Pedro Giménez

Tests DSS (2019)

Tests de Junio/Julio del 2014 al 2018

- **Correctas**
- **Dudosas**
- **Comprobadas**

¿Qué mide el rendimiento (performance) de un sistema?

a) El tiempo de respuesta.

b) El tiempo que el sistema está funcionando.

c) El tiempo que el sistema funciona sin fallos.

¿Cuándo es recomendable usar el patrón Data Mapper?

a) Cuando el modelo de dominio es simple.

b) Cuando el modelo de dominio es complejo.

c) Cuando el sistema usa una base de datos orientada a objetos.

¿Cuál de las siguientes afirmaciones sobre el patrón Remote Facade es FALSA?

a) Las fachadas remotas no deben contener lógica de dominio.

b) Las fachadas remotas no deben tener estado.

c) Las fachadas remotas deben ofrecer un único método para acceder a todas las propiedades de un objeto

En el patrón Model-View-Controller, cuando hay una serie de operaciones comunes que se deben realizar para cada petición del usuario, es conveniente usar el patrón:

a) Page Controller.

b) Front Controller.

c) Application Controller.

5. ¿Cuál de los siguientes patrones no se aplica a la capa de dominio?

a) Transaction Script.

b) Table Module.

c) Table Data Gateway.

Al usar el patrón Transaction Script:

a) Cada procedimiento creado es independiente del resto de capas del sistema.

b) Surgen dos tipos de procedimientos: los que acceden a la base de datos y los que se comunican con el resto de capas del sistema.

c) Cada procedimiento representa una acción que el usuario puede ejecutar.

En una arquitectura de 3 capas, ¿cómo se relacionan las capas de dominio y presentación?

a) La capa de dominio nunca debe depender de la capa de presentación.

b) La capa de presentación nunca debe depender de la capa de dominio.

c) Nunca deben estar relacionadas.

Al usar el patrón Active Record, ¿dónde debe situarse el código para acceder a la base de datos?

a) En el controlador del sistema.

b) En clases especializadas para el acceso a datos.

c) En las clases (objetos) de dominio.

¿Cuál es la diferencia entre los patrones Table Module y Domain Model?

a) Table Module pertenece a la capa de acceso a datos, y Domain Model a la capa de lógica de dominio.

b) Normalmente, con Table Module hay un objeto por cada tabla, mientras que con Domain Model hay un objeto por cada fila de la tabla.

c) Las dos son ciertas.

¿En qué capa se deben situar las operaciones complejas?

a) En la capa de dominio.

b) En la capa de servicio.

c) En la capa de presentación.

¿Qué patrón podemos usar para mantener la consistencia cuando se modifican dos instancias de un mismo objeto desde distintas partes del sistema?

a) Unit of Work.

b) Identity Map.

c) Lazy Load.

¿Cuál de las siguientes responsabilidades NO corresponden a la capa de presentación?

a) Comunicarse con capas superiores para ofrecer funcionalidades complejas.

b) Gestionar la interacción del usuario.

c) Comunicarse con la capa de dominio para notificar los datos modificados por el usuario.

En una arquitectura en capas, ¿qué capas pueden verse afectadas por una nueva funcionalidad?

a) Solamente la capa de dominio.

b) Solamente la capa de presentación.

c) Puede haber varias capas afectadas.

El uso de interfaces de grano fino mejora el rendimiento de los sistemas distribuidos

a) Verdadero.

b) Falso.

c) Sólo si el diseño es orientado a objetos.

¿Con qué patrón de lógica de dominio se combina normalmente el patrón Row Data Gateway?

a) Table Data Gateway.

b) Domain Model.

c) Transaction Script.

Para que un diseño de clases sea más fácil de entender y usar, la cohesión debe ser:

a) Baja.

b) Alta.

c) La cohesión no influye.

¿Qué patrón está pensado para poder intercambiar distintos comportamientos en tiempo de ejecución?

a) Strategy.

b) Proxy.

c) Builder.

¿Cuál de los siguientes patrones NO es de comportamiento?

a) Observer.

b) Proxy.

c) Command.

El uso del patrón GRASP Creador implica:

a) Disminuye el acoplamiento entre clases.

b) Aumenta el acoplamiento entre clases.

c) No afecta al acoplamiento entre clases.

¿Qué inconveniente tiene el uso del patrón Composite?

a) El cliente no distingue entre clases simples y compuesta.

b) No permite añadir nuevas clases simples.

c) Resulta complicado imponer restricciones sobre la estructura de los objetos compuestos.

¿Qué inconveniente tiene el uso del patrón Abstract Factory?

a) Los productos de distintas familias comparten el mismo interfaz.

b) No es fácil añadir nuevos productos.

c) Para el cliente no es fácil seleccionar la familia de productos a crear.

¿Con qué patrón podemos reducir el acoplamiento entre dos partes de un sistema, cuando una parte usa un conjunto de clases de la otra?

a) Composite.

b) Facade.

c) Proxy.

¿Qué ventaja tiene el patrón Builder?

a) El director no necesita conocer los pasos del proceso de construcción.

b) Cada constructor tiene un interfaz distinto.

c) Permite crear distintos productos siguiendo el mismo proceso.

¿Cuál es la principal motivación de los patrones GRASP?

a) Crear modelos que no cambiarán a lo largo del proyecto.

b) Proteger al sistema frente a posibles variaciones.

c) Identificar las clases del modelo de dominio que se comunican con el resto del sistema.

En el patrón Command, ¿qué información necesitan los comandos para ejecutarse?

a) Qué clase ha instanciado la acción.

b) Qué clase ha invocado la acción.

c) Qué clase es la receptora de la acción.

¿Qué patrón sería más adecuado para llevar un recuento del número de veces que se llama a cada método de un objeto?

a) Strategy.

b) Observer.

c) Proxy.

En el patrón Observer, ¿qué rol es el encargado de llevar un registro de los objetos que deben ser notificados cuando hay un cambio?

a) Subject.

b) Observer.

c) Client.

Cuál de los siguientes tipos de asociación implica un menor grado de acoplamiento

a) Uso.

b) Herencia.

c) Implementación de interfaces.

El patrón GRASP indirección implica:

a) Relacionar dos clases mediante otra intermedia.

b) Favorecer las relaciones directas entre clases.

c) Asignar menos responsabilidades a las clases que hay en la frontera con otras capas.

Para disminuir la dependencia entre clases y favorecer la reutilización de código el acoplamiento deber ser

a) Lo más bajo posible.

b) Lo más alto posible.

c) El acoplamiento no influye.

Según el patrón GRASP Creador, la clase A debe ser la encargada de crear una instancia de la clase B si:

a) A contiene o agrega instancias de B.

b) B usa instancias de A.

c) Las dos son ciertas.

¿Qué patrón GOF permite crear estructuras jerárquicas de objetos, de forma que el cliente pueda manejar objetos simples o compuestos indistintamente?

a) Builder.

b) Strategy.

c) Composite.

¿Cuál es la diferencia entre los patrones de diseño y los frameworks?

a) Los Frameworks se basan en uno o varios patrones para solucionar problemas concretos.

b) Los patrones usan frameworks para ofrecer soluciones reutilizables.

c) Ninguna, los frameworks se usan para el diseño detallado de un sistema.

El patrón GOF Abstract Factory:

a) Provee un interfaz para crear familias de productos de forma consistente.

b) Permite controlar el número de instancias de los objetos que se crearán.

c) Disminuye notablemente el número de clases del sistema.

¿Cuál de las siguientes afirmaciones es CIERTA? Para que un diseño sea fácil de mantener y usar:

a) Debemos mantener un bajo acoplamiento y una baja cohesión.

b) Debemos mantener un alto acoplamiento y una baja cohesión.

c) Debemos mantener un bajo acoplamiento y una alta cohesión.

Los objetos Data Transfer Object:

a) Solamente pueden contener datos de un objeto de dominio para garantizar la consistencia.

b) Pueden contener datos de varios objetos de dominio para mejorar el rendimiento.

c) No deben contener datos de los objetos de dominio para aumentar la cohesión.

Cuando se desea simplificar el acceso a un subsistema o capa, ¿qué patrón GOF es el más indicado?

a) Strategy.

b) Proxy.

c) Façade.

¿A qué nos referimos normalmente cuando hablamos de la arquitectura de un sistema?

a) A la infraestructura de hardware que dará soporte al sistema.

b) A los principales componentes del sistema y sus relaciones.

c) A la estructura de bajo nivel de cada componente del sistema.

El acoplamiento entre clases es una medida de:

a) El grado de dependencia entre las clases del sistema.

b) Cuantas funcionalidades distintas se asigna a cada clase.

c) Cuantas clases intermedias hay que recorrer para llegar de una clase A a otra B.

Las clases de diseño que surgen como resultado del patrón Fabricación Pura suelen aparecer:

a) Para representar objetos de dominio.

b) Por descomposición funcional, para dividir responsabilidades.

c) Por la aparición de jerarquías de herencia.

¿Cuál es la principal ventaja de usar patrones GOF a la hora de diseñar un sistema?

a) Permite disminuir el número de clases del sistema.

b) Hace que el sistema sea más fácil de comprender, a cambio de disminuir las posibilidades de reutilización de código.

c) Aumenta la flexibilidad del sistema frente a futuros cambios.

¿Cuáles son las capas típicas de un sistema de 3 capas?

a) Acceso a datos, servicio y lógica de dominio.

b) Servicio, lógica de dominio y presentación.

c) Acceso a datos, lógica de dominio y presentación.

¿Qué problema pretende evitar el patrón Lazy Load?

a) Problemas de integridad referencial al cargar objetos relacionados.

b) Problemas de rendimiento al cargar objetos relacionados.

c) Problemas por el uso excesivo de herencia en la lógica de dominio.

¿Cuándo conviene aplicar el patrón GRASP "Experto en Información" en cascada?

a. Cuando queremos prevenir la aparición de nuevas dependencias entre clases.

b. Cuando queremos romper intencionadamente la encapsulación de información.

c. Cuando queremos traspasar datos de la lógica de negocio a la capa de acceso a datos.

¿Qué tipo de acoplamiento debemos evitar para facilitar los cambios de tecnología en la capa de presentación?

a) Que la capa de presentación tenga como dependencias clases de la capa de lógica de negocio.

b) Que la capa de lógica de negocio tenga como dependencias clases de la capa de presentación.

c) Que las clases de la capa de presentación tengan dependencias entre sí.

¿Qué ventaja NO podemos conseguir con el patrón GOF "Proxy"?

a) Controlar el acceso a un objeto.

b) Simular un objeto remoto de forma local.

c) Proporcionar una implementación alternativa para un objeto.

¿Quién debe ser el encargado de crear los objetos de transferencia de datos (DTO)?

a) El objeto de dominio que contiene los datos.

b) Un objeto proxy que ocupa el lugar del DTO.

c) Un objeto Assembler que tiene acceso a los objetos de dominio.

¿Qué tipo de interfaces son deseables para llamadas entre objetos distribuidos?

a) Interfaces sin estado.

b) Interfaces de grano fino.

c) Interfaces de grano grueso.

¿Qué patrón es más recomendable para añadir nuevas funcionalidades a un objeto?

a) Decorator

b) Composite

c) Adapter

Un lenguaje específico de dominio (DSL)…

a) Se construye a partir de un diagrama de clases UML.

b) Puede representar todo o una parte del dominio para el que está construido.

c) Es menos abstracto que un lenguaje de propósito general.

¿Qué patrón permite reutilizar un mismo objeto de acceso a datos para distintas vistas?

a) Model View Controller

b) Model View Presenter

c) Code-behind

¿En qué se basa el patrón GRASP "Polimorfismo"?

a) Las instancias de clases hijas se pueden comportar como si se tratase de la clase padre.

b) Las instancias de una clase padre se pueden comportar como si se tratase de una clase hija

c) La introducción de una jerarquía de herencia disminuye el acoplamiento.

Los objetos de transferencia de datos (DTO)…

a) Se usan para pasar información entre distintas capas.

b) Contienen los métodos CRUD que se comunican con la base de datos.

c) Pueden contener tanto datos como métodos de lógica de negocio.

En una arquitectura de tipo Microkernel…

a) La escalabilidad es alta debido al pequeño tamaño del núcleo.

b) Cada plugin debe tener una interfaz independiente de las demás.

c) Se pueden añadir nuevas funcionalidades en tiempo de ejecución.

En una arquitectura en capas abiertas…

a) Se permite que las capas inferiores se comuniquen con las superiores.

b) Las capas superiores pueden saltarse algunas de las capas inferiores.

c) Se define una capa de servicio que puede ser usada opcionalmente por la capa de lógica de negocio.

¿Con qué patrón podemos recibir notificaciones cuando cambia el estado de un objeto?

a) Adapter.

b) Observer.

c) Command.

En el patrón GOF "Factory Method", la clase Creator proporciona una funcionalidad genérica independientemente del tipo de producto que se quiera crear.

a) Verdadero.

b) Falso, esa responsabilidad corresponde a la clase ConcreteCreator.

c) Falso, esa responsabilidad corresponde a la clase ConcreteProduct.

¿En qué consiste el patrón GRASP "Indirección"?

a) Invertir el flujo de llamadas entre dos clases, facilitando así la automatización de pruebas.

b) Separar un conjunto grande de responsabilidades en dos clases distintas, favoreciendo así la cohesión.

c) Asignar una responsabilidad a una clase intermedia, desacoplando así dos clases del sistema.

En el patrón GOF "Builder", ¿qué clases debe conocer la secuencia de pasos necesaria para construir un producto?

a) La clase Builder.

b) La clase ConcreteBuilder.

c) La clase Director.

¿Qué beneficio obtenemos al usar patrones de software?

a) Disminuye la complejidad de los diseños.

b) Se reduce la cantidad de código que hay que crear.

c) Se simplifica la introducción de nuevas funcionalidades en el futuro.

¿Cuál de los siguientes tipos de acoplamiento es más fuerte?

a) Cuando hay una jerarquía de herencia.

b) Cuando una clase implementa un interfaz.

c) Cuando una clase recibe una lista de instancias de otra clase como parámetro en un método.

¿Con qué otro patrón GRASP está relacionado el patrón "Creador"?

a) Bajo acoplamiento, ya que disminuye el número de dependencias del sistema.

b) Alta cohesión, ya que aumenta la cohesión de la clase creadora.

c) Controlador, ya que la clase creadora puede controlar a la clase creada.

¿En qué se diferencian un modelo de dominio y un diagrama de diseño de clases?

a) El modelo de dominio se deriva a partir del diagrama de diseño de clases.

b) El diagrama de diseño de clases se deriva a partir del modelo del dominio.

c) El modelo de dominio asigna responsabilidades a las clases, mientras que el diagrama de diseño de clases únicamente identifica las relaciones entre clases.

¿En qué caso está más indicado aplicar el patrón GRASP "Creador"?

a) Cuando queremos limitar el número de instancias de la clase creada.

b) Cuando la creación de instancias sigue una lógica condicional.

c) Cuando queremos almacenar instancias del objeto creado.

¿Cuándo es necesario introducir una clase "Fabricación Pura"?

a) Cuando queremos una clase intermediaria para desacoplar dos clases ya existentes.

b) Cuando el aumento de responsabilidades de una clase pone en peligro su cohesión.

c) Cuando necesitamos aplicar el patrón "Experto en información" en cascada.

¿Qué patrón GOF permite implementar de forma sencilla la funcionalidad "deshacer"?

a) Proxy.

b) Command.

c) Strategy.

¿Cuándo es preferible usar una clase Factoría en lugar de aplicar el patrón GRASP Creador?

a) Cuando el tipo concreto del objeto a crear depende de un conjunto de condiciones.

b) Cuando se debe inicializar el objeto en el momento de su creación.

c) Cuando el objeto creador debe almacenar los objetos creados.

¿Qué problema se puede derivar de un excesivo acoplamiento en el diseño de un sistema?

a) Cada clase asume demasiadas responsabilidades.

b) Los cambios en una clase pueden afectar a un gran número de clases distintas.

c) Ninguno, si el acoplamiento no es entre clases de distintas capas.

¿Qué patrón de lógica de negocios permite agrupar cada funcionalidad del sistema en un único método?

a) Domain Model.

b) Table Data Gateway.

c) Transaction Script.

¿Cuál es el principal problema del patrón "Class Table Inheritance"?

a) Es difícil usar un campo identificador único para todas las subclases.

b) Las operaciones join necesarias pueden afectar al rendimiento.

c) Las columnas que no usan todas la subclases desperdician espacio en la base de datos.

¿Cuándo es necesario dividir una clase en dos o más clases distintas?

a) Cuando aplicamos el patrón "Experto en información" en cascada.

b) Cuando el acoplamiento de la clase es demasiado alto.

c) Cuando la cohesión de la clase es demasiado baja.

¿Con qué patrones GRASP está relacionado el patrón GOF Facade?

a) Controlador y bajo acoplamiento.

b) Controlador y experto en información.

c) Bajo acoplamiento y experto en información.

¿Qué información contiene un objeto Data Transfer Object?

a) Con qué Fachada Remota debe comunicarse el cliente para recuperar los datos.

b) Reglas para transformar un objeto del modelo de dominio a representación textual.

c) Datos de uno o más objetos del modelo de dominio.

¿Qué técnica de diseño de interfaces consiste en realizar un diseño separado de un sitio web para dispositivos móviles?

a) Adaptive web design.

b) Responsive web design.

c) Device-oriented web design.

¿En qué situación NO podemos reemplazar una jerarquía de herencia por una solución distinta en la composición?

a) Cuando la herencia se usa únicamente para heredar un comportamiento.

b) Cuando necesitamos usar el polimorfismo.

c) Cuando las clases hijas sobreescriben métodos de la clase padre.

En el patrón State, ¿qué clase es la encargada de decidir cuál es el siguiente estado cuando hay un cambio de estado?

a) Solamente la clase Context puede tener esa información.

b) Puede ser la clase Context o la clase ConcreteState.

c) Ninguna de las clases implicadas en el patrón deberían tomar esa decisión.

¿Qué técnica podemos usar para evitar acoplar una clase con la implementación concreta de una funcionalidad?

a) Inversión de dependencias.

b) Inyección de dependencias.

c) Inversión de control.

En una arquitectura en capas, ¿por qué la capa de lógica de negocio se sitúa por encima de la capa de acceso de datos?

a) No es necesario, al ofrecer normalmente las mismas funcionalidades se puede intercambiar su orden.

b) Para evitar que la capa de acceso a datos acceda directamente a la capa de servicios.

c) Para desacoplar a las capas superiores de los detalles de acceso a la base de datos.

¿Cuál de las siguientes NO es una ventaja de usar un framework arquitectural?

a) Podemos cambiar fácilmente la arquitectura del sistema.

b) Proporciona una infraestructura que podemos extender con comportamiento personalizado.

c) Establece las reglas mediante las que deben interactuar los componentes del sistema.

Según el principio de inversión de dependencias...

a) Los módulos de bajo nivel no deben depender nunca de abstracciones.

b) Los módulos de alto nivel no deben depender nunca de abstracciones.

c) Los módulos de alto y bajo nivel deben depender de abstracciones.

¿Qué objetos colaboran con el Front Controller para realizar comprobaciones de seguridad?

a) Middleware.

b) Router.

c) Application Controller.

¿Qué patrón de capa de presentación es más adecuado cuando una sola interfaz presenta información de muchos objetos distintos?

a) Model View Controller.

b) Model View Presenter.

c) Model View ViewModel.

Queremos desarrollar una aplicación de escritorio para gestionar colecciones de música, con la posibilidad de que otras personas puedan extender sus funcionalidades una vez publicada ¿Qué patrón arquitectural sería el más adecuado?

a) Arquitectura orientada a eventos.

b) Tuberías y filtros.

c) Microkernel.

En una aplicación web, ¿en qué objetos de la capa de presentación se deben hacer transformaciones sobre los datos de entrada que afecten a varias funcionalidades(rutas) distintas?

a) Middleware.

b) Router.

c) Application Controller.

Al implementar el patrón Composite, ¿en qué clase deben estar los métodos que permiten gestionar los componentes de los objetos compuestos para evitar un uso incorrecto de estos métodos?

a) <<Composite>>

b) <<Component>>

c) <<Leaf>>

¿Que patrón de lógica de negocios permite implementar funcionalidades que involucran a varias entidades distintas (tablas en la base de datos) en los métodos de una misma clase?

a) Table Module. **

b) Domain model. **

c) Transaction Script. **

¿Qué arquitectura divide los sistemas en pequeños componentes independientes para favorecer la escalabilidad?

a) Microservicios.

b) Arquitectura en capas.

c) Tuberías y filtros.

¿Qué patrón ORM sirve para evitar tener objetos duplicados en memoria al recuperar datos de la base de datos?

a) Unit of work.

b) Identity map.

c) Lazy load.

¿Cuál es el principal propósito de los sistemas de rejillas para diseño responsive?

a) Favorecer el uso de etiquetas semánticas..

b) Facilitar el posicionamiento y organización del contenido de las páginas.

c) Evitar el uso de *media queries* para el posicionamiento.

¿Cuándo una clase necesita una instancia de otra, ¿qué nombre recibe la técnica que consiste en pasarle esa instancia desde fuera en lugar de crearla dentro de la case?

a) Inversión de control.

b) Inversión de dependencias.

c) Inyección de dependencias. *(Porque dice que: Un objeto distinto se encarga de crear la instancia de una implementación concreta y proporcionársela al objeto que la usará.)*

En una arquitectura en capas, ¿con qué capa no debería comunicarse nunca la capa de presentación?

a) Acceso a datos.

b) Servicios.

c) Lógica de negocio.

¿Qué patron ORM para mapear la herencia implica que hay que realizar un join de varias tablas para recuperar todos los datos de un objeto?

a) Single table inheritance.

b) Concrete table inheritance.

c) Class table inheritance. *(Dice que un aspecto negativo de este patrón es que necesita hacer join para cada consulta.)*

¿Cuál de los siguientes tipos de acoplamiento es más débil?

a) Cuando hay una jerarquía de herencia.

b) Cuando una clase implementa una interfaz.

c) Cuando una clase recibe una lista de instancias de otra clase como parámetro en un método.

Al usar un objeto *ServiceLocator* para inyección de dependencias…

a) El cliente solicita directamente las dependencias al ServiceLocator.

b) El cliente solicita las dependencias al objeto Assembler, que las inyecta al ServiceLocator.

c) El ServiceLocator inyecta las dependencias en el cliente a través de su constructor.

¿Qué patrón junta en los mismos objetos la lógica de negocio con la lógica de acceso a datos?

a) Active Record.

b) Data Access Gateway.

c) Domain Model. *(Porque dice que un objeto del modelo de dominio incorpora tanto el comportamiento como los datos.)*

¿Qué mecanismos de inyección de dependencias se pueden usar de forma conjunta?

a) Service Locator e inyección en el constructor.

b) Service Locator e inyección con métodos setter.

c) Inyección en el constructor y con método setter.

¿Qué patrones de lógica de negocio necesitan combinarse normalmente con una capa de servicios para implementar funcionalidades complejas?

a) Table Module y Domain Model.

b) Transaction Script y Table Module.

c) Transaction Script y Domain Model.

¿Qué responsabilidad NO corresponde a la capa de presentación?

a) Mostrar el resultado de ejecutar la lógica de negocio.

b) Activar funcionalidades de capas inferiores.

c) Alojar funcionalidades de alto nivel que no encajan en ninguna clase del modelo de dominio.

¿Cuál de las siguientes afirmaciones sobre la arquitectura en capas es FALSA?

a) La separación en capas favorece un diseño de capas individuales con menor cohesión.

b) La comunicación entre capas disminuye el rendimiento de la aplicación.

c) La separación den capa poco acopladas facilita el diseño de pruebas automatizadas.

GRASP es el acrónimo de patrones generales de software para la asignación de responsabilidades. Con respecto a dicha responsabilidades, ¿Cuáles de las siguientes afirmaciones es falsa?

a) Los métodos se implementan para cubrir responsabilidades.

b) Cada responsabilidad se traduce en un método que se debe asignar alguna clase software.

c) Los métodos pueden colaborar con otros métodos u objetos para cubrir una determinada responsabilidad.

¿Cuál de las siguientes afirmaciones relacionadas con la herencia es verdadera?

a) La herencia escala bien cuando aumenta el número de variaciones del sistema.

b) Siempre que tenemos una jerarquía de herencia, es posible sustituirla por una agregación o composición (delegación), lo cual aumenta la flexibilidad del sistema.

c) La herencia mejora el acoplamiento del sistema.

¿Cuál de los siguientes efectos se puede deber al alto acoplamiento entre clases?

a) Mayor cohesión de las clases fuertemente acopladas.

b) Mayor facilidad de reuso.

c) Alta probabilidad de propagación inadvertida de errores ante modificaciones en el código.

¿Con qué patrón GOF se relaciona el patrón GRASP controlador?

a) Fachada.

b) Fabricación pura.

c) Factory Method.

¿Cuáles de estas clases es más probable que requieran la aplicación de un patrón singletón?

a) Los comandos de un patrón command.

b) Los componentes de un patrón composite.

c) Las factorías concretas de un patrón abstract factory.

Hemos implementado un sistema al que queremos añadir ahora la posibilidad de leer una serie de datos de configuración desde distintas fuentes, como por ejemplo ficheros XML, base de datos, etc. ¿Qué patrón GOFT se adapta mejora este problema?

a) Command.

b) Strategy.

c) Façade.

Nos han pedido diseñar un sistema de aprendizaje... En él, vamos a tener distintos objetos de aprendizaje que se deben instanciar en función del nivel de experiencia del alumno (De momento tres: K12, Adulto, Profesional). ¿Qué patrón aplicarías para solucionar este problema de manera flexible y extensible a nuevos niveles de experiencia? (e.g alumnos universitarios).

a) Builder.

b) Abstract Factory.

c) Proxy.

En el sistema de aprendizaje uno de los requisitos es la construcción de guías docentes. Estas guías están compuestas de un contexto, una serie de objetivos, contenidos, una descripción de la metodología docente, un cronograma, y unos criterios de evaluación. ¿Qué patrón GOFT es el más adecuado para diseñar esta construcción?

a) Builder.

b) Factory method.

c) Abstract factory.

La clase fachada del patrón fachada.

a) Se limita a transmitir invocaciones desde el exterior hacia los objectos del sistema que oculta, preservando la interfaz de estos últimos.

b) Realiza la adaptación entre su interfaz y la interfaz de los objetos del sistema que este oculta.

c) Es una clase abstracta de la que heredan las fachadas concretas de cada componente interno del sistema.

El patrón builder se utiliza para...

a) Permitir la variación de la representación interna de un producto.

b) Simplificar el código cliente que crea objetos complejos.

c) Todas son ciertas.

En patrón OBSERVER ...

a) La clase SUBJEST debe tener acceso de algún modo a los datos del OBSERVER. **

b) La clase OBSERVER debe tener acceso de algún modo a los datos del SUBJEST.

c) No existe ningún acoplamiento entre las clases SUBJEST Y OBSERVER.

El patrón Abstract Factory contribuye a preservar el principio OPEN-CLOSED de Bertrant Merey (Diseños abiertos a la extensión y cerrados al cambio) cuando...

a) El punto más probable de variación supone la adición de nuevas factorías concretas

b) El punto más probable de variación supone la adición de nuevas familias de productos.

c) Los puntos más probables de variación suponen (a) la adición de nuevas familias de producto y/o (b) la adición de nuevas factorías concretas. **

Patrones GOF

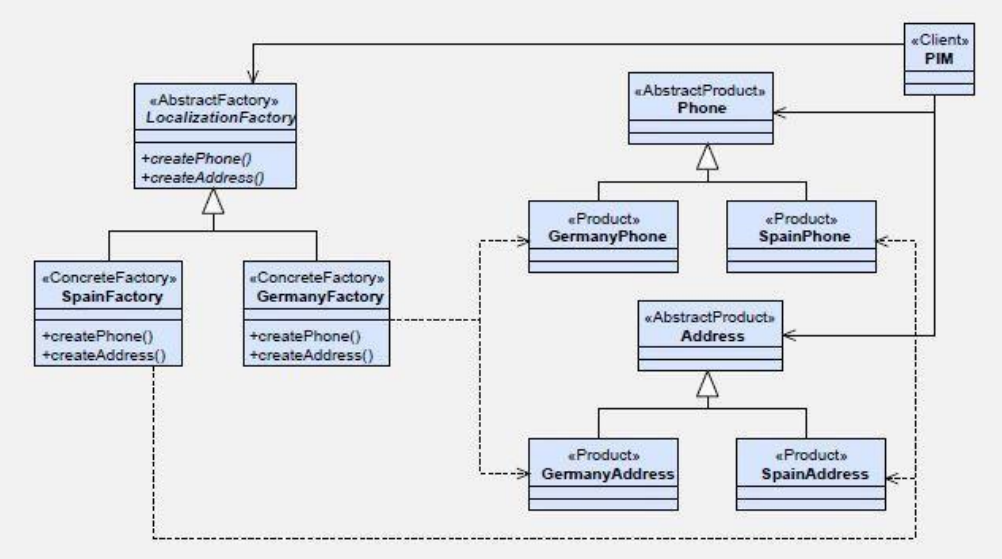
1. Creacionales

– Abstract Factory

Provee una interfaz para crear familias de objetos “producto” relacionados o que dependen entre sí, sin especificar sus clases concretas.

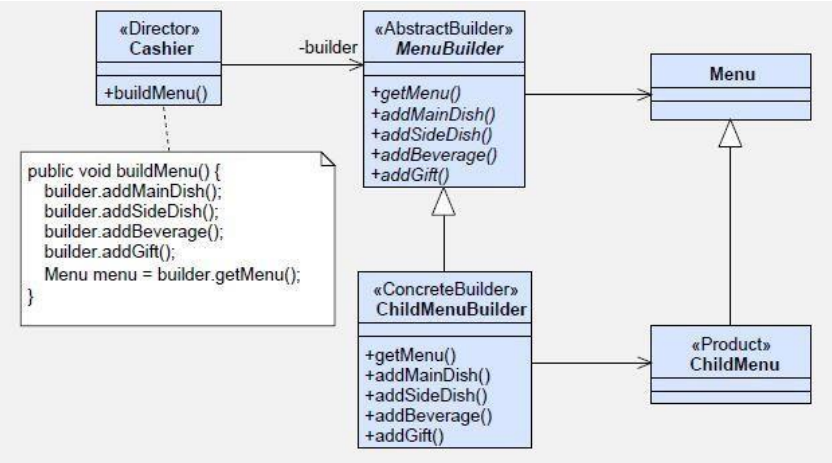
Uso:

- Cuando un sistema debe configurarse con una de las múltiples familias de productos
- Cuando un sistema debe ser independiente de cómo se crean, componen y representan sus productos
- Cuando los productos de la misma familia deben usarse en conjunto, los productos de familias diferentes no deben usarse juntos y esta restricción debe garantizarse
- Solo se revelan las interfaces del producto, las implementaciones permanecen ocultas para los clientes



- **Builder**

  Define una instancia para crear un objeto, pero deja que las subclases decidan qué instanciar y permite un control preciso sobre el proceso de construcción.

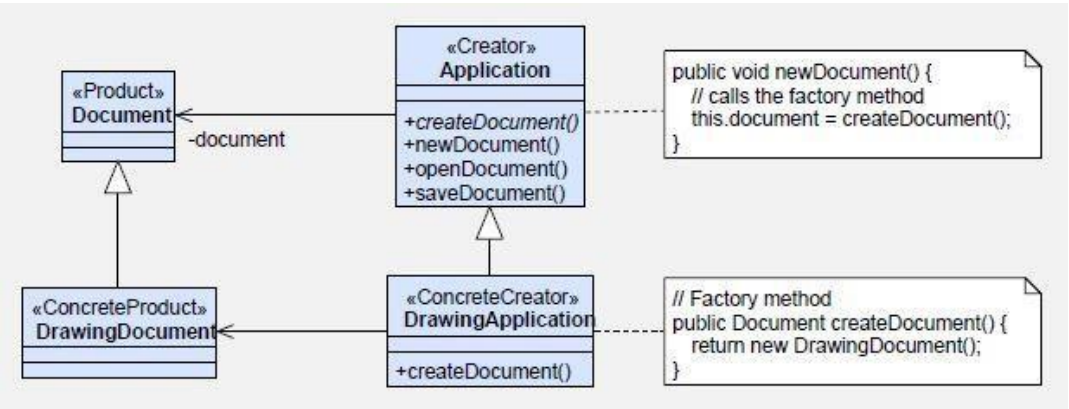


- **Factory Method**

  Define una interfaz para crear objetos, pero deja a las subclases decidir qué clase instanciar y hace referencia al nuevo objeto creado a través de una interfaz común.

  Uso:

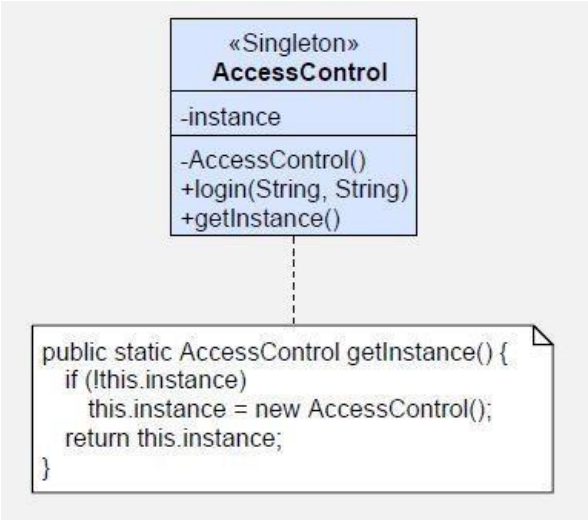
  - Cuando un framework delega la creación de objetos derivados de una superclase común a la fábrica
  - Cuando necesitamos flexibilidad para agregar nuevos tipos de objetos que la clase debe crear
  - Cuando la clase de fábrica base no sabe de qué clases concretas se necesita crear instancias. Delega en sus subclases la creación de objetos concretos
  - Cuando subclases de subclases de fábrica son necesarias para conocer las clases concretas que se deben instanciar



- Singleton

  Asegura que solo se crea una instancia de la clase y provee un punto de acceso global para el objeto.

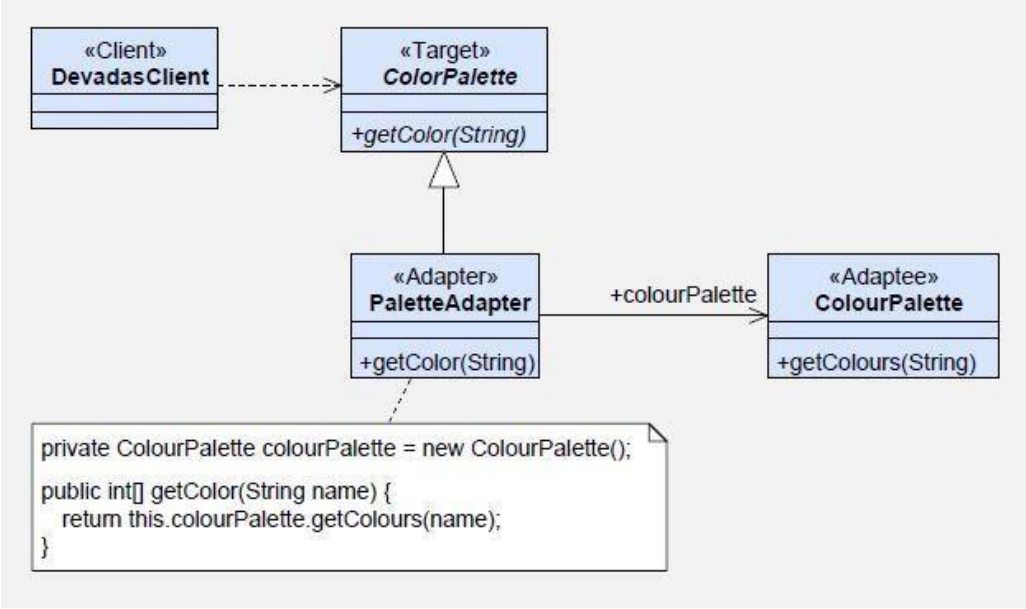
  Uso: Debe usarse cuando debemos asegurarnos de que solo se crea una instancia de una clase y cuando la instancia debe estar disponible a través de todo el código. Se debe tener especial cuidado en entornos milti-threading cuando varios subprocesos deben acceder a los mismos recursos a través del mismo objeto singleton.



2. Estructurales

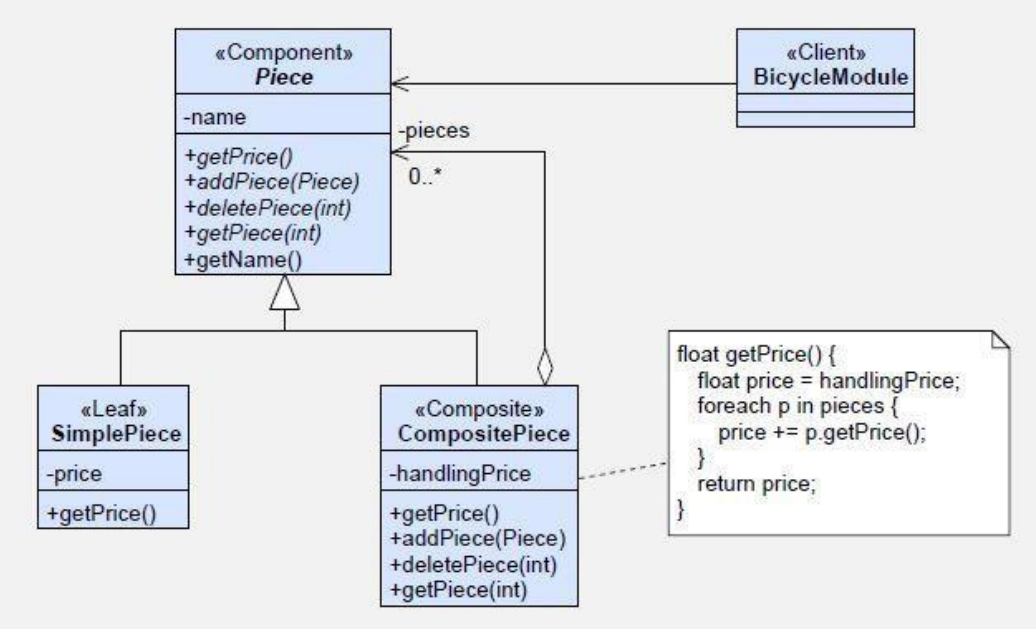
- Adapter

  Convierte la interfaz de una clase en otra interfaz que los clientes esperan. Adapter permite que clases funcionen juntas, que de otra forma no podría ser debido a interfaces incompatibles.



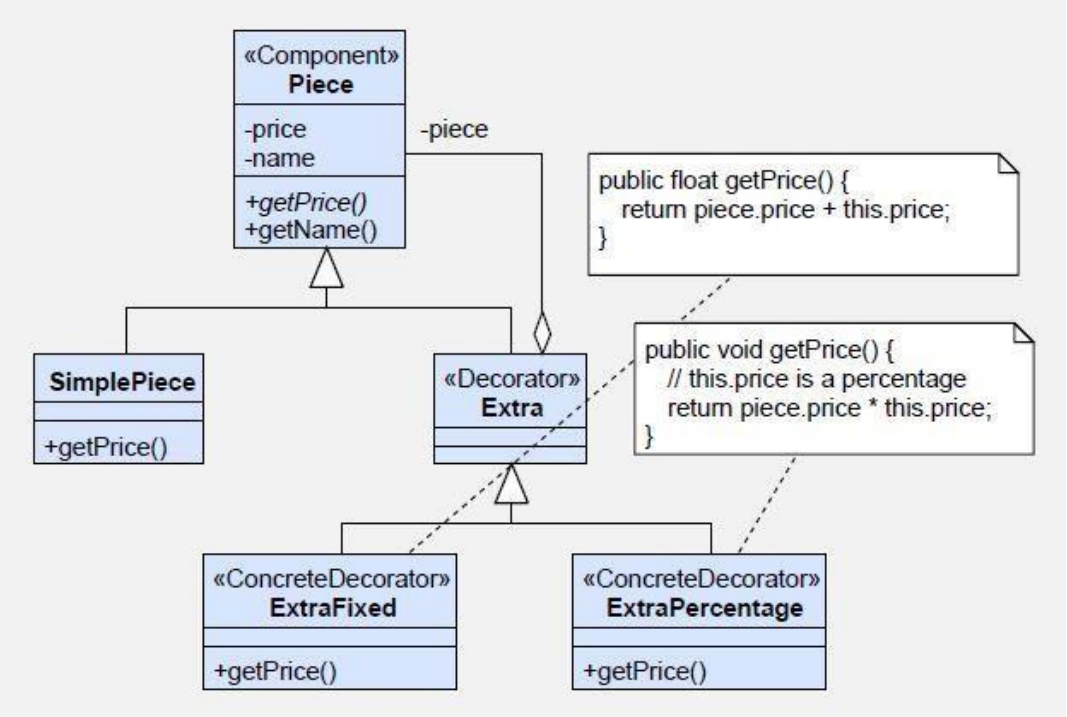
- ## Composite

Compone objetos en estructuras de árboles para representar jerarquías parte-todo. Permite que los clientes traten de manera uniforme a los objetos individuales y a los complejos. Si todas las piezas simples tienen el mismo comportamiento debería evitarse tener una clase para cada pieza simple, sino que es mejor agruparlas en la misma clase.



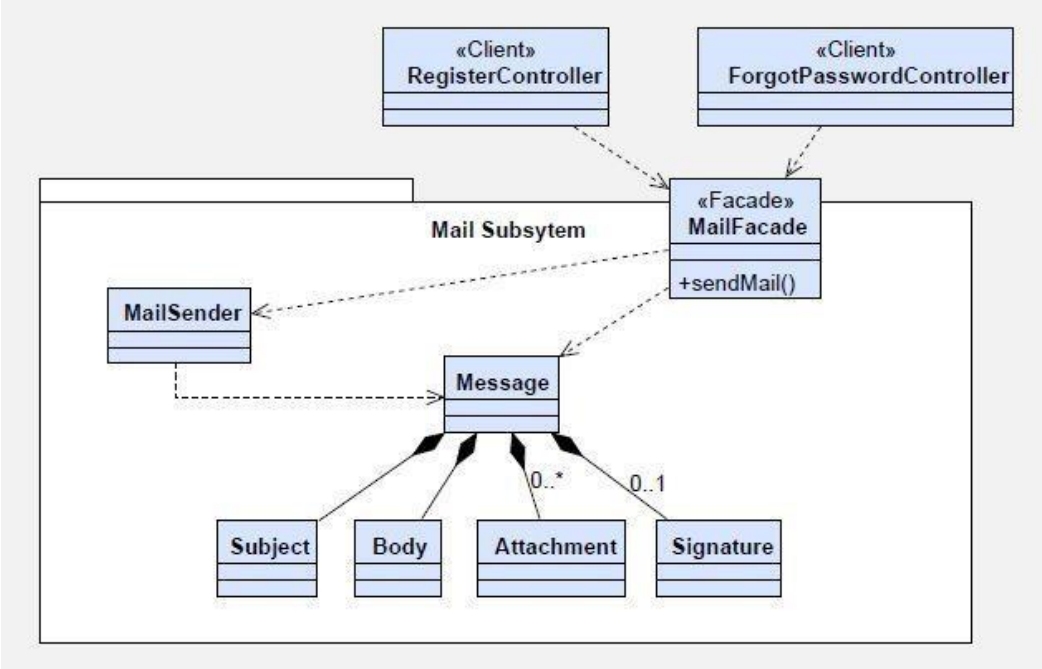
- ## Decorator

Agregar responsabilidades/funcionalidades adicionales dinámicamente a un objeto.



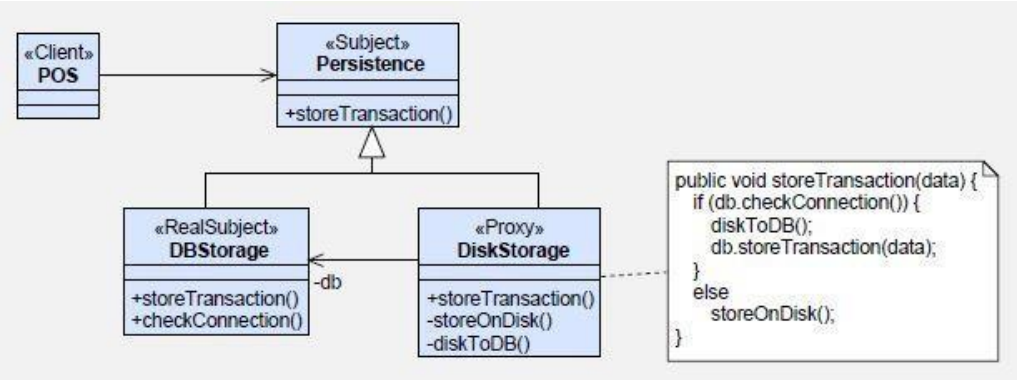
- Facade

  Estructura un entorno de programación y reduce su complejidad con la división en subsistemas, minimizando las comunicaciones y dependencias entre estos.



- Proxy

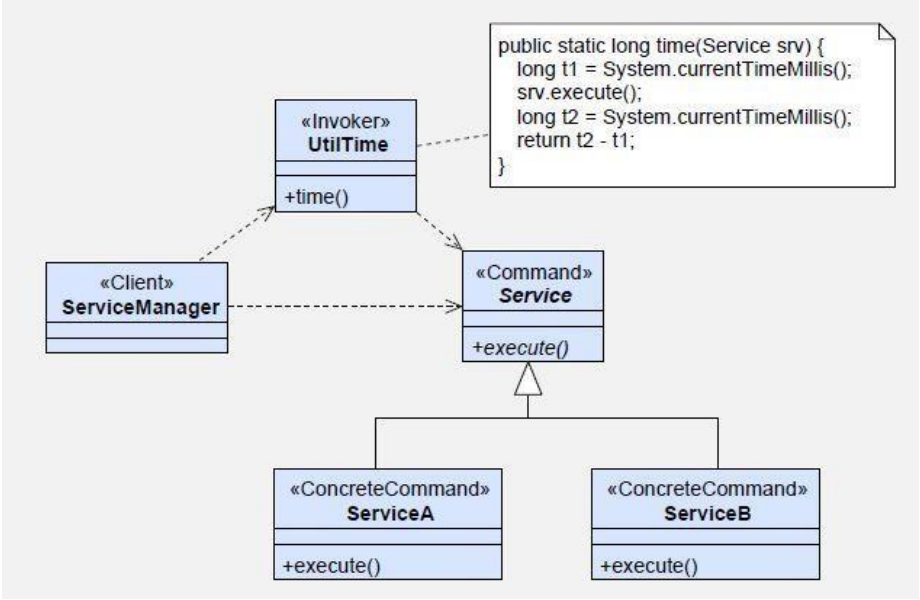
  Proporcionar un intermediario (placeholder) para que un objeto controle las referencias a él.



3. De Comportamiento

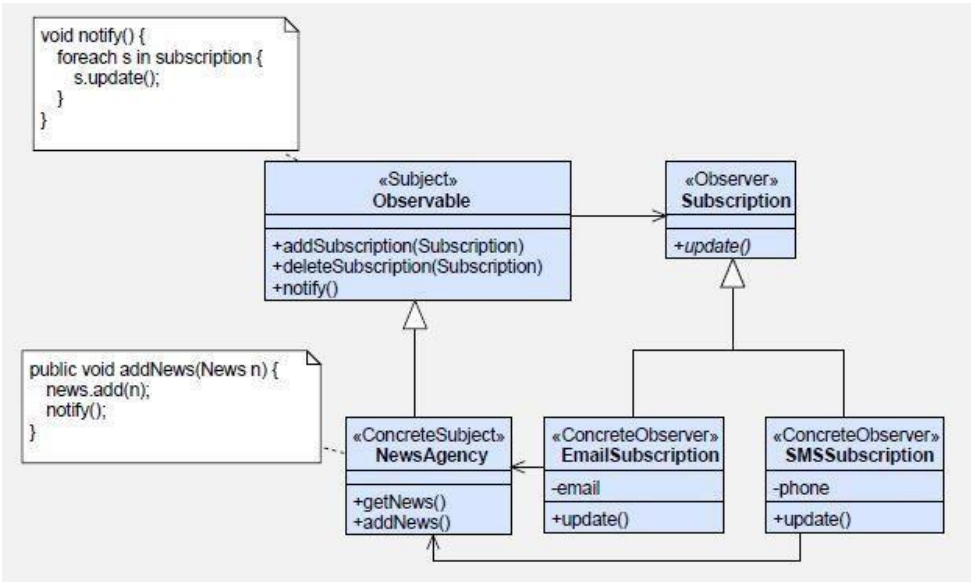
- ## Command

Encapsula una solicitud en un objeto, permite la parametrización de clientes con diferentes solicitudes y permite guardar las solicitudes en una cola. Delega la ejecución de los servicios, de manera que ahora será una clase (Invoker) la encargada de ejecutarlos. Para esto es necesario convertir los servicios (operaciones) en objetos, manteniendo su inicialización donde se hacía originalmente, y añadiéndoles un método que permita ejecutarlos.



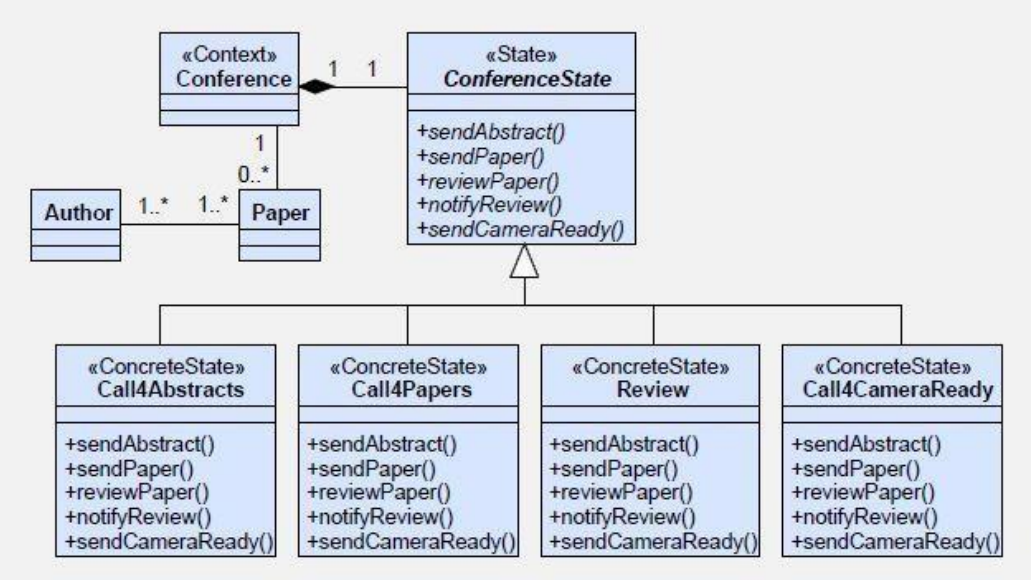
- ## Observer

Define una dependencia de uno a muchos entre los objetos para que cuando un objeto cambie de estado, todos sus dependientes sean notificados y actualizados automáticamente.



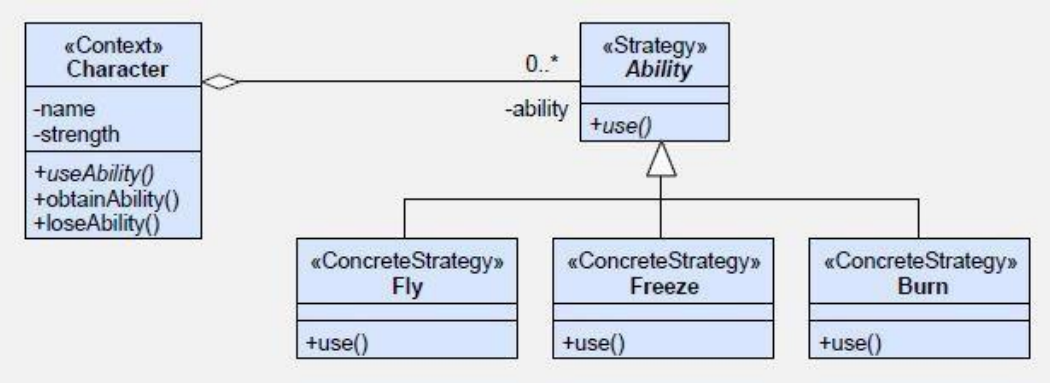
- State

Se utiliza cuando el comportamiento de un objeto cambia dependiendo del estado del mismo.



- Strategy

Define una jerarquía de clases que representan algoritmos, encapsula cada una y hace que sean intercambiables. Estos algoritmos pueden ser intercambiados por la aplicación en tiempo de ejecución. Strategy permite que el algoritmo varíe independientemente de los clientes que lo usan.



- Template Method

  Define el esqueleto de un algoritmo en una operación, difiriendo algunos pasos a subclases. Template Method permite a las subclases redefinir ciertos pasos de un algoritmo sin permitirles cambiar la estructura del algoritmo.