### Exercício 12.1 – Aplicação com Padrões de Design

Este exercício abrange conceitos fundamentais de **padrões de design**, **separação de preocupações** e **relacionamentos entre entidades**. Seguindo estas instruções, pode reproduzir o projeto no **Spring Boot** utilizando o **Spring Initializr**.

### Parte 1: Configuração do Projeto no Spring Initializr

1. **Acessar o Spring Initializr:**
   * Vá ao site [Spring Initializr](https://start.spring.io/).
2. **Configurar o projeto:**
   * **Project:** Maven
   * **Language:** Java
   * **Spring Boot:** Última versão estável.
   * **Group:** com.example
   * **Artifact:** designpatterns
   * **Name:** designpatterns
   * **Package Name:** com.example.designpatterns
   * **Packaging:** Jar
   * **Java Version:** 21.
3. **Adicionar dependências:**
   * **Spring Web:** Para criar APIs REST.
   * **Spring Data JPA:** Para interação com a base de dados.
   * **H2 Database:** Para uma base de dados em memória nos testes.
   * **Spring Boot DevTools (opcional):** Para facilitar o desenvolvimento.
4. **Gerar o projeto:**
   * Clique em **Generate** para descarregar o projeto como um ficheiro ZIP.
5. **Abrir o projeto no VS Code:**
   * Extraia o ficheiro ZIP.
   * No VS Code, clique em **File > Open Folder** e selecione a pasta do projeto.

### Parte 2: Estrutura do Projeto

1. **Criar as pastas necessárias:**
   * Em src/main/java/com/example/designpatterns, crie as pastas:
     + config para o exemplo do padrão Singleton.
     + factory para o exemplo do padrão Factory.
     + entity para as entidades.
     + repository para os repositórios.
     + service para os serviços.
     + controller para os controladores.
2. A estrutura deverá ser:

src/  
├── main/  
│ ├── java/  
│ │ └── com.example.designpatterns/  
│ │ ├── config/  
│ │ ├── factory/  
│ │ ├── entity/  
│ │ ├── repository/  
│ │ ├── service/  
│ │ └── controller/  
├── resources/  
 └── application.properties

### Parte 3: Implementação do Padrão Singleton

1. **Criar a classe ConfiguracaoGlobal em config:**

package com.example.designpatterns.config;  
  
import org.springframework.stereotype.Component;  
  
@Component  
public class ConfiguracaoGlobal {  
  
 private String appName = "Minha Aplicação";  
  
 public String getAppName() {  
 return appName;  
 }  
  
 public void setAppName(String appName) {  
 this.appName = appName;  
 }  
}

**Explicação:**  
A classe é anotada com @Component, o que faz com que o Spring a gerencie como um bean Singleton por padrão.

### Parte 4: Implementação do Padrão Factory

1. **Criar a classe ServicoFactory em factory:**

package com.example.designpatterns.factory;  
  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
  
@Configuration  
public class ServicoFactory {  
  
 @Bean  
 public Servico exemploServico() {  
 return new ServicoConcreto();  
 }  
}  
  
interface Servico {  
 void executar();  
}  
  
class ServicoConcreto implements Servico {  
 @Override  
 public void executar() {  
 System.out.println("Serviço Concreto Executado!");  
 }  
}

**Explicação:**  
O método @Bean cria e gerencia a instância da classe ServicoConcreto, que é fornecida sempre que o bean for solicitado.

### Parte 5: Separação de Preocupações

1. **Criar a entidade Cliente em entity:**

package com.example.designpatterns.entity;  
  
import jakarta.persistence.Entity;  
import jakarta.persistence.GeneratedValue;  
import jakarta.persistence.GenerationType;  
import jakarta.persistence.Id;  
  
@Entity  
public class Cliente {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
 private String nome;  
  
 // Getters e Setters  
}

1. **Criar o repositório ClienteRepository em repository:**

package com.example.designpatterns.repository;  
  
import com.example.designpatterns.entity.Cliente;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
public interface ClienteRepository extends JpaRepository<Cliente, Long> {  
}

1. **Criar o serviço ClienteService em service:**

package com.example.designpatterns.service;  
  
import com.example.designpatterns.entity.Cliente;  
import com.example.designpatterns.repository.ClienteRepository;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
  
@Service  
public class ClienteService {  
  
 private final ClienteRepository clienteRepository;  
  
 public ClienteService(ClienteRepository clienteRepository) {  
 this.clienteRepository = clienteRepository;  
 }  
  
 public List<Cliente> listarClientes() {  
 return clienteRepository.findAll();  
 }  
}

1. **Criar o controlador ClienteController em controller:**

package com.example.designpatterns.controller;  
  
import com.example.designpatterns.entity.Cliente;  
import com.example.designpatterns.service.ClienteService;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
import java.util.List;  
  
@RestController  
public class ClienteController {  
  
 private final ClienteService clienteService;  
  
 public ClienteController(ClienteService clienteService) {  
 this.clienteService = clienteService;  
 }  
  
 @GetMapping("/clientes")  
 public List<Cliente> listarClientes() {  
 return clienteService.listarClientes();  
 }  
}

### Parte 6: Relacionamentos @OneToMany e @ManyToOne

1. **Atualizar a entidade Cliente:**

@OneToMany(mappedBy = "cliente")  
private List<Encomenda> encomendas;

1. **Criar a entidade Encomenda:**

package com.example.designpatterns.entity;  
  
import jakarta.persistence.\*;  
  
@Entity  
public class Encomenda {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
 private String descricao;  
  
 @ManyToOne  
 @JoinColumn(name = "cliente\_id")  
 private Cliente cliente;  
  
 // Getters e Setters  
}

### Parte 7: Configuração no application.properties

spring.datasource.url=jdbc:h2:mem:testdb  
spring.datasource.driverClassName=org.h2.Driver  
spring.datasource.username=sa  
spring.datasource.password=password  
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

### Parte 8: Testar o Projeto

1. **Iniciar o projeto:**
   * Use o comando mvn spring-boot:run.
2. **Acessar os endpoints:**
   * GET /clientes para listar clientes.