

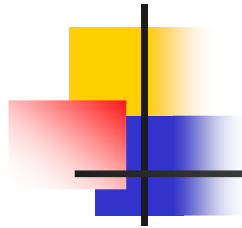
Padrões de projeto, testes automatizados e XML

02. Testes automatizados



Esp. Márcio Palheta

Gtalk: marcio.palheta@gmail.com



Testes sem framework

- As classes do capítulo anterior funcionam aparentemente bem;
- No entanto, vamos olhar com mais cuidado;
- Será que nosso código funciona quando passamos apenas um negócio?
- E o que acontece quando não há negócio a ser lançado?

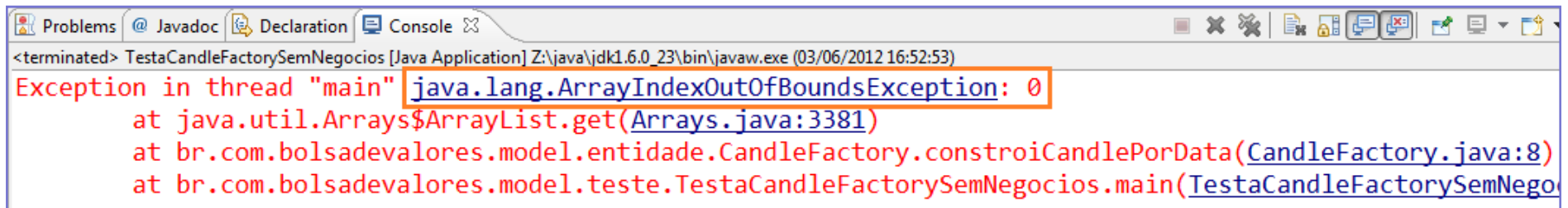
Exercício 01 – teste com apenas **um** negócio

```
package br.com.bolsadevalores.model.teste;
import java.util.Arrays;
public class TestaCandleFactoryComUmNegocio {
    public static void main(String[] args) {
        Calendar hoje = Calendar.getInstance();
        //Definicao de objetos de negocio
        Negocio negocio1 = new Negocio(40.5, 100, hoje);
        //Montagem da lista de negocios
        List<Negocio> lista = Arrays.asList(negocio1);
        //Definicao da fabrica de Candles
        CandleFactory fabrica = new CandleFactory();
        //Criacao do objeto Candle
        Candle candle = fabrica.constroiCandlePorData(hoje, lista);
        System.out.println(candle.getAbertura());
        System.out.println(candle.getFechamento());
        System.out.println(candle.getMinimo());
        System.out.println(candle.getMaximo());
        System.out.println(candle.getVolume());
    }
}
```

Exercício 02 – teste **sem** negócio lançado

```
package br.com.bolsadevalores.model.teste;
import java.util.Arrays;
public class TestaCandleFactorySemNegocios {
    public static void main(String[] args) {
        Calendar hoje = Calendar.getInstance();
        //Montagem da lista de negocios
        List<Negocio> lista = Arrays.asList();
        //Definicao da fabrica de Candles
        CandleFactory fabrica = new CandleFactory();
        //Criacao do objeto Candle
        Candle candle = fabrica.constroiCandlePorData(hoje, lista);
        System.out.println(candle.getAbertura());
        System.out.println(candle.getFechamento());
        System.out.println(candle.getMinimo());
        System.out.println(candle.getMaximo());
        System.out.println(candle.getVolume());
    }
}
```

Exercício 02 – teste **sem** negócio lançado: resultado



The screenshot shows an IDE console window with the following text:

```
<terminated> TestaCandleFactorySemNegocios [Java Application] Z:\java\jdk1.6.0_23\bin\javaw.exe (03/06/2012 16:52:53)  
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0  
    at java.util.Arrays$ArrayList.get(Arrays.java:3381)  
    at br.com.bolsadevalores.model.entidade.CandleFactory.constroiCandlePorData(CandleFactory.java:8)  
    at br.com.bolsadevalores.model.teste.TestaCandleFactorySemNegocios.main(TestaCandleFactorySemNegocios.java:1)
```

- **ArrayIndexOutOfBoundsException** – é uma boa exceção para indicar a falta de objetos da classe Negócio?
- Retornar:
 - Exception própria, Candle com valor especial ou, em último caso, **null**;

Exercício 03 – alterando a inicialização das variáveis

```
public class CandleFactory {  
    public Candle constroiCandlePorData(Calendar data, List<Negocio> negocios) {  
        //Definicao de valor muito pequeno  
        double maximo = Double.MIN_VALUE;  
        //Definicao de valor muito grande  
        double minimo = Double.MAX_VALUE;  
        //Verificando valor inicial  
        double abertura = negocios.isEmpty() ? 0:negocios.get(0).getPreco();  
        double fechamento =  
            negocios.isEmpty() ? 0:negocios.get(negocios.size() - 1).getPreco();  
        double volume = 0;  
        //Digite: foreach  
        for (Negocio negocio : negocios) {  
            volume += negocio.getVolume();  
            if (negocio.getPreco() > maximo) {  
                maximo = negocio.getPreco();  
            } else if (negocio.getPreco() < minimo) {  
                minimo = negocio.getPreco();  
            }  
        }  
        return new Candle(abertura, fechamento, minimo, maximo, volume, data);  
    }  
}
```

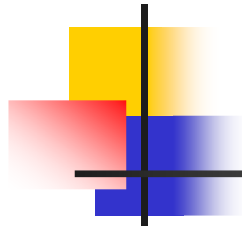
Exercício 04 – testando novamente

- Rode novamente as classe de teste
- TestaCandleFactoryComUmNegocio
 - Funciona normalmente;
- TestaCandleFactorySemNegocios
 - Parou de lançar a exceção;
 - Mas... Gerou um erro novo: Min e Max;
 - É comum o programador consertar uma coisa e quebrar outra?



Detalhando melhor o sistema

- Perguntas interessante:
- Uma negociação de petrobrás a 30 reais, com uma quantidade negativa de negócios é válido? E com número zero de negócios?
- Em outras palavras, posso dar new num Negocio com esses dados?
- ...



Detalhando melhor o sistema

- Uma negociação com data nula é válida? `new Negocio(10, 5, null)`?
- Um candle é realmente imutável? Não podemos mudar a data de um candle de maneira alguma?
- Quando preço de `abertura` é igual ao preço de `fechamento`, o candle é de alta ou de baixa? ...



Detalhando melhor o sistema

- Como geramos um candle de um dia que não houve negócios?
- E se a ordem dos negócios passadas ao **CandlestickFactory** não estiver na ordem crescente das datas? Devemos aceitar?
- E se esses Negócios forem de dias diferentes que a data passada como argumento para a factory?

TDD - Test Driven Development



- técnica que usa pequenas **iterações**;
- novos casos de testes são criados **antes** mesmo do início da **implementação**;
- O teste escrito deve **falhar**, pois **não** existe a funcionalidade implementada;
- Teste **antes** do código: não influenciado
- Menor acoplamentos com testes em classes pequenas;

TDD - Test Driven Development



- Nenhum código é escrito por **achamos** que vamos precisar dele;
- TDD é algo **difícil** de se implementar;
- mas depois que você constrói um sistema dessa maneira...
- O **hábito** é adquirido e as vantagens dessa técnica são reconhecidas;



Testes unitários

- Testam apenas uma classe ou método;
- Verificam se o comportamento está de acordo com o desejado;
- Envolver o **mínimo possível** de classes ou dependências;
- **Unidade** é a menor parte testável de uma aplicação. Em java, é o **método**;



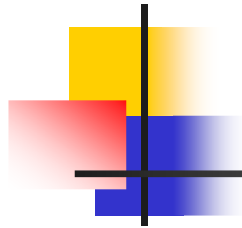
Testes unitários

- Nos testes, simulamos a execução de métodos da classe a ser testada;
- Passamos os **parâmetros** ao método testado; e
- Definimos o **resultado** que esperamos.
- Se o resultado for igual ao esperado, o teste passa. Do contrário, falha.



Testes unitários vs Testes de Integração

- Testes unitários são simples e pequenos, testando apenas métodos;
- Testes de integração são responsáveis pelo teste do sistema como um todo, não apenas das unidades;

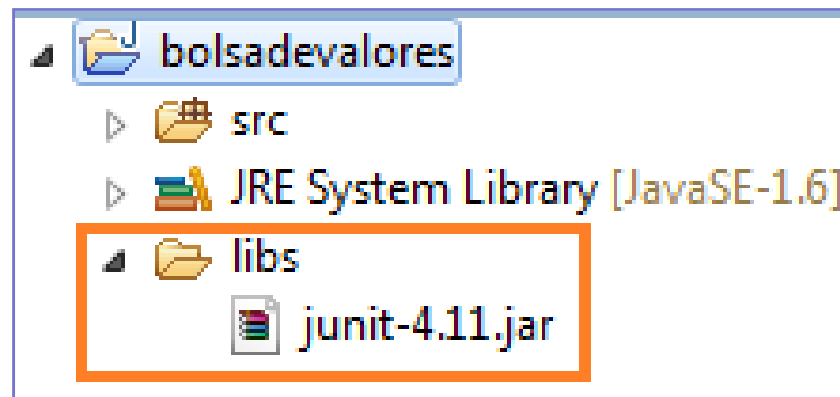


O framework de testes - JUnit

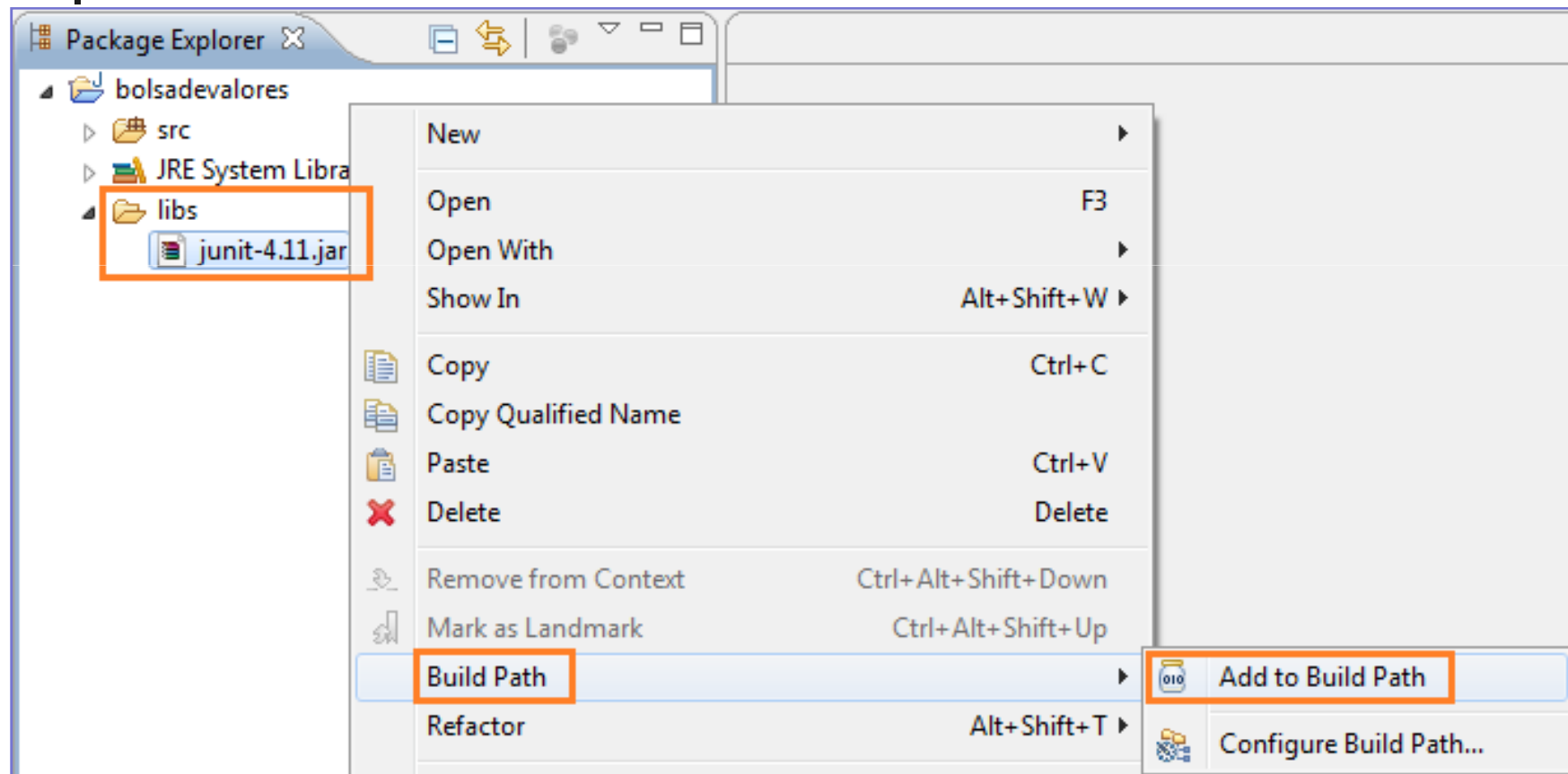
- O framework **JUnit** (junit.org) - facilita a criação de testes unitários;
- Uma **asserção** é uma afirmação que avaliamos em um ponto do código;
- Garante que novo **bugs** não são introduzidos ao longo do código;

Configuração do JUnit

- Baixe o biblioteca do JUnit:
 - <http://junit.org/>
- No projeto **bolsadevalores**, crie a pasta **libs** e copie o arquivo baixado:



Configuração do classpath





Trabalhando com Annotations

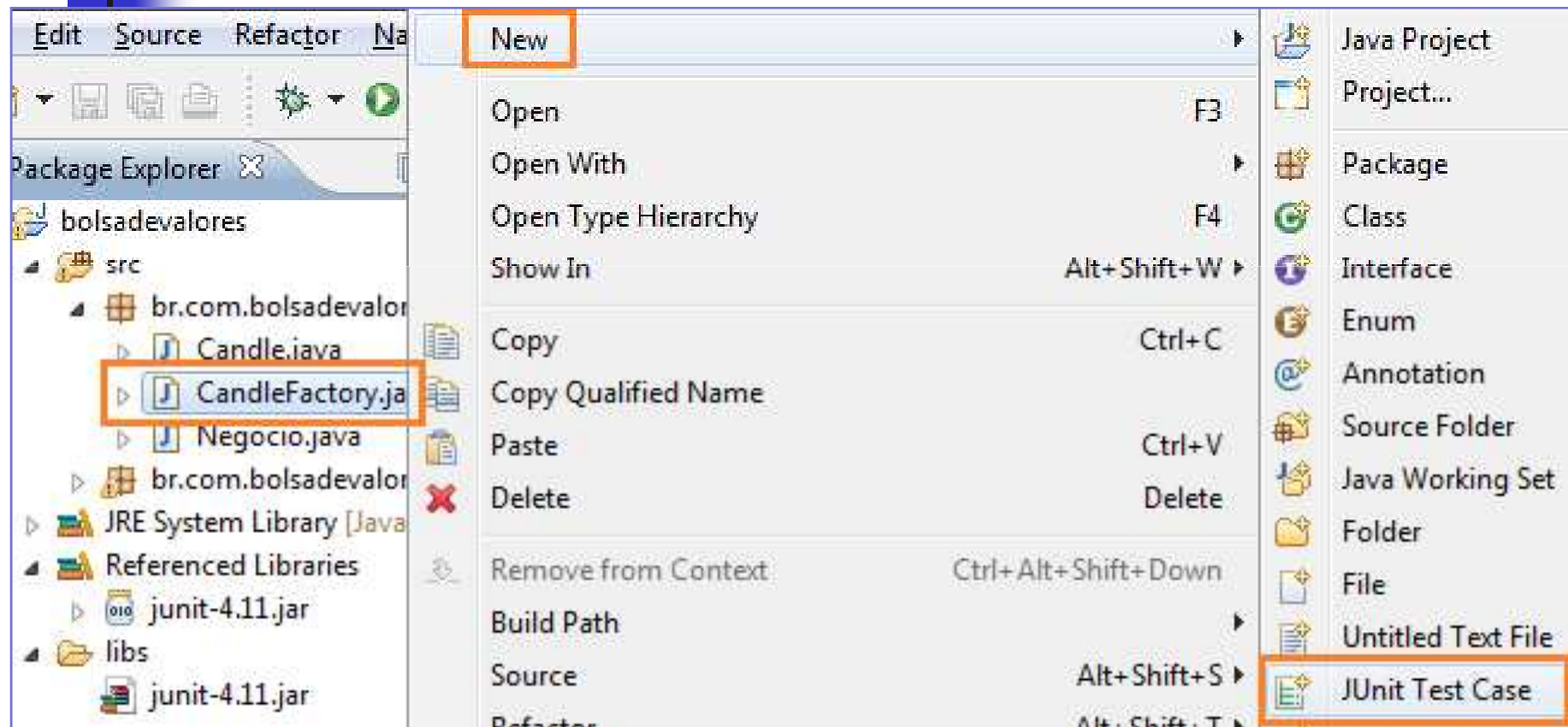
- **Annotation** é a maneira de descrever metadados – desde Java 1.5;
- É usada em **frameworks** para indicar como uma classe deve ser processada;
- Algumas annotations apenas indicam algo ao compilador: **@Override**
- Anotações podem receber **parâmetros**

Aplicando JUnit 4 ao nosso projeto

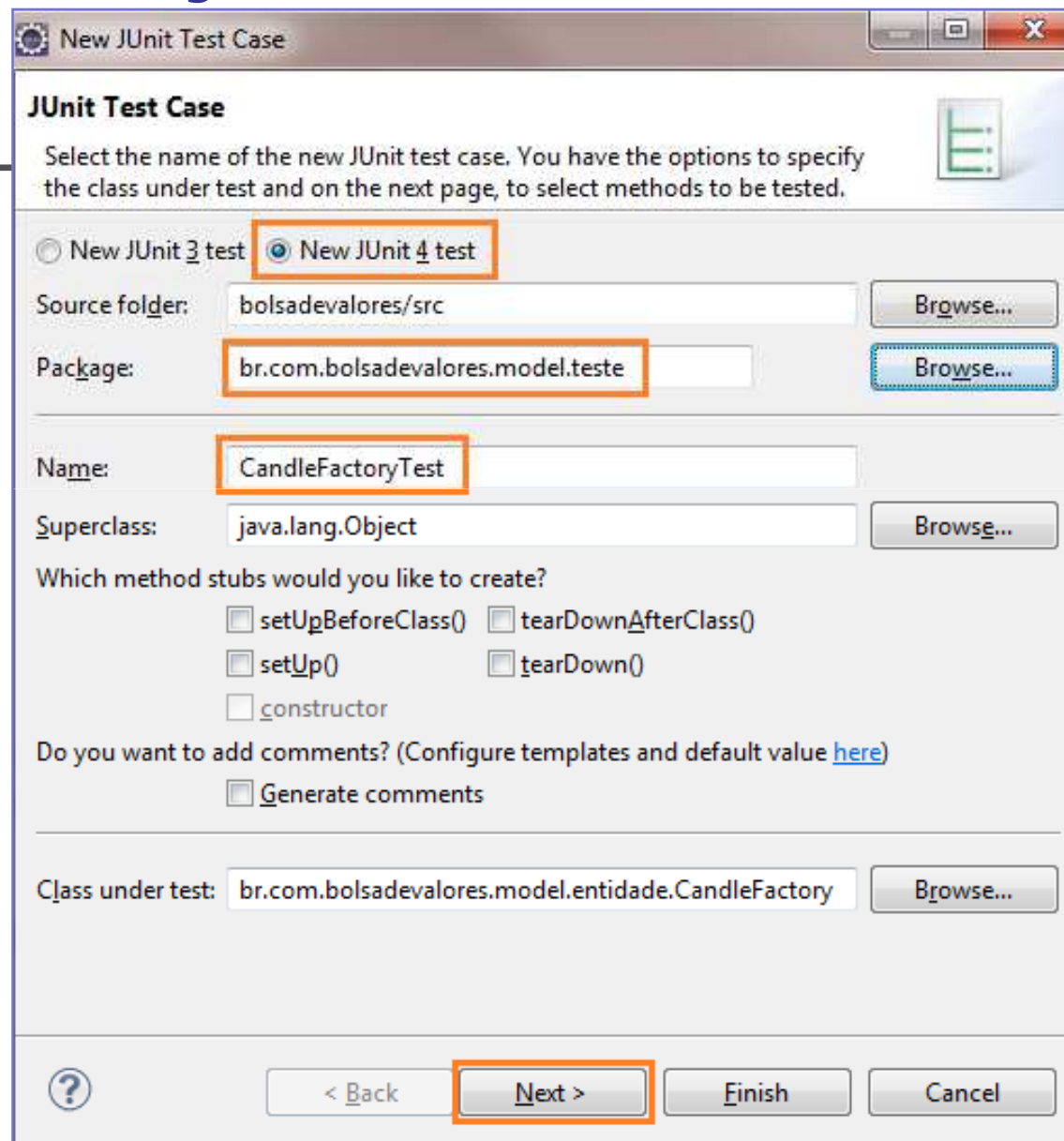


- A cada classe do nosso projeto, teremos uma classe para teste dos seus métodos - sufixo **Test**;
- Vamos criar a classe **CandleFactoryTest** para testar a classe **CandleFactory**.
- Clique com o botão direito na classe **CandleFactory** e escolha: **JUnitTestCase**

Criação da JUnit Test Case



Definição do JUnit Test Case



New JUnit Test Case

Select the name of the new JUnit test case. You have the options to specify the class under test and on the next page, to select methods to be tested.

☐ New JUnit 3 test ☒ New JUnit 4 test

Source folder:

Package:

Name:

Superclass:

Which method stubs would you like to create?

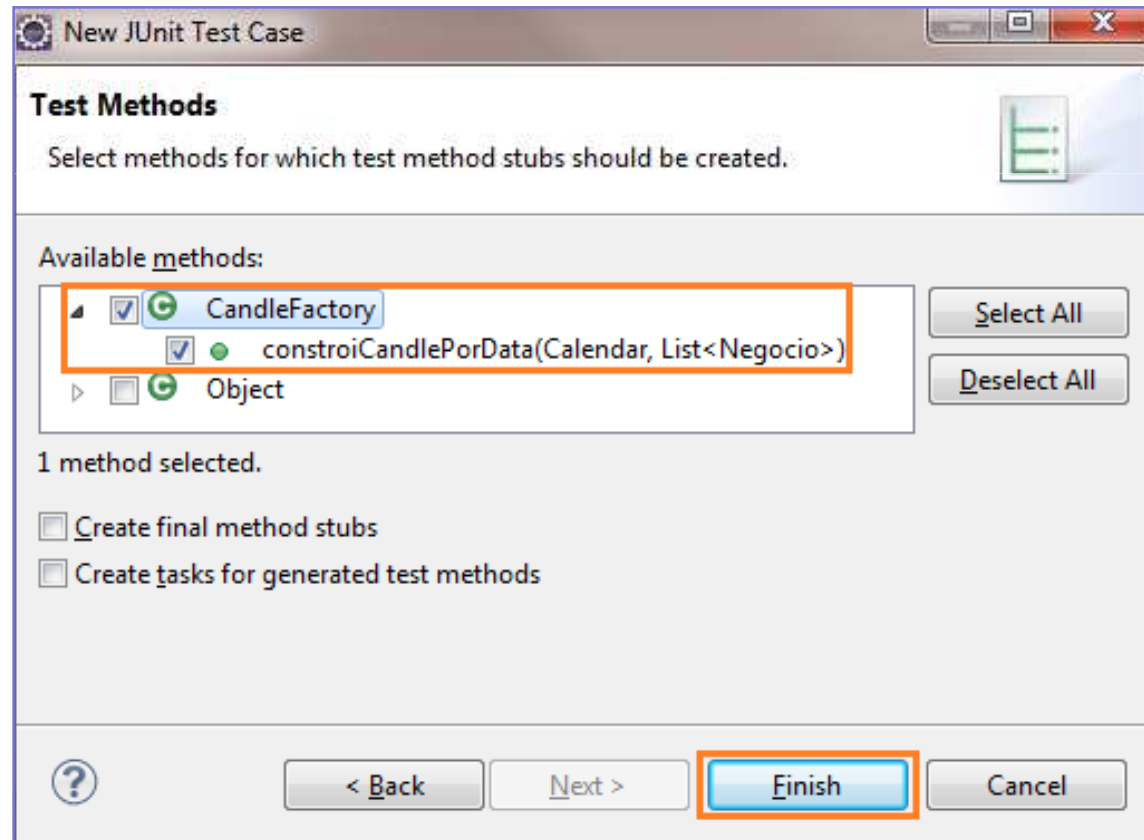
☐ setUpBeforeClass() ☐ tearDownAfterClass()
☐ setUp() ☐ tearDown()
☐ constructor

Do you want to add comments? (Configure templates and default value [here](#))
☐ Generate comments

Class under test:

Selecionando métodos a testar

- Selecione os métodos que deseja testar





Código gerado

- A classe **CandleFactoryTest** foi gerada;
- O método **testConstroiCandlePorData**

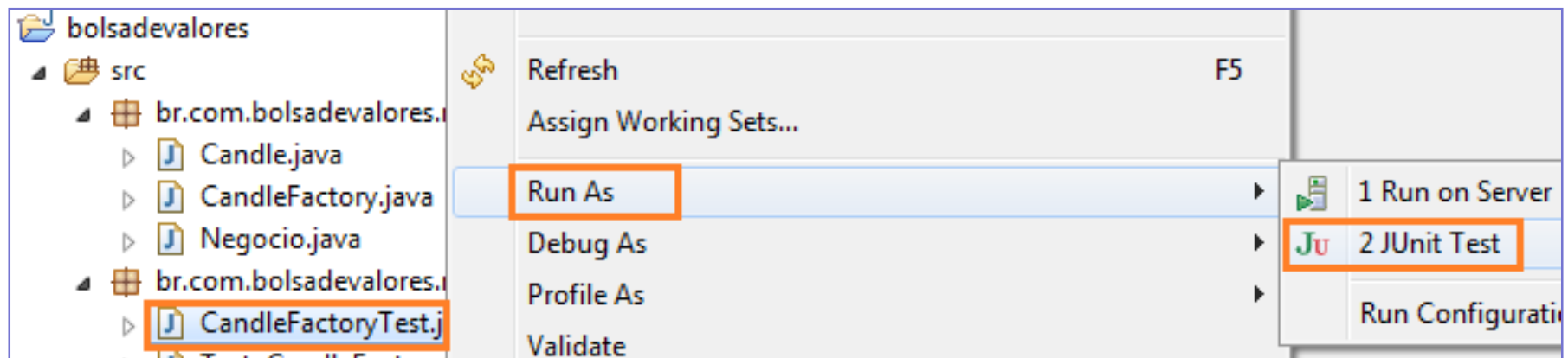
```
CandleFactoryTest.java ✕  
  
package br.com.bolsadevalores.model.teste;  
import static org.junit.Assert.*;  
import org.junit.Test;  
public class CandleFactoryTest {  
    @Test  
    public void testConstroiCandlePorData() {  
        fail("Not yet implemented");  
    }  
}
```


Atualizando o método de teste

```
package br.com.bolsadevalores.model.teste;
import java.util.Arrays;
public class CandleFactoryTest {
    @Test
    public void testConstroiCandlePorData() {
        Calendar hoje = Calendar.getInstance();
        Negocio negocio1 = new Negocio(40.5, 100, hoje);
        Negocio negocio2 = new Negocio(45.0, 100, hoje);
        Negocio negocio3 = new Negocio(39.8, 100, hoje);
        Negocio negocio4 = new Negocio(42.3, 100, hoje);
        List<Negocio> negocios = Arrays.asList(negocio1, negocio2,
            negocio3, negocio4);
        CandleFactory fabrica = new CandleFactory();
        Candle candle = fabrica.constroiCandlePorData(hoje, negocios);
        Assert.assertEquals(40.5, candle.getAbertura(), 0.00001);
        Assert.assertEquals(42.3, candle.getFechamento(), 0.00001);
        Assert.assertEquals(39.8, candle.getMinimo(), 0.00001);
        Assert.assertEquals(45.0, candle.getMaximo(), 0.00001);
        Assert.assertEquals(1676.0, candle.getVolume(), 0.00001);
    }
}
```

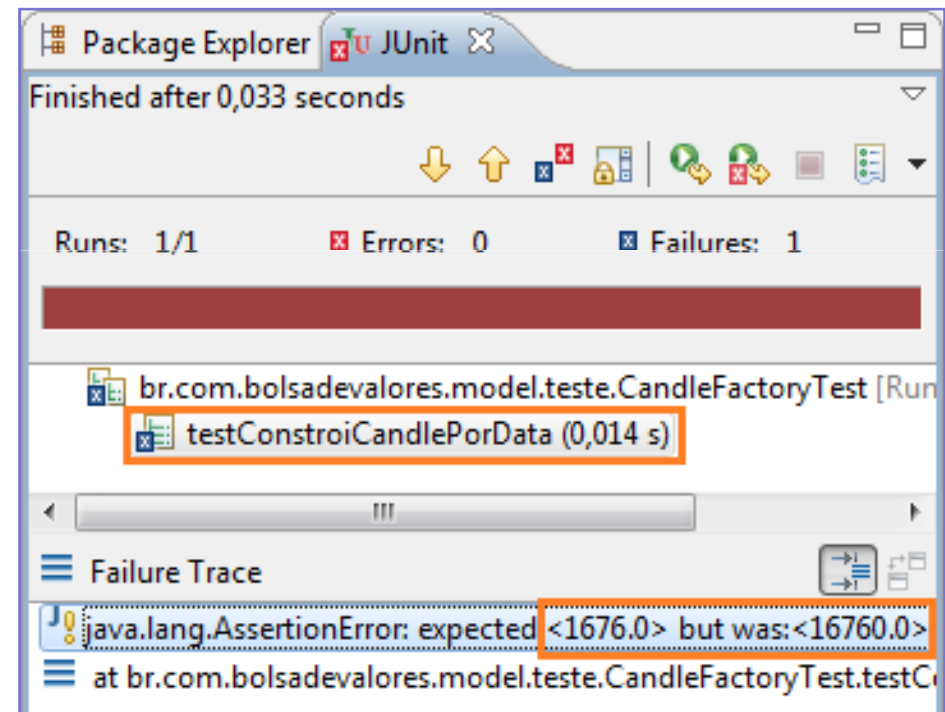
Execução do teste

- Clique com botão direito na classe;
- Selecione **Run as**
- Selecione **JUnit Test**



Ocorreu um problema

- O número esperado para o **volume** está errado no teste;
- O eclipse associa a falha à linha onde ocorreu a falha;

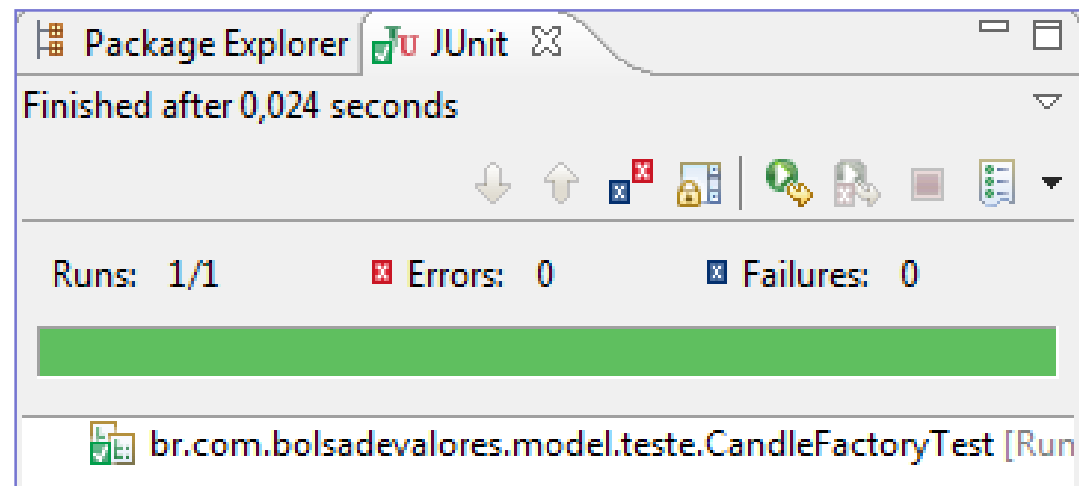


Atualizando a assertiva

- Altere o volume para 16760.0:

```
Assert.assertEquals(39.8, candle.getMinimo(), 0.00001);  
Assert.assertEquals(45.0, candle.getMaximo(), 0.00001);  
Assert.assertEquals(16760.0, candle.getVolume(), 0.00001);
```

- Teste novamente:





Adicionando novo método

- Adicione um novo método à classe CandleFactoryTest - **sem negócios;**

```
public class CandleFactoryTest {  
    @Test  
    public void testSemNegocios() {  
        Calendar hoje = Calendar.getInstance();  
        List<Negocio> negocios = Arrays.asList();  
        CandleFactory fabrica = new CandleFactory();  
        Candle candle = fabrica.constroiCandlePorData(hoje, negocios);  
        Assert.assertEquals(0.0, candle.getVolume(), 0.00001);  
    }  
}
```

- Execute o teste novamente



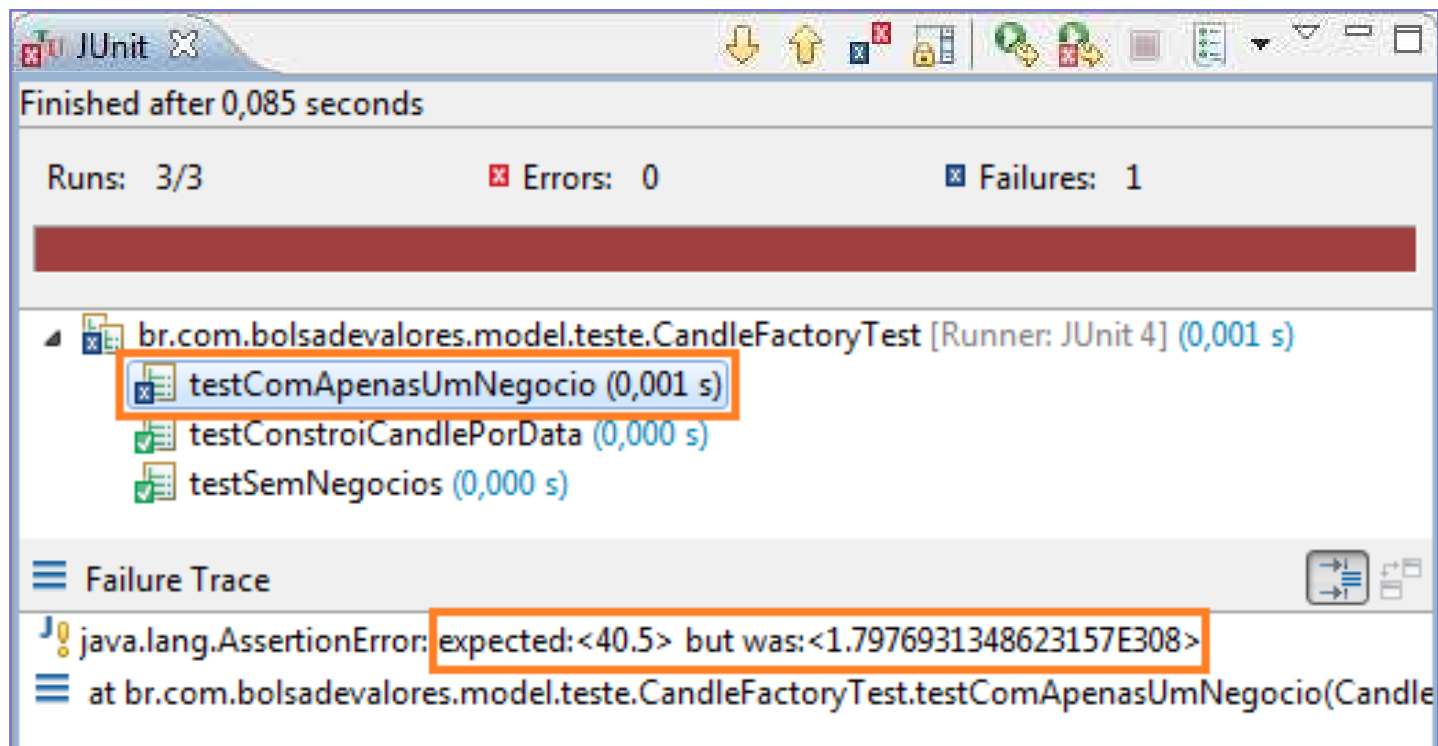
Novo método de teste

- Teste com apenas **um** negócio;

```
public class CandleFactoryTest {  
    @Test  
    public void testComApenasUmNegocio() {  
        Calendar hoje = Calendar.getInstance();  
        Negocio negocio1 = new Negocio(40.5, 100, hoje);  
        List<Negocio> negocios = Arrays.asList(negocio1);  
        CandleFactory fabrica = new CandleFactory();  
        Candle candle = fabrica.constroiCandlePorData(hoje, negocios);  
        Assert.assertEquals(40.5, candle.getAbertura(), 0.00001);  
        Assert.assertEquals(40.5, candle.getFechamento(), 0.00001);  
        Assert.assertEquals(40.5, candle.getMinimo(), 0.00001);  
        Assert.assertEquals(40.5, candle.getMaximo(), 0.00001);  
        Assert.assertEquals(4050.0, candle.getVolume(), 0.00001);  
    }  
}
```

Resultado dos testes

- Dois métodos funcionaram, mas um falhou. Qual foi o erro? Como **consertar**?





Novo caso de testes

- Vamos criar uma nova classe, clicando com o botão direito na classe **Negocio**
 - New / JUnit Test Case
- Vamos criar um método para testar se a data está realmente imutável;
- Implemente a classe a seguir;
- Rode o **teste** e verifique o **resultado**;



A classe **NegocioTest**

```
package br.com.bolsadevalores.model.teste;
import java.util.Calendar;
public class NegocioTest {
    @Test
    public void testDataDoNegocioEhImutavel() {
        Calendar c = Calendar.getInstance();
        c.set(Calendar.DAY_OF_MONTH, 15);
        Negocio n = new Negocio(10, 5, c);

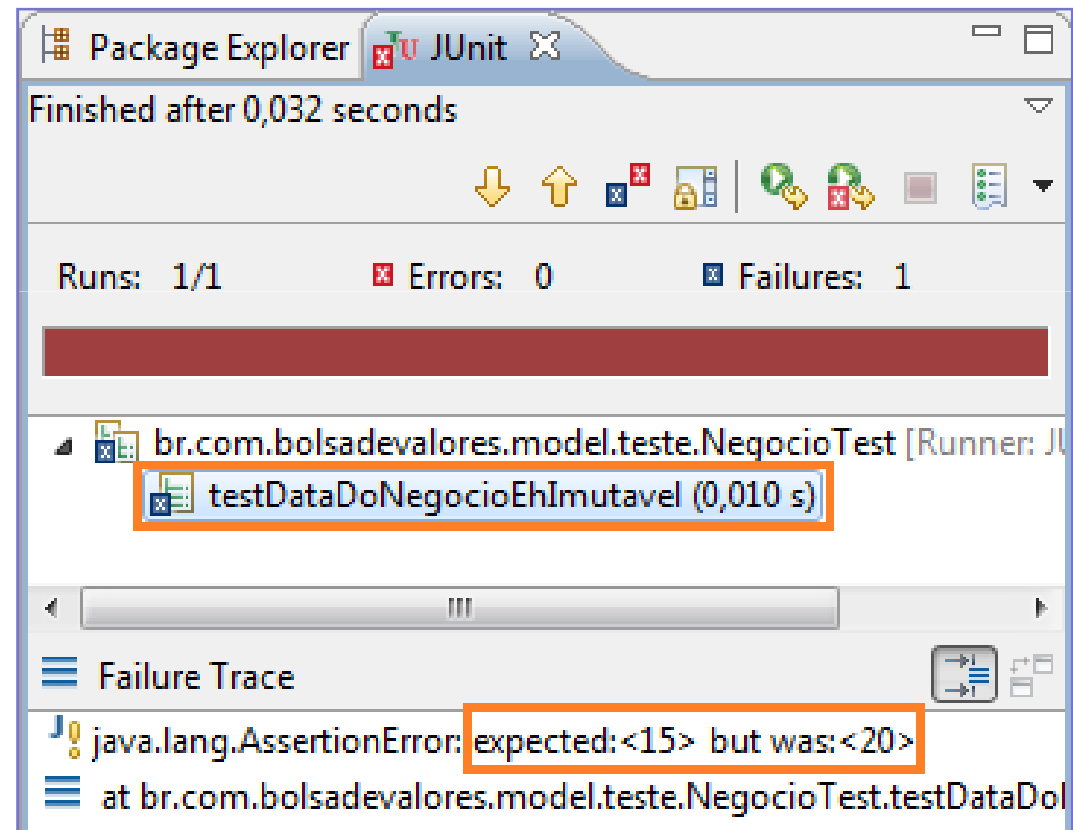
        // Podemos mudar a data?
        n.getData().set(Calendar.DAY_OF_MONTH, 20);

        //Verificar se o dia continua igual a 15
        Assert.assertEquals(15, n.getData().get(Calendar.DAY_OF_MONTH));
    }
}
```

Resultado do teste da classe

NegocioTest

- Por que o teste falhou?
- Que erro aconteceu?
- Como resolver?
- Precisamos rever o código da classe **Negocio**





Clonando objetos

- O método `Negocio.getData()` retorna uma referência ao atributo;
- Vamos devolver uma **cópia** do objeto;

```
package br.com.bolsadevalores.model.entidade;
import java.util.Calendar;
public final class Negocio {
    private final double preco;
    private final int quantidade;
    private final Calendar data;

    public Calendar getData() {
        return (Calendar) this.data.clone();
    }
}
```



Comentários

- Após as alterações, teste novamente a classe **NegocioTest**;
- O que mudou? Por que funcionou?
- E se não existisse a clonagem?

```
public Calendar getData() {  
    Calendar copia = Calendar.getInstance();  
    copia.setTimeInMillis(this.data.getTimeInMillis());  
    return copia;  
}
```



Testes com Exceções

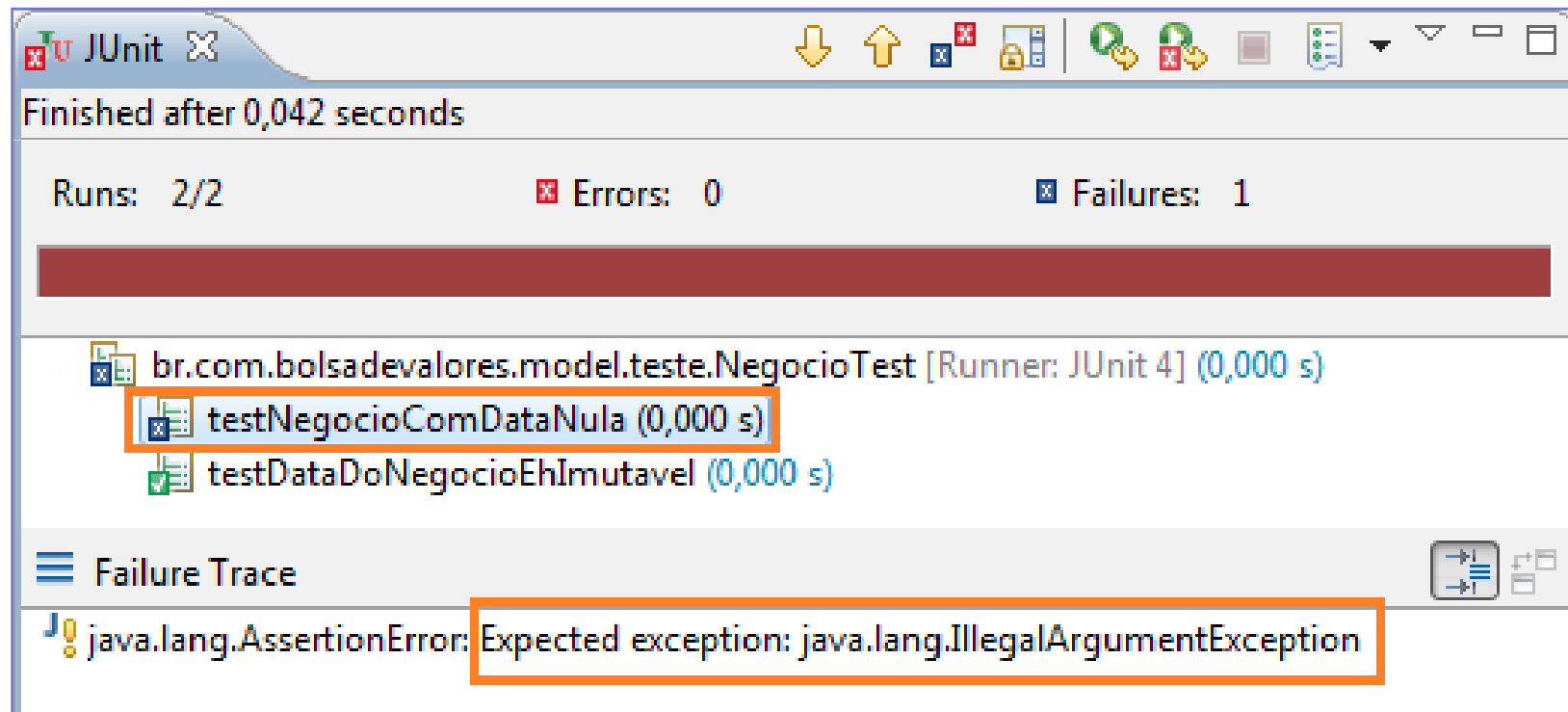
- Um **Negocio** pode ter data **nula** ? **NÃO**.
- Vamos criar um teste que espera que seja lançada uma exceção:

```
public class NegocioTest {  
    @Test(expected = IllegalArgumentException.class)  
    public void testNegocioComDataNula() {  
        new Negocio(10, 5, null);  
    }  
}
```

- Adotamos uma exceção padrão;
- Exceção personalizada;

Realizando o teste

- Execute a classe NegocioTest:





Atualizando o construtor da classe Negocio

- Atualize o código da classe Negocio
- Teste novamente. Qual o resultado?

```
public final class Negocio {  
    private final double preco;  
    private final int quantidade;  
    private final Calendar data;  
  
    public Negocio(double preco, int quantidade, Calendar data) {  
        if(data == null){  
            throw new IllegalArgumentException("Data inválida");  
        }  
        this.preco = preco;  
        this.quantidade = quantidade;  
        this.data = data;  
    }  
}
```



Exercício 01

- Uma Candle pode ter preço máximo menor que o preço mínimo?
- Crie um novo teste, o **CandleTest**.
- Crie `testPrecoMaximoNaoMenorQueMinimo` e faça um **new** com parâmetros inválidos.
- O teste espera **IllegalArgumentException**.
- **new** `Candle(10, 20, 20, 10, 10000, Calendar.getInstance());`



Exercício 02

- Uma Candle pode ter data nula? Pode ter algum valor negativo?
- Teste, verifique o que está errado, altere código para que os testes passem! Pegue o ritmo, essa será sua rotina daqui para a frente.



Bibliografia

- Java - Como programar, de Harvey M. Deitel
- Use a cabeça! - Java, de Bert Bates e Kathy Sierra
- (Avançado) Effective Java Programming Language Guide, de Josh Bloch



Referências WEB

- **Site oficial:**

- SUN: www.java.sun.com

- **Fóruns e listas:**

- Javaranch: www.javaranch.com
- GUJ: www.guj.com.br

- **Apostilas:**

- Argonavis: www.argonavis.com.br
- Caelum: www.caelum.com.br

Padrões de projeto, testes automatizados e XML

02. Testes automatizados



Esp. Márcio Palheta

Gtalk: marcio.palheta@gmail.com