



Integração Contínua – Prática – Etapa A

Prof. Jean Carlo Rossa Hauck
Prof. Osmar de Oliveira Braz Junior
Prof. Richard Henrique de Souza

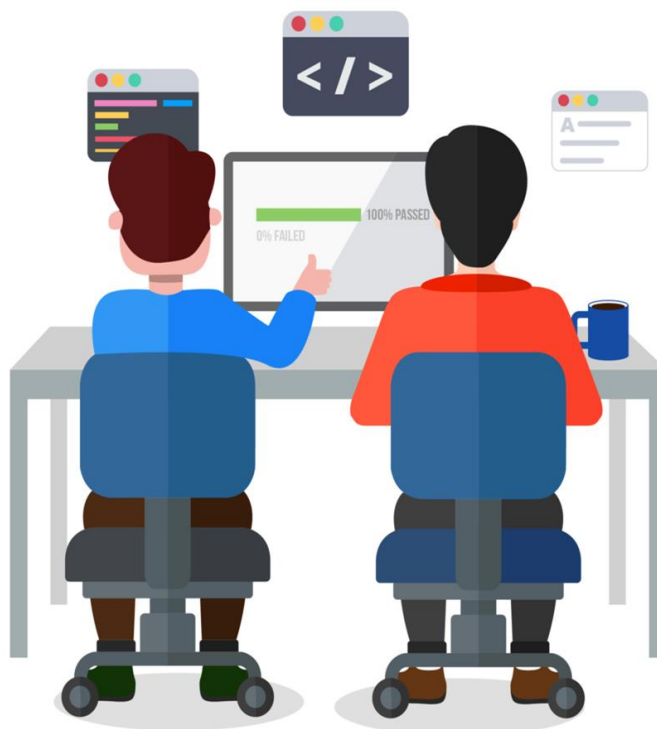
Objetivos

- Realizar um exemplo de **integração contínua** da **análise** até **cobertura** do código.
- A **integração** continua será realizada em 3 ambientes distintos com tarefas distintas.
- A **análise** irá considerar diversas métricas de qualidade de software como confiabilidade, manutibilidade, segurança, **cobertura** e duplicação de código.
- Nesta etapa iremos **criar** um projeto e os **testes** unitários.

Atividade em Grupo

Para esta atividade crie grupos de 2 alunos, para desenvolver a atividade segundo ***Pair Programming***.

Navegador



Piloto

Pair Programming

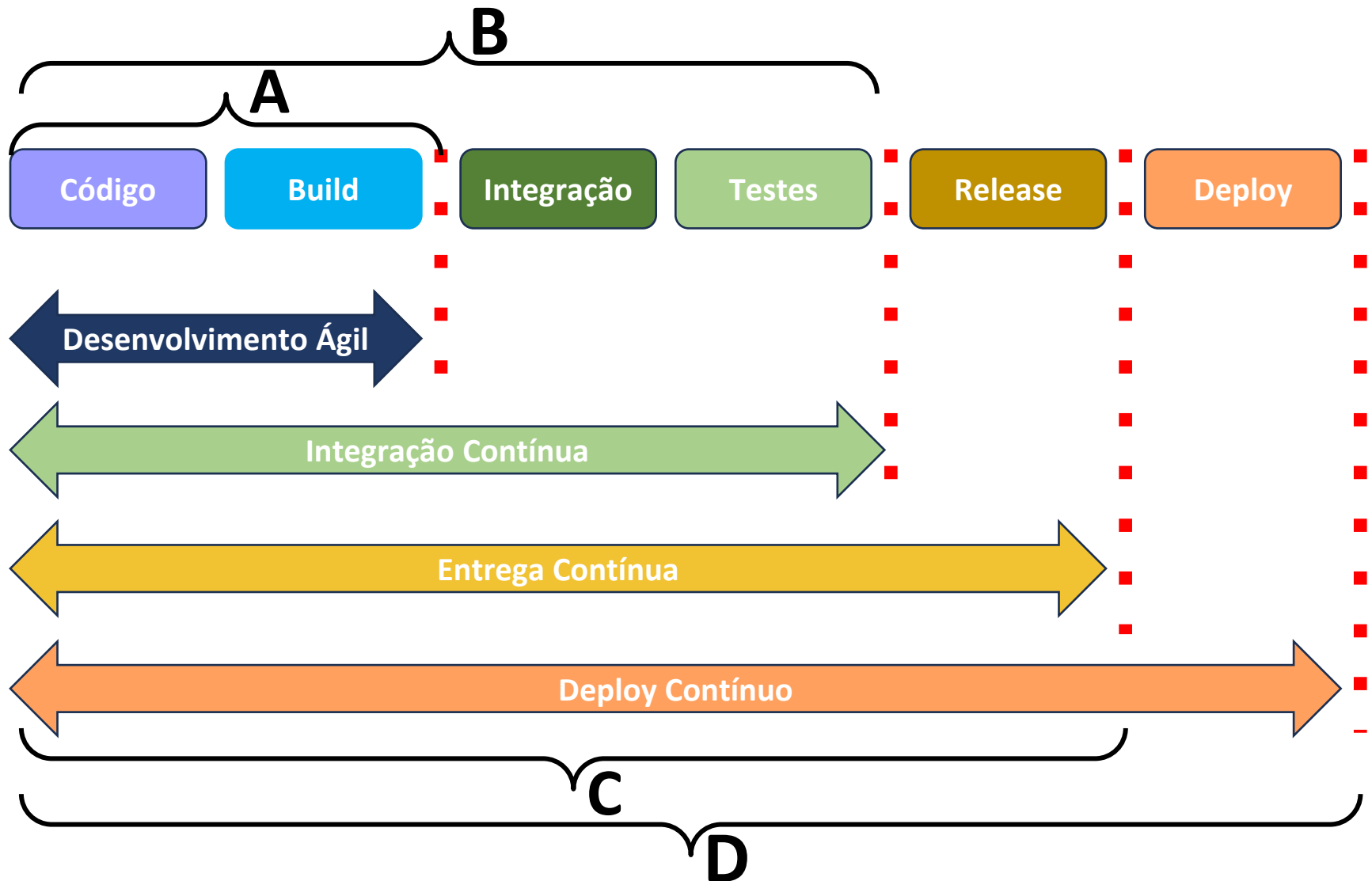
- Um é o **piloto**, responsável por escrever o código, o outro o navegador, acompanha a escrita de código e verificar se está de acordo com os **padrões do projeto** e de encontro à solução necessária.
- A intenção desta técnica é **evitar** erros de lógica, e ter um código mais confiável e melhor estruturado, utilizando-se para isso a máxima de que “**duas cabeças pensam melhor do que uma**”.

Integração, Entrega e Implantação Contínua

■ Abstração do Pipeline



Integração, Entrega e Implantação Contínua



Ferramentas utilizadas

- IDE com suporte
 - Apache Maven
 - JUnit 4
 - Github
- Github
 - Github Actions
 - Sonarcloud
 - JaCoCo

maven

JUnit



sonarcloud 

JACOCO
Java Code Coverage

Atividades práticas

- A - Criação de Projeto e testes unitários
 - Criar projeto na IDE
 - Automatizado com Apache Maven
 - Criar testes unitários com JUnit 4
 - Armazenar projeto no Github
- B - Integração Contínua
 - Github Actions
 - JUnit 4
- C - Análise do Código
 - Sonarcloud
 - Integração com Github Actions
- D - Cobertura do código
 - Jacoco
 - Integração com Github Actions
 - Integração com Maven e Sonarcloud

A - Criação do projeto e testes



- Criar projeto na **IDE**
- Automatizado com **Apache Maven**
- Desenvolver sistema
- Criar testes unitários com **JUnit 4**
- Armazenar projeto no **Github**



IDE

JUnit

Utilize uma IDE que possua suporte:



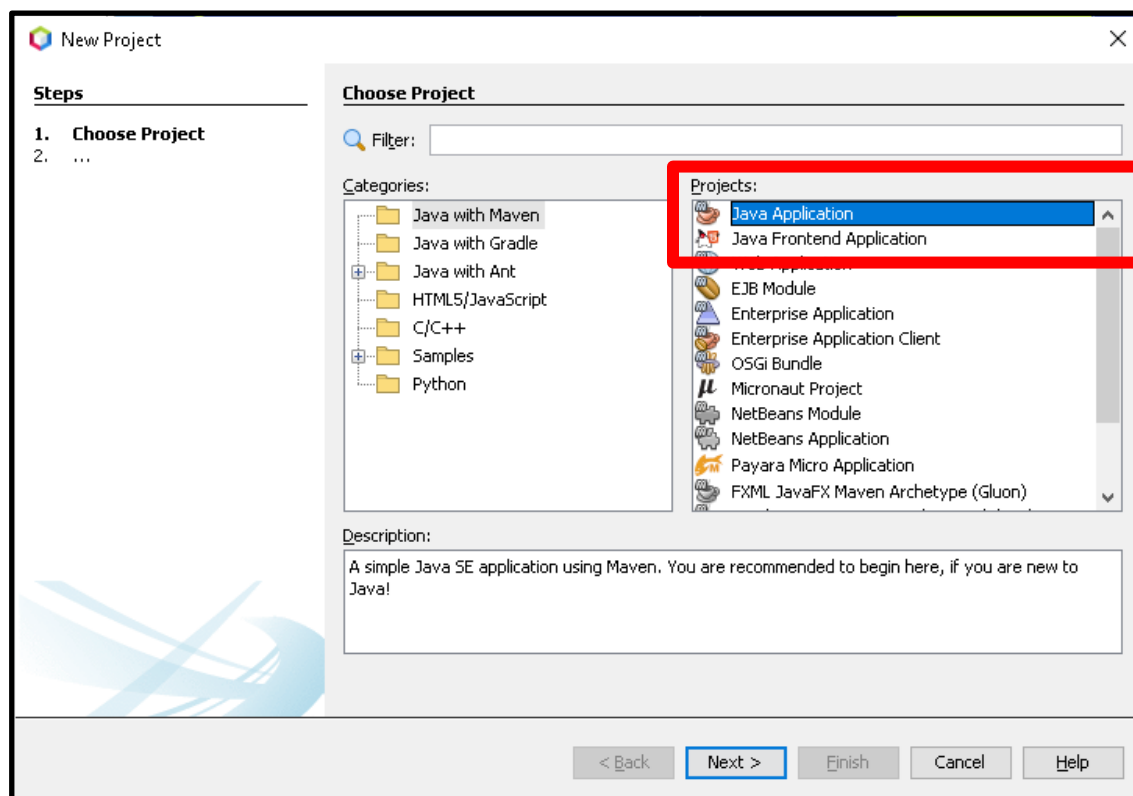
maven



Passo 1

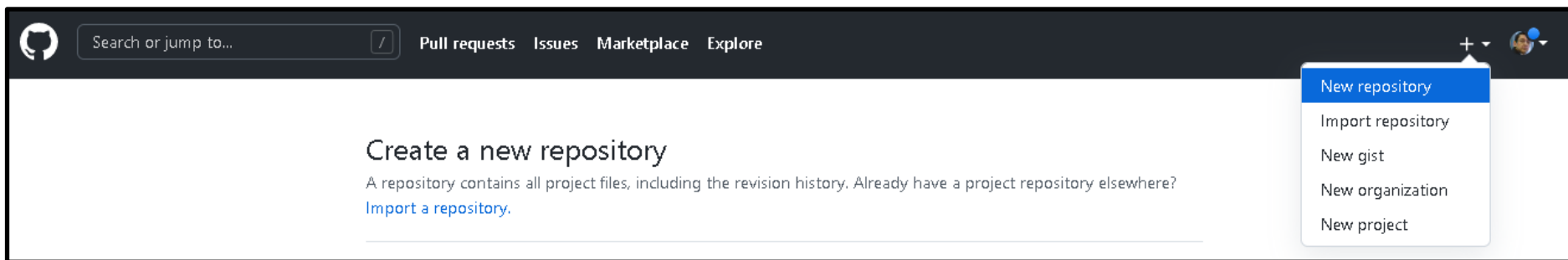
Crie um projeto em sua IDE segundo o padrão de automatização do Apache Maven.

Para criar o
arquivo
pom.xml



Passo 1 - Alternativa


Criar um repositório vazio no Github e clone em sua IDE.



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?
[Import a repository.](#)

Owner * **Repository name ***

 osmarbraz ▾ ✓


Great repository names are [Your new repository will be created as calculadora-](#) about [stunning-sniffle?](#)

Description (optional)

Choose a license

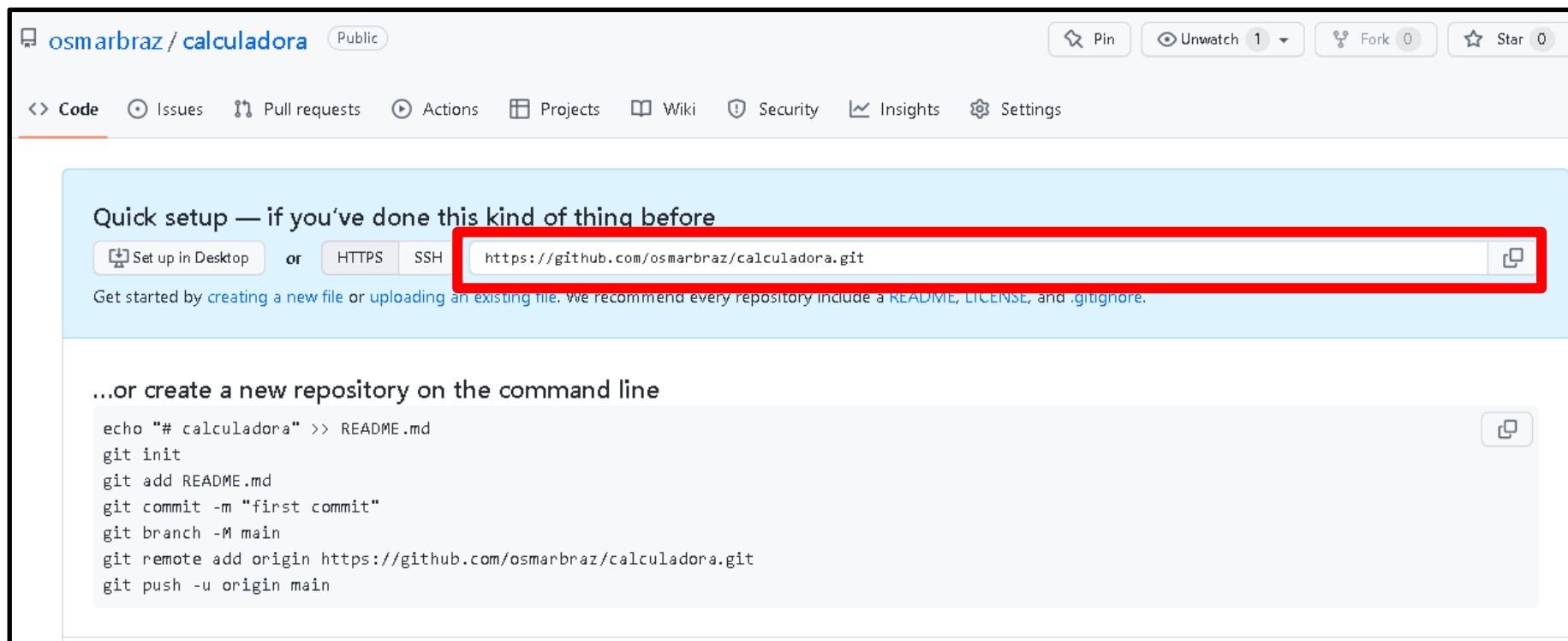
A license tells others what they can and can't do with your code. [Learn more.](#)

License: None ▾

 You are creating a public repository in your personal account.

Passo 1 - Alternativa

Criar um repositório vazio no Github e clone em sua IDE e selecione o tipo projeto.



The screenshot shows the GitHub interface for the repository 'osmarbraz/calculadora'. The repository is public. The 'Quick setup' section is highlighted with a red box, showing the repository URL: `https://github.com/osmarbraz/calculadora.git`. Below this, there is a section for creating a new repository on the command line, which includes a list of commands to initialize a repository, add a README, commit, create a branch, add a remote, and push.

osmarbraz / calculadora Public

Pin Unwatch 1 Fork 0 Star 0

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH `https://github.com/osmarbraz/calculadora.git`

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# calculadora" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/osmarbraz/calculadora.git
git push -u origin main
```

Passo 2

Crie o pacote calculadora e adicione a classe ao projeto.

Calculadora.java

```
package calculadora;

public class Calculadora {
    private double valorA;
    private double valorB;

    public Calculadora() {
        this(0.0, 0.0);
    }

    public Calculadora(double valorA, double valorB) {
        setValorA(valorA);
        setValorB(valorB);
    }

    public double getValorA() {return valorA; }
    public double getValorB() {return valorB; }
    public void setValorA(double valorA) {this.valorA = valorA; }
    public void setValorB(double valorB) { this.valorB = valorB; }
    public double getAdicao() {
        return (getValorA() + getValorB());
    }

    public double getSubtracao() {
        return (getValorA() - getValorB());
    }

    public double getProduto() {
        return (getValorA() * getValorB());
    }

    public double getDivisao() {
        return (getValorA() / getValorB());
    }
}
```

Passo 3

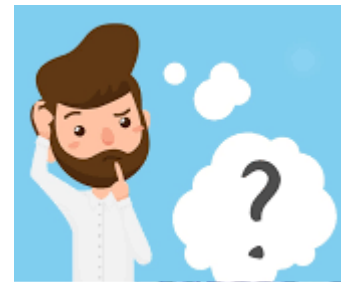
Crie a classe principal que irá utilizar a calculadora.

Principal.java

```
import javax.swing.JOptionPane;
import calculadora.Calculadora;
public class Principal {
    public static void main(String[] args) {
        String opcao = "";
        Calculadora calculadora = new Calculadora();
        while (!opcao.equals("9")) {
            opcao = JOptionPane.showInputDialog("1 - Leitura \n2 - Soma \n3 - Subtração "
                + "\n4 - Produto \n5 - Divisão \n9 - Sair");
            switch (Integer.parseInt(opcao)) {
                case 1:
                    calculadora.setValorA(Double.parseDouble(JOptionPane.showInputDialog("Digite o valor A")));
                    calculadora.setValorB(Double.parseDouble(JOptionPane.showInputDialog("Digite o valor B")));
                    break;
                case 2:
                    JOptionPane.showMessageDialog(null, "Soma: " + calculadora.getAdicao());
                    break;
                case 3:
                    JOptionPane.showMessageDialog(null, "Subtração: " + calculadora.getSubtracao());
                    break;
                case 4:
                    JOptionPane.showMessageDialog(null, "Produto: " + calculadora.getProduto());
                    break;
                case 5:
                    JOptionPane.showMessageDialog(null, "Divisão: " + calculadora.getDivisao());
                    break;
            }
        }
    }
}
```

Passo 4

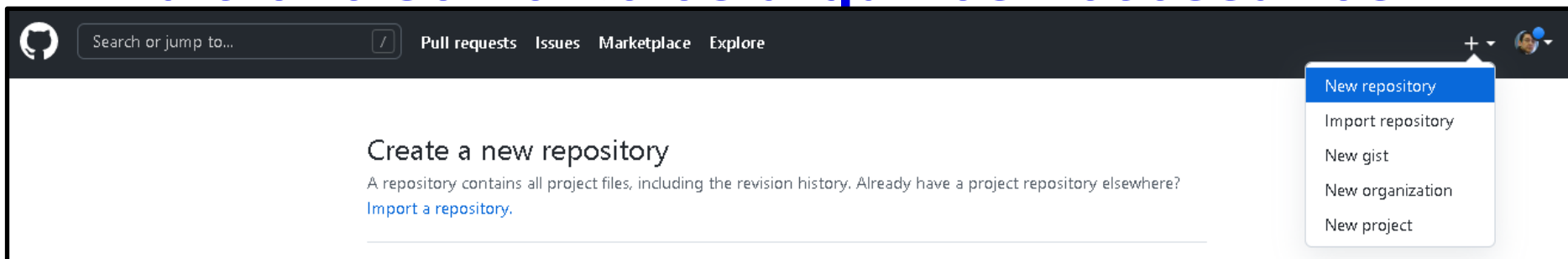
- Terminamos a construção
 - Compile o seu projeto e faça execução.
 - Pense, o que você fez aqui ?



Passo 5

Salve o seu projeto em um repositório no **github** com o mesmo nome do projeto em sua **IDE**.


Adicione somente os arquivos necessários!



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?
[Import a repository.](#)

Owner * **Repository name ***

 osmarbraz ▾ ✓

Great repository names are [Your new repository will be created as calculadora-](#) about [stunning-sniffle?](#)

Description (optional)

Choose a license

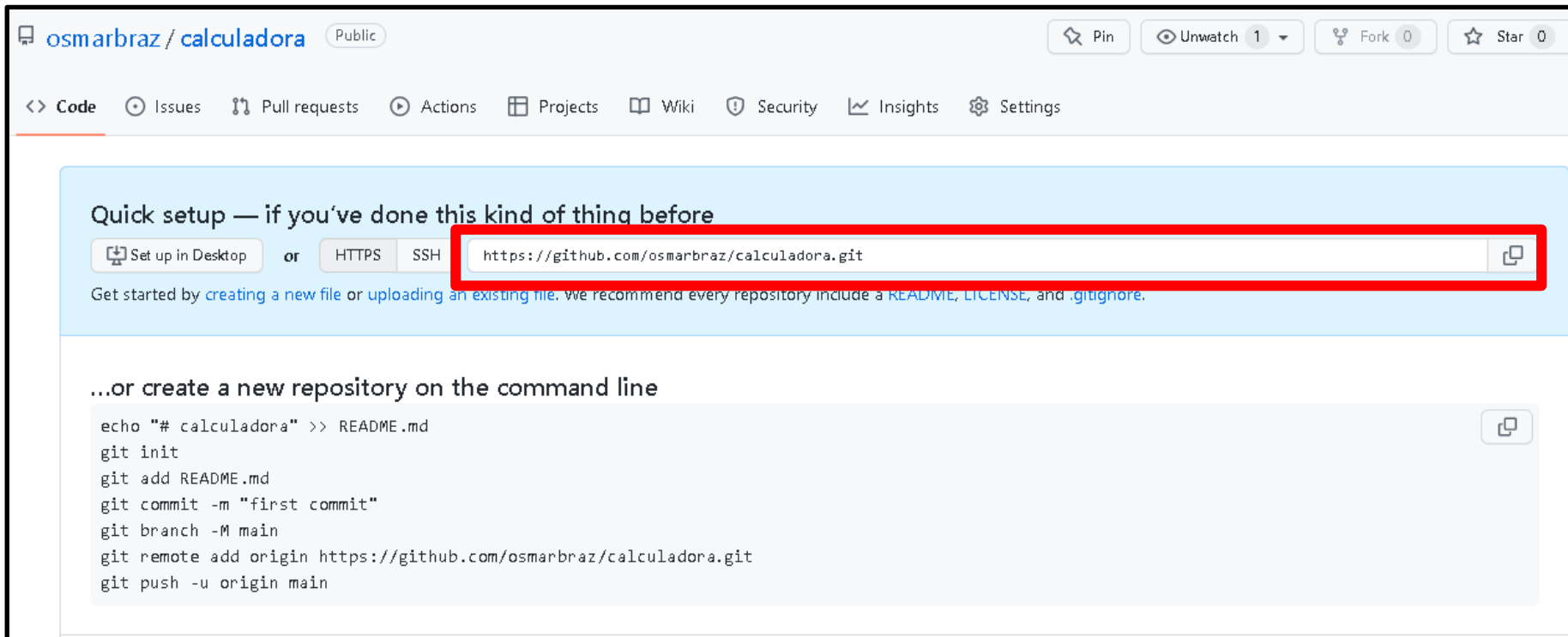
A license tells others what they can and can't do with your code. [Learn more.](#)

License: None ▾

ⓘ You are creating a public repository in your personal account.

Passo 5 - Continuação

Em seu projeto na IDE, inicialize o repositório git localmente, realize o **commit** (somente para **src** e **pom.xml**) e envie remotamente(**push**) para o link.

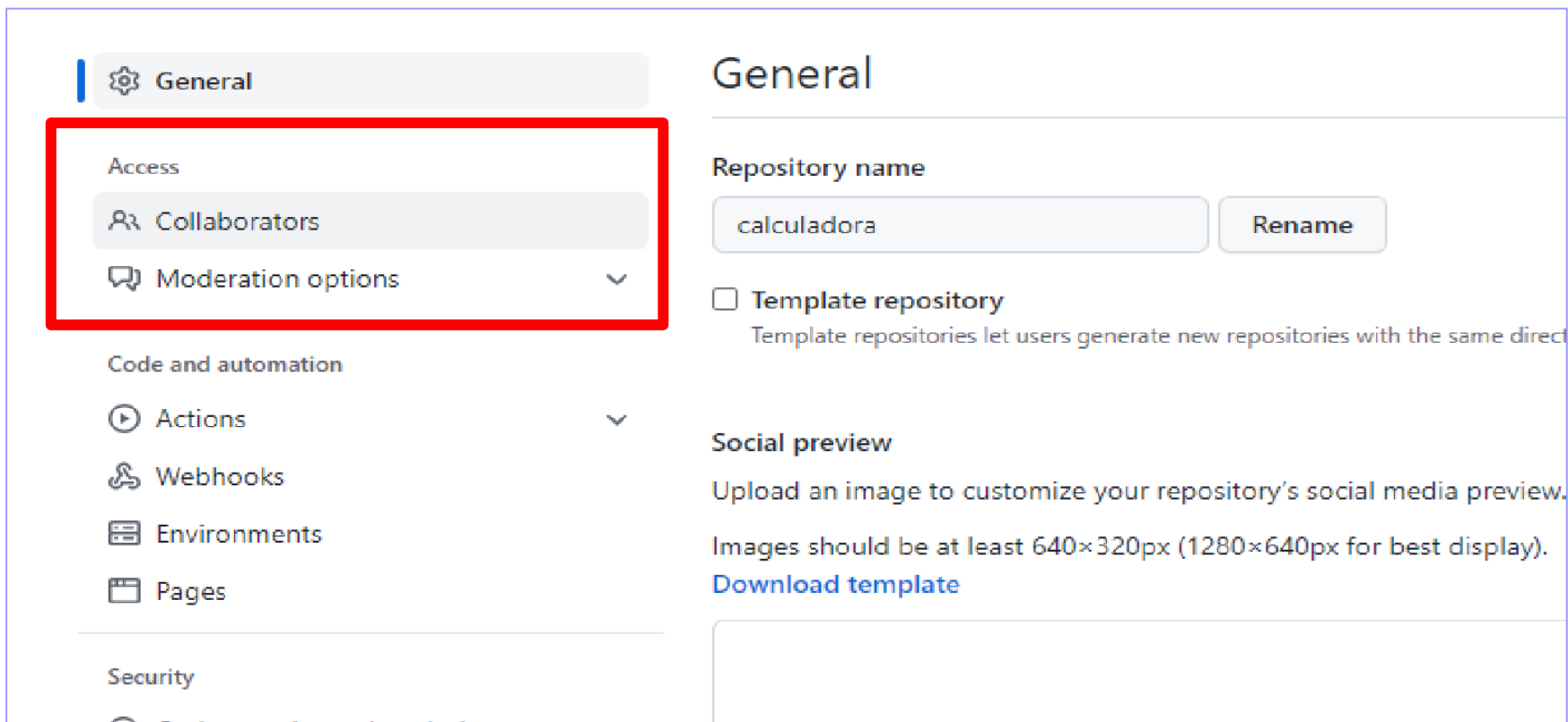


The screenshot shows the GitHub repository page for 'osmarbraz/calculadora'. The repository is public. The 'Quick setup' section is highlighted in light blue. It contains a text input field with the URL 'https://github.com/osmarbraz/calculadora.git' highlighted by a red rectangle. Below the input field, there is a link to 'Get started by creating a new file or uploading an existing file. we recommend every repository include a README, LICENSE, and .gitignore.' Below the 'Quick setup' section, there is a section titled '...or create a new repository on the command line' with a list of commands to initialize a new repository and push it to the remote.

```
echo "# calculadora" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/osmarbraz/calculadora.git
git push -u origin main
```

Passo 6

Adicione todos os integrantes do grupo como colaboradores do repositório no **github**.



The screenshot shows the GitHub repository settings page for a repository named 'calculadora'. The 'General' tab is selected. On the left sidebar, the 'Access' section is highlighted with a red box, containing 'Collaborators' and 'Moderation options'. The main content area shows the 'Repository name' as 'calculadora' with a 'Rename' button. Below this is a checkbox for 'Template repository' which is unchecked. The 'Social preview' section is also visible, with instructions on image dimensions and a 'Download template' link.

General

Access

- Collaborators
- Moderation options

Repository name

calculadora [Rename](#)

☐ **Template repository**
Template repositories let users generate new repositories with the same direct

Social preview
Upload an image to customize your repository's social media preview.
Images should be at least 640×320px (1280×640px for best display).
[Download template](#)

Passo 7

- **Sincronize** o projeto entre todos os membros do grupo.
- Todos do grupo devem ter uma cópia funcional(**executando**)!
- Não avance se algum integrante do grupo não estiver com uma cópia(**clone**) funcionando.



Observação

Dependendo da **IDE**, talvez precise criar uma chave(**token**) de acesso ao GIT (para usar essa chave ao invés de sua senha)

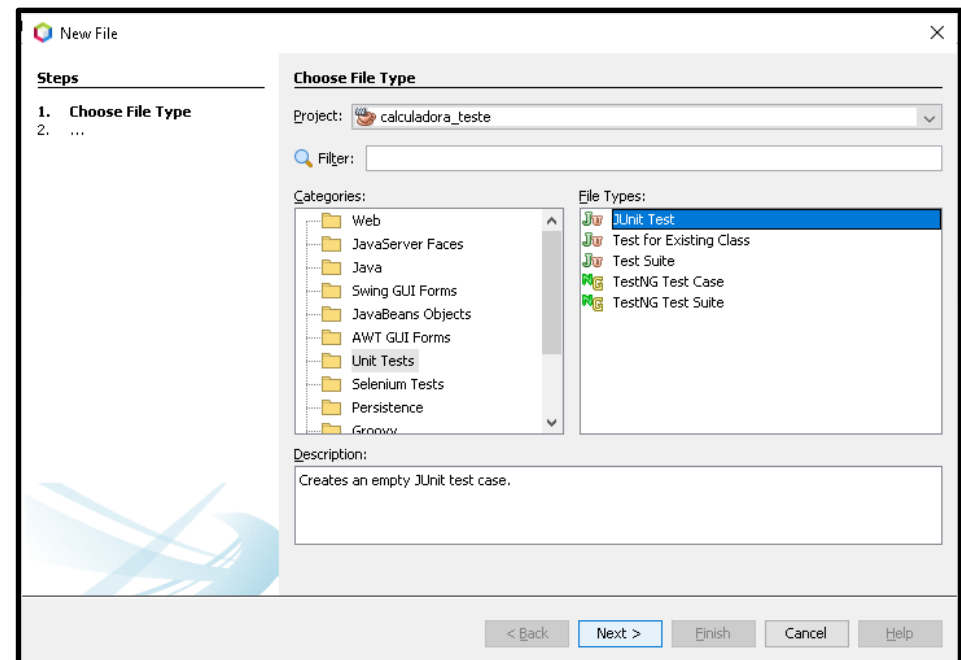
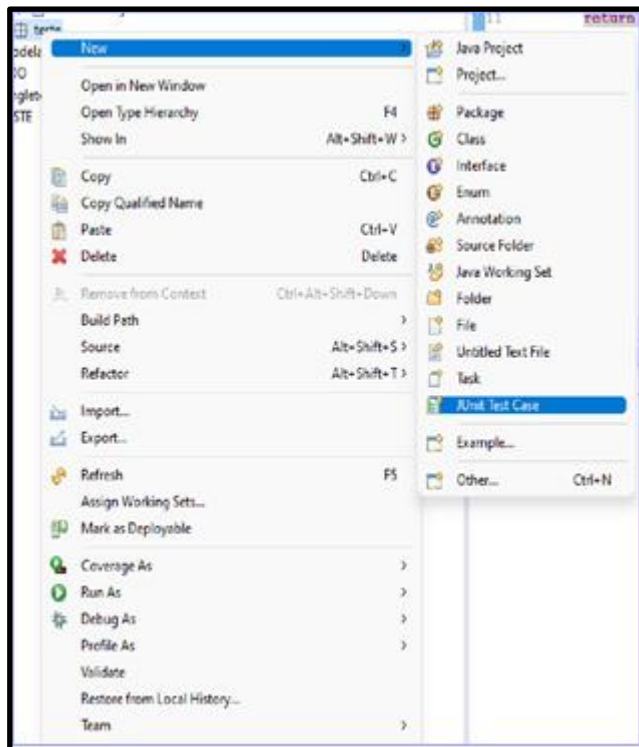
No site do **github**:

1. Menu do usuário->**Settings**
2. Depois no menu lateral esquerdo-><>**Developer Settings**
3. Acesse ->**Personal access token** e clique em **Generate new token**
 - a) De um nome ao token(**Note**)
 - b) Selecione o tempo de validade(**Expiration**)
 - c) Selecione as permissões de acesso (**Select scopes**)



Passo 8

Adicione um teste unitário(*TestCase*) ao projeto chamado *TestCalculadora.java*.



Passo 8 – Continuação

Desenvolva o teste unitário da adição no método **testGetAdicao()** na classe **TestCalculadora**.

TestCalculadora.java

```
package calculadora;

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

public class TestCalculadora {

    @Test
    public void testGetAdicao() {
        Calculadora calculadora = new Calculadora(4.0, 2.0);
        double retornoEsperado = 6.0;
        double retornoFeito = calculadora.getAdicao();
        assertEquals(retornoEsperado, retornoFeito, 0);
    }
}
```

Passo 9

Execute o teste unitário.

```
package calculadora;

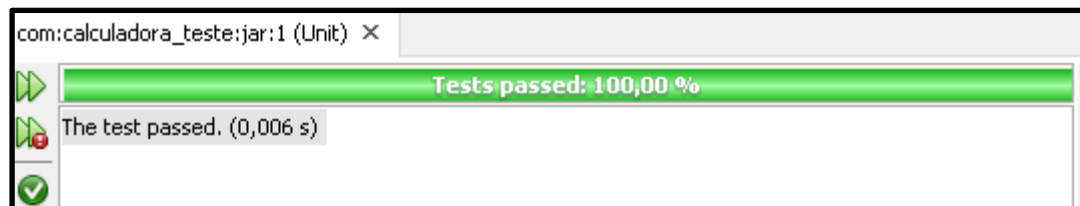
import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class TestCalculadora {

    @Test
    public void testGetAdicao() {
        Calculadora calculadora = new Calculadora();
        double retornoEsperado = 6.0;
        double retornoFeito = calculadora.getAdicao(2, 4);
        assertEquals(retornoEsperado, retornoFeito);
    }
}
```

Navigate	>
Show Javadoc	Alt+F1
Find Usages	Alt+F7
Call Hierarchy	
<hr/>	
Insert Code...	Alt+Insert
Fix Imports	Ctrl+Shift+I
Refactor	>
Format	Alt+Shift+F
<hr/>	
Run File	Shift+F6
Debug File	Ctrl+Shift+F5
Test File	Ctrl+F6

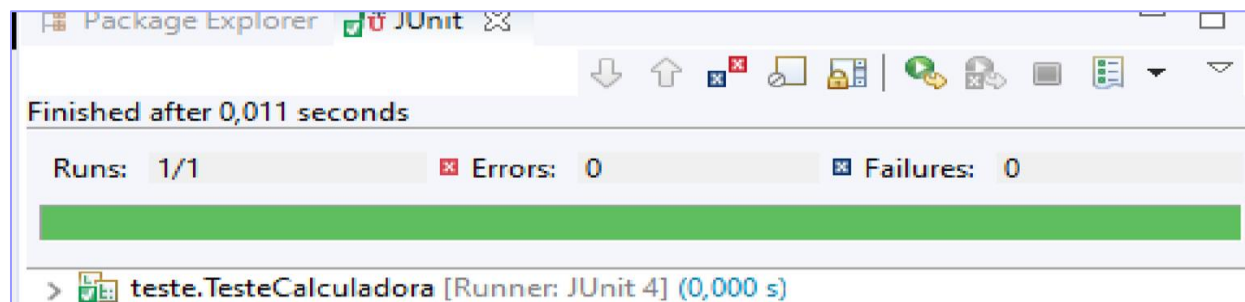
Passo 10



Avalie o resultado do teste unitário na **IDE**.

```
-----  
T E S T S  
-----  
Running calculadora.TestCalculadora  
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.065 sec  
  
Results :  
  
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0  
  
-----  
BUILD SUCCESS  
-----  
  
Total time: 1.623 s  
Finished at: 2022-05-15T11:28:31-03:00  
-----
```

JUnit



Passo 10 – Problemas execução teste

Caso a execução do teste **não** funcione verifique se foi adicionado as **dependências** das bibliotecas do **JUnit4** ao projeto no arquivo **pom.xml**.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com</groupId>
  <artifactId>calculadora</artifactId>
  <version>1</version>
  <packaging>jar</packaging>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.13.2</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>
</project>
```

Passo 11

Crie um suite de teste (*TestSuite*) para agrupar os testes unitários (*TestCases*), só temos um teste neste projeto!

TesteSuite.java

```
import junit.framework.JUnit4TestAdapter;
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

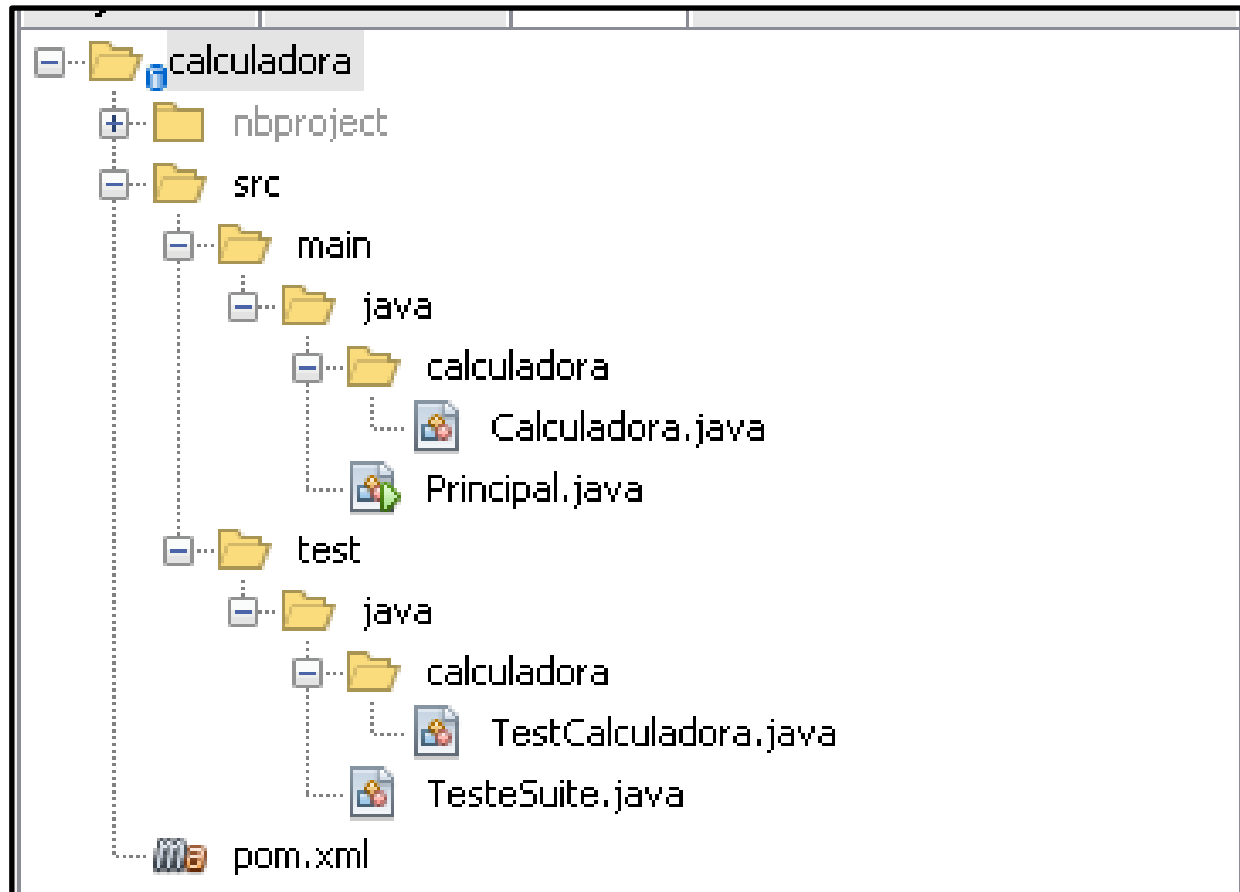
import calculadora.TestCalculadora;

@RunWith(Suite.class)
@SuiteClasses({TestCalculadora.class})
public class TesteSuite {

    public static junit.framework.Test suite() {
        return new JUnit4TestAdapter(TesteSuite.class);
    }
}
```

Passo 11

Antes de **executar os testes verifique** se a estrutura de pastas e arquivos estar organizado desta forma:



Passo 12

Execute o teste pelo *TesteSuite*.

```
-----  
T E S T S  
-----  
Running TesteSuite  
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.075 sec  
  
Results :  
  
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0  
  
-----  
BUILD SUCCESS  
-----  
  
Total time: 1.622 s  
Finished at: 2022-05-15T11:32:51-03:00  
-----
```

Passo 13

Salve os testes unitários e submeta os arquivos dos testes unitários(*TestCase* e *TestSuite*) ao repositório **github**.

Atenção no repositório deve existir somente a pasta **src** e o arquivo **pom.xml**.





Conclusão

- A Integração Continua é um processo essencial a qualquer software que deseja manter vivo por um período de tempo mais longo.
- Conhecer e dominar as ferramentas é um ponto crítico para garantir agilidade no processo de distribuição do software.
- Nesta etapa criamos o projeto e os testes unitários.

Referências

- PRESSMAN, Roger; MAXIM, Bruce. Engenharia de software: uma abordagem profissional. 8.ed. Bookman, 2016. E-book. Disponível em: <https://integrada.minhabiblioteca.com.br/books/9788580555349>
- SOMMERVILLE, Ian. Engenharia de software. 9. ed. São Paulo: Pearson Prentice Hall, 2011. E-book. Disponível em: <https://plataforma.bvirtual.com.br/Leitor/Publicacao/2613/epub/0>
- LARMAN, Craig. Utilizando UML e padrões: uma introdução à análise e ao projeto orientados a objetos e desenvolvimento iterativo. 3. ed Porto Alegre: Bookman, 2007. E-book. Disponível em: <https://integrada.minhabiblioteca.com.br/books/9788577800476>





Fim