### **Documentação do código**

**Data:** 25/11/2024

**Equipe:** Guilherme Ferreira Sampaio, João Victor Costa Nascimento, Marcos Vinicios Sousa Batista, Otávio Xavier da Silva Marinho, Eduardo Rocha dos Santos, Caio Oliveira Sampaio, Murilo Bassilo Lopes, Geraldo Manoel de Lima Alencar.

### **1. br.com.ucsal.controller.annotation.Command**

**Propósito**:  
Essa interface serve como uma "marca" para todas as classes que representam comandos dentro do padrão MVC. O método execute() deve ser implementado em todas as classes que a implementam.

**Detalhamento**:

* **Método**:
  + execute(HttpServletRequest request, HttpServletResponse response) – Define a ação que o comando vai executar ao ser chamado.  
    Ele recebe dois parâmetros: request e response, que representam, respectivamente, a requisição HTTP e a resposta HTTP.

### **2. br.com.ucsal.controller.annotation.Inject**

**Propósito**:  
A anotação @Inject indica que a dependência de um campo ou atributo da classe será injetada automaticamente pelo mecanismo de injeção de dependência da aplicação.

**Detalhamento**:

* **Uso**:
  + Normalmente é usada em campos de classe para injeção automática de instâncias. Essa anotação funciona em conjunto com o gerenciador de injeção (como o InjectionManager).
  + É aplicada sobre variáveis que dependem de algum serviço ou repositório, permitindo a injeção de suas implementações.

### **3. br.com.ucsal.controller.annotation.Rota**

**Propósito**:  
A anotação @Rota define o caminho (URL) para o qual a classe que a utiliza será associada. No contexto da aplicação, ela mapeia os caminhos de URL para as classes (geralmente servlets).

**Detalhamento**:

* **Atributo**:
  + caminho – Representa o caminho da URL a ser mapeado, ou seja, quando essa URL for acessada, a classe anotada com @Rota será invocada.

### **4. br.com.ucsal.controller.annotation.Singleton**

**Propósito**:  
A anotação @Singleton indica que a classe deve seguir o padrão de design Singleton, ou seja, deverá ter uma única instância durante o ciclo de vida da aplicação.

**Detalhamento**:

* **Uso**:
  + É aplicada em classes para que, ao serem solicitadas, sempre seja retornada a mesma instância da classe. Esse comportamento é gerenciado pelo SingletonManager.

### **5. br.com.ucsal.controller.managers.InjectionManager**

**Propósito**:  
O InjectionManager é responsável pela injeção de dependências nas classes, onde ele analisa os campos anotados com @Inject e os inicializa automaticamente, vinculando suas dependências.

**Detalhamento**:

* **Função**:
  + Fornece a funcionalidade de injeção de dependências por meio da reflexão, criando instâncias e configurando os valores das dependências.

### **6. br.com.ucsal.controller.managers.SingletonManager**

**Propósito**:  
Gerencia a criação e acesso às instâncias Singleton da aplicação, garantindo que apenas uma instância da classe anotada com @Singleton seja criada.

**Detalhamento**:

* **Métodos**:
  + getInstance(Class<T> clazz) – Retorna a instância única da classe fornecida, criando-a se necessário, utilizando reflexão e a configuração de acesso ao construtor.

### **7. br.com.ucsal.controller.InicializadorListener**

**Propósito**:  
Listener responsável pela inicialização de recursos durante a inicialização da aplicação. Ele percorre as classes dentro de pacotes específicos, mapeando rotas e inicializando classes Singleton.

**Detalhamento**:

* **Funções**:
  + contextInitialized(ServletContextEvent sce) – Inicializa recursos quando a aplicação é carregada.
  + processDirectory(File directory, ServletContext context) – Processa diretórios e mapeia as classes anotadas com @Rota e @Singleton.

### **8. br.com.ucsal.controller.ProdutoAdicionarServlet**

**Propósito**:  
Classe Servlet que trata da adição de produtos, dividida entre métodos doGet() e doPost(). Quando o método é GET, exibe o formulário de adição de produto. Quando POST, processa os dados e cria um novo produto.

**Detalhamento**:

* **Métodos**:
  + doGet(HttpServletRequest request, HttpServletResponse response) – Exibe o formulário de adição de produto.
  + doPost(HttpServletRequest request, HttpServletResponse response) – Recebe os dados do formulário e adiciona um novo produto.

### **9. br.com.ucsal.controller.ProdutoController**

**Propósito**:  
Controlador responsável por mapear as rotas de produtos e delegar a execução para os respectivos comandos (será usado como um dispatcher de comandos).

**Detalhamento**:

* **Funções**:
  + init() – Mapeia as rotas para comandos específicos.
  + service(HttpServletRequest request, HttpServletResponse response) – Recebe a requisição e determina qual comando deve ser executado.

### **10. br.com.ucsal.controller.ProdutoEditarServlet**

**Propósito**:  
Classe Servlet responsável pela edição de um produto. Ela carrega um produto, exibe o formulário com os dados do produto e, quando o formulário for enviado, atualiza as informações.

**Detalhamento**:

* **Métodos**:
  + doGet(HttpServletRequest request, HttpServletResponse response) – Exibe o formulário de edição com os dados do produto.
  + doPost(HttpServletRequest request, HttpServletResponse response) – Atualiza as informações do produto após a edição.

### **11. br.com.ucsal.controller.ProdutoExcluirServlet**

**Propósito**:  
Classe Servlet que trata a exclusão de um produto, recebendo o id do produto a ser removido e invocando o serviço para realizar a exclusão.

**Detalhamento**:

* **Método**:
  + execute(HttpServletRequest request, HttpServletResponse response) – Executa a exclusão do produto e redireciona para a lista de produtos.

### **12. br.com.ucsal.controller.ProdutoListarServlet**

**Propósito**:  
Classe Servlet responsável por listar todos os produtos cadastrados. Ela consulta os produtos no banco de dados ou repositório em memória e os passa para a visualização.

**Detalhamento**:

* **Método**:
  + execute(HttpServletRequest request, HttpServletResponse response) – Recupera os produtos e os passa para a página JSP.

### **13. br.com.ucsal.model.Produto**

**Propósito**:  
Classe de modelo que representa o produto, com atributos como id, nome e preco, além de métodos getters e setters para manipulação desses dados.

**Detalhamento**:

* **Atributos**:
  + id, nome, preco – Representam os dados do produto.
* **Métodos**:
  + Métodos getters e setters para acessar e modificar os atributos da classe.

### **14. br.com.ucsal.persistencia.HSQLProdutoRepository**

**Propósito**:  
Implementação do repositório de produtos utilizando o banco de dados HSQL (em memória). Ele implementa a interface ProdutoRepository e provê métodos para adicionar, remover, atualizar e listar produtos.

**Detalhamento**:

* **Métodos**:
  + adicionar(Produto entidade) – Adiciona um novo produto.
  + remover(Integer id) – Remove um produto pelo ID.
  + listar() – Retorna a lista de todos os produtos.
  + atualizar(Produto entidade) – Atualiza as informações de um produto existente.
  + obterPorID(Integer id) – Retorna um produto específico pelo ID.

### **15. br.com.ucsal.persistencia.MemoriaProdutoRepository**

**Propósito**:  
Implementação em memória do repositório de produtos. Utiliza um mapa em memória para armazenar os produtos e um contador atômico para gerar IDs únicos.

**Detalhamento**:

* **Métodos**:
  + Implementa as funções de adicionar, remover, atualizar, listar e obter produtos, mas usa armazenamento em memória.

### **16. br.com.ucsal.persistencia.PersistenciaFactory**

**Propósito**:  
Fábrica para instanciar repositórios de produtos. Dependendo do tipo de persistência desejada, ela retorna o repositório adequado, seja em memória (MemoriaProdutoRepository) ou HSQL (HSQLProdutoRepository).

**Detalhamento**:

* **Método**:
  + getProdutoRepository(int type) – Retorna uma instância do repositório de produtos com base no tipo fornecido.

### **17. br.com.ucsal.persistencia.ProdutoRepository**

**Propósito**:  
Interface que define as operações básicas que um repositório de produtos deve oferecer, como adicionar, remover, atualizar, listar e obter por ID.

**Detalhamento**:

* **Métodos**:
  + Operações de CRUD (criar, ler, atualizar e excluir) para a entidade Produto.

### **18. br.com.ucsal.service.ProdutoService**

**Propósito**:  
Serviço que abstrai as operações de manipulação de produtos. Ele delega as operações de adicionar, remover, atualizar e listar produtos para o repositório.

**Detalhamento**:

* **Métodos**:
  + adicionarProduto(String nome, double preco) – Adiciona um novo produto.
  + removerProduto(Integer id) – Remove um produto.
  + obterProdutoPorId(Integer id) – Obtém um produto pelo ID.
  + atualizarProduto(Produto p) – Atualiza um produto existente.
  + listarProdutos() – Lista todos os produtos.

### **19. br.com.ucsal.util.DatabaseInitializationListener**

**Propósito**:  
Listener responsável pela inicialização do banco de dados, garantindo que o banco HSQL seja criado e preenchido com dados iniciais ao iniciar a aplicação.

**Detalhamento**:

* **Métodos**:
  + contextInitialized(ServletContextEvent sce) – Inicializa o banco de dados quando a aplicação começa.
  + contextDestroyed(ServletContextEvent sce) – Avisa quando a aplicação está sendo finalizada.

### **20. br.com.ucsal.util.DatabaseUtil**

**Propósito**:  
Utilitário para facilitar a interação com o banco de dados HSQL. Ele gerencia a conexão e cria a estrutura inicial do banco de dados.

**Detalhamento**:

* **Métodos**:
  + iniciarBanco() – Inicializa o banco de dados com uma tabela produtos e insere alguns dados iniciais.
  + getConnection() – Estabelece uma conexão com o banco de dados.