



Patrones de Diseño

Ing. César Cappetto, Junio 2019
Dpto. de Ingeniería en Sistemas de Información
Cátedra Diseño de Sistemas



Agenda

- Definición, Principios SOLID, Importancia
- Clasificación de Patrones
- Patrones Creacionales: Singleton, Builder, Factory Method
- Patrones Estructurales: Adapter y Composite
- Patrones de Comportamiento: Strategy, State
- Ejercitación
- Conclusiones



Requisitos

Comprensión y conocimiento básico de los patrones.

Se recomienda haber leído previamente los patrones presentados en clase

Esta presentación NO es patrones desde 0, se asume un conocimiento mínimo.



Qué son los Patrones de Diseño ?

Son soluciones para problemas típicos y recurrentes que nos podemos encontrar en cualquier desarrollo de software. Describe cómo resolver un problema.

Son soluciones probadas, aceptadas y documentadas.



Porque son importantes?

- Nos ayudan a cumplir con los principios **SOLID**
- Nos ayudan a manejar el Acoplamiento y la Cohesión
- Maximizan la Reutilizacion deCodigo

Utilizar Patrones de Diseño contribuye con el objetivo de desarrollar aplicaciones robustas y fáciles de mantener



Qué tipos de patrones existen?

- **Creacionales:** Se aplican cuando el problema a solucionar es la creación de los objetos
- **Estructurales:** Se utilizan para crear objetos que están incluidos en estructuras más complejas
- **De Comportamiento:** Se aplican cuando el problema a resolver está relacionado con cómo se comportan los objetos y se relacionan entre sí

By Purpose				
		Creational	Structural	Behavioral
By Scope	Class	<ul style="list-style-type: none"> Factory Method 	<ul style="list-style-type: none"> Adapter (class) 	<ul style="list-style-type: none"> Interpreter Template Method
	Object	<ul style="list-style-type: none"> Abstract Factory Builder Prototype Singleton 	<ul style="list-style-type: none"> Adapter (object) Bridge Composite Decorator Façade Flyweight Proxy 	<ul style="list-style-type: none"> Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor



Patrones Creacionales: Singleton

Busca garantizar que una clase solo sea instanciada una vez

Provee de un mecanismo para limitar el número de instancias de una clase, por lo tanto el mismo objeto es siempre compartido por distintas partes del código.



Singleton

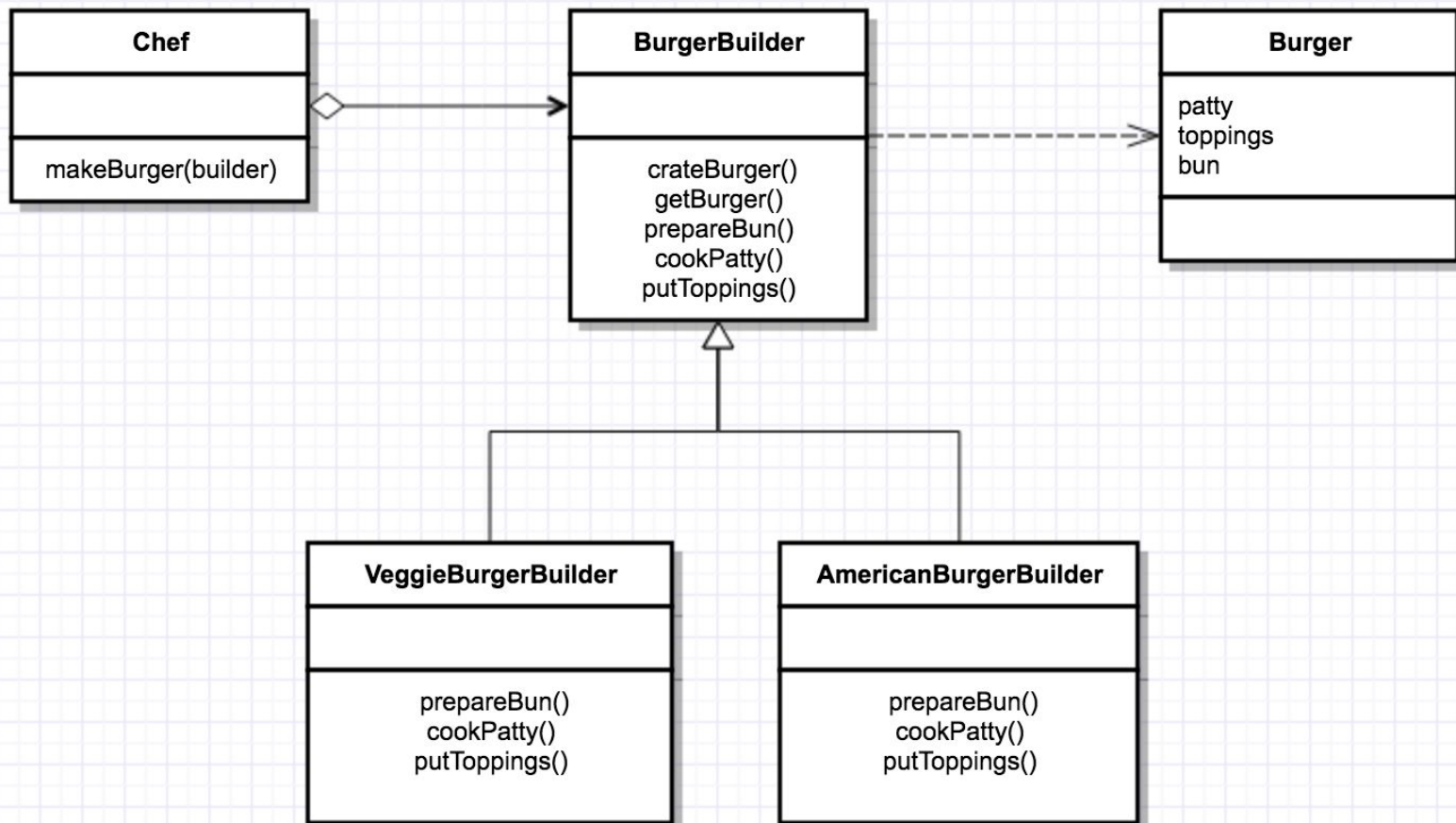
getInstance()
inc()
dec()
getCounter()



Patrones Creacionales: Builder

Permite la creación de un objeto complejo, a partir de una variedad de partes. El objeto final resulta del ensamblaje de dichas partes

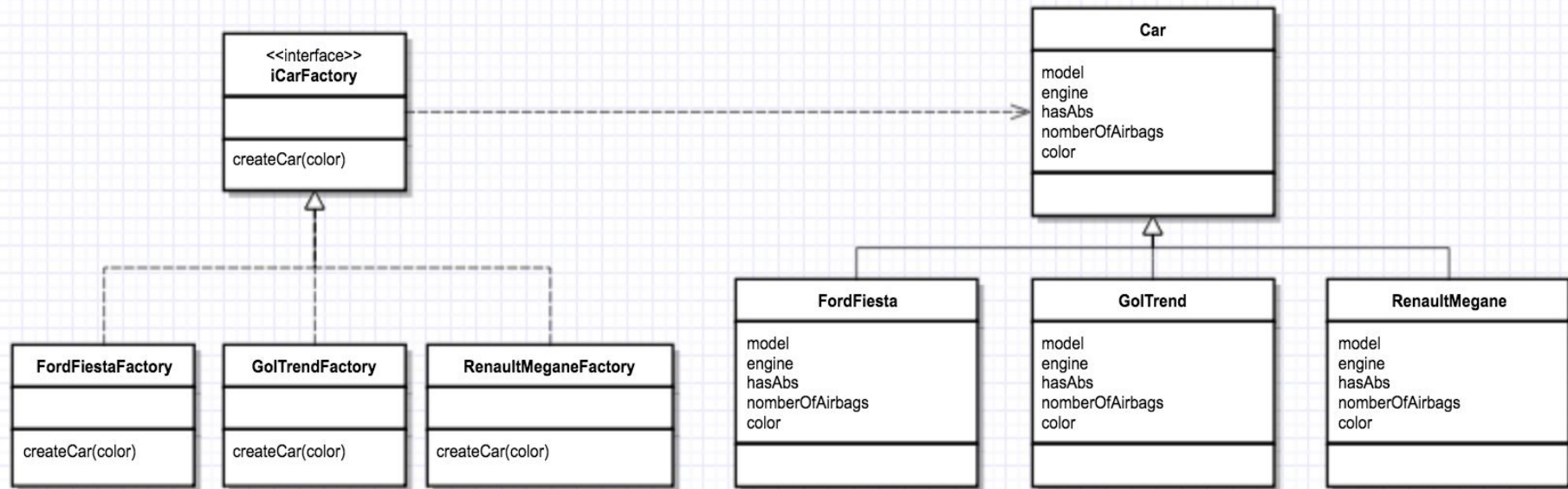
Centraliza el proceso de creación en un único punto, de tal forma que el mismo proceso de construcción pueda crear representaciones diferentes





Patrones Creacionales: Factory Method

Define una interfaz de creación de un cierto tipo de objeto y permite que las subclases decidan qué clase concreta necesitan instanciar.

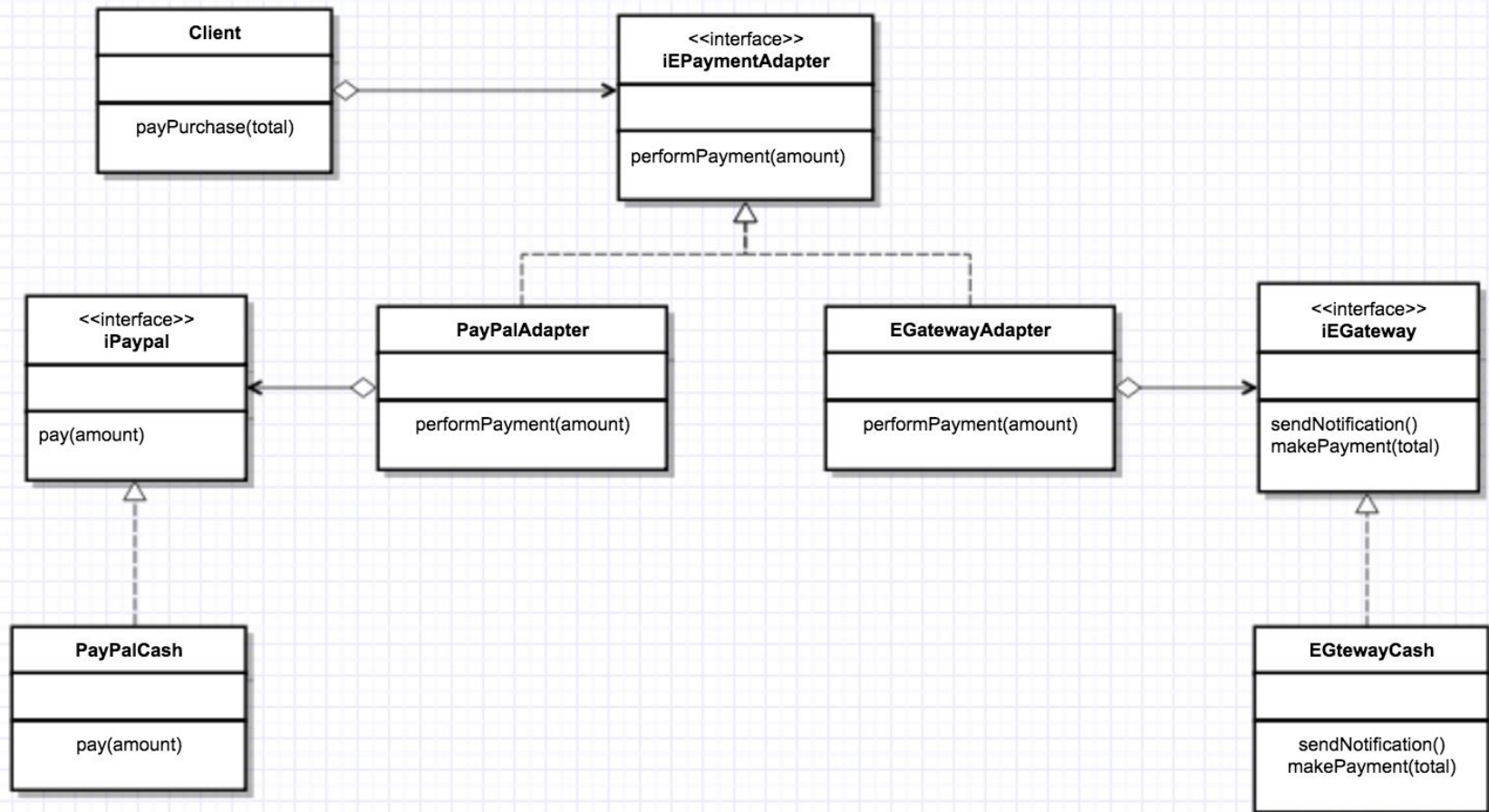




Patrones Estructurales: Adapter

Sirve para que dos interfaces diferentes puedan comunicarse

Se añade un adaptador intermedio que se encarga de realizar la conversión de una interface a otra

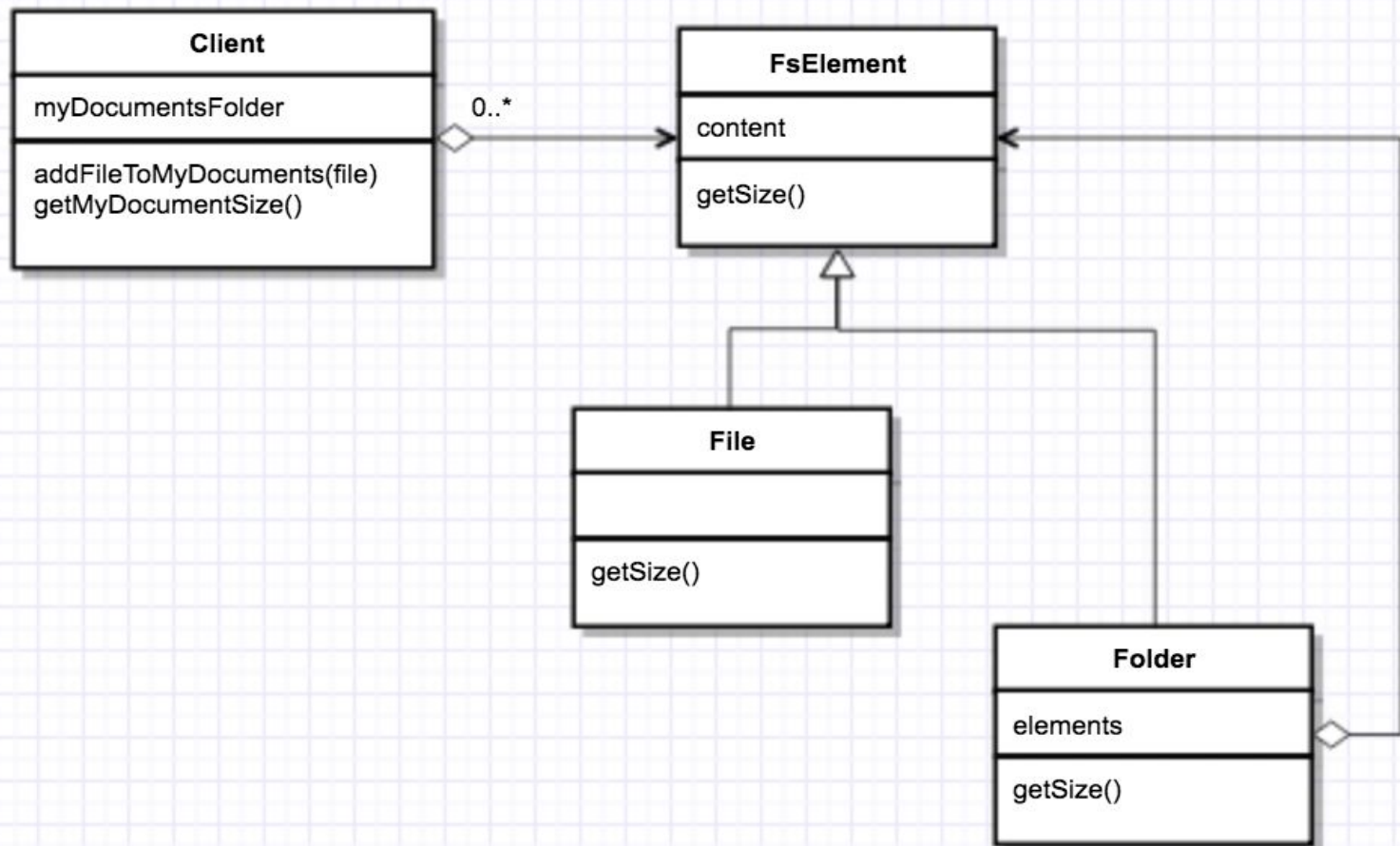




Patrones Estructurales: Composite

Compone objetos en árboles, para crear jerarquías del tipo *Todo-Parte*

Permite tratar objetos individuales y objetos compuestos de manera uniforme

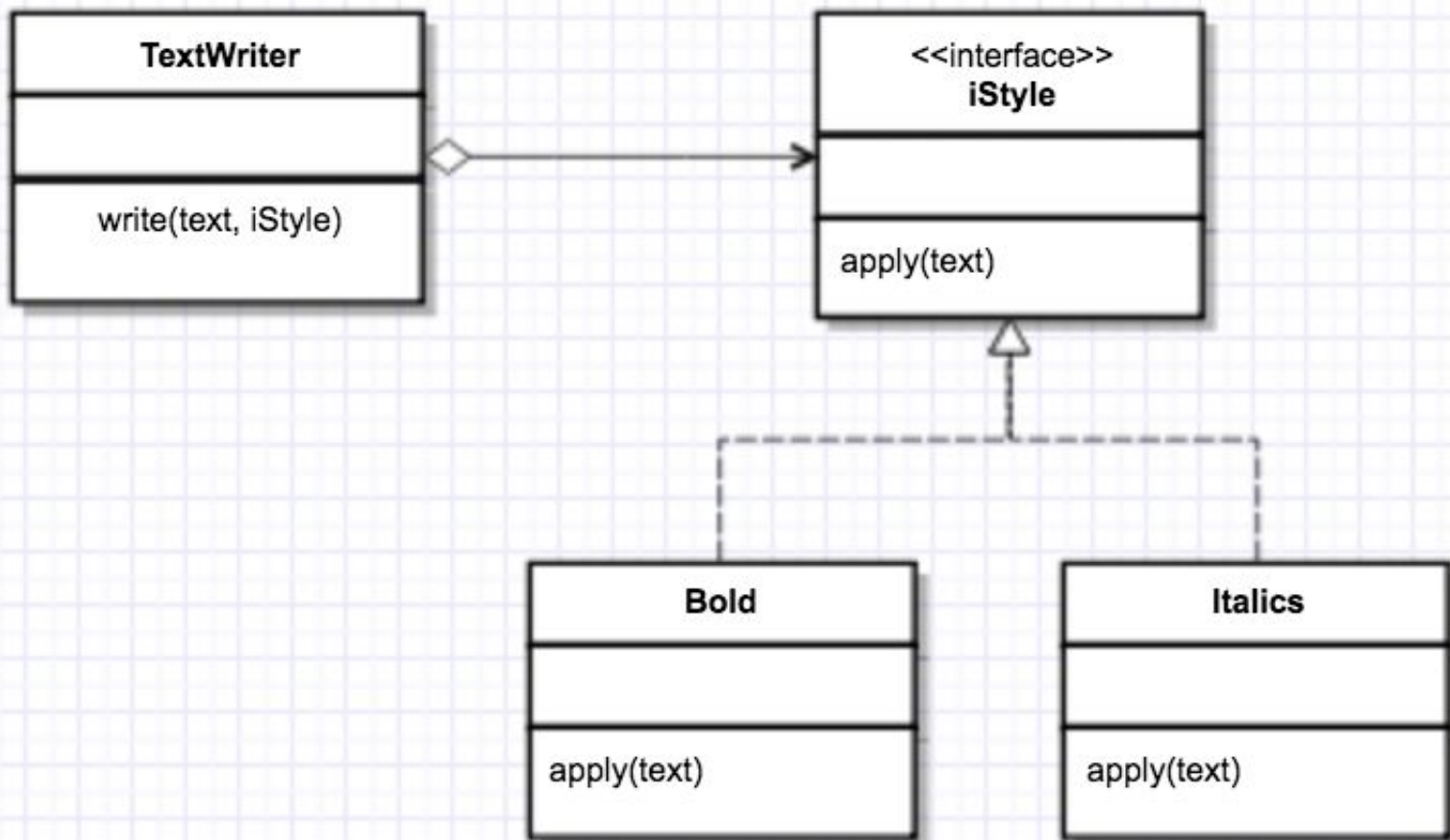




Patrones de Comportamiento: Strategy

Permite que un objeto pueda tener diferentes comportamientos de manera dinámica

Encapsula los distintos comportamientos en objetos y los hace intercambiables

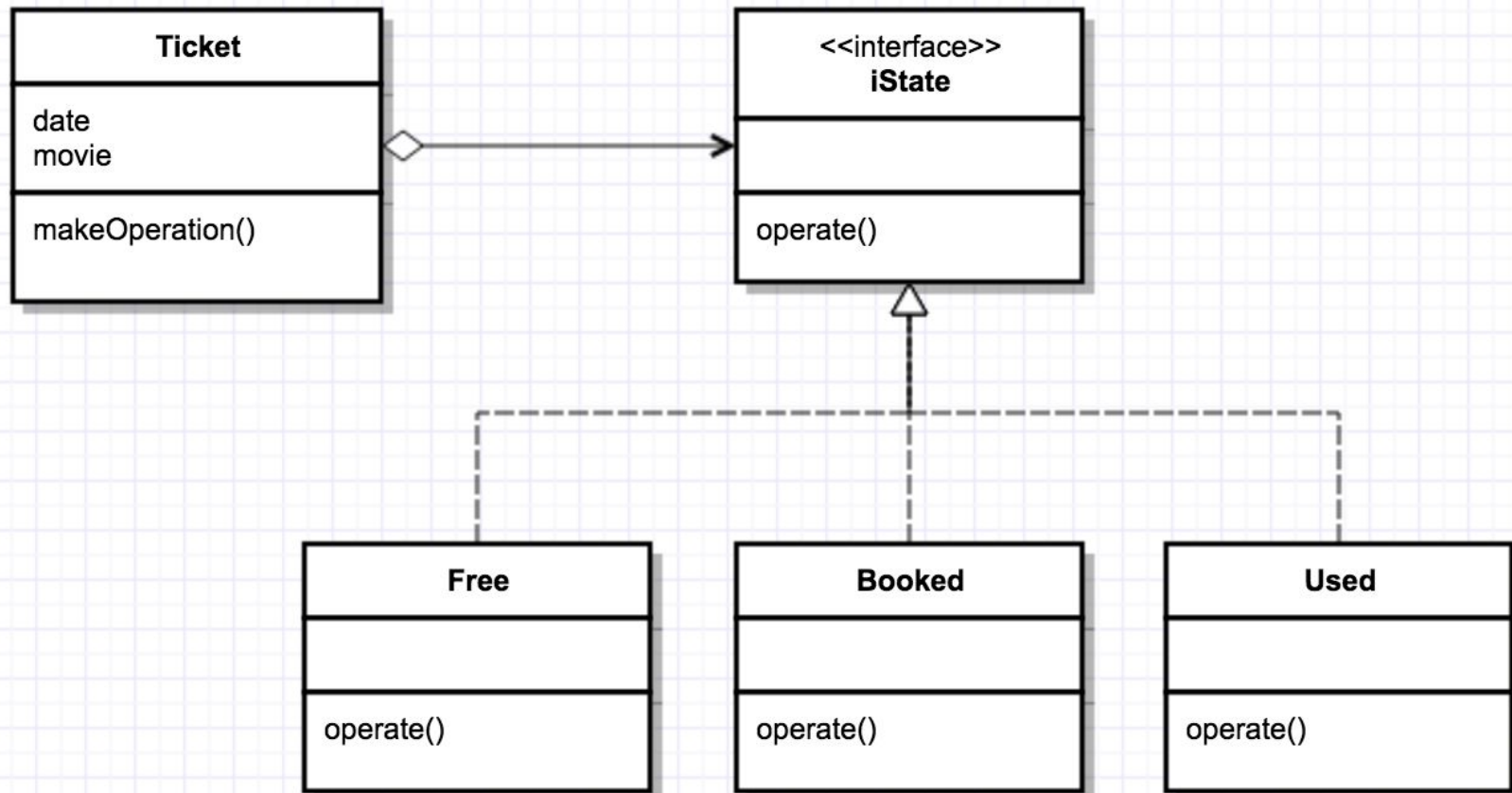





Patrones de Comportamiento: State

Permite que un objeto modifique su comportamiento cada vez que cambie su estado interno.

Busca que un objeto pueda reaccionar según su estado interno






Ejercicio 1: El camping “Bolso Grande” decidió ingresar al mundo 2.0, ofreciéndoles a sus clientes una app que les permite reservar su estadía de antemano, y sin tener que realizar un llamado para consultar la disponibilidad del camping. Sin embargo, una gran cantidad de clientes, realizan una reserva a la que luego no concurren, generándole al camping una pérdida en sus ingresos.

Debido a esta razón, el camping decidió implementar una regla de negocio que ajuste automáticamente sus tarifas en función de la demanda. Para eso se generó la siguiente tabla

- Reservas con “Baja Demanda”: 100 USD x noche
- Reservas con “Demanda Media”: 150 USD x noche
- Reservas con “Alta Demanda”: 250 USD x noche

Todos los sectores disponibles para reserva comienzan siendo de “Baja Demanda” pero a medida que incremente la misma, podrán pasar a ser de Demanda Media o Alta



Ejercicio 2: Una empresa de venta de zapatillas vende sus productos a través de su sitio que funciona como E-Commerce. Una vez finalizado el checkout, la nueva compra se registra en el sistema con la fecha del día, los datos del cliente y queda en “Pago Pendiente” esperando por la confirmación del pago que luego será ingresada manualmente por un administrador.

Una vez que la compra pasa a “Pago Aprobado”, esta lista para ser embalada, despachada y entregada al cliente. Todos los días, el sistema arroja un listado de todas las compras aprobadas que están listas para que se les asigne un producto y ser embaladas y despachada

Una vez que las compras fueron despachadas, son marcadas como “En Tránsito”, y finalmente como “Compra Finalizada”, cuando son entregadas al cliente



Conclusión

Los patrones contribuyen a la estandarización de la solución de diseño, haciendo que el mismo sea más comprensible para otros desarrolladores.

Son muy buenas herramientas, y como tales deberíamos aplicarlas, pero siempre con criterio, y no olvidar nunca, el hecho de que: Aplicar un patrón, no soluciona un mal diseño



¿Preguntas?



Información Extra

- Herramienta de Documentacion: <https://go.gliffy.com>
- Repositorio con ejemplos: <https://github.com>



iMuchas Gracias!