FIAP GRADUAÇÃO

Prof. Renato Jardim Parducci

COMPLIANCE & QUALITY ASSURANCE

DISCIPLINA: **COMPLIANCE & QUALITY ASSURANCE**

AULA:
2 – EXEMPLIFICAÇÃO DE COMO FERRAMENTAS E PROCESSOS IMPACTAM A QUALIDADE– JUNIT

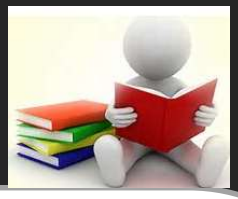
PROFESSOR:
RENATO JARDIM PARDUCCI

PROFRENATO.PARDUCCI@FIAP.COM.BR

AGENDA DA AULA

- ✓ Exemplificação do impacto de processos de trabalho e ferramentas sobre a Qualidade
- ✓ Uso do JUNIT para criar testes automatizados para programas JAVA
- ✓ Uso de Git/GitHub no controle de versão de fontes e documentos do projeto

ESTUDO DE CASO SIMULADO



Apesar das colocações feitas por C e sua equipe de consultoria em qualidade de software, D, o dono da software house, acredita que uma ferramenta mais eficientes para testar software possa eliminar grande parte dos seus problemas com os clientes.

Como a empresa faz programação em linguagem JAVA, C indicou o uso da ferramenta JUNIT que é integrada ao Eclipse.

C preparou um treinamento para aprendizado passo a passo do uso de JUNIT para criar scripts de teste que possam ser executados quantas vezes forem necessários, criando um autômato de teste.

Vamos seguir as instruções do programa de treinamento de C que vem a seguir, a qual inicial com uma pequena introdução sobre testes para conscientizar os desenvolvedores.

GERENCIAMENTO DO TESTE DE SOFTWARE

Teste é uma atividade exaustiva e muitas vezes pouco interessante aos olhos do programador mas,... Quando testar se torna também uma atividade de programação, tudo fica mais divertido (os programadores ficam estimulados a testar)!

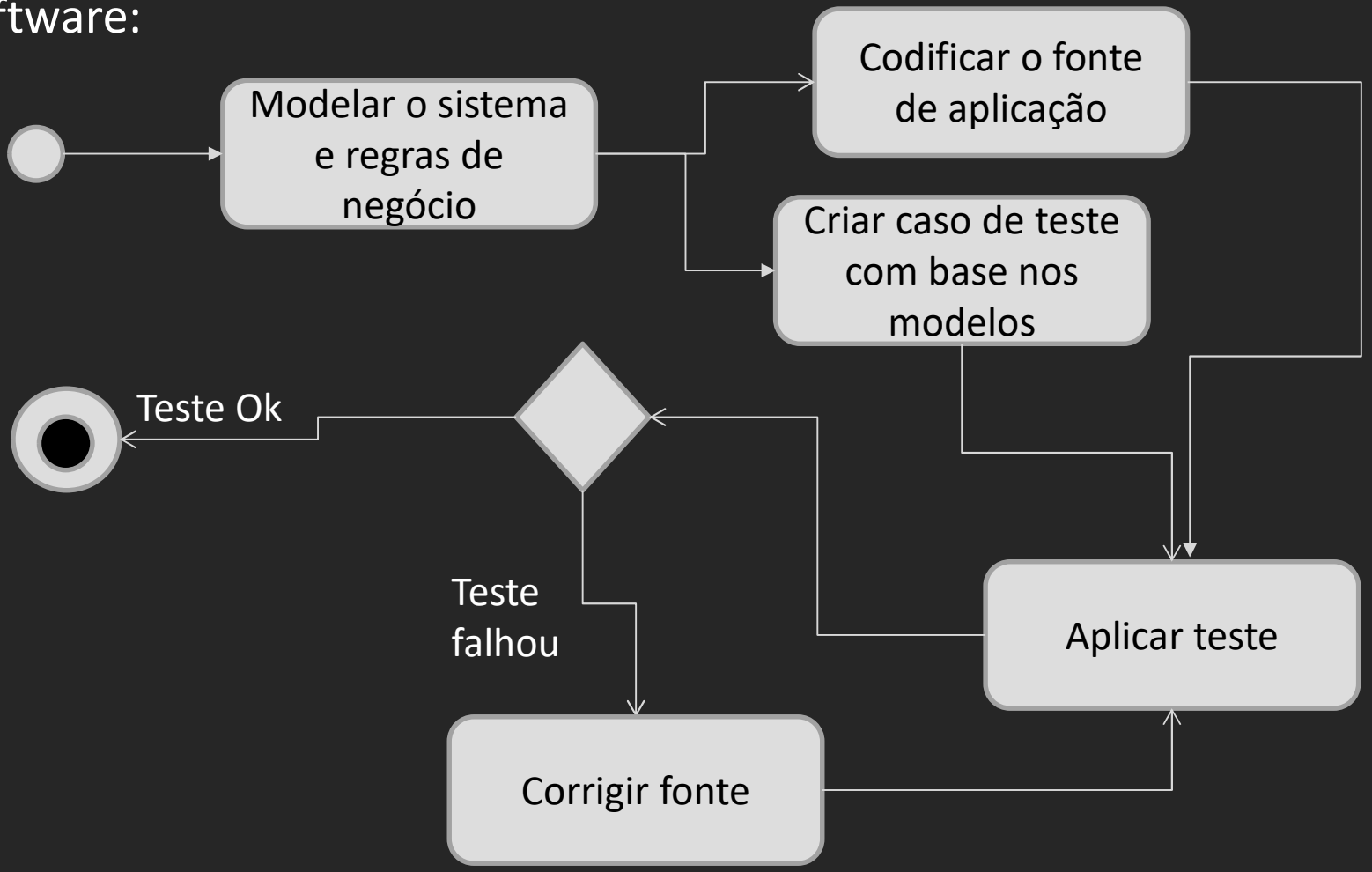
GERENCIAMENTO DO TESTE DE SOFTWARE

Utilizar técnicas apropriadas e ferramentas adequadas para testar programas de aplicação permite:

- Disciplinar o formato dos testes;
- Documentar os casos de teste;
- Aplicar um mesmo teste múltiplas vezes, se necessário (repetir);
- Aproveitar um teste criado por um programador por outras pessoas do time de desenvolvimento (reusar);
- Agilizar a execução dos casos de testes e avalição dos resultados;
- Avaliar se os testes criados cobrem as situações previstas na lógica de um programa de aplicação (análise de cobertura dos testes).

GERENCIAMENTO DO TESTE DE SOFTWARE

Vamos trabalhar neste momento com o processo convencional de teste de software:



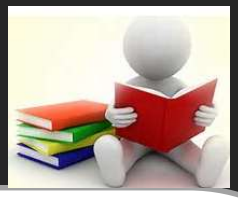
GERENCIAMENTO DO TESTE DE SOFTWARE

Para criar os casos de testes, vamos utilizar a ferramenta aplicada a programas JAVA:

GERENCIAMENTO DO TESTE DE SOFTWARE

Vamos usar JUNIT dentro do Eclipse, aprendendo na prática a criar os testes automatizados:

ESTUDO DE CASO SIMULADO



Você contou a C que não poderá participar do treinamento, por conta de estar ocupado com o desenvolvimento e entrega “pra ontem” de um programa de aplicação de calculadora digital.

C, então, propôs que seja usado o seu programa como exemplo de como usar JUNIT em testes.

Excelente para você!
Vai aprender e ganhar tempo!

GERENCIAMENTO DO TESTE DE SOFTWARE

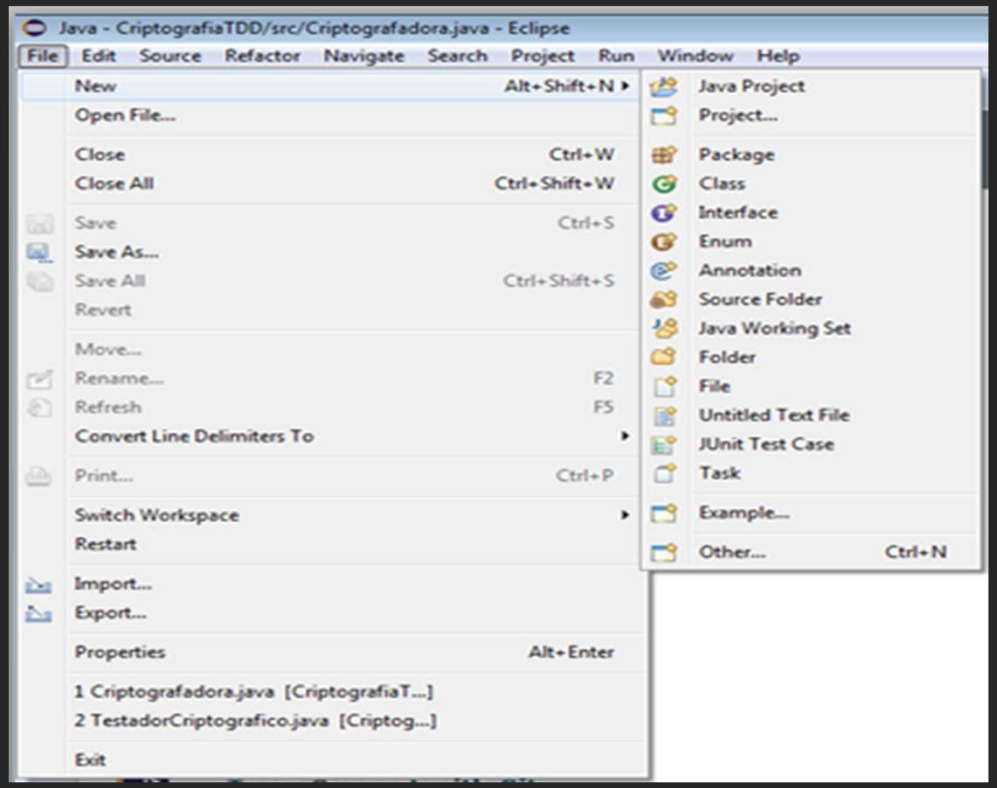
Vamos iniciar pela criação de uma Classe JAVA para trabalharmos os testes.

Nos exemplos a seguir, é considerado que a Classe foi escrita exatamente conforme os algoritmos de especificação e modelo UML que foram definidos para seus atributos e métodos.

Queremos confirmar que ela funciona adequadamente, através dos testes.

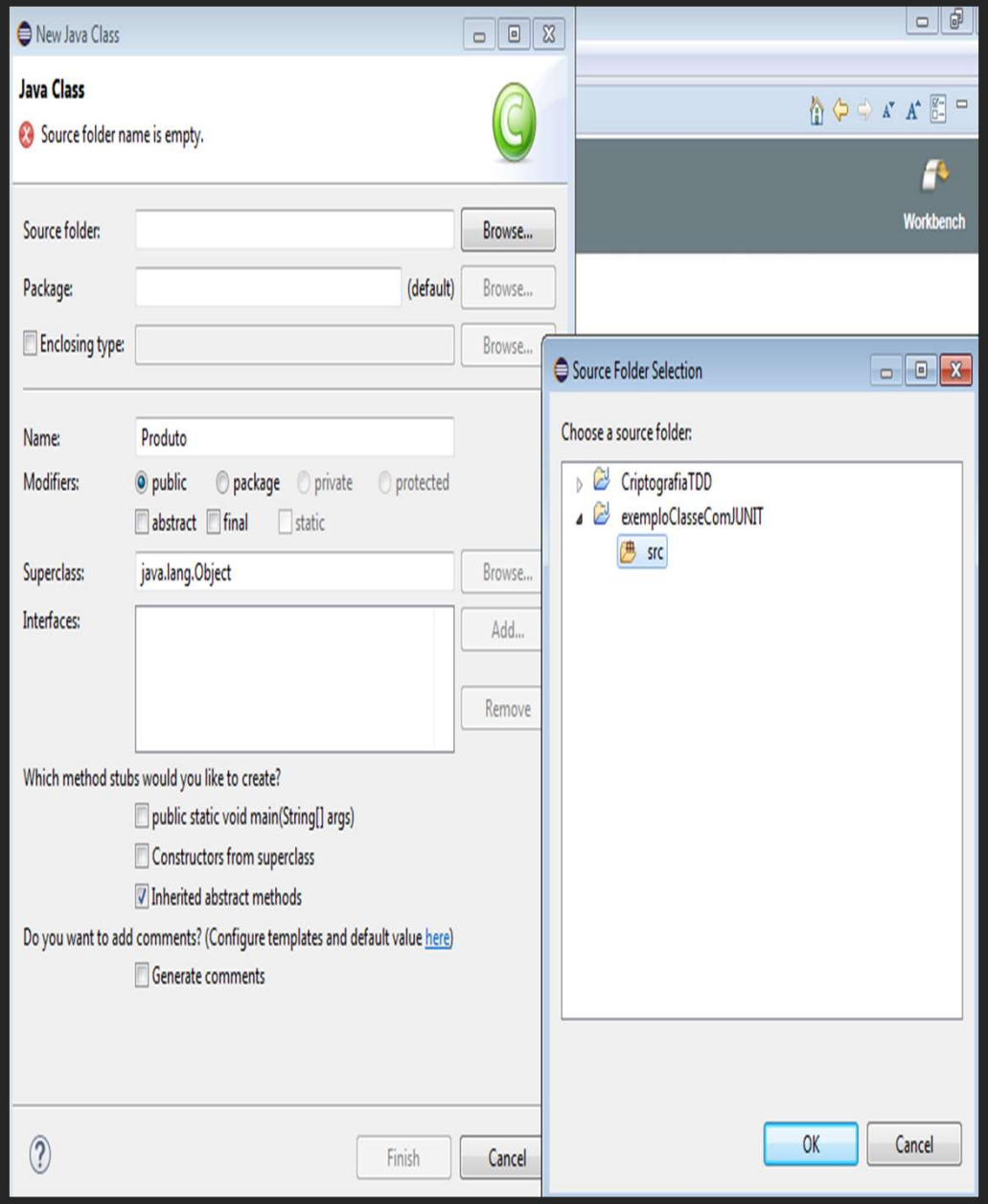
COMPLIANCE & QUALITY ASSURANCE

Crie um projeto no Eclipse e depois, ... a Classe JAVA Calculadora, descrita ao lado *(new JAVA Class EJB)*




```
public class Calculadora{

    // atributo
    private int resultado = 0;

    // método somar
    public int somar( int n1, int n2 ){

       resultado = n1 + n2;
       return resultado;
    }

    // método subtrair
    public int subtrair( int n1, int n2 ){

       resultado = n1 - n2;
       return resultado;
    }

    // método multiplicar
    public int multiplicar( int n1, int n2 ){

       resultado = n1 * n2;
       return resultado;
    }

    // método dividir
    public int dividir( int n1, int n2 ){

       resultado = n1 / n2;
       return resultado;
    }
}
```

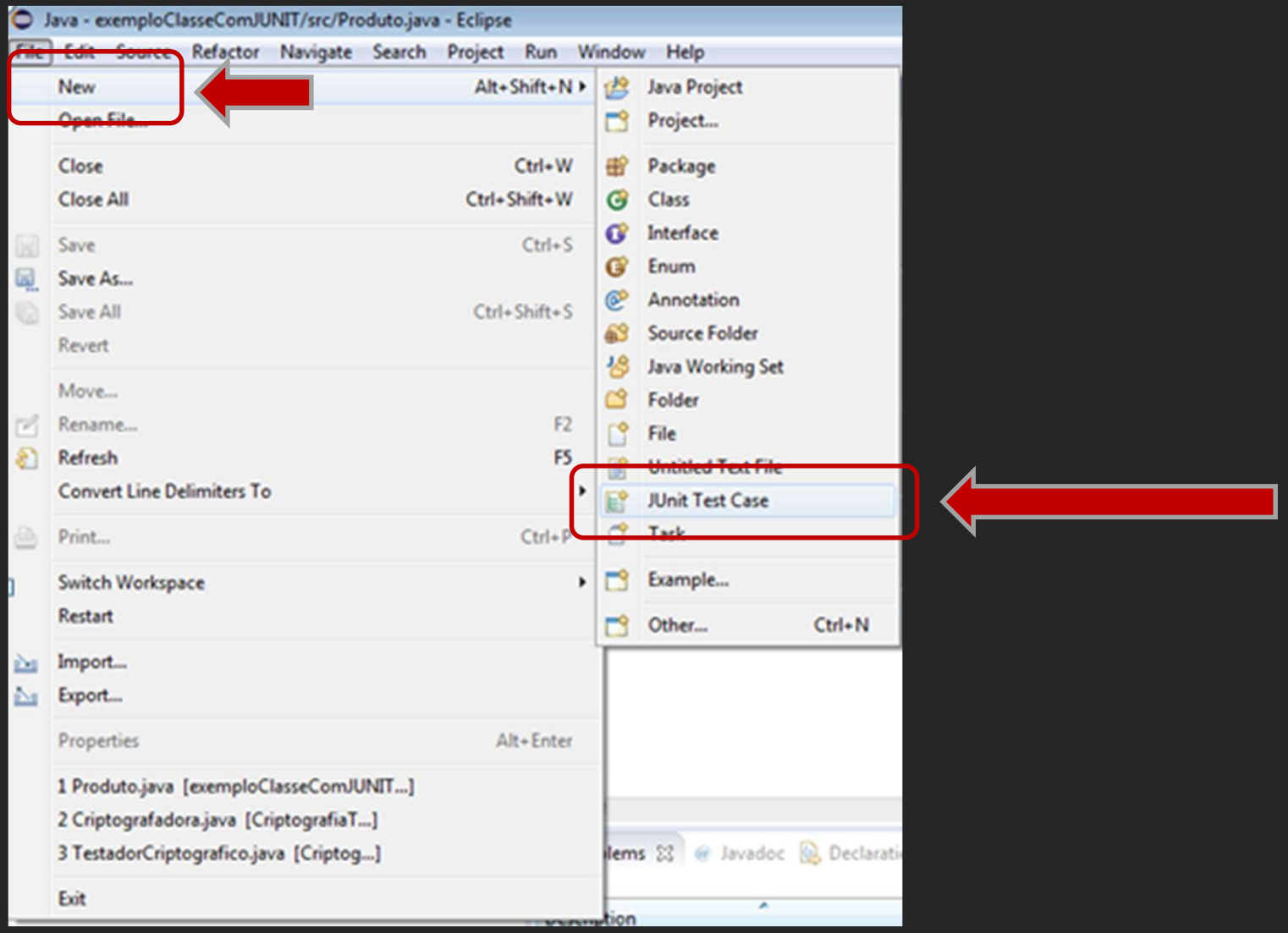
COMPLIANCE & QUALITY ASSURANCE



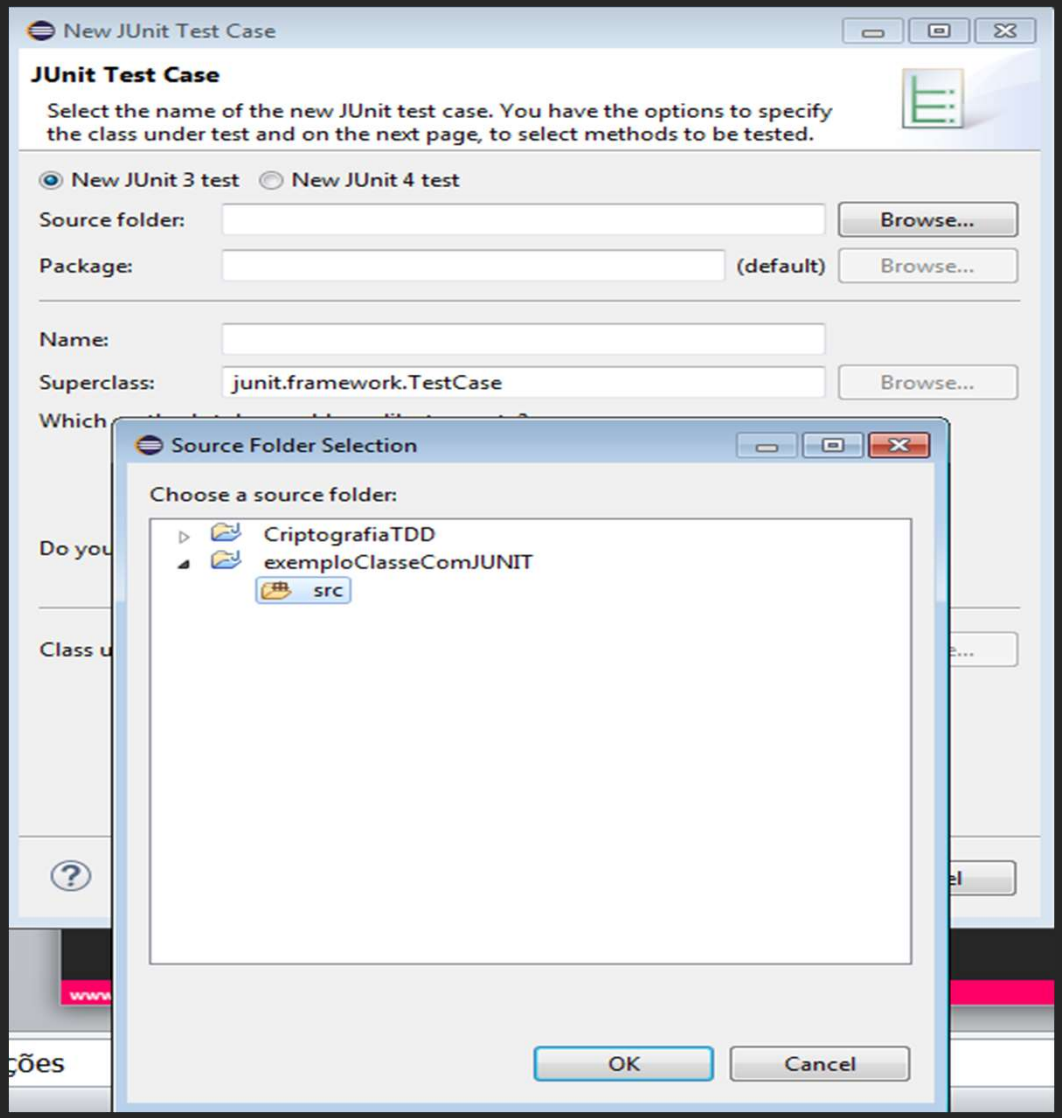

```
public class Calculadora{

    // atributo
    private int resultado = 0;

    // método somar
    public int somar( int n1, int n2 ){

       resultado = n1 + n2;
       return resultado;
    }

    // método subtrair
    public int subtrair( int n1, int n2 ){

       resultado = n1 - n2;
       return resultado;
    }

    // método multiplicar
    public int multiplicar( int n1, int n2 ){

       resultado = n1 * n2;
       return resultado;
    }

    // método dividir
    public int dividir( int n1, int n2 ){

       resultado = n1 / n2;
       return resultado;
    }
}
```

Agora, vamos aos testes...

Crie uma JUNIT Test Case (new JUNIT)



Dê o nome de TesteCalculadora para a Classe de teste que será criada.



GERENCIAMENTO DO TESTE DE SOFTWARE

O processo que vamos seguir para realizar a criação dos testes será:

1º) Criar uma Classe de Teste para Classe de Implementação;

2º) Criar um Método de Teste diferente dentro da Classe de Teste para cada simulação de comportamento que for necessária (um Método para cada Caso de Teste).

3º) Criar e aplicar um Método de Teste por vez, isolando problemas (keep it simple – faça as coisas de forma simples).

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

import junit.framework.TestCase;

public class TesteCalculadoraTest extends TestCase {

    /**
     * Teste de somar na Calculadora.
     */
    @Test
    public void testeSomar() {
        int nro1 = 5;
        int nro2 = 5;
        Calculadora calc= new Calculadora();
        int resultadoEsperado = 10;
        int resultadoReal= calc.somar(nro1, nro2);
        assertEquals(resultadoEsperado, resultadoReal);
    }

}
```

Implementação do Método de teste SOMA, dentro da Classe de Teste de um Objeto Calculadora

COMPLIANCE & QUALITY ASSURANCE

```
import static org.junit.Assert.assertEquals;
import junit.framework.TestCase;

public class TesteCalculadoraTest extends TestCase {

        ...

    /**
     * Teste de subtrair na  Calculadora.
     */
    @Test
    public void testeSubtrair() {
        int nro1 = 5;
        int nro2 = 3;
        Calculadora calc = new Calculadora();
        int resultadoEsperado= 2;
        int resultadoReal= calc.subtrair(nro1, nro2);
        assertEquals(resultadoEsperado resultadoReal);
    }

}
```

Acrescente esse Método de teste da SUBTRAÇÃO logo em seguida do Método de teste de SOMA que você fez anteriormente

```
import static org.junit.Assert.assertEquals;
import junit.framework.TestCase;

public class TesteCalculadoraTest extends TestCase {

   • • •

   /**
    * Teste de multiplicar na Calculadora.
    */
   @Test
   public void testeMultiplicar() {
      int nro1 = 3;
      int nro2 = 3;
      Calculadora calc = new Calculadora();
      int resultadoEsperado = 9;
      int resultadoReal = calc.multiplicar(nro1, nro2);
      assertEquals(resultadoEsperado, resultadoReal);
   }
}
```

Acrescente esse Método de teste da MULTIPLICAÇÃO logo em seguida do Método de teste de SUBTRAÇÃO que você fez anteriormente

```
import static org.junit.Assert.assertEquals;
import junit.framework.TestCase;

public class TesteCalculadoraTest extends TestCase {

   • • •

    /**
     * Teste de dividir na Calculadora.
     */
    @Test
    public void testeDividir() {
        int nro1 = 6;
        int nro2 = 2;
        Calculadora calc = new Calculadora();
        int resultadoEsperado= 3;
        int resultadoReal = calc.dividir(nro1, nro2);
        assertEquals(resultadoEsperado, resultadoReal);
    }

}
```

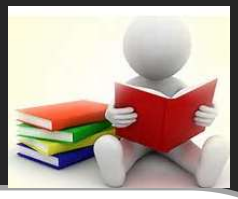
Acrescente esse Método de teste da DIVISÃO logo em seguida do Método de teste de MULTIPLICAÇÃO que você fez anteriormente

Agora, crie a Classe descrita ao lado e a JUNIT para testar todos os métodos da Classe.

Faça os testes para criar um objeto e instanciá-lo e depois testar a recuperação de dados.

```
public class  Produto{

    private double peso;
    private double altura;

    public double getPeso() {
      return peso;
    }

    public void setPeso(double peso) {
      this.peso = peso;
    }

    public double getAltura() {
      return altura;
    }

    public void setAltura(double altura) {
      this.altura = altura;
    }

}
```

ESTUDO DE CASO SIMULADO



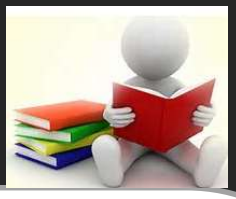
Após o treinamento, um dos participantes falou que muitas vezes, uma aplicação que está desenvolvendo envolve validar várias classes, uma a uma.

Ele quer saber se é possível usar a JUNIT para fazer vários testes simultâneos, de forma organizada.

Uma preocupação adicional é poder ajustar o ambiente de teste (situação de tabelas, conexão de banco, iniciação de variáveis, antes e depois de cada caso de teste aplicado, de forma a evitar que “lixo” (instâncias) de um teste feito, contaminem um teste seguinte.

C apontou que esses recursos existem e vai usar um programa que está em desenvolvimento pela equipe para explicar como funcionam esses recursos.

ESTUDO DE CASO SIMULADO

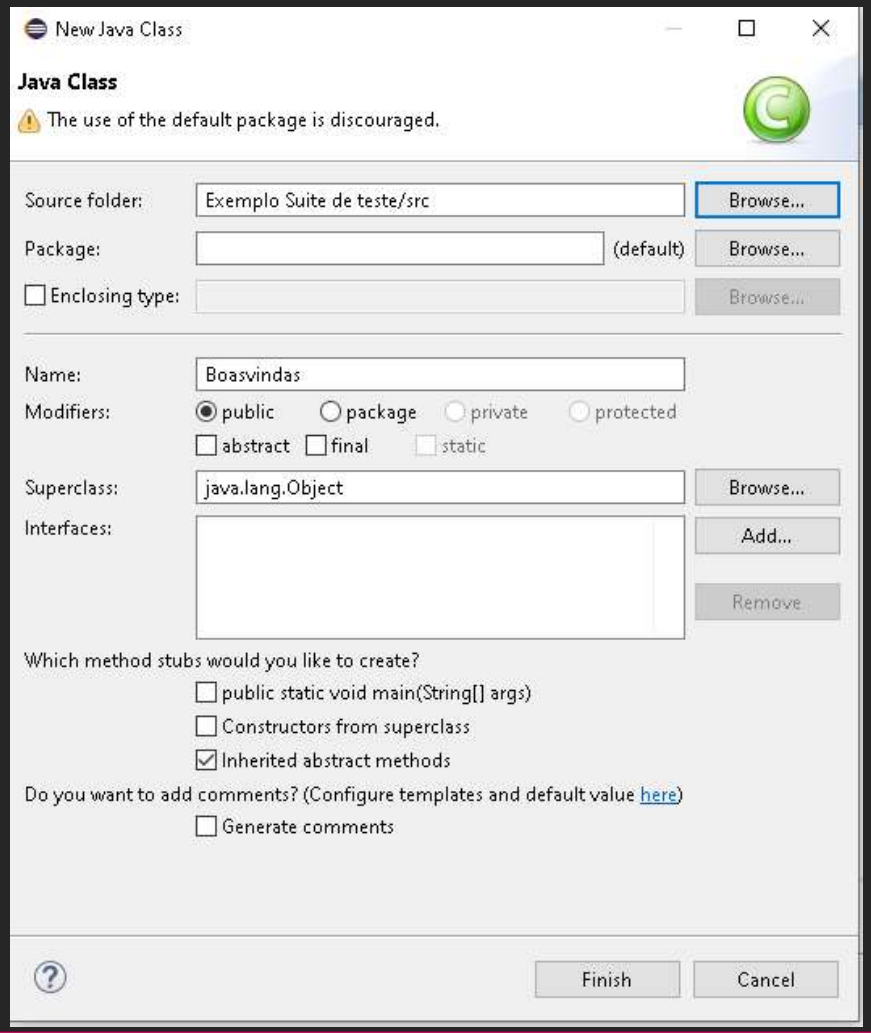


1º) Crie os testes para a Classe a seguir, a qual vai ser validada junto com a Calculadora que você acabou de desenvolver!

Essa Classe, exibe uma mensagem de boas vindas ao usuário da Calculadora digital.

Crie essa Classe dentro do mesmo projeto da Calculadora!

Crie um projeto no Eclipse e depois, ...
a Classe JAVA para exibir mensagem de boas vindas customizada ao usuário de um sistema




```
public class Boasvindas{

  private String menssagem;

  //Construtor de Objeto na Classe
  public Boasvindas(String mens){
    this.menssagem = mens;
  }

  // Exibição da mensagem
  public String exibirMenssagem(){
    System.out.println(this.menssagem);
    return this.menssagem;
  }

  // Exibição da parte fixa da mensagem
  public String completarMenssagem(){
      String compmens;
      compmens = "Ola! Seja bem vindo a sua calculadora pessoal"
      System.out.println(compmens);
      return compmens;
  }
}
```

Agora, vamos criar os testes para essa classe!

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

public class TesteMensagem {

  @Test
  public void testeCriaMesnsagemPadrao() {
                String mensx;
                mensx = "Pedro Bo";
                Boasvindas mensagemUsuario = new Boasvindas(mensx);
                String mensretorno = mensagemUsuario.completarMenssagem();
                assertEquals("Ola! Seja bem vindo a sua calculadora pessoal", mensretorno);
  }

  @Test
  public void testeExibeMesnsagem() {
                String mensx;
                mensx = "Pedro Bo";
                Boasvindas mensagemUsuario = new Boasvindas(mensx);
                String mensRetorno;
                mensRetorno = mensagemUsuario.exibirMenssagem();
                assertEquals(mensx, mensRetorno);
  }
}
```

Você agora tem duas Classe e duas Classes de testes no mesmo projeto!

Para rodá-las todas juntas, precisará criar uma Suite de Teste!

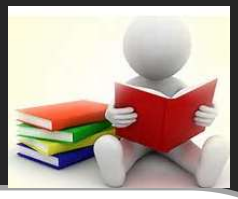
TesteMensagem.java

```
1  import static org.junit.Assert.*;
4
5  public class TesteMensagem {
6      /**
7      * Teste de boas vindas a Calculadora.
8      */
9      @Test
10         public void testeCriaMesnsagemPadrao() {
11             String mensx;
12             mensx = "Pedro Bo";
13             Boasvindas mensagemUsuario = new Boasvindas(mensx);
```

Boasvindas.java | Calculadora.java | TesteCalculadoraTest.java

```
8      * Teste de somar na Calculadora.
9      */
10     @Test
11     public void testeSomar() {
12         int nro1 = 5;
13         int nro2 = 5;
14         Calculadora calc= new Calculadora();
15         int resultadoEsperado = 10;
16         int resultadoReal= calc.somar(nro1, nro2);
17         assertEquals(resultadoEsperado, resultadoReal);
18     }
19
```

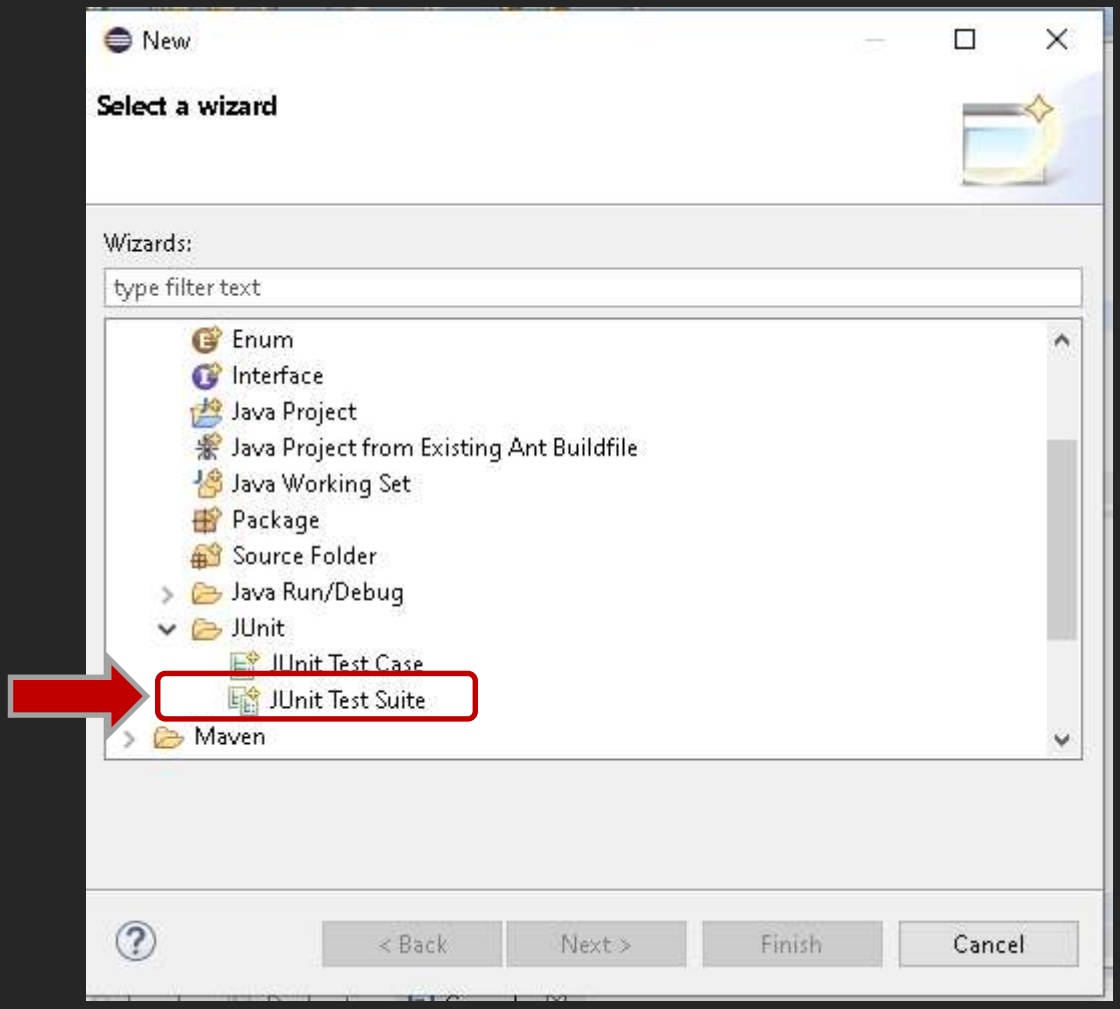
ESTUDO DE CASO SIMULADO



2º) Agora, vamos mixar esses testes com os que você criou anteriormente!

Siga os passos...

Crie a suíte de teste, selecionando o Folder do seu projeto*!*

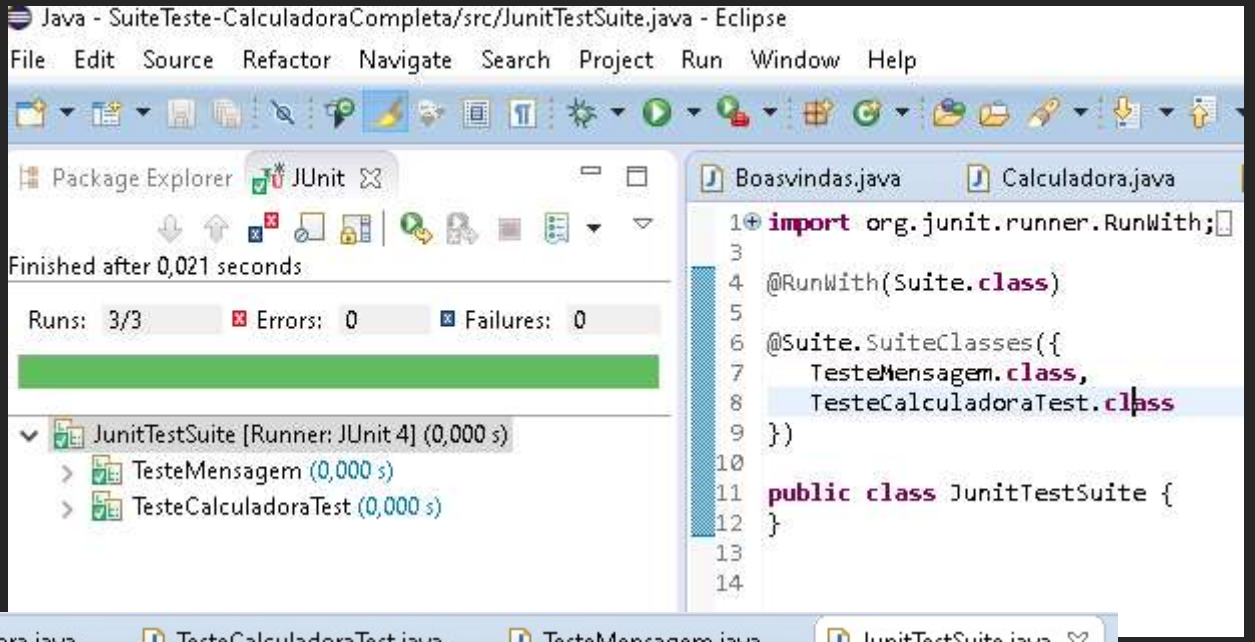


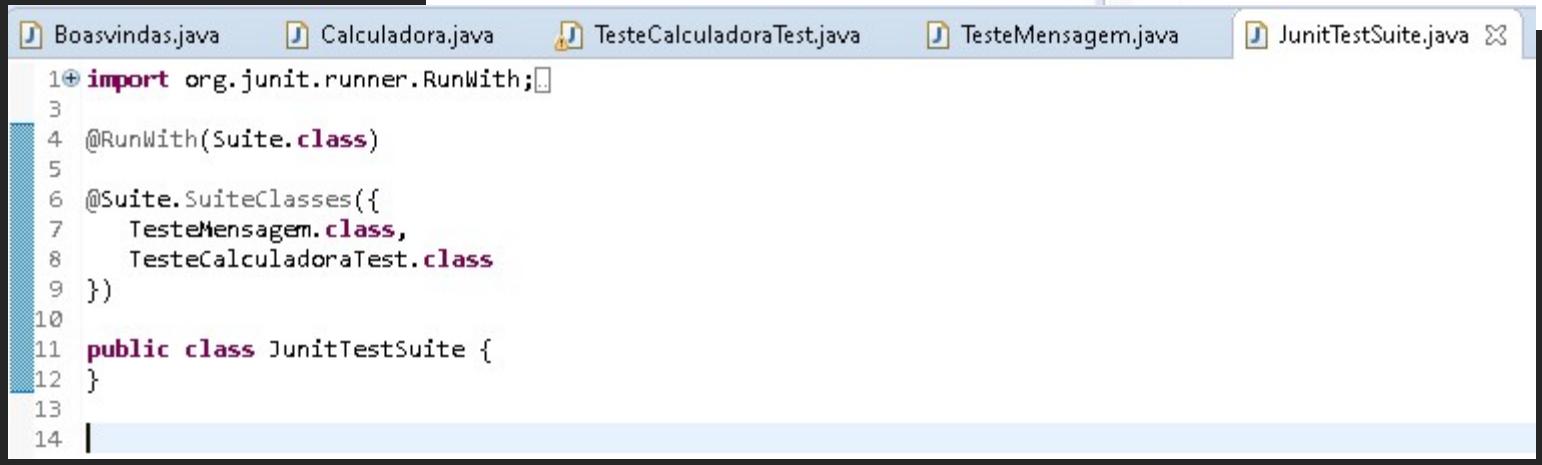
Vamos renomear a suíte para TesteCompletoCalculadora e incluir as chamadas de todas as Classes JUNIT*!*

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

@RunWith(Suite.class)

@Suite.SuiteClasses({
  TesteMensagem.class,
  TesteCalculadoraTest.class
})

public class JunitTestSuite {
}
```

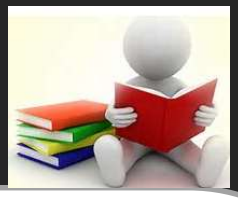
Depois, é só selecionar a Suite na hora de executar os testes e ela mostrará o resultado de cada Classe/Método!

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

@RunWith(Suite.class)

@Suite.SuiteClasses({
  TesteMensagem.class,
  TesteCalculadoraTest.class
})

public class JunitTestSuite {
}
```





ESTUDO DE CASO SIMULADO



Os desenvolvedores querem poder disparar os testes via linha de comandos (prompt)!

É possível? Perguntaram ao C.

C respondeu que é necessário criar um job executor de testes (test runner).
Veja como fazer...

Vamos criar uma nova Classe, chamada Executoradetestes, no mesmo projeto onde está a Suite, as Classes do sistema e as Junit Classes*!*

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

public class Executoradetestes {
  public static void main(String[] args) {
    Result resultado = JUnitCore.runClasses(JunitTestSuite.class);

    for (Failure failure : resultado.getFailures()) {
      System.out.println(failure.toString());
    }

    System.out.println(resultado.wasSuccessful());
  }
}
```

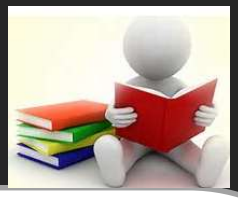
COMPLIANCE & QUALITY ASSURANCE

Você pode executar como uma aplicação JAVA no próprio Eclipse ou executar via linha de comando, após compilação de todas as Classes e Junits no JAVA (comando JAVAR <nome da classe>).

```
C:\JUNIT_WORKSPACE>javac Calculadora.java Boasvindas.java
TesteCalculadoraTest.java TesteMensagem.java Executoradetestes.java
```

```
C:\JUNIT_WORKSPACE>java  Executoradetestes
```

ESTUDO DE CASO SIMULADO



Você está em um projeto de um software e precisa criar uma função que colha 3 digitações de números inteiros, compare os três e diga qual o maior e qual o menor número digitado.

1º) Crie a Classe JAVA e o Método de Captura de Digitação de um número, mais o Método exibir o Maior e outro para exibir o Menor número entre três digitados.

2º) Crie a Classe de Testes e seus Métodos, com JUNIT.

3º) Crie depois a Suíte de teste.
Crie a classe executora dos testes.

4º) Depois que tudo estiver funcionando, transfira a Classe e as JUNITs para o projeto que foi aplicado no treinamento (Calculadora Digital) e ajuste a Test Suite Class e a Test Runner para acomodar essa nova classe e seus testes.

APROVEITE PARA AUTOMATIZAR TESTES UNITÁRIOS COM JUNIT EM TODOS OS SEUS PROJETOS JAVA, DAQUI POR DIANTE!

D ú v I D A S

EXEMPLIFICAÇÃO DE COMO FERRAMENTAS E PROCESSOS IMPACTAM A QUALIDADE E GOVERNANÇA – JUNIT

FIM

PROFESSOR:
RENATO JARDIM PARDUCCI

PROFRENATO.PARDUCCI@FIAP.COM.BR