UFRN
UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

Aula 2: Ferramenta JUnit

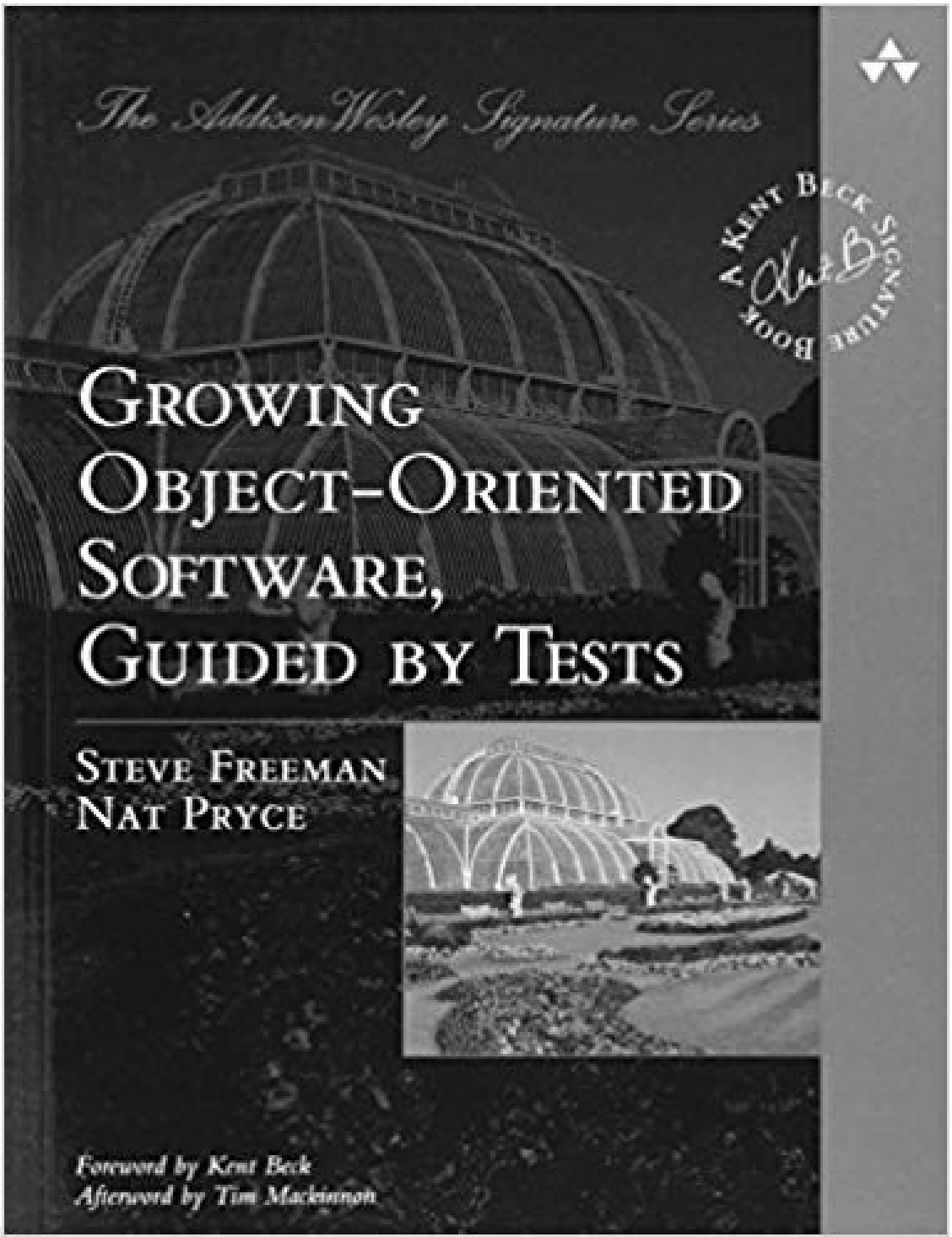
Profa. Roberta Coelho
Departamento de Informática e Matemática Aplicada - DIMAp

Referências:

- Artigo:
  - Test Infected: Programmers Love Writing Tests

  http://junit.sourceforge.net/doc/testinfected/testing.htm
- Tutorial:
  - How to write hard to test code?
  - http://misko.hevery.com/2009/10/28/how-to-write-hard-to-test-code-what-to-look-for-when-reviewing-other-peoples-hard-to-test-code/

Referências:



Objetivos da aula

- Discutir a importância dos testes de unidade
- Entender o framework JUnit
- Construir e executar testes de unidade
- Discutir sobre o que dificulta os Testes de Unidade

Agenda

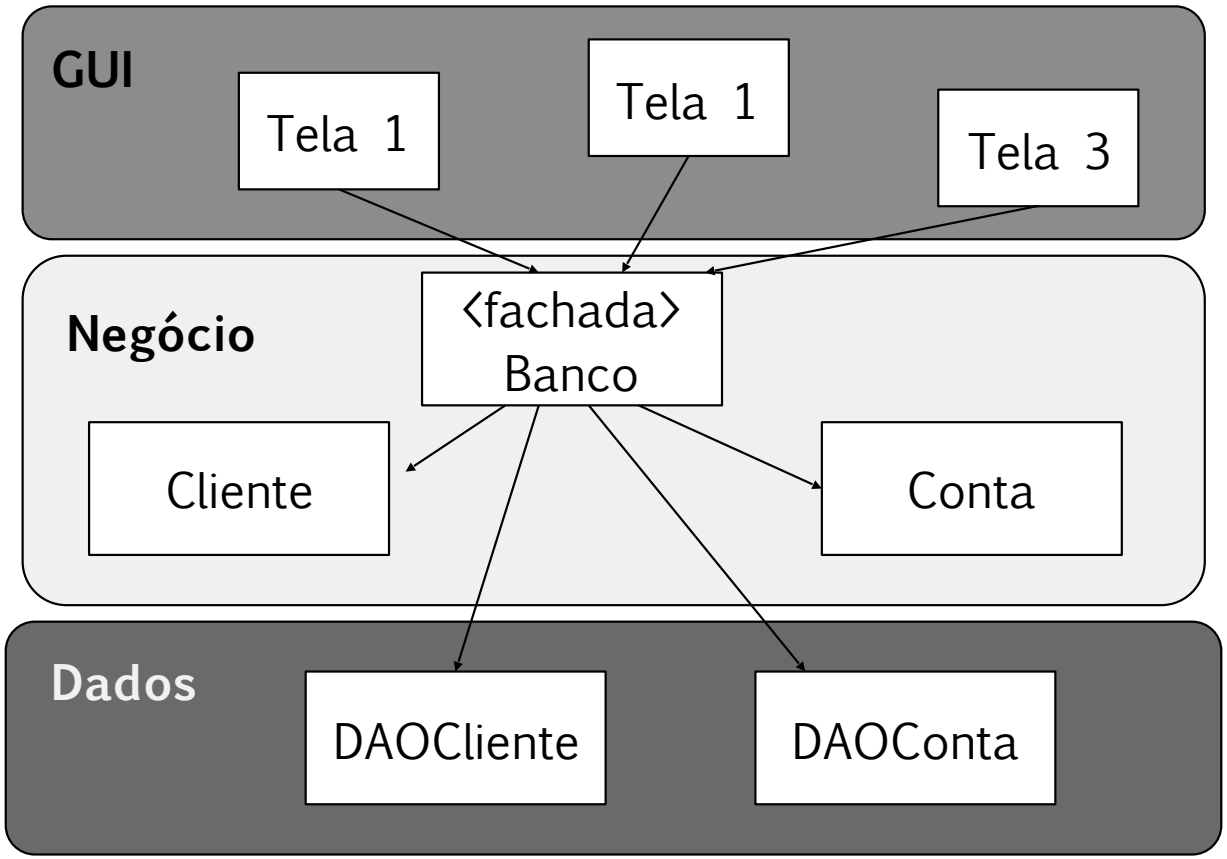
- Motivação
- O framework JUnit
- Passo a passo...
- Diferenças JUnit 3, JUnit4, JUnit5
- Mão na Massa

Agenda

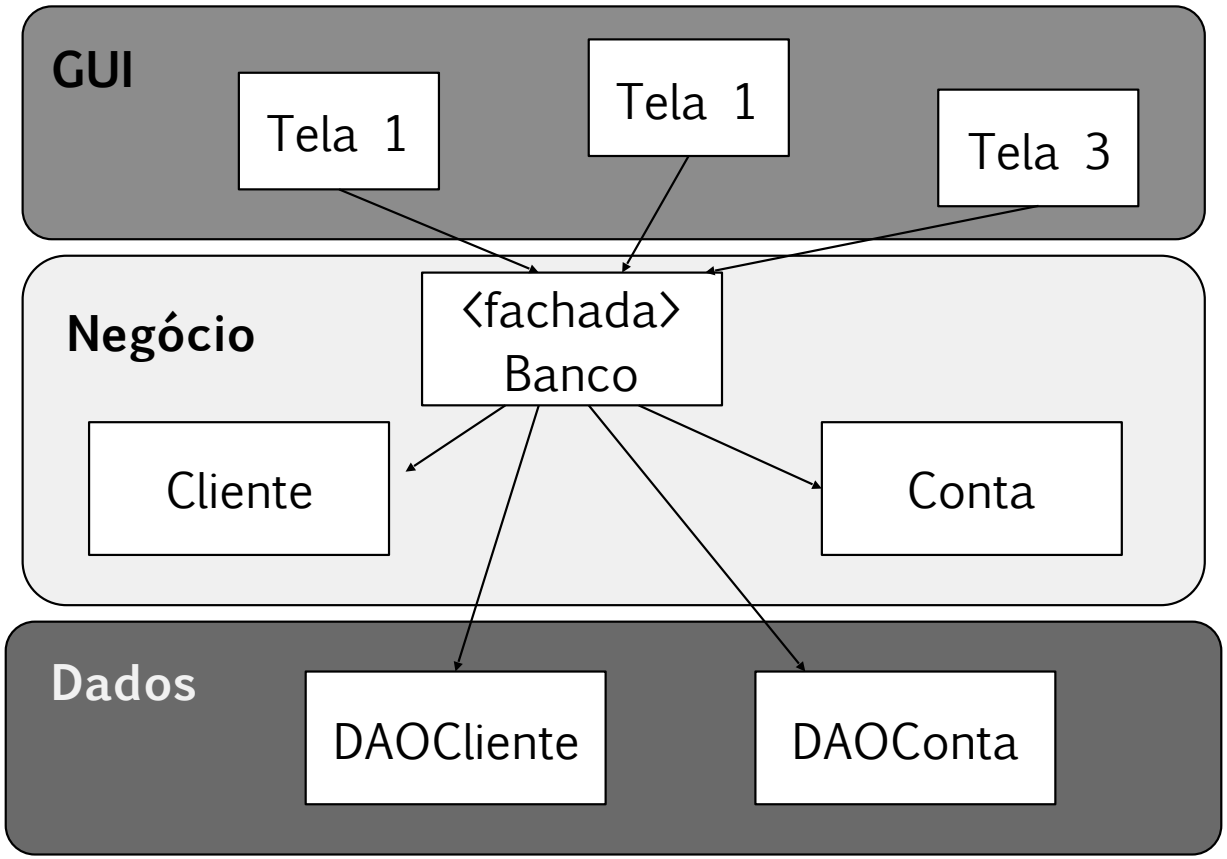
- Motivação
- O framework JUnit
- Passo a passo...
- Diferenças JUnit 3, JUnit4, JUnit5
- Mão na Massa

QualitiInternetBanking

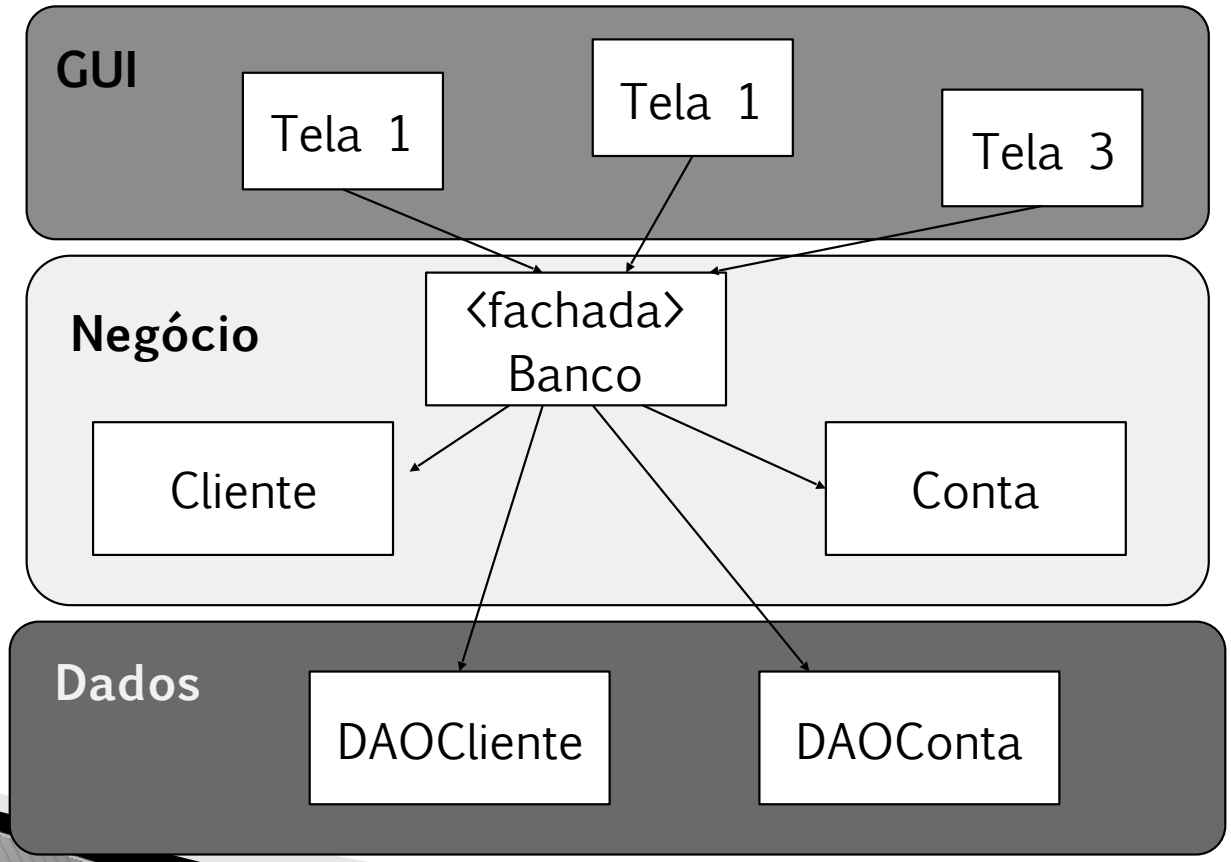
Obs: Um sistema que não tem testes



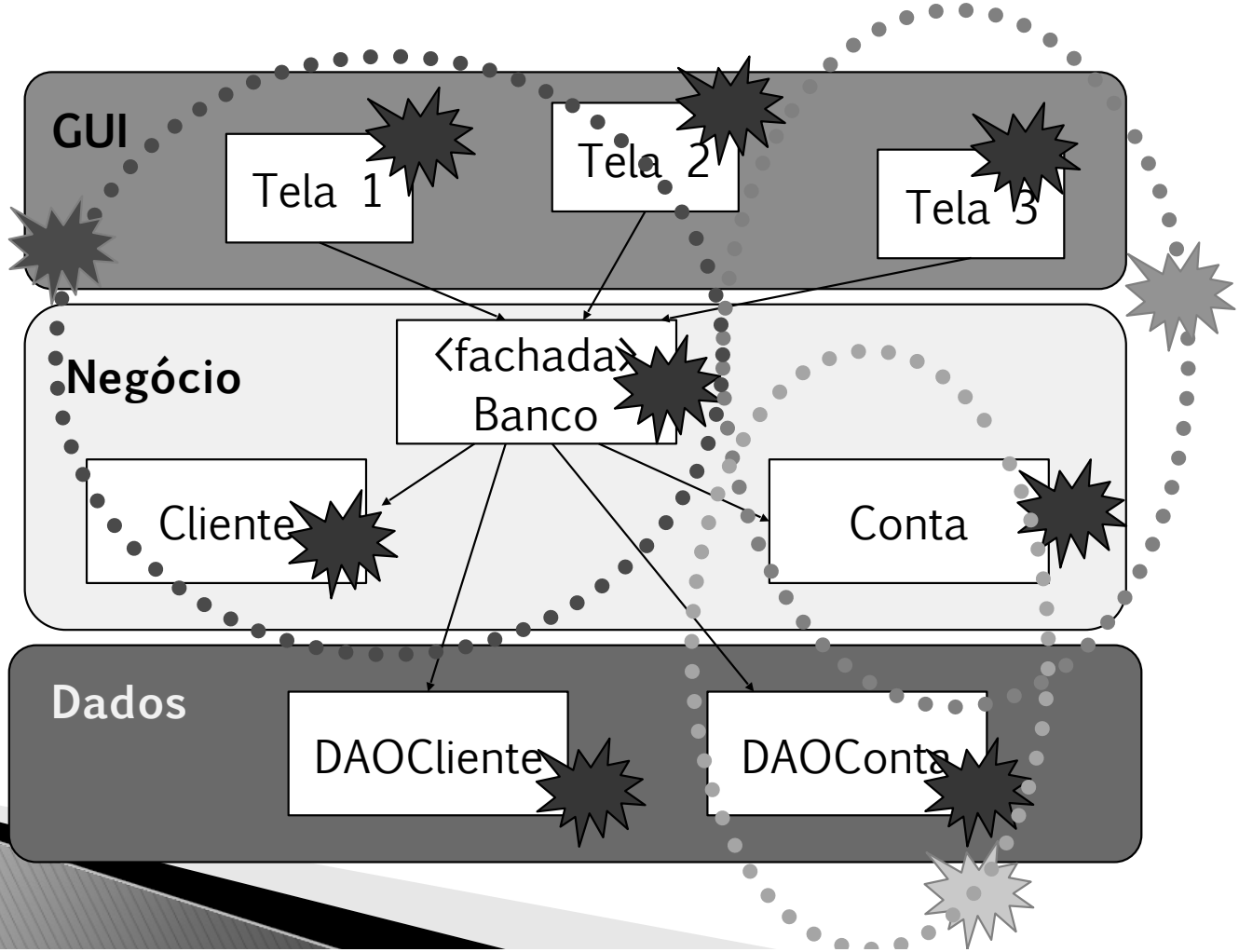
Imagine que...
na release 4.0 a Transferência entre Contas deixa de funcionar...



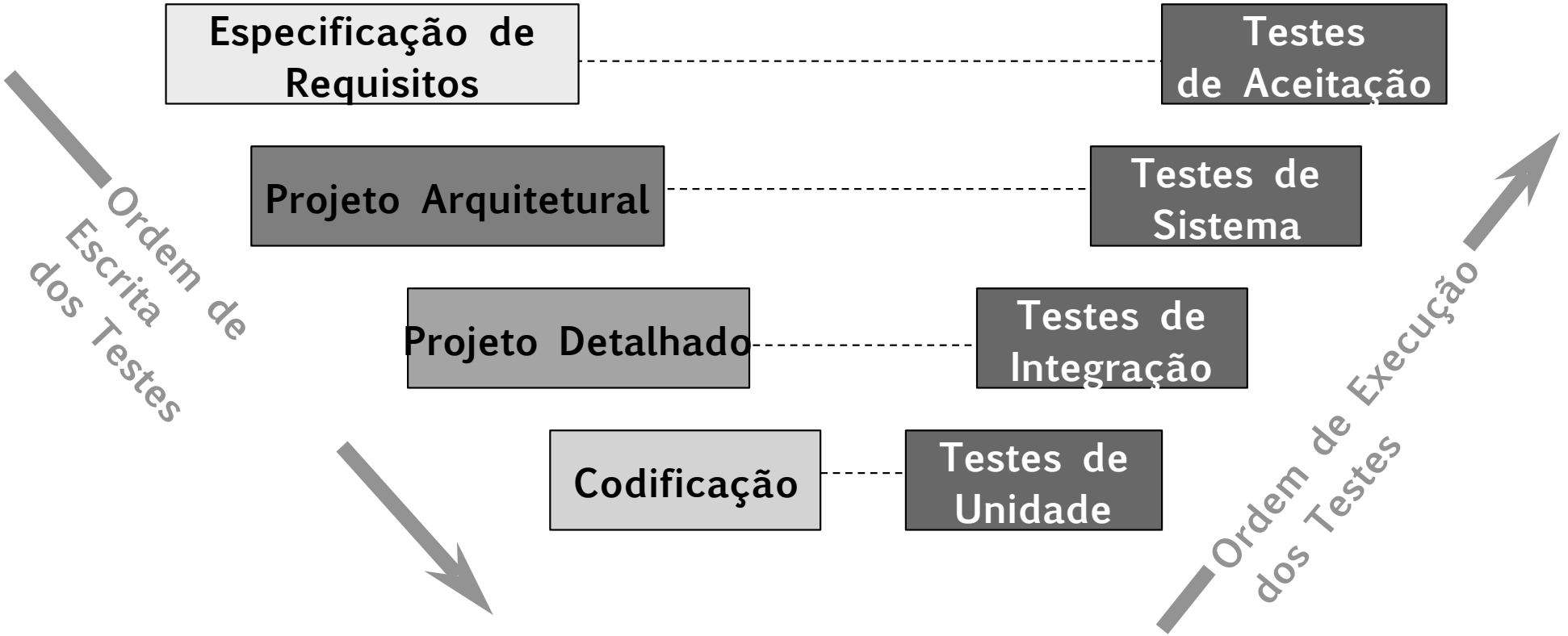
O Bug pode estar em qualquer lugar...



O Bug pode estar em qualquer lugar...



Estágios de Teste



Testes de Sistema

- Vantagem:
  - Ganhamos confiança nos "happy paths" do sistema
- Desvantagens:
  - Custosos.
  - Necessário Debug

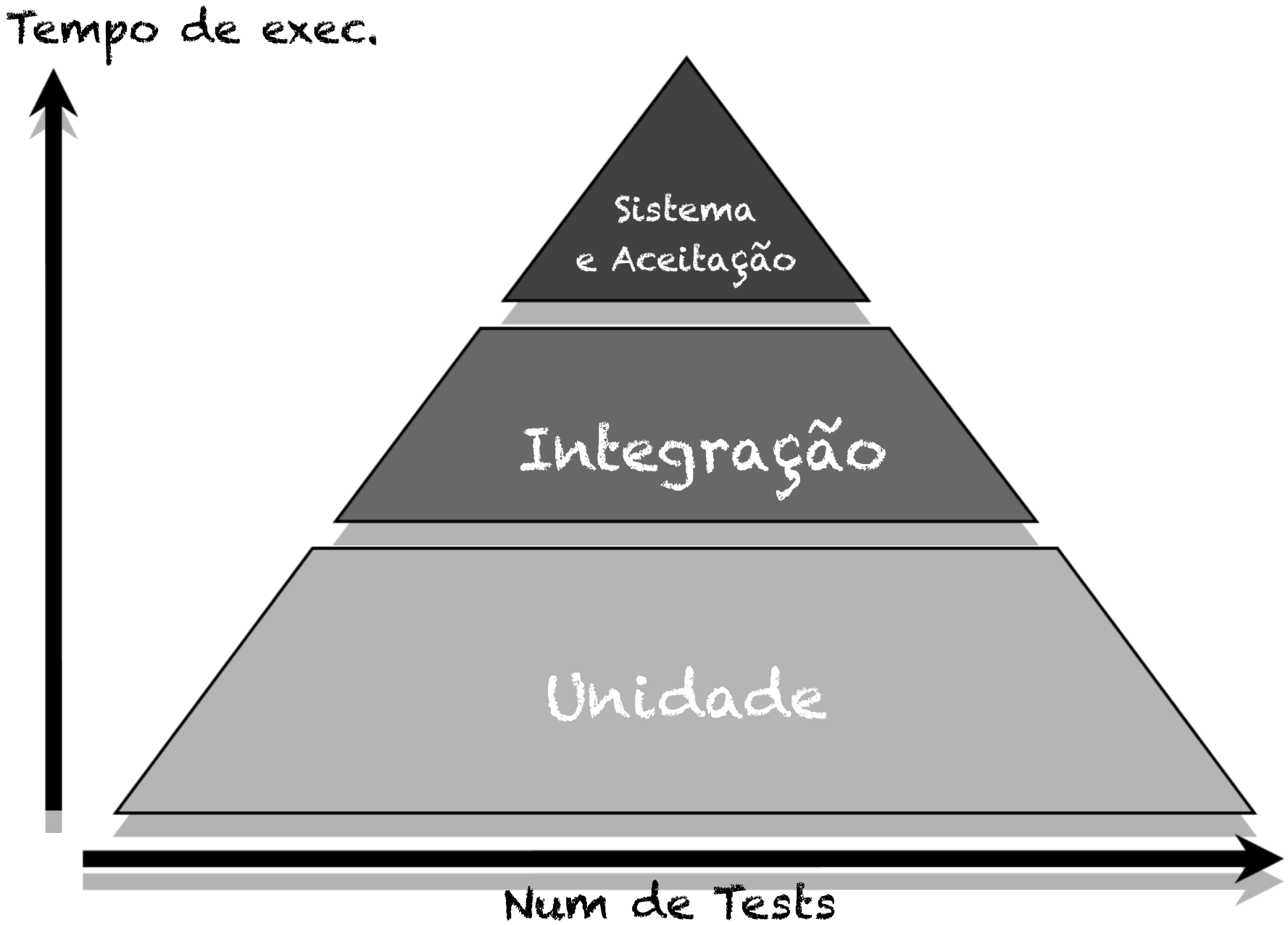
Teste de Integração

- Vantagem:
  - Ganhamos confiança nos subsistemas
  - Testes focam na interação entre as classes
- Desvantagem
  - É mais fácil de simular cenários de falha mas...
  - ... ainda precisamos de debug

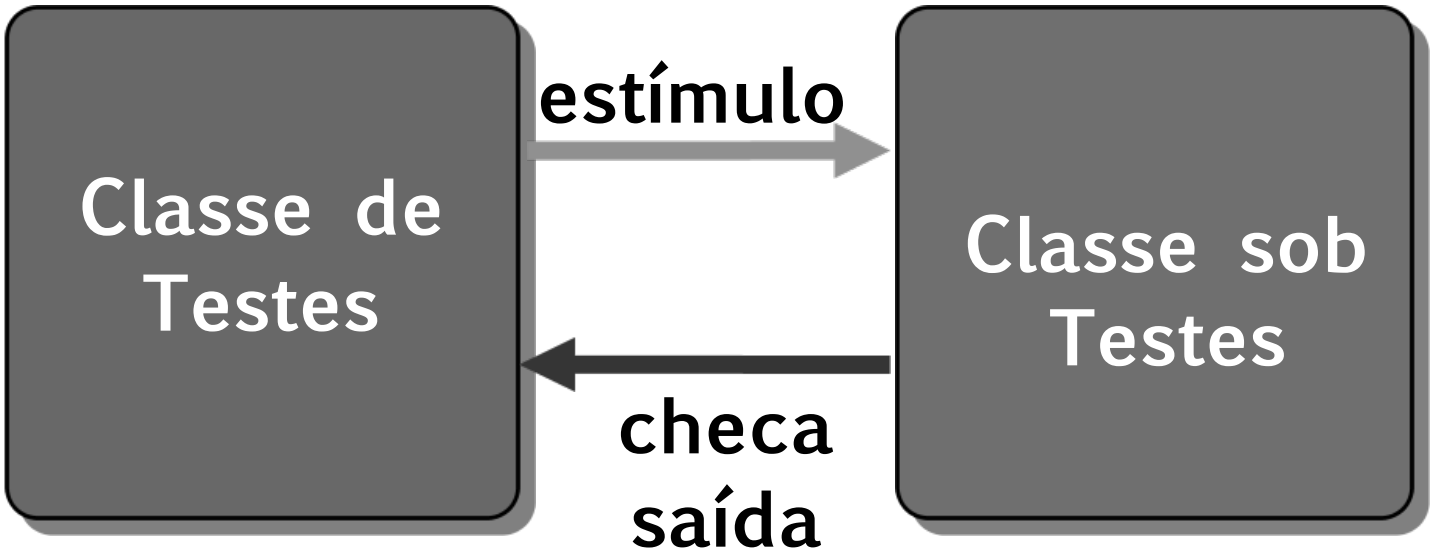
Teste de Unidade

- Vantagem:
  - Testa classes isoladamente
  - Pode simular condições de erro
  - Executa mais rápido que os testes anteriores
- Desvantagem
  - Testa partes do sistema isoladamente
  - Bugs de integração e sistema não são detectados

Testes: Desempenho x Quantidade



Motivação para Teste de Unidae



Quiz:

Testar o método debitar (sem usar o JUnit)

```
public class Conta{

   …
  public Conta (String p_id, double p_saldo) {
        this.saldo = p_saldo;
        this.codigo = p_id;
  }
  public int getSaldo() {…}

  public boolean debitar(int valor) {

   if(valor > 0 && saldo >= valor){
         saldo = saldo-valor;
         return true;
    }else{
      return false;
    }
  }
```

Resposta 1

```
public class TesteConta{
 …
 public void testDebitar() {
    Conta c  = new Conta (“123”, 10);

      c.debitar(5);
      if ( c.getSaldo(5) == 5 ){
         System.out.println(“CORRETO”);
      }else {
         System.out.println(“ERRO”);
       }
   }
```

Resposta 2

```
public class TesteConta{
 …
 public void testDebitar() {
    Conta c  = new Conta ("123", 10);

      if (c.debitar(5)){
         if (  c.getSaldo(5.0) == 5){
            System.out.println("CORRETO");
         }else {
            System.out.println("ERRO");
          }
        }else{
            System.out.println("ERRO");
        }
   }
```

Respostas 1 e 2 (cont.)

```
public class TesteConta{
 …
public static void main (String args[] ){
      TesteConta ct = new TesteConta ();
      ct.testDebitar();
 }

}
```

Quais são as limitações desta solução?

Quiz

Imagine se criarmos vários métodos de testes... deste jeito...

```
public static void main (String args[] ){
     TesteConta ct = new TesteConta ();
      ct.testDebitar(5);
       ct.testDebitarNegativo(-1);
      ct.testCreditar(5);
      ct.testCreditarNegativo(-1);
   ...
 }
```

Quiz

O log vai ficar mais ou menos assim:

```
CORRETO
CORRETO
CORRETO
CORRETO
ERRO
...
```

Quiz

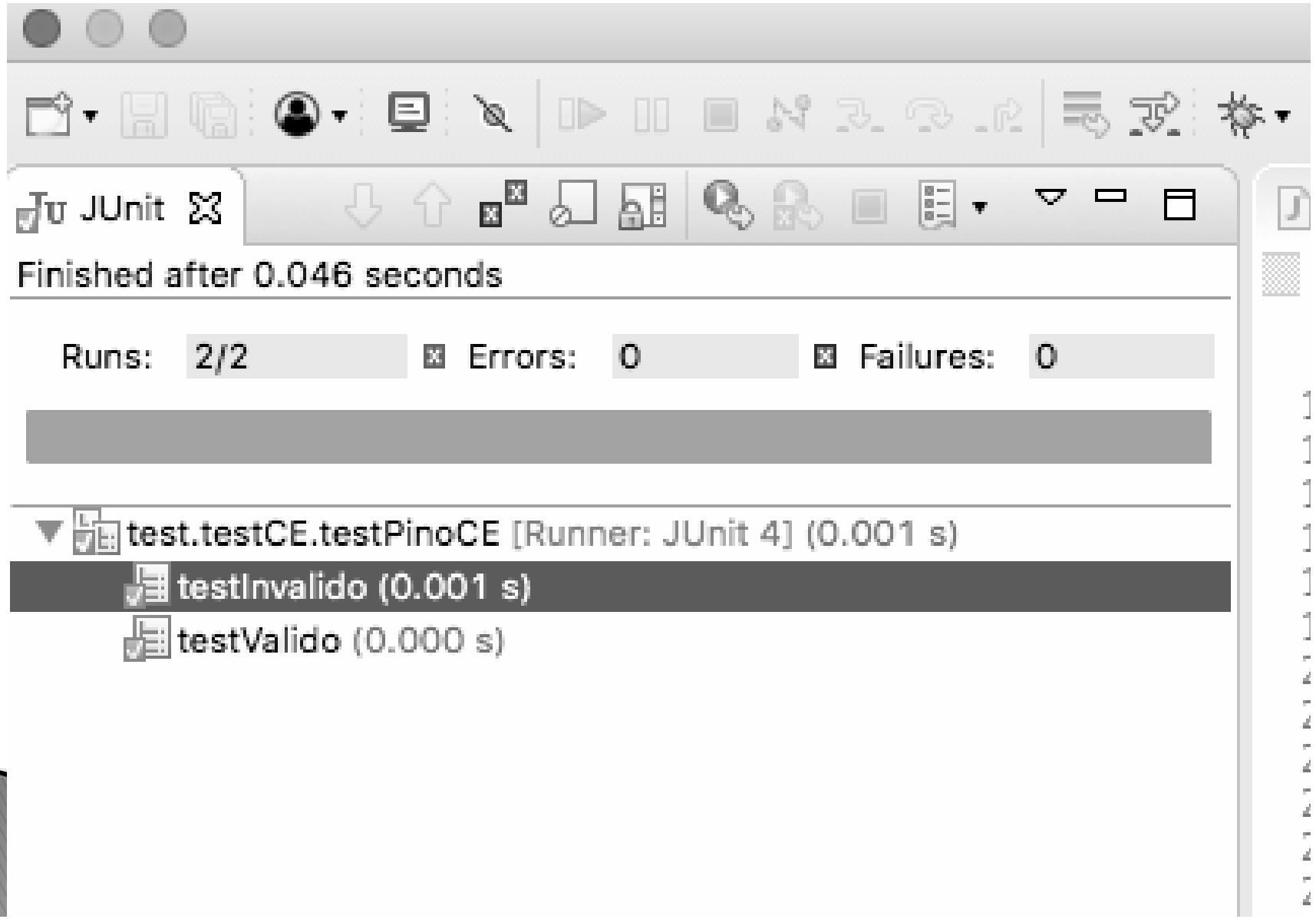
Olhar a saída dos testes no log, dificulta a vida de quem testa...

Como seria com JUnit?

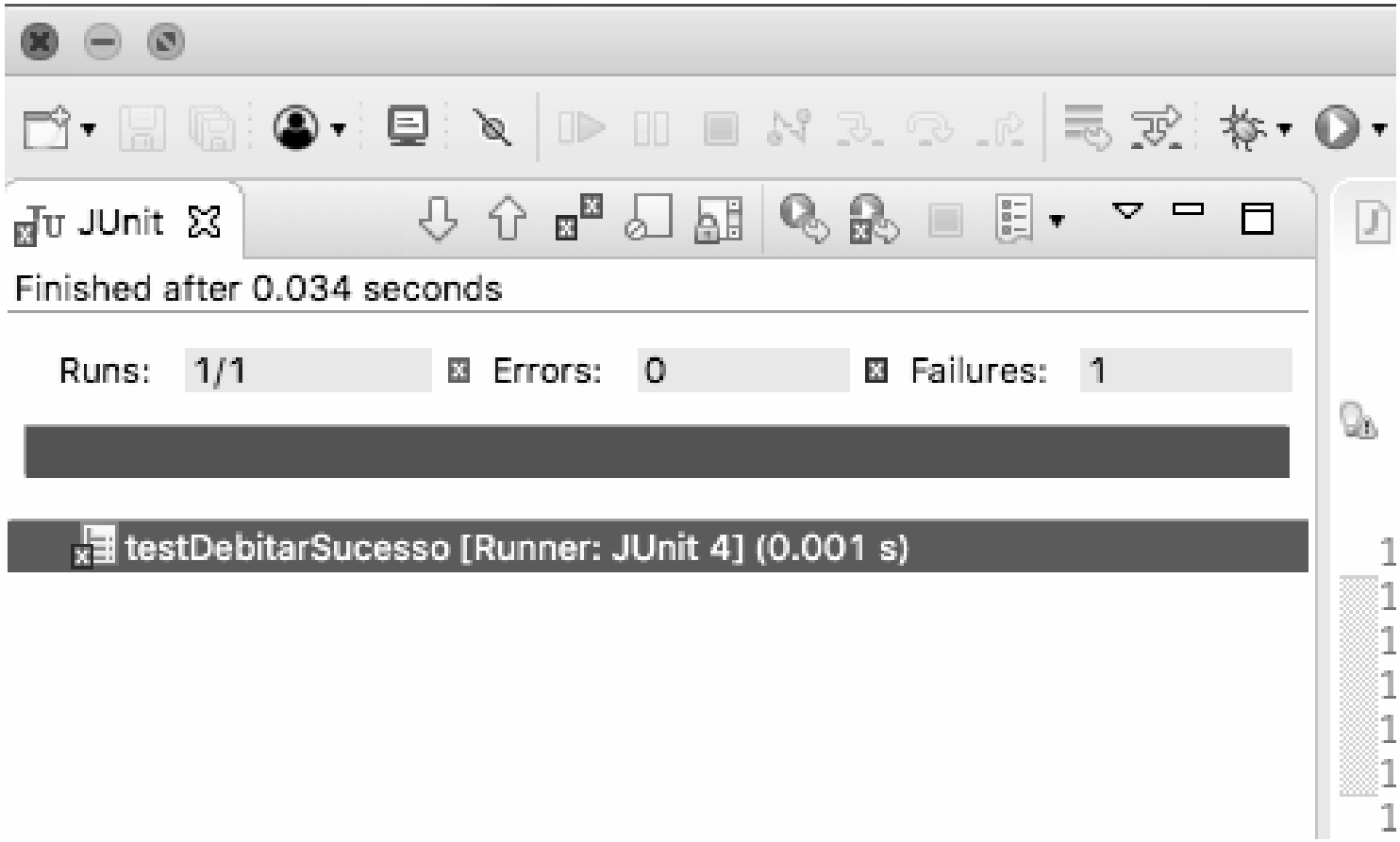
Resposta COM JUNIT

```
public class TesteConta{

 @Test
 public void testDebitar() {
     Conta c  = new Conta ("123", 10);
     assertTrue(c.debitar(5));
     assertEquals(5, c.getSaldo(5));

 }
```

SE O TESTE PASSA - BARA VERDE



SE O TESTE NÃO PASSA - BARRA VEMELHA



Agenda

- Motivação
- O framework JUnit
- Passo a passo...
- Diferenças JUnit 3, JUnit4, JUnit5
- Mão na Massa

Agenda

- Motivação
- O framework JUnit
- Passo a passo...
- Diferenças JUnit 3, JUnit4, JUnit5
- Mão na Massa

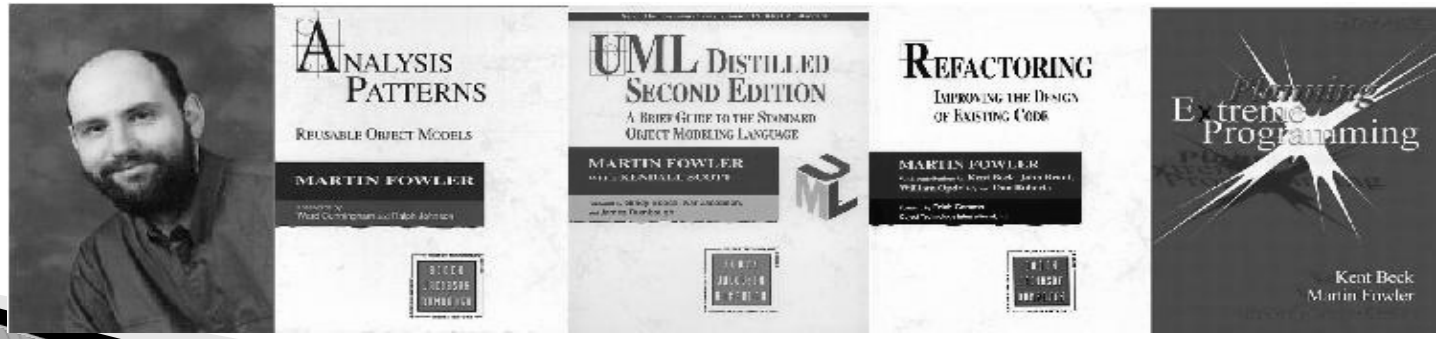
JUnit

"Never in the field of software development was so much owed by so many to so few lines of code."

Martin Fowler

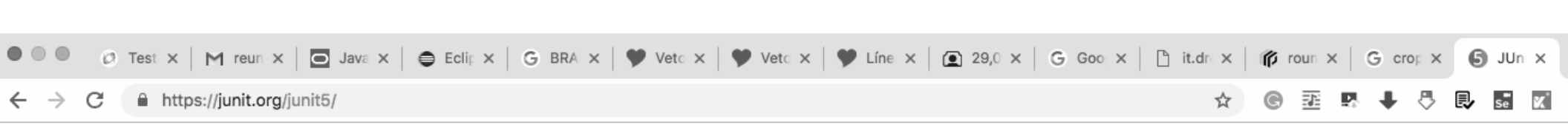
"Nunca o desenvolvimento de software deveu tanto a tão poucas linhas de código."

Martin Fowler



Framework JUnit

Dá suporte a **definição** e **execução** de testes de unidade de programas Java.



JUnit 5

JUnit 4

The new major version of the programmer-friendly testing framework for Java

User Guide | Javadoc | Code & Issues | Q & A | Support JUnit

About

JUnit 5 is the next generation of JUnit. The goal is to create an up-to-date foundation for developer-side testing on the JVM. This includes focusing on Java 8 and above, as well as enabling many different styles of testing.

JUnit 5 is the result of JUnit Lambda and its crowdfunding campaign on Indiegogo.

Resources

Upcoming Events

2018-10-08
Spock vs JUnit 5 - Clash of the Titans at JDD in Cracow, Poland
Marcin Zajączkowski

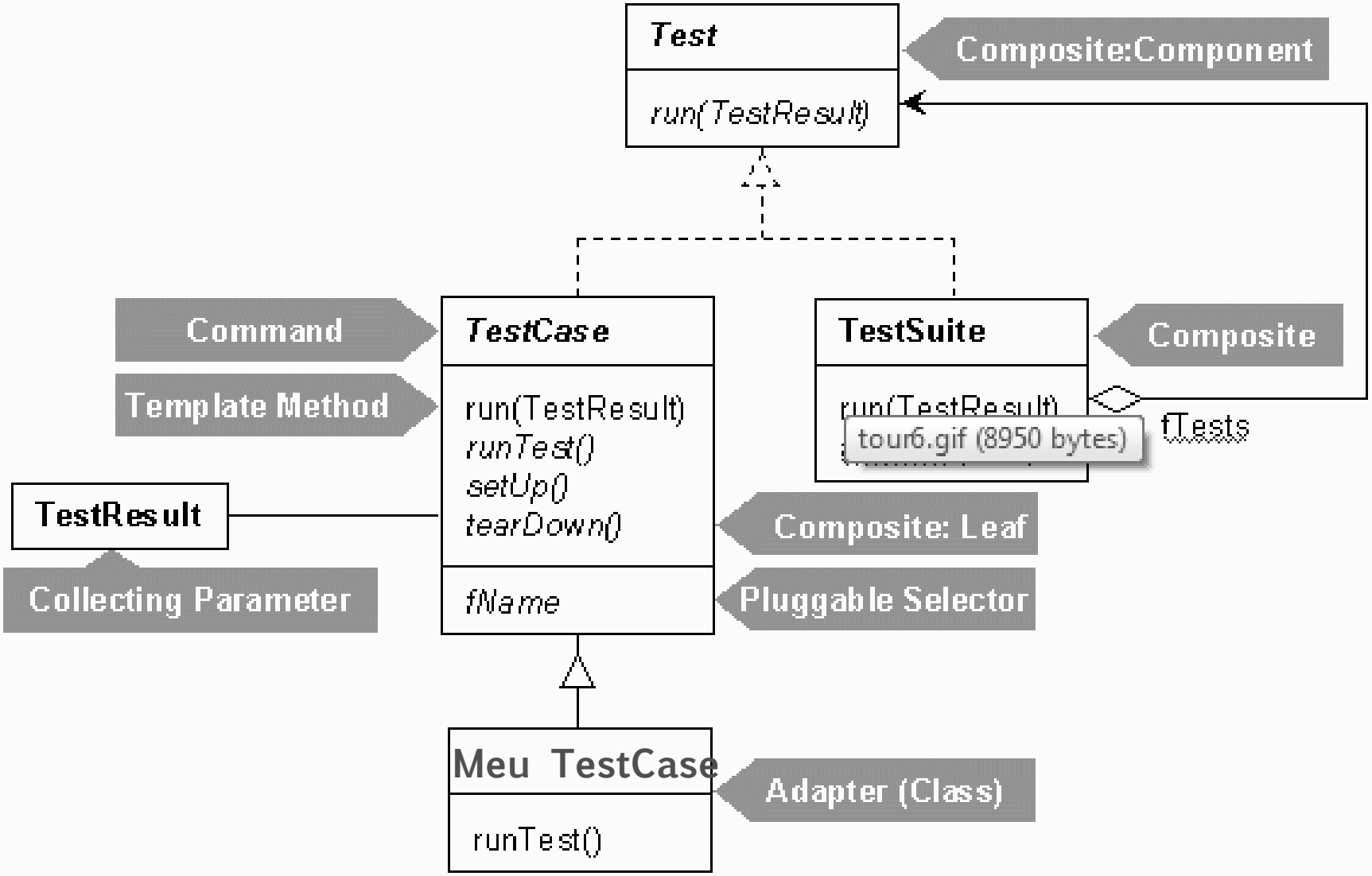
2018-11-06
JUnit 5: Platform & Jupiter API at JUG Bonn in Bonn,

Framework JUnit

- Desenvolvido por Erich Gamma (*Design Patterns*) and Kent Beck (XP).

- Um excelente exemplo da aplicação de vários *Padrões de Projeto*.

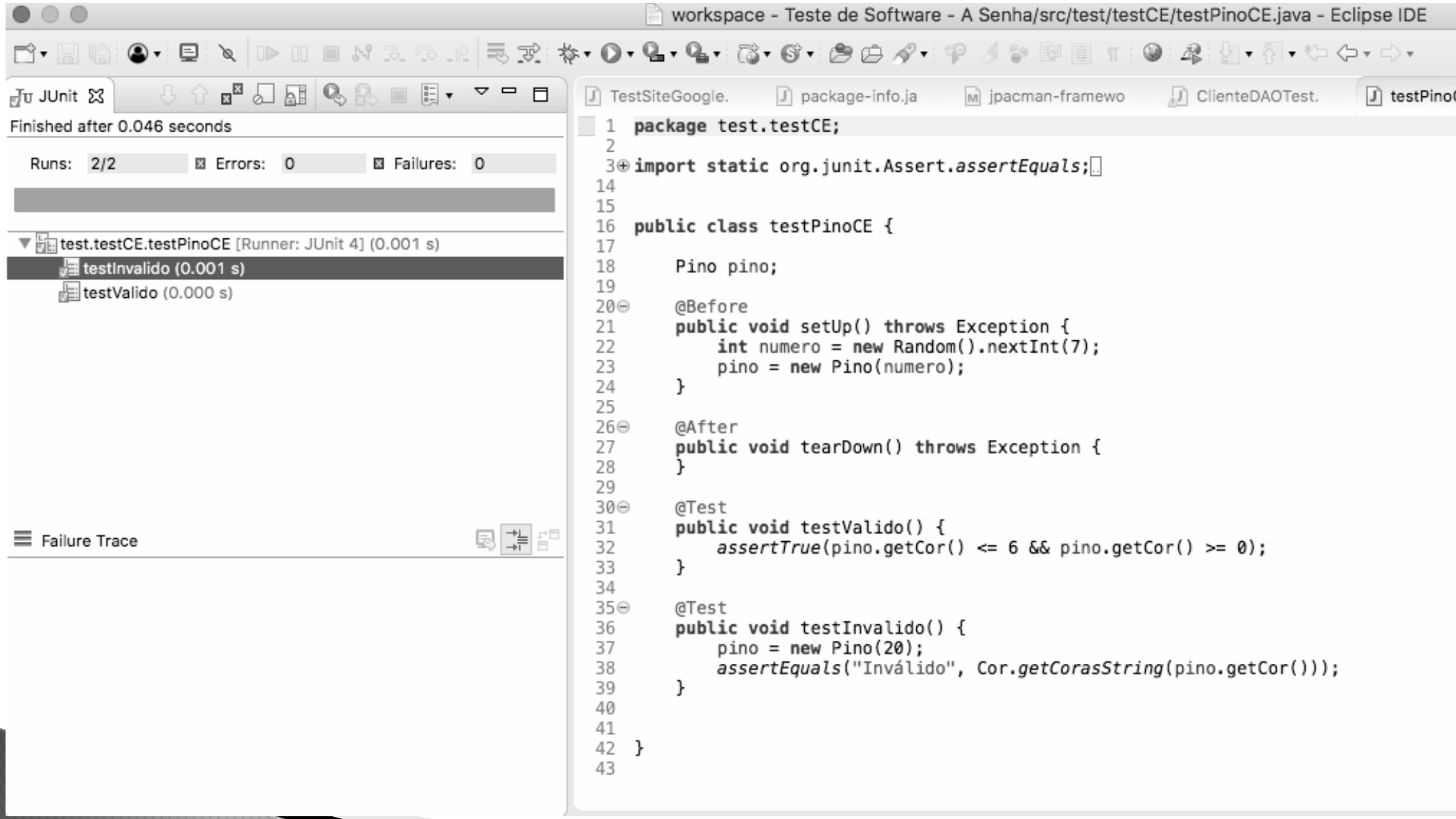
Arquitetura



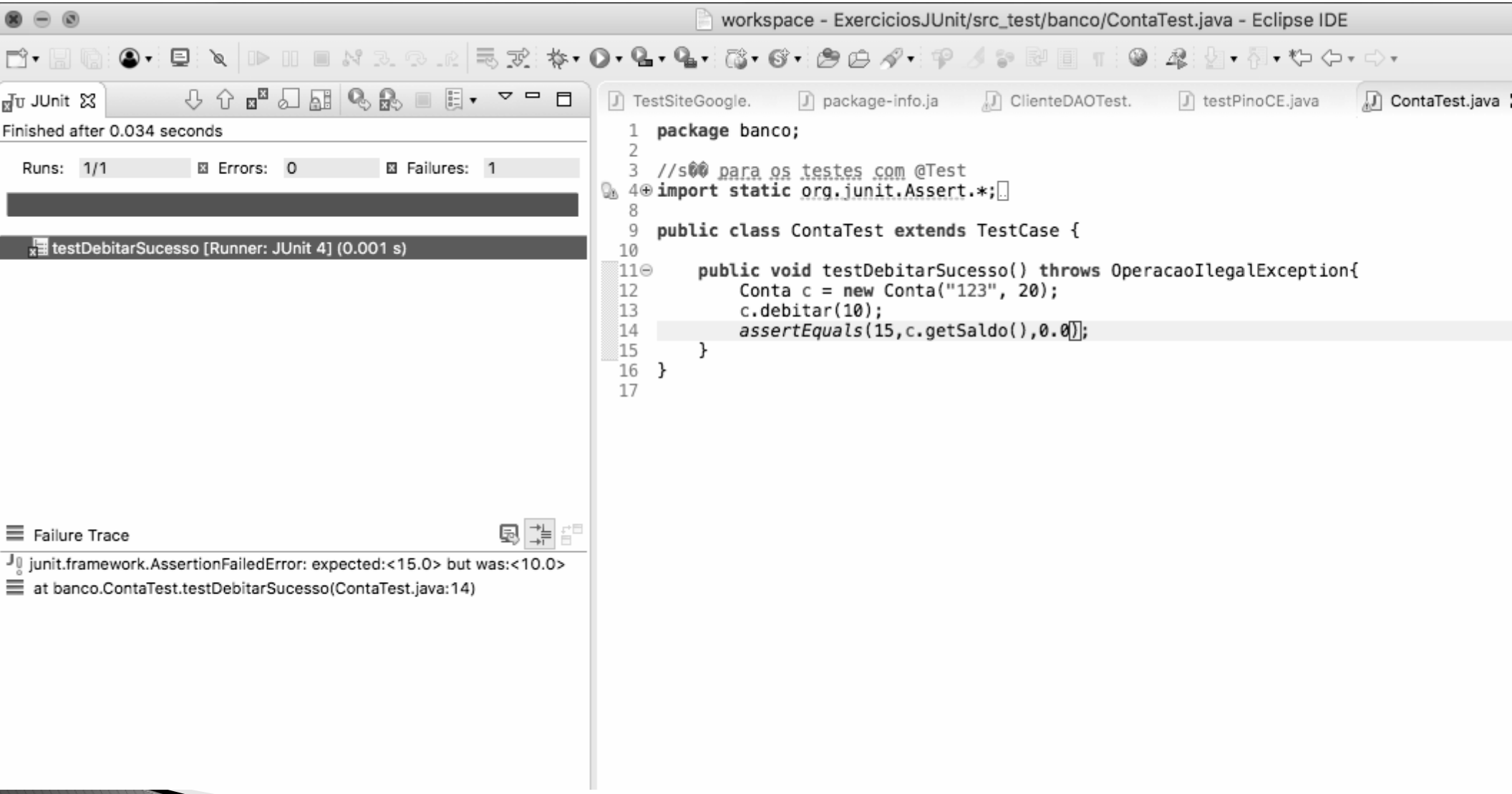
JUnit

- Idéia básica:
  - Para testar uma classe, criamos uma classe de testes correspondente.

Integração com IDEs: Eclipse, NetBeans...



Integração com IDEs: Eclipse, NetBeans...



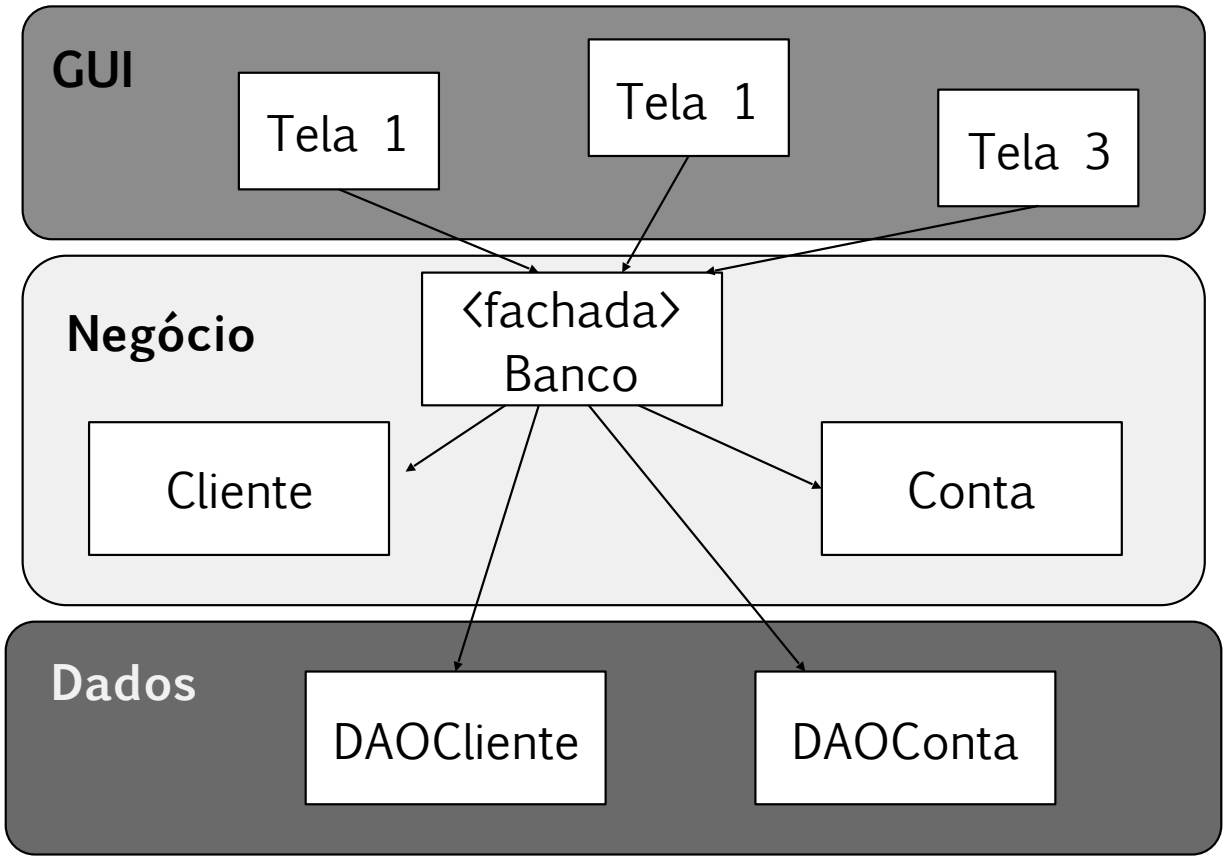
Agenda

- Motivação
- O framework JUnit
- Passo a passo...
- Diferenças JUnit 3, JUnit4, JUnit5
- Mão na Massa

Agenda

- Motivação
- O framework JUnit
- Passo a passo...
- Diferenças JUnit 3, JUnit4, JUnit5
- Mão na Massa

Construindo Testes de Unidade para o QualitiInternetBanking



Passo 1: Selecionar a classe a ser testada.

Test Driven Development:

Implementar o teste de Unidade ser antes da classe a ser testada.

Passo 1 (cont.)

```
public class Conta{
   double saldo;
   String codigo;

  public Conta (String p_id, double p_saldo) {
        this.saldo = p_saldo;
        this.codigo = p_id;
  }

 public void debitar(double valor) throws
          OperacaoIlegalException {
    if(valor > 0 && saldo >= valor){
         saldo = saldo-valor;
    }else{
      throw new OperacaoIlegalException();
    }
  }
```

Passo 1 (cont.)

```
  ...
  public void creditar(double valor) throws
                              OperacaoIlegalException{
      if(valor > 0){
         saldo = saldo+valor;
      }else{
            throw new OperacaoIlegalException();
       }
    }

    //Retorna true se o saldo e o numero da conta
    //forem os mesmos e false caso contrario.
    public boolean equals (Conta c2){
   ...
    }
}
```

Passo 2: Construir a classe de testes

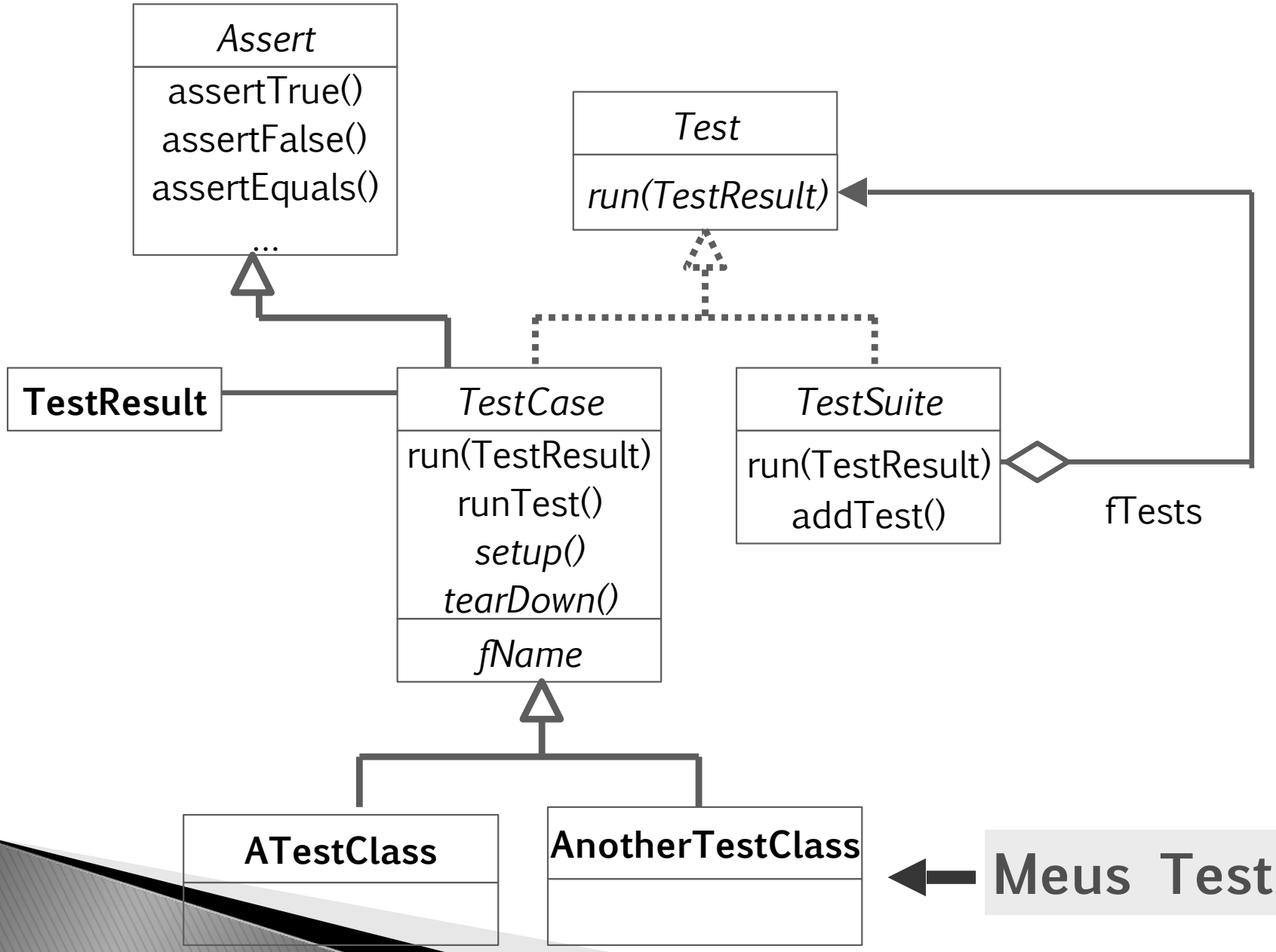
- Herda da classe TestCase do framework JUnit. (**Versão 3.x**)
- Padrão de nomenclatura sugerido:
  - <Nome da Classe> + Test = ContaTest

Passo 2: Construir a classe de testes

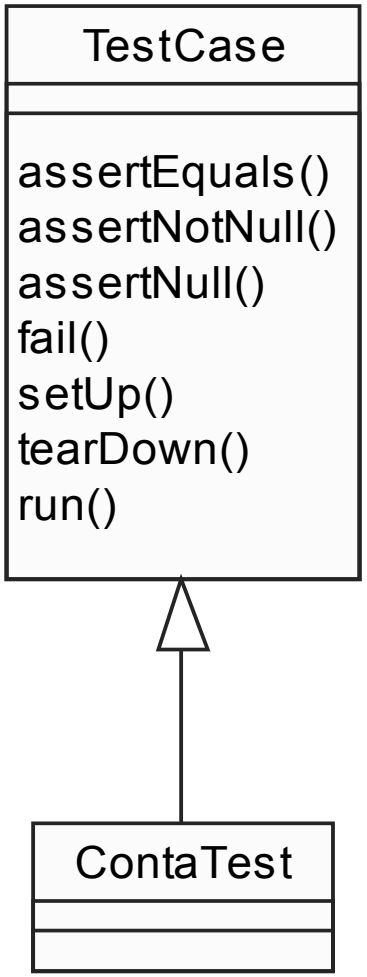
- Herda da classe TestCase do framework JUnit. (**Versão 3.x**)
- Padrão de nomenclatura sugerido:
  - <Nome da Classe> + Test = ContaTest

Ferramentas XUnit (PHP, C++) se assemelham a versão 3.

Passo 2: Construir a classe de testes



Passo 2: Construir a classe de testes



Passo 3: Criar os métodos de teste.

- Restrição até a versão 3.x:
  - devem iniciar com a palavra “test” e
  - possuir tipo de retorno void.

- Devem testar condições de sucesso e falha.

Passo 3.1: Testando cenários de sucesso

```
1. public class ContaTest extends TestCase {

2.     public void testSaqueBemSucedido(){

3.         Conta c1 = new Conta("123", 50.0);

4.         c1.debitar(49.0);

5.         assertEquals(c1.getSaldo(),1.0);
6.     }
7. }
```

Passo 3.2: Testando cenários de falha - Lançamento de Exceções

```
   public void testSaqueUltrapassaLimite(){

      Conta c1 = new Conta ("123", 50.0);
      c1.debitar(70.0);

   }
  ...
}
```

Este teste está correto?
Vamos executar...

Java - Banco/src/br/ufrn/sinfo/ContaTest.java - Eclipse SDK

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer | Hierarchy | JUnit

Finished after 0,047 seconds

Runs: 2/2 Errors: 1 Failures: 0

br.ufrn.sinfo.ContaTest [Runner: JUnit 3] (0,000 s)

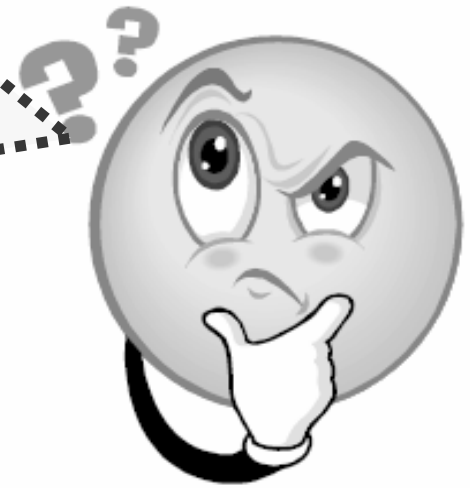
testSaqueBemSucedido (0,000 s)

testSaqueUltrapassaLimite (0,000 s)

Conta.java | OperacaoIlegalExcept | ContaTest.java

```
public void testSaqueBemSucedi

    Conta c1 = new Conta("123'
    c1.debitar(49.0);
    assertEquals(c1.getSaldo()

}

public void testSaqueUltrapass

    Conta c1 = new Conta ('
    c1.debitar(70.0);
```

Mas se a exceção foi lançada o comportamento do método esta correto. O teste só deve falhar quando um defeito é encontrado.



Possíveis Resultados

- *Sucesso*
- *Falha*
- *exceção* **ou Error**

Possíveis Resultados

- **Sucesso**
- **Falha**
- **exceção ou Error**

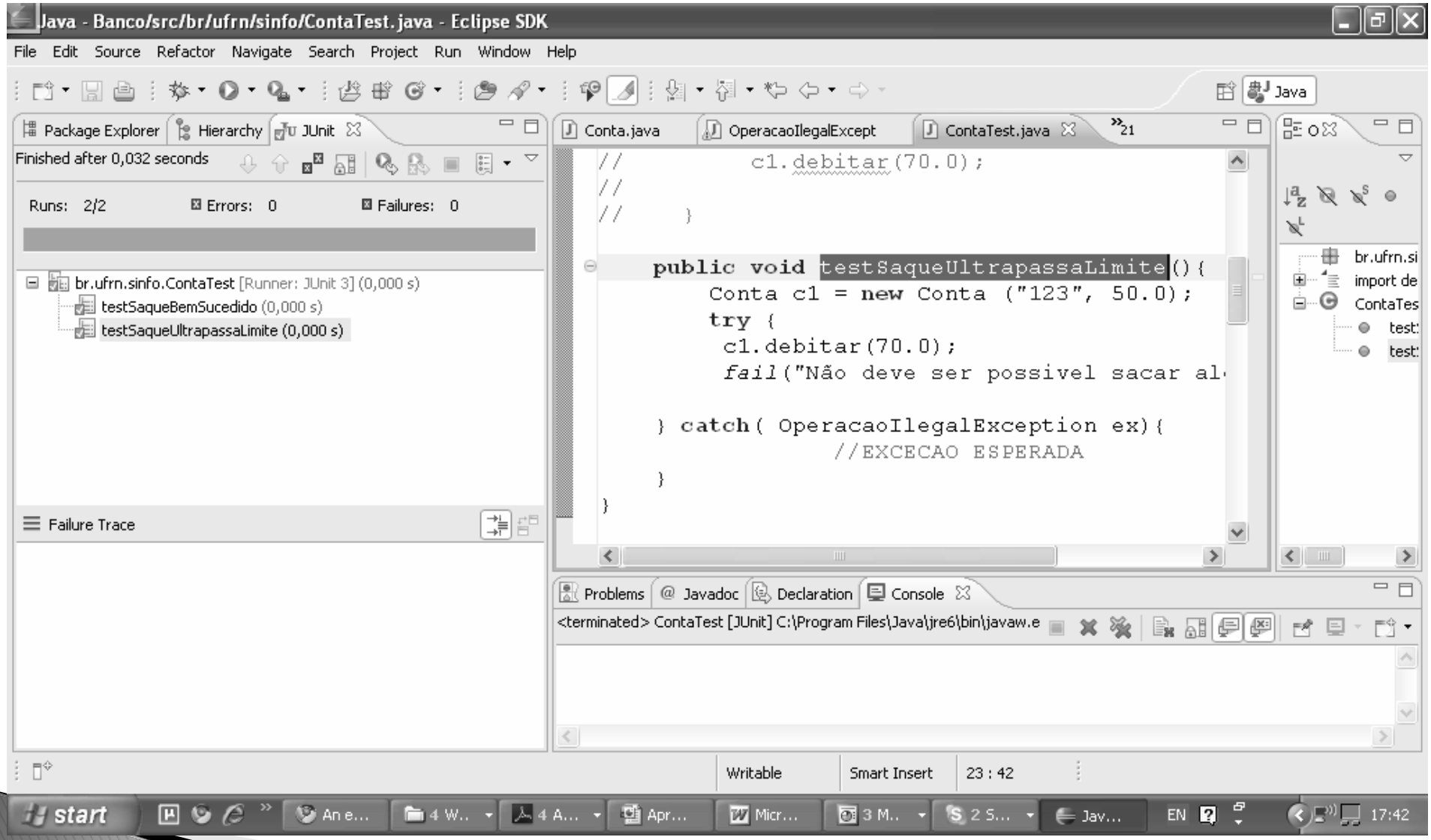
Algum dos métodos assert lançou AssertionFailException

Uma exceção que não foi lançada pelo JUnit escapou!!

Versão correta do teste

```
12.     public void testSaqueUltrapassaLimite(){

13.       Conta c1 = new Conta ("123", 50.0);

14.         try {
16.            c1.debitar(70.0);

17.            fail("Não deve ser possivel sacar
                    além do saldo!");

18.         } catch( OperacaoIlegalException ex){
19.             //EXCECAO ESPERADA
20.         }
21.      }
   ...
}
```

Nova execução



Passo 4: Organizar s configurações comuns aos métodos de teste.

- **setUp():**
  - chamado automaticamente antes de cada método de testes. (alocação de recursos, inicialização de objetos)
- **tearDown():**
  - chamado automaticamente ao final de cada método de testes (liberação de recursos)

Passo 4: **Organizar s configurações comuns aos métodos de teste.**

```
public class ContaTest extends TestCase {

  Conta c1;

  public void setUp(){
     c1 = new Conta("123", 50.0);
  }
  ...
}
```

Agenda

- Motivação
- O framework JUnit
- Passo a passo...
- Diferenças JUnit 3, JUnit4, JUnit5
- Mão na Massa

Agenda

- Motivação
- O framework JUnit
- Passo a passo...
- Diferenças JUnit 3, JUnit4, JUnit5
- Mão na Massa

JUnit 4

JUnit4 - Principais Diferenças

- Necessita: Java 5 ou superior
- Anotações no lugar de nomes padrão.
  - **@Test —> "testXXX"**
  - **@Before —> setUp()**
  - **@After —> tearDown()**

JUnit4 - Principais Diferenças

- JUnit 4 pode executar testes em JUnit 3
- Classe de testes **não herda de TestCase**
- `import static org.junit.Assert.*` substitui herança

Teste na versão JUnit 4.x

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

public class ContaTestNovo {

 @BeforeClass
 public static void setUpClass() throws Exception {
 }

 @AfterClass
 public static void tearDownClass() throws Exception {
 }
```

Mesmo Teste na versão JUnit 4.8.2

```
…

 @Before
 public void setUp() throws Exception {
 }

 @After
 public void tearDown() throws Exception {
 }

 @Test
 public void testDebitar() {
      …
 }
...

}
```

Testando o lançamento de exceções

```
@Test (expected=OperacaoIlegalException.class)
public void testSaqueUltrapassaLimite(){

    Conta c1 = new Conta ("123", 50.0);
    c1.debitar(70.0);

}
```

Testando o lançamento de exceções

```
@Test(timeout=500)
public void testarCretido() {
    …
}

@Ignore @Test
public void getSaldoTest() {

}
```

Mesmas Assertivas

```
@Test
public void exemplosAsserts() {
    ...
   //Verifica que uma condição é verdadeira.
   assertTrue(result)

   //Verifica que uma condição é falsa.
   assertFalse(result)

   //Verifica que uma referência não é null.
   assertNotNull(result)

  //Verifica que uma referência é null.
  assertNull (result)
```

Mesmas Assertivas

```
public void testarCretido() {
    ...
    //Verifica que duas referências apontam
    para o mesmo objecto.
   assertSame(esperado, result)

    //Verifica que duas referências não apontam para
      o mesmo objecto.
   assertNotSame(esperado, result)

   //Verifica que os conteúdos de dois arrays são
     iguais.
   assertArrayEquals(arEsperado, arResult)

   //Verifica se dois objetos são iguais – equals()
   assertEquals(esperado, result)
```

Criando uma Suite

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

@RunWith(Suite.class)
@Suite.SuiteClasses({ ClasseTeste1.class,
ClasseTeste2.class })
public class AllTests {

   /* Classe vazia apenas para agrupar
   todos os testes */

}
```

Executando uma Suite

```
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;

public class JunitRunner {

   public static void main(String[] args) {

       Result result =
       JUnitCore.runClasses(AllTests.class);

       for (Failure fail : result.getFailures())
        System.out.println(fail.toString());

       if(result.wasSuccessful())
        System.out.println("Todos os testes OK.");
   }
}
```

Executando uma Suite

- Com o org.junit.runner.JUnitCore podemos executar testes JUnit a partir de uma classe Java.

- O método runClasses() recebe uma classe de teste como argumento e retorna um valor do tipo Result.

- O valor retornado contém uma lista de resultados que permite determinar o sucesso/insucesso de cada teste.

Mão na Massa

Mão na Massa 1

- Remova os comentários de forma incremental e analise e corrija os testes da classe JUnitBasicoTest.

Mão na Massa 2

- Crie casos de testes o método transferir da classe Conta: 1 para para o *happy path* e 1 para o cenário em a `OperacaoIlegalException` deve ser lançada.

```
/* Este método transfere o valor da conta origem
para a conta destino. Se o valor for negativo ou
superior que o saldo da conta origem uma
OperacaoIlegalException é lançada*/
```

```
public void transferir (Conta destino, double
 valor) throws OperacaoIlegalException
```

Mão na Massa 3

- Implemente caos de teste de unidade para classe Fila.

- Dicas: Focar no objetivo desta estrutura de dados. Checar se método inserir insere no final da fila e se o método de remover da fila remove no início da fila.

JUnit 4

Testes Parametrizados

https://junit.org/junit4/javadoc/4.12/org/junit/runners/Parameterized.html

Testes Parametrizados

Usando as anotações como `@RunWith(Parameterized.class)` e `@Parameters` podemos executar o mesmo teste para múltiplos datasets.

```
@RunWith(Parameterized.class)
public class FibonacciTest {

    private int entrada;
    private int esperado;

    public FibonacciTest (int in, int esp) {
        entrada = in;
        esperado = esp;
    }

   @Parameters
   public static Object[][] data() {
        return new Object[][] {
                { 0, 0 },
                { 1, 1 },
                { 2, 1 },
                { 6, 8 } };
    }

    @Test
    public void test() {
        assertEquals(esperado, Fibonacci.compute(entrada));
    }
}
```

```
@RunWith(Parameterized.class)
public class FibonacciTest {

    private int entrada;
    private int esperado;

    public FibonacciTest (int in, int esp) {
        entrada = in;
        esperado = esp;
    }

   @Parameters(name= "{index}: fib[{0}]={1}")
   public static Object[][] data() {
        return new Object[][] {
                { 0, 0 },
                { 1, 1 },
                { 2, 1 },
                { 6, 8 } };
    }

    @Test
    public void test() {
        assertEquals(esperado, Fibonacci.compute(entrada));
    }
}
```

```
@RunWith(Parameterized.class)
public class FibonacciTest {

    private int entrada;
    private int esperado;

    public FibonacciTest (int in, int esp) {
        entrada = in;
        esperado = esp;
    }

    @Parameters(name= "{index}: fib[{0}]={1}")
    public static Object[][] data() {
        return ne  Object[][] {

}
```

{index} - o indice do parâmetro corrente

{0} - o valor para o primeiro parâmetro

{1} - o valor para o segundo parâmetro

```
@RunWith(Parameterized.class)
public class FibonacciTest {


 @Parameter(0)
 public int entrada;

 @Parameter(1)
 public int esperado;


 @Parameters
 public static Iterable<Object[]> data() {
     return Arrays.asList(new Object[][] {
                 { 0, 0 }, { 1, 1 }, { 2, 1 },
                 { 3, 2 }, { 4, 3 }, { 5, 5 }, { 6, 8 } });
 }

@Test
 public void test() {
     assertEquals(esperado, Fibonacci.compute(entrada));
 }
}
```


Mão na Massa

Mão na Massa 4

Crie testes parametrizados para testar o método **calculaImposto** da classe **CalculoImpostoRenda**. Refatore o método se necessário.

Mão na Massa 5

Crie casos de teste parametrizados para o método **converte()** da classe Conversor Temperatura, que converte temperaturas de Celsius para Fahrenheit e vice-versa.

Unit 5

- Mundanças -

JUnit5

Necessita do Java 8

Suporte a expressões Lambda. (n) -> n*n

Principal mudança foi arquitetural.

Passou a ser composto por 3 projetos.

JUnit 5 = JUnit Platform + JUnit Jupiter + JUnit Vintage

JUnit5 - Partes

- **JUnit Platform** - suporte ao desenvolvimento de frameworks de teste.
- **JUnit Jupiter** - permite construção de tests no JUnit 5.
- **JUnit Vintage** - suporta a execução de testes JUnit 3 e JUnit 4 (backward compatibility).

JUnit5 x JUnit4
Algumas Diferenças

JUNIT 4	JUNIT 5
`@Test`	`@Test`
`@BeforeClass`	`@BeforeAll`
`@AfterClass`	`@AfterAll`
`@Before`	`@BeforeEach`
`@After`	`@AfterEach`
`@Ignore`	`@Disabled`

Testes Parametrizados no JUnit5

https://junit.org/junit5/docs/current/user-guide/#writing-tests-parameterized-tests

```
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.ValueSource;

import static
  org.junit.jupiter.api.Assertions.assertNotNull;

@DisplayName(“Testes Param Exemplo Classe")
class ExemploTest {

    @DisplayName(“Nome do Teste")
    @ParameterizedTest
    @ValueSource(strings = {"Hello", "World"})
    void metodoTesteParam(String message) {
        assertNotNull(message);
    }

}

//Apenas 1 parâmetro
```

@ValueSource pode ser:

short

byte

int

long

float

double

char

java.lang.String

java.lang.Class

```
@ParameterizedTest
@MethodSource("stringIntAndListProvider")
void testWithMultiArgMethodSource
            (String str, int num, List<String> list) {

        assertEquals(3, str.length());
        assertTrue(num >=1 && num <=2);
        assertEquals(2, list.size());

}

static Stream<Arguments> stringIntAndListProvider() {

    return Stream.of(
        arguments("foo", 1, Arrays.asList("a", "b")),
        arguments("bar", 2, Arrays.asList("x", "y"))
    );

}
```

//Método gerador

```
@ParameterizedTest
@CsvFileSource(resources = "/arq.csv", numLinesToSkip = 1)
void test(String first, int second) {
    assertNotNull(first);
    assertNotEquals(0, second);
}
```

_ _ _ _ _ _ _ _ _ _ _ _ _

arq.csv

```
Country, reference
Sweden, 1
Poland, 2
"United States of America", 3
```

//Arquivo com dados de teste

No curso focaremos no

JUnit4

pois algumas APIs e ferramentas que utilizaremos ainda não suportam o JUnit5


Mão na Massa

Mão na Massa 6

Crie casos de teste JUnit para testar o Jogo Senha.

- Execute o Jogo.
- Analise e selecione as classes a serem testadas
- *Refatore* o código quando necessário para viabilizar a criação dos testes unitários.

UFRN
UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

Extra:
Test Driven Development (TDD)

Profa. Roberta Coelho
Departamento de Informática e Matemática Aplicada - DIMAp

Objetivos

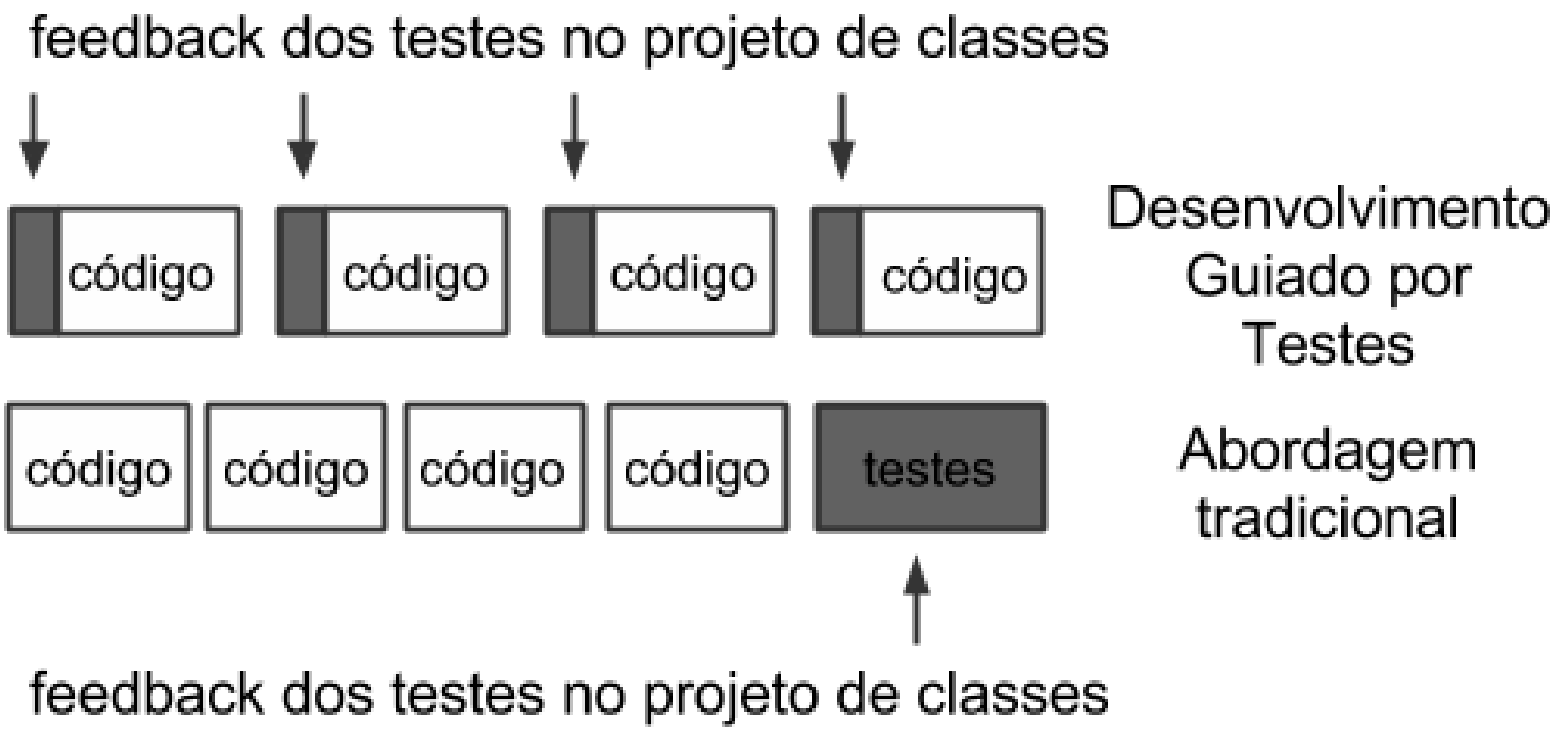
- Discutir sobre TDD na Pratica!

Benefícios Prometidos

1 - Feedback

a granulosidade fina do teste-então-codifique permite contínuo feedback ao desenvolvedor.

1 - Feedback


feedback dos testes no projeto de classes
código
código
código
código
Desenvolvimento Guiado por Testes
código
código
código
código
testes
Abordagem tradicional
feedback dos testes no projeto de classes

Benefícios Prometidos

2 - Ajuda no design da solução

Os testes servem de especificação do comportamento dos métodos (e possíveis exceções).

Benefícios Prometidos

3 - Inclusão x Detecção

Reduzir o tempo entre a inclusão e a detecção do bug.

Benefícios Prometidos

4 - Testes Automatizados

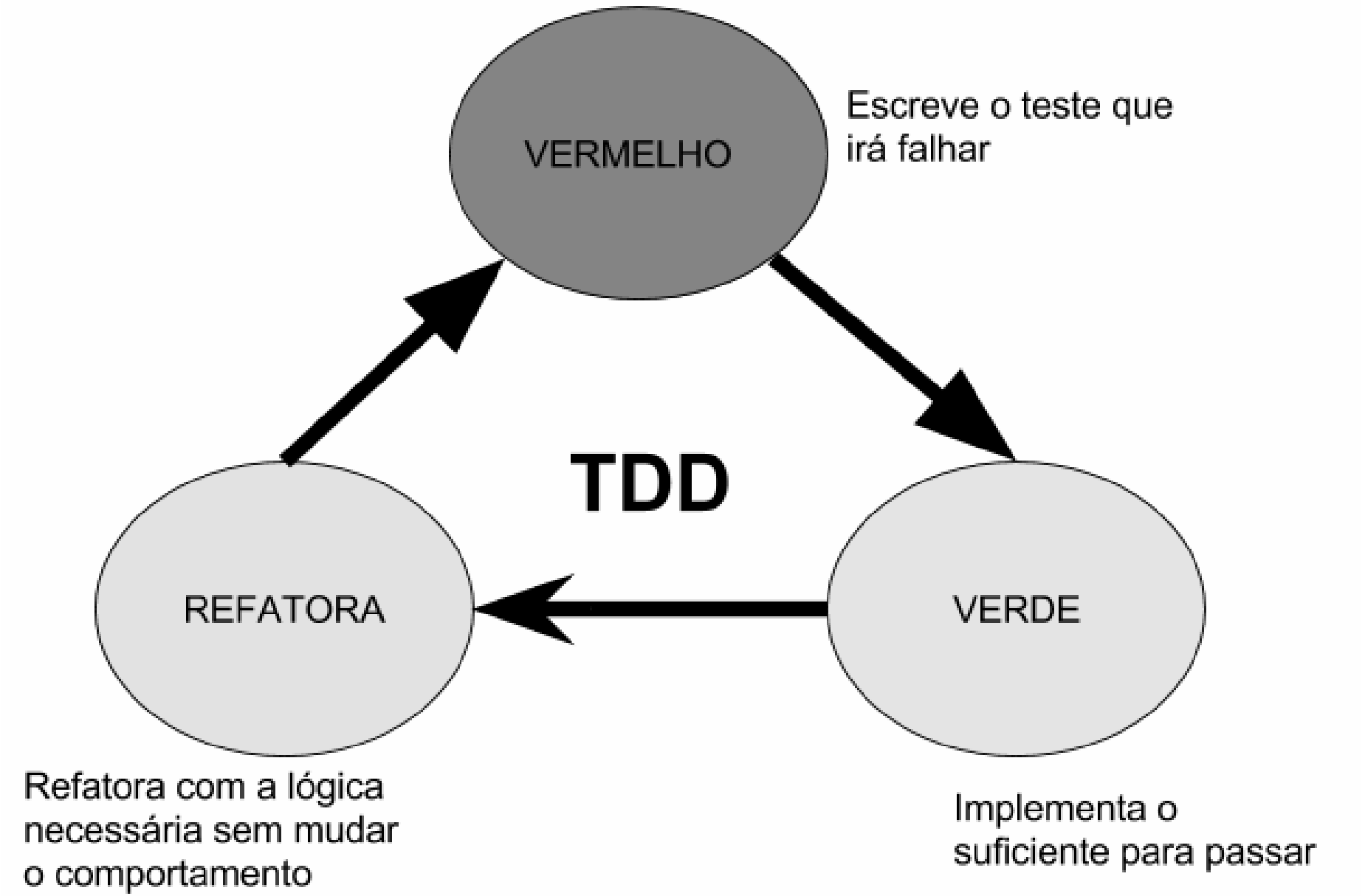
Será criado um suite testes unitários para realizar testes de regressão.

Algumas métricas da indústria

Table 3: Project A- Outcome Measures

Metric Description	Value
Actual defects/KLOC (using TDD)	X
Defects/KLOC of comparable team in org but not using TDD	2.6X
Increase in time taken to code the feature because of TDD (%) [Management estimates]	25-35%

Ciclo TDD


VERMELHO
Escreve o teste que irá falhar
TDD
REFATORA
VERDE
Refatora com a lógica necessária sem mudar o comportamento
Implementa o suficiente para passar

Quando usar TDD?

- Quando se precisa entender melhor o que deve ser implementado.

- funcionalidades críticas do sistema.


Mão na Massa

Mão na Massa 7

Vamos implementar a classe tdd.ContaCorrente a partir de uma suite de testes pre-existente.

Na prática no TDD o próprio desenvolvedor constrói os testes, esta suite foi definida para tornar a atividade mais rápida.

Resumo

- JUnit diferentes versões

- Mão na massa

- TDD