1 - Sobre o xUnit

xUnit é o nome genérico para qualquer estrutura de testes unitários automatizados. O teste unitário ou de unidade é um processo que consiste na verificação da menor unidade do projeto de software, que no caso do Java é um método.

Um teste consiste em enviar uma mensagem para a unidade a ser testada e verificar se tem o retorno previsto. No Java, o termo enviar uma mensagem significa chamar o método - passando os parâmetros - e verificar se a resposta confere com o previsto. Como exemplo, para testar o método em três situações, tem-se de chamar o método três vezes, em cada uma delas passando os parâmetros distintos.

O teste unitário é da responsabilidade do próprio programador durante a implementação, ou seja, após codificar uma classe, por exemplo, será executado o teste de unidade. Esta técnica permite que o teste encontre erros mais facilmente, visto que cada classe é testada após ser codificada, em vez de fazer o teste de todo a aplicação ou parte dela.

O teste de uma classe pode ser escrito antes ou após a codificação da classe.

O teste unitário é considerado o primeiro de uma cadeia de testes a qual um software pode ser submetido, pois a ideia é testar do menor módulo (classe) até o maior (todas as funcionalidades da aplicação).

Um bom teste é aquele que tem elevada probabilidade de revelar um erro, e um teste para ser bem sucedido é aquele que revele erros ainda não revelados. Os testes têm de ser sempre documentados e reproduzíveis. Além disso, deve apresentar características tais como:

- Ser operável;
- Observável;
- Controlável;
- Ter decomposição;
- Simplicidade;
- Estabilidade;
- Compreensão.

2 - Sobre o JUnit

O JUnit é um framework de código-aberto, criado por Erich Gamma e Kent Beck, com suporte à criação de testes automatizados na linguagem de programação Java.

Esse framework facilita a criação de código para a automação de testes. Com ele, podem ser verificados se cada método de uma classe funciona da forma esperada, exibindo possíveis erros ou falhas, podendo ser utilizado também em baterias de testes.

O JUnit permite a realização de testes de unidades, conhecidos como caixa branca, facilitando assim a correção de métodos e classes.

Algumas vantagens de se utilizar JUnit:

- Permite a criação rápida de código de teste enquanto possibilita um aumento na qualidade do sistema sendo desenvolvido e testado;
- Uma vez escritos, os testes são executados rapidamente sem que, para isso, seja interrompido o processo de desenvolvimento;
- JUnit checa os resultados dos testes e fornece uma resposta imediata;

- Pode-se criar uma hierarquia de testes que permitirá testar apenas uma parte do sistema ou todo ele;
- Escrever testes com JUnit permite que o programador perca menos tempo depurando seu código;
- JUnit é livre.

Como os testes são executados por métodos, então antes de codificar os testes é importante saber quais métodos serão testados e em quais condições, pois não existe a necessidade de testar todos os métodos da classe e em quaisquer condições. Considere os seguintes pontos antes de codificar os testes:

- A principal regra para saber o que testar é:
  - Imaginar as condições de contorno e testar nesses limites, por exemplo, um método que válida se a idade e "de menor". Bastaria testar os parâmetros 17, 18 e 19, ou seja, não precisa testar outros valores;
- Comece pelos mais simples e deixe os testes complexos para o final;
- Use apenas dados suficientes, ou seja, não teste 10 condições, se apenas 3 forem o suficiente;
- Não teste métodos triviais, tais como, get e set;
- No caso de um método set, só faça o teste caso exista validação de dados;

Uma vez definidos os testes, eles precisam ser codificados no IDE Eclipse. Os testes podem ser programados sem o uso do JUnit, mas o objetivo deste framework é facilitar a codificação e execução dos testes.

3 - Codificação de um teste no Eclipse

Crie um novo projeto no Eclipse acessando File -> New -> Project... e na sequência escolha Java Project. Nomeie o projeto de Aula1, pode finalizar sem adicionar a biblioteca do JUnit. A Figura 1 mostra a estrutura do projeto. Antes de prosseguir, apenas verifique se o JUnit está disponível na instalação do seu Eclipse acessando o menu Window -> Show View -> Other para acessar a tela da Figura 2 e, na sequência, selecione o item JUnit.

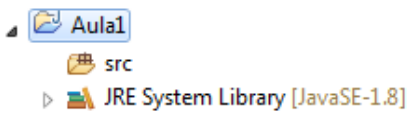

Figura 1 – Estrutura do projeto no Eclipse.

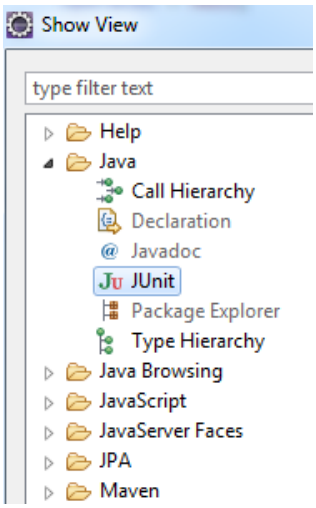

Figura 2 – Adicionar a biblioteca JUnit no projeto.

Antes de criar as classes do projeto Aula1, será criado o pacote aula (clique com o botão direito do mouse sobre src e acesse a opção New -> Package). Na sequência tem-se de adicionar a classe que contém o método main no pacote aula (clique com o botão direito do mouse sobre aula e acesse a opção New -> Class para acessar a tela da Figura 3). Na sequência crie uma

classe de nome Operacao no pacote aula, ao final o projeto terá a estrutura exibida na Figura 4. Programe o código da Figura 5 na classe Operacao.

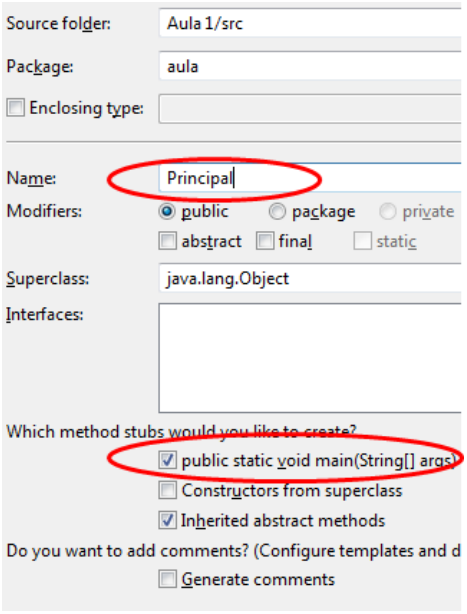

Figura 3 – Adicionar a classe Principal no projeto.

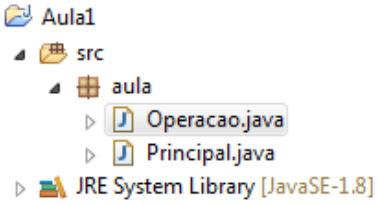

Figura 4 – Estrutura do projeto.

```
package aula;

public class Operacao {
    public double potencia( int a, int b ){
        return Math.pow(a, b);
    }

    public double divisao( int a, int b ){
        return a/b;
    }
}
```

Figura 5 – Código da classe Operacao.

Os testes do projeto não devem ficar dentro do folder src, pois desta forma esses códigos seriam incluídos no deploy (códigos finais a serem entregues para o cliente) do projeto. Então, usa-se criar um folder de nome test no projeto (clique com o botão direito do mouse sobre o nome do projeto Aula1 e acesse a opção New -> Source Folder), assim como na Figura 6. Na pasta test iremos criar a mesma estrutura da pasta src, isso não é uma obrigação, mas uma boa prática.

Na sequência crie um pacote de nome aula na pasta test, assim como na Figura 7.

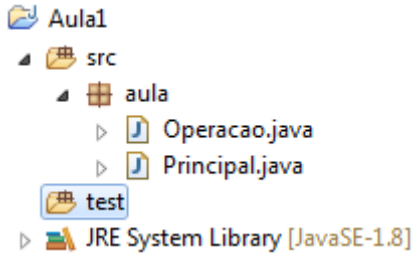

Figura 6 – Estrutura do projeto com a pasta de teste.

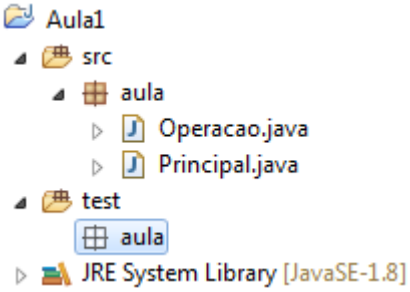

Figura 7 – Estrutura do projeto com o package de teste aula.

Para ter acesso a tela da Figura 8, clique sobre o pacote aula da pasta test e acesse o New -> Other... Na tela seguinte forneça o nome da classe que é OperacaoTest. Garanta que a opção escolhida seja New JUnit 4 test, para usar a versão 4 do framework. Após finalizar será exibida a janela da Figura 9 para adicionar a biblioteca no projeto. A Figura 10 mostra a estrutura do projeto.

Para codificar o teste utilize o código da Figura 11 na classe OperacaoTest. Para executar o teste acesse o menu Run -> Run as -> JUnit Test.

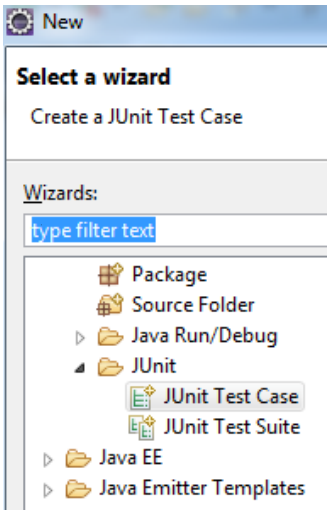

Figura 8 – Criação de uma classe de teste.

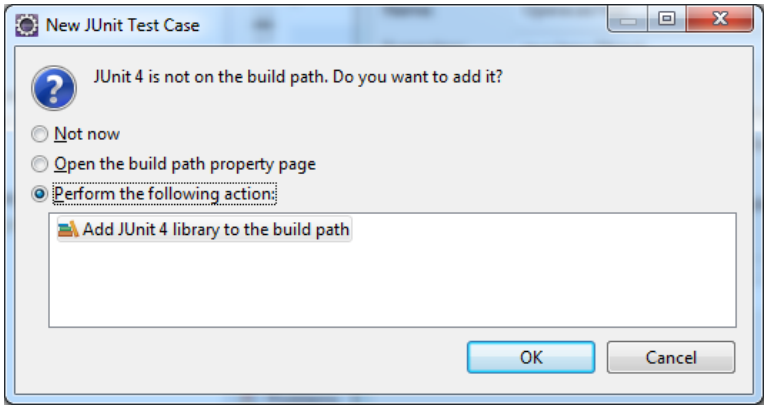

Figura 9 – Adicionar a biblioteca no projeto.

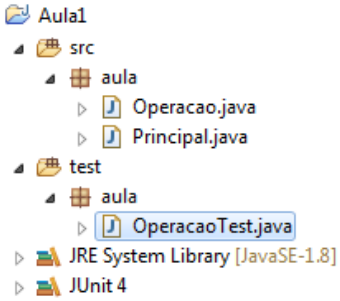

Figura 10 – Estrutura do projeto com a biblioteca JUnit.

```java
package aula;

import static org.junit.Assert.*;

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

public class OperacaoTest {
    private Operacao operacao;

    /** É chamado antes de cada método de teste */
    @Before
    public void setUp() {
        operacao = new Operacao();
    }

    /** É chamado após cada método de teste para limpar o lixo */
    @After
    public void tearDown() {
        operacao = null;
    }

      @Test
      public void testPotencia() {
            assertEquals("Teste 1", 8, operacao.potencia(2, 3),0.0);
      }
```

```
	@Test
	public void testDivisao() {
		/* Parâmetros: mensagem, valor esperado, operação, delta */
		assertEquals("Teste 1", 1.25, operacao.divisao(5, 4),0.5);
		assertEquals("Teste 2", 1.25, operacao.divisao(5, 4),0);
		assertEquals("Teste 3", 1.25, operacao.divisao(5, 0),0.5);
	}

	/** Testa se o método lança a exceção na operação */
	 @Test(expected=ArithmeticException.class)
	 public void testExceptionDivisao() {
		 assertEquals("Teste 1", 1.25, operacao.divisao(5, 0),0);
	 }

}
```

Figura 11 – Código da classe OperacaoTest.