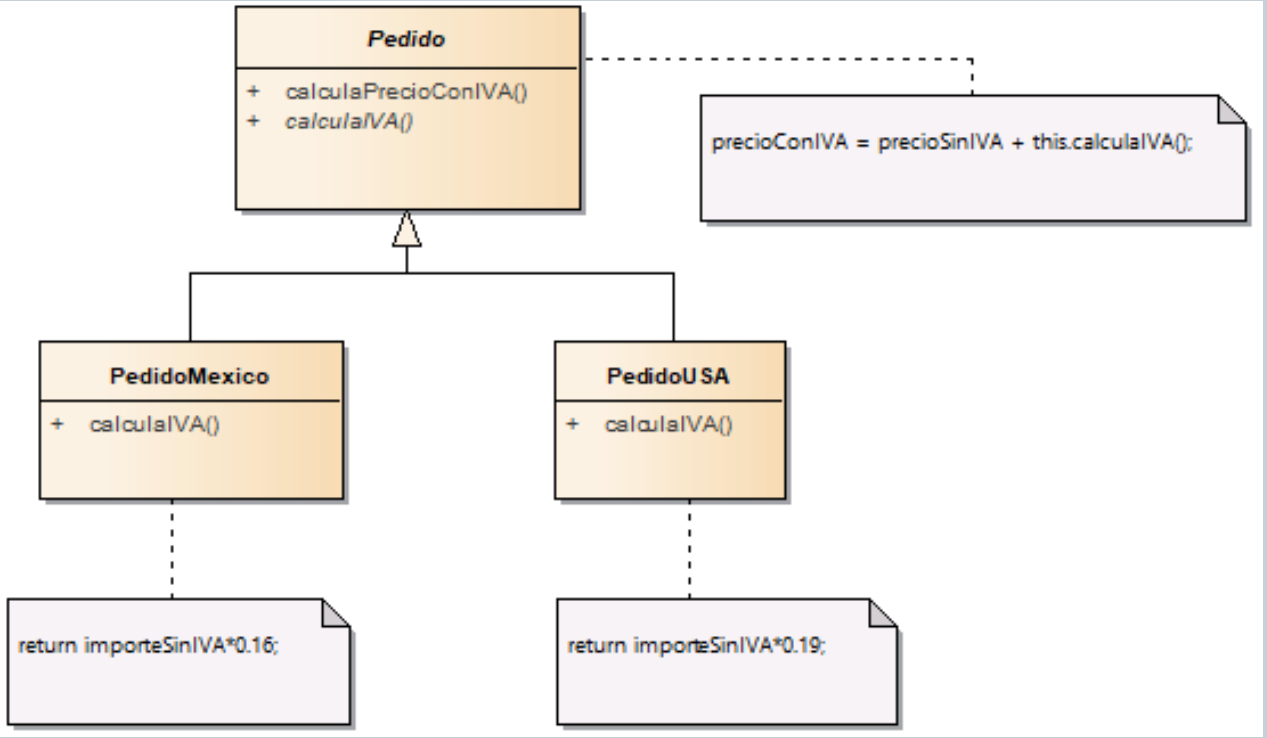
Una introducción a los patrones de diseño con Java

ABRAHAM SÁNCHEZ LÓPEZ
GRUPO MOVIS
FCC-BUAP

Patrones de diseño, I

- Un patrón de diseño (design pattern) consiste en un diagrama de objetos que forma una solución a un problema conocido y frecuente.
- El diagrama de objetos está constituido por un conjunto de objetos descritos por clases y las relaciones que vinculan los objetos.
- Los patrones responden a problemas de diseño de software (aplicaciones) en el marco de la programación orientada a objetos.
- Se trata de soluciones conocidas y probadas cuyo diseño proviene de la experiencia de los programadores.
- No existe un aspecto teórico en los patrones, en particular no existe una formalización (a diferencia de los algoritmos).
- Los patrones de diseño están basados en las buenas prácticas de la programación orientada a objetos.
- Por ejemplo, en la siguiente figura se muestra el patrón Template method que se describirá más adelante.

Patrones de diseño, II



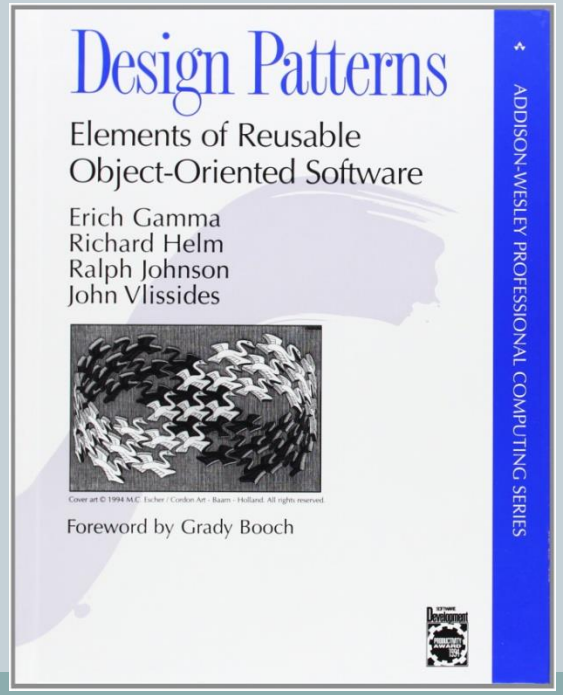
- En este patrón, el método calculaPrecioConIVA invoca al método calculaIVA que es abstracto en la clase Pedido. Este está defnido en las subclases Pedido, es decir las clases PedidoUSA y PedidoMexico.

Patrones de diseño, III

- En efecto, el IVA varía en función del país.
- Al método calculaPrecioConIVA se le llama “modelo” (template method). Este introduce un algoritmo basado en un método abstracto.
- Este patrón está basado en el polimorfismo, una propiedad importante de la programación orientada a objetos.
- El precio de un pedido en USA o en México está sometido a un impuesto sobre el valor agregado.
- No obstante el impuesto no es el mismo, y el cálculo del IVA difiere en México y USA.
- Por consiguiente, el patrón Template method constituye una buena ilustración del polimorfismo.

Un poco de historia

- Los patrones se introducen en 1995 con el libro del llamado “GoF” (Gang of Four), en referencia a la “banda de los cuatro” autores (llamado Design Patterns – Elements of Reusable Object-Oriented Software, escrito por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides).
- Este libro constituye la obra de referencia acerca de los patrones de diseño.



Descripción de los patrones de diseño, I

- Hemos decidido describir los patrones de diseño con ayuda de los siguientes lenguajes:
  - El lenguaje de modelización UML introducido por el OMG.
  - El lenguaje de programación creado inicialmente por Sun y ahora mantenido por Oracle:
    - Java
- Es común en los diferentes lenguajes presentarlos de la siguiente forma: los patrones de diseño, los patrones de construcción, patrones de comportamiento.
- Para cada patrón, se presentan los siguientes elementos:
  - El nombre del patrón.
  - La descripción del patrón.
  - Un ejemplo que describa el problema y la solución basada en el patrón descrito mediante un diagrama de clases UML. En este diagrama, se describe el cuerpo de los métodos utilizando notas.
  - La estructura genérica del patrón sería:

Descripción de los patrones de diseño, II

- Su esquema, extraído de cualquier contexto particular, bajo la forma de un diagrama de clases.
- La lista de participantes del patrón.
- Las colaboraciones en el patrón.
- Los dominios de la aplicación del patrón.
- Un ejemplo, presentado esta vez bajo la forma de un programa Java completo y documentado. Este programa no utiliza una interfaz gráfica sino exclusivamente las entradas/salidas por pantalla y teclado.

Catalogo de patrones de diseño, I

- En estas notas, se presentan los veintitrés patrones de diseño descritos en el libro de referencia del “GoF”.
- Estos patrones son diversas respuestas a problemas conocidos de la programación orientada a objetos. La lista que se presenta a continuación no es exhaustiva y es resultado, como se ha explicado, de la experiencia.
- **Abstract Factory**. Tiene como objetivo la creación de objetos reagrupados en familias sin tener que conocer las clases concretas destinadas a la creación de estos objetos.
- **Builder**. Permite separar la construcción de objetos complejos de su implementación de modo que un cliente pueda crear estos objetos complejos con diferentes implementaciones.
- **Factory Method**. Tiene como objetivo presentar un método abstracto para la creación de un objeto, reportando a las subclases concretas la creación efectiva.
- **Prototype**. Permite crear nuevos objetos por duplicación de objetos existentes llamados prototipos que disponen de la capacidad de clonación.

Catalogo de patrones de diseño, II

- **Singleton**. Permite asegurar que de una clase concreta, existe una única instancia y proporciona un método único que la devuelve.
- **Adapter**. Tiene como objetivo convertir la interfaz de una clase existente en la interfaz esperada por los clientes también existentes para que puedan trabajar de forma conjunta.
- **Bridge**. Tiene como objetivo separar los aspectos conceptuales de una jerarquía de clases de su implementación.
- **Composite**. Proporciona un marco de diseño de una composición de objetos con una profundidad de composición variable, basándose en el diseño de un árbol.
- **Decorator**. Permite agregar dinámicamente funcionalidades suplementarias a un objeto.
- **Facade**. Tiene como objetivo reagrupar las interfaces de un conjunto de objetos en una interfaz unificada que resulte más fácil de utilizar.
- **Flyweight**. Facilita la compartición de un conjunto importante de objetos con granularidad fina.
- **Proxy**. Construye un objeto que se substituye por otro objeto y que controla su acceso.

Catalogo de patrones de diseño, III

- **Chain of responsability**. Crea una cadena de objetos tal que si un objeto de la cadena no puede responder a una petición, la puede transmitir a sus sucesores hasta que uno de ellos responda.
- **Command**. Tiene como objetivo transformar una consulta en un objeto, facilitando operaciones como la anulación, la actualización de consultas y su seguimiento.
- **Interpreter**. Proporciona un marco para dar una representación mediante objetos de la gramática de un lenguaje con el objetivo de evaluar, interpretándolas, expresiones escritas en este lenguaje.
- **Iterator**. Proporciona un acceso secuencial a una colección de objetos sin que los clientes se preocupen de la implementación de esta colección.
- **Mediator**. Construye un objeto cuya vocación es la gestión y el control de las interacciones en el seno de un conjunto de objetos, sin que estos elementos se conozcan mutuamente.
- **Memento**. Salvaguarda y restaura el estado de un objeto.

Catalogo de patrones de diseño, IV

- **Observer**. Construye una dependencia entre un sujeto y sus observadores de modo que cada modificación del sujeto sea notificada a los observadores para que puedan actualizar su estado.
- **State**. Permite a un objeto adaptar su comportamiento en función de su estado interno.
- **Strategy**. Adapta el comportamiento y los algoritmos de un objeto en función de una necesidad concreta sin por ello cargar las interacciones con los clientes de este objeto.
- **Template Method**. Permite reportar en las subclases ciertas etapas de una de las operaciones de un objeto, estando éstas descritas en las subclases.
- **Visitor**. Construye una operación a realizar en los elementos de un conjunto de objetos. Es posible agregar nuevas operaciones sin modificar las clases de estos objetos.

Elección y uso de un patrón de diseño, I

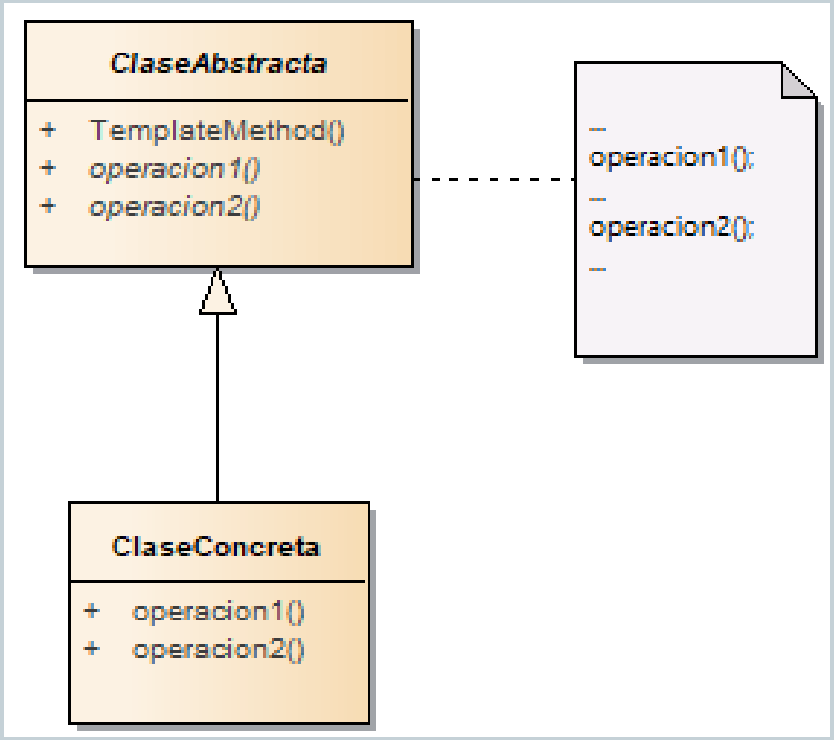
- Para saber si existe un patrón de diseño que responde a un problema concreto, la primera etapa consiste en ver las descripciones de la sección anterior y determinar si existe uno o varios patrones cuya descripción se acerque a la del problema.
- A continuación, conviene estudiar con detalle el o los patrones descubiertos a partir de su descripción completa que se encuentra en las siguientes secciones.
- En particular, conviene estudiar a partir del ejemplo que se proporciona y de la estructura genérica si el patrón responde de forma pertinente al problema.
- Este estudio debe incluir principalmente la posibilidad de adaptar la estructura genérica y, de hecho, averiguar si el patrón una vez adaptado responde al problema.
- Esta etapa de adaptación es una etapa importante del uso del patrón para resolver un problema. La describiremos a continuación.
- Una vez elegido el patrón, su uso en una aplicación comprende las siguientes etapas:
  - Estudiar profundamente su estructura genérica, que sirve como base para utilizar un patrón.

Elección y uso de un patrón de diseño, II

- Renombrar las clases y los métodos introducidos en la estructura genérica. En efecto, en la estructura genérica de un patrón, el nombre de las clases y de los métodos es abstracto. Y al revés, una vez integrados en una aplicación, estas clases y métodos deben nombrarse de acuerdo a los objetos que describen y a las operaciones que realizan respectivamente. Esta etapa supone el trabajo mínimo esencial para poder utilizar un patrón.
- Adaptar la estructura genérica para responder a las restricciones de la aplicación, lo cual puede implicar cambios en el diagrama de objetos.

- Veamos a continuación un ejemplo de adaptación del patrón Template method a partir del ejemplo presentado anteriormente.
- La estructura genérica de este patrón se muestra en la siguiente figura.

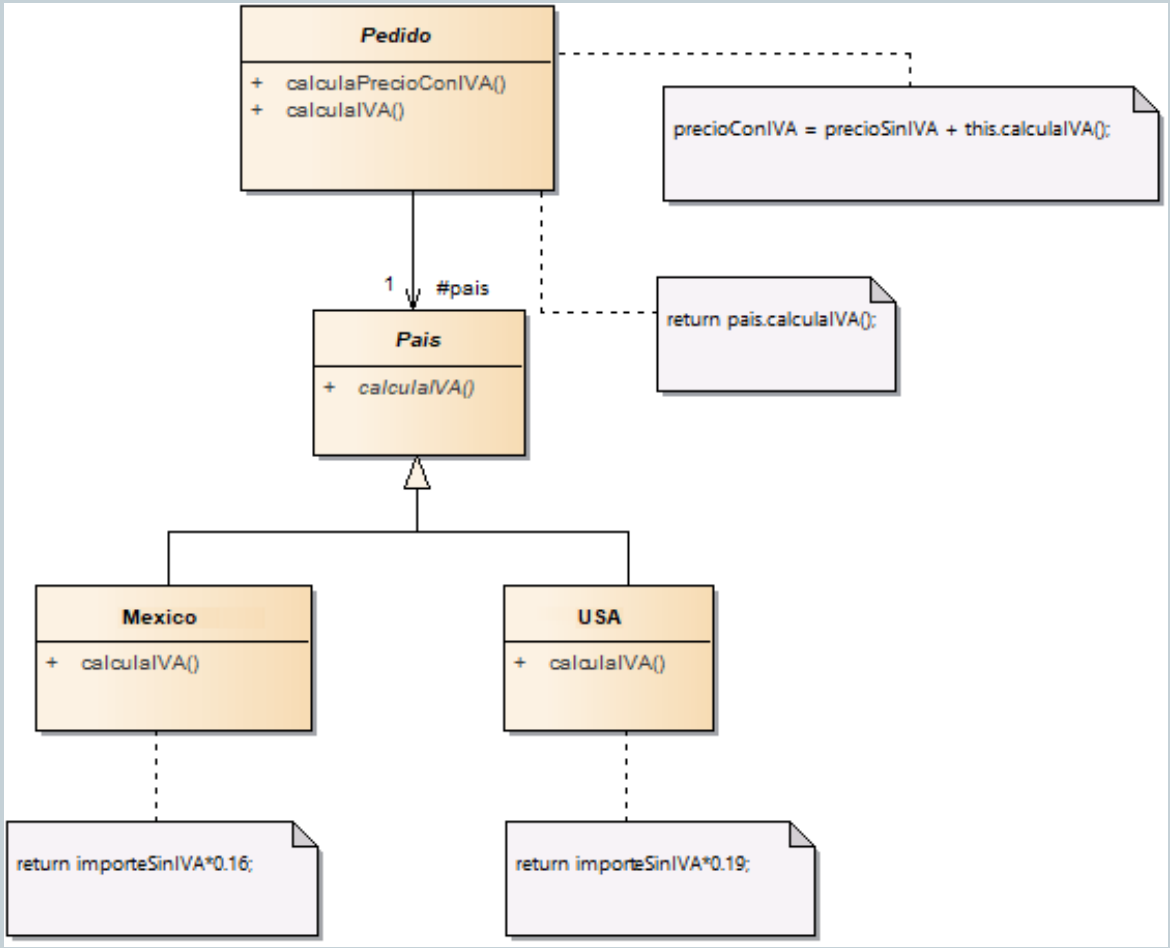
Estructura genérica del patrón Template Method



Elección y uso de un patrón de diseño, III

- Queremos adaptar esta estructura en el marco de una aplicación comercial donde el método de cálculo del IVA no esté incluido en la clase Pedido sino en las subclases concretas de la clase abstracta País.
- Estas subclases contienen todos los métodos de cálculo de los impuestos específicos de cada país.
- El método calculaIVA de la clase Pedido invocará a continuación al método calcularIVA de la subclase del país afectado mediante una instancia de esta subclase.
- Esta instancia puede pasarse como parámetro en la creación del pedido.
- La siguiente figura ilustra el patrón adaptado listo para su uso en la aplicación.

Ejemplo de uso con adaptación



Organización del catalogo de patrones

- Para organizar el catálogo de patrones de diseño, retomamos la clasificación del "GoF" que organiza los patrones según su vocación: construcción, estructuración y comportamiento.
- Los patrones de construcción tienen como objetivo organizar la creación de objetos. Los patrones de construcción son cinco: Abstract, Factory, Builder, Factory Method, Prototype y Singleton.
- Los patrones de estructuración facilitan la organización de la jerarquía de clases y de sus relaciones. Los patrones de estructuración son siete: Adapter, Bridge, Composite, Decorator, Facade, Flyweight y Proxy.
- Finalmente, los patrones de comportamiento proporcionan soluciones para organizar las interacciones y para repartir el procesamiento entre los objetos.
- Los patrones de comportamiento son 11: Chain of responsability, Command, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method y Visitor.

Caso de estudio, I

- En esta parte del curso, tomaremos un ejemplo de diseño de un sistema para ilustrar el uso de los 23 patrones de diseño.
- El sistema que vamos a diseñar es un sitio web de venta online de vehículos como por ejemplo, automóviles o motocicletas.
- Este sistema permite distintas operaciones como la visualización de un catálogo, la recolección de un pedido, la gestión y el seguimiento de los clientes.
- Además, estará accesible bajo la forma de un servicio web.

Descripción del sistema

- El sitio permite visualizar un catálogo de vehículos puestos a la venta, realizar búsquedas en el catalogo, realizar el pedido de un vehículo, seleccionar las opciones para el mismo mediante un sistema de carrito de la compra virtual.
- Las opciones incompatibles también deben estar gestionadas (por ejemplo “asientos deportivos” y “asientos de cuero” son opciones incompatibles).
- También es posible volver a un estado anterior del carrito de compra.

Caso de estudio, II

- El sistema debe administrar los pedidos. Debe ser capaz de calcular los impuestos en función de país de entrega del vehículo.
- También debe gestionar los pedidos pagados al contado y aquellos que están ligados a una solicitud de crédito. Para ello, se tendrá en cuenta las peticiones de crédito.
- El sistema administra los estados del pedido: en curso, validado y entregado.
- Al realizar el pedido de un vehículo, el sistema elabora el conjunto de documentos necesarios como la solicitud de placa de matriculación, el certificado de cesión y la orden de pedido.
- Estos documentos estarán disponibles en formato PDF o en formato HTML.
- El sistema también permite rebajar los vehículos de difícil venta, como por ejemplo, aquellos que se encuentran en stock, ya pasado algún tiempo en bodega.
- También permite realizar una gestión de los clientes, en particular de empresas que poseen filiales para proporcionarles, por ejemplo, la compra de una flota de vehículos.

Caso de estudio, III

- Después de la virtualización del catálogo, es posible visualizar animaciones asociadas a un vehículo. El catálogo puede presentarse con uno o tres vehículos por cada línea de resultados.
- La búsqueda en el catálogo puede realizarse con ayuda de palabras clave y de operadores lógicos (y, o).
- Es posible acceder al sistema mediante una interfaz web clásica o a través de una aplicación que utilice servicios web.

Caso de estudio, IV

Descripción de la parte del sistema	Patrón de diseño
Construir los objetos del dominio (auto de gasolina, auto diesel, auto eléctrico, etc.).	Abstract Factory
Construir el paquete de documentos necesarios en caso de comprar un vehículo.	Builder, Prototype
Crear los pedidos.	Factory Method
Crear el paquete en blanco de los documentos.	Singleton
Gestionar los documentos PDF.	Adapter
Implementar los formularios en HTML o mediante un applet.	Bridge
Representar las empresas clientes.	Composite
Visualizar los vehículos del catálogo.	Decorator, Observer, Strategy
Proporcionar la interfaz mediante servicios web del sitio web.	Facade

Caso de estudio, V

Descripción de la parte del sistema	Patrón de diseño
Administrar las opciones de un vehículo en un pedido.	Flyweight, Memento
Administrar la visualización de animaciones para cada vehículo del catálogo.	Proxy
Administrar la descripción de un vehículo.	Chain of responsability
Rebajar los vehículos en stock que han estado un periodo determinado de tiempo.	Command
Realizar búsquedas en la base de datos de vehículos mediante una búsqueda escrita en forma de expresión lógica.	Interpreter
Devolver secuencialmente los vehículos del catálogo.	Iterator
Gestionar el formulario de una solicitud de crédito.	Mediator
Gestionar los estados de un pedido.	State
Calcular el importe de un pedido.	Template Method
Enviar propuestas comerciales por correo electrónico a ciertas empresas clientes.	Visitor