

Trabalho 3 POO

Giovana Fernandes Nascimento, Ana Beatriz Sureke Buese Henrique e Diego Camargo

Testes Ana Beatriz:

ID do Teste	Objetivo do Teste	Código
1	<p>Verifica se a inserção está funcionando correta da seguinte forma: insere um conto e um herói, e então, procura o herói relacionado ao conto.</p> <p>O objetivo é verificar se este herói não está vazio.</p>	<pre> public void testInsercao() throws SQLException { ContoDAO contoDAO = new ContoDAO(); HeroiDAO heroiDAO = new HeroiDAO(); Heroi heroi = new Heroi(); heroi.setName("heroi"); heroi.setId(4); heroi.setDescricao("heroi"); heroi.setUrl("nao ha"); heroiDAO.insere(heroi); Conto conto = new Conto(); conto.setDescricao("conto"); conto.setName("conto"); conto.setNameHeroi("heroi"); contoDAO.insere(conto); open(); Heroi heroi2 = new Heroi(); heroi2 = heroiDAO.buscaPorNome(conto.getNameHeroi()); sql = "SELECT * FROM conto WHERE heroiid = ?"; statement = connection.prepareStatement(sql); statement.setInt(parameterIndex 1, heroi.getId()); result = statement.executeQuery(); assertNotNull(heroi2.getId()); } </pre>
2	<p>Verifica se o resultado de uma busca por um conto não existente possui resultado nulo, como é esperado.</p>	<pre> @Test public void TestaBusca() throws SQLException { ContoDAO contoDAO = new ContoDAO(); Conto conto = new Conto(); conto = contoDAO.buscaPorNome(nomeConto: "nao ha"); assertNull(conto); } </pre>
3	<p>Verifica se um id de um herói mais antigo é menor que o id de um herói mais novo</p>	<pre> @Test public void TestaIdEmInsercao() throws SQLException { HeroiDAO heroiDao = new HeroiDAO(); Heroi heroi = new Heroi(); heroi.setName("teste"); heroi.setDescricao("testando"); heroi.setUrl("nao ha"); heroiDao.insere(heroi); Heroi heroi2 = new Heroi(); heroi2.setName("teste2"); heroi2.setDescricao("testando2"); heroi.setUrl("nao"); heroiDao.insere(heroi2); Heroi heroi3 = new Heroi(); Heroi heroi4 = new Heroi(); heroi3 = heroiDao.buscaPorNome(heroi.getName()); heroi4 = heroiDao.buscaPorNome(heroi2.getName()); assertEquals(heroi3.getId() < heroi4.getId()); heroiDao.remove(heroi3.getId()); heroiDao.remove(heroi4.getId()); } </pre>
4	<p>Verifica se a atualização está funcionando corretamente, verificando se um id continua o mesmo após atualização.</p>	<pre> @Test public void TestAtualiza() throws SQLException{ HeroiDAO heroiDAO = new HeroiDAO(); Heroi heroi = new Heroi(); heroi.setName("testan"); heroi.setDescricao("testara"); heroi.setUrl("nao"); heroiDAO.insere(heroi); heroi = heroiDAO.buscaPorNome(heroi.getName()); heroi.setName("aaaaa"); Heroi heroi2 = new Heroi(); heroi2 = heroiDAO.buscaPorNome(nomeHeroi: "testan"); heroiDAO.atualiza(heroi); heroi = heroiDAO.buscaPorNome(nomeHeroi: "aaaaa"); assertEquals(heroi.getId(), heroi2.getId()); heroiDAO.remove(heroi.getId()); } </pre>

Testes Diego:

ID do Teste	Objetivo do Teste	Código
1	Testa a criação do usuário passando como parâmetro o "userName" e o "password" verificando se realmente foi criado.	<pre> public void TestGetuser(){ User usuario = new User(userName: "IHWA", password: "YEON"); assertNotNull(usuario.getUserName()); assertNotNull(usuario.getPassword()); } </pre>
2	Testa a alteração do nome e da senha de um usuário já criado e com nome e senha anteriormente definidos	<pre> @Test @diegocamargo55555 * public void TestGetuser(){ User usuario = new User(userName: "MOMO", password: "MONO"); usuario.setUserName("NHY"); usuario.setPassword("Fiona Heylon"); assertEquals(expected: "NHY", usuario.getUserName()); assertEquals(expected: "Fiona Heylon", usuario.getPassword()); } </pre>
3	Testa a inserção de um userName no banco de dados, depois consulta o banco buscando o nome inserido e atribui esse nome a um novo usuário o qual deve ser o mesmo que foi inserido	<pre> @Test @diegocamargo55555 * public void Testgetinsere() throws SQLException { UserDao u = new UserDao(); User usuario = new User(userName: "ihwa", password: "YEON"); User userTest = new User(); u.remove(id: "ihwa"); u.insere(usuario); open(); sql = "SELECT UserID FROM usuario WHERE UserID = 'ihwa'"; statement = connection.prepareStatement(sql); result = statement.executeQuery(); Heroi h = new Heroi(); if (result.next()) { userTest.setUserName(result.getString(columnLabel: "UserID")); // Se a coluna for uma String } close(); assertEquals(expected: "ihwa", userTest.getUserName()); } </pre>
4	verificar se inicialmente o Session Cookie é nulo, depois ver se está recebendo o userName	<pre> @Test new * public void TestGetuser(){ User usuario = new User(userName: "beastars", password: "LOUIS"); assertNull(Session.getCookie()); usuario.setUserName("Livia Priscilla"); usuario.setPassword("Abel Heylon"); Session.setCookie(usuario.getUserName()); assertEquals(Session.getCookie(), usuario.getUserName()); assertSame(Session.getCookie(), usuario.getUserName()); } </pre>

Método	Descrição	Teste passa se
assertEquals(a,b)	Compara dois valores	a.equals(b)
assertFalse(a)	Avalia uma expressão booleana	a == false
assertTrue(a)		a == true
assertNotNull(a)	Compara uma variável com nulo	a != null
assertNull(a)		a == null
assertNotSame(a,b)	Compara dois objetos	a == b
assertSame(a,b)		a != b
fail()	Causa uma falha no teste atual	

Testes Giovana:

ID do Teste	Objetivo do Teste	Código
1	Verificar se o retorno do ID está correto, então a validação é feita e, se o valor retornado for igual ao de entrada, será declarado como válido.	<pre> public class TestAtualizaContoController { @Test public void testValidarIdValido() { AtualizaContoController atualizaConto = new AtualizaContoController(); int idValido = atualizaConto.validarId("123", "Conto"); assertEquals(123, idValido); } </pre>
2	Nesse caso, decidi fazer dois testes da mesma coisa. Em alguns testes o resultado retornado não estava sendo '-1' quando existia um problema. Por isso da decisão, para poder averiguar esse problema.	<pre> @Test public void testValidarIdInvalido() { AtualizaContoController atualizaConto = new AtualizaContoController(); int idInvalido = atualizaConto.validarId("a", "Conto"); assertEquals(-1, idInvalido); } </pre>

3	<p>Nesse caso, verificamos a inclusão de heroi. Se o dado entrada for retornado exatamente igual. O resultado será válido. Depois de alguns testes, ele está válido.</p> <p>Tive alguns problemas em relação ao inserir por isso a verificação é feita diretamente em Heroi</p>	<pre> package projeto1.appmitologia.controller; import org.junit.Test; import projeto1.appmitologia.model.Heroi; import static org.junit.Assert.assertEquals; public class TestIncluiHeroiController { @Test public void testIncluiHeroi() { IncluiHeroiController atualizaHeroi = new IncluiHeroiController(); Heroi heroi = new Heroi(); heroi.setNome("Novo Nome"); heroi.setDescricao("Nova Descrição"); heroi.setUrl("nova-imagem.com"); assertEquals("Novo Nome", heroi.getNome()); assertEquals("Nova Descrição", heroi.getDescricao()); assertEquals("nova-imagem.com", heroi.getUrl()); } } </pre>
4	<p>Nesse caso, verificamos a inclusão de Conto. Se o dado entrada for retornado exatamente igual. O resultado será válido. Depois de alguns testes, ele está válido.</p> <p>Tive os mesmos problemas que em inserir Herói, por isso o uso direto de Conto</p>	<pre> package projeto1.appmitologia.controller; import org.junit.Test; import projeto1.appmitologia.model.Conto; import static org.junit.Assert.assertEquals; public class TestIncluiContoController { @Test public void testIncluiConto() { IncluiContoController atualizaHeroi = new IncluiContoController(); Conto conto = new Conto(); conto.setNome("Novo Nome"); conto.setDescricao("Nova Descrição"); conto.setNomeHeroi("Novo NomeHeroi"); conto.setLocalizacao("Novo Localizacao"); assertEquals("Novo Nome", conto.getNome()); assertEquals("Nova Descrição", conto.getDescricao()); assertEquals("Novo NomeHeroi", conto.getNomeHeroi()); assertEquals("Novo Localizacao", conto.getLocalizacao()); } } </pre>