SÃOJUDAS UNIVERSIDADE	FTCE Faculdade de Tecnologia e Ciências Exatas

USJT - 2017 - Práticas de Programação

Professores	Teoria: Bonato, Hamilton Laboratório: Calvetti, Hamilton, Keity e Rodrigo

Aula: 2
Assunto: Testes Unitários em Java com JUnit

Conceitos básicos abordados pelo professor:

Os testes unitários, inegavelmente, aumentam a qualidade do software desenvolvido pois reduzem enormemente a quantidade de bugs (BECK, 2000, p.118). E resultados melhores são obtidos quando as baterias de testes são repetidas a cada alteração significativa do código, os chamados testes de regressão. Porém, retestar tudo, sempre, é uma tarefa cara e tediosa. Por este motivo, utilizam-se ferramentas de automação de testes. Uma das ferramentas de uso consagrado é o JUnit (BECK; GAMMA, 2004).

Teste unitário - definição

Um teste unitário é um código escrito por um programador com o objetivo de testar uma funcionalidade específica do código a ser testado. Seu alvo é a menor unidade de código: um método em uma classe. São testes denominados **testes de caixa preta**, pois verificam apenas a saída gerada por um certo número de parâmetros de entrada, não se preocupando com o que ocorre dentro do método.

O quê testar?

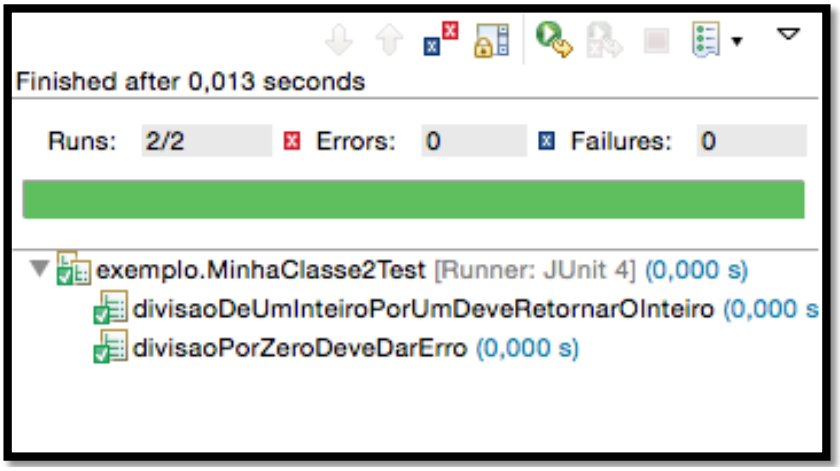
Este assunto sempre gera controvérsias, mas um código trivial, como métodos get e set, não precisa ser testado. Estes testes geram trabalho e pouco resultado. Por outro lado, código mais complexo, como métodos que contenham regras de negócio, estes sim devem ser alvos de diversos testes. Um conjunto sólido de testes protege o sistema de erros de regressão (basicamente, estragar sem querer código que não foi mexido) quando acontecem alterações no sistema.

O JUnit

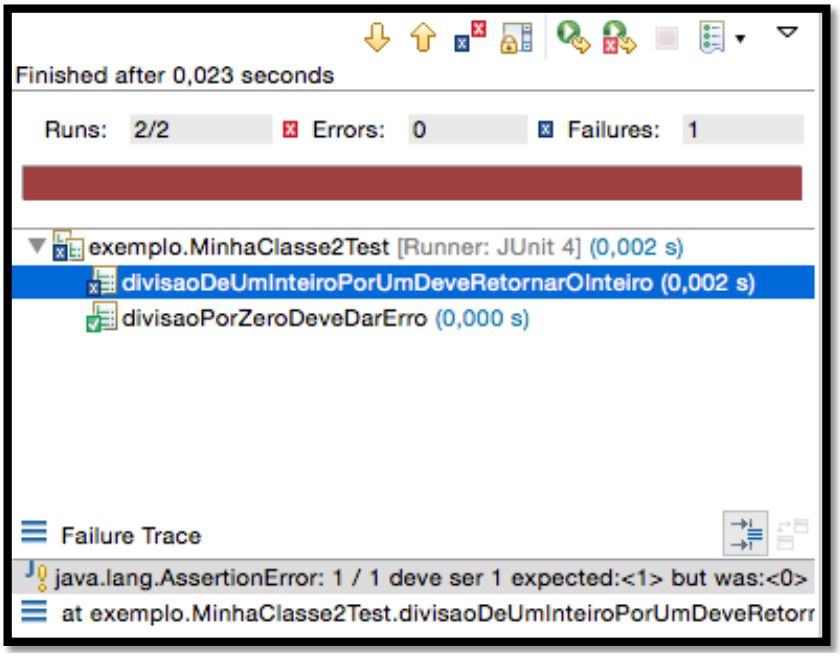
O JUnit é um framework open source de testes criado por Kent Beck (Extreme Programming) e Erich Gamma (Design Patterns), disponível em www.junit.org.
A versão atual do framework é a 4.x, que faz o uso extensivo de anotações (Java Annotations) para identificar os métodos de teste. O download pode ser feito de https://github.com/junit-team/junit.

Exemplos de código JUnit

Veja os exemplos 1 e 2. No exemplo 1 é descrita uma classe com um método que multiplica dois inteiros recebidos por parâmetro.
O exemplo 2 mostra a classe de teste deste método. Há um método de teste, indicado pela *annotation* @Test. E, dentro dele, 3 chamadas ao método *AssertEquals*, que tem um parâmetro String com a identificação do caso de teste, um parâmetro com a chamada do método a ser testado e um parâmetro com o resultado esperado. O *AssertEquals* compara o retorno do método multiplica com o resultado que o próprio programador indicou como esperado e retorna **true** se for igual e **false** se for diferente.
Ao se executar o teste, o JUnit mostra uma barra de progressão dos testes. Ela começa na cor verde e continua verde enquanto nenhum erro for encontrado. Se algum caso de teste der erro, a barra muda para vermelho continua assim até o fim dos testes.
No final, o JUnit mostra os retornos de todos os casos de teste, destacando os que apresentaram erro e mostrando o resultado esperado e o obtido.



Teste sem erros.



Teste com erros. Note no rodapé o detalhe do erro.

O exemplo 3 mostra uma classe com um único método que faz a divisão de dois inteiros. O caso de teste, no exemplo caso, possui 2 métodos: um que testa a divisão por 1, novamente usando o AssertEquals. E outro que testa a divisão por zero. Note que, neste caso, a *annotation* diz qual a exceção esperada. Como esta classe tem dois métodos de teste, foi usado o método setUp para instanciar a classe a ser testada.
O exemplo 5 mostra uma classe com um método de ordenação BubbleSort (já visto em ALGED). A classe de testes (exemplo 6) testa a ordenação de 3 número inteiros, testando todas as combinações de entrada possíveis. Note que, desta vez, o método usado é o AssertArrayEquals, que compara dois vetores. Se você usar o AssertEquals ele irá comparar a referência do vetor, e não seu conteúdo.

Conjuntos de Testes (*Test Suites*)

Os diversos casos de teste podem ser reunidos em um único conjunto de testes, de modo que todos seja executados, automaticamente, em sequência. Você pode ver isso no exemplo 7.

Anotações disponíveis no JUnit

Annotation	Descrição
@Test public void method	Identifica um método como um método de teste.
@Test (expected = Exception.class)	Falha se o método não lançar a exceção esperada
@Test(timeout=100)	Falha se o método levar mais de 100 milissegundos para retornar
@Before public void method()	É executado antes de cada método de teste da classe. Usado para preparar o ambiente de testes (ler dados de entrada, instanciar classes, etc).
@After public void method()	É executado ao final de cada método de teste da classe. Usado para limpar o ambiente de testes, como deletar dados temporários ou restaurar defaults.
@BeforeClass public static void method()	Executado uma única vez, antes de todos os testes da classe. Usado para atividades demoradas, como fazer uma conexão com o banco de dados. Note que este método deve ser static.
@AfterClass public static void method()	Executado uma única vez, depois de todos os testes. Analogamente ao anterior, deve ser usado para atividades demoradas, como fechar a conexão com o banco de dados. Note que este método deve ser static.
@Ignore	Ignora um método de teste. Usado quando você não quer executar um determinado método de teste mas não quer apagá-lo ou comentá-lo.

Asserts disponíveis no JUnit

Annotation	Descrição
fail(String)	Faz com que um método de teste falhe. Todo método de teste falha antes de ser implementado. O parâmetro é opcional.
assertTrue([mensagem], boolean condição)	Verifica se a condição lógica é verdadeira.
assertFalse([mensagem], boolean condição)	Verifica se a condição lógica é falsa.
assertEquals([String mensagem], esperado, obtido)	Verifica se dois valores são iguais. Não compara conteúdo de vetores.
assertEquals([String mensagem], esperado, obtido, tolerance)	Verifica de dois double são iguais dentro da tolerância indicada, que é o número de casas decimais que devem ser iguais.
assertArrayEquals([String mensagem], esperado, obtido)	Compara o conteúdo de dois vetores, elemento por elemento.
assertNull([mensagem], objeto)	Verifica se o objeto é nulo.
assertNotNull([mensagem], objeto)	Verifica se o objeto não é nulo.
assertSame([String], esperado, obtido)	Verifica se ambas as variáveis se referem ao mesmo objeto.
assertNotSame([String], esperado, obtido)	Verifica se as variáveis se referem a diferentes objeto.

Nomenclatura dos Testes

Nenhum padrão é obrigatório. O padrão sugerido é que as classes tenham o mesmo nome da classe a ser testada com o sufixo Test e os métodos indiquem o que está sendo testado. Veja os exemplos 2, 4 e 6.

Ordem de Execução dos Testes

O JUnit executa os testes, em uma classe de testes, na ordem determinada por ele. Se em alguma situação for necessário executar os testes em uma ordem definida pelo desenvolvedor use a anotação @FixMethodOrder(MethodSorters.NAME_ASCENDING) antes da declaração da classe de testes.

Uso do JUnit Integrado ao Eclipse

O JUnit vem integrado ao pacote do Eclipse. Para criar uma classe de teste, selecione a classe que quer testar e clique em **Arquivo > Novo > Outro > Java > JUnit > JUnit Test Case** Selecione os métodos que quer criar (exceto os de teste) e clique em **Next**. Então selecione os métodos que quer testar e clique em **Finish**. O Eclipse gera a classe e os métodos.
Para gerar um conjunto de testes, clique em **Arquivo > Novo > Outro > Java > JUnit > JUnit > JUnit Test Suite.** O Eclipse irá pedir para você selecionar as classes que quer que façam parte do conjunto de testes.

Testes de Bancos de Dados

Bancos são especialmente complicados de testar. Principalmente se as chaves primárias forem do tipo auto incremento. Porém, seguir alguns passos pode facilitar as coisas.

1. Tenha uma fixture em cada tabela a ser testada. Fixture é um conjunto de dados de testes cujo resultado é conhecido. No caso do banco, é uma linha na tabela cujos valores, inclusive os da pk, são conhecidos.

2. Use a fixture para testar o método selecionar do CRUD. Como? Crie um objeto com valores iguais ao da fixture e crie um outro objeto vazio. No objeto vazio, chame o método selecionar passando a PK da fixture por parâmetro. Agora chame o assertEquals comparando os dois objetos. Não se esqueça de sobrepor o método equals deste objeto.

3. Crie métodos para testar o inserir, o atualizar e o remover, nesta ordem. Numere os métodos, use o @FixMethodOrder. O teste do selecionar é o zero.

4. Nos métodos criados em 3, tenha sempre um objeto de controle. Faça a operação referente ao método de teste - inserir, por exemplo - em um objeto vazio, chame o selecionar do objeto vazio e compare como o objeto de controle.

5. No método inserir, se sua tabela tiver chave auto increment, não se esqueça de armazenar a chave gerada em uma variável static. Caso contrário não conseguirá rodar o atualizar e o remover.

6. Crie um método setUp para instanciar os objetos de controle, que devem ser variáveis de instância da classe de testes.

Veja os exemplos 10 e 11.

Exemplo1: Classe a ser testada

```
package exemplo;

public class MinhaClasse1 {

   public int multiplica(int x, int y) {
      return x * y;
   }
}
```

Exemplo2: Classe de Teste do Exemplo 1

```
package exemplo;

import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.Test;
```

```
public class MinhaClasse1Test {

   @Test
   public void multiplicacaoDeUmInteiroPorZeroDeveRetornarZero() {

      //Classe a ser testada
      MinhaClasse1 teste = new MinhaClasse1();

      //Testes
      assertEquals("10 X 0 deve ser 0", teste.multiplica(10, 0), 0);
      assertEquals("0 X 10 deve ser 0", teste.multiplica(0, 10), 0);
      assertEquals("0 X 0 deve ser 0", teste.multiplica(0, 0), 0);
   }

}
```

Exemplo3: Classe a ser testada

```
package exemplo;

public class MinhaClasse2 {

   public int divide(int x, int y){
      return x/y;
   }
}
```

Exemplo4: Classe de Teste do Exemplo 3

```
package exemplo;

import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.Test;

public class MinhaClasse2Test {
   MinhaClasse2 teste;

   @Before
   public void setUp(){
      //Classe a ser testada
      teste = new MinhaClasse2();
   }

   @Test
   public void divisaoDeUmInteiroPorUmDeveRetornarOInteiro() {

      //Testes
      assertEquals("10 / 1 deve ser 10", teste.divide(10, 1), 10);
      assertEquals("0 / 1 deve ser 0", teste.divide(0, 1), 0);
      assertEquals("1 / 1 deve ser 1", teste.divide(1, 1), 1);
   }

   @Test(expected = ArithmeticException.class)
   public void divisaoPorZeroDeveDarErro() {

      //Teste de Exceção
```

```
    assertEquals("10 / 0 deve dar ArithmeticException", teste.divide(10,
0));
   }

}
```

Exemplo5: Classe a ser testada

```
package exemplo;

public class MinhaClasse3 {
   /**
    * @param v vetor de inteiros a ser ordenado
    * Ordena usando uma ordenacao por bolha
    * @return vetor v ordenado em ordem crecente
    */
   public int[] crescente(int[] v) {
      for (int i = v.length - 1; i > 0; i--) {
         for (int j = 0; j <= i - 1; j++) {
            if (v[j] > v[j + 1]) {
               int aux = v[j + 1];
               v[j + 1] = v[j];
               v[j] = aux;
            }
         }
      }
      return v;
   }
}
```

Exemplo6: Classe de Teste do Exemplo 5

```
package exemplo;

import static org.junit.Assert.assertArrayEquals;
import org.junit.Test;

public class MinhaClasse3Test {

   @Test
   public void tresInteirosEmQualquerOrdemSaoOrdenadosEmOrdemCrescente() {
      //Classe a ser testada
      MinhaClasse3 teste = new MinhaClasse3();

      //Testes
      // testar as 6 permutacoes possiveis
      int[] res = { 1, 2, 3 };
      int[] v1 = { 1, 2, 3 };
      int[] v2 = { 1, 3, 2 };
      int[] v3 = { 2, 1, 3 };
      int[] v4 = { 2, 3, 1 };
      int[] v5 = { 3, 1, 2 };
      int[] v6 = { 3, 2, 1 };
      // testar as 6 permutacoes possiveis
      assertArrayEquals("Entrada 1, 2, 3", teste.crescente(v1), res);
      assertArrayEquals("Entrada 1, 3, 2", teste.crescente(v2), res);
      assertArrayEquals("Entrada 2, 1, 3", teste.crescente(v3), res);
      assertArrayEquals("Entrada 2, 3, 1", teste.crescente(v4), res);
      assertArrayEquals("Entrada 3, 1, 2", teste.crescente(v5), res);
```

```
      assertArrayEquals("Entrada 3, 2, 1", teste.crescente(v6), res);
   }

}
```

Exemplo7: Classe a ser Testada

```
package exemplo;

public class Aluno {
   private int ra;
   private String nome;

   public Aluno(int ra, String nome) {
      this.ra = ra;
      this.nome = nome;
   }
   public int getRa() {
      return ra;
   }
   public void setRA(int rA) {
      ra = rA;
   }
   public String getNome() {
      return nome;
   }
   public void setNome(String nome) {
      this.nome = nome;
   }
   @Override
   public boolean equals(Object obj) {
      if (this == obj)
         return true;
      if (obj == null)
         return false;
      if (getClass() != obj.getClass())
         return false;
      Aluno other = (Aluno) obj;
      if (ra != other.ra)
         return false;
      return true;
   }
}
```

Exemplo8: Classe de Teste do Exemplo 7

```
package exemplo;

import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;

public class AlunoTest {
   Aluno aluno;

   @Before
   public void setUp() throws Exception {
```

```
      aluno = new Aluno(12345, "Joao");
   }

   @Test
   public void testEquals(){
      assertEquals("Aluno 12345, Joao", aluno, new Aluno(12345, "Joao"));
   }

}
```

Exemplo9: Conjunto de Testes

```
package exemplo;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses({ MinhaClasse1Test.class, MinhaClasse2Test.class,
     MinhaClasse3Test.class, AlunoTest.class })
public class AllTests {

}
```

Exemplo 10: Classe Cliente

```
package model;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class Cliente {
   private int id;
   private String nome;
   private String fone;
   private String email;

   static {
      try {
         Class.forName("com.mysql.jdbc.Driver");
      } catch (ClassNotFoundException e) {
         throw new RuntimeException(e);
      }
   }

   public Cliente(){

   }

   public Cliente(int id, String nome, String fone, String email) {
      this.id = id;
      this.nome = nome;
```

```
    this.fone = fone;
    this.email = email;
}

// Obtém conexão com o banco de dados
public Connection obtemConexao() throws SQLException {
    return DriverManager

.getConnection("jdbc:mysql://localhost/vendas?user=alunos&password=alunos");
}

public void criar() {
    String sqlInsert = "INSERT INTO cliente(nome, fone, email) VALUES (?, ?, ?)";
    // usando o try with resources do Java 7, que fecha o que abriu
    try (Connection conn = obtemConexao();
          PreparedStatement stm = conn.prepareStatement(sqlInsert);) {
       stm.setString(1, getNome());
       stm.setString(2, getFone());
       stm.setString(3, getEmail());
       stm.execute();
       String sqlQuery  = "SELECT LAST_INSERT_ID()";
       try(PreparedStatement stm2 = conn.prepareStatement(sqlQuery);
          ResultSet rs = stm2.executeQuery();) {
          if(rs.next()){
             setId(rs.getInt(1));
          }
       } catch (SQLException e) {
          e.printStackTrace();
       }
    } catch (SQLException e) {
       e.printStackTrace();
    }
}

public void atualizar() {
    String sqlUpdate = "UPDATE cliente SET nome=?, fone=?, email=? WHERE id=?";
    // usando o try with resources do Java 7, que fecha o que abriu
    try (Connection conn = obtemConexao();
          PreparedStatement stm = conn.prepareStatement(sqlUpdate);) {
       stm.setString(1, getNome());
       stm.setString(2, getFone());
       stm.setString(3, getEmail());
       stm.setInt(4, getId());
       stm.execute();
    } catch (Exception e) {
       e.printStackTrace();
    }
}

public void excluir() {
    String sqlDelete = "DELETE FROM cliente WHERE id = ?";
    // usando o try with resources do Java 7, que fecha o que abriu
    try (Connection conn = obtemConexao();
          PreparedStatement stm = conn.prepareStatement(sqlDelete);) {
       stm.setInt(1, getId());
       stm.execute();
    } catch (Exception e) {
       e.printStackTrace();
```

```
      }
   }

   public void carregar() {
      String sqlSelect = "SELECT nome, fone, email FROM cliente WHERE cliente.id =
?";
      // usando o try with resources do Java 7, que fecha o que abriu
      try (Connection conn = obtemConexao();
            PreparedStatement stm = conn.prepareStatement(sqlSelect);) {
         stm.setInt(1, getId());
         try (ResultSet rs = stm.executeQuery();) {
            if (rs.next()) {
               setNome(rs.getString("nome"));
               setFone(rs.getString("fone"));
               setEmail(rs.getString("email"));
            } else {
               setId(-1);
               setNome(null);
               setFone(null);
               setEmail(null);
            }
         } catch (SQLException e) {
            e.printStackTrace();
         }
      } catch (SQLException e1) {
         System.out.print(e1.getStackTrace());
      }
   }

   public int getId() {
      return id;
   }

   public void setId(int id) {
      this.id = id;
   }

   public String getNome() {
      return nome;
   }

   public void setNome(String nome) {
      this.nome = nome;
   }

   public String getFone() {
      return fone;
   }

   public void setFone(String fone) {
      this.fone = fone;
   }

   public String getEmail() {
      return email;
   }

   public void setEmail(String email) {
```

```
      this.email = email;
   }

   @Override
   public String toString() {
      return "Cliente [id=" + id + ", nome=" + nome + ", fone=" + fone
            + ", email=" + email + "]";
   }

   @Override
   public boolean equals(Object obj) {
      if (this == obj)
         return true;
      if (obj == null)
         return false;
      if (getClass() != obj.getClass())
         return false;
      Cliente other = (Cliente) obj;
      if (email == null) {
         if (other.email != null)
            return false;
      } else if (!email.equals(other.email))
         return false;
      if (fone == null) {
         if (other.fone != null)
            return false;
      } else if (!fone.equals(other.fone))
         return false;
      if (id != other.id)
         return false;
      if (nome == null) {
         if (other.nome != null)
            return false;
      } else if (!nome.equals(other.nome))
         return false;
      return true;
   }
}
```

Exemplo 10: Classe de Teste de Cliente

```
package test;


import static org.junit.Assert.assertEquals;
import model.Cliente;

import org.junit.Before;
import org.junit.FixMethodOrder;
import org.junit.Test;
import org.junit.runners.MethodSorters;

@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class ClienteTest {
   Cliente cliente, copia;
   static int id = 0;
```

```
   /*
   * Antes de rodar este teste, certifique-se que nao ha no banco nenhuma
   * linha com o id igual ao do escolhido para o to instanciado abaixo. Se
   * houver, delete.
   * Certifique-se também que sobrecarregou o equals na classe
   * Cliente.
   * Certifique-se que a fixture cliente1 foi criada no banco.
   * Além disso, a ordem de execução dos testes é importante; por
   * isso a anotação FixMethodOrder logo acima do nome desta classe
   */
   @Before
   public void setUp() throws Exception {
      System.out.println("setup");
      cliente = new Cliente(id, "Batista Cepelos", "(11) 91234-4321", "btcp@usjt.br"
);
      copia = new Cliente(id, "Batista Cepelos", "(11) 91234-4321", "btcp@usjt.br"
);
      System.out.println(cliente);
      System.out.println(copia);
      System.out.println(id);
   }

   @Test
   public void test00Carregar() {
      System.out.println("carregar");
      //para funcionar o cliente 1 deve ter sido carregado no banco por fora
      Cliente fixture = new Cliente(1, "Carlos Drummond de Andrade", "(11) 91234-
4321", "cda@usjt.br" );
      Cliente novo = new Cliente(1, null, null, null);
      novo.carregar();
      assertEquals("testa inclusao", novo, fixture);
   }

   @Test
   public void test01Criar() {
      System.out.println("criar");
      cliente.criar();
      id = cliente.getId();
      System.out.println(id);
      copia.setId(id);
      assertEquals("testa criacao", cliente, copia);
   }

   @Test
   public void test02Atualizar() {
      System.out.println("atualizar");
      cliente.setFone("999999");
      copia.setFone("999999");
      cliente.atualizar();
      cliente.carregar();
      assertEquals("testa atualizacao", cliente, copia);
   }

   @Test
   public void test03Excluir() {
      System.out.println("excluir");
      copia.setId(-1);
      copia.setNome(null);
```

```
     copia.setFone(null);
     copia.setEmail(null);
     cliente.excluir();
     cliente.carregar();
     assertEquals("testa exclusao", cliente, copia);
   }
}
```

Script para criação da tabela de clientes

```
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';

-- -----------------------------------------------------
-- Schema vendas
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema vendas
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `vendas` DEFAULT CHARACTER SET utf8 ;
USE `vendas` ;

-- -----------------------------------------------------
-- Table `vendas`.`Cliente`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `vendas`.`Cliente` (
  `id` INT(11) UNSIGNED NOT NULL AUTO_INCREMENT,
  `nome` VARCHAR(100) NOT NULL,
  `fone` VARCHAR(15) NULL DEFAULT NULL,
  `email` VARCHAR(60) NOT NULL UNIQUE,
  PRIMARY KEY (`id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;

SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

Script para carga de clientes

```
USE VENDAS;
INSERT INTO `cliente` (`id`,`nome`,`fone`,`email`) VALUES (1,'Carlos
Drummond de Andrade','(11) 91234-4321','cda@usjt.br');
INSERT INTO `cliente` (`id`,`nome`,`fone`,`email`) VALUES (2,'Manuel
Bandeira','(11) 91234-4322','mb@usjt.br');
INSERT INTO `cliente` (`id`,`nome`,`fone`,`email`) VALUES (3,'Olavo
Bilac','(11) 91234-4323','ob@usjt.br');
INSERT INTO `cliente` (`id`,`nome`,`fone`,`email`) VALUES (4,'Vinícius
de Moraes','(11) 91234-4324','vm@usjt.br');
INSERT INTO `cliente` (`id`,`nome`,`fone`,`email`) VALUES (5,'Cecília
Meireles','(11) 91234-4325','cm@usjt.br');
```

```
INSERT INTO `cliente` (`id`,`nome`,`fone`,`email`) VALUES (6,'Castro
Alves','(11) 91234-4326','ca@usjt.br');
INSERT INTO `cliente` (`id`,`nome`,`fone`,`email`) VALUES (7,'Gonçalves
Dias','(11) 91234-4327','gd@usjt.br');
INSERT INTO `cliente` (`id`,`nome`,`fone`,`email`) VALUES (8,'Ferreira
Gullar','(11) 91234-4328','fg@usjt.br');
INSERT INTO `cliente` (`id`,`nome`,`fone`,`email`) VALUES (9,'Machado
de Assis','(11) 91234-4329','ma@usjt.br');
INSERT INTO `cliente` (`id`,`nome`,`fone`,`email`) VALUES (10,'Mário de
Andrade','(11) 91234-4330','mda@usjt.br');
INSERT INTO `cliente` (`id`,`nome`,`fone`,`email`) VALUES (11,'Cora
Coralina','(11) 91234-4331','cc@usjt.br');
INSERT INTO `cliente` (`id`,`nome`,`fone`,`email`) VALUES (12,'Manoel
de Barros','(11) 91234-4332','mdb@usjt.br');
INSERT INTO `cliente` (`id`,`nome`,`fone`,`email`) VALUES (13,'João
Cabral de Melo Neto','(11) 91234-4333','jcmn@usjt.br');
INSERT INTO `cliente` (`id`,`nome`,`fone`,`email`) VALUES (14,'Casimiro
de Abreu','(11) 91234-4334','ca1@usjt.br');
INSERT INTO `cliente` (`id`,`nome`,`fone`,`email`) VALUES (15,'Paulo
Leminski','(11) 91234-4335','pl@usjt.br');
INSERT INTO `cliente` (`id`,`nome`,`fone`,`email`) VALUES (16,'Álvares
de Azevedo','(11) 91234-4136','aa@usjt.br');
INSERT INTO `cliente` (`id`,`nome`,`fone`,`email`) VALUES
(17,'Guilherme de Almeida','(11) 91234-4337','ga@usjt.br');
INSERT INTO `cliente` (`id`,`nome`,`fone`,`email`) VALUES
(18,'Alphonsus de Guimarães','(11) 91234-4338','ag@usjt.br');
INSERT INTO `cliente` (`id`,`nome`,`fone`,`email`) VALUES (19,'Gregório
de Matos','(11) 91234-4339','gm@usjt.br');
INSERT INTO `cliente` (`id`,`nome`,`fone`,`email`) VALUES
(20,'Gonçalves de Magalhães','(11) 91234-4340','gdm@usjt.br');
INSERT INTO `cliente` (`id`,`nome`,`fone`,`email`) VALUES
(21,'Junqueira Freire','(11) 91234-4341','jf@usjt.br');
INSERT INTO `cliente` (`id`,`nome`,`fone`,`email`) VALUES (22,'Fabrício
Carpinejar','(11) 91234-4342','fc@usjt.br');
INSERT INTO `cliente` (`id`,`nome`,`fone`,`email`) VALUES (23,'Santa
Rita Durão','(11) 91234-4343','srd@usjt.br');
INSERT INTO `cliente` (`id`,`nome`,`fone`,`email`) VALUES (24,'Basílio
da Gama','(11) 91234-4344','bg@usjt.br');
INSERT INTO `cliente` (`id`,`nome`,`fone`,`email`) VALUES (25,'Angélica
Freitas','(11) 91234-4345','af@usjt.br');
INSERT INTO `cliente` (`id`,`nome`,`fone`,`email`) VALUES (26,'Fagundes
Varela','(11) 91234-4346','fv@usjt.br');
INSERT INTO `cliente` (`id`,`nome`,`fone`,`email`) VALUES (27,'Augusto
dos Anjos','(11) 91234-4347','ada@usjt.br');
INSERT INTO `cliente` (`id`,`nome`,`fone`,`email`) VALUES (28,'Sílvio
Romero','(11) 91234-4348','sr@usjt.br');
INSERT INTO `cliente` (`id`,`nome`,`fone`,`email`) VALUES (29,'Orestes
Barbosa','(11) 91234-4349','ob2@usjt.br');
INSERT INTO `cliente` (`id`,`nome`,`fone`,`email`) VALUES (30,'Torquato
Netto','(11) 91234-4350','tn@usjt.br');
```

Exercícios

1. Revisão da Teoria

a. Crie um projeto Java no Eclipse. Ainda não precisa ser Web.

b. Arraste as classes Cliente e ClienteTest fornecidas e corrija eventuais erros com as packages. Não se esqueça de colocar o driver de JDBC no BuildPath.

c. Rode os scripts de criação do banco e de carga de clientes no MySQL Workbench.

d. Execute o ClienteTest (clique sobre a classe no Project Browser com o botão direito, escolha Run As > JUnit Test) e veja se funcionou.

2. Exercício para Nota (correção em aula)

a. Crie uma tabela Pais com os atributos id (int autoincrement) , nome(varchar 100), populacao (bigint) e area (number 15,2). Carregue alguns países. Consulte a Wikipedia para obter as informações de população e área.

b. Crie uma classe Pais com os atributos id (int), nome (String), populacao (long) e area (double).

c. Crie um construtor com os campos, um construtor padrão, gets e sets.

d. Crie os métodos do CRUD de Pais.

e. Crie um método que retorna o país com maior número de habitantes.

f. Crie um método que retorna o país com menor área.

g. Crie um método que retorne um vetor de 3 países.

h. Crie uma classe de testes que, além de testar os CRUDs, teste também os métodos maiorPopulacao, menorArea e vetorTresPaises.

Bibliografia

BECK, K. Extreme Programming explained: embrace change. [S.l.]: Addison-Wesley Longman Publishing Co., Inc., 2000. ISBN 0-201-61641-6.

BECK, K.; GAMMA, E. JUnit Open Source Testing Framework. abr 2015. On-line. Disponível em: <http://www.junit.org/>.

KOSKELA, K; Effective Unit Testing: A guide for Java developers; 1a Ed. Manning Publications, 2015. ISBN 1935182579.

MESZAROS, G; XUnit Test Patterns: refactoring test code; 1ª Ed. Person Education, 2007. ISBN 0-13-149505-4.