Teste de Software - JUnit 5 - Ambiente de testes

👣 Passo 01 - Criar os Métodos Construtores na Classe Usuario (Camada Model)

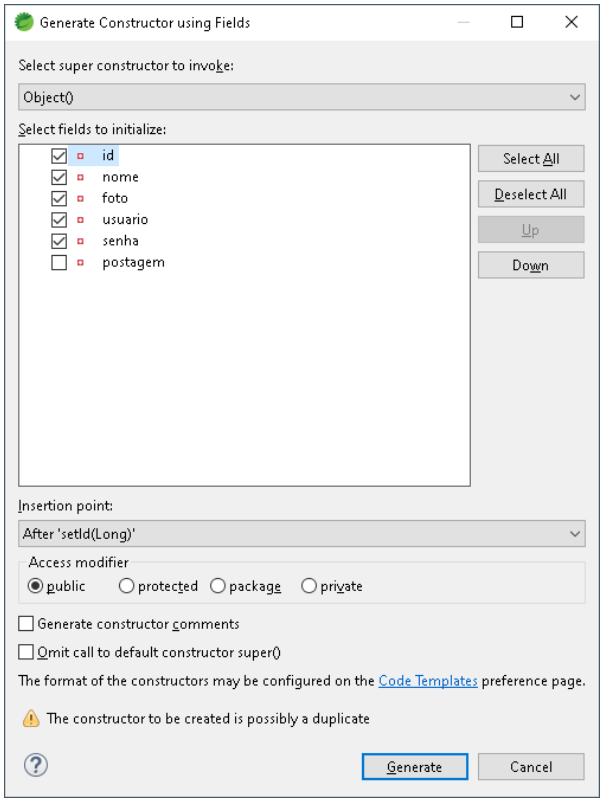
Na **Classe Usuario**, na camada Model, vamos criar 2 métodos construtores: o primeiro com todos os atributos (exceto o atributo postagens, que tem a função de listar as postagens associadas ao usuário, logo é um atributo preenchido automaticamente pelo Relacionamento entre as Classes) e um segundo método construtor vazio, ou seja, sem atributos como mostra o trecho de código abaixo. Através destes dois métodos iremos instanciar alguns objetos da Classe Usuario nas nossas classes de teste.

```
    public Usuario(Long id, String nome, String foto, String usuario, String
senha) {
        this.id = id;
        this.nome = nome;
        this.foto = foto;
        this.usuario = usuario;
        this.senha = senha;
    }

    public Usuario() { }
```

1. Para criar o Primeiro Construtor, posicione o cursor após o último atributo da Classe (em nosso exemplo postagem) e clique no menu **Source → Generate Constructor using fields**.

2. Na janela **Generate Constructor using fields**, selecione todos os atributos, exceto postagem e marque a opção **Omit call to default constructor super()** como mostra a figura abaixo:



3. Clique no botão **Generate** para concluir. O Construtor será gerado com todas as anotações nos parâmetros, como mostra a figura abaixo:

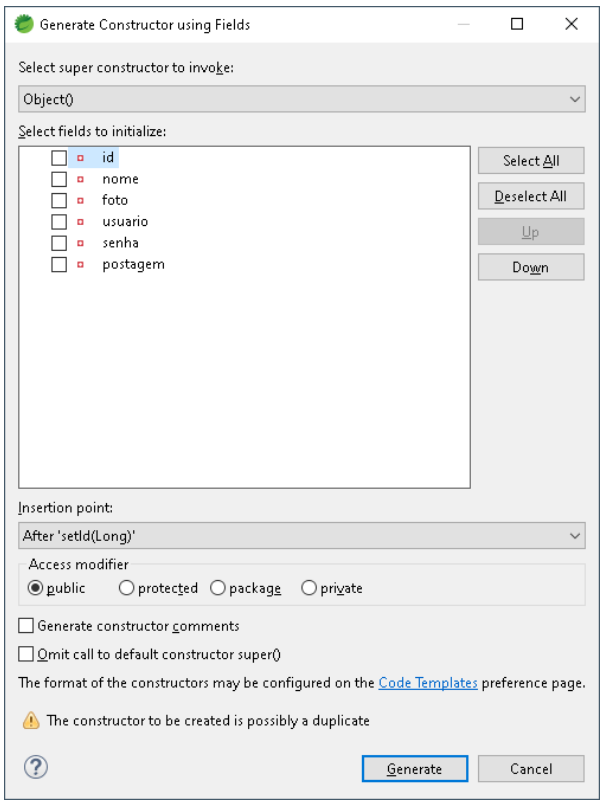
```
47
48    public Usuario(Long id,
49                   @NotNull(message = "O atributo Nome é Obrigatório!") String nome,
50                   @Size(max = 5000, message = "O link da foto não pode ser maior do que 5000 caracteres") String foto,
51                   @NotNull(message = "O atributo Usuário é Obrigatório!")
52                   @Email(message = "O atributo Usuário deve ser um email válido!") String usuario,
53                   @NotBlank(message = "O atributo Senha é Obrigatório!")
54                   @Size(min = 8, message = "A Senha deve ter no mínimo 8 caracteres") String senha) {
55        this.id = id;
56        this.nome = nome;
57        this.foto = foto;
58        this.usuario = usuario;
59        this.senha = senha;
60    }
61
```

4. Apague todas as anotações dos parâmetros do Método Construtor. O Método ficará igual ao trecho de código abaixo:

```
    public Usuario(Long id, String nome, String foto, String usuario, String
senha) {
        this.id = id;
        this.nome = nome;
        this.foto = foto;
        this.usuario = usuario;
        this.senha = senha;
    }
```

Agora vamos criar o segundo Método Construtor:

1. Posicione o cursor após o Método Construtor com parâmetros e clique no menu **Source → Generate Constructor using fields**.
2. Na janela **Generate Constructor using fields**, desmarque todos os atributos e marque a opção **Omit call to default constructor super()** como mostra a figura abaixo:



3. Clique no botão **Generate** para concluir.
4. O construtor vazio ficará igual a imagem abaixo:

```
    public Usuario() {  }
```

Código fonte: Usuario.java

Passo 02 - Atualizar a Interface UsuarioRepository (Camada Repository)

Na Interface UsuarioRepository, na camada Repository, vamos criar o método **findAllByNomeContainingIgnoreCase(String nome)** para efetuar alguns testes na Camada Repository. O código da Interface UsuarioRepository atualizado ficará assim:

```
@Repository
public interface UsuarioRepository extends JpaRepository<Usuario, Long> {

    public Optional<Usuario> findByUsuario(String usuario);

    public List <Usuario> findAllByNomeContainingIgnoreCase(String nome);
}
```

Código fonte: UsuarioRepository.java

Passo 03 - Configurações iniciais

1. Configurar a Dependência Spring Testing

Vamos Configurar a Dependência Springboot Starter Test para aceitar apenas o JUnit 5. No arquivo, **pom.xml**, vamos alterar a dependência Springboot Starter Test conforme o código abaixo:

```
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
        <exclusions>
            <exclusion>
                <groupId>org.junit.vintage</groupId>
                <artifactId>junit-vintage-engine</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
```

*Essa alteração irá ignorar as versões anteriores ao **JUnit 5** (vintage).

2. Adicionar a Dependência do Banco de Dados H2

Para utilizar o Banco de Dados H2 no seu projeto será necessário inserir a Dependência no seu arquivo **pom.xml**. No arquivo, **pom.xml**, vamos adicionar as linhas abaixo:

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
```

*Sugerimos adicionar esta dependência logo abaixo da dependência do MySQL.

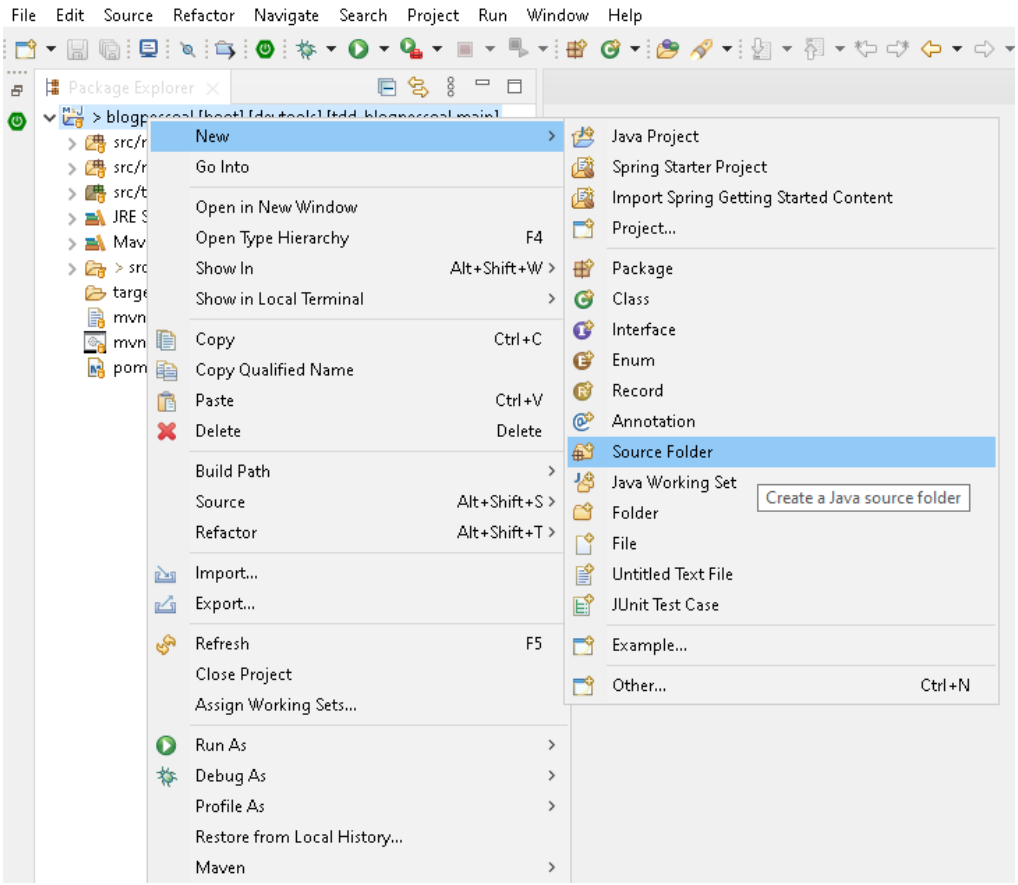
Código fonte: pom.xml

ALERTA DE BSM: ***Mantenha a atenção aos detalhes nos próximos passos. Até o Passo 03, todas ações foram realizadas dentro da Source Folder Principal (src/main/java e src/main/resources). A partir do Passo 04, todas as ações serão efetuadas dentro da Source Folder de Testes (src/test/java e src/test/resources).***

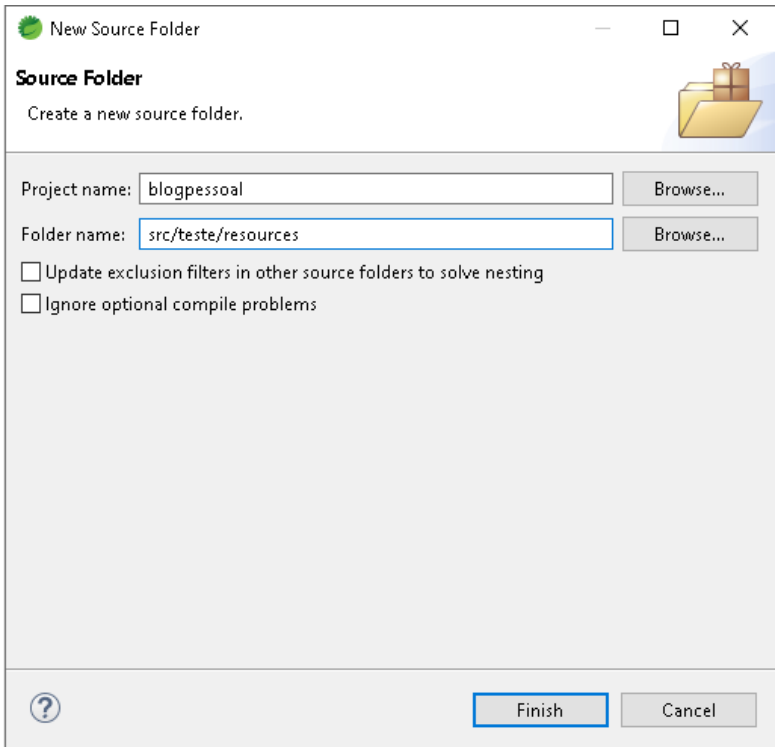
Passo 04 - Configurar o Banco de dados H2

Agora vamos configurar um Banco de dados para executar os testes para não usar o Banco de dados principal da aplicação. Como não temos em nosso projeto a **Source Folder resources**, dentro da **Source Folder src/test**, vamos cria-la e na sequência inserir o arquivo application.properties para configurarmos o Banco de dados de testes (H2). Vamos utilizar nos testes o Banco de dados H2 porque não precisaremos persistir os dados dos testes após a sua conclusão.

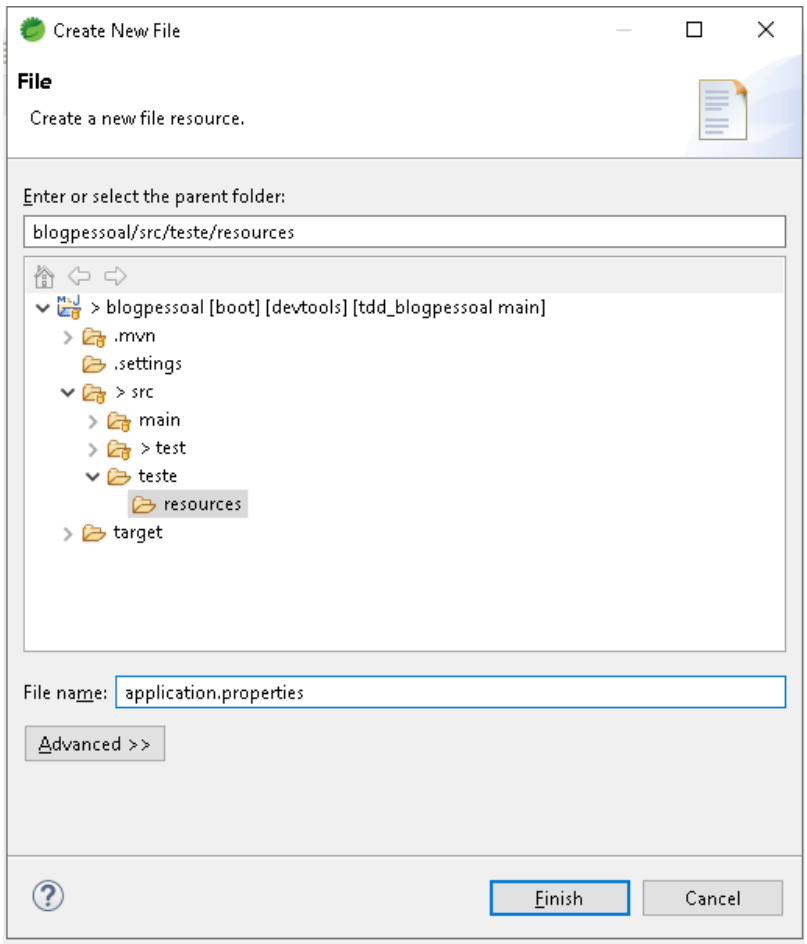
1. No lado esquerdo superior, na Guia **Package Explorer**, clique sobre a pasta do projeto com o botão direito do mouse e clique na opção **New → Source folder**



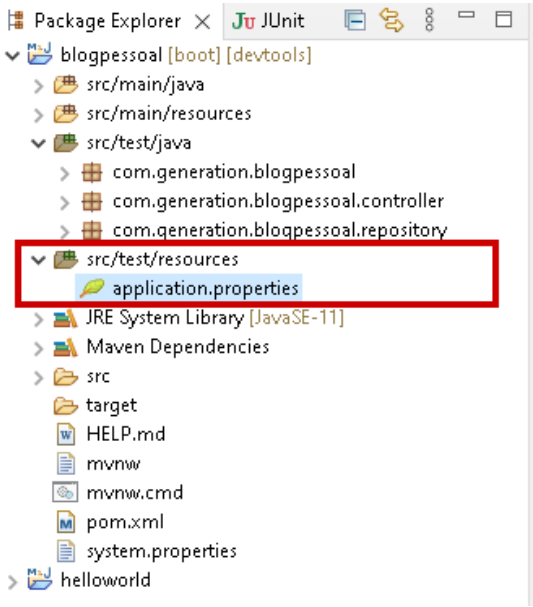
2. Em **Source Folder**, no item **Folder name**, informe o caminho como mostra a figura abaixo (**src/test/resources**), e clique em **Finish**:



3. No lado esquerdo superior, na Guia **Package explorer**, na Source Folder **src/test/resources**, clique com o botão direito do mouse e clique na opção **New → File**.
4. Em File name, digite o nome do arquivo (**application.properties**) e clique em **Finish**.



6. Veja o arquivo criado na **Package Explorer**



7. Insira no arquivo **application.properties** criado em **src/test/resources** o código abaixo, para configurar o Banco de dados H2:

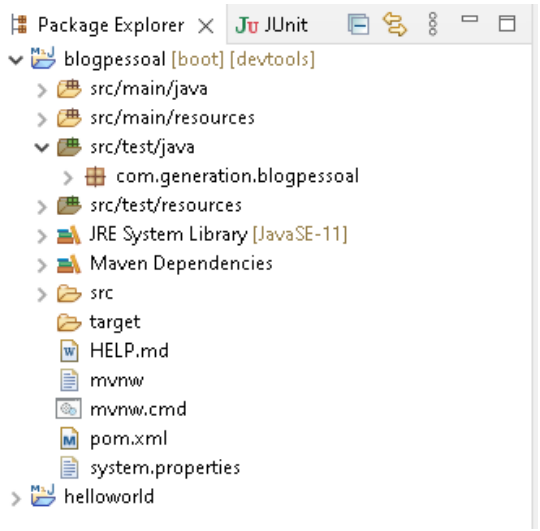
```
spring.datasource.url=jdbc:h2:mem:db_blogpessoal_test
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=sa
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
```

Linha	Descrição
spring.datasource.url	Define o nome do Banco de dados de teste (db_blogpessoal_test)
spring.datasource.driverClassName	Define o Driver do Banco de dados (H2)
spring.datasource.username	Define o usuário do H2 (sa)
spring.datasource.password	Define a senha do usuário do H2 (sa)
spring.jpa.database-platform	Configura o tipo do Banco de dados (H2).

Código fonte: application.properties (src/test/resources)

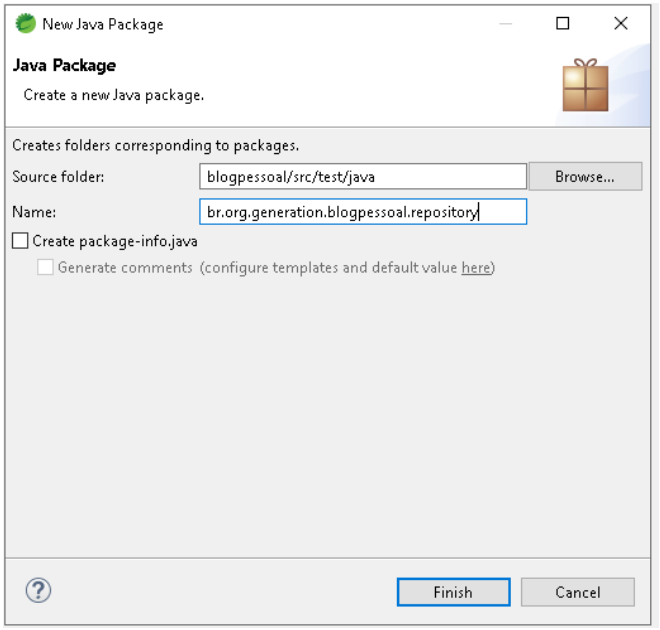
Passo 05 - Criar os pacotes em src/test/java

Na Source Folder de Testes (**src/test/java**), observe que existe a mesma estrutura de pacotes da Source Folder Principal (**src/main/java**).



Vamos criar em **src/test/java** as packages **Repository** e **Controller**:

1. No lado esquerdo superior, na Guia **Package explorer**, clique com o botão direito do mouse sobre a Package **com.generation.blogpessoal**, na Source Folder **src/test/java** e clique na opção **New → Package**.
2. Na janela **New Java Package**, no item **Name**, acrescente no final do nome da Package **.repository**, como mostra a figura abaixo:



3. Clique no botão **Finish** para concluir.
4. **Repita os passos 1-3** para criar a **Package .controller**