FACULTAD DE INGENIERIA
UNIVERSIDAD NACIONAL DE JUJUY

Programación Orientada a Objetos

San Salvador de Jujuy

UNJu – Facultad de Ingeniería
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
//
```

- 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

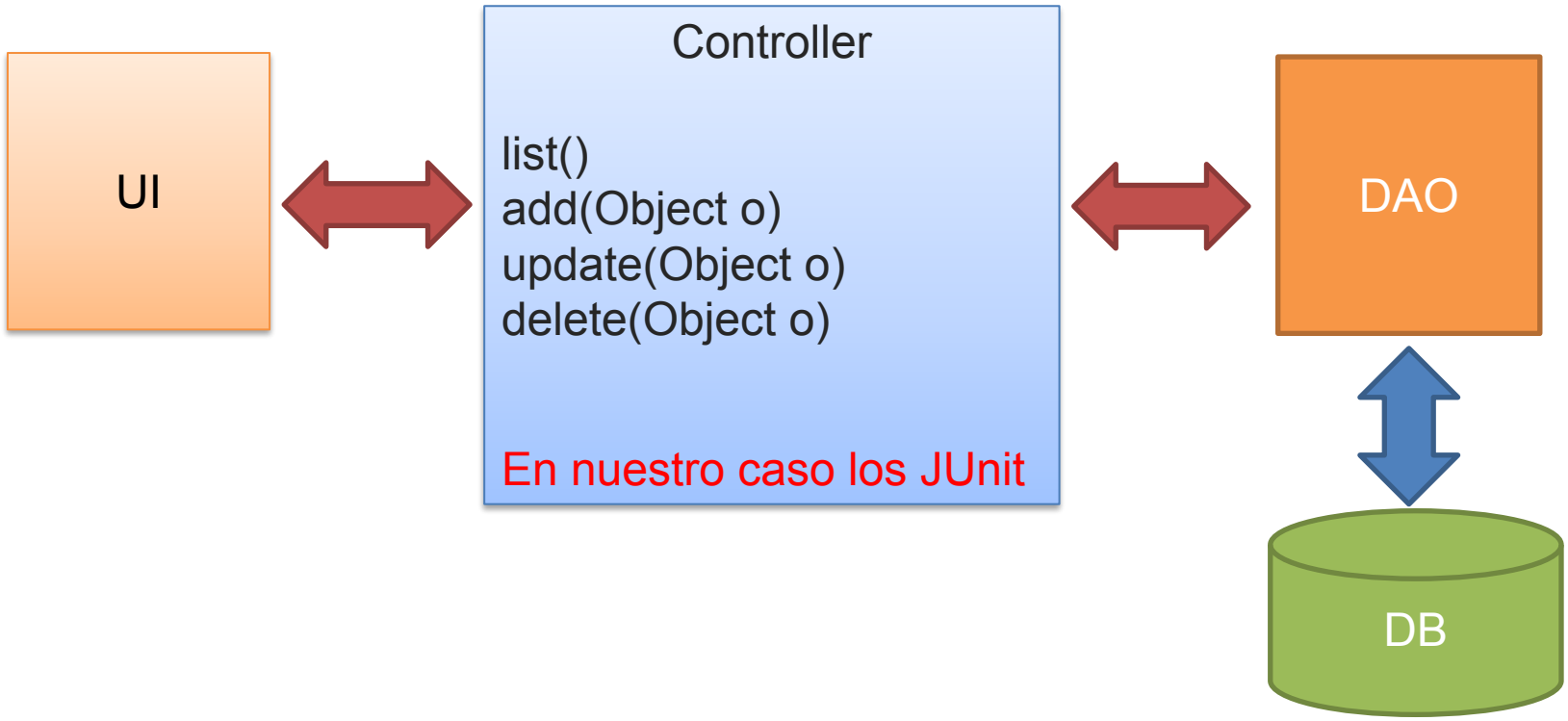
Partes de un patrón de diseño

En general, un patrón tiene 4 partes esenciales:

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

Arquitectura sin capa de Servicios

- Arquitectura de una aplicación sin la capa de Servicios



Consecuencias del modelo anterior

- El controlador accede directamente a la capa **DAO** para gestionar los datos de la base de datos
- Los datos recibidos se utilizan para representar la vista en el navegador.
- Gran riesgo porque se expone la conexión de base de datos a la clase controlador
- Para realizar una transacción de negocio hay que escribir todo el código en la clase de controlador, lo cual no es una buena práctica

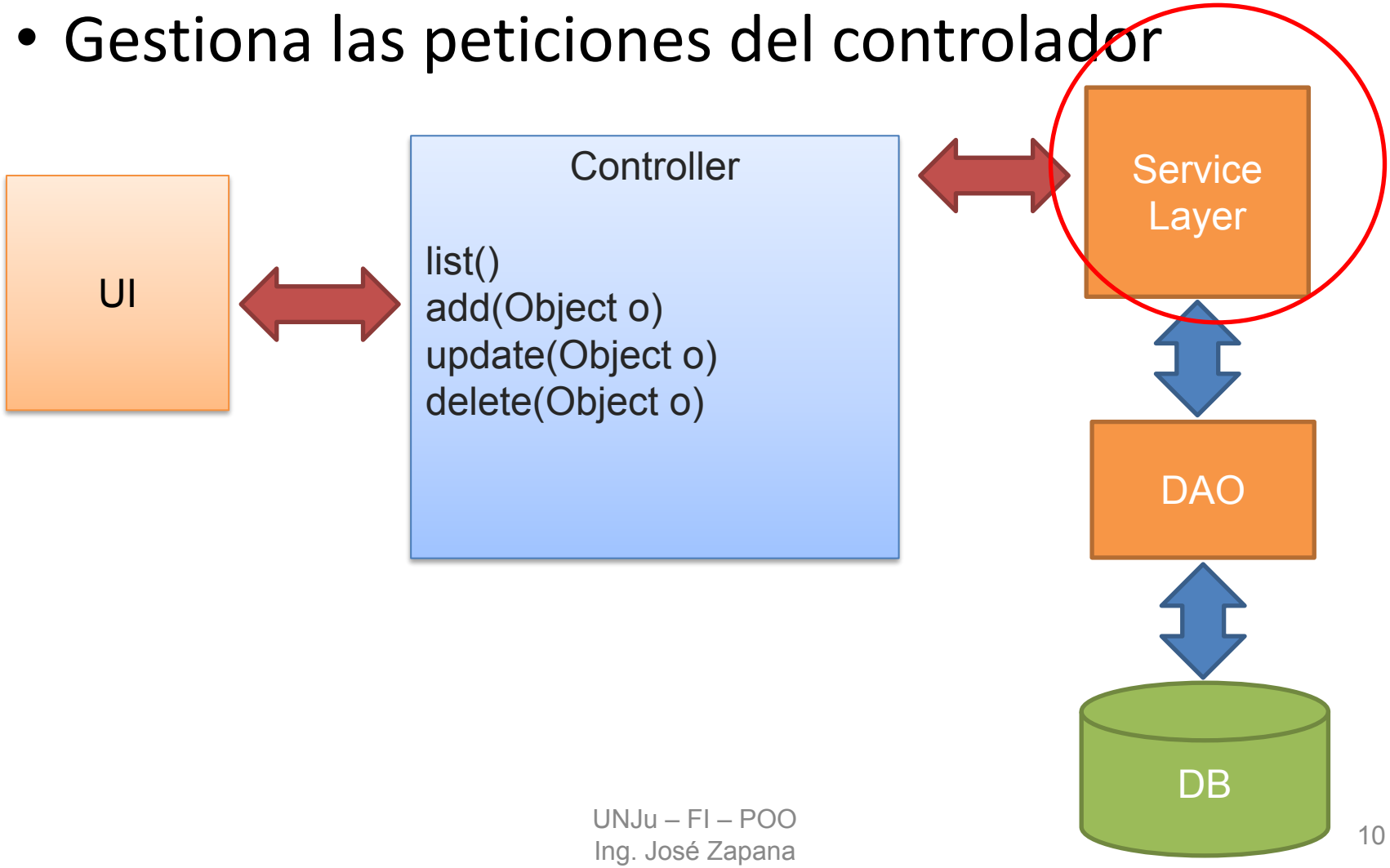
Service Layer

- Define el límite de una aplicación con una capa de servicios, que establece un conjunto de operaciones disponibles y coordina la respuesta de la aplicación en cada operación.

by Martin Fowler
(Patterns of Enterprise Application Architecture)

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.

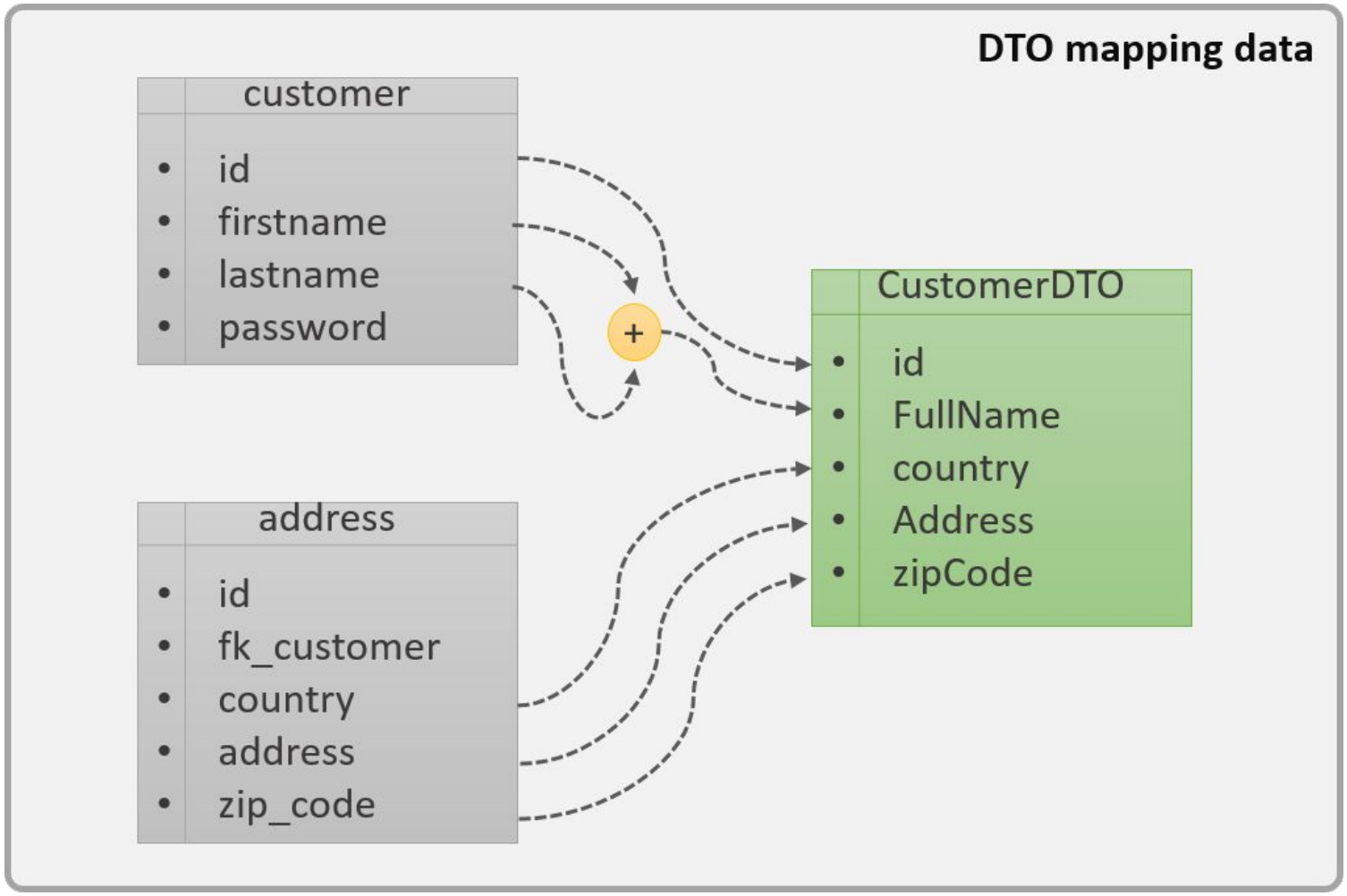
Implementación en Java

- Contenedor (spring framework)
  - Contenedor IoC que inyecta a cada objeto los objetos necesarios según las relaciones de dependencia registradas en la configuración previa.
- Declaración de Inyecciones
  - Normalmente se usan interfaces para bajar el nivel de acoplamiento.

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

```
@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;
    }

}
```

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)