Patrones de diseño

UNJu - Programación Orientada a Objetos

Ing. José Zapana

¿Así programamos?

¿Y así lo documentamos?

```
//
// Querido programador:
//
// Cuando escribí este código, sólo Dios y yo
// sabíamos cómo funcionaba.
// Ahora, ¡sólo Dios lo sabe!
//
// Así que si está tratando de 'optimizar'
// esta rutina y fracasa (seguramente),
// por favor, incremente el siguiente contador
// como una advertencia
// para el siguiente colega:
//
// total_horas_perdidas_aquí = 189
//
```

Christopher Alexander

- Según Christopher Alexander, “cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, así como la solución a ese problema, de tal modo que se pueda aplicar esta solución un millón de veces, sin hacer lo mismo dos veces”

04/10/1936 - 17/03/2022

Clasificación

- Creacionales
  - Builder
  - Singleton
  - Abstract Factory, otros
- Estructurales
  - Decorator
  - DAO
  - Service Layer, otros
- Comportamiento
  - State
  - Strategy
  - otros

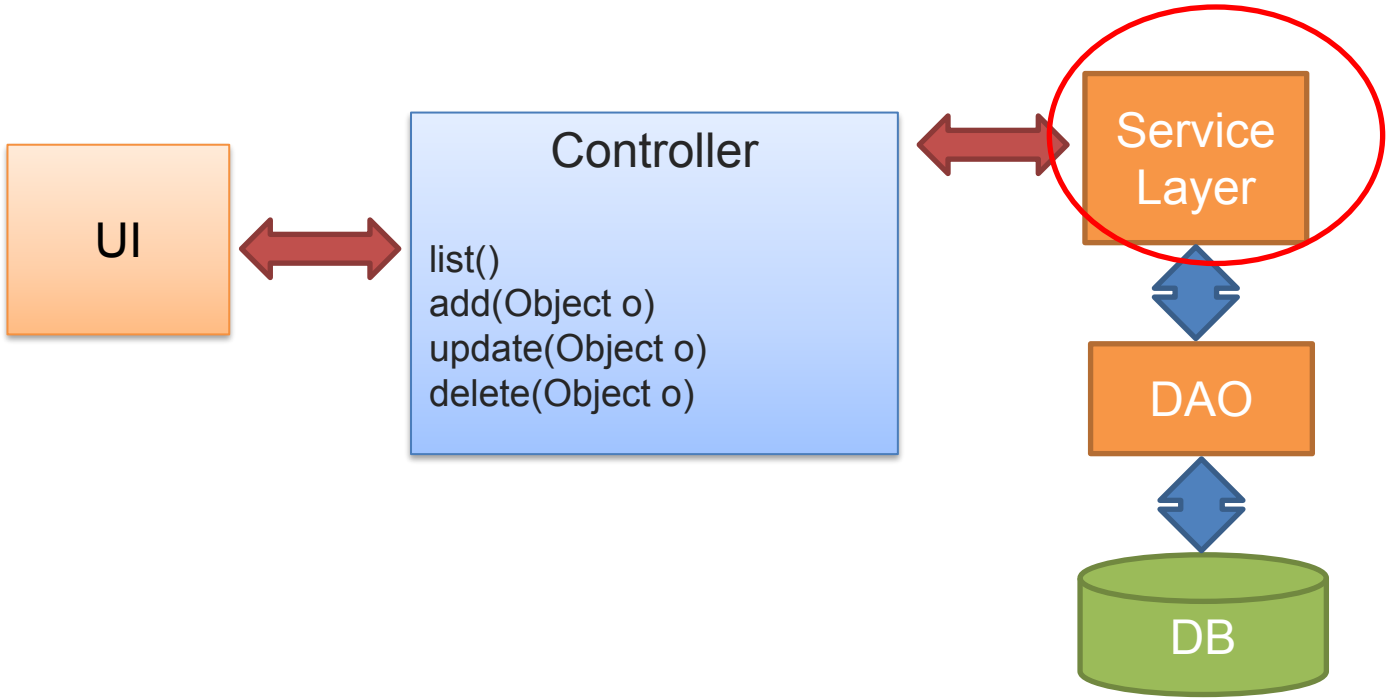
Partes de un patrón de diseño

En general, un patrón tiene 4 partes esenciales:

- **Nombre del patrón:** describir en 1 o 2 palabras, un problema de diseño junto con sus soluciones y consecuencias.
- **El problema:** describe cuándo aplicar el patrón.
- **La solución:** describe los elementos que constituyen el diseño, sus relaciones, responsabilidades y colaboraciones
- **Las consecuencias:** son los resultados así como las ventajas e inconvenientes de aplicar el patrón

Arquitectura con capa de Servicios

- Gestiona las peticiones del controlador



Consecuencias

- Separación de lógica de negocio
  - La lógica de negocios y las reglas se especifican en la capa de servicio que a su vez llama capa DAO, la capa DAO solo es responsable de interactuar con DB.
- Seguridad
  - Solo es posible acceder a la base de datos o a través del servicio.
- Bajo nivel de acoplamiento
  - Dado que un servicio puede contener varias operaciones de la base de datos y modificarlos puede ser imperceptible para quien consume el servicio

Ejemplo conceptual

- Implementación en Java

```
public class ProductServiceImpl implements ProductService{
    private ProductDaoImpl productDao;

    public Product save(Product p) {
        productDao.save(p);
        return p;
    }

}
```

Patrón de diseño Inyección de Dependencia (DI)

- En este patrón de diseño se suministran objetos a una clase en lugar de ser la propia clase la que cree dichos objetos.
- Esos objetos cumplen contratos que necesitan nuestras clases para poder funcionar
- Existe un contexto o contenedor que es el encargado de inyectar las implementaciones deseadas
- Analizado por primera vez por Martin Fowler

Motivación

- Alto nivel de acoplamiento entre componentes
- Aparecen nuevos conceptos en el diseño como la Inversión de Control (IoC) implementada a través de Inyección de Dependencias.

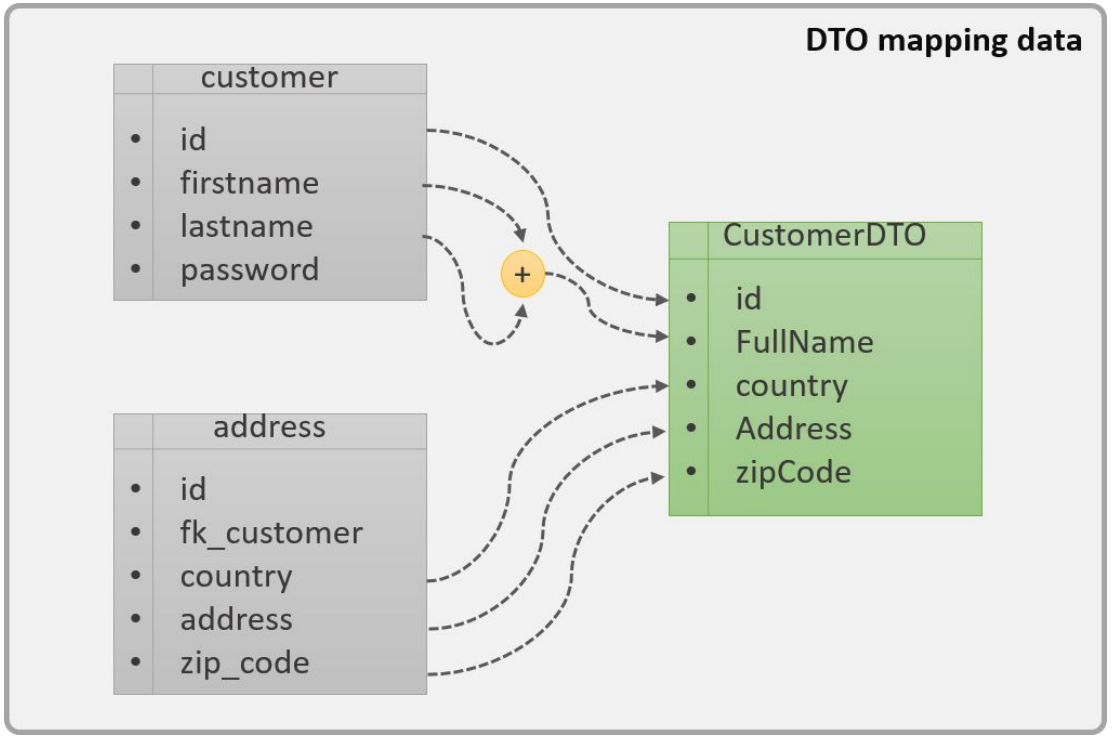
Implementación en java

- Contenedor (spring framework)
- Declaración de Inyecciones

Patrón de diseño DTO

- Concepto
  - Objeto de transferencia de datos
  - Define un objeto que transporta datos entre procesos
- Motivación
  - La comunicación entre procesos se realiza generalmente mediante interfaces remotas
  - Cada llamada es una operación costosa
  - Permite agregar datos de varias entidades
  - Protege atributos de las entidades

Diagrama representativo



Implementación en Java

- Modificación del servicio usando un DTO

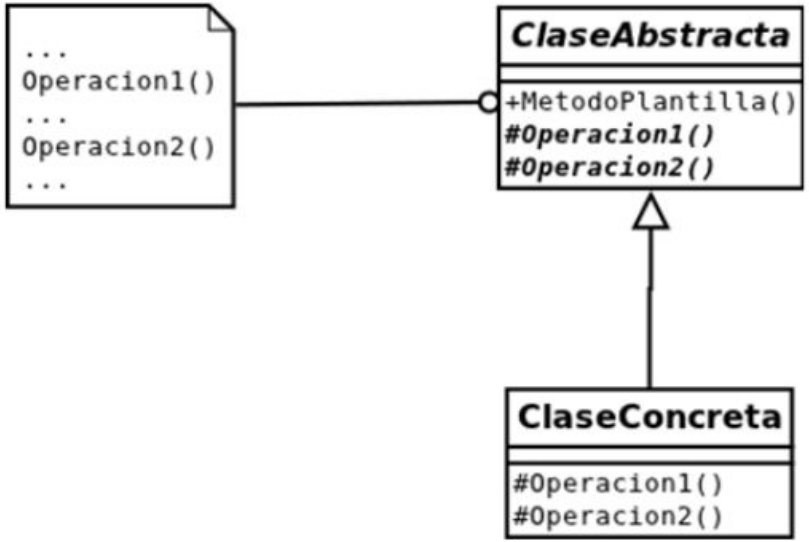
```java
@Service
public class ProductServiceImpl implements ProductService{
    private ModelMapper mapper = new ModelMapper();
    @Inject private ProductDao productDao;

    public ProductDTO save(ProductDTO p) {
        Product product = new Product();
        mapper.map(p, product);
        productDao.save(product);
        return p;
    }

}
```

Template Method

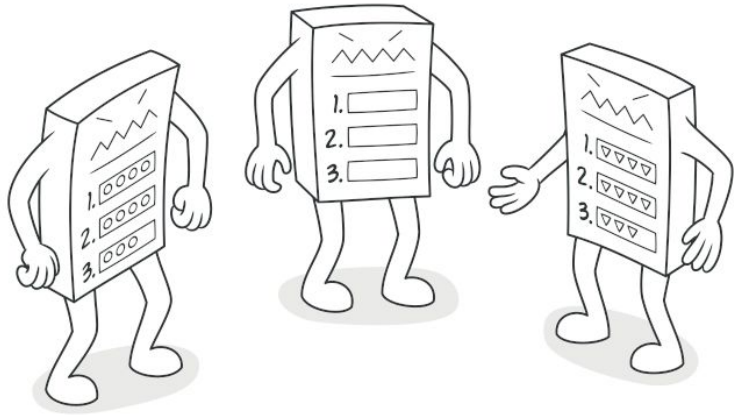
- Propósito
  - Es un patrón de diseño de comportamiento que define el esqueleto de un algoritmo en la superclase pero permite que las subclases sobrescriban pasos del algoritmo sin cambiar su estructura.
- Motivación
  - Reutilización de código común en la clase base.
  - Permitir la personalización de algunos pasos del proceso en subclases.
  - Mantener la consistencia en la estructura del algoritmo.

Estructura del patrón Template Method

- **Diagrama UML**



- **Representación**



Aplicabilidad

- Utiliza el patrón Template Method cuando quiera permitir a sus clientes que extiendan únicamente pasos particulares de un algoritmo, pero no todo el algoritmo o su estructura.
- Permite convertir un algoritmo monolítico en una serie de pasos individuales que se pueden extender fácilmente con subclases, manteniendo intacta la estructura definida en una superclase.

Pasos para implementarlo

1. Analizar el algoritmo objetivo para ver si puedes dividirlo en pasos. Considera qué pasos son comunes a todas las subclases y cuáles siempre serán únicos.
2. Crea la clase base abstracta y declara el método plantilla y un grupo de métodos abstractos que representen los pasos del algoritmo.
3. Los pasos pueden ser abstractos, sin embargo algunos pueden estar implementados.
4. Para cada variación del algoritmo, crea una nueva subclase concreta. Ésta debe implementar todos los pasos abstractos, pero también puede sobrescribir algunos de los opcionales.

Referencias

- Why to use Service Layer in Spring MVC,
- Patterns of Enterprise Application Architecture
- Inyección de dependencias
- Implementando Contextos Java e Inyección de Dependencia (CDI)
- Data Transfer Object (DTO)
- https://refactoring.guru/es/design-patterns/template-method