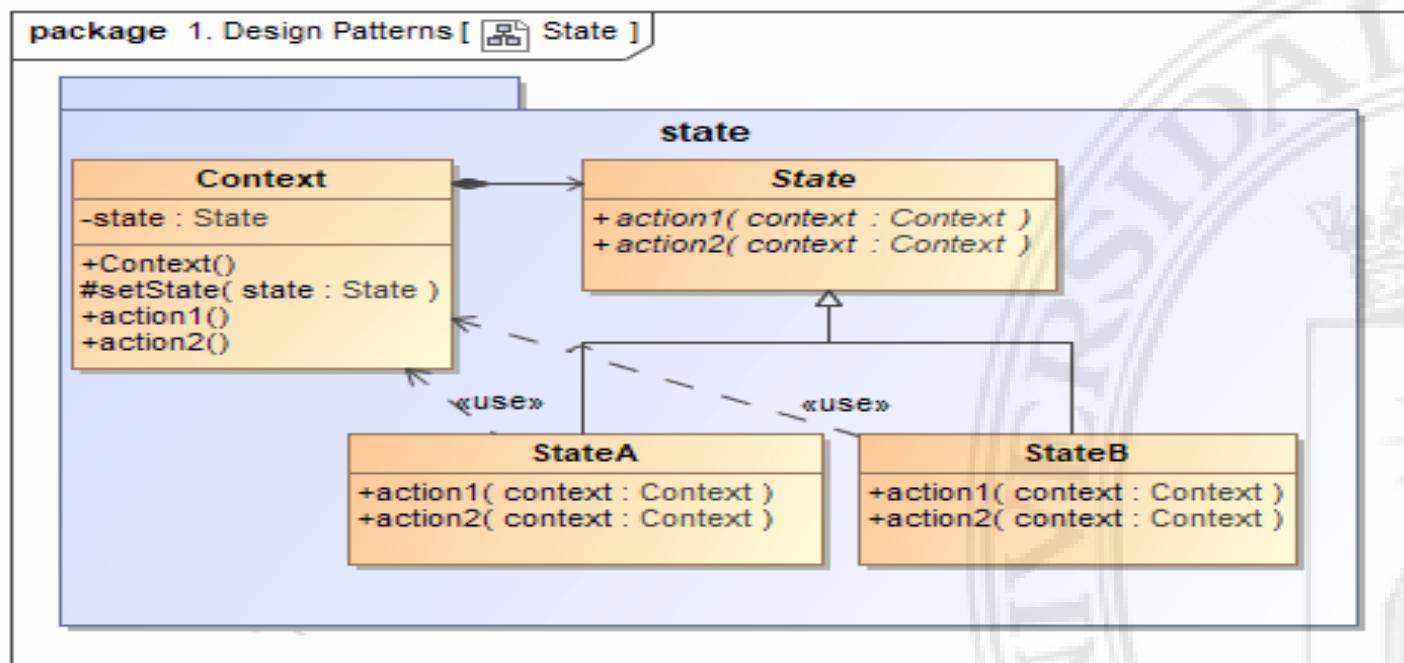
State (Estado)

- Sólo se permiten las acciones marcada, el resto debe lanzar la excepción: *UnsupportedOperationException*



State (Estado)

- Propósito
 - Permite que un objeto cambie su comportamiento cada vez que cambie su estado interno
- Creación de objetos:
 - Crearlos nuevos cada vez que se necesite
 - Crearlos al principio y no destruirlos

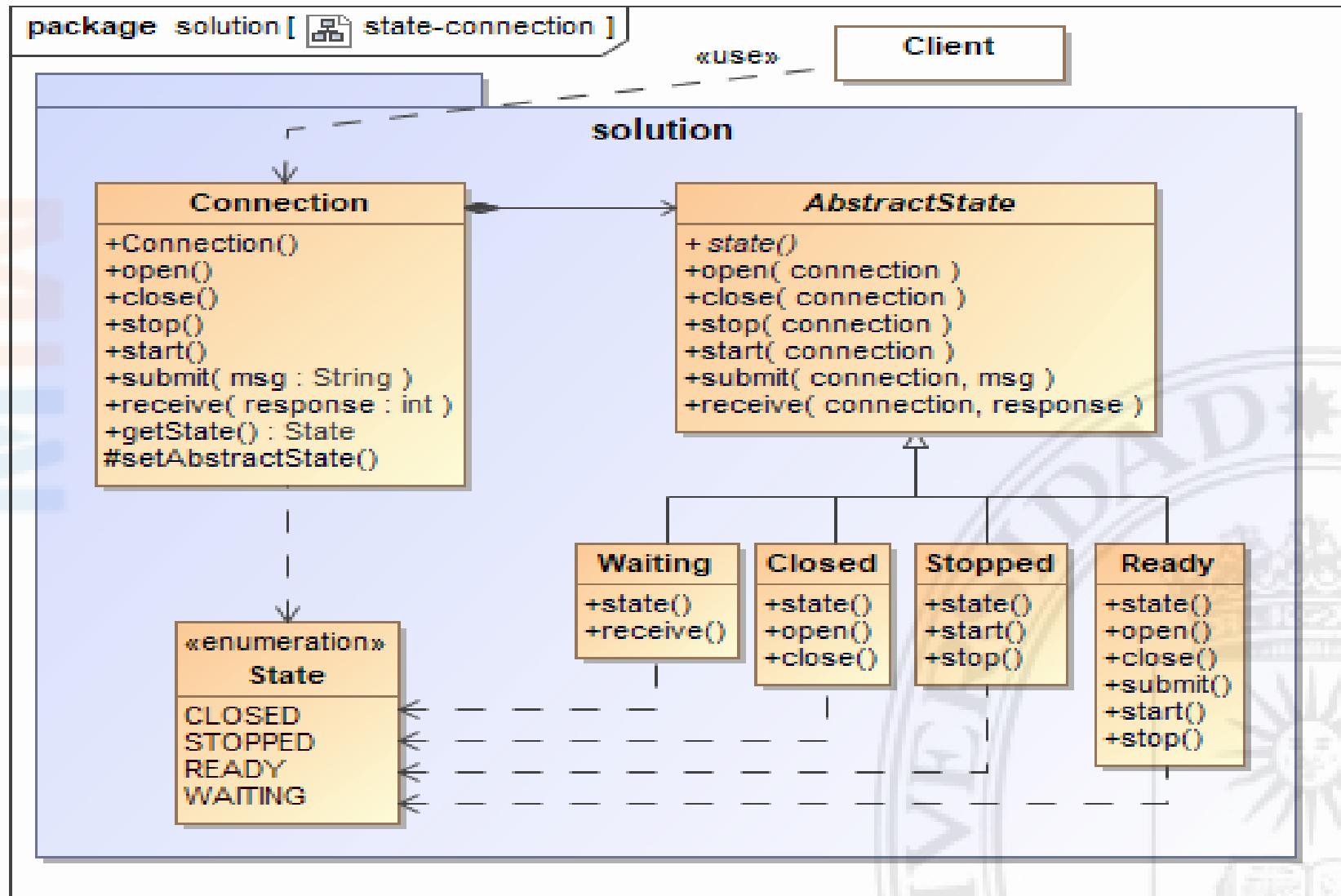




Connection

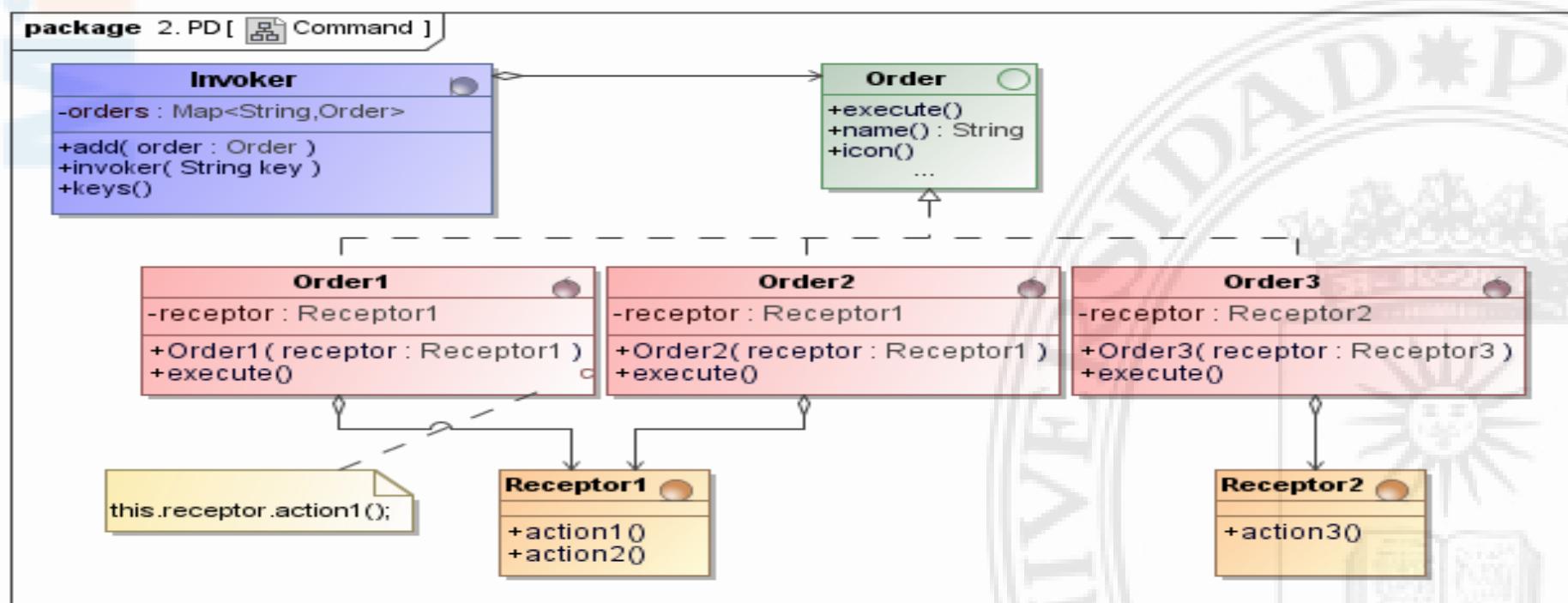
- Refactorizar la clase *Connection* para aplicar el patrón *State*
- Se dispone de *ConnectionTest*

Connection



Command (Orden)

- También conocido: Action (Acción), Transaction (Transacción)
- Motivación
 - A veces es necesario realizar peticiones a objetos sin conocer la petición ni a quien va dirigida
- Propósito
 - Se desacopla el objeto que invoca a la operación asociada, mediante un objeto. Ello permite realizar órdenes compuestas (patrón composite) o llevar una cola y deshacer operaciones

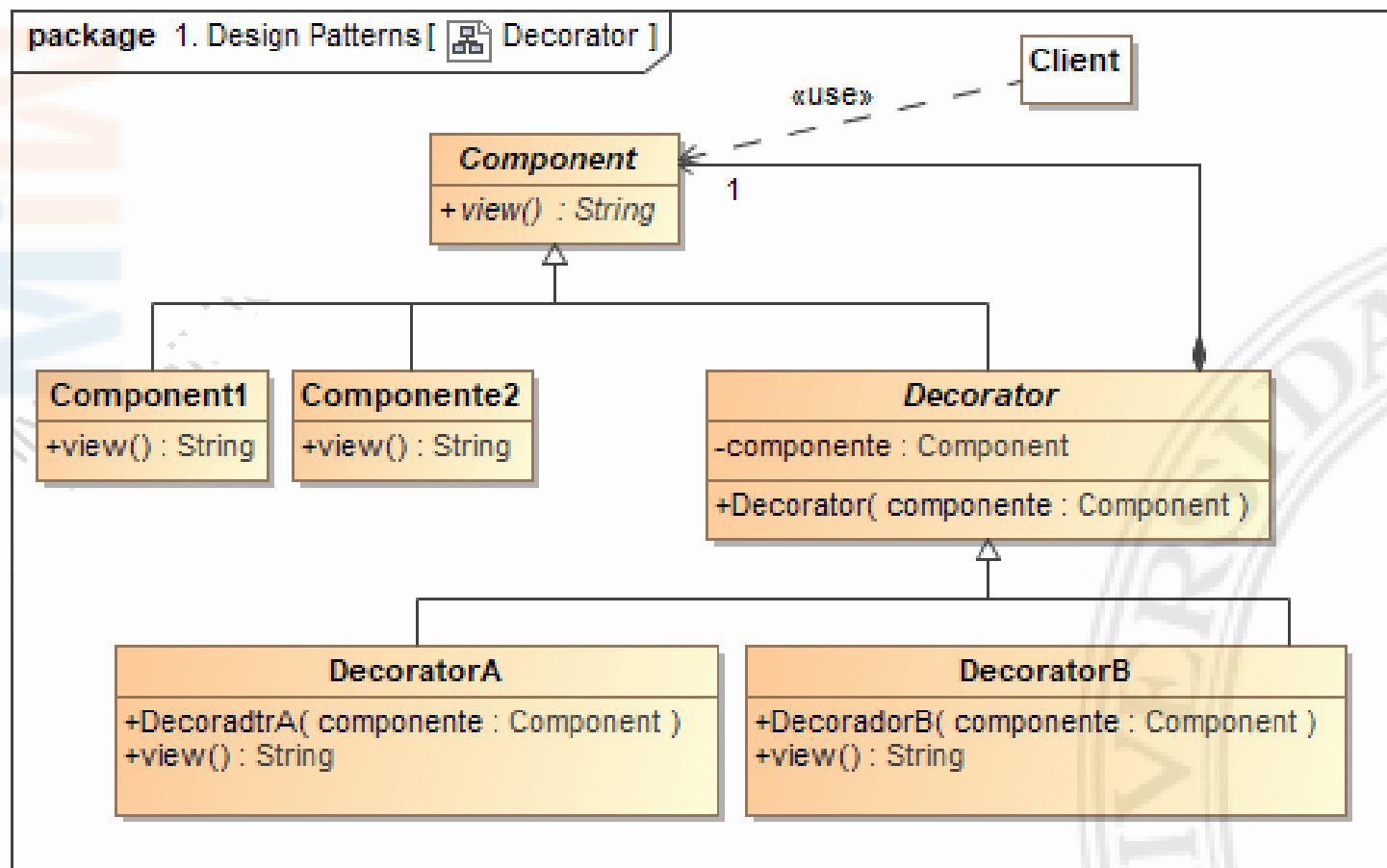


Decorator (Decorador)

- También conocido
 - Wrapper (Envoltorio)
- Motivación
 - Se pretende añadir una nueva responsabilidad a un objeto, pero no a su clase. Un ejemplo sería añadir barras de Scroll a un componente visual
- Fuentes: paquete decorator

Decorator (Decorador)

- Propósito
 - Asigna responsabilidades de forma dinámica a objetos, proporcionando una alternativa flexible a la herencia



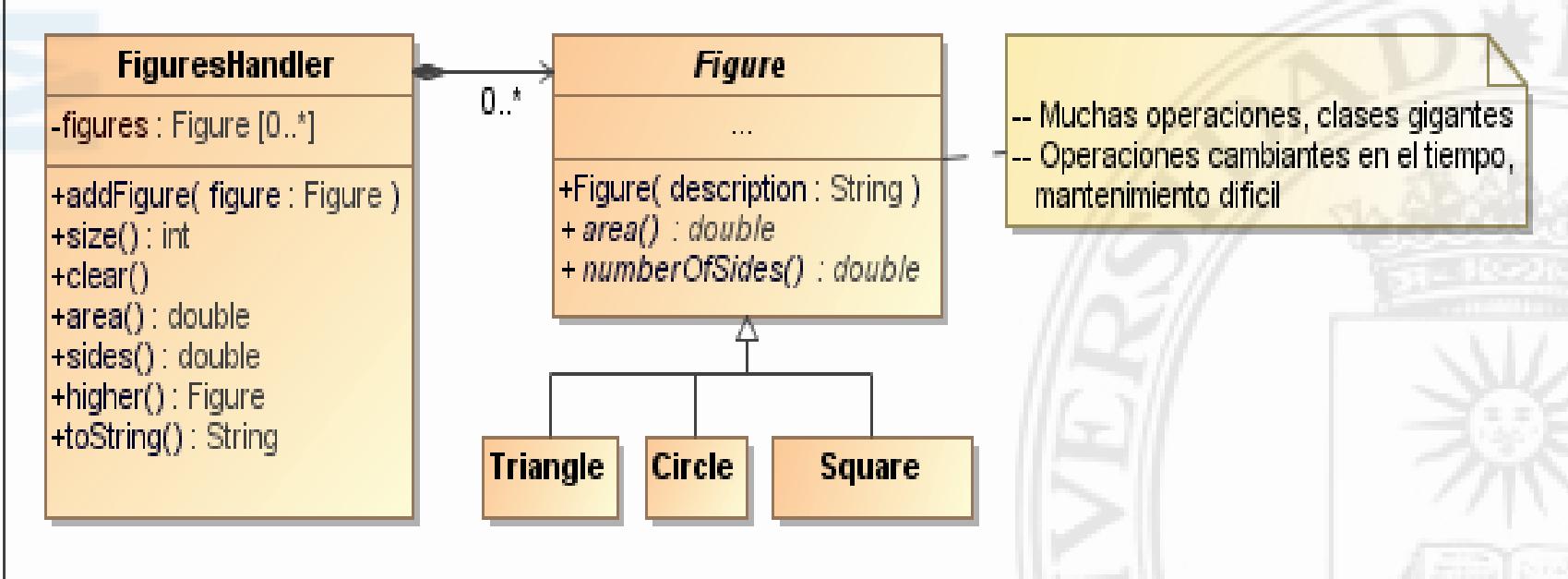
✍ Vehículo

- Se pretende realizar la gestión de vehículos. Se Parte de un vehículo con modelo y precio
- A los diferentes vehículos se les puede añadir extras: GPS, MP3, EDS... Cada extra tiene un precio y una descripción
- Finalmente, al objeto se le puede consultar su descripción (debe informar de los extras incorporados) y el precio final
 - public String descripcionFinal();
 - public int precioFinal();

Visitor (Visitante)

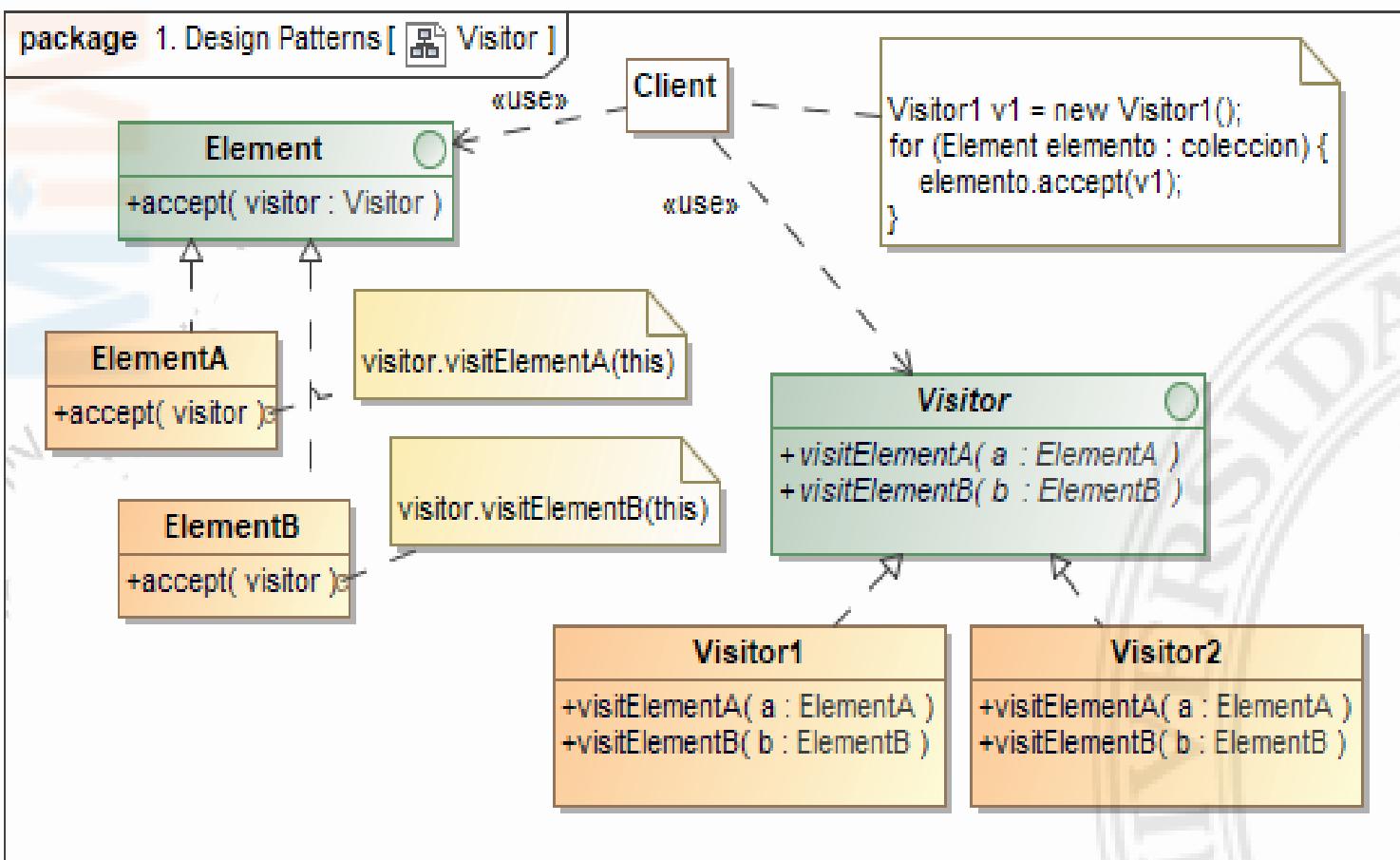
- Motivación
 - Dada un compilador con programas representados con arboles sintácticos. Se necesitan realizar múltiples operaciones sobre los datos y se pretende separar las operaciones de los datos
- Fuentes: paquete visitor

```
package miw.pd [  FiguresHandler.lackOfVisitor ]
```



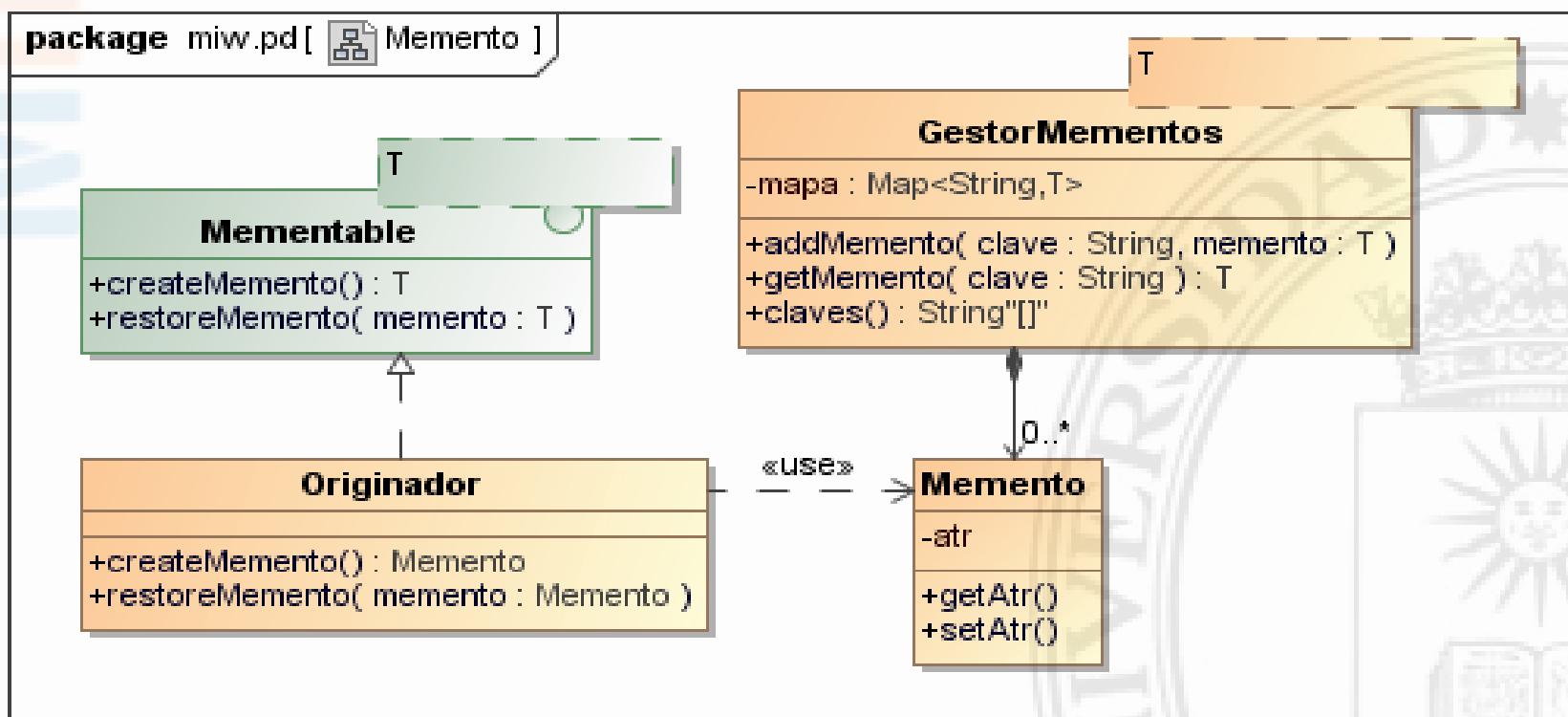
Visitor. Implementación

- Propósito
 - Definir un conjunto de operaciones sobre una estructura de datos de forma independiente



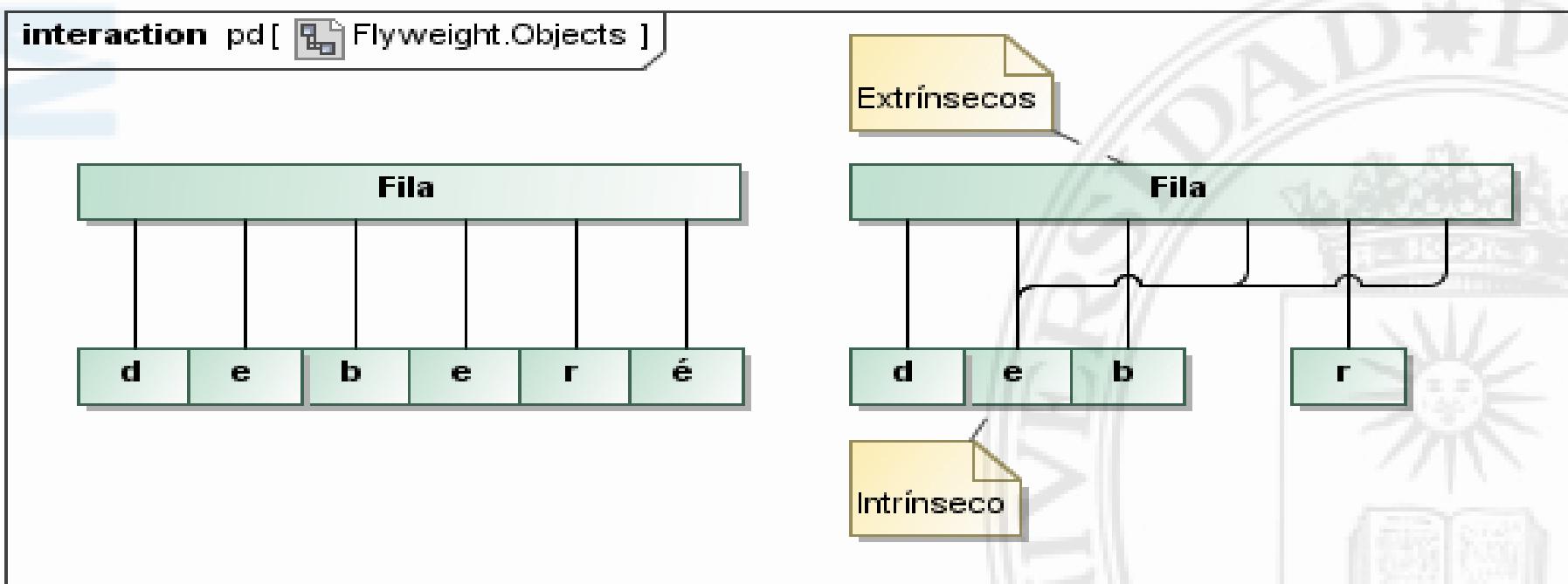
Memento (Recuerdo)

- También conocido *Token*
- Motivación
 - Cuando se requiere volver a situaciones anteriores
- Propósito
 - Externaliza el estado interno de un objeto sin violar la encapsulación



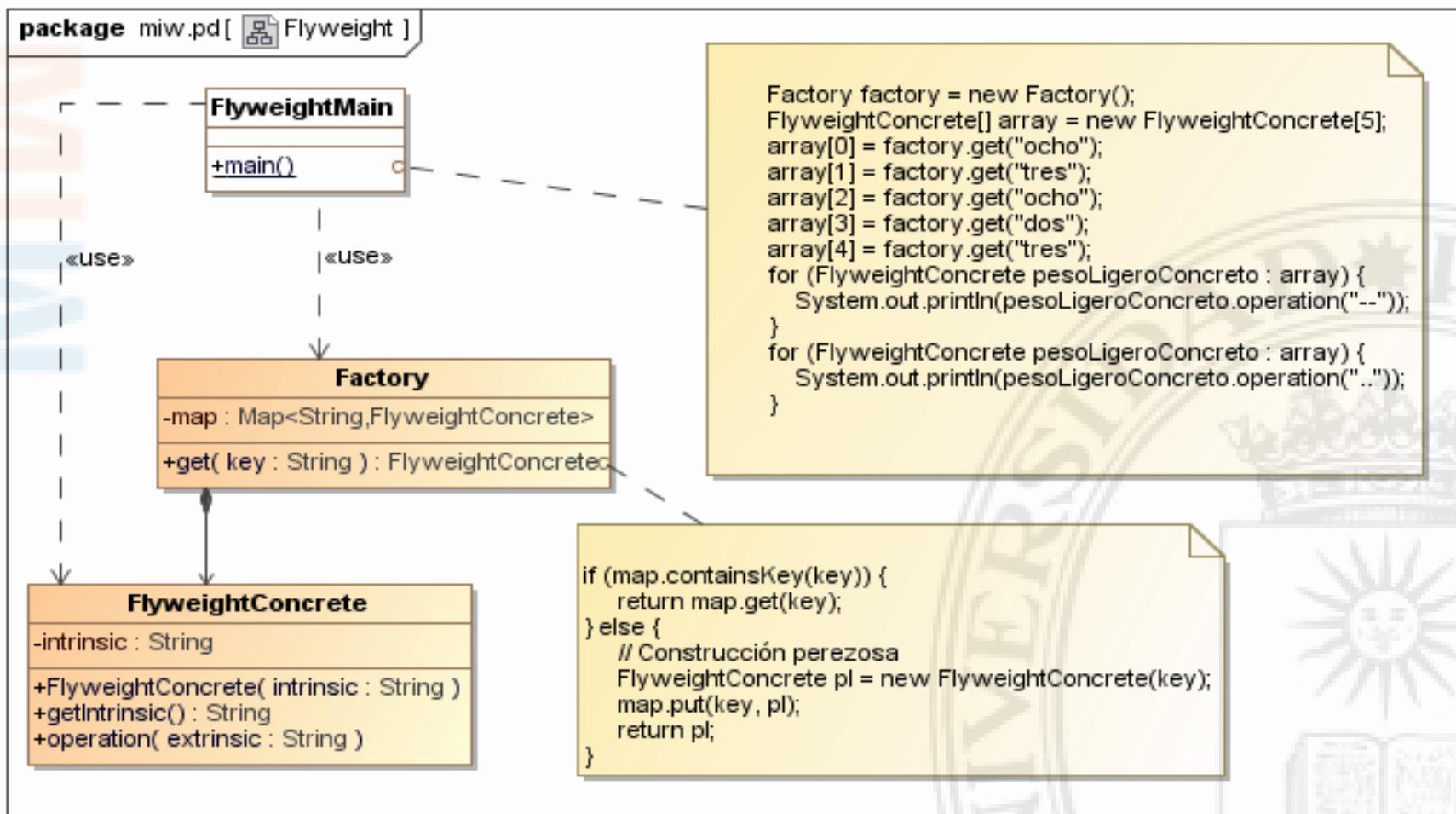
Flyweight (Peso ligero)

- Motivación
 - Eliminar o reducir la redundancia cuando tenemos gran cantidad de objetos que contienen información idéntica
 - En un editor de texto se podría plantear los caracteres como objetos, pero su número hace inviable esta solución. Una alternativa es plantear los caracteres como referencias de objetos carácter compartidos. Se distinguen los datos intrínsecos que son compartidos y se sitúan en el objeto compartido y los datos extrínsecos dependientes del contexto
- Fuentes: paquete flyweight



Flyweight. Implementación

- Propósito
 - Compartir objetos de grado fino de forma eficiente



✍ Strategy (Estrategia)

- También conocido
 - Policy (Política)
- Propósito
 - Define un conjunto de algoritmo haciéndolos intercambiables dinámicamente
- Motivación
 - Existen muchos algoritmos de ordenación, dependiendo su eficacia del número de elementos a ordenar
- Buscar por Internet un ejemplo para comprenderlo

Adapter (Adaptador)

- También conocido
 - Wrapper (Envoltorio)
- Motivación
 - Reduce la dependencia entre clases. A veces, se requiere reutilizar una clase en otra aplicación, pero poseen dominios diferentes
- Propósito
 - Convierte una interface de una clase en otra que es la que esperan los clientes

Proxy (Apoderado)

- También conocido: Surrogate (Sustituto)
- Motivación. Supongamos que el coste de un objeto es muy grande. Se pretende retrasar la creación del mismo con otro objeto que puede responder a ciertas peticiones más sencillas
- Propósito. Se proporciona un sustituto o representante para controlar el acceso a un objeto
- Aplicabilidad
 - Proxy virtual
 - Proxy remoto. Un representante local de otro remoto
 - Proxy de protección. Se controla el acceso al objeto original
 - Referencia inteligente. Se realizan operaciones adicionales

Bridge (Puente)

- También conocido
 - Handle, Body (Manejador, Cuerpo)
- Propósito
 - Desacopla una abstracción de su implementación, permitiendo modificaciones independientes de ambas
- Motivación
 - Cuando una estructura de abstracciones puede tener varias implementaciones, por ejemplo, plantillas diferentes de apariencia o para sistemas diferentes; la herencia no suele ser una buena solución

Chain of Responsibility (Cadena de responsabilidad)

- Propósito

- Se establecer una cadena de objetos receptores a través de los cuales se pasa una petición formulada por un objeto emisor. Cualquiera de los objetos receptores puede responder a la petición en función de un criterio establecido

- Motivación

- En un servicio de ayuda sensible al contexto



Prototype (Prototypo)

- Propósito
 - Crear los objetos a partir de prototipos mediante la clonación
- Motivación
 - Decidir dinámicamente el tipo de objeto a crear dependiendo del contexto



Protocolo HTTP (HTTP/2)

- HTTP (*Hyper Text Transfer Protocol*) es el protocolo para la capa de aplicación usado en cada transacción de la Web mediante petición-respuesta
 - Petición (*request*): envío por parte del cliente de una petición al servidor
 - URI (*Uniform Resource Identifier*)
 - `http://server:80/resource?params=one¶ms=two¶m=three`
 - Encabezado (*Header*)
 - Método: **GET, POST, PUT, PATCH, DELETE, OPTIONS, HEAD, TRACE...**
 - Parámetros
 - Cuerpo (*Body*)
 - Respuesta (*response*): envío por parte del servidor de una respuesta al cliente
 - Encabezado
 - Estado: 1xx: Respuestas informativas, 2xx: Peticiones correctas, 3xx: Redirecciones, 4xx Errores del cliente y 5xx Errores internos
 - Parámetros
 - Cuerpo
- HTTP es un protocolo sin estado, es decir, que se guarda ninguna información sobre conexiones anteriores

API REST

- También referenciado como RESTful
- End-point. Cada una de las posibles peticiones
 - GET /resurso1, POST /recurso1...
- Relación de Cliente-Servidor
- Servidor sin estado. El servidor no gestiona los recursos de los clientes. En cada petición el cliente debe enviar toda la información
- Se permite la elección del tipo de representación (JSON, HTML, XML,...) a través de los tipos MIME. El cliente decide el tipo MIME
- Se permite la caché. En las respuestas se indica si es cacheable, en tal caso, el cliente lo almacena para no volver a preguntarlo
- El servicio se organiza como un conjunto de recursos con cuatro operaciones (CRUD): Create (post), Read (get), Update (put/patch) y Delete (delete); pero ofrece un servicio por encima de la capa DAO
 - Seguro. No cambia su estado ni provoca efectos secundarios
 - Idempotente. Operación que repetida siempre da el mismo resultado

Buenas prácticas

- Usar SSL siempre.
- Documentar API.
 - OpenAPI (*Swagger*).
- Recursos sencillos y autodescriptivos.
- Para versiones utilizar v* en el nivel mas alto.
- Las Uris deben ser estables en el tiempo.
- Se organiza en recursos (sustantivos) y se referencia por URIs. Mejor utilizar plurales para los recursos frente a singulares.
 - <https://api.server.com/v0/orders>
- Las Uris **NO** referencian acciones (post, get, put, patch, delete).
 - <https://api.server.com/v0/get-orders>
- Las Uris **NO** referencian formatos de representación.
 - <https://api.server.com/v0/orders/pdf>
- Uris: spinal-case. Cuerpo: lowerCamelCase.
 - <https://api.server.com/v0/recent-orders>

Buenas prácticas

- “/” partes jerárquicas. “,” y “;” para partes no jerárquicas, matrices de valores.
- Identificación (id), procurar que no sea deducible, código hash.
 - <https://api.server.com/v0/orders/vkg23ds5>
 - <https://api.server.com/v0/orders/vkg23ds5/name>
- Controlar la granularidad. Datos que van juntos suministrarlos en una sola llamada. Evitar respuestas demasiadas voluminosas.
- Búsquedas: GET
 - `GET /orders/fixed-search`
 - `GET /orders/search?q=name:jes`
 - `GET /search/repo?q=tetris+language:java&sort=stars`
 - Respuestas parciales: ... `?fields=id,description`
 - Paginación: ... `?page=1&size=30`
 - Filtros: ... `?type=3,5`
 - Orden: ... `?sort=name,surname&desc=id`
- Atributo boolean: **post:true, delete:false**

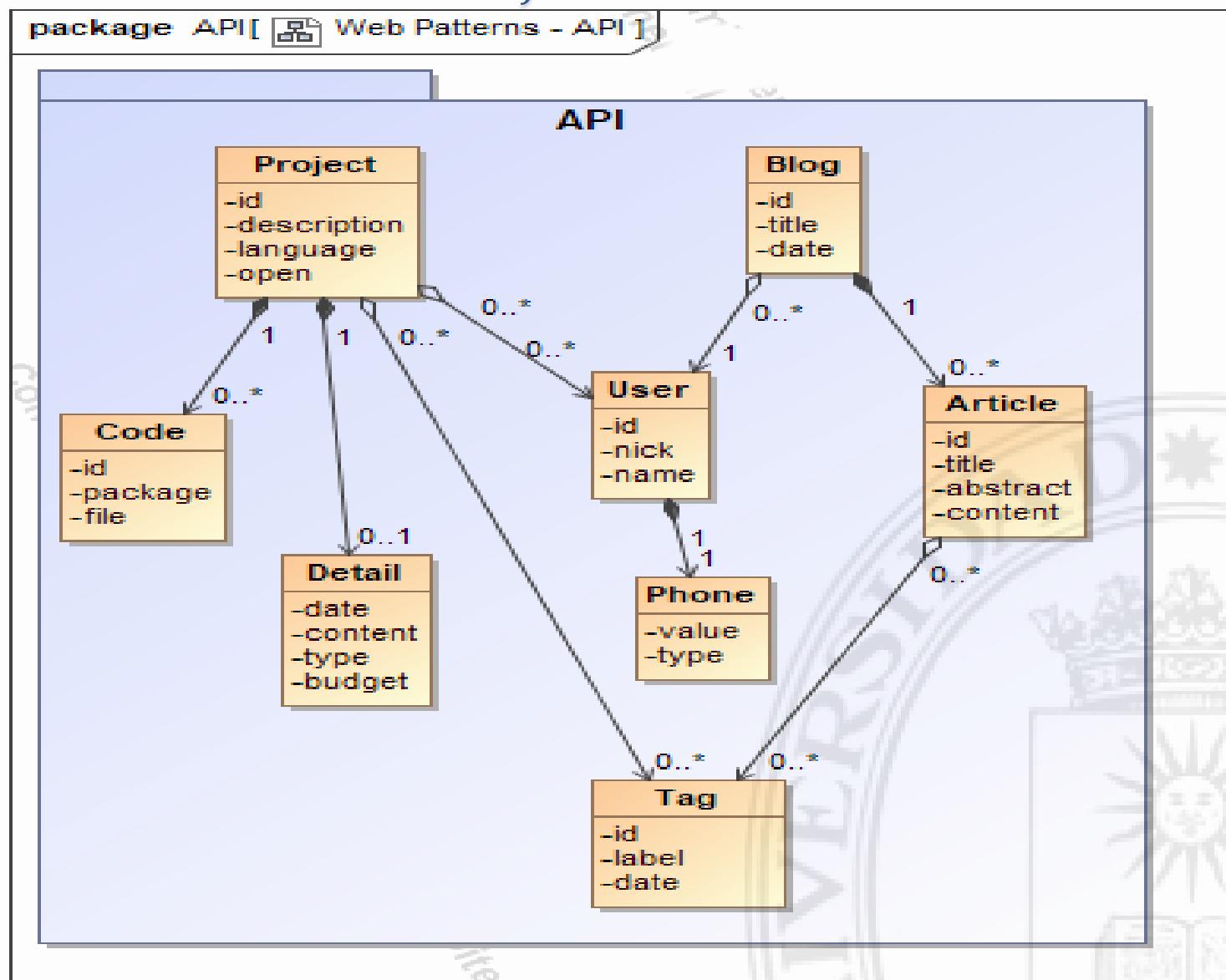
Buenas prácticas

- Utilizar mensajes de error: mensaje corto, mensaje largo y url del api con descripción completa
- Utilizar el dominio cruzado: **CORS**
- Seguridad: Header + SSL
 - nombre:clave
 - token (JWT)
- OAuth 2 debería ser usado para facilitar la transferencia del token seguro a terceros
- OpenID
- *Pretty print* por defecto y asegura que gzip sea soportado

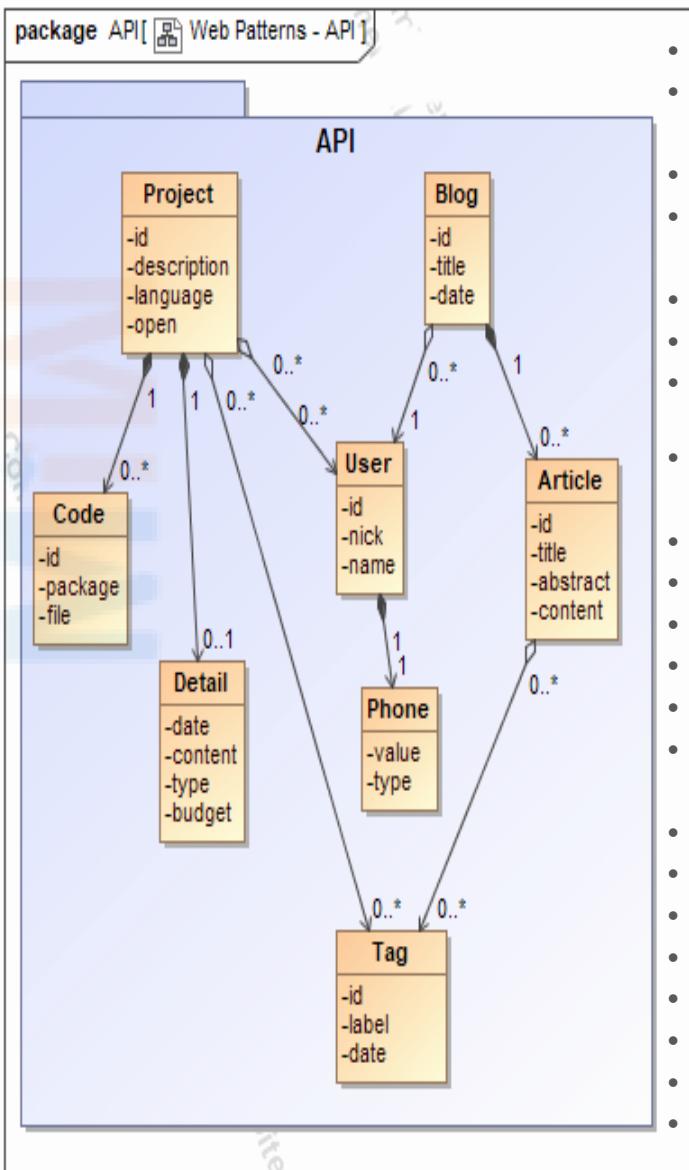
Operaciones: CRUD

- Métodos de HTTP:
 - **Create:** `post`. Para crear un nuevo recurso
 - Operación no segura y no idempotente
 - Respuesta: 201 (OK) o el tipo de error
 - Recomendable devolver una referencia del nuevo recurso
 - Recomendable devolver el recurso
 - **Read:** `get`. Para obtener un recurso o un conjunto
 - Operación segura e idempotente
 - Respuesta: 200 (OK) o el tipo de error
 - Devolver el recurso o un conjunto
 - **Update:** `put`. Para actualizar un recurso
 - Operación no segura e idempotente
 - Respuesta 200 (OK) o el tipo de error
 - Recomendable devolver el recurso
 - **Update:** `patch`. Para actualizar *parte* de un recurso o varios recursos
 - Operación no segura e idempotente
 - Respuesta 200 (OK) o el tipo de error
 - Recomendable devolver el recurso
 - **Delete:** `delete`. Para eliminar un recurso
 - Operación no segura e idempotente
 - Respuesta 204 (OK) sin contenido o el tipo de error

Ejercicio



Ejercicio



<http://server.com/api/v0/>
<http://api.server.com/v0/>

- GET /users response: [{id, nick}]
- POST /users request: {nick, name...}

- GET /users/{id} response:{id, nick, name, phone{value, type}}
- PUT /users/{id} request:{...}
- PATCH /users/{id} request:{nick:new-nick, phone/value:new-value}
- Son un conjunto de operaciones atómicas
- DELETE /users/{id}

- GET /blogs response:{id,title}
- DELETE /blogs/{id}
- GET /blogs/{id}/user
- POST /blogs request: {...}
- GET /blogs/{id}/articles/{id}/abstract response:{abstract}
- GET /tags response:{...}

- GET /projects/search?q=language:java&sort=id
- GET /projects/{id}/tags?sort=-date,label
- GET /projects?fields=language,description
- GET /users?page=2&size=20
- GET /search?q=language:java+language:typescript&sort=id
- GET /projects/opened
- GET /projects/search?q=tag:id
- GET/projects/{id}/open

Data Transfer Object (DTO)

- Motivación
 - En la capa de presentación se necesita obtener diferentes datos de diferentes partes de la capa de negocio. La aplicación se puede ver degradada por el uso intensivo de red
- Propósito
 - Permitir el intercambio de datos eficiente entre la capa de presentación y la capa de negocio
- Ventajas
 - Al utilizar un *DTO* para encapsular los datos de negocio, se realiza una única llamada a un método para enviar y recuperar el *DTO*, optimizando los recursos de red y de servidor
- Api Rest: los objetos devueltos serán *DTOs*, normalmente en JSON

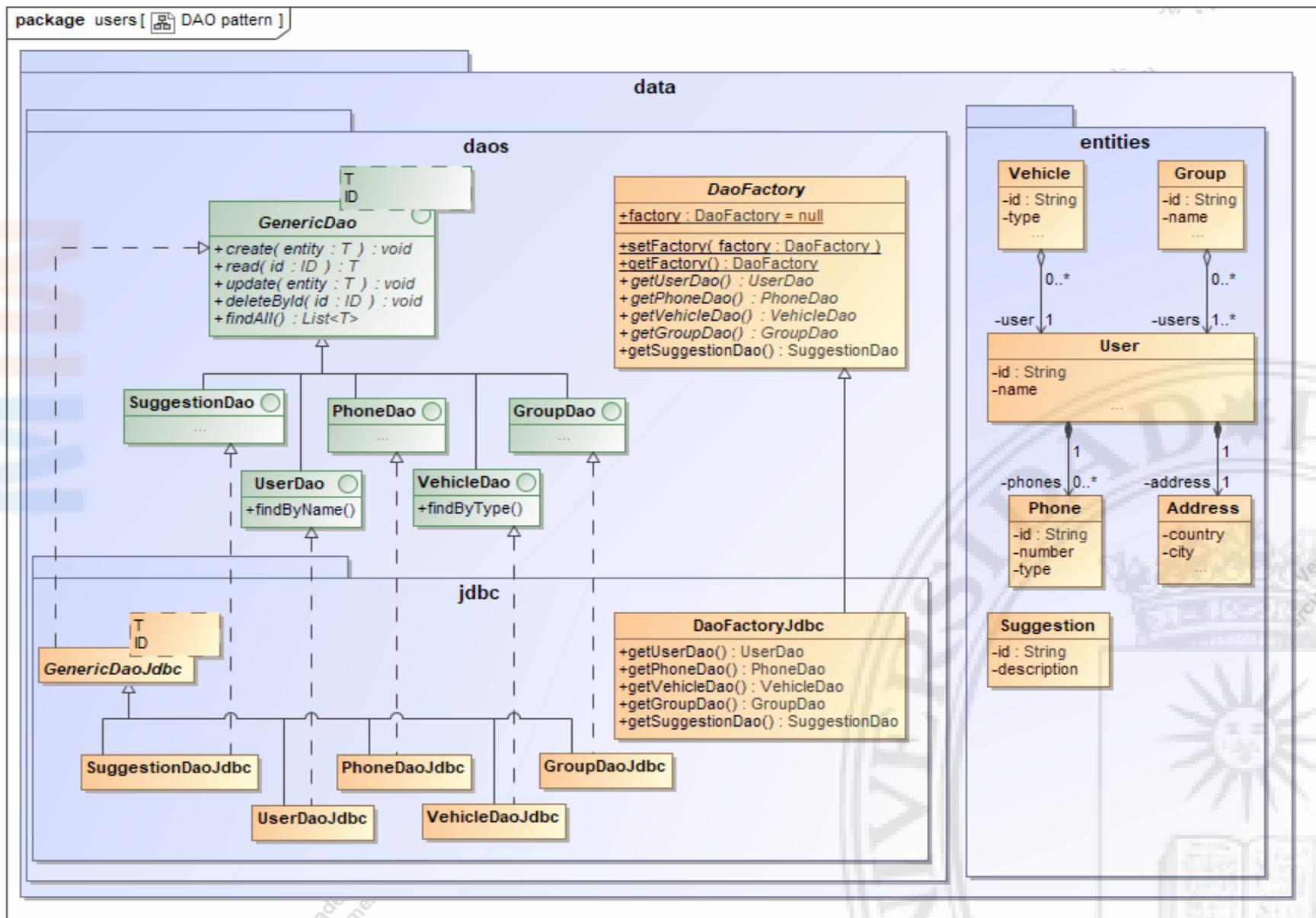
DAO (Data Access Object)

- La persistencia consiste en guardar los datos de forma permanente y de una manera ordenada, normalmente en Bases de Datos (relacionales, NoSQL...), ficheros XML/JSON...
- Data Transfer Object (DTO), normalmente llamado *entidad* (*Entity-SQL*), *documento* (*Document-MongoDB*). Son los objetos en donde se guardan los datos que vienen o van a la capa de persistencia
- DAOs (*Repositories*). Son las clases que se encargan de convertir los datos almacenados de forma persistente en BD en *objetos* o viceversa. Estas clases están conectadas con las BD
- Propósito
 - Abstraer y encapsular el acceso a la capa de persistencia, permitiendo desacoplar la capa de negocio con la capa de persistencia
 - Adapta el modelo de persistencia con el modelo de objetos
 - Permite el intercambio de implementaciones de persistencia sin afectar a la capa de negocio

Diseño: DAO

- Los DAOs transforman los datos del modelo de persistencia al modelo de Objetos
- En general, existirá un DAO por cada *Entidad-Documento*, componiendo un conjunto de DAOs relacionados
- Existirá un tipo de DAO por cada modelo de persistencia, y para que la *capa de negocio* esté desacoplada del modelo de persistencia, se aplicará el patrón *Abstract Factory*
- Para que exista una sola factoría, se aplicara el patrón *Singleton*

DAO

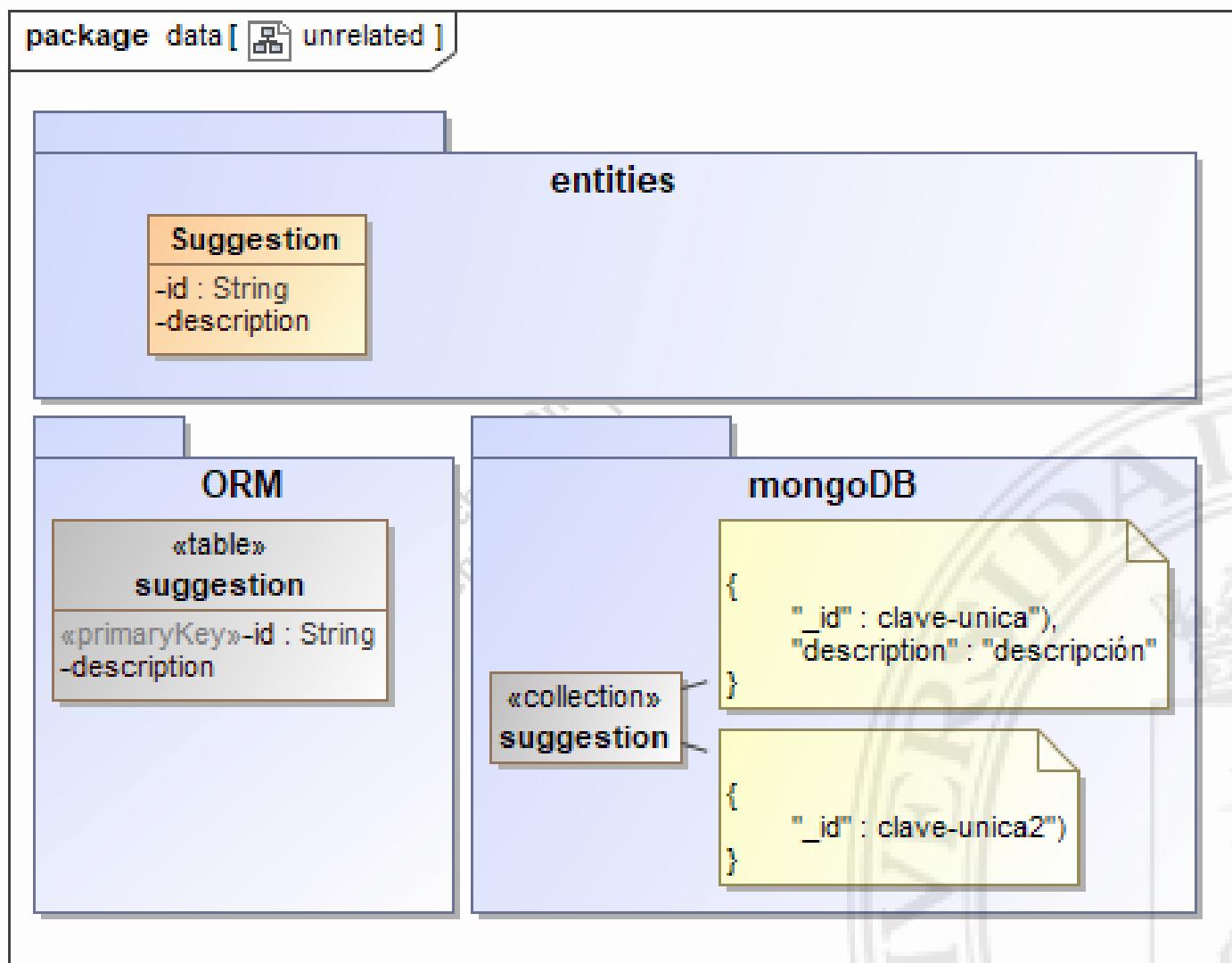


Mapeo Objeto-Relacional: ORM

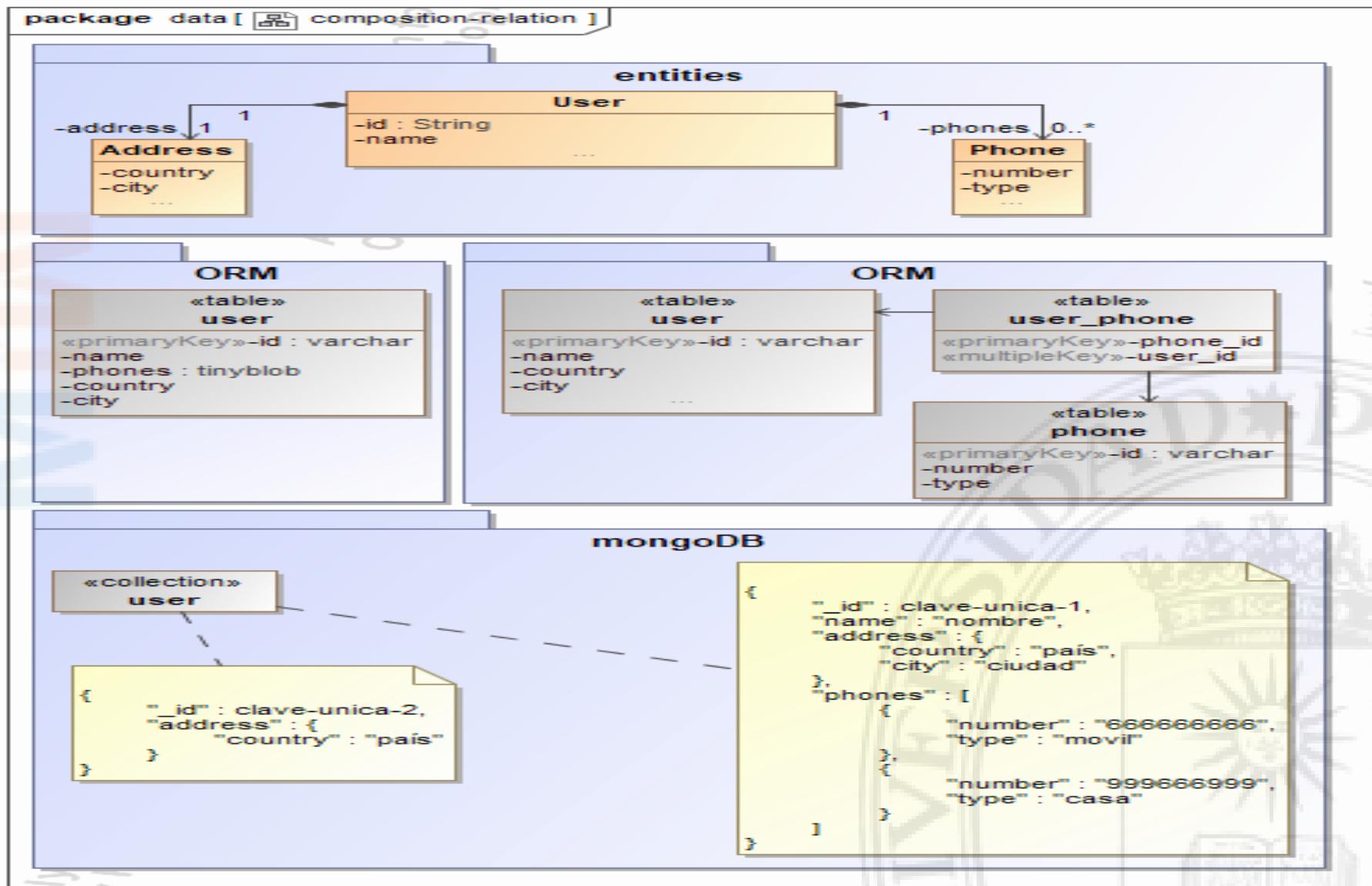
- El mapeo objeto-relacional (Object-Relational Mapping ORM) es una técnica para convertir datos de un lenguaje de programación orientado a objetos a una base de datos relacional
- Existen varias alternativas para el mapeo
- Eficiencia dependiendo de cada caso
 - Ocupación de memoria
 - Velocidad de acceso



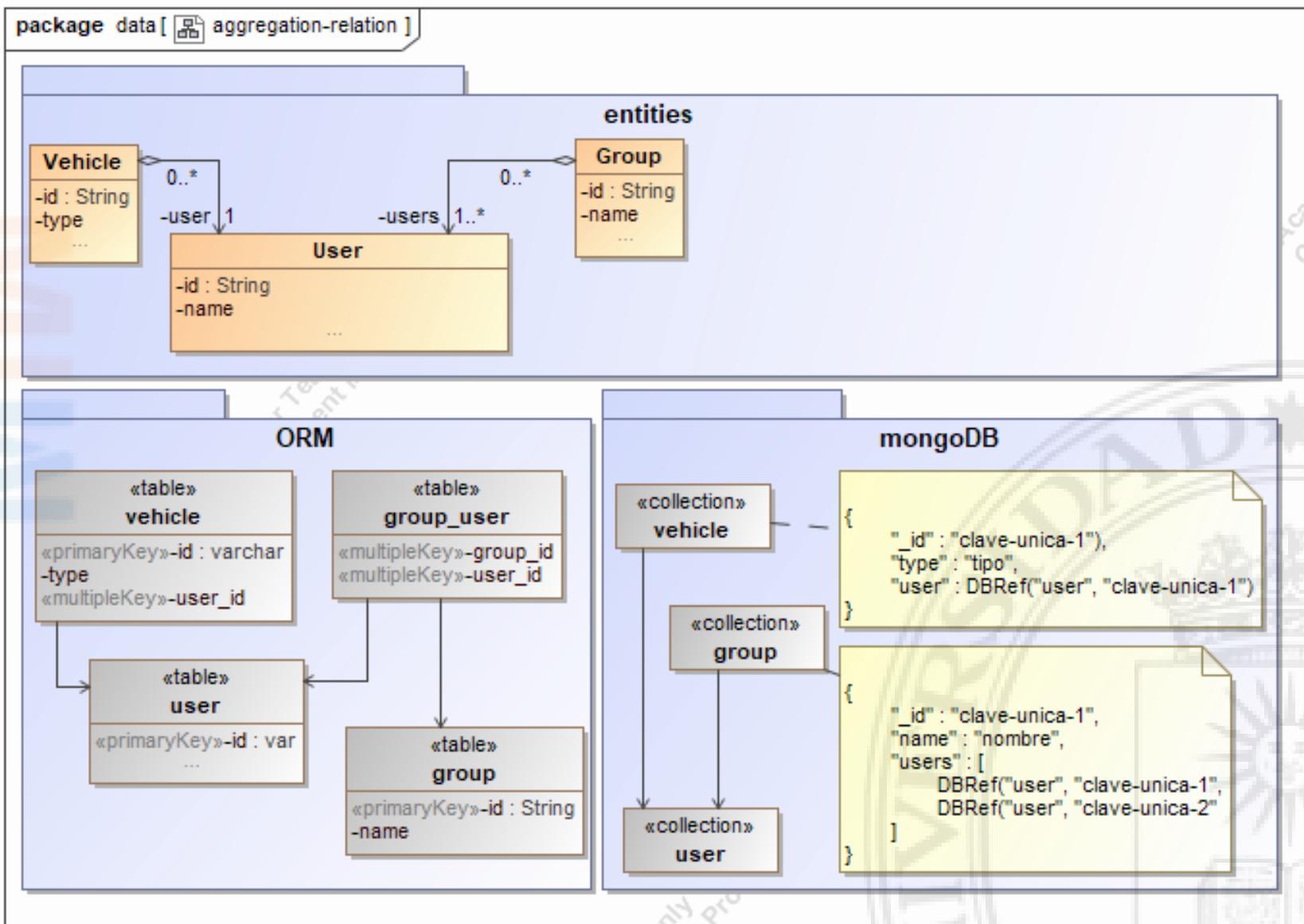
Sin relación



Relación de composición

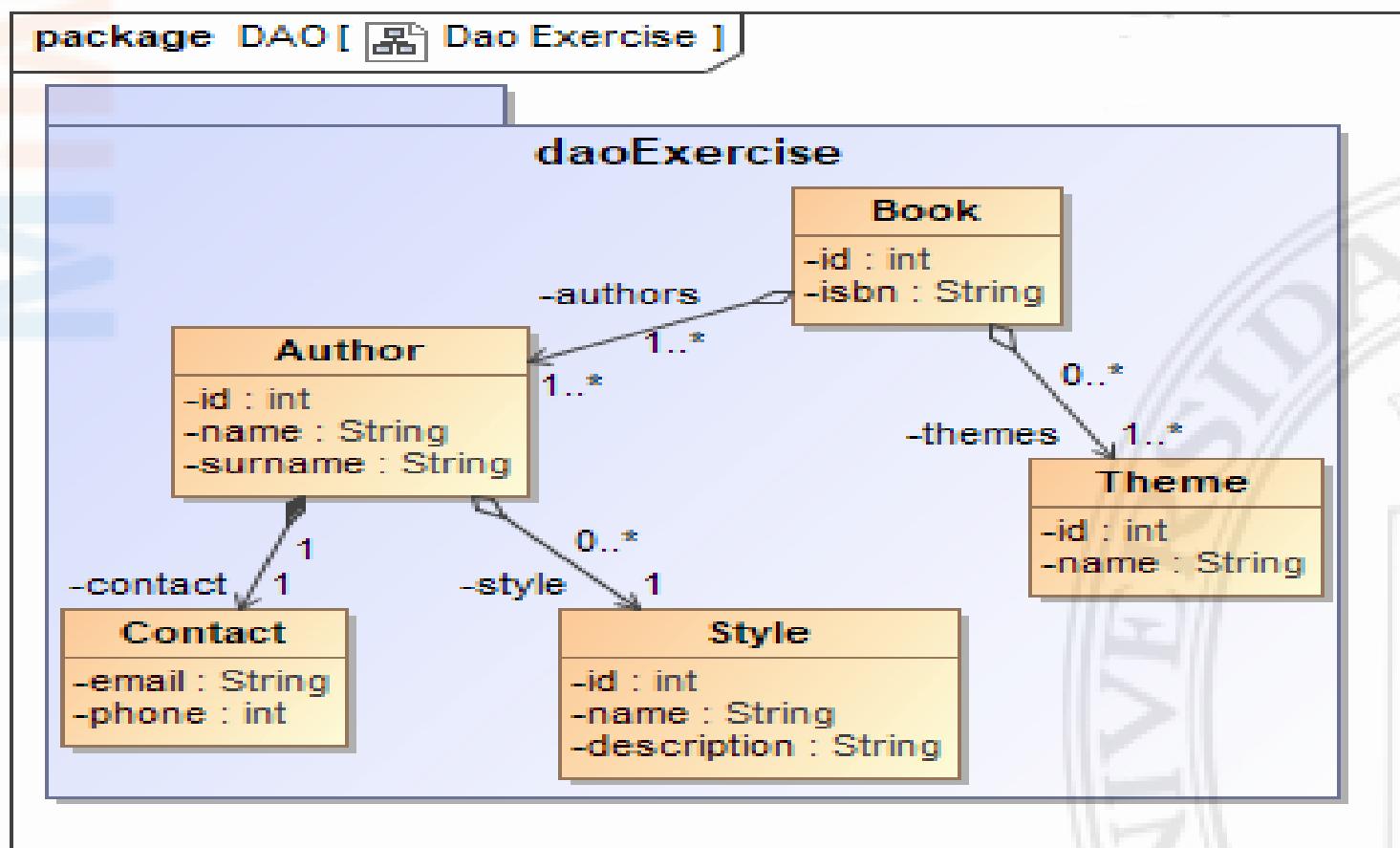


Relación de agregación & asociación

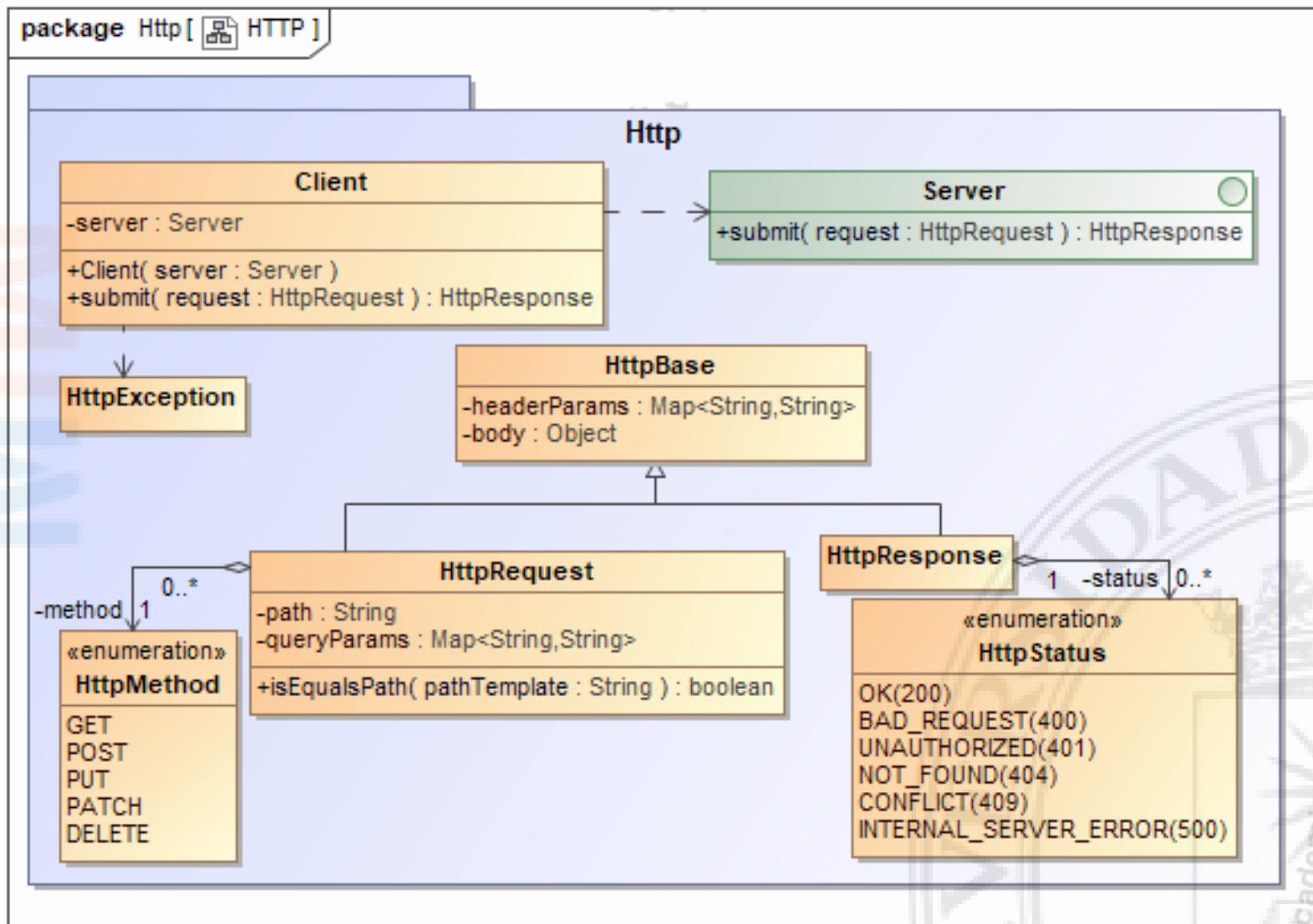


✍ Ejercicio

- Implementar el paquete *entities*
- Implementar el paquete *daos*
- Buscar diseños alternativos



Protocolo HTTP



Intercepting Filter

Motivación

- En una gestión de peticiones, cada tipo de petición conlleva un tipo de procesamiento propio, resulta necesario identificar y estandarizar dicho proceso
- Una petición web puede recorrer varias comprobaciones previas a su procesamiento (autentificación, validación de permisos, navegador, registrar...). En estas comprobaciones, se evalúa si se continua o no la petición
- Se necesita una forma flexible y simple para añadir y eliminar componentes de procesamiento

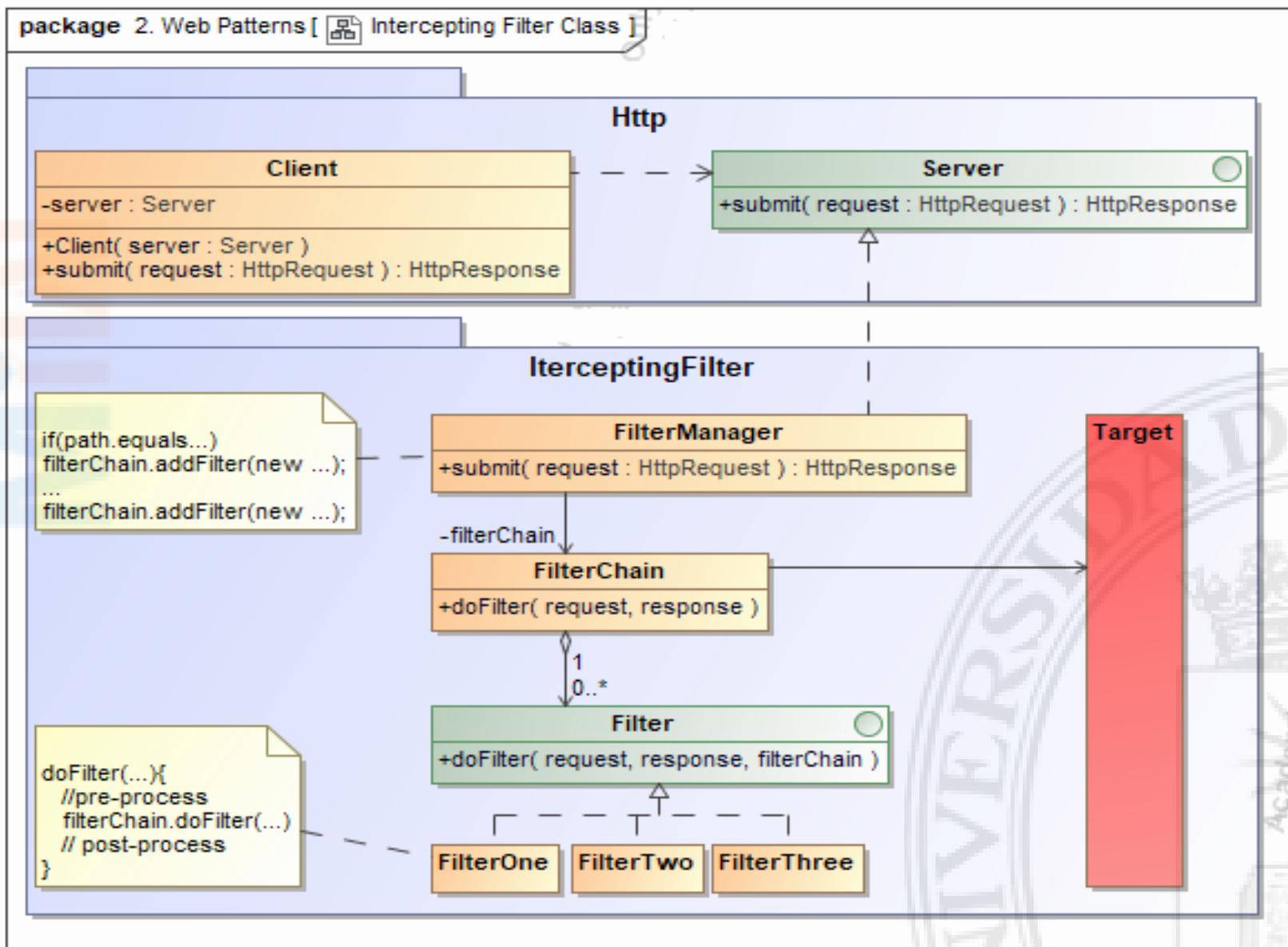
Propósito

- Crear un conjunto de filtros conectables para procesar servicios comunes de una forma estándar sin requerir cambios en el código principal del procesamiento de la petición

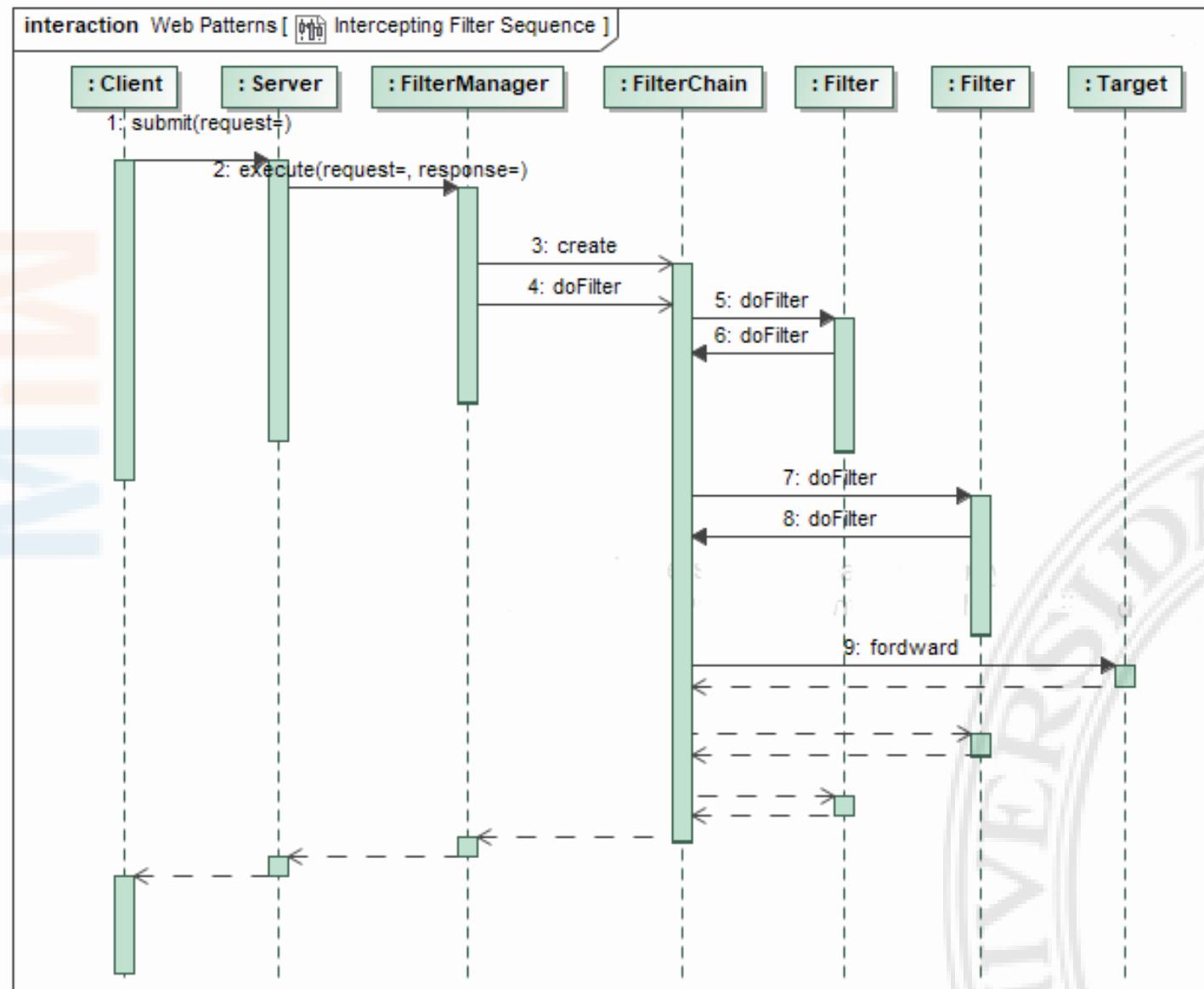
Ventajas

- Podemos añadir y eliminar estos filtros, sin necesitar cambios en el código existente
- Podemos, decorar nuestro procesamiento principal con una variedad de servicios comunes, como la seguridad, el logging, el depurado, etc.
- Estos filtros son componentes independientes del código de la aplicación principal, y pueden añadirse o eliminarse de forma declarativa

Intercepting Filter



Intercepting Filter



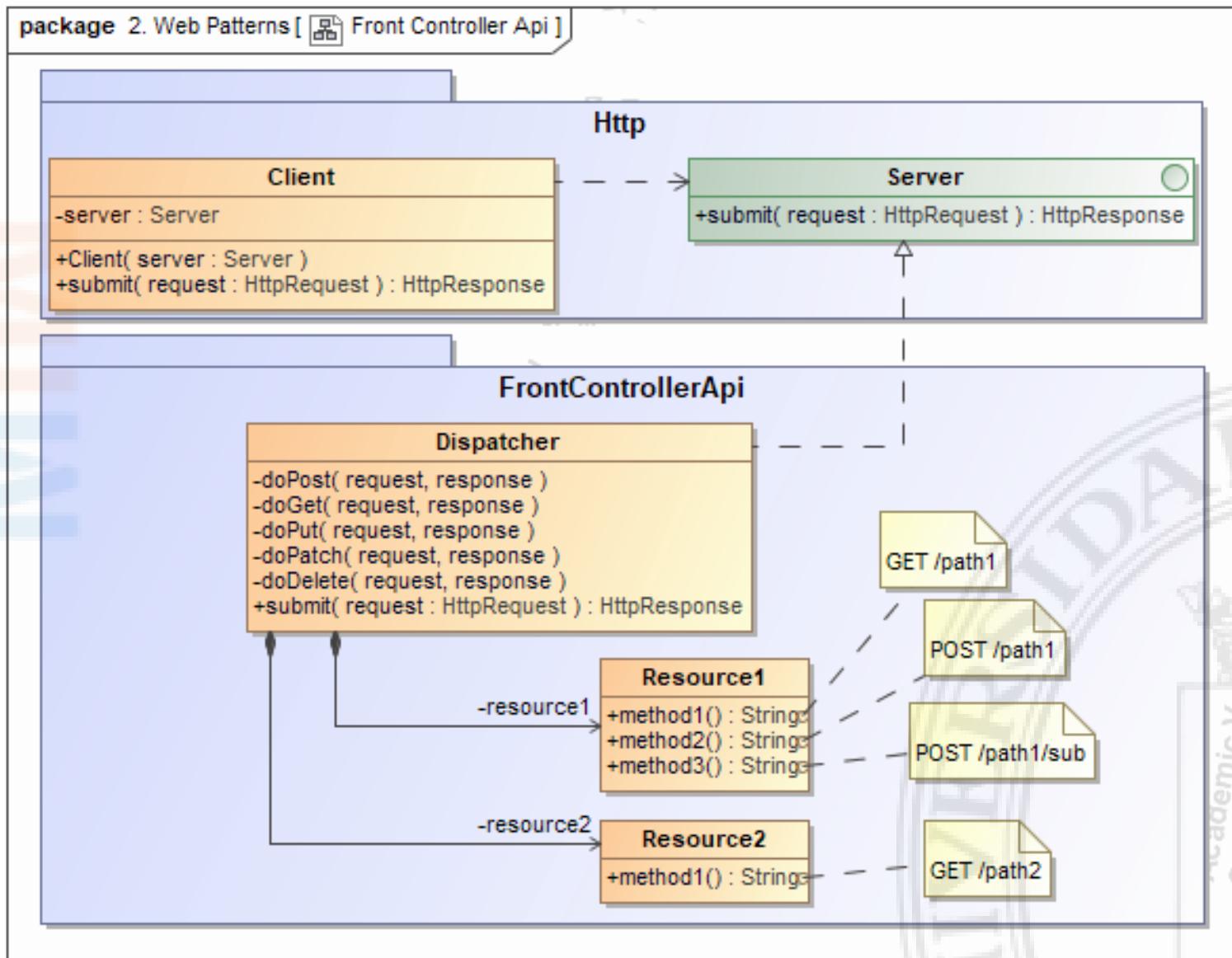
Intercepting Filter

```
■ [INFO ] 27/sep 23:20:20 http.HttpClientService      --> GET /public/debug?param=value
■ [INFO ] 27/sep 23:20:20 ingfilter.AuthenticationFilter --> Authenticating pre-process...
■ [INFO ] 27/sep 23:20:20 interceptingfilter.TimeFilter   --> Time pre-process: Wed Sep 27 23:20:20
■ [INFO ] 27/sep 23:20:20 interceptingfilter.DebugFilter  --> Debuging pre-process...
■ [INFO ] 27/sep 23:20:20 interceptingfilter.Target       --> -----> Executing TARGET.GET
/public/debug?param=value
■ [INFO ] 27/sep 23:20:20 interceptingfilter.DebugFilter  --> Debuging post-process...
■ [INFO ] 27/sep 23:20:20 interceptingfilter.TimeFilter   --> Time post-process: 16ms
■ [INFO ] 27/sep 23:20:20 ingfilter.AuthenticationFilter --> Authenticating post-process...
■ [INFO ] 27/sep 23:20:20 http.HttpClientService      --> HTTP/1.1 200 OK
headerParams={debug=DebugFilter, time=16ms}
■ [INFO ] 27/sep 23:20:20 http.HttpClientService      --> -----
■ [INFO ] 27/sep 23:20:20 http.HttpClientService      --> GET /noPublic?param=value
■ [INFO ] 27/sep 23:20:20 ingfilter.AuthenticationFilter --> Authenticating pre-process...
■ [INFO ] 27/sep 23:20:20 ingfilter.AuthenticationFilter --> Authenticating post-process...
■ [INFO ] 27/sep 23:20:20 http.HttpClientService      --> HTTP/1.1 401 UNAUTHORIZED
headerParams={auth=AuthenticationFilter}
■ [INFO ] 27/sep 23:20:20 http.HttpClientService      --> -----
■ [INFO ] 27/sep 23:20:20 http.HttpClientService      --> GET /public?param=value
■ [INFO ] 27/sep 23:20:20 ingfilter.AuthenticationFilter --> Authenticating pre-process...
■ [INFO ] 27/sep 23:20:20 interceptingfilter.TimeFilter   --> Time pre-process: Wed Sep 27 23:20:20
■ [INFO ] 27/sep 23:20:20 interceptingfilter.Target       --> -----> Executing TARGET.GET
/public?param=value
■ [INFO ] 27/sep 23:20:20 interceptingfilter.TimeFilter   --> Time post-process: 0ms
■ [INFO ] 27/sep 23:20:20 ingfilter.AuthenticationFilter --> Authenticating post-process...
■ [INFO ] 27/sep 23:20:20 http.HttpClientService      --> HTTP/1.1 200 OK
headerParams={time=0ms}
■ [INFO ] 27/sep 23:20:20 http.HttpClientService      --> -----> ooo-----
```

Front Controller

- Motivación
 - En una Aplicación Web se ofrece un gran variedad de peticiones en donde existen necesidades comunes de proceso, esto nos lleva a código duplicado y aumentar la dificultad de mantener robusta la aplicación
- Propósito
 - Centralizar el acceso de las peticiones web provenientes del cliente, mediante un controlador único
- Ventajas
 - El controlador maneja el control de peticiones, incluyendo la invocación de los servicios de seguridad como la autentificación y autorización, la delegación del procesamiento de negocio, el control de la elección de una vista apropiada, el manejo de errores... Permitiendo reutilizar el código entre peticiones diferentes y aumentando la mantenibilidad y reusabilidad del mismo

Front Controller API



Front Controller API

- [INFO] 27/sep 23:18:48 http.HttpClientService --> GET /path1?param=value
- [INFO] 27/sep 23:18:48 http.HttpClientService -->
HTTP/1.1 200 OK, body={"name":"Resource1:method1:value"}
- [INFO] 27/sep 23:18:48 http.HttpClientService --> -----ooo-----

- [INFO] 27/sep 23:18:48 http.HttpClientService --> POST /path1
- [INFO] 27/sep 23:18:48 http.HttpClientService -->
HTTP/1.1 200 OK, body={"name":"Resource1:method2"}
- [INFO] 27/sep 23:18:48 http.HttpClientService --> -----ooo-----

- [INFO] 27/sep 23:18:48 http.HttpClientService --> POST /path1/sub
- [INFO] 27/sep 23:18:48 http.HttpClientService -->
HTTP/1.1 200 OK, body={"name":"Resource1:method3"}
- [INFO] 27/sep 23:18:48 http.HttpClientService --> -----ooo-----

- [INFO] 27/sep 23:18:48 http.HttpClientService --> GET /path2?param=value
- [INFO] 27/sep 23:18:48 http.HttpClientService -->
HTTP/1.1 200 OK, body={"name":"Resource2:method1"}
- [INFO] 27/sep 23:18:48 http.HttpClientService --> -----ooo-----

- [INFO] 27/sep 23:18:48 http.HttpClientService --> GET /no?param=value
- [INFO] 27/sep 23:18:48 http.HttpClientService -->
HTTP/1.1 400 BAD_REQUEST, body={"error":"Path Error"}
- [INFO] 27/sep 23:18:48 http.HttpClientService --> -----ooo-----

Framework API

