Desarrollo de Aplicaciones Open Source

Profesor: *Rafael O. Castro Veramendi, Ph.D.*

Aplicación con Domain Driven Design

Profesor: *Rafael O. Castro Veramendi, Ph.D.*

¿Tienen consultas sobre la clase anterior?

Temario

Consideraciones para la creación de una aplicación con el approach Domain-Driven Design.

Utilidad del Tema

¿Por qué es importante el approach domain-driven design en la creación de aplicaciones?

El enfoque Domain-Driven Design (DDD) es importante en la creación de aplicaciones porque centra el desarrollo en el dominio del negocio, asegurando que el software refleje fielmente las reglas, procesos y necesidades del negocio al que está destinado.

Logro de Aprendizaje de la Sesión

Al finalizar la sesión, el estudiante reconoce los pasos para crear una aplicación que usa el approach Domain-Driven Design, a través de la implementación de la aplicación.

Conocimientos Previos del Tema

¿Qué es una aplicación?

Una **aplicación** (o *app*, del inglés *application*) es un **programa informático diseñado para realizar tareas específicas**, ya sea en dispositivos electrónicos (como computadoras, smartphones, tablets) o en entornos web.

Secuencia y Explicación

A continuación hablamos sobre el tema de hoy …

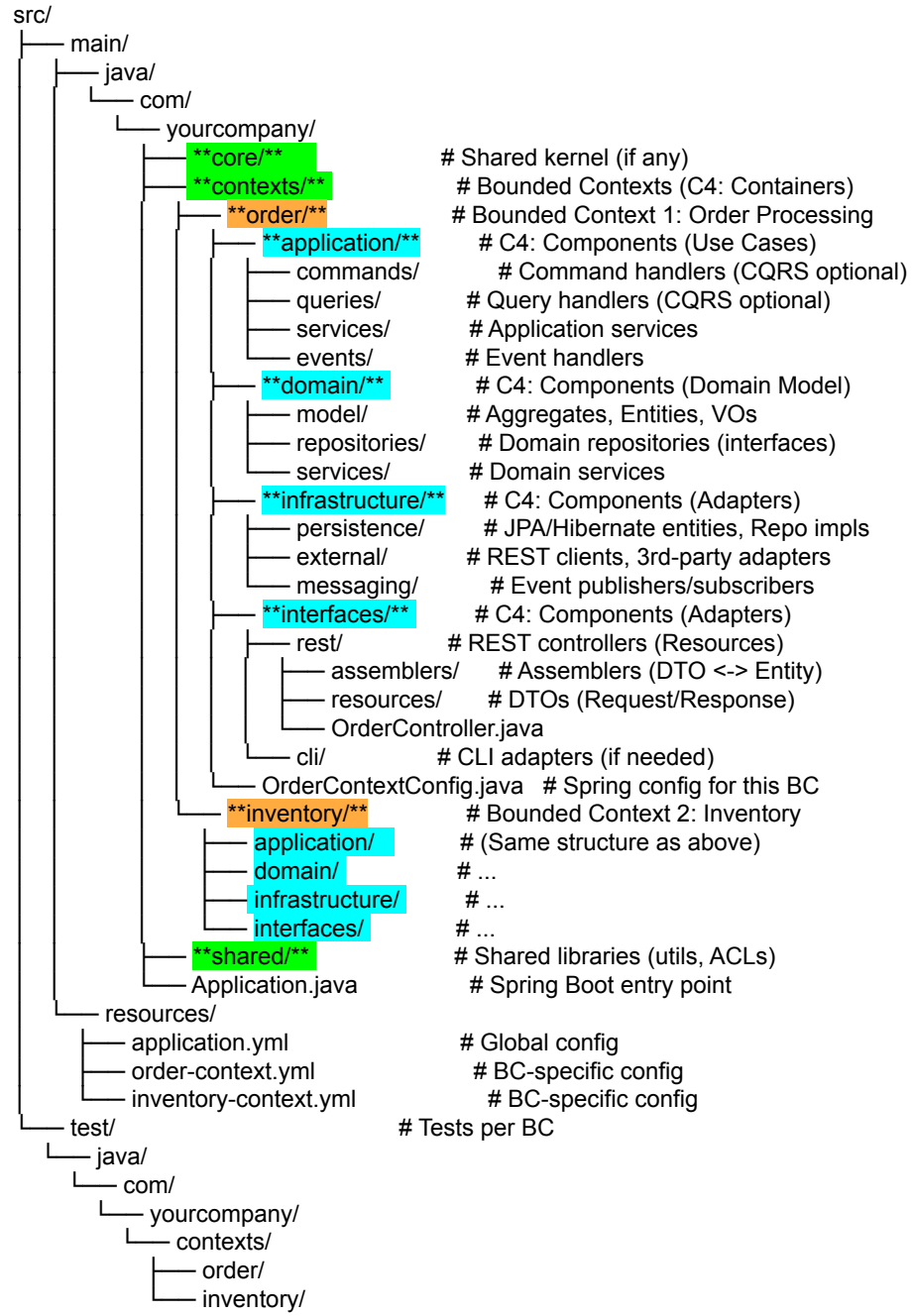
Consideraciones para la creación de una aplicación con el approach Domain-Driven Design.

Estructura de Directorios de una Aplicación que usa el approach Domain-Driven Design

`core/` (Núcleo Compartido - Shared Kernel)

Propósito

- Contiene elementos **compartidos entre múltiples Bounded Contexts** que deben ser consistentes.
- Es un **"Shared Kernel"** (patrón de DDD) para evitar duplicación y garantizar coherencia.
- **¡Usar con cuidado!** Un `core` sobredimensionado puede convertirse en un "God Context" (anti-patrón).



Estructura de Directorios de una Aplicación que usa el approach Domain-Driven Design

core/ (Núcleo Compartido - Shared Kernel)

Estructura y Contenido

```
core/
├── **domain/**              # Modelos de dominio compartidos (si son realmente universales)
│   ├── SharedUser.java      # Ejemplo: Usuario base usado en varios contextos
│   └── ValueObjects.java    # VOs como Money, Email, etc. (si son universales)
├── **application/**         # Servicios compartidos (ej: autenticación básica)
│   └── SecurityService.java # Servicio genérico de seguridad
├── **infrastructure/**      # Utilidades técnicas compartidas
│   ├── logging/             # Configuración de logs
│   ├── exceptions/          # Excepciones globales (ej: NotFoundException)
│   └── time/                # Utilidades de fecha/hora
└── **events/**              # Eventos de dominio compartidos (ej: Evento de integración)
    └── IntegrationEvent.java # Evento base para comunicación entre BCs
```

```
src/
├── main/
│   ├── java/
│   │   └── com/
│   │       └── yourcompany/
│   │           ├── **core/**                    # Shared kernel (if any)
│   │           ├── **contexts/**                # Bounded Contexts (C4: Containers)
│   │           │   ├── **order/**               # Bounded Context 1: Order Processing
│   │           │   │   ├── **application/**     # C4: Components (Use Cases)
│   │           │   │   │   ├── commands/        # Command handlers (CQRS optional)
│   │           │   │   │   ├── queries/         # Query handlers (CQRS optional)
│   │           │   │   │   ├── services/        # Application services
│   │           │   │   │   └── events/          # Event handlers
│   │           │   │   ├── **domain/**          # C4: Components (Domain Model)
│   │           │   │   │   ├── model/           # Aggregates, Entities, VOs
│   │           │   │   │   ├── repositories/    # Domain repositories (interfaces)
│   │           │   │   │   └── services/        # Domain services
│   │           │   │   ├── **infrastructure/**  # C4: Components (Adapters)
│   │           │   │   │   ├── persistence/     # JPA/Hibernate entities, Repo impls
│   │           │   │   │   ├── external/        # REST clients, 3rd-party adapters
│   │           │   │   │   └── messaging/       # Event publishers/subscribers
│   │           │   │   ├── **interfaces/**      # C4: Components (Adapters)
│   │           │   │   │   ├── rest/            # REST controllers (Resources)
│   │           │   │   │   │   ├── assemblers/  # Assemblers (DTO <-> Entity)
│   │           │   │   │   │   ├── resources/   # DTOs (Request/Response)
│   │           │   │   │   │   └── OrderController.java
│   │           │   │   │   └── cli/             # CLI adapters (if needed)
│   │           │   │   └── OrderContextConfig.java # Spring config for this BC
│   │           │   └── **inventory/**          # Bounded Context 2: Inventory
│   │           │       ├── application/        # (Same structure as above)
│   │           │       ├── domain/             # ...
│   │           │       ├── infrastructure/     # ...
│   │           │       └── interfaces/         # ...
│   │           ├── **shared/**                  # Shared libraries (utils, ACLs)
│   │           └── Application.java             # Spring Boot entry point
│   └── resources/
│       ├── application.yml                      # Global config
│       ├── order-context.yml                    # BC-specific config
│       └── inventory-context.yml                # BC-specific config
└── test/                                        # Tests per BC
    └── java/
        └── com/
            └── yourcompany/
                └── contexts/
                    ├── order/
                    └── inventory/
```

Estructura de Directorios de una Aplicación que usa el approach Domain-Driven Design

`core/` (Núcleo Compartido - Shared Kernel)

Cuándo usar `core/`

- Cuando **varios Bounded Contexts (BCs)** necesitan compartir:
  - **Value Objects inmutables** (ej: `Money`, `Email`).
  - **Utilidades técnicas** (manejo de fechas, logs).
  - **Eventos de integración** entre BCs.

¡Advertencia!

- **No poner lógica de negocio específica** aquí (eso pertenece a los BCs).
- Si un elemento solo es usado por **un BC**, debe ir en ese BC, no en `core`.

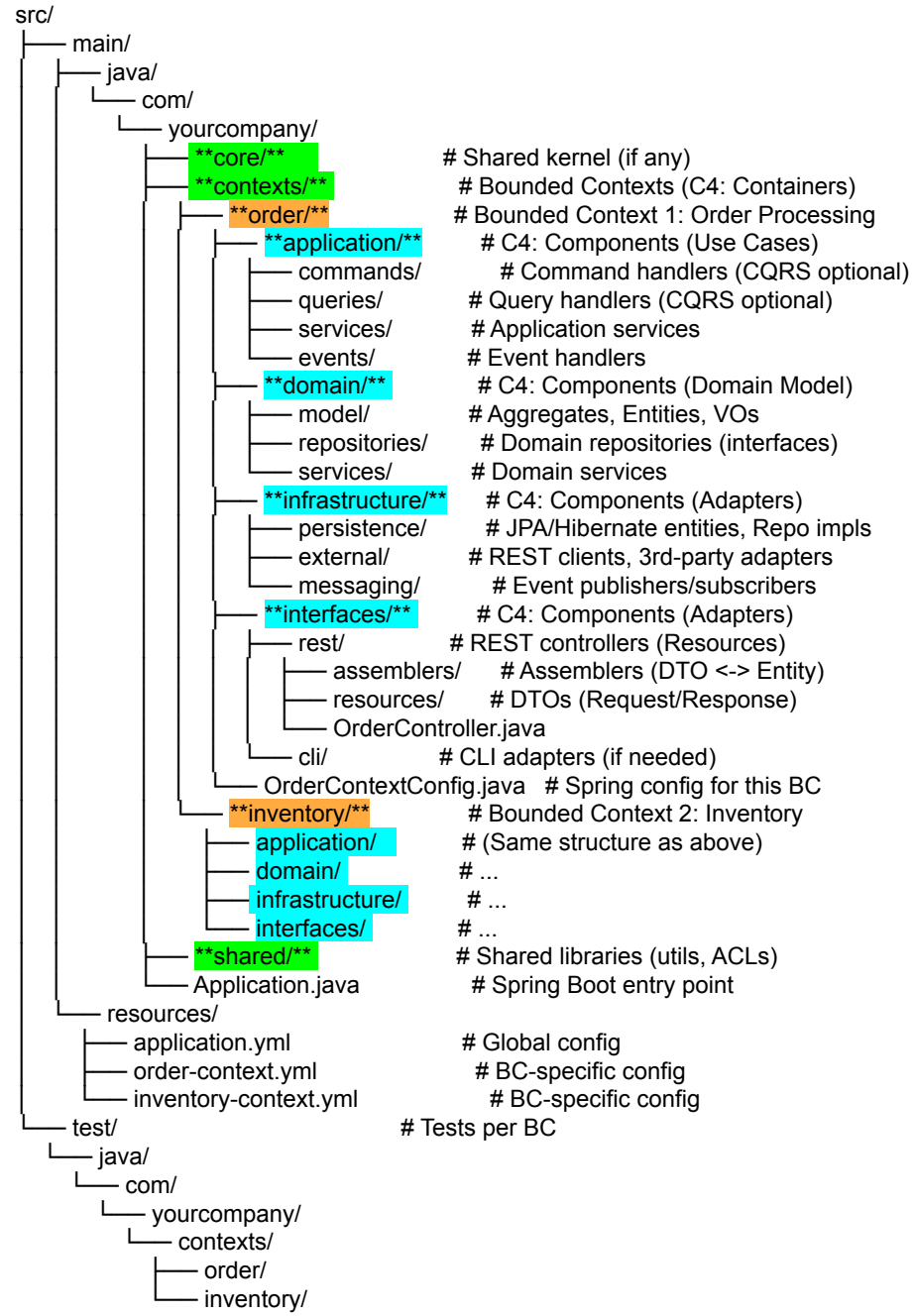
```
src/
├── main/
│   ├── java/
│   │   └── com/
│   │       └── yourcompany/
│   │           ├── **core/**                    # Shared kernel (if any)
│   │           ├── **contexts/**                # Bounded Contexts (C4: Containers)
│   │           │   ├── **order/**               # Bounded Context 1: Order Processing
│   │           │   │   ├── **application/**     # C4: Components (Use Cases)
│   │           │   │   │   ├── commands/        # Command handlers (CQRS optional)
│   │           │   │   │   ├── queries/         # Query handlers (CQRS optional)
│   │           │   │   │   ├── services/        # Application services
│   │           │   │   │   └── events/          # Event handlers
│   │           │   │   ├── **domain/**          # C4: Components (Domain Model)
│   │           │   │   │   ├── model/           # Aggregates, Entities, VOs
│   │           │   │   │   ├── repositories/    # Domain repositories (interfaces)
│   │           │   │   │   └── services/        # Domain services
│   │           │   │   ├── **infrastructure/**  # C4: Components (Adapters)
│   │           │   │   │   ├── persistence/     # JPA/Hibernate entities, Repo impls
│   │           │   │   │   ├── external/        # REST clients, 3rd-party adapters
│   │           │   │   │   └── messaging/       # Event publishers/subscribers
│   │           │   │   ├── **interfaces/**      # C4: Components (Adapters)
│   │           │   │   │   ├── rest/            # REST controllers (Resources)
│   │           │   │   │   │   ├── assemblers/  # Assemblers (DTO <-> Entity)
│   │           │   │   │   │   ├── resources/   # DTOs (Request/Response)
│   │           │   │   │   │   └── OrderController.java
│   │           │   │   │   └── cli/             # CLI adapters (if needed)
│   │           │   │   └── OrderContextConfig.java  # Spring config for this BC
│   │           │   └── **inventory/**          # Bounded Context 2: Inventory
│   │           │       ├── application/        # (Same structure as above)
│   │           │       ├── domain/             # ...
│   │           │       ├── infrastructure/     # ...
│   │           │       └── interfaces/         # ...
│   │           ├── **shared/**                  # Shared libraries (utils, ACLs)
│   │           └── Application.java             # Spring Boot entry point
│   └── resources/
│       ├── application.yml                      # Global config
│       ├── order-context.yml                    # BC-specific config
│       └── inventory-context.yml                # BC-specific config
└── test/                                        # Tests per BC
    └── java/
        └── com/
            └── yourcompany/
                └── contexts/
                    ├── order/
                    └── inventory/
```

Estructura de Directorios de una Aplicación que usa el approach Domain-Driven Design

shared/ (Utilidades y Anti-Corruption Layer - ACL)

Propósito

- Contiene **herramientas técnicas** y adaptadores para evitar acoplamiento entre BCs.
- **No es parte del dominio**, sino infraestructura compartida.
- Aquí vive el **Anti-Corruption Layer (ACL)** para comunicaciones entre BCs.



Estructura de Directorios de una Aplicación que usa el approach Domain-Driven Design

shared/ (Utilidades y Anti-Corruption Layer - ACL)

Estructura y Contenido

```
shared/
├── **acl/**                  # Anti-Corruption Layer (protege un BC de otro)
│   ├── inventory/            # Ejemplo: Adaptador para el BC de Inventario
│   │   └── InventoryACL.java # Traduce modelos de Inventario a modelos locales
│   └── payment/              # Adaptador para pasarela de pagos externa
├── **messaging/**            # Utilidades para eventos/mensajería
│   ├── EventPublisher.java   # Publicador de eventos (Kafka/RabbitMQ)
│   └── EventSubscriber.java  # Consumidor de eventos
├── **http/**                 # Clientes HTTP compartidos
│   └── RestClient.java       # Cliente REST genérico
└── **serialization/**        # Herramientas para JSON/XML
    └── JsonUtils.java        # Utilidades para serialización
```

```
src/
├── main/
│   ├── java/
│   │   └── com/
│   │       └── yourcompany/
│   │           ├── **core/**                  # Shared kernel (if any)
│   │           ├── **contexts/**              # Bounded Contexts (C4: Containers)
│   │           │   ├── **order/**             # Bounded Context 1: Order Processing
│   │           │   │   ├── **application/**   # C4: Components (Use Cases)
│   │           │   │   │   ├── commands/      # Command handlers (CQRS optional)
│   │           │   │   │   ├── queries/       # Query handlers (CQRS optional)
│   │           │   │   │   ├── services/      # Application services
│   │           │   │   │   └── events/        # Event handlers
│   │           │   │   ├── **domain/**        # C4: Components (Domain Model)
│   │           │   │   │   ├── model/         # Aggregates, Entities, VOs
│   │           │   │   │   ├── repositories/  # Domain repositories (interfaces)
│   │           │   │   │   └── services/      # Domain services
│   │           │   │   ├── **infrastructure/** # C4: Components (Adapters)
│   │           │   │   │   ├── persistence/   # JPA/Hibernate entities, Repo impls
│   │           │   │   │   ├── external/      # REST clients, 3rd-party adapters
│   │           │   │   │   └── messaging/     # Event publishers/subscribers
│   │           │   │   ├── **interfaces/**    # C4: Components (Adapters)
│   │           │   │   │   ├── rest/          # REST controllers (Resources)
│   │           │   │   │   │   ├── assemblers/  # Assemblers (DTO <-> Entity)
│   │           │   │   │   │   ├── resources/   # DTOs (Request/Response)
│   │           │   │   │   │   └── OrderController.java
│   │           │   │   │   └── cli/           # CLI adapters (if needed)
│   │           │   │   └── OrderContextConfig.java # Spring config for this BC
│   │           │   └── **inventory/**         # Bounded Context 2: Inventory
│   │           │       ├── application/       # (Same structure as above)
│   │           │       ├── domain/            # ...
│   │           │       ├── infrastructure/    # ...
│   │           │       └── interfaces/        # ...
│   │           ├── **shared/**                # Shared libraries (utils, ACLs)
│   │           └── Application.java           # Spring Boot entry point
│   └── resources/
│       ├── application.yml                    # Global config
│       ├── order-context.yml                  # BC-specific config
│       └── inventory-context.yml              # BC-specific config
└── test/                                      # Tests per BC
    └── java/
        └── com/
            └── yourcompany/
                └── contexts/
                    ├── order/
                    └── inventory/
```

Estructura de Directorios de una Aplicación que usa el approach Domain-Driven Design

Diferencias entre `core/` y `shared/`

`core/`	`shared/`
Contiene **modelos de dominio compartidos** (Shared Kernel).	Contiene **utilidades técnicas** (no de dominio).
Usado cuando **varios BCs necesitan la misma lógica de dominio.**	Usado para **evitar acoplamiento** (ej: ACL, mensajería).
Ejemplo: `Money`, `Email`.	Ejemplo: `RestClient`, `EventPublisher`.

```
src/
├── main/
│   ├── java/
│   │   └── com/
│   │       └── yourcompany/
│   │           ├── **core/**                    # Shared kernel (if any)
│   │           ├── **contexts/**                # Bounded Contexts (C4: Containers)
│   │           │   ├── **order/**               # Bounded Context 1: Order Processing
│   │           │   │   ├── **application/**     # C4: Components (Use Cases)
│   │           │   │   │   ├── commands/        # Command handlers (CQRS optional)
│   │           │   │   │   ├── queries/         # Query handlers (CQRS optional)
│   │           │   │   │   ├── services/        # Application services
│   │           │   │   │   └── events/          # Event handlers
│   │           │   │   ├── **domain/**          # C4: Components (Domain Model)
│   │           │   │   │   ├── model/           # Aggregates, Entities, VOs
│   │           │   │   │   ├── repositories/    # Domain repositories (interfaces)
│   │           │   │   │   └── services/        # Domain services
│   │           │   │   ├── **infrastructure/**  # C4: Components (Adapters)
│   │           │   │   │   ├── persistence/     # JPA/Hibernate entities, Repo impls
│   │           │   │   │   ├── external/        # REST clients, 3rd-party adapters
│   │           │   │   │   └── messaging/       # Event publishers/subscribers
│   │           │   │   ├── **interfaces/**      # C4: Components (Adapters)
│   │           │   │   │   ├── rest/            # REST controllers (Resources)
│   │           │   │   │   │   ├── assemblers/  # Assemblers (DTO <-> Entity)
│   │           │   │   │   │   ├── resources/   # DTOs (Request/Response)
│   │           │   │   │   │   └── OrderController.java
│   │           │   │   │   └── cli/             # CLI adapters (if needed)
│   │           │   │   └── OrderContextConfig.java  # Spring config for this BC
│   │           │   └── **inventory/**          # Bounded Context 2: Inventory
│   │           │       ├── application/        # (Same structure as above)
│   │           │       ├── domain/             # ...
│   │           │       ├── infrastructure/     # ...
│   │           │       └── interfaces/         # ...
│   │           ├── **shared/**                  # Shared libraries (utils, ACLs)
│   │           └── Application.java             # Spring Boot entry point
│   └── resources/
│       ├── application.yml                      # Global config
│       ├── order-context.yml                    # BC-specific config
│       └── inventory-context.yml                # BC-specific config
└── test/                                        # Tests per BC
    └── java/
        └── com/
            └── yourcompany/
                └── contexts/
                    ├── order/
                    └── inventory/
```

Estructura de Directorios de una Aplicación que usa el approach Domain-Driven Design

Bounded Contexts (Ejemplo: `order/`, `inventory/`)

Estructura Detallada de un BC (ej: `order/`)

```
order/
├── **application/**          # Capa de aplicación (casos de uso)
│   ├── commands/             # Comandos (CQRS opcional)
│   │   └── PlaceOrderCommand.java
│   ├── queries/              # Consultas (CQRS opcional)
│   │   └── GetOrderQuery.java
│   ├── services/             # Servicios de aplicación
│   │   └── OrderService.java # Orquesta el flujo de negocio
│   └── events/               # Manejadores de eventos
│       └── OrderPlacedHandler.java
├── **domain/**               # Lógica de negocio pura
│   ├── model/                # Agregados, Entidades, VOs
│   │   ├── Order.java        # Agregado raíz
│   │   └── OrderItem.java    # Entidad/VO
│   ├── repositories/         # Interfaces de repositorio
│   │   └── OrderRepository.java
│   └── services/             # Servicios de dominio
│       └── OrderValidator.java
├──
```

```
src/
├── main/
│   ├── java/
│   │   └── com/
│   │       └── yourcompany/
│   │           ├── **core/**                  # Shared kernel (if any)
│   │           ├── **contexts/**              # Bounded Contexts (C4: Containers)
│   │           │   ├── **order/**             # Bounded Context 1: Order Processing
│   │           │   │   ├── **application/**   # C4: Components (Use Cases)
│   │           │   │   │   ├── commands/      # Command handlers (CQRS optional)
│   │           │   │   │   ├── queries/       # Query handlers (CQRS optional)
│   │           │   │   │   ├── services/      # Application services
│   │           │   │   │   └── events/        # Event handlers
│   │           │   │   ├── **domain/**        # C4: Components (Domain Model)
│   │           │   │   │   ├── model/         # Aggregates, Entities, VOs
│   │           │   │   │   ├── repositories/  # Domain repositories (interfaces)
│   │           │   │   │   └── services/      # Domain services
│   │           │   │   ├── **infrastructure/** # C4: Components (Adapters)
│   │           │   │   │   ├── persistence/   # JPA/Hibernate entities, Repo impls
│   │           │   │   │   ├── external/      # REST clients, 3rd-party adapters
│   │           │   │   │   └── messaging/     # Event publishers/subscribers
│   │           │   │   ├── **interfaces/**    # C4: Components (Adapters)
│   │           │   │   │   ├── rest/          # REST controllers (Resources)
│   │           │   │   │   │   ├── assemblers/  # Assemblers (DTO <-> Entity)
│   │           │   │   │   │   ├── resources/   # DTOs (Request/Response)
│   │           │   │   │   │   └── OrderController.java
│   │           │   │   │   └── cli/           # CLI adapters (if needed)
│   │           │   │   └── OrderContextConfig.java # Spring config for this BC
│   │           │   └── **inventory/**         # Bounded Context 2: Inventory
│   │           │       ├── application/       # (Same structure as above)
│   │           │       ├── domain/            # ...
│   │           │       ├── infrastructure/    # ...
│   │           │       └── interfaces/        # ...
│   │           ├── **shared/**                # Shared libraries (utils, ACLs)
│   │           └── Application.java           # Spring Boot entry point
│   └── resources/
│       ├── application.yml                    # Global config
│       ├── order-context.yml                  # BC-specific config
│       └── inventory-context.yml              # BC-specific config
└── test/                                      # Tests per BC
    └── java/
        └── com/
            └── yourcompany/
                └── contexts/
                    ├── order/
                    └── inventory/
```

Estructura de Directorios de una Aplicación que usa el approach Domain-Driven Design

Bounded Contexts (Ejemplo: `order/`, `inventory/`)

Estructura Detallada de un BC (ej: `order/`)

```
order/
├── **application/**          # Capa de aplicación (casos de uso)
├── **domain/**               # Lógica de negocio pura
├── **infrastructure/**       # Implementaciones técnicas
│   ├── persistence/          # JPA/Hibernate
│   │   ├── OrderJpaEntity.java  # Entidad de persistencia
│   │   └── JpaOrderRepository.java # Impl. del repositorio
│   ├── external/             # Adaptadores para APIs externas
│   │   └── PaymentGatewayAdapter.java
│   └── messaging/            # Eventos de integración
│       └── OrderEventPublisher.java
└── **interfaces/**           # Capa de presentación (API REST)
    ├── rest/
    │   ├── assemblers/       # Assemblers (DTO <-> Entidad)
    │   │   └── OrderAssembler.java
    │   ├── resources/        # DTOs (Request/Response)
    │   │   ├── OrderRequest.java
    │   │   └── OrderResponse.java
    │   └── OrderController.java # Recurso REST
    └── cli/                  # CLI (si aplica)
```

```
src/
├── main/
│   ├── java/
│   │   └── com/
│   │       └── yourcompany/
│   │           ├── **core/**                    # Shared kernel (if any)
│   │           ├── **contexts/**                # Bounded Contexts (C4: Containers)
│   │           │   ├── **order/**               # Bounded Context 1: Order Processing
│   │           │   │   ├── **application/**     # C4: Components (Use Cases)
│   │           │   │   │   ├── commands/        # Command handlers (CQRS optional)
│   │           │   │   │   ├── queries/         # Query handlers (CQRS optional)
│   │           │   │   │   ├── services/        # Application services
│   │           │   │   │   └── events/          # Event handlers
│   │           │   │   ├── **domain/**          # C4: Components (Domain Model)
│   │           │   │   │   ├── model/           # Aggregates, Entities, VOs
│   │           │   │   │   ├── repositories/    # Domain repositories (interfaces)
│   │           │   │   │   └── services/        # Domain services
│   │           │   │   ├── **infrastructure/**  # C4: Components (Adapters)
│   │           │   │   │   ├── persistence/     # JPA/Hibernate entities, Repo impls
│   │           │   │   │   ├── external/        # REST clients, 3rd-party adapters
│   │           │   │   │   └── messaging/       # Event publishers/subscribers
│   │           │   │   ├── **interfaces/**      # C4: Components (Adapters)
│   │           │   │   │   ├── rest/            # REST controllers (Resources)
│   │           │   │   │   │   ├── assemblers/  # Assemblers (DTO <-> Entity)
│   │           │   │   │   │   ├── resources/   # DTOs (Request/Response)
│   │           │   │   │   │   └── OrderController.java
│   │           │   │   │   └── cli/             # CLI adapters (if needed)
│   │           │   │   └── OrderContextConfig.java  # Spring config for this BC
│   │           │   └── **inventory/**          # Bounded Context 2: Inventory
│   │           │       ├── application/        # (Same structure as above)
│   │           │       ├── domain/             # ...
│   │           │       ├── infrastructure/     # ...
│   │           │       └── interfaces/         # ...
│   │           ├── **shared/**                  # Shared libraries (utils, ACLs)
│   │           └── Application.java             # Spring Boot entry point
│   └── resources/
│       ├── application.yml                      # Global config
│       ├── order-context.yml                    # BC-specific config
│       └── inventory-context.yml                # BC-specific config
└── test/                                        # Tests per BC
    └── java/
        └── com/
            └── yourcompany/
                └── contexts/
                    ├── order/
                    └── inventory/
```

Estructura de Directorios de una Aplicación que usa el approach Domain-Driven Design

Bounded Contexts (Ejemplo: `order/`, `inventory/`)

Características Principales de Bounded Contexts (BCs)

Un **Bounded Context** en DDD es un límite conceptual donde un **modelo de dominio** específico es definido y aplicable. Sus características clave son:

1. **Ubiquitous Language**
   - Vocabulario compartido entre desarrolladores y expertos del dominio.
   - Ejemplo: En un BC de `Pedidos`, "Cliente" puede ser diferente a un BC de `Facturación`.
2. **Autonomía**
   - Tiene su propia lógica de negocio, base de datos y reglas.
   - Ejemplo: El BC `Inventario` maneja stock, mientras `Pedidos` gestiona órdenes.

```
src/
├── main/
│   ├── java/
│   │   └── com/
│   │       └── yourcompany/
│   │           ├── **core/**                    # Shared kernel (if any)
│   │           ├── **contexts/**                # Bounded Contexts (C4: Containers)
│   │           │   ├── **order/**               # Bounded Context 1: Order Processing
│   │           │   │   ├── **application/**     # C4: Components (Use Cases)
│   │           │   │   │   ├── commands/        # Command handlers (CQRS optional)
│   │           │   │   │   ├── queries/         # Query handlers (CQRS optional)
│   │           │   │   │   ├── services/        # Application services
│   │           │   │   │   └── events/          # Event handlers
│   │           │   │   ├── **domain/**          # C4: Components (Domain Model)
│   │           │   │   │   ├── model/           # Aggregates, Entities, VOs
│   │           │   │   │   ├── repositories/    # Domain repositories (interfaces)
│   │           │   │   │   └── services/        # Domain services
│   │           │   │   ├── **infrastructure/**  # C4: Components (Adapters)
│   │           │   │   │   ├── persistence/     # JPA/Hibernate entities, Repo impls
│   │           │   │   │   ├── external/        # REST clients, 3rd-party adapters
│   │           │   │   │   └── messaging/       # Event publishers/subscribers
│   │           │   │   ├── **interfaces/**      # C4: Components (Adapters)
│   │           │   │   │   ├── rest/            # REST controllers (Resources)
│   │           │   │   │   │   ├── assemblers/  # Assemblers (DTO <-> Entity)
│   │           │   │   │   │   ├── resources/   # DTOs (Request/Response)
│   │           │   │   │   │   └── OrderController.java
│   │           │   │   │   └── cli/             # CLI adapters (if needed)
│   │           │   │   └── OrderContextConfig.java  # Spring config for this BC
│   │           │   └── **inventory/**          # Bounded Context 2: Inventory
│   │           │       ├── application/        # (Same structure as above)
│   │           │       ├── domain/             # ...
│   │           │       ├── infrastructure/     # ...
│   │           │       └── interfaces/         # ...
│   │           ├── **shared/**                 # Shared libraries (utils, ACLs)
│   │           └── Application.java            # Spring Boot entry point
│   └── resources/
│       ├── application.yml                     # Global config
│       ├── order-context.yml                   # BC-specific config
│       └── inventory-context.yml               # BC-specific config
└── test/                                       # Tests per BC
    └── java/
        └── com/
            └── yourcompany/
                └── contexts/
                    ├── order/
                    └── inventory/
```

Estructura de Directorios de una Aplicación que usa el approach Domain-Driven Design

Bounded Contexts (Ejemplo: `order/`, `inventory/`)

Características Principales de Bounded Contexts (BCs)

3. Componentes Principales

- **Domain Layer**: Entidades, Agregados, Value Objects, Domain Services.
- **Application Layer**: Casos de uso (Commands, Queries, Event Handlers).
- **Infrastructure Layer**: Persistencia (Repositorios), APIs externas.
- **Interfaces**: REST APIs (Controllers), DTOs, Assemblers.

4. Comunicación entre BCs

- **Eventos de Dominio**: Ej: `OrderPlacedEvent` para notificar a `Inventario`.
- **Anti-Corruption Layer (ACL)**: Adaptadores para traducir modelos entre BCs.

5. Context Mapping

- Define relaciones entre BCs (Ej: *Partnership*, *Customer-Supplier*).

```
src/
├── main/
│   ├── java/
│   │   └── com/
│   │       └── yourcompany/
│   │           ├── **core/**                    # Shared kernel (if any)
│   │           ├── **contexts/**                # Bounded Contexts (C4: Containers)
│   │           │   ├── **order/**               # Bounded Context 1: Order Processing
│   │           │   │   ├── **application/**     # C4: Components (Use Cases)
│   │           │   │   │   ├── commands/        # Command handlers (CQRS optional)
│   │           │   │   │   ├── queries/         # Query handlers (CQRS optional)
│   │           │   │   │   ├── services/        # Application services
│   │           │   │   │   └── events/          # Event handlers
│   │           │   │   ├── **domain/**          # C4: Components (Domain Model)
│   │           │   │   │   ├── model/           # Aggregates, Entities, VOs
│   │           │   │   │   ├── repositories/    # Domain repositories (interfaces)
│   │           │   │   │   └── services/        # Domain services
│   │           │   │   ├── **infrastructure/**  # C4: Components (Adapters)
│   │           │   │   │   ├── persistence/     # JPA/Hibernate entities, Repo impls
│   │           │   │   │   ├── external/        # REST clients, 3rd-party adapters
│   │           │   │   │   └── messaging/       # Event publishers/subscribers
│   │           │   │   ├── **interfaces/**      # C4: Components (Adapters)
│   │           │   │   │   ├── rest/            # REST controllers (Resources)
│   │           │   │   │   │   ├── assemblers/  # Assemblers (DTO <-> Entity)
│   │           │   │   │   │   ├── resources/   # DTOs (Request/Response)
│   │           │   │   │   │   └── OrderController.java
│   │           │   │   │   └── cli/             # CLI adapters (if needed)
│   │           │   │   └── OrderContextConfig.java  # Spring config for this BC
│   │           │   └── **inventory/**           # Bounded Context 2: Inventory
│   │           │       ├── application/         # (Same structure as above)
│   │           │       ├── domain/              # ...
│   │           │       ├── infrastructure/      # ...
│   │           │       └── interfaces/          # ...
│   │           ├── **shared/**                  # Shared libraries (utils, ACLs)
│   │           └── Application.java             # Spring Boot entry point
│   └── resources/
│       ├── application.yml                      # Global config
│       ├── order-context.yml                    # BC-specific config
│       └── inventory-context.yml                # BC-specific config
└── test/                                        # Tests per BC
    └── java/
        └── com/
            └── yourcompany/
                └── contexts/
                    ├── order/
                    └── inventory/
```

Estructura de Directorios de una Aplicación que usa el approach Domain-Driven Design

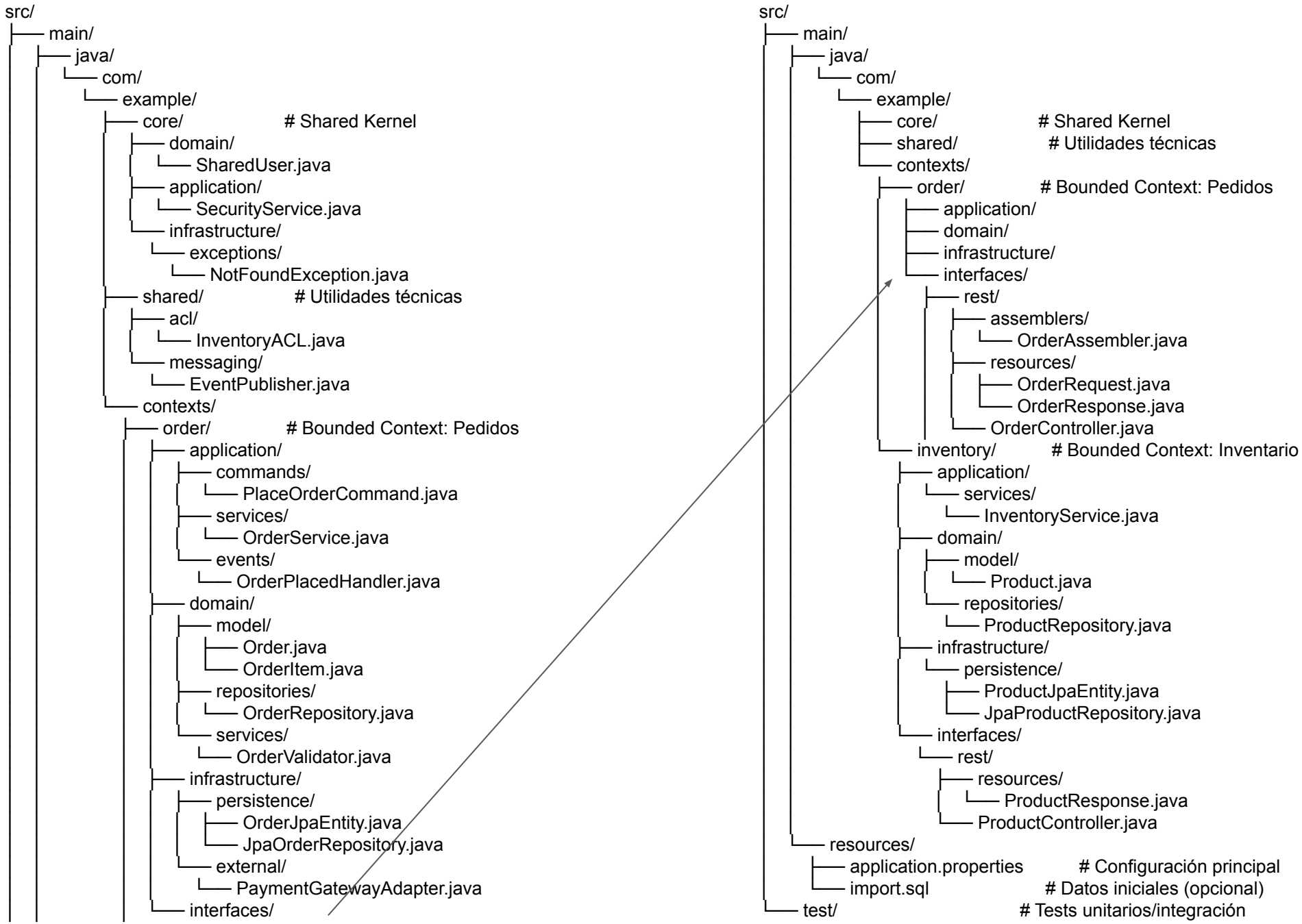
Bounded Contexts (Ejemplo: `order/`, `inventory/`)

Elementos Clave de un Bounded Context

Elemento	Descripción	Ejemplo
Agregados	Raíces de consistencia que agrupan entidades/VOs.	`Order` (con `OrderItem`).
Repositorios	Interfaces para persistencia (implementadas en Infraestructura)	`OrderRepository`.
Domain Services	Lógica de dominio que no pertenece a una entidad específica.	`OrderValidator`.
Domain Events	Eventos que reflejan cambios importantes en el dominio.	`OrderPlacedEvent`.
Application Services	Orquestan flujos de negocio (llamadas a Domain Layer).	`PlaceOrderService`.
DTOs & Assemblers	Transforman datos entre Domain Model y APIs.	`OrderDTO`, `OrderAssembler`.

```
src/
├── main/
│   ├── java/
│   │   └── com/
│   │       └── yourcompany/
│   │           ├── **core/**                    # Shared kernel (if any)
│   │           ├── **contexts/**                # Bounded Contexts (C4: Containers)
│   │           │   ├── **order/**               # Bounded Context 1: Order Processing
│   │           │   │   ├── **application/**     # C4: Components (Use Cases)
│   │           │   │   │   ├── commands/        # Command handlers (CQRS optional)
│   │           │   │   │   ├── queries/         # Query handlers (CQRS optional)
│   │           │   │   │   ├── services/        # Application services
│   │           │   │   │   └── events/          # Event handlers
│   │           │   │   ├── **domain/**          # C4: Components (Domain Model)
│   │           │   │   │   ├── model/           # Aggregates, Entities, VOs
│   │           │   │   │   ├── repositories/    # Domain repositories (interfaces)
│   │           │   │   │   └── services/        # Domain services
│   │           │   │   ├── **infrastructure/**  # C4: Components (Adapters)
│   │           │   │   │   ├── persistence/     # JPA/Hibernate entities, Repo impls
│   │           │   │   │   ├── external/        # REST clients, 3rd-party adapters
│   │           │   │   │   └── messaging/       # Event publishers/subscribers
│   │           │   │   ├── **interfaces/**      # C4: Components (Adapters)
│   │           │   │   │   ├── rest/            # REST controllers (Resources)
│   │           │   │   │   │   ├── assemblers/  # Assemblers (DTO <-> Entity)
│   │           │   │   │   │   ├── resources/   # DTOs (Request/Response)
│   │           │   │   │   │   └── OrderController.java
│   │           │   │   │   └── cli/             # CLI adapters (if needed)
│   │           │   │   └── OrderContextConfig.java  # Spring config for this BC
│   │           │   └── **inventory/**          # Bounded Context 2: Inventory
│   │           │       ├── application/        # (Same structure as above)
│   │           │       ├── domain/             # ...
│   │           │       ├── infrastructure/     # ...
│   │           │       └── interfaces/         # ...
│   │           ├── **shared/**                  # Shared libraries (utils, ACLs)
│   │           └── Application.java             # Spring Boot entry point
│   └── resources/
│       ├── application.yml                      # Global config
│       ├── order-context.yml                    # BC-specific config
│       └── inventory-context.yml                # BC-specific config
└── test/                                        # Tests per BC
    └── java/
        └── com/
            └── yourcompany/
                └── contexts/
                    ├── order/
                    └── inventory/
```

Ejemplo


src/
main/
java/
com/
example/
core/ # Shared Kernel
domain/
SharedUser.java
application/
SecurityService.java
infrastructure/
exceptions/
NotFoundException.java
shared/ # Utilidades técnicas
acl/
InventoryACL.java
messaging/
EventPublisher.java
contexts/
order/ # Bounded Context: Pedidos
application/
commands/
PlaceOrderCommand.java
services/
OrderService.java
events/
OrderPlacedHandler.java
domain/
model/
Order.java
OrderItem.java
repositories/
OrderRepository.java
services/
OrderValidator.java
infrastructure/
persistence/
OrderJpaEntity.java
JpaOrderRepository.java
external/
PaymentGatewayAdapter.java
interfaces/
src/
main/
java/
com/
example/
core/ # Shared Kernel
shared/ # Utilidades técnicas
contexts/
order/ # Bounded Context: Pedidos
application/
domain/
infrastructure/
interfaces/
rest/
assemblers/
OrderAssembler.java
resources/
OrderRequest.java
OrderResponse.java
OrderController.java
inventory/ # Bounded Context: Inventario
application/
services/
InventoryService.java
domain/
model/
Product.java
repositories/
ProductRepository.java
infrastructure/
persistence/
ProductJpaEntity.java
JpaProductRepository.java
interfaces/
rest/
resources/
ProductResponse.java
ProductController.java
resources/
application.properties # Configuración principal
import.sql # Datos iniciales (opcional)
test/ # Tests unitarios/integración

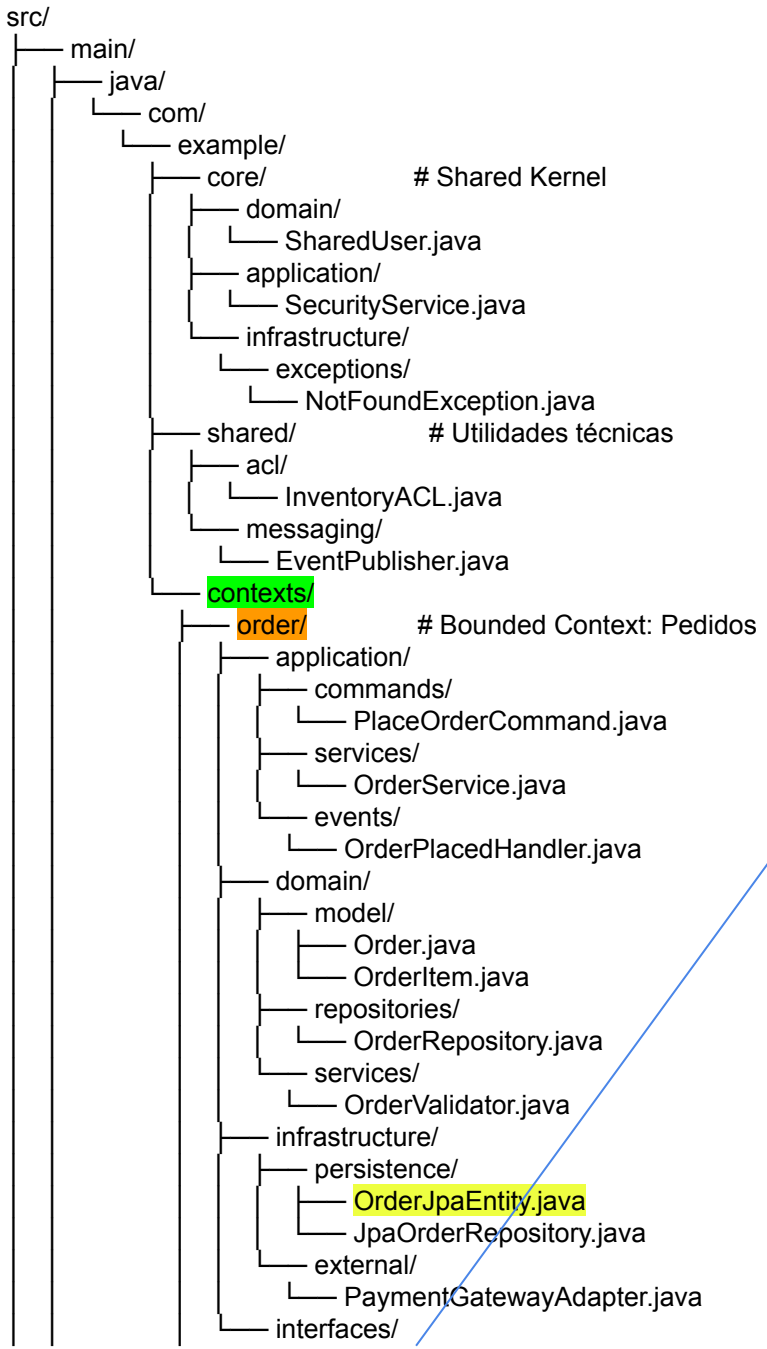
Ejemplo

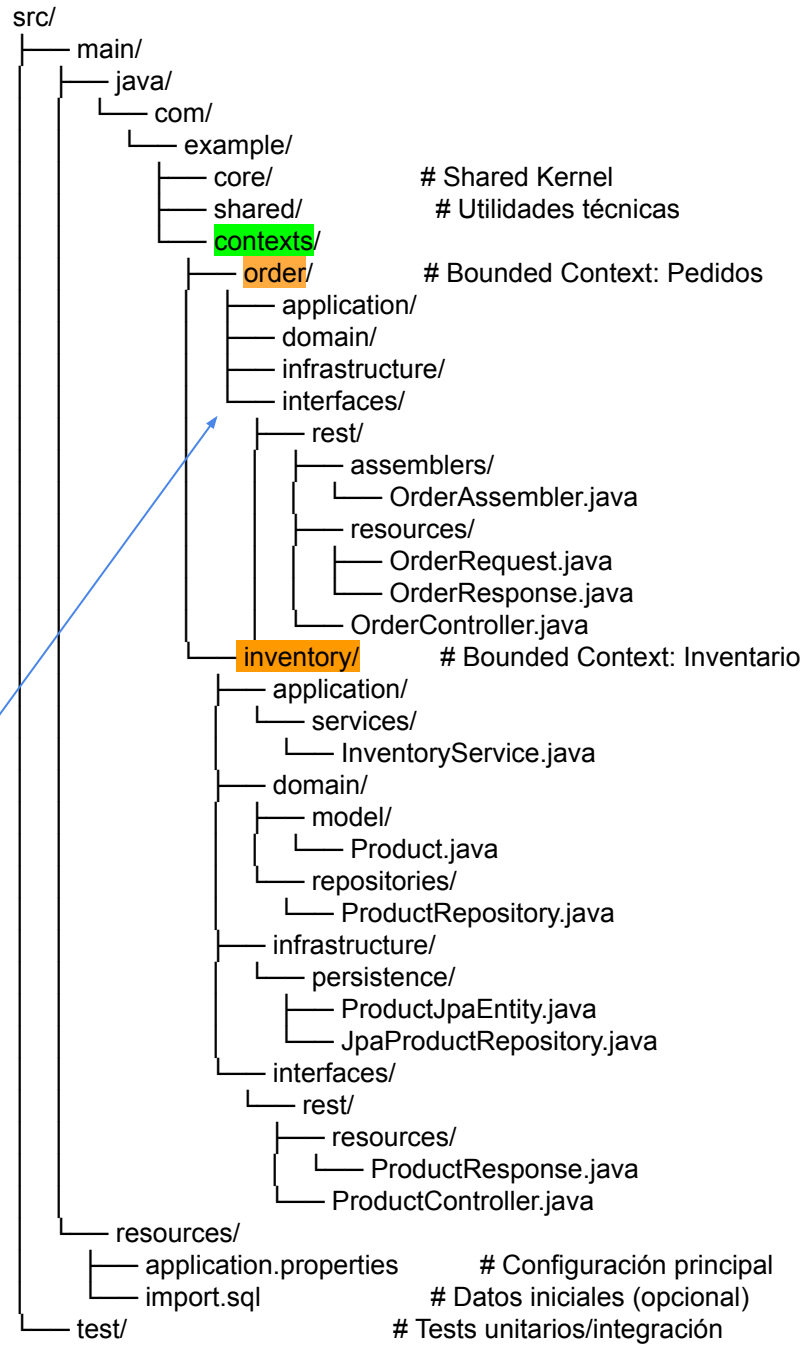
Bounded Context: Order

contexts/order/infrastructure/persistence/OrderJpaEntity.java

```
package
com.example.contexts.order.infrastructure.persistence;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

@Entity
@Table(name = "orders")
public class OrderJpaEntity {
    @Id
    private String id;
    private String customerId;
    // Getters y setters
}
```

Propósito: Entidad JPA para persistencia.





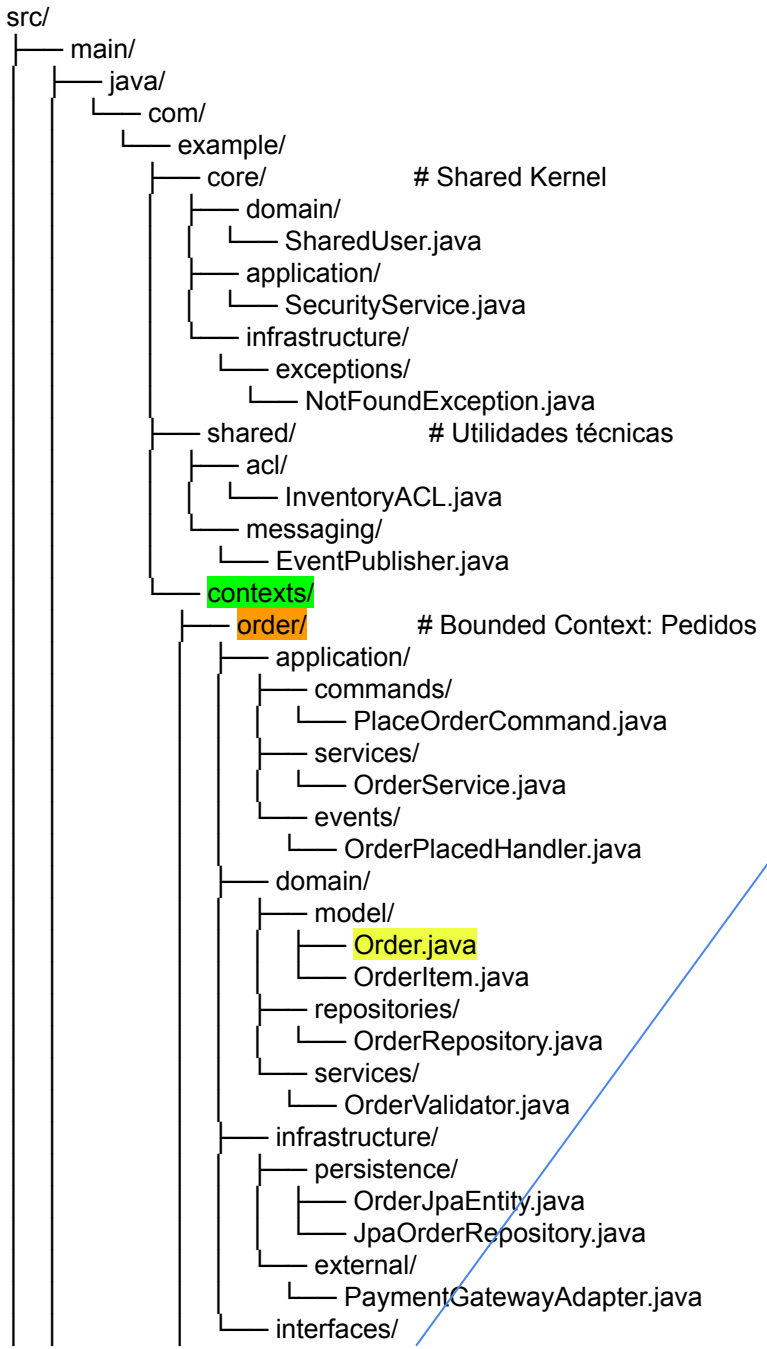
Ejemplo

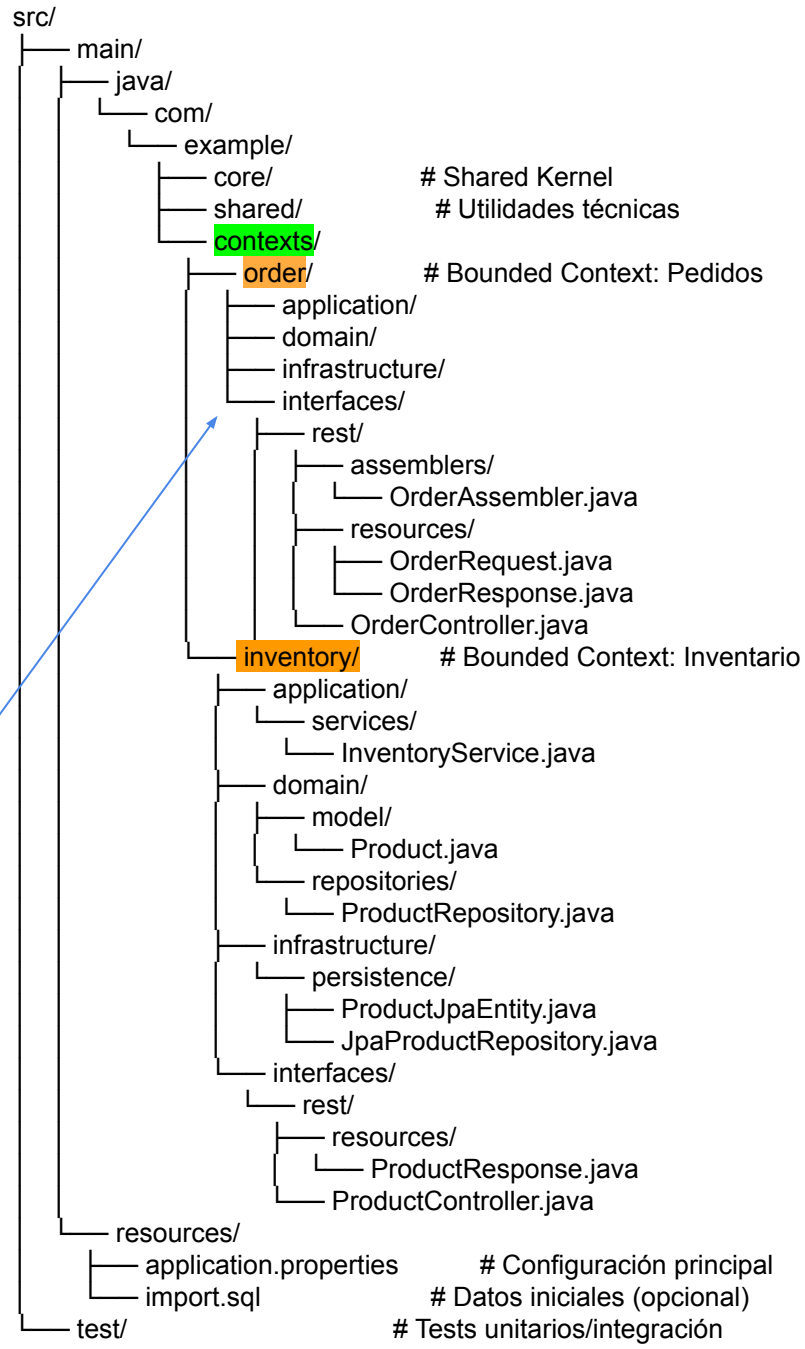
Bounded Context: Order

`contexts/order/domain/model/Order.java`

```
package
com.example.contexts.order.domain.model;

import java.util.List;

public class Order {
    private String id;
    private String customerId;
    private List<OrderItem> items;
    // Getters y setters
}
```

Propósito: Agregado raíz del BC Order.





Ejemplo

Bounded Context: Order

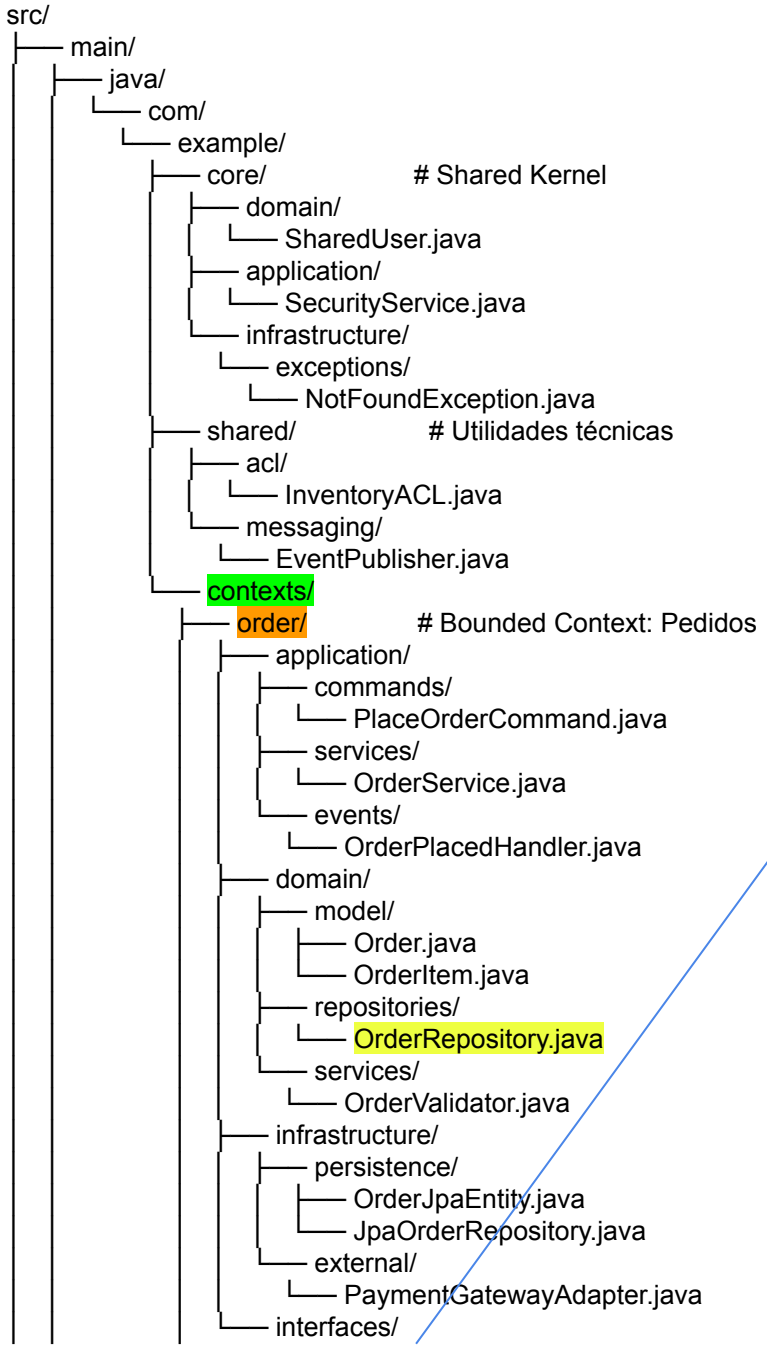
contexts/order/domain/repositories/OrderRepository.java

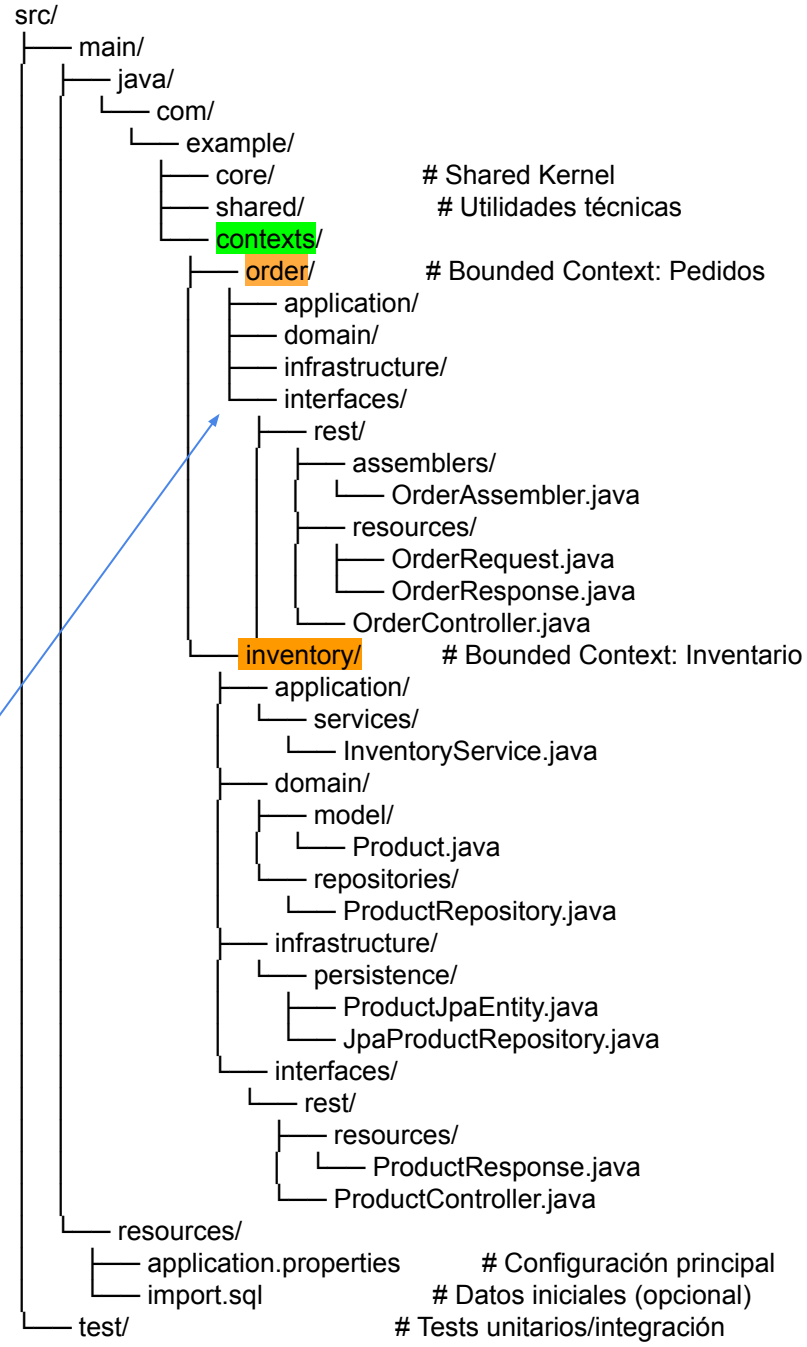
```
package com.example.contexts.order.domain.repositories;

import com.example.contexts.order.domain.model.Order;

import
org.springframework.data.jpa.repository.JpaRepository;

public interface OrderRepository extends
JpaRepository<Order, String> {

  // Spring Data JPA genera automáticamente los métodos
CRUD básicos:

  // save(), findById(), delete(), findAll(), etc.

}
```

Propósito

Patrón Repository:

- Abstraction entre la **capa de dominio** y la **capa de persistencia**.
- Permite al dominio acceder a datos sin conocer detalles de infraestructura (ej: JPA, MySQL).





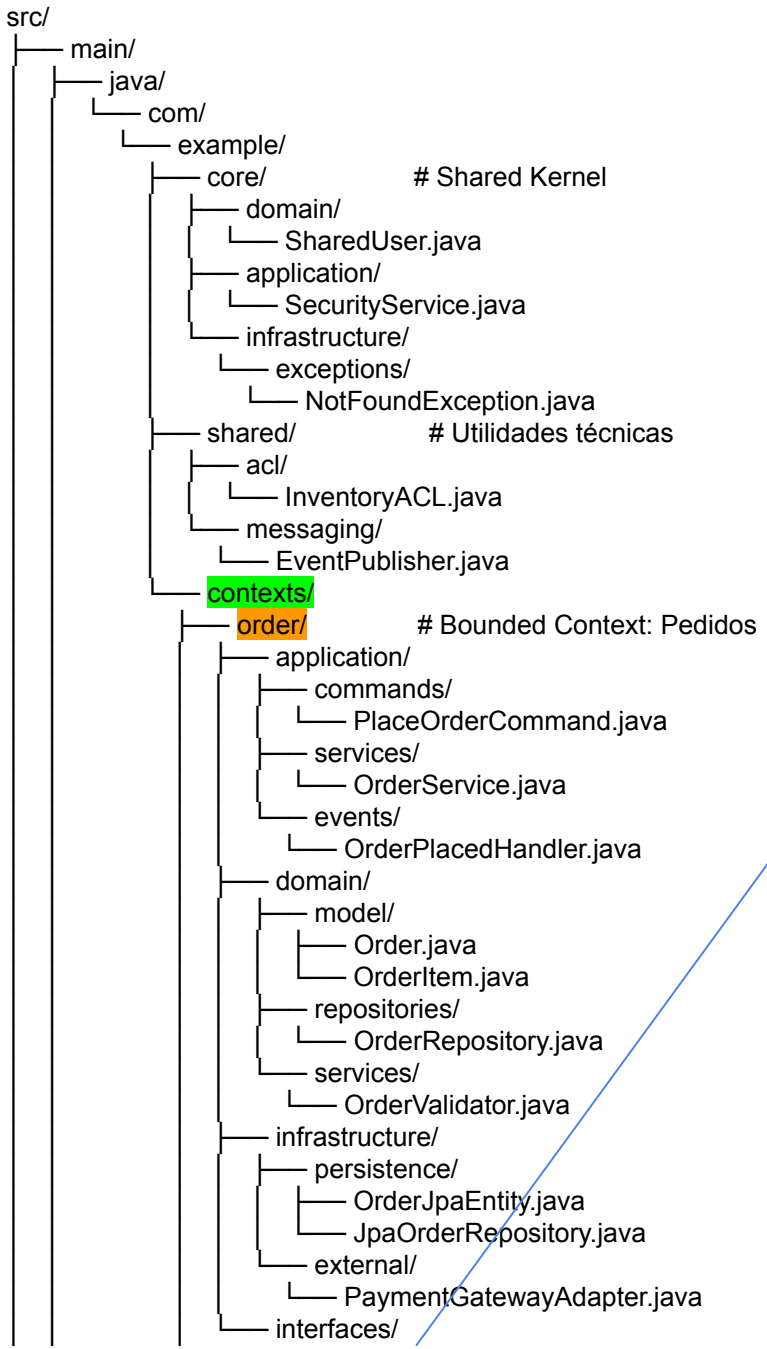
Ejemplo

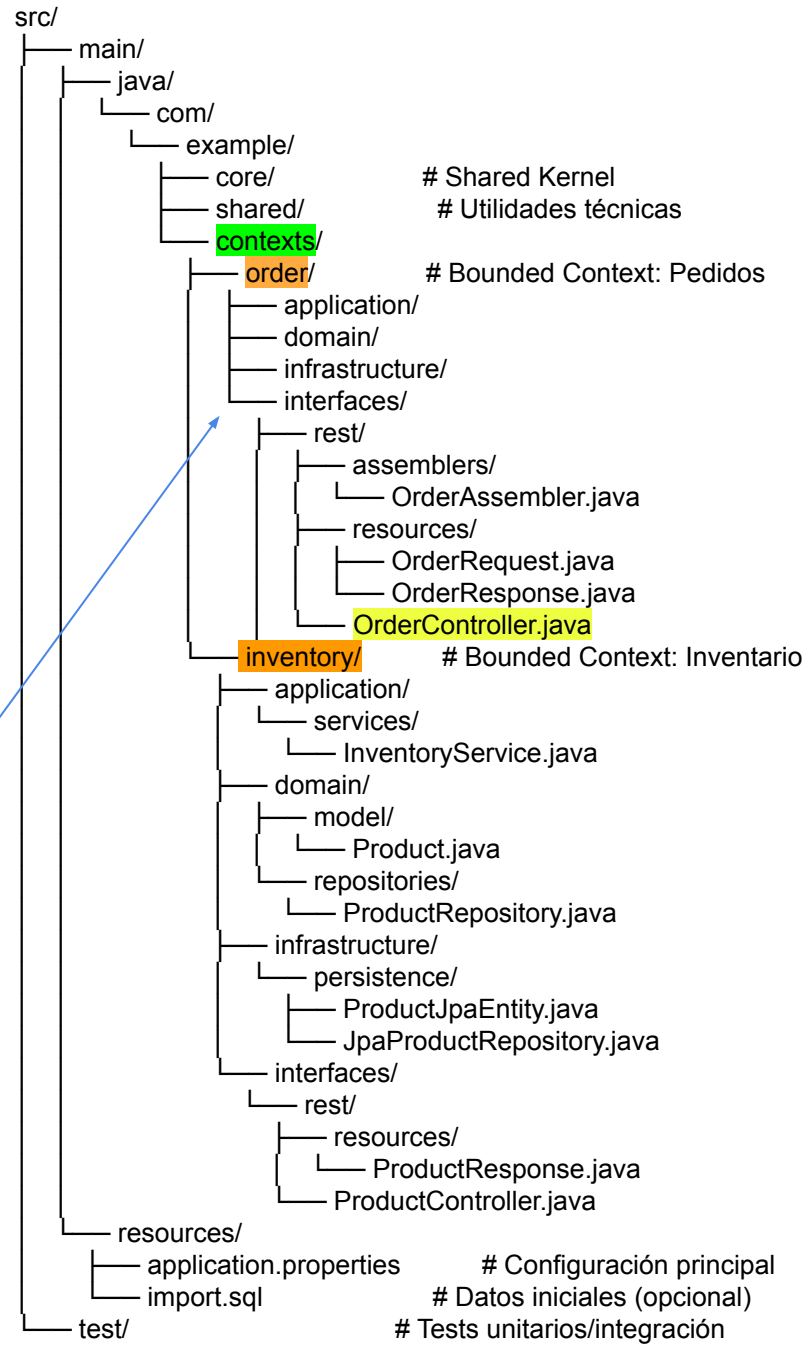
Bounded Context: Order

contexts/order/interfaces/rest/OrderController.java

```
package com.example.contexts.order.interfaces.rest;

import
com.example.contexts.order.application.services.OrderService;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/orders")
public class OrderController {
    private final OrderService orderService;

    public OrderController(OrderService orderService) {
        this.orderService = orderService;
    }

    @PostMapping
    public String placeOrder(@RequestBody OrderRequest
request) {
        return orderService.placeOrder(request.toDomain());
    }
}
```

Propósito: Controlador REST para el BC Order.





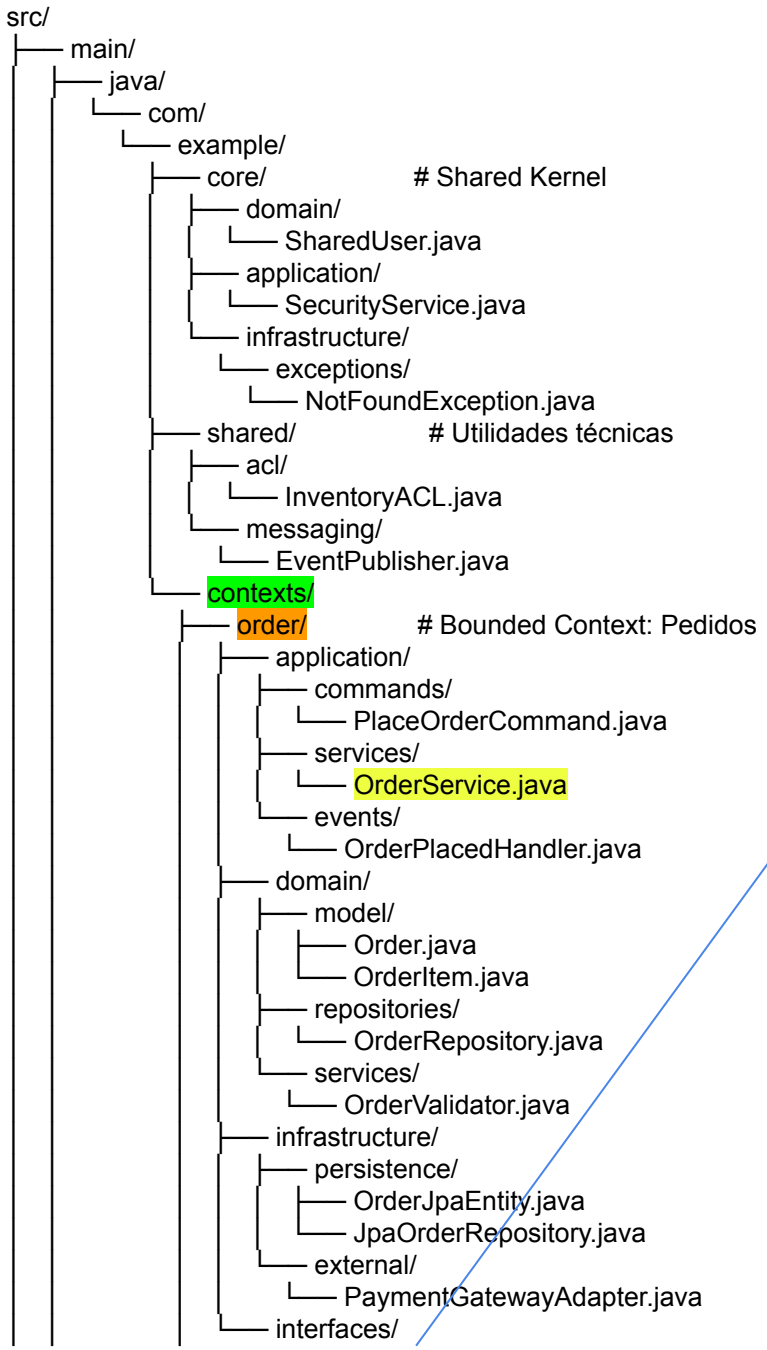
Ejemplo

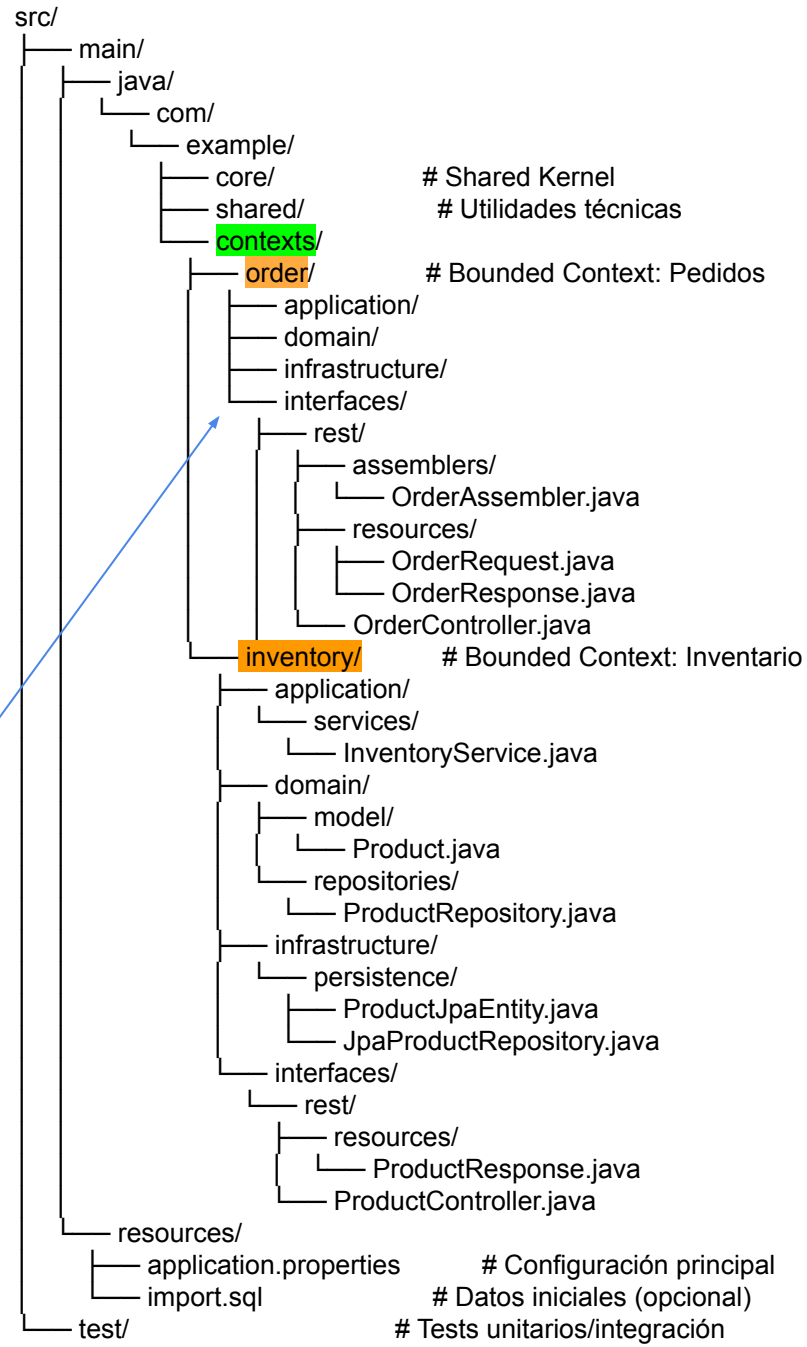
Bounded Context: Order

contexts/order/application/services/OrderService.java

```
package
com.example.contexts.order.application.services;

import com.example.contexts.order.domain.model.Order;
import
com.example.contexts.order.domain.repositories.OrderRe
pository;

public class OrderService {
    private final OrderRepository orderRepository;

    public OrderService(OrderRepository orderRepository)
{
        this.orderRepository = orderRepository;
    }

    public String placeOrder(Order order) {
        orderRepository.save(order);
        return order.getId();
    }
}
```

Propósito: Servicio de aplicación para orquestar la lógica de pedidos.





Ejemplo

Bounded Context: Order

`contexts/order/interfaces/rest/resources/OrderRequest.java`

```
package
com.example.contexts.order.interfaces.rest.resources;

import
com.example.contexts.order.domain.model.OrderItem;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.NotNull;
import java.util.List;

public class OrderRequest {
    @NotBlank(message = "Customer ID is required")
    private String customerId;

    @NotNull(message = "Items list cannot be null")
    private List<OrderItemRequest> items;

    // Getters y Setters
    public String getCustomerId() {
        return customerId;
    }

    public void setCustomerId(String customerId) {
        this.customerId = customerId;
    }

    public List<OrderItemRequest> getItems() {
        return items;
    }

    public void setItems(List<OrderItemRequest> items) {
        this.items = items;
    }

    // Clase interna para los items del pedido
    public static class OrderItemRequest {
        @NotBlank(message = "Product ID is required")
        private String productId;

        @NotNull(message = "Quantity is required")
        private Integer quantity;

        // Getters y Setters
        public String getProductId() {
            return productId;
        }

        public void setProductId(String productId) {
            this.productId = productId;
        }

        public Integer getQuantity() {
            return quantity;
        }

        public void setQuantity(Integer quantity) {
            this.quantity = quantity;
        }
    }
}
```

```
src/
├── main/
│   ├── java/
│   │   └── com/
│   │       └── example/
│   │           ├── core/                    # Shared Kernel
│   │           ├── shared/                  # Utilidades técnicas
│   │           └── contexts/
│   │               ├── order/               # Bounded Context: Pedidos
│   │               │   ├── application/
│   │               │   ├── domain/
│   │               │   ├── infrastructure/
│   │               │   └── interfaces/
│   │               │       ├── rest/
│   │               │       │   ├── assemblers/
│   │               │       │   │   └── OrderAssembler.java
│   │               │       │   ├── resources/
│   │               │       │   │   ├── OrderRequest.java
│   │               │       │   │   └── OrderResponse.java
│   │               │       │   └── OrderController.java
│   │               └── inventory/           # Bounded Context: Inventario
│   │                   ├── application/
│   │                   │   └── services/
│   │                   │       └── InventoryService.java
│   │                   ├── domain/
│   │                   │   ├── model/
│   │                   │   │   └── Product.java
│   │                   │   └── repositories/
│   │                   │       └── ProductRepository.java
│   │                   ├── infrastructure/
│   │                   │   └── persistence/
│   │                   │       ├── ProductJpaEntity.java
│   │                   │       └── JpaProductRepository.java
│   │                   └── interfaces/
│   │                       └── rest/
│   │                           ├── resources/
│   │                           │   └── ProductResponse.java
│   │                           └── ProductController.java
│   └── resources/
│       ├── application.properties           # Configuración principal
│       └── import.sql                       # Datos iniciales (opcional)
└── test/                                    # Tests unitarios/integración
```

Ejemplo

Bounded Context: Order

`contexts/order/interfaces/rest/resources/OrderResponse.java`

```
package com.example.contexts.order.interfaces.rest.resources;
import java.time.LocalDateTime;
import java.util.List;

public class OrderResponse {
    private String orderId;
    private String customerId;
    private LocalDateTime createdAt;
    private List<OrderItemResponse> items;
    private String status; // e.g., "CREATED", "SHIPPED"
    // Constructor (para inmutabilidad)
    public OrderResponse(String orderId, String customerId,
LocalDateTime createdAt, List<OrderItemResponse> items,
String status) {
        this.orderId = orderId;
        this.customerId = customerId;
        this.createdAt = createdAt;
        this.items = items;
        this.status = status;
    }
    // Getters (sin setters para inmutabilidad)
    public String getOrderId() {
        return orderId;
    }
    public String getCustomerId() {
        return customerId;
    }
    public LocalDateTime getCreatedAt() {
        return createdAt;
    }
    public List<OrderItemResponse> getItems() {
        return items;
    }

    public String getStatus() {
        return status;
    }

    // Clase interna para items
    public static class OrderItemResponse {
        private String productId;
        private int quantity;
        private double price;

        public OrderItemResponse(String productId, int quantity,
double price) {
            this.productId = productId;
            this.quantity = quantity;
            this.price = price;
        }

        // Getters
        public String getProductId() {
            return productId;
        }

        public int getQuantity() {
            return quantity;
        }

        public double getPrice() {
            return price;
        }
    }
}
```

```
src/
├── main/
│   ├── java/
│   │   └── com/
│   │       └── example/
│   │           ├── core/                    # Shared Kernel
│   │           ├── shared/                  # Utilidades técnicas
│   │           └── contexts/
│   │               ├── order/               # Bounded Context: Pedidos
│   │               │   ├── application/
│   │               │   ├── domain/
│   │               │   ├── infrastructure/
│   │               │   └── interfaces/
│   │               │       ├── rest/
│   │               │       │   ├── assemblers/
│   │               │       │   │   └── OrderAssembler.java
│   │               │       │   ├── resources/
│   │               │       │   │   ├── OrderRequest.java
│   │               │       │   │   └── OrderResponse.java
│   │               │       │   └── OrderController.java
│   │               └── inventory/           # Bounded Context: Inventario
│   │                   ├── application/
│   │                   │   └── services/
│   │                   │       └── InventoryService.java
│   │                   ├── domain/
│   │                   │   ├── model/
│   │                   │   │   └── Product.java
│   │                   │   └── repositories/
│   │                   │       └── ProductRepository.java
│   │                   ├── infrastructure/
│   │                   │   └── persistence/
│   │                   │       ├── ProductJpaEntity.java
│   │                   │       └── JpaProductRepository.java
│   │                   └── interfaces/
│   │                       └── rest/
│   │                           ├── resources/
│   │                           │   └── ProductResponse.java
│   │                           └── ProductController.java
│   └── resources/
│       ├── application.properties           # Configuración principal
│       └── import.sql                       # Datos iniciales (opcional)
└── test/                                    # Tests unitarios/integración
```

Espacio Práctico

Realizando la implementación del código …

Cierre de Sesión

¿Qué se aprendió hoy?

Cuáles son los pasos a seguir para crear una aplicación en java conectada a una base de datos MySQL.

Conclusiones

- La estructura de archivos y directorios es una sugerida y podría tener algunas mínimas variantes.

Recordatorio de cosas por hacer

- Revisar esta referencia respecto a OpenAPI.
- Preparar el siguiente avance del “Final Project”.

Gracias.