# Etapa 3 — Aplicação de Padrões de Projeto

## Introdução

O sistema Controle de Almoxarifado tem como objetivo gerenciar o cadastro de fornecedores, materiais e requisições de retirada de estoque.
Para garantir um código flexível, reutilizável e coeso, foram aplicados três padrões de projeto:

- Factory (Padrão Criacional)
- Adapter (Padrão Estrutural)
- Observer (Padrão Comportamental)

Cada padrão foi escolhido de acordo com uma necessidade específica do sistema, respeitando os princípios SOLID e favorecendo a manutenção e extensão do código.

## 1. Padrão Criacional — Factory

Pacote: com.example.ControleAlmoxarifado.design_patterns.creational.factory

Classes principais:

- EntityFactory<T, R>
- FornecedorFactory
- MaterialFactory

### Problema Resolvido

Ao criar entidades a partir de DTOs (Data Transfer Objects), o sistema estava repetindo código nos serviços, convertendo manualmente cada DTO para a respectiva entidade.

### Solução com o Factory

O padrão Factory Method foi usado para centralizar a lógica de criação de entidades, desacoplando a criação dos objetos da lógica de negócio.
Cada tipo de entidade possui uma factory própria:

- FornecedorFactory cria instâncias de Fornecedor.
- MaterialFactory cria instâncias de Material.

### Benefícios

- Reduz duplicação de código.
- Facilita futuras alterações na criação das entidades.
- Segue o princípio Open/Closed (OCP) — é possível criar novas fábricas sem alterar o código existente.

## 2. Padrão Estrutural — Adapter

Pacote: com.example.ControleAlmoxarifado.design_patterns.structural.adapter

Classes principais:

- MaterialAdapter
- ExcelMaterialAdapter
- ListaExcelMaterialAdapter

**Problema Resolvido**

O sistema precisava importar materiais a partir de planilhas Excel, que possuem estrutura de dados diferente das entidades do sistema (Material e Fornecedor).

**Solução com o Adapter**

O padrão Adapter foi aplicado para converter os dados da planilha (ExcelMaterial) em entidades do sistema compatíveis (Material e Fornecedor).

**Benefícios**

- Permite integração com fontes externas (planilhas, APIs, etc.) sem alterar as entidades.
- Facilita futuras importações em outros formatos (CSV, JSON, XML).
- Segue o princípio Single Responsibility — conversão e negócio ficam separados.
- Reaproveitar código ao tratar listas usando um Adapter composto (ListaExcelMaterialAdapter).

**3. Padrão Comportamental — Observer**

Pacote: com.example.ControleAlmoxarifado.design_patterns.behavioral.observer

Classes principais:

- RequisicaoObserver (interface)
- AtualizarEstoqueObserver
- RequisicaoEventManager
- RequisicaoService

**Problema Resolvido**

Quando uma requisição é atendida, o estoque dos materiais deve ser automaticamente atualizado. Antes, o serviço de requisição fazia essa atualização diretamente, violando o princípio de responsabilidade única.

**Solução com o Observer**

O padrão Observer desacoplou a lógica de atualização do estoque. Agora, quando uma requisição muda de status, o sistema notifica todos os observadores interessados (no caso, o AtualizarEstoqueObserver).

**Benefícios**

- Desacopla a atualização de estoque da lógica de requisição.
- Permite adicionar novos observadores (ex: envio de e-mail, auditoria, etc.) sem alterar o serviço principal.
- Segue o princípio Open/Closed — o sistema pode ser estendido sem ser modificado.

## Conclusão

| Tipo de Padrão | Nome | Função no Sistema | Benefício Principal |
|---|---|---|---|
| **Criacional** | Factory | Criação de entidades (Material, Fornecedor) | Centraliza e simplifica a criação de objetos |
| **Estrutural** | Adapter | Conversão de planilhas Excel para entidades | Permite integração com formatos externos |
| **Comportamental** | Observer | Atualização automática de estoque | Desacopla eventos e ações reativas |