

Ferramentas para programação em
linguagens orientadas a objetos.

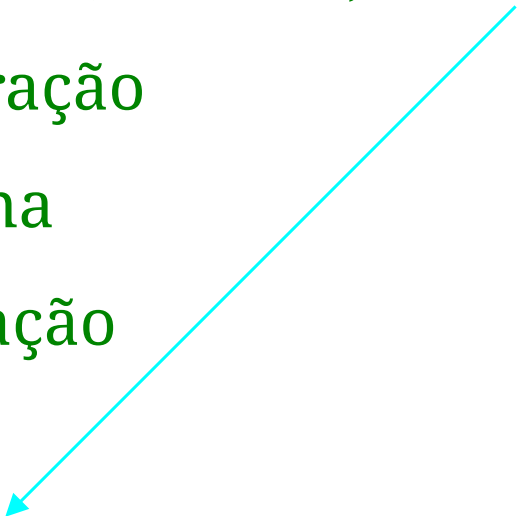
Programação Orientada a Objetos

Testes

- Essenciais para que o produto entregue tenha os
 - Requisitos de qualidade imposto pelo cliente
 - Cenários nos quais o sistema será implementado
- 3 conceitos básicos
 - **Defeito:** implementação incorreta feita em uma aplicação
 - **Erro:** manifestação desse defeito ou falha
 - **Falha:** comportamento externo que resulta em problemas de execução de um sistema

- Objetivo
 - Aplicação tenha seu número de defeitos minimizado e, assim, menor probabilidade de erros
 - Aumentar a confiabilidade do sistema e seu desenvolvimento
- Podemos executar diversas formas de testes para obter um cenário com poucos erros
- Documentação acurada
- Sistema de controle de versão de código

Testes unitários em Java

- Existem diversas categorias de testes
 - de módulo (unidade, unitário)
 - de integração
 - de sistema
 - de validação
- Unitário 
 - Verifica se os métodos estão funcionando
 - Ferramenta **JUnit** : usa um método contido em uma classe que é usado para teste

Teste unitário

- Valida o funcionamento de um método
- Detecta problemas
- Valida para cada elemento
 - Coesão
 - Uma classe deve ter apenas uma única responsabilidade
 - Acoplamento
 - Quanto uma classe depende da outra para funcionar
- Necessário que alguma ferramenta auxilie no processo
- O Eclipse possui integração para testes

Teste unitário

código da classe
Calculadora (sem os testes)

```
package U3S1;

public class Calculadora {
    public double somar(double a, double b)    {
        return a + b;
    }
    public double subtrair(double a, double b) {
        return a - b;
    }
    public double dividir(double a, double b) {
        return a / b;
    }
    public double multiplicar(double a , double
b) {
        return a * b;
    }
}
```

Quadro 3.2 | Classe exemplo para
a construção de testes unitários

```

1. package testes;
2. import static org.junit.jupiter.api.Assertions.*;
3. import org.junit.jupiter.api.Test;
4. class CalculadoraTeste {
5.     @Test
6.     void testSomar() {
7.         Calculadora c = new Calculadora();
8.         double res = c.somar(10, 50);
9.         assertEquals(60, res);
    }

```

configurado de forma automática quando se cria a classe pelo Eclipse

```

10. @Test
11. void testSubtrair() {
12.     Calculadora c = new Calculadora();
13.     double res = c.subtrair(10, -15);
14.     assertEquals(25, res);
    }

```

JUnit. verifica se o resultado não é correto, e registra

(continua)

```
15.      @Test
16.      void testDividir() {
17.          Calculadora c = new Calculadora();
18.          double res = c.dividir(10.0, 0);
19.          assertEquals(Double.isFinite(res),
false);
      }
      @Test
20.      void testMultiplicar() {
21.          Calculadora c = new Calculadora();
22.          double res = c.multiplicar(120.115,
12.5465);
23.          assertEquals(1507.01, 0.5, res);
24.      }
25. }
```

valor esperado, real, e margem de erro

Comentários do código anterior

- Classe `CalculadoraTeste`
 - 4 métodos
 - Cada um relacionado a um método da classe `Calculadora`
- Marcação (anotação) `@Test`
 - Referente ao `JUnit` (framework para testes)
 - Afirma que o método imediatamente abaixo é um teste
- São a primeira linha de defesa contra bugs
 - Garante que cada parte do sistema esteja funcionando
 - Modificações, acréscimos, etc.

Alguns tipos de testes (JUnit)

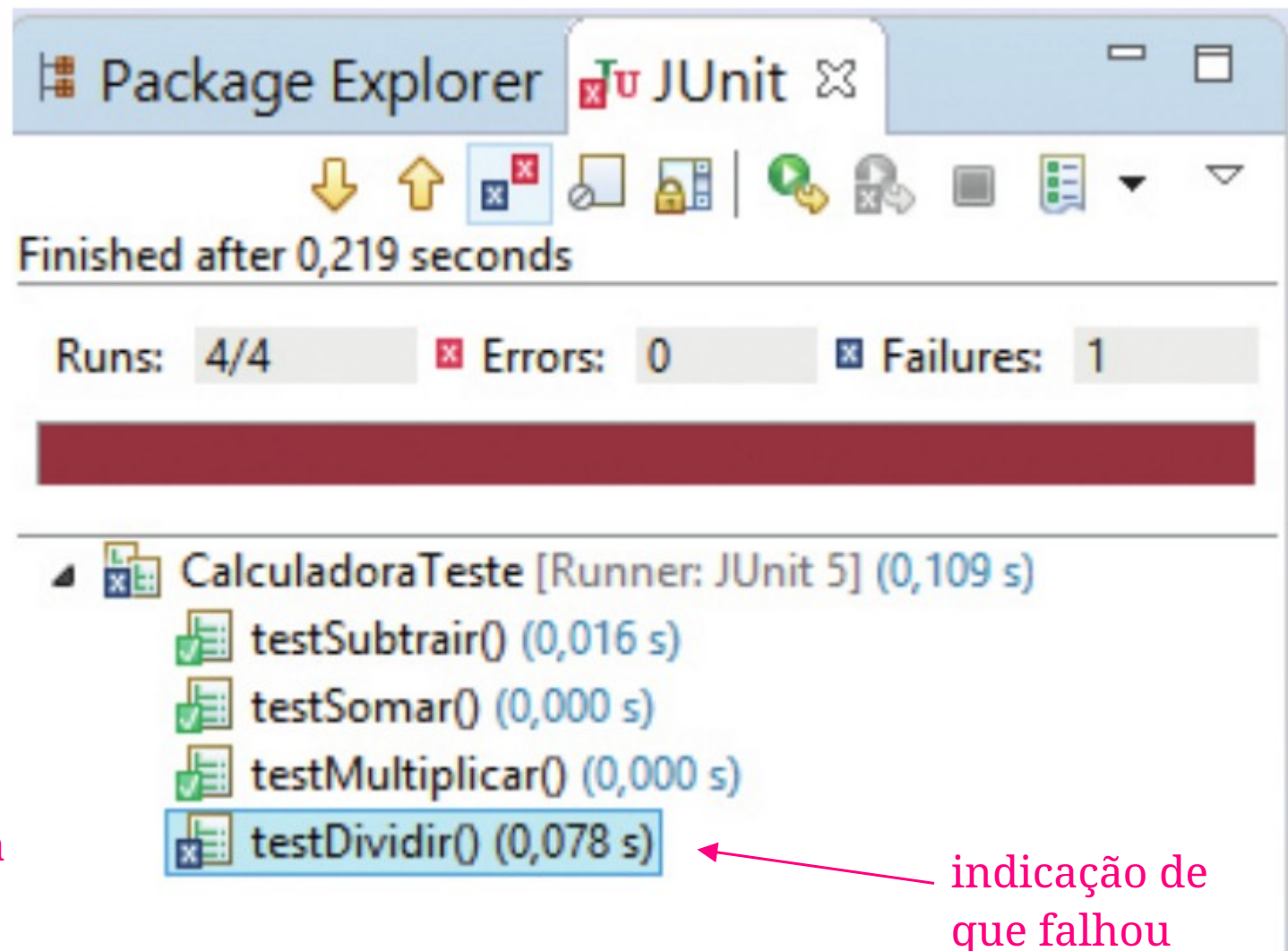
Quadro 3.4 | Alguns tipos de testes para a resposta dos sistemas

Tipo de assert:	Descrição:
<code>assertArrayEquals(float[] expected, float[] actual)</code>	Verifica se <code>float[] expected</code> é igual ao <code>float[] actual</code> , existem diversas sobrecargas para <code>double</code> , <code>int</code> e outros.
<code>assertNull(Object actual)</code>	Verifica se a instância passada é <code>null</code> .
<code>assertTrue(boolean condition)</code>	Verifica se a condição passa é <code>true</code> .

Eclipse

(precisa ser configurado antes)

Figura 3.1 | Resultado da execução do teste



clica com o botão direito do mouse na classe de teste, escolhe Run As, e o Eclipse detecta se a classe é um teste

indicação de que falhou (divisão por 0)

- Algumas ferramentas não permitem que o software seja liberado ou integrado a uma aplicação sem que antes todos os testes unitários tenham sucesso
- Uma outra forma, também, de garantir que defeitos não sejam gerados em um sistema é a utilização correta da documentação

Javadoc

- Utiliza **anotações** e **comentários** no decorrer do código
- Gera uma versão em **HTML** para explicar o funcionamento de cada classe, método e outros
- Explica a função de cada elemento do código
- **Anotações** estruturam partes específicas no código para a geração do **HTML**

Exemplos de marcação (anotação)

Quadro 3.5 | Exemplos de marcação do Javadoc

Marcação	Descrição
<code>@author</code>	Define o autor da classe.
<code>@see</code>	Classes correlatas à que está sendo desenvolvida.
<code>@since</code>	Versão da classe.
<code>@param</code>	Parâmetro utilizado no método.
<code>@return</code>	Descrição do retorno do método.
<code>@code</code>	Altera a fonte do texto para indicar que é um código Java.

Javadoc

```
/**
 * A classe calculadora é responsável por executar as
operações matemáticas básicas
 * @author Fabio Andrijauskas
 * @see java.lang.Math
 * @since 1.0
 */
public class Calculadora {
    /**
     * O método ({@code somar} faz a soma de dois nú-
meros
     *
     * @param a valor do primeiro número a ser soma-
do.
     * @param b valor do segundo número a ser soma-
do.
```

(continua)

```
* @return valor da soma de a e b
* @see Math
*/
public double somar(double a, double b) {
    return a + b;
}
/**
 * O método ({@code subtrair}) faz a subtração de
dois números
 *
 * @param a valor do primeiro número a ser sub-
traído.
 * @param b valor do segundo número a ser sub-
traído.
 * @return valor da subtração de a e b
 * @see Math
*/
```

(continua)


```
public double subtrair(double a, double b) {  
    return a - b;  
}  
/**
```

```
 * O método ({@code dividir}) faz a divisão de  
dois números  
 *  
 * @param a valor do primeiro número a ser divi-  
dido.  
 * @param b valor do segundo número a ser divi-  
dido.  
 * @return valor da divisão de a e b  
 * @see Math  
 */  
public double dividir(double a, double b) {  
    return a / b;  
}
```

(continua)

```
/**
 * O método ({@code multiplicar}) faz a multipli-
 * cação de dois números
 *
 * @param a valor do primeiro número a ser mul-
 * tiplicado.
 * @param b valor do segundo número a ser multi-
 * plicado.
 * @return valor da multiplicação de a e b
 * @see Math
 */
public double multiplicar(double a , double b) {
    return a * b;
}
```

- Eclipse permite gerar toda a documentação com o Javadoc utilizando a opção **Project > Generate Javadoc**
- Grande avanço na qualidade do desenvolvimento
 - Testes unitários
 - Mecanismo de documentação

[PACKAGE](#) **CLASS** [USE](#) [TREE](#) [DEPRECATED](#) [INDEX](#) [HELP](#)[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#) [ALL CLASSES](#)[SUMMARY: NESTED | FIELD | CONSTR | METHOD](#) [DETAIL: FIELD | CONSTR | METHOD](#)

Class Calculadora

java.lang.Object
Calculadora

```
public class Calculadora  
extends java.lang.Object
```

A classe calculadora é responsável por executar as operações matemáticas básicas

Since:

1.0

Author:

Fabio Andrijauskas

See Also:

Math

Constructor Summary

Constructors

Constructor and Description

calculadora()

Documentação
gerada por
Javadoc

Controle de Versão Git

- Controle de Versão Git+GitHub em qualquer projeto
<https://youtu.be/LOQT9bb0dG0>
- O que é controle de versão - Git e Github para Iniciantes
<https://youtu.be/8YsQ8AeKGvk>
- Git, SVN e CVS — comparação dos principais VCS
<https://blog.geekhunter.com.br/git-svn-e-cvs-comparacao-dos-principais-vcs/>

Sistemas de controle de versão

- Concurrent Versions System (CVS)
- Apache Subversion (SVN)
- Git distributed version control system

- O Github usa Git

- Simples, versátil

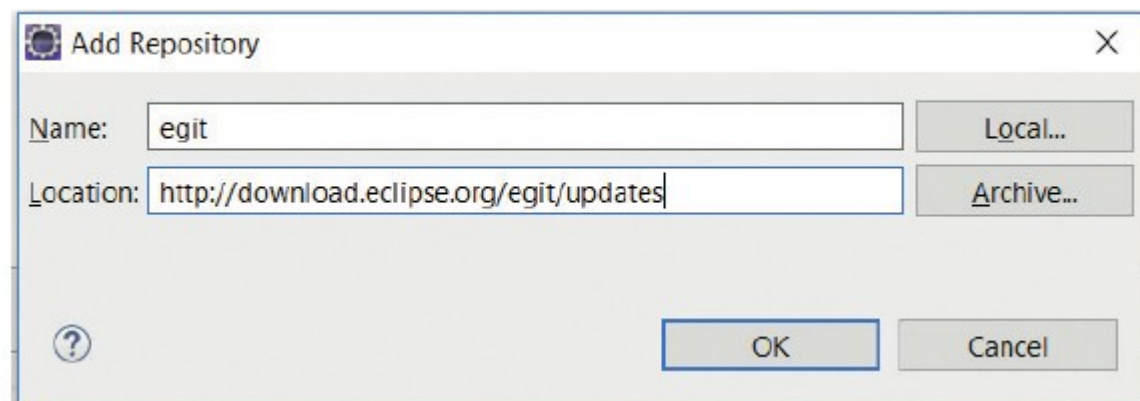
- Integra no Eclipse através de plugin:

- Help > Install new software

- Botão Add ao lado do Manage

- Link do plugin

Figura 3.3 | Inclusão de repositório do egit para o Eclipse



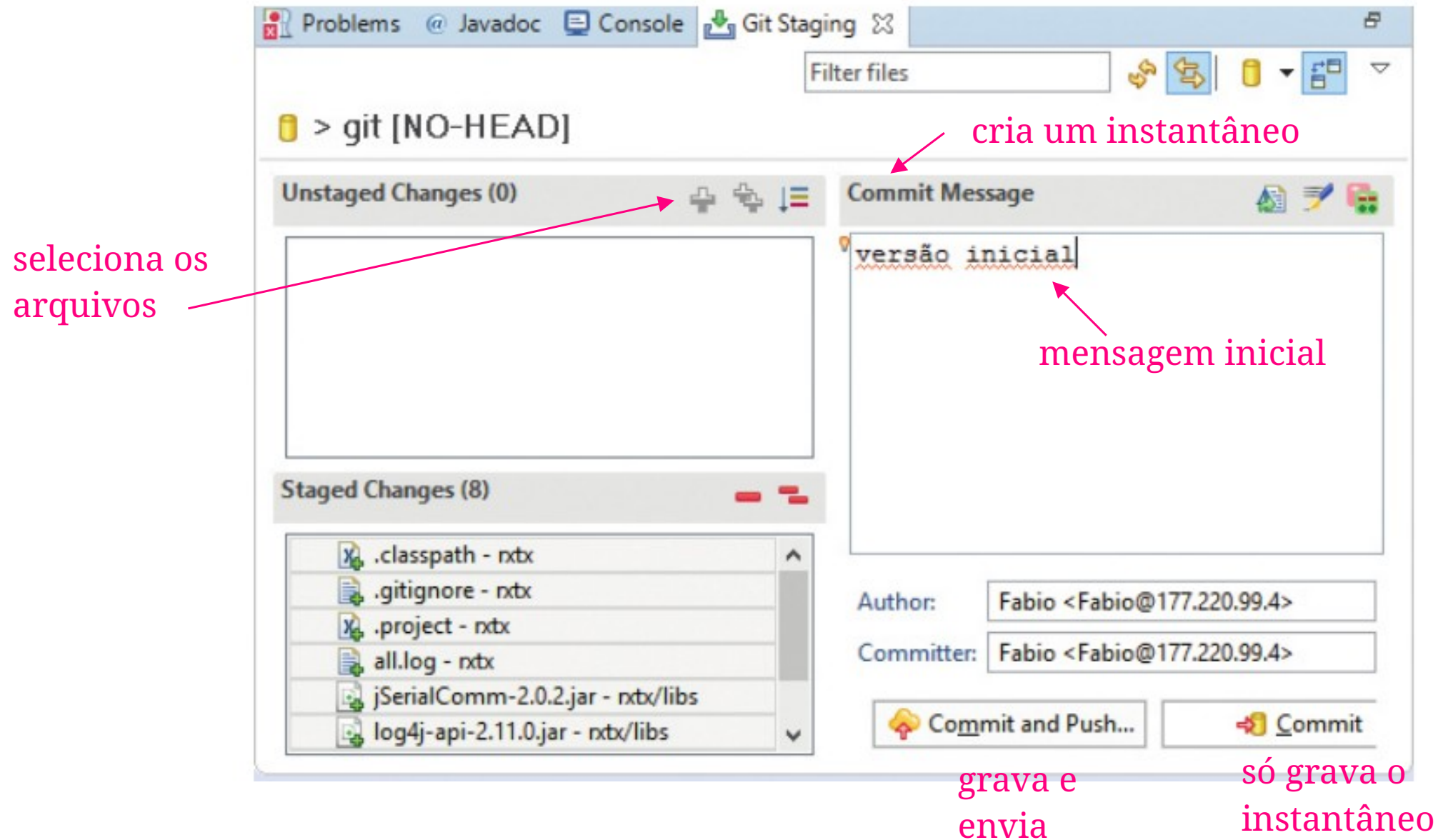
- Uso: selecionar **egit** em **Work with**
- Necessário criar um repositório git local
- Botão direito do mouse sob o projeto no **Package Explorer**, selecionar a opção **Team > Share Project**
 - **Create** para criar um repositório local
- Em seguida, é necessário fazer
 - **Commit** (gravação das modificações do repositório)
 - O primeiro push (envio dos arquivos)



cria um instantâneo

No Eclipse → Team > Commit

Figura 3.4 | Envio da primeira versão utilizando o *git*



Desenvolvimento com Git

- O Git precisa estar instalado na máquina
- Criar o projeto/repositório na máquina (um diretório)
 - Se estiver usando o Github precisa criar nele também
- Compartilhar (exemplo: um servidor de Git)
 - Git tem as opções offline/centralizado/distribuído
 - Pode usar serviços de terceiros como o Github/Gitlab
 - Existe a opção de clonar (baixar o repo completo)
- Enviar a primeira versão “*Commit and Push*” ← Update remote
- Desenvolver o código ← Record changes to the repository
- Enviar demais versões com “*Commit and Push*”
- Existem muitos outros recursos

Exemplo: usando o serviço Github

- <https://github.com>
- Cria a conta
- Cria o repo
- Com os dados, configura o Eclipse
- O Github tem vários outros recursos, além do Git

Existem diversas formas de utilizar os repositórios, e uma delas é da Figura 3.5, em que se cria um repositório no gitbub. Nessa interface é necessário inserir os dados da conta do github.

Figura 3.5 | Criação de repositório no github

The screenshot shows a dialog box titled "Push Branch master" with a standard Windows window frame. The main section is titled "Destination Git Repository" and includes the instruction "Enter the location of the destination repository." In the top right corner of this section is a yellow cloud icon with a red arrow pointing up to it. Below the instruction, there are several input fields and sections:

- Remote name:** A text box containing the word "origin". A red arrow points from this field down to the "URI:" field.
- Location:** A section containing:
 - URI:** A text box with the value "https://github.com/biozit/rtx.git". To its right is a button labeled "Local File..."
 - Host:** A text box with the value "github.com".
 - Repository path:** A text box with the value "/biozit/rtx.git".
- Connection:** A section containing:
 - Protocol:** A dropdown menu currently set to "https".
 - Port:** An empty text box.
- Authentication:** A section containing:
 - User:** A text box with the value "biozit".
 - Password:** A text box filled with ten black dots.
 - ☐ **Store in Secure Store**

At the bottom of the dialog box, there is a row of buttons: a help button (question mark icon), "< Back", "Next >" (highlighted with a blue border), "Finish", and "Cancel".

- Todas as modificações no código são feitas e salvas localmente
- Depois são enviadas clicando com o botão direito do mouse no projeto em [Team > Commit and push](#)

Exemplo 1

- Problemas de testes básicos em métodos
- Falta de documentação e entendimento das classes e métodos
- Problemas de sincronização de códigos quando alterados ou desenvolvidos pela equipe que aumentou
- Foi aplicada uma ferramenta de teste unitário

```
package U3S1;
```

```
public class ControleNotas {  
    private double N1;  
    private double N2;  
    private double N3;  
    private double mediaFinal;  
  
    public ControleNotas(double n1, double n2, double  
n3) {  
        N1 = n1;  
        N2 = n2;  
        N3 = n3;  
        mediaFinal = 0.0;  
    }  
    public double calculaNotaFinal()  
    {  
        mediaFinal = (N1*0.4) + (N2*0.3) + (N3*0.3);  
        return mediaFinal;  
    }  
}
```

Quadro 3.1 | Classe para apresentar as formas de controle de testes, versão e documentação

```
package U3S1;
```

```
import static org.junit.jupiter.api.Assertions.*;  
import org.junit.jupiter.api.Test;
```

```
class ControleNotasTest {
```

```
    @Test
```

```
        void testControleNotas() {  
            ControleNotas c = new ControleNotas(10.0,  
10.0, 10.0);  
            assertNotNull(c);  
        }
```

```
    @Test
```

```
        void testCalculaNotaFinal() {  
            ControleNotas c = new ControleNotas(10.0,  
10.0, 10.0);  
            double notafinal = c.calculaNotaFinal();  
            assertEquals(10.0, notafinal);  
        }  
    }
```

Quadro 3.7 | Testes unitários
para a classe ControleNotas

Quadro 3.8 | Classe de cálculo de nota documentada

Javadoc

```
package U3S1;

/**
 * A classe para calcular a nota final do aluno.
 * @author Fabio Andrijauskas
 * @see java.lang.Math
 * @since 1.0
 */

public class ControleNotas {
    /**
     * Nota do primeiro bimestre do aluno.
     */
    private double N1;
```

(continua)


```
private double N2;
```

```
/**
```

```
 * Nota da avaliação final semestral do aluno.
```

```
 */
```

```
private double N3;
```

```
/**
```

```
 * Media final do semestre
```

```
 */
```

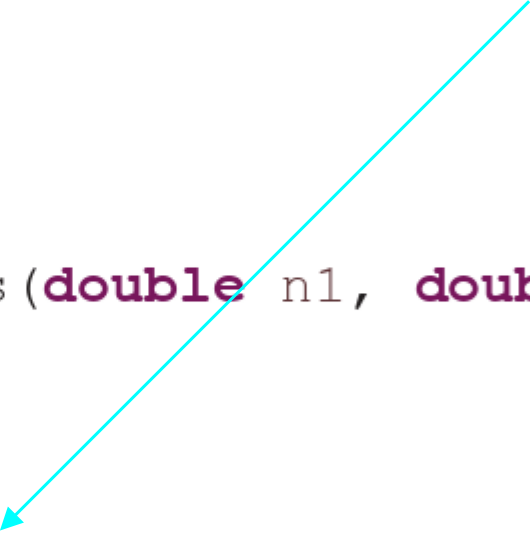
```
private double mediaFinal;
```

```
/**
```

```
 * O construtor da classe ({@code ControleNotas})  
para média final dos alunos
```

(continua)

```
*  
* @param n1 Nota do primeiro bimestre do aluno.  
* @param n2 Nota do segundo bimestre do aluno.  
* @param n3 Nota do final bimestre do aluno.  
* @see Math  
*/  
  
public ControleNotas(double n1, double n2, double  
n3) {  
    N1 = n1;  
    N2 = n2;  
    N3 = n3;  
    mediaFinal = 0.0;  
}
```



- Para fazer o controle de versão, é necessário:
 - Instalar o plugin [egit](#) no Eclipse
 - Compartilhar o código fonte mais avançado com a opção [Team > Share project](#)
 - Fazer o [Commit and Push](#) nos casos necessários

Exemplo 2

- Robustez de um sistema de controle de temperatura
- Guarda o valor de N temperaturas do ambiente
- Teste unitário para verificação do método
- Antes de o código entrar em produção, é possível verificar o seu funcionamento

Quadro 3.9 | Classe em que se deve implementar o teste unitário

```
package U3S1;

public class MediaTemperatura {

    private int qtdMax;
    private double[] temperaturas;
    private int atual;

    public MediaTemperatura(int parQtdMax)
    {
        qtdMax = parQtdMax;
        temperaturas = new double[qtdMax];
        atual = 0;
    }

    public void setLeitura(double parTemp)
    {
        temperaturas[atual] = parTemp;
        atual++;
    }
}
```

(continua)

continuação da classe em que se deve implementar o teste unitário

```
public double mediaTemp()  
{  
    double soma = 0.0;  
    for (int i = 0; i < atual; i++) {  
        soma = soma + temperaturas[i];  
    }  
    return soma/atual;  
}  
}
```

(teste unitário no próximo slide)

```
package U3S1;
```

Quadro 3.10 | Teste unitário para a média da temperatura

```
import static org.junit.jupiter.api.Assertions.*;  
import org.junit.jupiter.api.Test;
```

```
class MediaTemperaturaTest {
```

```
    @Test
```

```
    void testMediaTemp() {
```

```
        MediaTemperatura m = new MediaTemperatura(5);
```

```
        m.setLeitura(25.0);
```

```
        m.setLeitura(30.0);
```

```
        m.setLeitura(30.0);
```

```
        m.setLeitura(28.0);
```

```
        m.setLeitura(20.0);
```

```
        double res = m.mediaTemp();
```

```
        assertEquals(26.6, res, 0.1);
```

```
    }
```

```
}
```