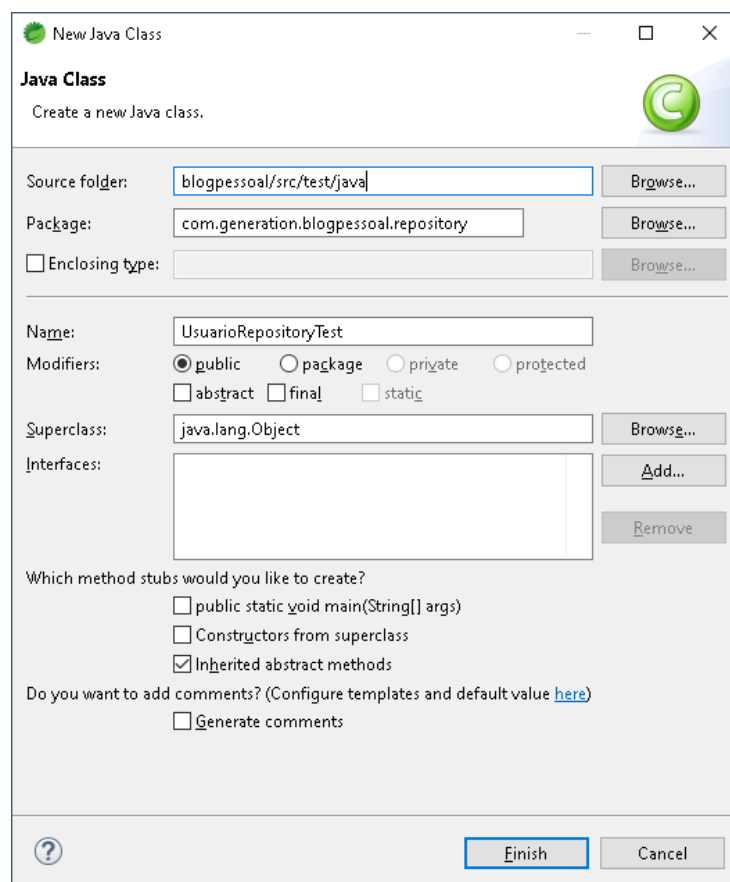


Teste de Software - JUnit 5 - Teste da Interface Repository

Passo 06 - Criar os Testes

A Classe **UsuarioRepositoryTest** será utilizada para testar a Classe Repository do Usuario.

1. No lado esquerdo superior, na Guia **Package Explorer**, clique com o botão direito do mouse sobre a Package **com.generation.blogpessoal.repository**, na Source Folder **src/test/java** e clique na opção **New → Class**.
2. Na janela **New Java Class**, no item **Name**, informe o nome da classe que será o mesmo nome da Classe Principal (**UsuarioRepository**) + a palavra **Test**, para indicar que se trata de uma Classe de Testes, ou seja, **UsuarioRepositoryTest**, como mostra a figura abaixo:



The screenshot shows the 'New Java Class' dialog box. The 'Name' field contains 'UsuarioRepositoryTest'. The 'Package' field contains 'com.generation.blogpessoal.repository'. The 'Source folder' field contains 'blogpessoal/src/test/java'. The 'Superclass' field contains 'java.lang.Object'. The 'Modifiers' section has 'public' selected. The 'Which method stubs would you like to create?' section has 'Inherited abstract methods' checked. The 'Do you want to add comments?' section has 'Generate comments' unchecked. The 'Finish' button is highlighted.

3. Clique no botão **Finish** para concluir.



IMPORTANTE: Toda a Classe de teste deve ter no final do seu nome a palavra **Test**.



ATENÇÃO: O Teste da Classe **UsuarioRepository**, na camada **Repository**, utiliza o **Banco de Dados**, entretanto ele não criptografa a senha ao gravar um novo usuário no Banco de dados porque ele não utiliza a Classe de Serviço **UsuarioService**, ele utiliza o método **save()**, da Interface **JpaRepository** de forma direta.

UsuarioRepositoryTest

```
23
24 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
25 @TestInstance(TestInstance.Lifecycle.PER_CLASS)
26 public class UsuarioRepositoryTest {
27
28     @Autowired
29     private UsuarioRepository usuarioRepository;
30
31     @BeforeAll
32     void start(){
33
34         usuarioRepository.deleteAll();
35
36         usuarioRepository.save(new Usuario(0L, "João da Silva", "https://i.imgur.com/h4t81oa.jpg", "joao@email.com.br", "13465278"));
37
38         usuarioRepository.save(new Usuario(0L, "Manuela da Silva", "https://i.imgur.com/NtyGneo.jpg", "manuela@email.com.br", "13465278"));
39
40         usuarioRepository.save(new Usuario(0L, "Adriana da Silva", "https://i.imgur.com/5M2p5Mb.jpg", "adriana@email.com.br", "13465278"));
41
42         usuarioRepository.save(new Usuario(0L, "Paulo Antunes", "https://i.imgur.com/FETvs20.jpg", "paulo@email.com.br", "13465278"));
43
44     }
45 }
```

Na **linha 24** a anotação **@SpringBootTest** indica que a Classe `UsuarioRepositoryTest` é uma Classe Spring Boot Testing. A Opção **environment** indica que caso a porta principal (8080 para uso local) esteja ocupada, o Spring irá atribuir uma outra porta automaticamente.

Na **linha 25** a anotação **@TestInstance** indica que o Ciclo de vida da Classe de Teste será por Classe.

Nas **linhas 28 e 29** foi injetado (**@Autowired**), um objeto da Interface `UsuarioRepository` para persistir os objetos no Banco de dados de testes.

Entre as **linhas 31 e 42**, o método **start()**, anotado com a anotação **@BeforeAll**, apaga todos os dados da tabela (linha 34), inicializa 4 objetos do tipo `Usuario` e insere no Banco de dados de testes através do método **.save()** uma única vez (**Lifecycle.PER_CLASS**). Observe que em todos os Objetos, o atributo `id` está com o valor `0L`, indicando que o atributo será preenchido automaticamente pelo Banco de dados (substituindo o zero pela Chave primária) e o `L` informa que o atributo é um Objeto da Classe `Long`.



[Documentação: @SpringBootTest](#)



[Documentação: @TestInstance](#)



[Documentação: Lifecycle](#)



[Documentação: @BeforeAll](#)

Método 01 - Retornar um Usuário

```
45
46     @Test
47     @DisplayName("Retorna 1 usuario")
48     public void deveRetornarUmUsuario() {
49
50         Optional<Usuario> usuario = usuarioRepository.findByUsuario("joao@email.com.br");
51         assertTrue(usuario.get().getUsuario().equals("joao@email.com.br"));
52     }
53 }
```

Na **linha 46**, o Método **deveRetornarUmUsuario()** foi anotado com a anotação **@Test** que indica que este método executará um teste.

Na **linha 47**, a anotação **@DisplayName** configura uma mensagem que será exibida ao invés do nome do método

Na **linha 50**, o objeto **usuario** recebe o resultado do método **findByUsuario()**.

Na **linha 51**, através do método de asserção **assertTrue()**, verifica se o usuário cujo e-mail é "joao@email.com.br" foi encontrado. Se o e-mail for encontrado o resultado do teste será: **Aprovado!**. Caso não encontre, o resultado do teste será : **Falhou!**.

 [Documentação: @Test](#)

 [Documentação: @DisplayName](#)

 [Documentação: assertTrue](#)

Método 02 - Retornar três Usuários

```
53
54     @Test
55     @DisplayName("Retorna 3 usuarios")
56     public void deveRetornarTresUsuarios() {
57
58         List<Usuario> listaDeUsuarios = usuarioRepository.findAllByNomeContainingIgnoreCase("Silva");
59         assertEquals(3, listaDeUsuarios.size());
60         assertTrue(listaDeUsuarios.get(0).getNome().equals("João da Silva"));
61         assertTrue(listaDeUsuarios.get(1).getNome().equals("Manuela da Silva"));
62         assertTrue(listaDeUsuarios.get(2).getNome().equals("Adriana da Silva"));
63
64     }
65
```

Na **linha 54**, o Método **deveRetornarTresUsuarios()** foi anotado com a anotação **@Test** que indica que este método executará um teste.

Na **linha 55**, a anotação **@DisplayName** configura uma mensagem que será exibida ao invés do nome do método

Na **linha 58**, o objeto **listaDeUsuarios** recebe o resultado do método **findAllByNomeContainingIgnoreCase()**.

Na **linha 59**, através do método de asserção **assertEquals()**, verifica se o tamanho da List é igual a 3 (quantidade de usuários cadastrados no método **start()** cujo sobrenome é "**Silva**"). O método **size()**, (java.util.List), retorna o tamanho da List. Se o tamanho da List for igual 3, o 1º teste será **Aprovado!**.

Nas **linhas 54 a 56**, através do método de asserção **AssertTrue()**, verifica em cada posição da Collection List **listaDeUsuarios** se os usuários, que foram inseridos no Banco de dados através no método **start()**, foram gravados na mesma sequência.

- O Teste da **linha 60** checará se o primeiro usuário inserido (João da Silva) está na posição 0 da List **listaDeUsuarios** (1ª posição da List),
- O Teste da **linha 61** checará se o segundo usuário (Manuela da Silva) está na posição 1 da List **listaDeUsuarios** (2ª posição da List).
- O Teste da **linha 62** checará se o terceiro usuário (Adriana da Silva) está na posição 2 da List **listaDeUsuarios** (3ª posição da List).

A posição na List é obtida através do método **get(int index)** (java.util.List), passando como parâmetro a posição desejada. O nome do usuário é obtido através do método **getNome()** da Classe Usuario. Se os três usuários foram gravados na mesma sequência do método **start()**, os três testes serão **Aprovados!**.

 [Documentação: assertEquals](#)



[Documentação: Collection List](#)



[Documentação: Collection List - Método Size\(\)](#)



[Documentação: Collection List - Método get\(int index\)](#)



IMPORTANTE: cuidado para não confundir o método `get(int index)` do pacote `java.util.List` com o método `get()` do pacote `java.util.Optional`.



ATENÇÃO: *Para que o método `deveRetornarTresUsuarios()` seja aprovado, os 4 testes (linhas 59 a 62) devem ser aprovados, caso contrário o JUnit indicará que o teste Falhou!.*



DESAFIO: Faça algumas alterações nos dados dos objetos e/ou escreva outros testes para praticar. A melhor forma de aprender e compreender como funcionam os testes é praticando!



[Código fonte: `UsuarioRepositoryTest.java`](#)