Java - Boas Praticas em Desenvolvimento – EducaCiencia FastCode

O Java, como uma das linguagens mais robustas e amplamente adotadas no mundo, oferece um ecossistema vasto e rico.

O sucesso no desenvolvimento em Java depende de boas práticas, desde a organização de código até o uso avançado de frameworks e integrações.

Este artigo apresenta diretrizes do básico ao avançado, com exemplos e tópicos adicionais para enriquecer o conhecimento do desenvolvedor.

1. Fundamentos do Java e Boas Práticas Básicas

1.1 Organização e Estrutura do Código

Uma base bem estruturada é essencial para facilitar o entendimento e a manutenção do código:

- **Nomes significativos:** Use nomes descritivos para classes, métodos e variáveis.
  Exemplo:

```
public class PedidoService {
    public void processarPedido(Pedido pedido) {
        // lógica de processamento
    }
}
```

- **Comentários úteis:** Documente trechos complexos com comentários que expliquem o *porquê*, não apenas o *como*.

1.2 Controle de Fluxo

- Evite estruturas de controle complicadas ou aninhadas demais. Prefira métodos auxiliares

```
public void validarPedido(Pedido pedido) {
    if (pedido == null || pedido.getItens().isEmpty()) {
        throw new IllegalArgumentException("Pedido inválido");
    }
}
```

1.3 Tratamento de Exceções

- **Especificidade:** Capture exceções específicas para melhorar o controle de erros.
- **Recurso try-with-resources:**

```
try (Connection connection = DriverManager.getConnection(url, user, password)) {
    // lógica com o banco de dados
} catch (SQLException e) {
    e.printStackTrace();
}
```

1.4 Modularidade

Divida o código em pacotes organizados. Por exemplo:

- com.empresa.app.service: Contém classes relacionadas à lógica de negócios.
- com.empresa.app.repository: Contém classes para acesso a dados.

2. Intermediário: Arquitetura e Design Patterns

2.1 Princípios SOLID

Adote os princípios SOLID para melhorar a qualidade do design.

- **Exemplo - Dependency Inversion Principle (DIP):**

```
public interface EnvioMensagem {
    void enviar(String mensagem);
}

public class EmailService implements EnvioMensagem {
    public void enviar(String mensagem) {
        System.out.println("Enviando email: " + mensagem);
    }
}

public class Notificacao {
    private final EnvioMensagem envioMensagem;

    public Notificacao(EnvioMensagem envioMensagem) {
        this.envioMensagem = envioMensagem;
    }

    public void notificar(String mensagem) {
        envioMensagem.enviar(mensagem);
    }
}
```

2.2 Design Patterns Comuns

- ***Builder Pattern:*** *Útil para criar objetos complexos.*

```
public class Cliente {
    private final String nome;
    private final int idade;

    private Cliente(Builder builder) {
        this.nome = builder.nome;
        this.idade = builder.idade;
    }

    public static class Builder {
        private String nome;
        private int idade;

        public Builder nome(String nome) {
            this.nome = nome;
            return this;
        }

        public Builder idade(int idade) {
            this.idade = idade;
            return this;
        }

        public Cliente build() {
            return new Cliente(this);
        }
    }
}

Cliente cliente = new Cliente.Builder().nome("João").idade(30).build();
```

- **Factory Method:** Simplifica a criação de objetos com variações.

3. Frameworks Populares no Ecossistema Java

3.1 Spring Framework e Spring Boot

O Spring simplifica o desenvolvimento Java moderno.

- **Exemplo - Injeção de Dependência com IoC:**

```
@Service
Public class PedidoService {
    Private final PedidoRepository repository;

    Public PedidoService(PedidoRepository repository) {
        This.repository = repository;
```

```
    }

    Public void salvarPedido(Pedido pedido) {
        Repository.save(pedido);
    }
}
```

- **Spring Data JPA:**
  Simplifica operações com bancos de dados.

  ```
  public interface ClienteRepository extends JpaRepository<Cliente, Long> {
      List<Cliente> findByNome(String nome);
  }
  ```

3.2 Hibernate (JPA)

- **Mapeamento Objeto-Relacional:**

  ```
  @Entity
  public class Produto {
      @Id
      @GeneratedValue(strategy = GenerationType.IDENTITY)
      private Long id;

      private String nome;

      private double preco;

      // Getters e setters
  }
  ```

3.3 Apache Maven e Gradle

Ferramentas para gerenciamento de dependências e *builds*:

- **Maven - Exemplo de `pom.xml`:**

  ```
  xml
  <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
      <version>3.0.0</version>
  </dependency>
  ```

- **Gradle - Exemplo de `build.gradle`:**

  ```
  groovy
  dependencies {
      implementation 'org.springframework.boot:spring-boot-starter-web:3.0.0'
  }
  ```

4. Avançado: Integrações, Testes e Monitoramento

4.1 Integração Contínua e Entrega Contínua

- Configure pipelines com **Jenkins**, **GitHub Actions** ou **GitLab CI/CD**.

```
yaml
name: Build and Test

on: [push]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Set up JDK 17
        uses: actions/setup-java@v3
        with:
          java-version: 17
      - name: Build with Gradle
        run: ./gradlew build
```

4.2 Testes Automatizados

- **Testes Unitários com JUnit:**

```
@Test
void deveCalcularTotal() {
    Pedido pedido = new Pedido();
    pedido.adicionarItem(new Item("Produto A", 2, 50.0));
    pedido.adicionarItem(new Item("Produto B", 1, 30.0));

    assertEquals(130.0, pedido.calcularTotal());
}
```

- **Mocks com Mockito:**

```
@Mock
private PedidoRepository repository;

@Test
void deveSalvarPedido() {
    Pedido pedido = new Pedido();
    when(repository.save(any())).thenReturn(pedido);

    Pedido resultado = service.salvarPedido(pedido);

    verify(repository).save(pedido);
    assertNotNull(resultado);
}
```

4.3 APIs RESTful e Segurança

- **Exemplo com Spring Security:**

```
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
            .antMatchers("/admin/**").hasRole("ADMIN")
            .anyRequest().authenticated()
            .and().formLogin();
    }
}
```

4.4 Monitoramento

- Utilize **Prometheus** e **Grafana** para monitorar métricas.
- **Actuators no Spring Boot:**

```
yaml
management:
  endpoints:
    web:
      exposure:
        include: "*"
```

Ao aplicar boas práticas desde o básico até o avançado, o desenvolvedor Java pode criar aplicações escaláveis, seguras e de alta qualidade.

O domínio de frameworks como Spring e Hibernate, aliado a ferramentas modernas de CI/CD e monitoramento, prepara o desenvolvedor para os desafios do mercado atual.

EducaCiência FastCode para a comunidade