avante
¿hasta dónde quieres llegar?

PROGRAMACIÓN JAVA

avante
¿hasta dónde quieres llegar?

Programación Java – Patrones de diseño

I. Presentación
II. Revisión de los principios de la POO en Java
III. Revisión de patrones de la "Banda de los 4"
IV. Explorar cambios en Java EE

avante
¿hasta dónde quieres llegar?

II. Revisión de los principios de la POO en Java

- Detallar cómo los conceptos OO se aplican a Java.
- Detalle de cómo los principios del OO se aplican en Java.
- Listado de objetivos de un lenguaje OO
- Cómo interpretar notación UML y cómo crear diagramas sencillos.
- Identificar los patrones de diseño seleccionados.

avante
¿hasta dónde quieres llegar?

Revisión de los conceptos básicos de OO

- POO y programación secuencial. Historia y diferencias.
  - POO no es un concepto nuevo (principios de los 60).
  - Se comenzó a utilizar de nuevo con la aparición de C++ en los años 80.
  - Su uso se popularizó en los años 90 sobre todo con el uso de Java.
- Características
  - Resolución de problemas de una manera más natural al ser humano.
- Elementos de la POO
  - Clases, Métodos y Atributos
  - Objetos

Revisión de los conceptos básicos de OO

- Características de la POO
  - Herencia: La clase hija hereda de la clase padre los métodos y atributos
  - Abstracción: Permite capturar la funcionalidad de las clases en métodos y atributos.
  - Encapsulamiento: Reúne todos los elementos pertenecientes a una misma entidad.
  - Modularidad: Permite dividir una entidad compleja en entidades menos complejas (módulos)
  - Ocultación: Permite ocultar parte de la funcionalidad de una clase u objeto a través de contratos o interfaces de funcionamiento
  - Polimorfismo: Permite asociar comportamientos diferentes a un mismo objeto.

Revisión de los conceptos básicos de OO

- OO en Java
  - Herencia: no se permiten herencias múltiples
  - Encapsulamiento: dependiendo del framework de desarrollo es más exigente. Establecido como buena práctica.
  - Modularidad: establecido como buena práctica.
  - Ocultación: establecido como buena práctica.
  - Polimorfismo: Implementado a través de interfaces.

Revisión de los conceptos básicos de OO

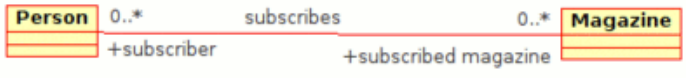
UML, un lenguaje de modelado

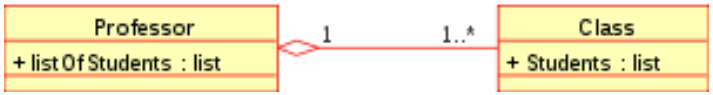
- Permite captar diferentes aspectos de la OO.
  - Estructura
  - Comportamiento
- Principales diagramas
  - Diagrama de clases: estructura de la información y su relación.
  - Diagrama de casos de uso: representación a alto nivel de interacción.
  - Diagrama de actividades: Representa un proceso del sistema.
  - Diagrama de secuencia: Representa la interacción entre objetos del sistema.

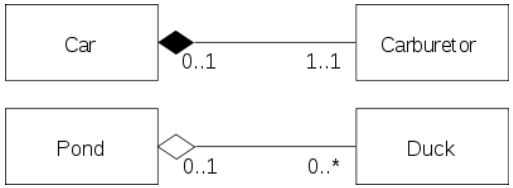
Revisión de los conceptos básicos de OO

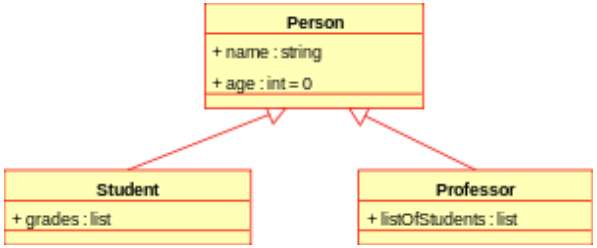
Diagrama de clases

- Asociación
  - Representa la relación entre dos clases
- Agregación
  - Representa una relación de pertenencia
- Composición
  - Representa una relación más fuerte de pertenencia
- Herencia
  - O también llamada generalización



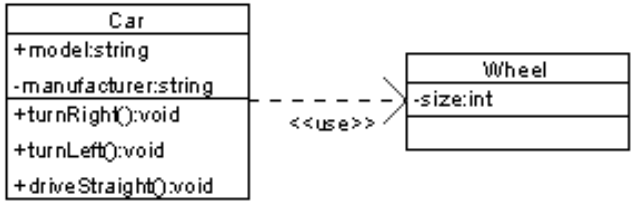






Revisión de los conceptos básicos de OO

Diagrama de clases (otros elementos)



- Dependencia
  - Es un enlace más débil para indicar que una clase puede depender de otra
- Multiplicidad
  - Indica el número de clases relacionadas

0..1	Sin instancias o una instancia
1	Tan solo una instancia
0..*	Cero o más instancias
1..*	Una o más instancias

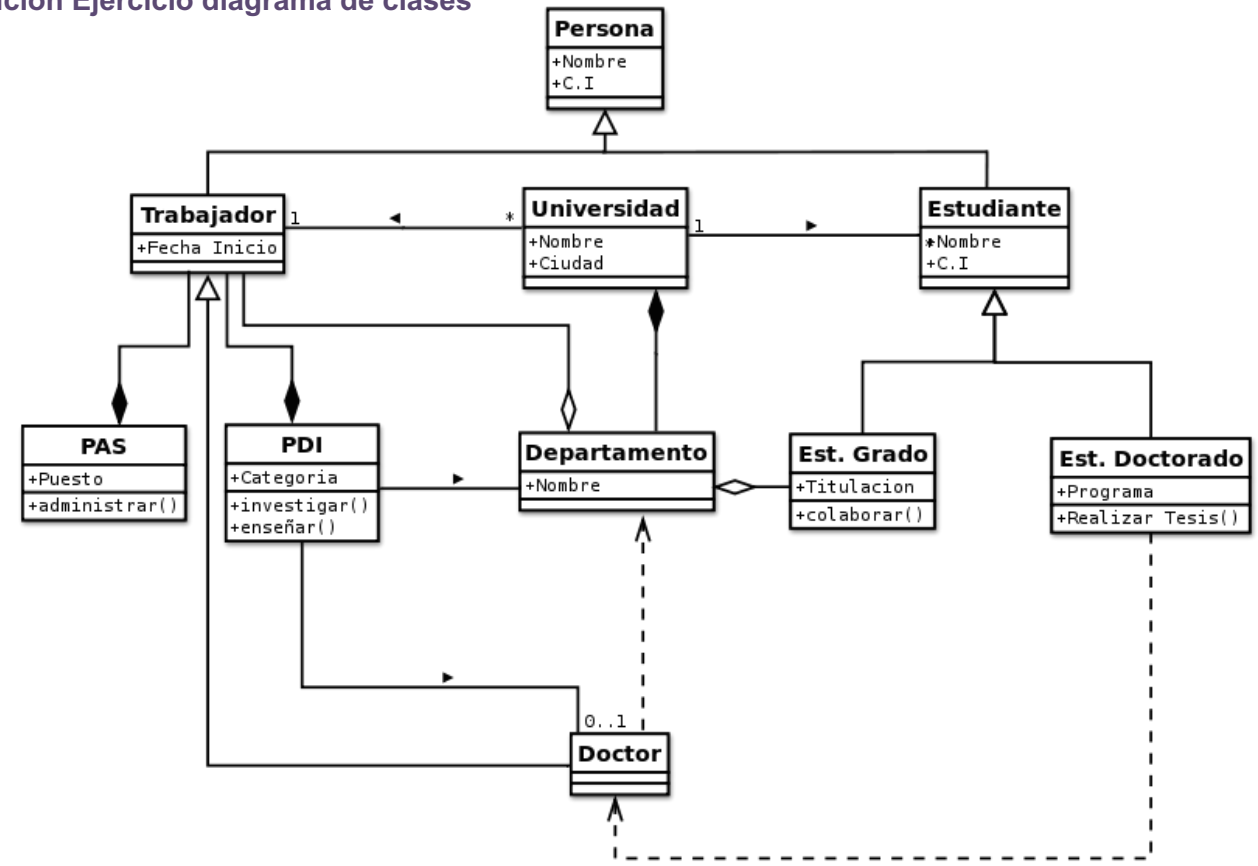
Revisión de los conceptos básicos de OO

Ejercicio diagrama de clases

Se quiere desarrollar un sistema de información para una Universidad según la siguiente descripción. La **Universidad** se caracteriza mediante su nombre y la ciudad donde se sitúa. En la Universidad están vinculados dos tipos de **Persona**: **Trabajadores**, que la Universidad emplea, y **Estudiantes**, que estudian en la Universidad. Cada Persona tiene una CI y un nombre. Los Trabajadores pertenecen a dos grupos: **PDI** y **PAS**. Cada Trabajador tiene asociada una fecha de inicio de su contrato. Cada miembro del PDI también tiene una categoría, mientras que cada miembro del PAS tiene un puesto. Los miembros del PDI pueden o no ser Doctores. Las actividades que desarrolla el PDI son investigar y enseñar, mientras que la actividad que desarrolla el PAS es administrar. La Universidad se compone de un conjunto de **Departamentos**, cada uno de los cuales tiene un nombre y un conjunto de Trabajadores adscrito. Un Trabajador no puede estar adscrito a más de un Departamento. Un PDI está adscrito obligatoriamente a un Departamento, mientras que un PAS, no. Cada Departamento está dirigido por un Doctor. Un Estudiante' puede ser bien **Estudiante de grado**, de una determinada titulación, bien **Estudiante de Doctorado**, de un determinado programa de Doctorado. Un Estudiante de grado puede también colaborar con un Departamento como becario y puede realizar un proyecto de fin de carrera dirigido por un miembro del PDI. Un Estudiante de Doctorado realiza una tesis dirigida por un **Doctor**. Puede suponer que un Estudiante no puede estudiar en más de una Universidad y que un Trabajador no puede ser empleado por más de una Universidad. Proporcione un modelo de esta descripción en forma de un diagrama de clases UML

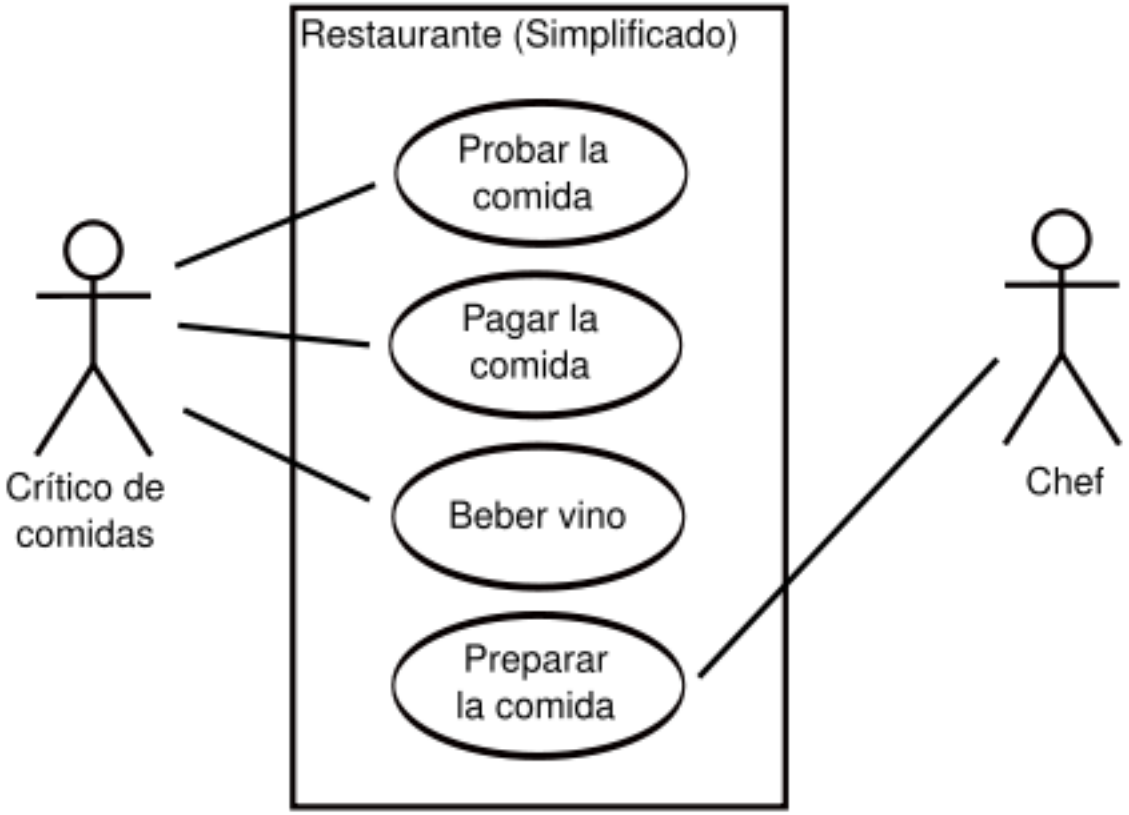
Revisión de los conceptos básicos de OO

Posible Solución Ejercicio diagrama de clases



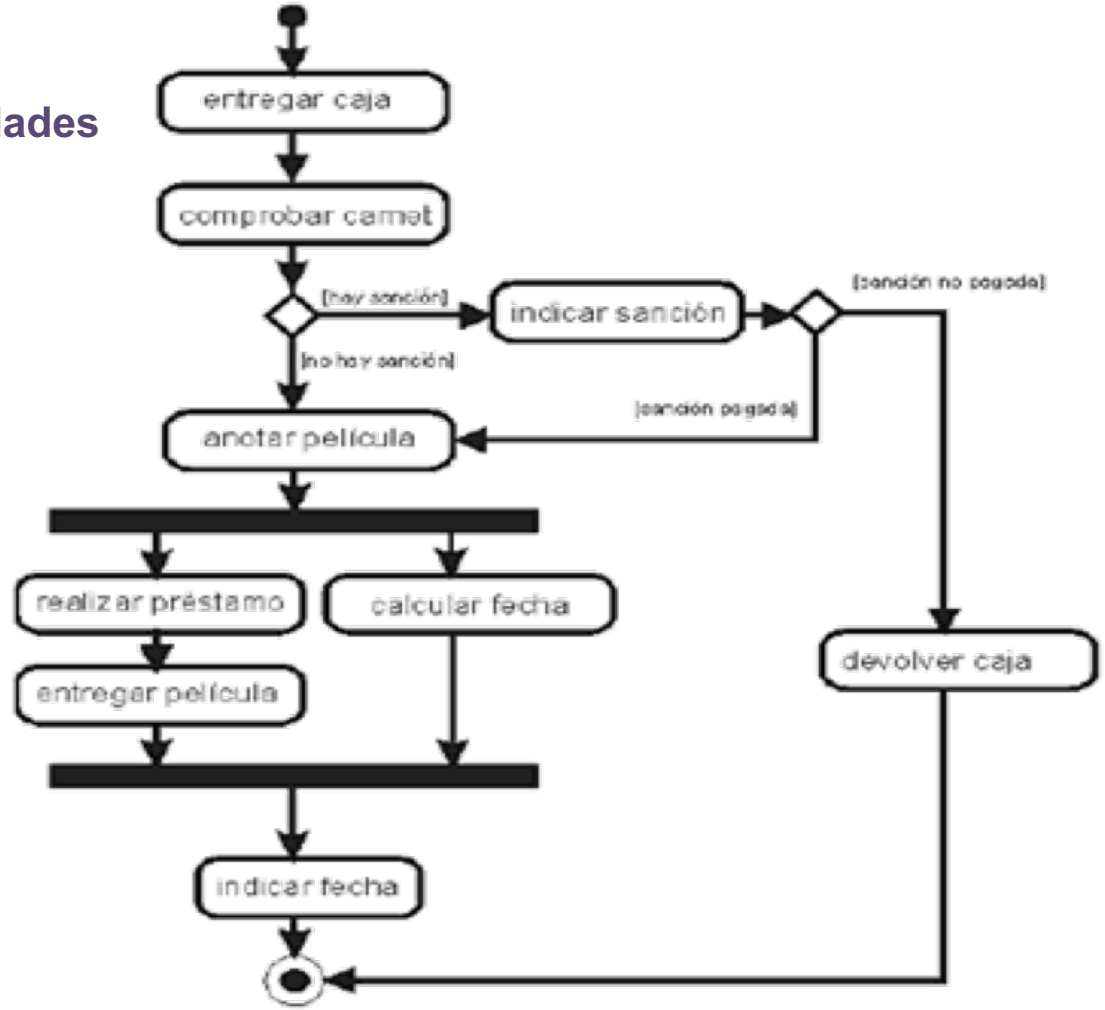
Revisión de los conceptos básicos de OO

Diagrama de casos de uso



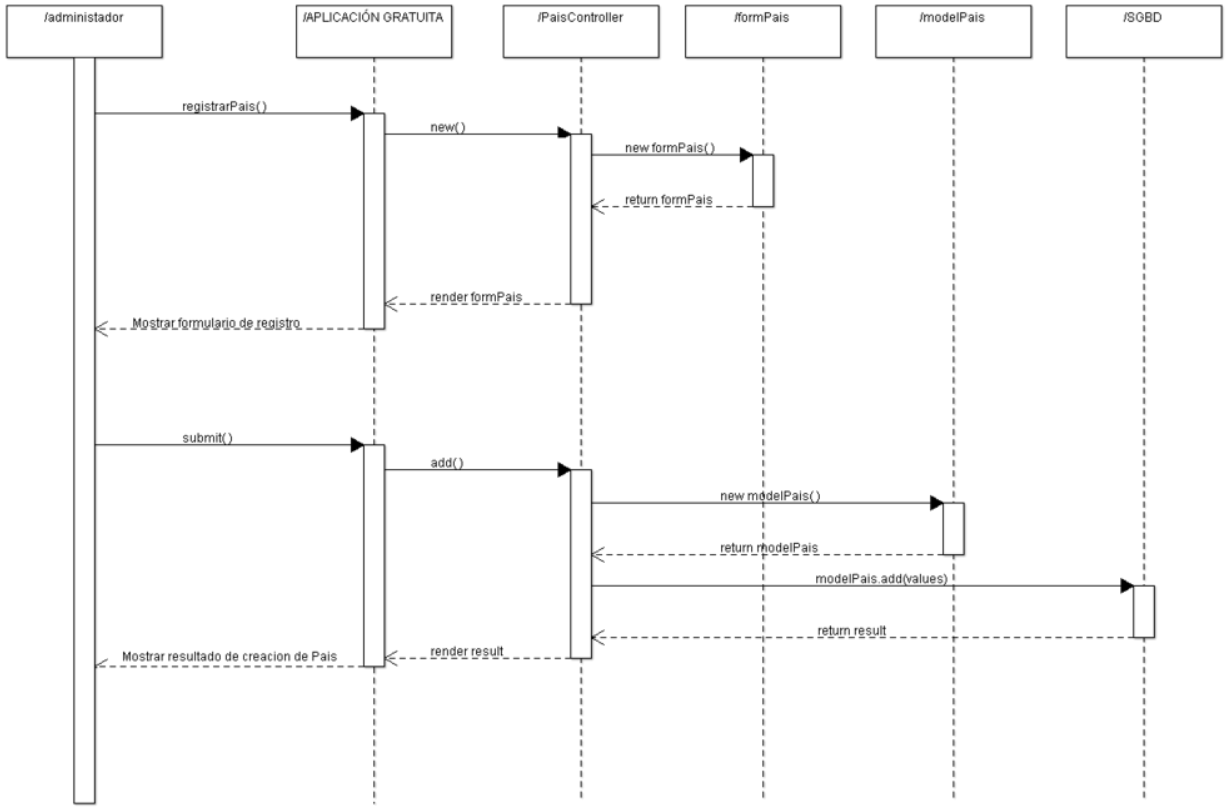
Revisión de los conceptos básicos de OO

Diagrama de actividades



Revisión de los conceptos básicos de OO

Diagrama de secuencia



Revisión de los conceptos básicos de OO

Otros diagramas

- Diagrama de paquetes
- Diagrama de despliegue
- Diagrama de objetos

Más recursos

- www.uml.org
- http://en.wikipedia.org/wiki/Unified_Modeling_Language
- UML 2.0 Pocket Reference

III. Revisión de la 'Banda de los 4'

- Enumerar las pautas clave de comportamiento, creaciones y estructurales.
- Aplicación del patrón 'Facade'.
- Aplicación del patrón 'Strategy'.
- Aplicación del patrón 'Observer'.
- Aplicación del patrón 'Composite'.
- Revisión del patrón 'Model-View-Controller' (MVC)
- Más patrones

Revisión de la 'Banda de los 4'

Antes de empezar…

- ¿Qué son los patrones de diseño?
  - Surgen de la arquitectura a finales de los 70, aplicados en programación en los 80 y popularizados por el GoF en los 90.
  - Resuelven problemas comunes en el desarrollo de software.
- ¿Para qué sirven?
  - Catálogos de elementos reutilizables.
  - Evitar reinventar la rueda.
  - Vocabulario común para los desarrolladores.
  - Estandarizar soluciones y facilitar su aprendizaje.

Revisión de la 'Banda de los 4'

Antes de empezar…

- Tipos de patrones
  - Originalmente había 3 tipos de patrones, pero han ido evolucionando con el tiempo para responder a una mayor diversidad de problemas.
  - Hay muchos tipos de patrones. La división suele basarse en el uso dado.
    - Patrones de creación: para creación de objetos
    - Patrones estructurales: para establecer relaciones entre objetos
    - Patrones de comportamiento: para la comunicación entre objetos
    - Patrones de concurrencia
    - Otros: de testeo, de Interfaz gráfica, de transacciones, de arquitectura distribuida…
- Controversia de los patrones de diseño
  - En ocasiones no son necesarios porque el lenguaje los implementa directamente.
  - En ocasiones pueden incrementar innecesariamente la complejidad del código o traer efectos indeseados sino se aplican correctamente..

avante
¿hasta dónde quieres llegar?

Revisión de la 'Banda de los 4'

Antes de empezar…

- ¿Cómo vamos a estudiar los patrones en este curso? Cubriendo los siguientes puntos.
  - Presentación del problema
  - Presentación del patrón
  - Solución propuesta
  - Posible Implementación
  - Ejemplos o ejercicios

Revisión de la 'Banda de los 4'

Patrón Fachada (AKA Façade)

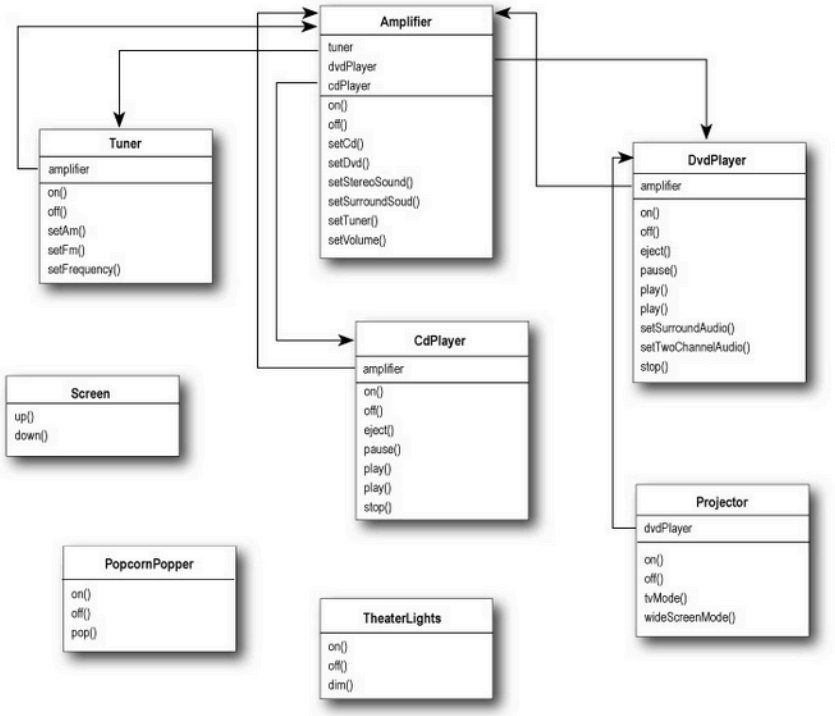
- Presentación del problema
  - Elementos con elevada complejidad que necesitan simplificarse para ser usados.
  - Ejemplo un sistema de creación de email
- Presentación del patrón
  - El patrón fachada proporciona una interfaz que se comunica con todos los elementos comunes.
  - Permite encapsular u ocultar la complejidad del sistema original.
  - Permite usar la funcionalidad original sin usar la interfaz.
- Solución propuesta
  - Crear una interfaz que recoja todos los elementos del sistema y permita hacer uso del mismo de una manera sencilla.

avante
¿hasta dónde quieres llegar?

Revisión de la 'Banda de los 4'

Patrón Fachada Ejemplo (ver una película)

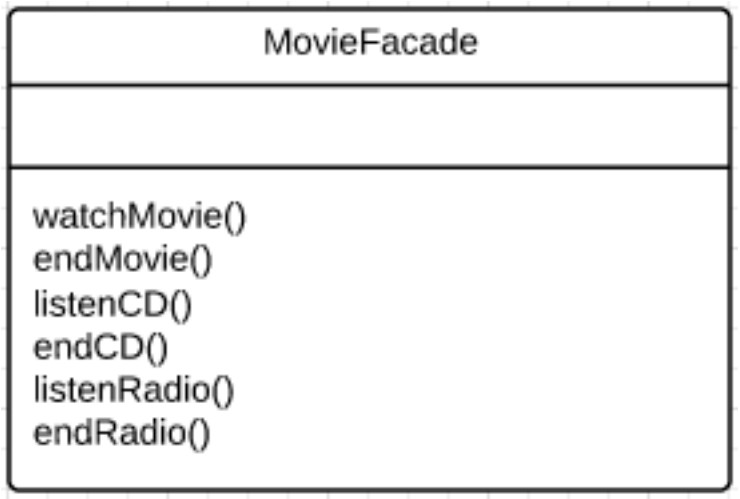
- Encender palomitas
- Hacer palomitas
- Bajar las luces
- Bajar pantalla
- Encender proyector
- Configurar proyector
- …



Revisión de la 'Banda de los 4'

Patrón Fachada Solución ejemplo (ver una película)

- Gestionado desde una clase
- Relacionada con todas las clases
- Complejidad oculta
- Podemos acceder a bajo nivel



Revisión de la 'Banda de los 4'

Patrón Strategy (AKA Policy)

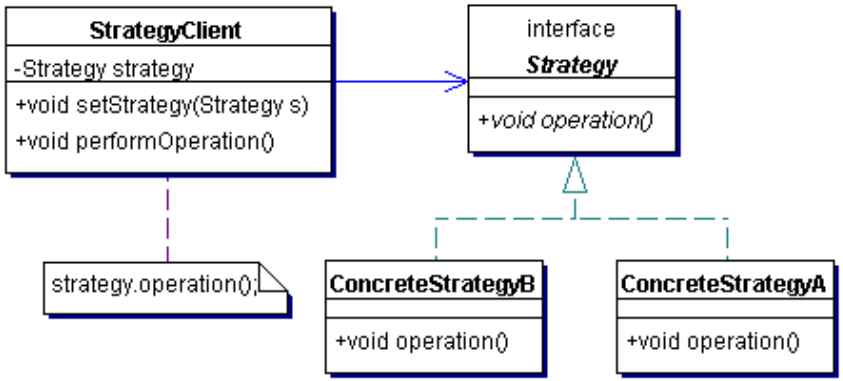
- Presentación del problema
  - A veces no sabemos que forma va a adoptar un problema hasta que se presenta.
  - Ejemplo un sistema de generación de archivos en distintos formatos.
  - Ejemplo sistema que muestre diversos calendarios.
- Presentación del patrón
  - El patrón strategy es un patrón de comportamiento que permite definir distintas clases para tratar varios aspectos de un problema.
- Solución propuesta
  - En lugar de implementar toda la lógica en una sola clase se generan distintas clases con los mismos métodos de acceso para tratar un problema de distinta manera.

Revisión de la 'Banda de los 4'

Patrón Strategy

Más sobre el patrón strategy…

- Muy relacionado con el patrón Factory.
- Simplifica el cliente y delega la complejidad a las implementaciones.
- Permite incrementar fácilmente expansiones en la funcionalidad.



Revisión de la 'Banda de los 4'

Patrón Strategy Ejemplo com.avante.ejemplo2

- Ejemplo de visor de contactos que puede implementar varios tipos de contactos.
- Una interfaz que define los métodos comunes de la estrategia.
- Dos tipos de contactos, personales y de organización.
- Una clase que gestiona los métodos comunes definidos en una interfaz y dos implementaciones distintas.

avante
¿hasta dónde quieres llegar?

Revisión de la 'Banda de los 4'

Patrón Strategy Ejercicio com.avante.ejercicio3

- Tenemos una interfaz que tiene un método sort y una clase que definen una estrategia para ordenar.
- Crear otras dos estrategias de ordenación.
- Un cliente que implemente la carga de la estrategia y ejecute la acción.
- Una clase de ejemplo que nos permita ejecutar nuestra estrategia.
- Solución en com.avante.solucion3

Revisión de la 'Banda de los 4'

Patrón Observer (AKA Publisher-Subscriber)

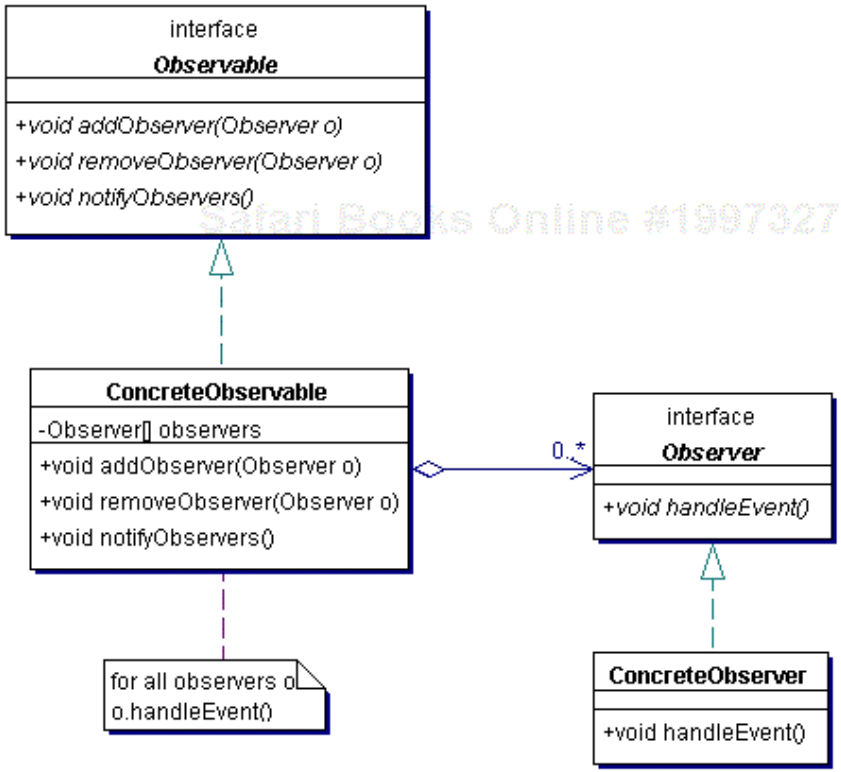
- Presentación del problema
  - Elementos que necesitan notificar que han cambiado.
  - Incrementar el número de sistemas u objetos que están escuchando.
- Presentación del patrón
  - El patrón observer permite definir unos métodos estándar que recogerán las clases que observan las notificaciones.
  - Permite tener un sistema fácilmente extensible y con una interfaz común.
  - Permite extender los objetos que observan los cambios de una manera sencilla.
- Solución propuesta
  - Crear una interfaz donde se añaden y quitan los observadores y se añaden los diversos tipos de notificaciones.
  - Una clase que implemente el interfaz y gestione todos los observadores.
  - Una interfaz que defina como se comunica con los clientes.
  - Por último, las clases que actúan al recibir las notificaciones.

Revisión de la 'Banda de los 4'

Patrón Observer

Más sobre el patrón Observer

- Patrón de comportamiento muy utilizado para el manejo de eventos.
- Muy útil para testeo al poder implementar observadores de prueba
- Presentan un problema dependiendo de la complejidad del mensaje a transmitir
  - Un mensaje genérico traslada complejidad a los observadores al tener que decidir si el evento es para el o no.
  - Un mensaje muy específico traslada la complejidad al componente observable para determinar donde se envían.



avante
¿hasta dónde quieres llegar?

Revisión de la 'Banda de los 4'

Patrón Observer Ejemplo com.avante.ejemplo3

- Ejemplo de Observer que envía notificaciones sobre el estado de una tarea a todos los escuchadores en un GUI.
- La clase Task es observada. TaskChangeObservable es la clase que añade los Observer y la que define cuando se debe notificar. En este caso ActionListener sería nuestra Interfaz.
- La interfaz TaskChangeObserver define los métodos que tiene que implementar los Observadores
- ObserverGui es una clase del soporte del ejemplo.
- Tenemos tres Observadores que implementan el comportamiento al recibir la notificación. La edición de una tarea (creación y edición), el histórico de cambios y un selector de tareas.

avante
¿hasta dónde quieres llegar?

Revisión de la 'Banda de los 4'

Patrón Observer Ejercicio

- Vamos a preparar un sistema que va a notificar a dos distintos observadores de la disponibilidad de un producto en stock.
- Crear una interfaz que permita registrar nuestros observadores y notificarlos.
- Un método donde se implementen los observadores y se haga gestión de una variable que indique si hay stock o no.
- Una interfaz con un método enviarComunicacion para gestionar la notificación
- Dos clases que nos permitan gestionar el evento de manera distinta
    - NotificacionEmail, que en teoría enviaría un correo.
    - NotificacionApp, que en teoría enviaría una notificación a través de un sistema de notificaciones para aplicaciones móviles.
- Una clase que nos permita ejecutar el ejemplo.
- Solución en com.avante.solucion4

Revisión de la 'Banda de los 4'

Patrón Composite

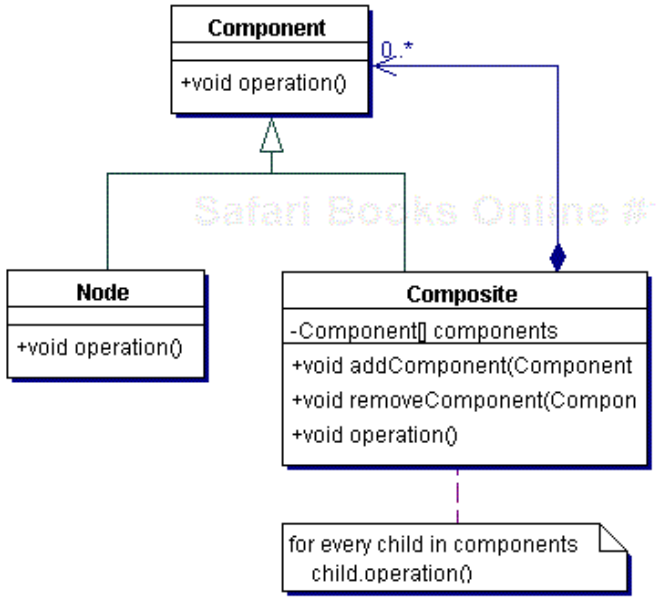
- Presentación del problema
  - A veces tenemos un objeto complejo que queremos separar en una jerarquía de objetos.
  - Al tener una relación compleja de objetos se incrementan mucho las relaciones y sus implementaciones.
  - Ejemplo estructura de un documento.
- Presentación del patrón
  - El patrón composite permite definir una interfaz única que ayuda a simplificar las distintas operaciones. Para ello crea una estructura jerárquica.
  - Permite incrementar el nivel de complejidad de forma dinámica de una manera sencilla.
  - Permite tratar de manera uniforme los componentes utilizando operaciones comunes.
- Solución propuesta
  - Crear una interfaz implementada por todos los objetos. Dicha interfaz define métodos comunes a la estructura jerárquica.
  - Una clase que contenga a todos los distintos objetos (el compuesto o Composite). Esta clase define el comportamiento común de los componentes. Es la rama de la estructura jerarárquica.
  - Las clases hoja que implementan el interfaz y no contienen referencias a otros componentes.

Revisión de la 'Banda de los 4'

Patrón Composite

Más sobre el patrón

- Nos proporciona flexibilidad a la hora de crear la estructura de componentes.
- La interfaz puede no ser suficiente a la hora de declarar funcionalidad aunque se podría usar una clase abstracta.
- Al ser tan flexible y dinámico puede ser difícil testearlo.



avante
¿hasta dónde quieres llegar?

Revisión de la 'Banda de los 4'

Patrón Composite Ejemplo com.avante.ejemplo4

- Ejemplo de Composite que calcula el tiempo requerido para completar un proyecto o parte de el.
- La clase Deliverable que representa un producto completado de una tarea completada
- La clase Project que representa el proyecto global y contiene una colección de elementos de proyecto.
- El Interfaz ProjectItem que representa el funcionalidad común a todos los elementos del proyecto.
- La clase Task que representa un objeto de proyecto.

avante
¿hasta dónde quieres llegar?

Revisión de la 'Banda de los 4'

Patrón Composite Ejercicio com.avante.ejercicio5

- Implementar a partir de las clases del ejemplo anterior un tipo Subtask.
- Como elemento diferenciador pondremos que las subtareas solo pueden estar compuestas por entregables.
- Dicho tipo debe integrarse como otro elemento compuesto más.
- ¿Qué cambios tendría que tener la clase Task para calcular el tiempo?
- Crear una clase de prueba que haga una estructura de proyecto y calcule el tiempo
- Solución en com.avante.solucion5

Revisión de la 'Banda de los 4'

Patrón Factory

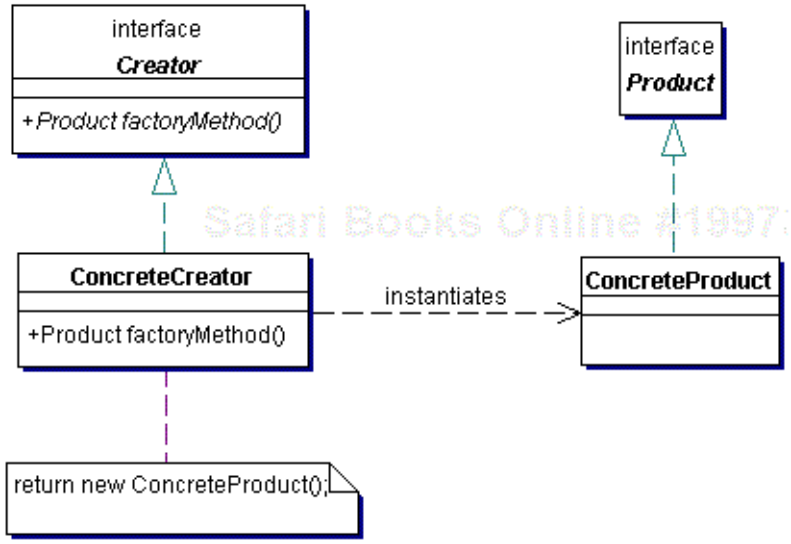
- Presentación del problema
  - A veces no sabemos que tipo de objeto debemos crear hasta que ejecutamos un problema.
  - Es posible que en el futuro necesitemos crear nuevos tipos de objetos y queremos mantener un código fácil de mantener.
  - Ejemplo estructura de creación de documentos genéricos. Tan solo necesitamos saber ciertos metadatos pero no los detalles que serán gestionados posteriormente.
  - Ejemplo de conexiones a bases de datos (JDBC)
- Presentación del patrón
  - El patrón factory permite definir un entorno de creación de objetos extensible. Donde podemos crear diferentes implementaciones de objetos con un marco común.
- Solución propuesta
  - Crear una interfaz implementada por todos los objetos. Dicha interfaz define los constructores permitidos de nuestra factoría.
  - La clase que implementa los constructores y la lógica de selección de los objetos a crear. Esta es la que usaremos para crear nuestros objetos.
  - Un interfaz que define las características comunes de nuestros objetos.
  - Las clases que implementan las características de nuestro producto y que serán instanciadas por nuestro creador.

Revisión de la 'Banda de los 4'

Patrón Factory

Más sobre el patrón

- Nos proporciona flexibilidad a la hora de crear y añadir nuevos tipos de componentes.
- Un problema puede ser que necesitamos crear una nueva clase por cada objeto y modificar la clase encargada de crear las distintas instancias. En algunos casos extremos se puede incrementar mucho la complejidad del creador.



avante
¿hasta dónde quieres llegar?

Revisión de la 'Banda de los 4'

Patrón Factory Ejemplo com.avante.ejemplo5

- Ejemplo de Factory que crea distintos tipos de perro.
- La interfaz CreadorPerros define el método constructor, en este caso es getPerro.
- El constructor el FactoriaPerro, donde se decide que instancia crear en función de un criterio definido.
- El interfaz Perro que define las capacidades de un perro, en este caso un único método ladra.
- Tres clases distintas de Perro que implementan maneras distintas de ladrar.

avante
¿hasta dónde quieres llegar?

Revisión de la 'Banda de los 4'

Patrón Factory Ejercicio

- Crear una factoría de formas geométricas.
- Debemos definir al menos tres formas geométricas que tengan un método para dibujarlas.
- Crear una clase de ejemplo que muestre como se utilizan.
- Solución en com.avante.solucion6

Revisión de la 'Banda de los 4'

Patrón Model-View-Controller o MVC o Modelo-Vista-Controlador

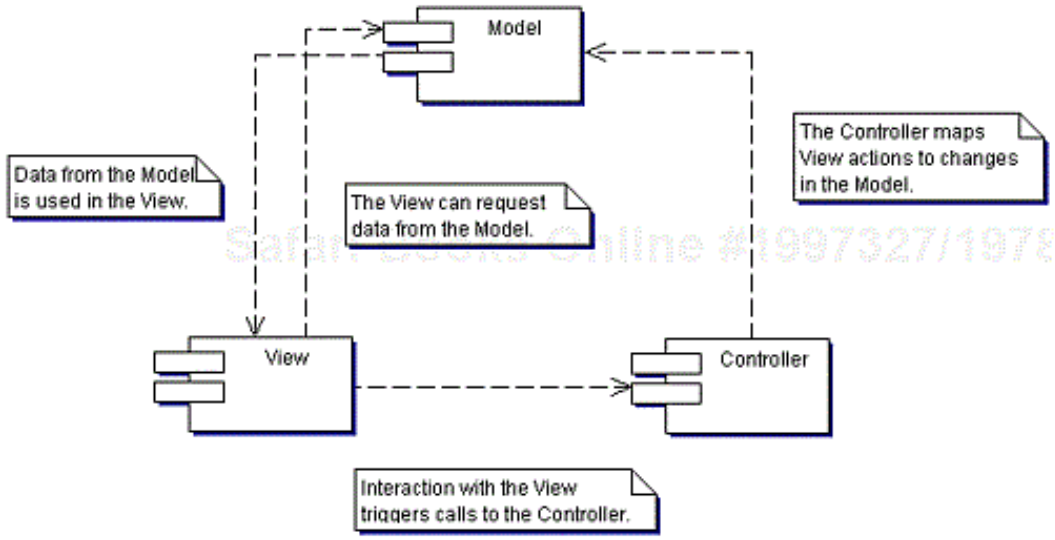
- Presentación del problema
  - A medida que creamos un sistema se va haciendo más compleja la división entre lógica, presentación y datos. Si todo está mezclado nos será más complejo escalar un sistema y dividir las tareas de desarrollo y mantenimiento.
  - También la extensión de nuestro sistema puede verse comprometida al ser cada vez más complejo introducir nuevos elementos en el sistema sin una separación apropiada.
- Presentación del patrón
  - El patrón MVC permite separar entre presentación, estructura de datos y la lógica que aúna esos elementos.
- Solución propuesta
  - El patrón define un modelo donde se especifica los estados del elemento y como se cambian.
  - También disponemos de una vista que nos da la representación de nuestro modelo.
  - Por último disponemos de un controlador donde se mapean las acciones de la vista y el impacto que tienen en el modelo.

Revisión de la 'Banda de los 4'

Patrón Modelo-Vista-Controlador

Más sobre el patrón

- Muchos frameworks de desarrollo en Java implementa de una u otra manera este Patrón. Esto es una ventaja ya que no tendremos que implementarlo nosotros, pero también cada framework lo trabaja de una manera distinta con diferente grado de complejidad.



avante
¿hasta dónde quieres llegar?

Revisión de la 'Banda de los 4'

Patrón Model-View-Controller Ejemplo com.avante.ejemplo6

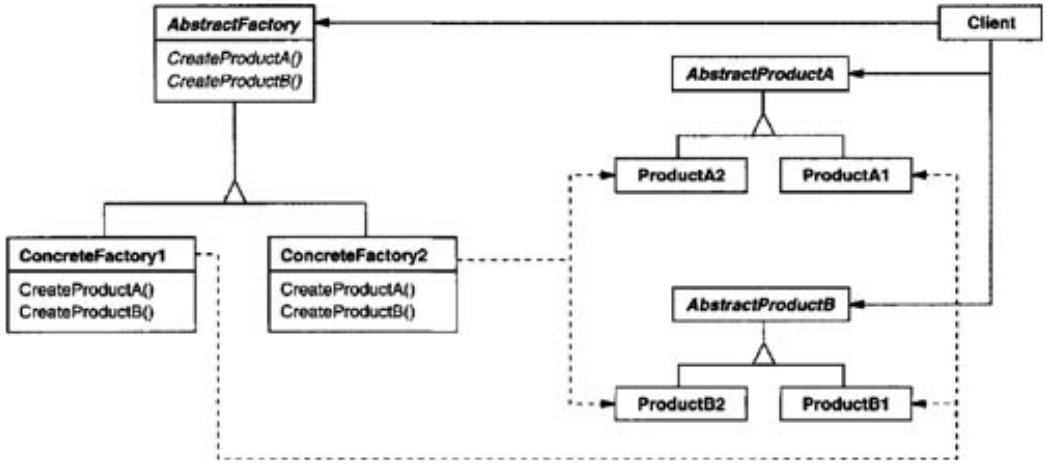
- Ejemplo de MVC que mantiene una lista de contactos.
- Un modelo de Contacto (ContactModel) donde se define el modelo a mantener.
- Dos vistas para mostrar nuestros contactos, una que solo permite ver el contacto (ContactDisplayView) y otra que permite editar nuestro contacto (ContactEditView)
- Una interfaz de vista de Contacto (ContactView) donde se especifica el método de las vistas.
- Un controlador donde se controlan las acciones sobre el modelo y las vistas a mostrar.

Revisión de la 'Banda de los 4'

Más patrones

Abstract Factory

- Es una variación del patrón Factory pero que nos permite definir una familia de factorías. Muy útil cuando queremos definir diferentes familias de productos con sus propias factorías.



Revisión de la 'Banda de los 4'

Más patrones

Abstract Factory Ejemplo com.avante.ejemplo8

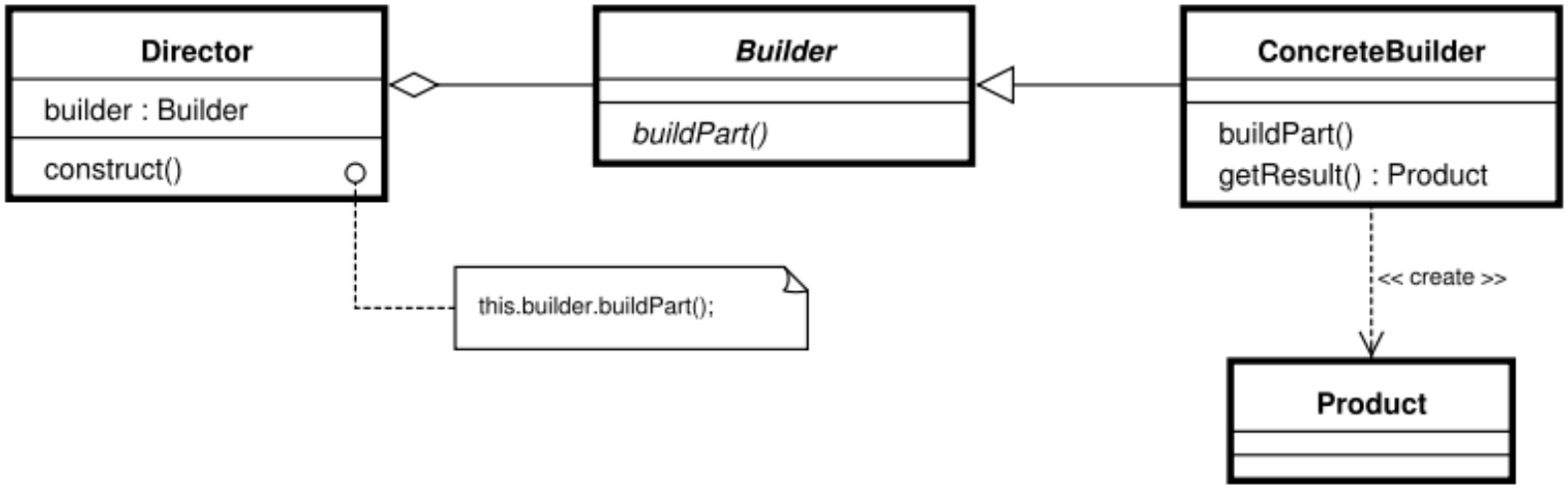
- Es una variación del ejercicio del patrón factoría.
- Tenemos una factoría de formas con las clases que implementan los métodos comunes y otra de colores que implementan otro método distinto.
- Tenemos una clase abstracta que define todos los métodos de la factoría abstracta.
- Por último tenemos una factoría de factorías que nos devuelve la factoría necesaria.
- Ejercicio. Crear clase de ejemplo de ejecución. Solución en com.avante.solucion7

Revisión de la 'Banda de los 4'

Más patrones

Builder

- Nos ayuda a construir un objeto complejo de una manera sencilla. Es usado a menudo con el patrón Composite.



avante
¿hasta dónde quieres llegar?

Revisión de la 'Banda de los 4'

Más patrones

Builder Ejemplo com.avante.ejemplo9

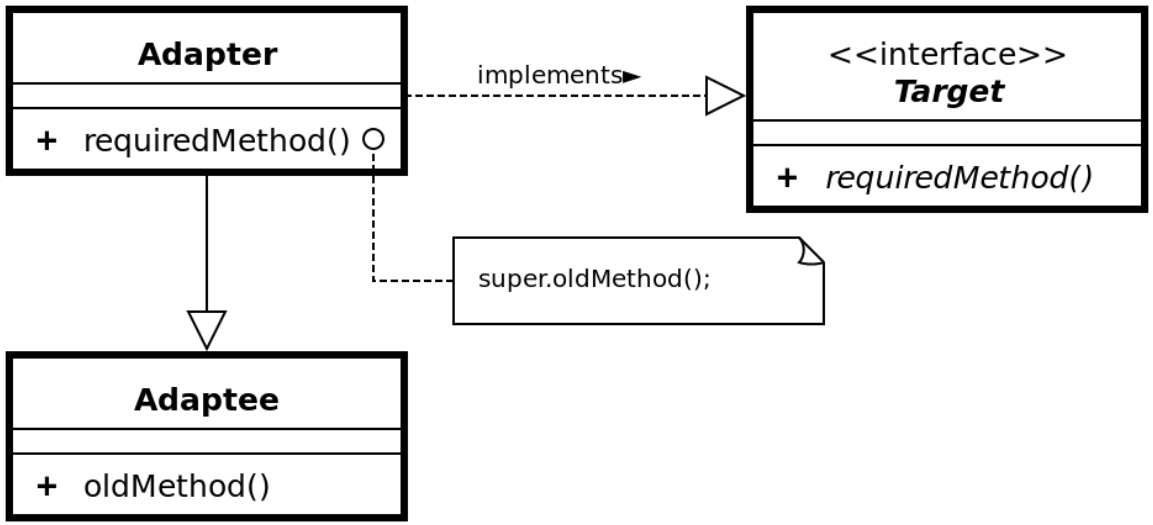
- Queremos crear una cocina que cree diversos tipos de pizzas.
- Necesitamos una clase que defina como son las pizzas (clase Pizza)
- También una clase que defina a grandes rasgos como se crea una pizza (PizzaBuilder). Todas las implementaciones de nuestra pizza serán subclases de esa clase.
- Por último necesitamos un director que sea el que construya nuestra clase.
- Este ejemplo es sencillo. Uno más complejo podría ser la creación de un documento según el ejemplo de Composite que hemos visto. La complejidad aumentaría al tener que definir una pequeña estructura jerárquica para definir correctamente la estructura de nuestro documento.
- Ejercicio. Crear clase de ejemplo de ejecución. Solución en com.avante.solucion8

Revisión de la 'Banda de los 4'

Más patrones

Adapter

- Como indica su nombre nos ayuda a adaptar clases entre si.



Revisión de la 'Banda de los 4'

Más patrones

Adapter Ejemplo com.avante.ejemplo10

- Queremos adaptar nuestro vehículo a utilizar un motor eléctrico o un motor común. Para ello vamos a utilizar el patrón Adapter.
- Tendremos una definición preliminar de lo que puede hacer un motor en nuestra clase Motor.
- En motor común tendremos una serie de comandos que nos ayudan a utilizar nuestro motor.
- El motor eléctrico es más complejo tal y como refleja su clase. Para poder utilizarlo en el mismo contexto que el motor común necesitamos un adaptador, en este caso MotorElectricoAdapter.

- Ejercicio. Crear clase de ejemplo de ejecución. Solución en com.avante.solucion9

avante
¿hasta dónde quieres llegar?

IV. Explorar los cambios en la tecnología JAVA EE

- Descripción de los objetivos de diseño del modelo de Java EE
- Descripción de las mejoras en el modelo Java EE 7.

Explorar los cambios en la tecnología JAVA EE

Breve descripción de JEE

- Java EE es una plataforma de desarrollo multi-capa y distribuida basado en Java.
- A lo largo del tiempo se ha alejado de configuraciones engorrosas en XML para utilizar anotaciones en Java.
- El desarrollador no tiene que preocuparse de abarcar toda la problemática de un sistema complejo y básicamente tiene que definir la lógica de negocio de la aplicación, delegando todo lo demás a JEE y sus diversos componentes.
- Es una especificación que define componentes. Puede haber varias implementaciones de cada componente.
- Dispone de varios componentes estos son algunos destacados:
    - Enterprise JavaBeans que nos ayuda a gestionar la persistencia, llamadas a procedimientos remotos, control de concurrencia, transacciones y control de acceso a objetos distribuidos.
    - Java Naming para gestionar servicios de directorio (búsqueda a través de nombres).
    - JTA para controlar transacciones.
    - JPA para controlar la persistencia.
    - JavaServer Pages (JSP) para creación de web dinámicas.
    - JavaServer Faces (JSF) para creación de interfaces a partir de componentes

Explorar los cambios en la tecnología JAVA EE

Cambios en JEE

Veamos brevemente algunos cambios en JEE7 con respecto a JEE6.

- Adición de nuevas tecnologías
  - Utilidades de concurrencia: Para soportar concurrencia asincrona a los componentes.
  - Java API para JSON-P: Para poder comunicarnos con JSON en diferentes dominios.
  - Java API para WebSockets: para comunicación a través de TCP.
  - Batch Applications: Para generar tareas por lotes.
- Mejoras en componentes
  - EJB: mejoras en sesiones locales asincronas
  - Servlets: mejoras en bloqueos para mejorar la escalabilidad de la aplicación.
  - JSF: mejoras en el lenguaje de marcado para HTML5, Flujos…
  - JMS: simplificación de la API y mejoras para su uso en Java SE 7.

Explorar los cambios en la tecnología JAVA EE

Críticas a JEE

- Complejidad de la plataforma: La plataforma dispone de muchos componentes y aunque no es necesario conocerlos todos en profundidad si que tiene una curva de aprendizaje alta.
- No es la plataforma más ágil. Aunque robusta y bien probada quizá tarda demasiado tiempo con respecto a otros Frameworks más activos como Spring.
- Muchas veces optamos por JEE cuando podríamos encontrar otras soluciones con menor complejidad de arquitectura. Un ejemplo podría ser el uso de JEE para aplicaciones que tan solo orientadas a la Web.
- Cada componente puede ser comparado con otras tecnologías que están en un estado más avanzado y con menor complejidad. Oracle ha ido reduciendo la complejidad pero no siempre puede ir a la misma velocidad que otras iniciativas.
- Aunque muchas tareas complejas o repetitivas pueden simplificarse por medio de herramientas todavía deben ser llevadas a cabo
- El manual de Java EE de la última versión tiene 980 páginas

Sin embargo

- No debemos olvidar que cuando usamos un Framework es porque hemos considerado que cubre de la mejor manera posible nuestra problemática.
- Java EE no es más que una respuesta a un conjunto de problemas. Es nuestra decisión utilizarlo o no.