

FIA/P GRADUAÇÃO

# ENTERPRISE APPLICATION DEVELOPMENT

Prof. Me. Thiago T. I. Yamamoto

#04 - DESIGN PATTERNS E JUNIT

# TRAJETÓRIA

---



JPA Introdução



JPA API



Design Patterns e JUnit

## #04 - AGENDA

---

- Design Patterns
- Singleton
- DAO Genérico
- JUnit





# DESIGN PATTERNS

- Padrão de solução para problemas **repetitivos**;
- Constitui um poderoso instrumento que baseado na orientação a objetos podem maximizar a **qualidade e a produtividade de software**;
- Permite criar **aplicações robustas** com **soluções já testadas** e minimizar o impacto de alterações durante o desenvolvimento;





# SINGLETON

- Padrão que permite uma **única instância** de um objeto dentro da aplicação;
- Possui um **atributo estático** (que será único), **um construtor privado** e o método estático **getInstance()**, que retorna o valor do atributo estático;
- O método valida **se já existe uma instância** no atributo e caso não exista, um **novo objeto** é instanciado;

```
public class ObjectSingleton {  
    private static Object unico;  
    private ObjectSingleton(){}  
    public static Object getInstance(){  
        if (unico == null){  
            unico = new Object();  
        }  
        return unico;  
    }  
}
```








# DAO GENÉRICO

- **Generics** é uma funcionalidade incorporada ao Java na versão 5.0;
- Permite uma **verificação** *typed-safety* em **tempo de compilação**, ou seja, confirma se o que está sendo **atribuído** a uma **instância** está de acordo com o **especificado**;

```
List<String> lista = new ArrayList<String>();
```

A interface **List** e a classe **ArrayList** utilizam generics para determinar o tipo de objeto que será armazenado na coleção;

```
public class TesteGenerico<T> {  
    public T teste(T objeto) { ... }  
}
```



```
TesteGenerico<Calendar> t = new TesteGenerico<Calendar>();  
Calendar c = t.teste(Calendar.getInstance());
```

A classe é declarada deixando a definição do tipo de dado genérico **T** para o momento da instanciação dos objetos.

A classe **TesteGenerico** após instanciada com a declaração **TesteGenerico<Calendar>**:

```
public class TesteGenerico<Calendar> {  
    public Calendar teste(Calendar objeto) { ... }  
}
```

- Os **mesmos métodos** do Entity Manager **são utilizados** para realizar as **operações básicas** (CRUD) com qualquer Entidade com JPA, a única mudança é o **próprio objeto da entidade**;
- Dessa forma, é possível criar um **DAO** que possa **ser reutilizado** (via herança) para as operações básicas de persistência, para todas as entidades;

## Passos para criar um DAO Genérico:

1. Criar uma **interface** para definir as **funções básicas** do DAO genérico;
2. Criar uma **classe (abstrata)** que **implementa a interface** para desenvolver as funcionalidades;
3. Para cada DAO do seu sistema (ClienteDAO, ProdutoDAO, etc...), crie uma **interface** e estenda da interface genérica criada em 1, informando a classe e o tipo da chave primária;
4. Crie uma **classe** para a interface criada em 3, estenda da classe do DAO genérico e implemente a interface, Pronto!

- Para obter qual o **tipo de dado genérico** passado como parâmetro para o DAO utilize o código abaixo:

```
this.classePersistencia = (Class) ((ParameterizedType) getClass()  
.getGenericSuperclass()).getActualTypeArguments()[0];
```

- Os métodos de negócio **específicos** da entidade devem ser implementados na classe **DAO filha** e não na DAO genérica!
- Aqui apresentaremos um **DAO Genérico básico**, apenas com o CRUD. Use a imaginação para agregar mais métodos genéricos!



# JUNIT

- É um **framework** *opensource* para criação de **testes automatizados** na linguagem de programação **Java**;
- **Testes unitários:** automáticos, repetíveis e rápidos;
- Vamos utilizar esse framework para realizar os **testes dos DAOs** que foram implementados;

*"Aqui está funcionando..."*

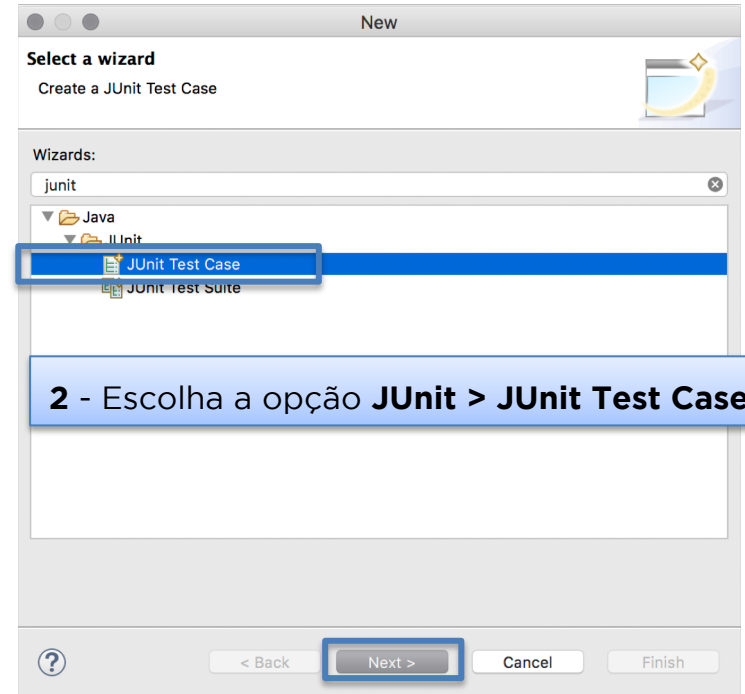
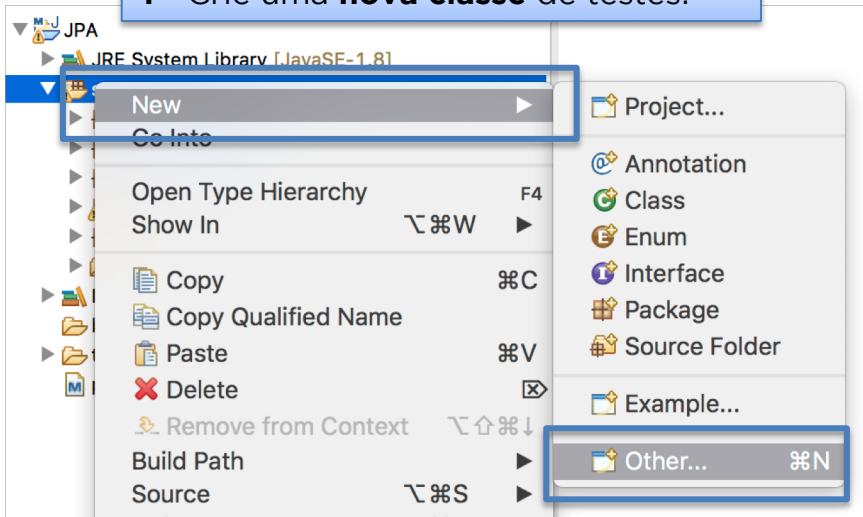
*"Estranho, isso não era para acontecer..."*

*"Nossa, como você conseguiu fazer isso?"*

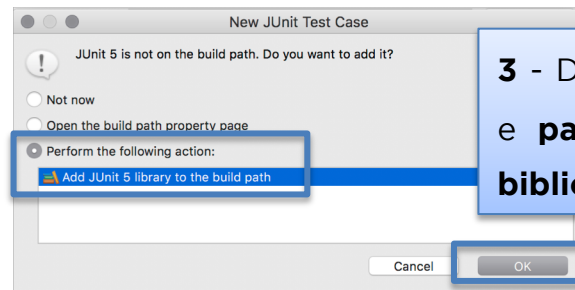


- Crie uma **classe de testes**:

1 - Crie uma **nova classe** de testes:



2 - Escolha a opção **JUnit > JUnit Test Case**



3 - Depois de configurar o **nome** e **pacote** da classe, adicione a **biblioteca** do **JUnit** no projeto;



- Para realizar os testes podemos utilizar um **banco de dados específico para testes**;
- Vamos utilizar um **banco H2** que é em **memória** e muito utilizado no **momento do desenvolvimento**;
- Para isso, vamos adicionar a dependência no **pom.xml**;

```
<dependency>  
  <groupId>com.h2database</groupId>  
  <artifactId>h2</artifactId>  
  <version>1.4.197</version>  
  <scope>test</scope>  
</dependency>
```

- Outra modificação necessária é adicionar uma **nova unidade de persistência** no arquivo **persistence.xml**, para configurar o novo banco de dados;

```
<persistence-unit name="teste" transaction-type="RESOURCE_LOCAL">
  <properties>
    <property name="javax.persistence.jdbc.driver" value="org.h2.Driver" />
    <property name="javax.persistence.jdbc.url" value="jdbc:h2:mem:test" />
    <property name="javax.persistence.jdbc.user" value="sa" />
    <property name="javax.persistence.jdbc.password" value="" />

    <property name="hibernate.dialect" value="org.hibernate.dialect.H2Dialect" />
    <property name="hibernate.hbm2ddl.auto" value="create-drop" />
    <property name="show_sql" value="true" />
    <property name="hibernate.temp.use_jdbc_metadata_defaults" value="false" />
  </properties>
</persistence-unit>
```

- **Cada teste** deve realizar **implementar** as seguintes **fases**:
  - **Arrange** – instanciação dos objetos que serão utilizados no teste;
  - **Act** – realiza a ação (chamada do método) que será testado;
  - **Assert** – validação do resultado da ação, validando se o teste passou ou não;



- As principais **anotações** que serão utilizadas nos testes:

Anotação	Descrição
<b>@Test</b>	Determina que o método é um método de teste.
<b>@DisplayName</b>	Configura um nome para a classe ou método de teste.
<b>@BeforeAll</b>	Método será executado antes de todos os testes.
<b>@BeforeEach</b>	Método será executado antes de cada teste.
<b>@Nested</b>	Testes aninhados, permitindo que uma classe fique dentro de outra classe de teste.

<https://junit.org/junit5/docs/snapshot/user-guide/>

- Os principais **métodos** para validar os testes:

Assertion	Descrição
<b>assertEquals()</b>	Valor atual e esperado são iguais;
<b>assertNotEquals()</b>	Valor atual e esperado são diferentes;
<b>assertNull()</b>	Valor atual é nulo;
<b>assertNotNull()</b>	Valor atual não é nulo;
<b>assertTrue()</b>	Valor atual é verdadeiro;
<b>assertFalse()</b>	Valor atual é falso;
<b>assertThrows()</b>	Espera pelo retorno da <i>exception</i> ;
<b>fail()</b>	Falha o teste;

# VOCE APRENDEU..

---



- O que são os **padrões de projetos**;
- Implementar o padrão **singleton** para manter somente um objeto da fabrica de entity manager;
- Desenvolver um **DAO genérico** para as operações básicas (CRUD), evitando a repetição de código;
- Realizar testes unitários utilizando **JUnit**;

**Copyright © 2013 – 2019**  
**Prof. Me. Thiago T. I. Yamamoto**

Todos direitos reservados. Reprodução ou divulgação total ou parcial deste documento é expressamente proibido sem o consentimento formal, por escrito, do Professor (autor).

*“Ter sucesso é falhar repetidamente, mas sem  
perder o entusiasmo”*