Testes Unitários - JUnit

Metodologias e Desenvolvimento de Software

Pedro Salgueiro
pds@uevora.pt
CLV-256

Testes unitários - JUnit

Junit

- Framework de testes
  - Java
- Integração com IDEs
  - Eclipse, Idea IntelliJ
- Execução automática de testes
- Baseado em anotações
- Testes estão organizados em classes
  - Classes de teste
- Suite de testes
  - Conjunto de classes de teste

Conceitos

- Classes de testes
  - Uma classe de testes para cada classe
  - **Pelo menos** um método de testes para cada método
    - Caso em que tudo corre bem
- Métodos de teste
  - Métodos que implementam um caso de testes
  - Cada método de testes deve implementar o maior nº possível de casos de testes
    - Alternativamente, podem-se criar métodos de teste para cada caso
- Suite de testes
  - Conjunto de classes de teste

Classes de testes

- Uma classe de testes para cada classe
- **Pelo menos** um método de testes para cada método
  - Caso em que tudo corre bem
- Convenções
  - Usar os mesmos nomes de pacotes
  - Nomes baseados nas classes que estão a ser testadas
    - Classe a ser testada: **MyClass**
    - Classe de testes: **MyClassTest**

Métodos de teste

- Estrutura
  - Setup do teste
    - Inicialização de variáveis
    - Definição do resultado esperado
  - Execução
    - método que se quer testar
  - Assert
    - Comparação entre o resultado obtido e o resultado esperado
- Convenções
  - Nomes descritivos
    - O que deve (ou não acontecer)
  - Exemplo
    - deveDividirDoisNumeros()
    - deveLancarExcepcaoDivPorZero()

Testes unitários - JUnit

Mecanismos base

- Anotações
  - Método para descobrir, organizar e ativar testes
- Assertions (afirmações)
  - Método usado para determinar se um teste passou ou não
  - Comparação
    - Valor esperado e o valor calculado

Testes unitários - JUnit

Métodos de teste

- Executados num ambiente isolado
  - Não devem depender de outros testes
- Baseados em anotações
  - `@org.junit.Test`

```
@Test
public void testMultiply() {

   // MyClass is tested
   MyClass tester = new MyClass();

   // check if multiply(10,5) returns 50
   assertEquals("10 x 5 must be 50", 50, tester.multiply(10, 5));
 }
```

Testes unitários - JUnit

Métodos de teste - exemplo

```
import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;

public class MyTests {

        @Test
        public void multiplicationOfZeroIntegersShouldReturnZero() {
                MyClass tester = new MyClass(); // MyClass is tested

                // assert statements
                assertEquals("10 x 0 must be 0", 0, tester.multiply(10, 0));
                assertEquals("0 x 10 must be 0", 0, tester.multiply(0, 10));
                assertEquals("0 x 0 must be 0", 0, tester.multiply(0, 0));
        }
}
```

Testes unitários - JUnit

Organização do projecto

- Boas práticas
  - Pasta “**src**”
    - Pasta com o source do projeto
    - Pasta com as classes que vão ser testadas
  - Pasta **“tests”**
    - Pasta com as classes de testes

Testes unitários - JUnit

Anotações Junit

- `@BeforeClass`
- `@AfterClass`
- `@Before`
- `@After`
- `@Test`
- `@Test(timeout = <delay>`
- `@Test(expected = <exception>.class)`
- `@ignore`

Anotações Junit

- @BeforeClass
  - Executado antes de todos os testes de uma classe de testes
  - Usado para inicializações

```
@BeforeClass
public void setUpBeforeClass() {
   // run once before all test cases
}
```

Anotações Junit

- @AfterClass
  - Executado depois de todos os testes de uma classe de testes
  - Atividades de limpeza
    - “teardown”

```
@AfterClass
public void tearDownAfterClass() {
// run for one time after all test cases
}
```

Anotações Junit

- @Before
  - Executado antes de **cada** teste de uma classe de testes
  - Setup de **todos** os testes de uma classe de testes
  - Setup do ambiente de execução do teste
  - exemplo: leitura de dados; inicialização de classes

```
@Before
public void setup() {
    simpleMath = new SimpleMath();
}
```

Anotações Junit

- @After
  - Executado depois de **cada** teste da classe de testes
  - Limpeza (teardown) de **todos** os testes

```
@After
public void tearDown() {
    simpleMath = null;
}
```

Anotações Junit

- @Test
  - Usado para “marcar”/anotar um método de testes

```
@Test
public void testAddition() {
    assertEquals(12, simpleMath.add(7, 5));
}
@Test
public void testSubtraction() {
    assertEquals(9, simpleMath.subtract(12, 3));
}
```

Anotações Junit

- @Test
  - Outro exemplo

```
@Test
public void listEquality() {
    List<Integer> expected = new ArrayList<Integer>();
    expected.add(5);

    List<Integer> actual = new ArrayList<Integer>();
    actual.add(5);

    assertEquals(expected, actual);
}
```

Testes unitários - JUnit

Anotações Junit

- @Test(timeout = <delay>)
  - Definir o tempo máximo de execução
  - Se o tempo exceder o valor, o teste falha

```
@Test(timeout = 1000)
public void testInfinity(){
    while (true);
}
```

```
@Test(timeout=1000)
public final void testTimeout() throws InterruptedException{
    Thread.sleep(900); //sleep for 900 ms
}
```

Anotações Junit

- @Test(expected = <exception>.class)
  - Teste passa se for lançada uma exceção
    - <exception>.class

```
@Test(expected = NullPointerException.class)
public void testInitFile() throws UserCancelException(){

    Files.initFile(null);

}
```

Testes unitários- JUnit

Anotações Junit

- @Ignore
  - Usado para ignorar testes
    - Testes incompletos/não terminados
  - Opcional: Adicionar uma string
    - Justificação para o ignore

```
@Ignore("Not Ready to Run")
@Test
public void multiplication() {

    assertEquals(15, simpleMath.multiply(3, 5));

}
```

Assertions

- assertEquals
- assertEquals for arrays
- assertNull / assertNotNull
- assertSame / assertNotSame
- assertTrue / assertFalse
- fail

Assertions

- assertEquals
  - Verifica se dois objectos são iguais
    - `expected.equals(actual) == true`
    - os dois objetos são **null**
  - Opcional
    - Incluir mensagem de falha
  - Cuidados a ter com os tipos double e float
    - Floating point errors
    - Maximum tolerance
  - Versões específicas para cada tipo primitivo

```
assertEquals(Object expected, Object actual);
assertEquals(String message, Object expected, Object actual);
assertEquals(Object expected, Object actual, delta);
assertEquals(String message, Object expected, Object actual, delta);
```

Assertions

- AssertEquals (arrays)
  - True
    - Quando os arrays têm o mesmo tamanho
    - Todos os elementos nas mesmas posições são iguais

```
public static void assertEquals(Object[] expected, Object[] actual);
public static void assertEquals(String message,
                                Object[] expected, Object[] actual);
```

Assertions

- assertNull
  - Verifica se um objeto é null
- assertNotNull
  - Verifica se um objeto não é null

```
assertNull(Object object),
assertNull(String message, Object object)
```

```
assertNotNull(Object object),
assertNotNull(String message, Object)
```

Testes unitários - JUnit

Assertions

- assertSame
  - Verifica se dois objectos são o mesmo
- assertNotSame
  - Verifica se dois objectos não são o mesmo
- Equivalente
  - aos operadores `==` e `!=`

```
assertSame(Object expected, Object actual)
assertSame(String message, Object expected, Object actual)
```

```
assertNotSame(Object expected, Object actual)
assertNotSame(String message, Object expected, Object actual)
```

Assertions

- assertTrue
  - Verifica se uma condição booleana é verdade
- assertFalse
  - Verifica se uma condição booleana é falsa

```
assertTrue(boolean condition)
assertTrue(String message, boolean condition)
```

```
assertFalse(boolean condition)
assertFalse(String message, boolean condition)
```

Testes unitários - JUnit

Assertions

- fail
  - Obriga um teste a falhar
  - Útil quando queremos forçar um teste a falhar quando perante condições *proibidas*

```
fail()
fail(String message)
```

Testes unitários - JUnit

Dicas

- Usar variáveis de classes nas classes de testes
- Ser exaustivo
- Não esquecer as exceções
- Documentar os *asserts*
- Verificar a cobertura de testes

Dicas

- Usar variáveis de classes nas classes de testes
  - Usar variáveis de classe
  - Como se fossem classes “normais”

```
public class InvoiceTest{
    private static Invoice fx, fy, fz;
    private static Client c1, c2, c3;
    private static Product p1, p2, p3;
    // maximum error when comparing real numbers
    private static final double EPSILON=0.0000001;
    ...
    ...
}
```

Testes unitários - JUnit

Dicas

- Ser exaustivo
  - Testar o máximo possível
  - exemplo: testar todos os dias do ano

```
@Test
public final void testValid1(){
    Calendar day = Calendar.getInstance();
    day.setTime(new Date());
    for (int i=1; i<=366; i++){
        String dayString = day.get(Calendar.YEAR)+
            "/"+(day.get(Calendar.MONTH)+1)+
            "/"+day.get(Calendar.DATE);
        assertTrue("The date was correctly set",
                              MyDate.valid(dayString));
        day.add(Calendar.DATE, 1);
    }
}
```

Testes unitários - JUnit

Dicas

- Lidar com excepções
  - Garantir que as excepções são lançadas quando devem ser lançadas

```
@Test(expected = MyDate.InvalidDateException.class)
public final void testValid2()
{
    // 2011 is not a leap year!
    MyDate.valid("2011/02/29");
}
```

```
@Test(expected = NumberFormatException.class)
public final void testValid3()
{
    // unsupported format!
    MyDate.valid("2011/February/21");
}
```

Dicas

- Documentar os asserts
  - Útil quando os testes falham

```
@Test
public final void testClone()
{
    Invoice clone = (Invoice) fx.clone();
    assertEquals("The invoice is equal to its clone", fx, clone);
    assertNotSame("The clone is a distinct object", fx, clone);
    assertArrayEquals("The invoice and its clone have the same lines",
                      fx.getLines().toArray(),
                      clone.getLines().toArray());
}
```

Testes unitários - JUnit

Exemplo

```
public class Calculator {
  public int evaluate(String expression) {
    int sum = 0;
    for (String summand: expression.split("\\+"))
      sum += Integer.valueOf(summand);
    return sum;
  }
}
```

Testes unitários - JUnit

Exemplo

```
public class Calculator {
  public int evaluate(String expression) {
    int sum = 0;
    for (String summand: expression.split("\\+"))
      sum += Integer.valueOf(summand);
    return sum;
  }
}
```

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class CalculatorTest {









}
```

Exemplo

```
public class Calculator {
  public int evaluate(String expression) {
    int sum = 0;
    for (String summand: expression.split("\\+"))
      sum += Integer.valueOf(summand);
    return sum;
  }
}
```

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class CalculatorTest {
  @Test
  public void evaluatesExpression() {



  }
}
```

Testes unitários - JUnit

Exemplo

```
public class Calculator {
  public int evaluate(String expression) {
    int sum = 0;
    for (String summand: expression.split("\\+"))
      sum += Integer.valueOf(summand);
    return sum;
  }
}
```

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class CalculatorTest {
  @Test
  public void evaluatesExpression() {
    Calculator calculator = new Calculator();


  }
}
```

Testes unitários - JUnit

Exemplo

```
public class Calculator {
  public int evaluate(String expression) {
    int sum = 0;
    for (String summand: expression.split("\\+"))
      sum += Integer.valueOf(summand);
    return sum;
  }
}
```

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class CalculatorTest {
  @Test
  public void evaluatesExpression() {
    Calculator calculator = new Calculator();
    int sum = calculator.evaluate("1+2+3");

  }
}
```

Testes unitários - JUnit

Exemplo

```
public class Calculator {
  public int evaluate(String expression) {
    int sum = 0;
    for (String summand: expression.split("\\+"))
      sum += Integer.valueOf(summand);
    return sum;
  }
}
```

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class CalculatorTest {
  @Test
  public void evaluatesExpression() {
    Calculator calculator = new Calculator();
    int sum = calculator.evaluate("1+2+3");
    assertEquals(6, sum);
  }
}
```

Testes unitários - JUnit

Verificar a cobertura de testes

- Periodicamente
- Usar os resultados
  - fazer mais testes
  - melhorar os testes
- Tipos de “cobertura de testes”
  - Statement
    - Cada statement
  - Branch
    - Fluxo de controle (if, while, etc...)
  - “caminhos”
    - “Caminhos” alternativos
    - “Impossível” considerar todas as situações
- Ferramentas para verificar a cobertura dos testes
  - EclEmma, CodeCover, Quilt, NoUnit, InsECT, Jester, jcoverage, Coverlipse, Hansel