guate-jug

Patrones de Diseño en Java

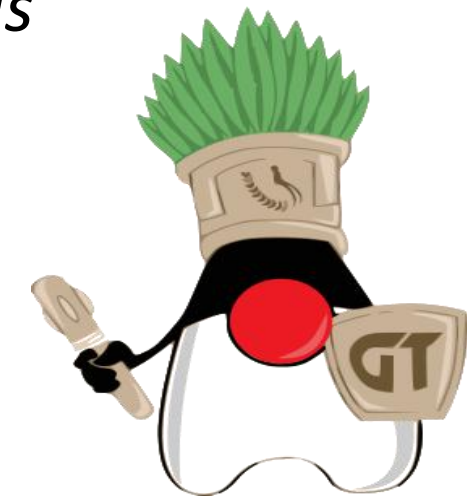
Mario Batres

Guatejug - 2020

Objetivos

- Reusabilidad en diseño
- Evitar búsqueda de soluciones a problemas ya conocidos.
- Estandarizar

No es obligatorio ni necesario utilizar patrones de diseño, solamente si el problema a resolver tiene características similares que se adecuan a un patrón en particular.



Beneficios

- Ya se encuentran definidos y proveen un enfoque estándar para resolver un problema recurrente.
- Promueve la reutilización que conduce a un código más robusto y altamente mantenible.
- Facilita el entendimiento y depuración del código fuente.
- Conduce a un desarrollo más rápido y los nuevos miembros del equipo lo entiende fácilmente.

Categorías (1)

- Creacionales
  - Proveen una solución para instanciar un objeto en la mejor manera posible para situaciones específicas.
    - Abstract Factory
    - Builder
    - Factory
    - Prototype
    - Singleton

Categorías (2)

- Estructurales.
  - Proveen diferentes formas de crear una estructura de clase, como utilizar la herencia y la composición para crear un objeto grande a partir de objetos pequeños.
    - Adapter
    - Bridge
    - Composite
    - Decorator
    - Facade
    - Flyweight
    - Proxy

Categorías (3)

- Conductuales
  - Proveen una solución para una mejor interacción entre los objetos y cómo proporcionar un acomplamiento flexible y tener la capacidad extenderse fácilmente.
    - Chain of Resposibility
    - Interpreter
    - Iterator
    - Mediator
    - Memento
    - Observer
    - State
    - Strategy
    - Template Method
    - Visitor

Facade Pattern

Facade Pattern (1)

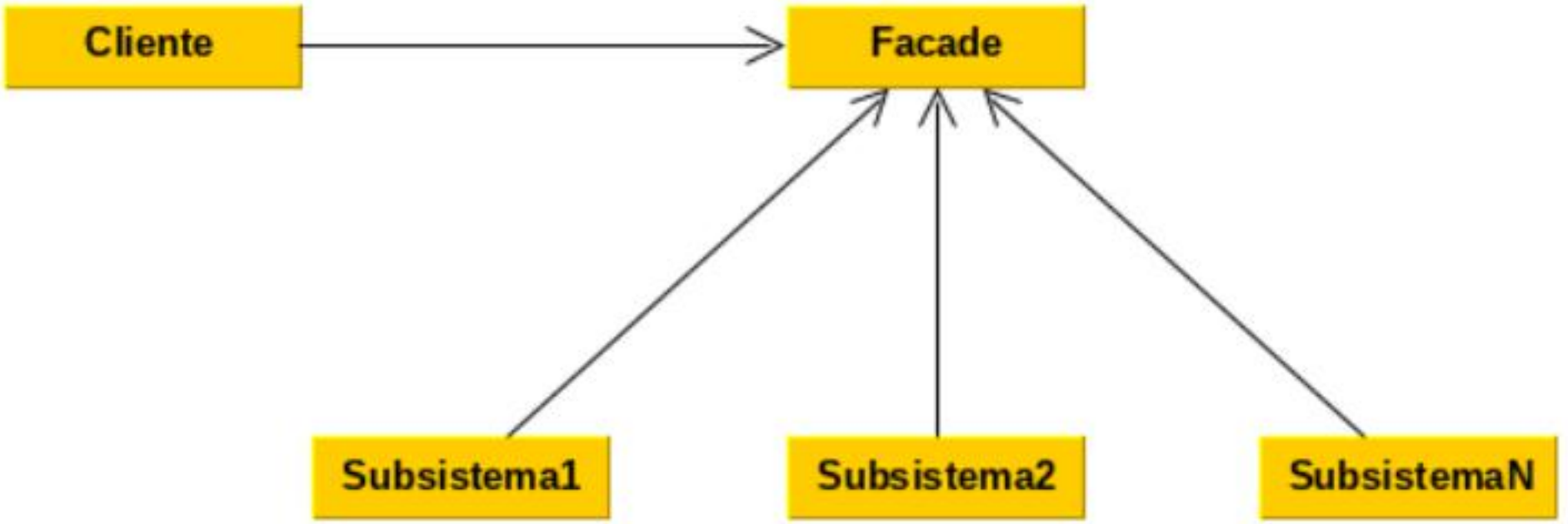
- Objetivos
  - Estructurar sistemas complejos o varios subsistemas.
  - Reducir la complejidad.
- Usabilidad
  - Cuando se necesita tener una interfaz limitada pero simple para un sistema complejo.
  - Cuando se quiere estructurar un subsistema dentro de capas.

Facade Pattern (2)

- Ventajas
  - Aislar el código de la complejidad de un subsistema.

- Desventajas
  - Un facade puede llegar a tener demasiada responsabilidad.

Facade Pattern (3)



Singleton Pattern

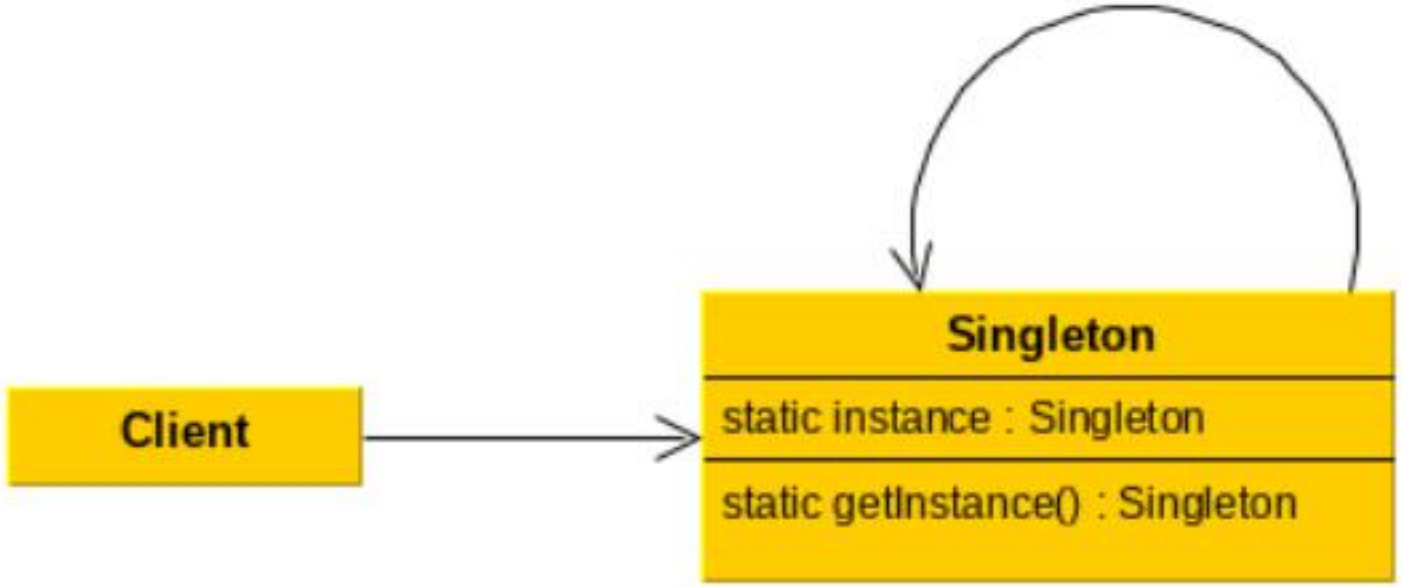
Singleton Pattern (1)

- Objetivos
  - Asegurar que la clase tiene una sola instancia.
  - Proporcionar un punto de acceso global a esa instancia.
- Usabilidad
  - Cuando una clase en un aplicación deba tener una sola instancia.
  - Cuando es necesario tener un control estricto sobre variables globales.

Singleton Pattern (2)

- Ventajas
  - Se puede asegurar que una clase tenga una sola instancia.
  - Acceso globala esa misma instancia.
  - Solo se inicializa cuando es solicitador por primera vez.
- Desventajas
  - Trato especial en entornos multihilos, múltiples hijos no creen objetos singleton varias veces.
  - Podría dificultar la creación y generación de pruebas unitarias.

Singleton Pattern (3)



Factory Method Pattern

Factory Method Pattern (1)

- Objetivos
  - Proporcionar una interface para crear objetos en una superclase.
  - Reemplazar las llamadas al constructor del objeto directamente.

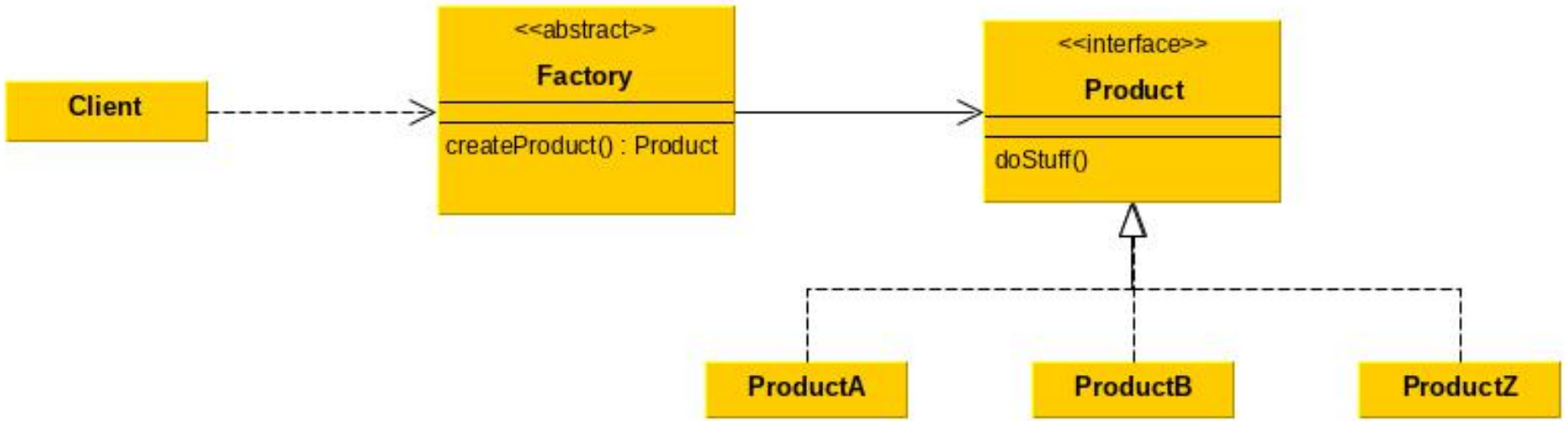
Factory Method Pattern (2)

- Usabilidad
  - Cuando no se conoce de antemano los tipos exactos y dependencias de los objetos.
  - Para ampliar los componentes internos.
  - Para ahorrar recursos del sistema reutilizando objetos y existentes en lugar de reconstruirlos cada vez.

Factory Method Pattern (3)

- Ventajas
  - Se evita el ***tight coupling*** entre el creador y los productos concretos.
  - Se pueden introducir nuevos tipos de productos dentro del programar sin afectar el código cliente existente.
- Desventajas
  - El código podría aumentar su complejidad si es necesario crear una nueva subclase que implementan el patrón.

Factory Method Pattern (4)



Decorator Pattern

Decorator Pattern (1)

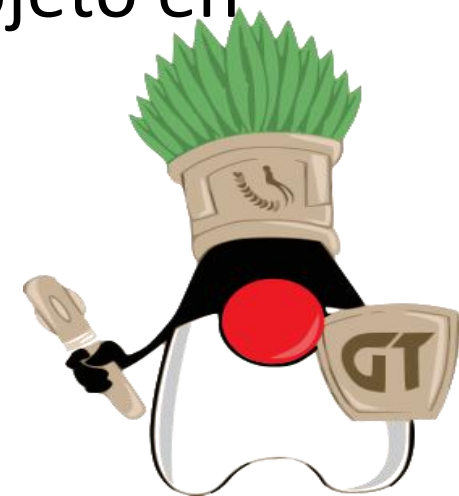
- Objetivos
  - Agregar nuevos comportamientos a los objetos al colocarlos dentro de objetos de envoltura especiales que contienen dichos comportamientos.
  - Agregar nuevas funcionalidades a un objeto existente sin alterar su estructura.

Decorator Pattern (2)

- Usabilidad
  - Cuando se necesite asignar comportamientos adicionales a los objetos en tiempo de ejecución sin remporte el código que usa estos objetos.
  - Cuando se incómodo o no sea posible extender el comportamiento de un objeto usando la herencia

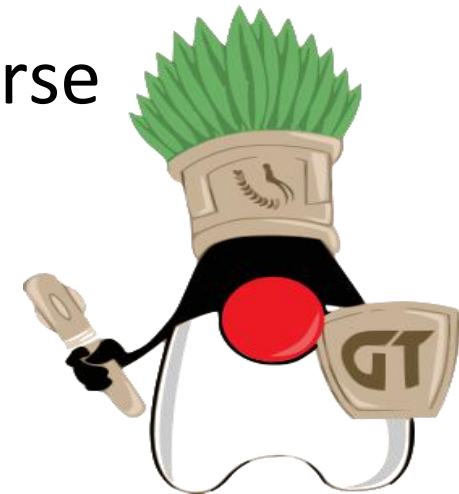
Decorator Pattern (3)

- Ventajas
  - Extiende el comportamiento de un objeto sin hacer un nueva subclase.
  - Agrega o remueve responsabilidades de un objeto en tiempo de ejecución.
  - Combina varios comportamientos envolviendo un objeto en múltiples decoradores.

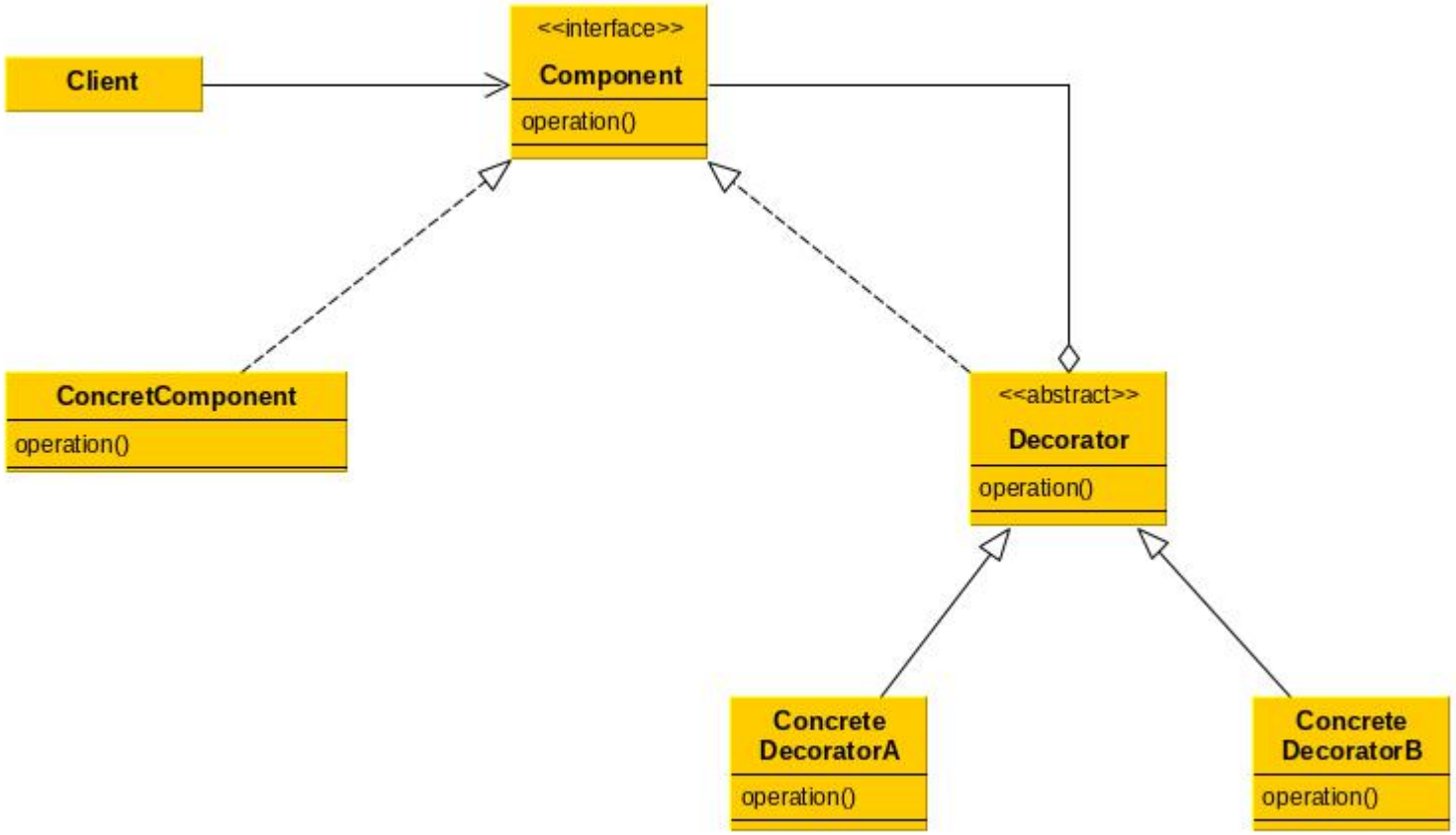


Decorator Pattern (4)

- Desventajas
  - Es complicado remover una envoltura específica de la pila de envolturas.
  - Es complicado implementar un decorador de tal manera que sus comportamiento no dependa sobre el orden en la pila de los decoradores.
  - La configuración inicial del código de capas podría verse bastante mal.



Decorator Pattern (5)



Muchas gracias

- Mario Batres
- @mariobatres7

guate-jug

LOGISTIKA360°

Your logistics, Our Passion