**コーディング規約**

**1. 緒言**

コーディング規約では、コードの一貫性、可読性、保守性を向上させることを目的とします。以下の規約に従い、プロジェクトの品質を維持します。

**2. コーディング規約: Java**

**2.1 命名規則**

クラス名

使用規則: パスカルケースを使用。クラス名は明確かつ簡潔、役割を反映する名前を付けます。

例:

Public class CustomerService {

// クラスの内容

}

public class OrderRepository {

// クラスの内容

}

メソッド名

使用規則: キャメルケースを使用。動詞で開始し、処理内容を明確化します。

例:

public Customer getCustomerById(Long id) {

// ロジック部分

}

変数名

使用規則: キャメルケースを使用。可読性を重視し、略語を避けます。

例:

String customerName = "John";

int orderCount = 5;

定数名

使用規則: 定数はすべて大文字で記述。単語間はアンダースコアで区切る。

例:

public static final int MAX\_USER\_COUNT = 1000;

public static final String DEFAULT\_MESSAGE = "Operation Successful";

**2.2 コーディングスタイル**

インデント規則

使用規則: スペース4つを使用します。タブは使用しません。

例:

public void exampleMethod() {

if (condition) {

System.out.println("Condition is true!");

}

}

ブランケット（波括弧）スタイル

使用規則: 開く波括弧は同じ行に記述し、閉じる波括弧はそのブロックとインデントを揃えます。

例:

public void exampleMethod() {

for (int i = 0; i < 10; i++) {

System.out.println(i);

}

}

クラス設置場所

パッケージ構造は以下を基準に設計します：

コントローラー (controller)：HTTPリクエストを受け付け、サービスロジックを呼び出します。

サービス層 (service)：ビジネスロジックを処理します。

リポジトリ層 (repository)：データベースとのやり取りを担います。

エンティティモデル (entity)：データベーステーブル構造に基づいたオブジェクト定義を行います。

サンプルパッケージ構造例:

com.example.project

├── controller

├── service

├── repository

├── entity

**3. コーディング規約: Spring Boot**

**3.1 レイヤードアーキテクチャ**

以下の構成を使用します。

コントローラー: HTTPリクエストとレスポンスを処理します。

サービス: ビジネスロジックを実装します。

リポジトリ: データベースとのやり取りを実行します。

コントローラの使用例

@RestController

@RequestMapping("/api/customers") // 基本エンドポイントを指定

public class CustomerController {

private final CustomerService customerService;

@Autowired

public CustomerController(CustomerService customerService) {

this.customerService = customerService;

}

@GetMapping("/{id}")

public ResponseEntity<Customer> getCustomerById(@PathVariable Long id) {

Customer customer = customerService.getCustomerById(id);

return ResponseEntity.ok(customer);

}

}

サービスクラスの使用例

@Service

public class CustomerService {

private final CustomerRepository customerRepository;

@Autowired

public CustomerService(CustomerRepository customerRepository) {

this.customerRepository = customerRepository;

}

public Customer getCustomerById(Long id) {

return customerRepository.findById(id)

.orElseThrow(() -> new EntityNotFoundException("Customer not found"));

}

}

リポジトリクラスの使用例

@Repository

public interface CustomerRepository extends JpaRepository<Customer, Long> {

Optional<Customer> findByEmail(String email);

}

**4. コーディング規約: Thymeleaf**

**4.1 HTMLテンプレート構造**

テンプレートは基本的にsrc/main/resources/templates以下に配置します。

データバインディングの構文例

以下のサンプルコードでは、Thymeleafテンプレート内でサーバーから渡されたデータを動的に表示します。

<!DOCTYPE html>

<html xmlns:th="http://www.thymeleaf.org">

<head>

<title th:text="${pageTitle}">Customer Management</title>

</head>

<body>

<h1 th:text="'Welcome, ' + ${userName}">Welcome, User</h1>

<ul>

<li th:each="customer : ${customers}">

<span th:text="${customer.name}">Customer Name</span> -

<span th:text="${customer.email}">Customer Email</span>

</li>

</ul>

</body>

</html>

**5. コーディング規約: データベースアクセス**

**5.1 SQL**

使用例: Native Query

以下はリポジトリ内でのネイティブクエリの使用です。

@Query(value = "SELECT \* FROM customer WHERE email = :email", nativeQuery = true)

Optional<Customer> findByEmail(@Param("email") String email);

パフォーマンス

必要な場合、インデックスを追加してクエリのパフォーマンスを向上させます。

SELECT文では、必要なカラムのみ指定します（SELECT \* の使用を避ける）。

5.2 Spring Data JPA

エンティティの設計例

@Entity

@Table(name = "customer")

public class Customer {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY) // 自動生成設定

private Long id;

@Column(name = "name", nullable = false)

private String name;

@Column(name = "email", unique = true)

private String email;

// ゲッター、セッター、省略されたコードはこちらに記述

}

リポジトリのメソッド例

以下は Spring Data JPA のメソッドクエリ構文を使用したリポジトリ例です。

List<Customer> findByName(String name);

Optional<Customer> findByEmail(String email);

@Query("SELECT c FROM Customer c WHERE c.name LIKE %:name%")

List<Customer> searchByName(@Param("name") String name);

**6. テストコード例**

**6.1 単体テスト: Controller**

以下は@WebMvcTestを使用したコントローラの単体テスト例です。

@WebMvcTest(CustomerController.class)

public class CustomerControllerTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private CustomerService customerService;

@Test

void testGetCustomerById() throws Exception {

// テストデータ

Customer customer = new Customer(1L, "John Doe", "john@example.com");

Mockito.when(customerService.getCustomerById(1L)).thenReturn(customer);

mockMvc.perform(get("/api/customers/1"))

.andExpect(status().isOk())

.andExpect(jsonPath("$.name").value("John Doe"));

}

}

**7. セキュリティ規約**

Spring Securityを使用し、認証情報は暗号化して保存する。

セッション管理は、可能な限りJWTトークン認証を使用してステートレス設計を採用します。

この規約を基にプロジェクトを進めることで、効率的かつ品質の高い開発が可能になります。必要に応じて規約を追加または修正してください。