

# Testes De Software

Professor Gilmar Luiz de Borba

## Prática

### Objetivos:

- Praticar a partir de um exemplo o TDD.
- Entender o processo de criação de uma classe usando o TDD.

(1) Criar o projeto : ProjetoTDD

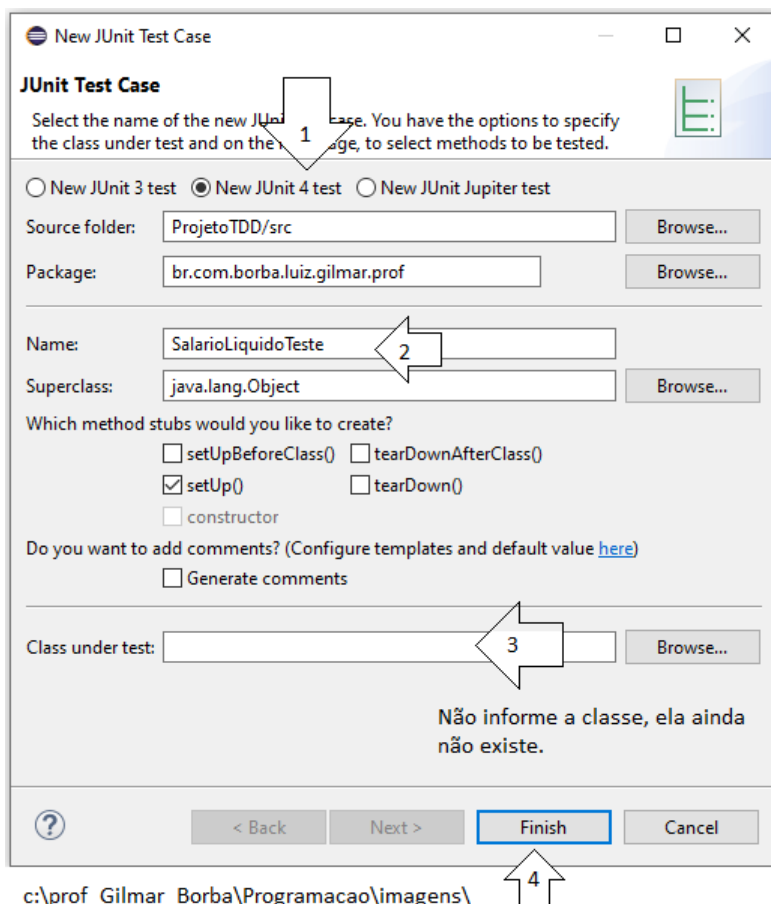
(2) Criar o pacote : com nome a seu critério

(3) Criar a class de teste, não esqueça use JUnit4Test.

Atenção nesse caso, não insira um nome para a classe base, por que? Porque ela ainda não existe!

Deixe a opção setup selecionada para que seja criado o método setup().

Nome da classe de teste: SalarioLiquidoTeste

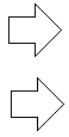


c:\prof\_Gilmar\_Borba\Programacao\imagens\

# Testes De Software

Professor Gilmar Luiz de Borba

(4) Na classe de teste definir o tipo "s" SalarioLiquido, criar o objeto "s". Use o método setup (@Before)



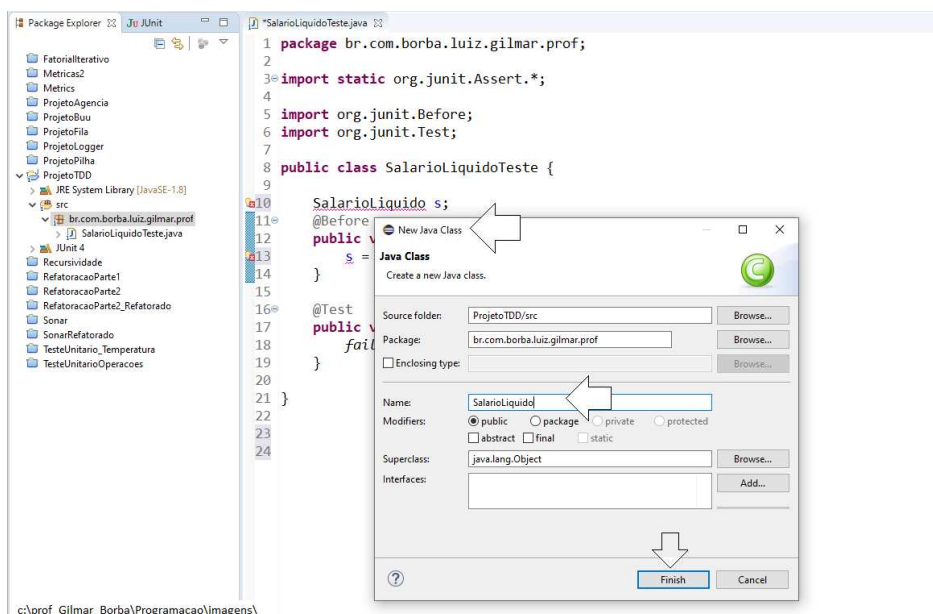
```
1 package br.com.borba.luiz.gilmar.prof;
2
3 import static org.junit.Assert.*;
4
5 import org.junit.Before;
6 import org.junit.Test;
7
8 public class SalarioLiquidoTeste {
9
10     SalarioLiquido s;
11     @Before
12     public void setUp() throws Exception {
13         s = new SalarioLiquido();
14     }
15
16     @Test
17     public void test() {
18         fail("Not yet implemented");
19     }
20
21 }
```

c:\prof\_Gilmar\_Borba\Programacao\imagens\

Observe que foi acusado um erro, porque a classe SalarioLiquido ainda não existe ...

(5) ... Nesse caso volte ao projeto e crie a classe SalarioLiquido. Não é necessário criar o método "main()".

Para calcular o salário líquido é necessário ter os valores dos descontos do INSS, IRRF (Imposto de renda retido na fonte), e benefícios como: plano odontológico, plano de saúde e proventos como por exemplo hora extra. Criaremos os métodos para cada funcionalidade.

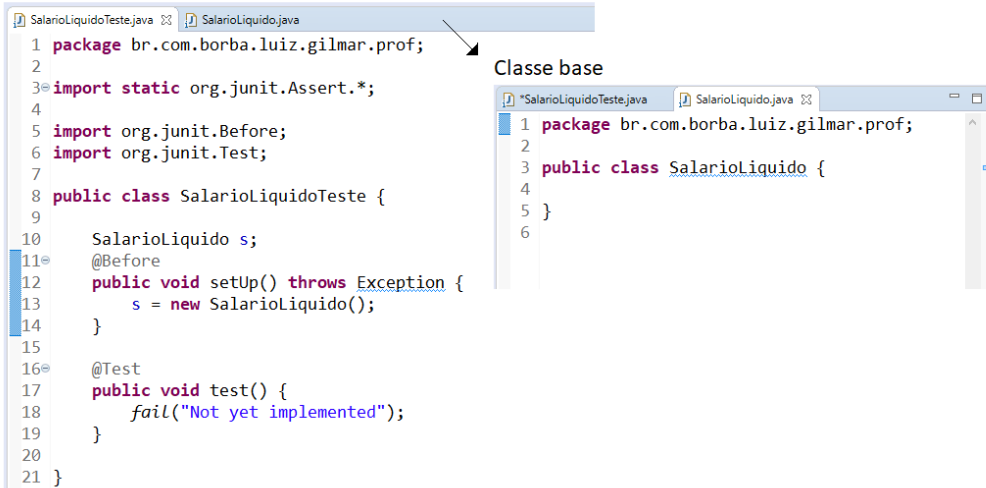


# Testes De Software

Professor Gilmar Luiz de Borba

(6) Agora temos nossa classe de teste e a classe SalarioLiquido criada. Observe que não há mais erro na classe de teste.

Classe de teste



```
SalarioLiquidoTeste.java
1 package br.com.borba.luiz.gilmar.prof;
2
3 import static org.junit.Assert.*;
4
5 import org.junit.Before;
6 import org.junit.Test;
7
8 public class SalarioLiquidoTeste {
9
10     SalarioLiquido s;
11     @Before
12     public void setUp() throws Exception {
13         s = new SalarioLiquido();
14     }
15
16     @Test
17     public void test() {
18         fail("Not yet implemented");
19     }
20
21 }
```

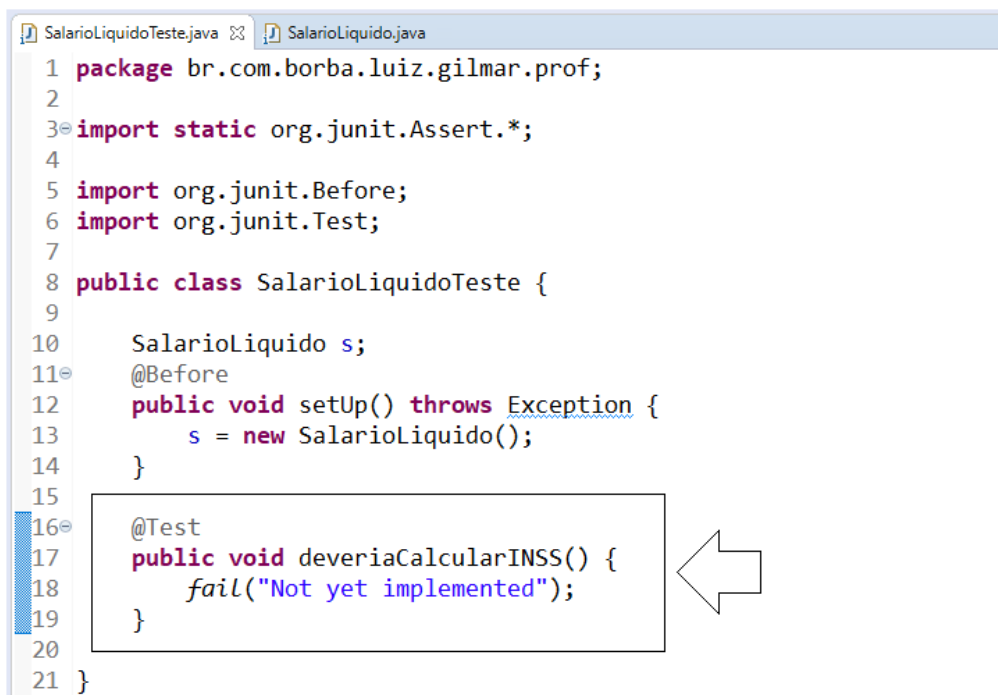
Classe base

```
SalarioLiquido.java
1 package br.com.borba.luiz.gilmar.prof;
2
3 public class SalarioLiquido {
4
5 }
6
```

## Cálculo do INSS

(7) Criar a classe de teste (SalarioLiquidoTeste).

(8) Criar o método "deveriaCalcularINSS", na nossa classe de Teste

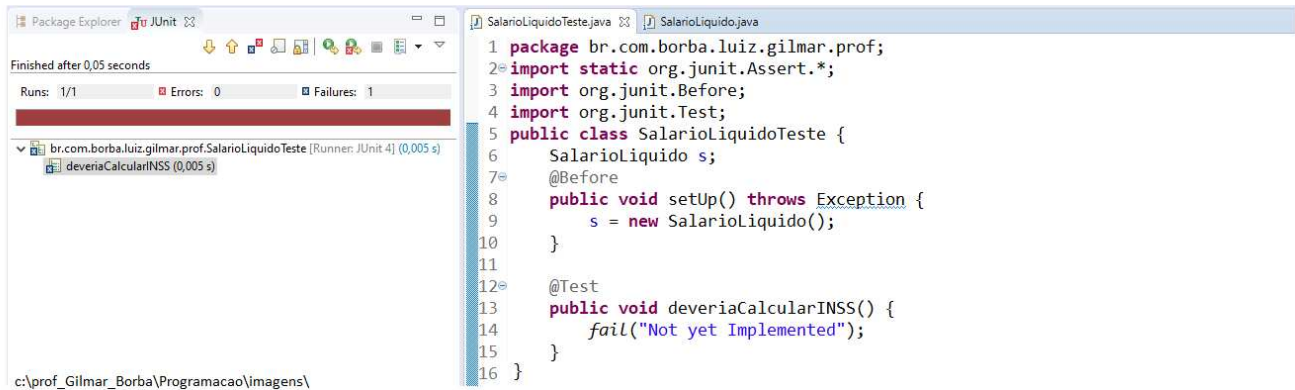


```
SalarioLiquidoTeste.java
1 package br.com.borba.luiz.gilmar.prof;
2
3 import static org.junit.Assert.*;
4
5 import org.junit.Before;
6 import org.junit.Test;
7
8 public class SalarioLiquidoTeste {
9
10     SalarioLiquido s;
11     @Before
12     public void setUp() throws Exception {
13         s = new SalarioLiquido();
14     }
15
16     @Test
17     public void deveriaCalcularINSS() {
18         fail("Not yet implemented");
19     }
20
21 }
```

# Testes De Software

Professor Gilmar Luiz de Borba

Observação. O teste, nesse momento, falhará, porque não existe implementação do mesmo ainda, mas pode falhar também em função de um erro no desenvolvimento da classe base .



```
1 package br.com.borba.l Luiz Gilmar Prof;
2 import static org.junit.Assert.*;
3 import org.junit.Before;
4 import org.junit.Test;
5 public class SalarioLiquidoTeste {
6     SalarioLiquido s;
7     @Before
8     public void setUp() throws Exception {
9         s = new SalarioLiquido();
10    }
11
12    @Test
13    public void deveriaCalcularINSS() {
14        fail("Not yet Implemented");
15    }
16 }
```

(9) Criar o método "calcularINSS" na classe base.

Comentário/História: INSS é usado para financiar benefícios como: aposentadoria, pensão etc. É calculado com base no salário bruto e varia de 8% a 11%. É um desconto obrigatório e é feito mensalmente.

Regras e casos de teste, baseados no Salário Bruto:

(a) até R\$ 1.693,72: desconto de 8%.

(CASO 1: 1.000,00 => desconto: 80,00)

(b) entre R\$ 1.693,73 e R\$ 2.822,90: o desconto será de 9%.

(CASO 2: 2.000,00 => desconto: 180,00)

(c) Salário bruto entre R\$ 2.822,90 e R\$ 5.645,80: o desconto será de 11%.

(CASO 3: 3.000,00 => desconto: 330,00)

(d) Salário bruto acima de R\$ 5.645,80: o desconto será fixo de R\$ 621,04.

(CASO 4: 8000, desconto: 621,04)

# Testes De Software

Professor Gilmar Luiz de Borba

Classe base “*SalarioLiquido*” implementada com o método *calcularINSS*.

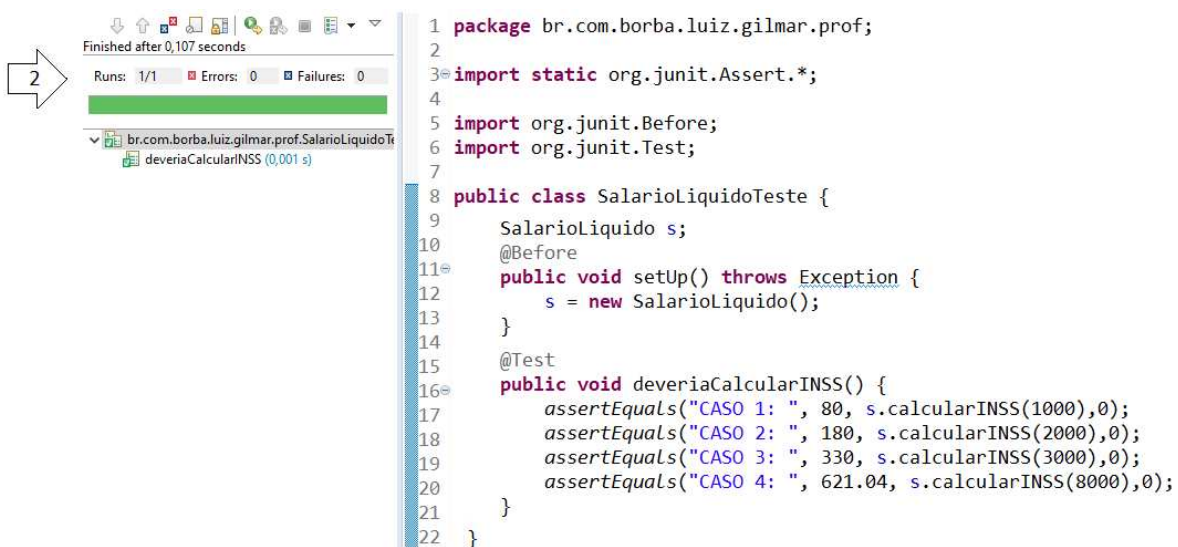
```
public double calcularINSS(double salarioBruto) {
    double valorINSS = 0;
    if (salarioBruto < 1693.72)
        valorINSS = (salarioBruto * 8/100); // desconto 8%.
    else if ((salarioBruto >= 1693.73) & (salarioBruto < 2822.90))
        valorINSS = (salarioBruto * 9/100); //desconto 9%.
    else if ((salarioBruto >= 2822.90) & (salarioBruto < 5645.80))
        valorINSS = (salarioBruto * 11/100); //desconto 9%.
    else
        //salarioBruto >= 5645.80
        valorINSS = 621.04; // desconto fixo 621,04
    return valorINSS;
}
```

c:\prof\_Gilmar\_Borba\Programacao\imagens\

Volte a classe de teste, implemente os casos de teste no método "deveriaCalcularINSS". Veja a classe de teste implementada.

```
@Test
public void deveriaCalcularINSS() {
    // fail("Not vet Implemented");
    assertEquals("CASO 1: ", 80, s.calcularINSS(1000),0);
    assertEquals("CASO 2: ", 180, s.calcularINSS(2000),0);
    assertEquals("CASO 3: ", 330, s.calcularINSS(3000),0);
    assertEquals("CASO 4: ", 621.04, s.calcularINSS(8000),0);
}
```

Imagem dos testes do método “deveriaCalcularINSS” aprovados:



# Testes De Software

Professor Gilmar Luiz de Borba

(9) **REFATORE!** verifique nomes de variáveis, métodos, se há necessidade de criar constantes etc.

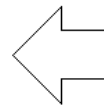
## Cálculo do IRRF

(10) ainda na classe de teste (SalarioLiquidoTeste), criar o método de teste: deveriaCalcularIRRF.

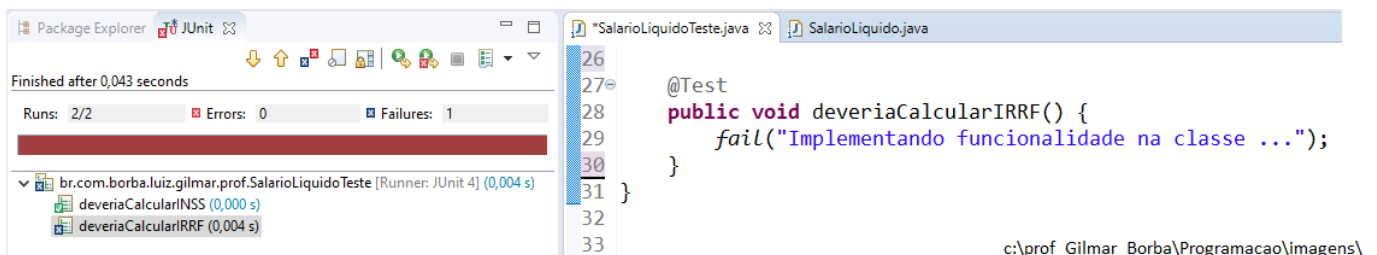
Comentário/História: Trata-se de uma antecipação do pagamento Imposto de Renda que é cobrado pela Receita Federal dos trabalhadores que possuem carteira assinada. É também um desconto obrigatório e é feito mensalmente.

```
...
@Test
public void deveriaCalcularINSS() {
    assertEquals("CASO 1: ", 80, s.calcularINSS(1000),0);
    assertEquals("CASO 2: ", 180, s.calcularINSS(2000),0);
    assertEquals("CASO 3: ", 330, s.calcularINSS(3000),0);
    assertEquals("CASO 4: ", 621.04, s.calcularINSS(8000),0);
}

@Test
public void deveriaCalcularIRRF() {
    fail("Implementando funcionalidade na classe base ...");
}
...
```



\* Note que é importante codificar o método "fail()" para evitar que testes ainda não implementado passem.



(11) Observe que não existe o método calcularIRRF, volte a classe base e implemente esse método.

# Testes De Software

Professor Gilmar Luiz de Borba

Comentário/História: Trata-se de uma antecipação do pagamento Imposto de Renda que é cobrado pela Receita Federal dos trabalhadores que possuem carteira assinada. É também um desconto obrigatório e é feito mensalmente. Para simplificar, nesse exemplo, será considerado somente um valor a deduzir, esquecendo a alíquota.

Regras e casos de teste, baseados no Salário Bruto:

- (a) até 1.903,98  
isento do IRRF.  
(CASO 5: 10000, desconto: 0,00)
- (b) entre 1.903,98 e 2.826,65:  
valor a deduzir de R\$ 150,00.  
(CASO 6: 2.200, desconto: 150,00)
- (c) entre 2.826,65 e 3.751,05:  
valor a deduzir de R\$ 350,00  
(CASO 7: 3.000, desconto: 350,00)
- (d) Salário bruto entre 3.751,05 e 4.664,68:  
valor a deduzir de R\$ 600,00.  
(CASO 8: 4.000, desconto: 600,00)
- (e) acima de 4.664,68:  
deduzir de R\$ 900,00.  
(CASO 9: 5.000, desconto: 900,00)

Classe ***calcularOImpostoDerendaRetidoNaFonte*** (classe base)

# Testes De Software

Professor Gilmar Luiz de Borba

...

```
public double calcularOImpostoDeRendaRetidoNaFonte(double salarioBruto) {  
    double valorIRRF=0;  
    if (salarioBruto <= 1903.98)  
        valorIRRF = 0;  
    else if ((salarioBruto > 1903.98)&(salarioBruto<=2826.65))  
        valorIRRF = 150.00;  
    else if ((salarioBruto > 2826.65)&(salarioBruto<=3751.05))  
        valorIRRF = 350.00;  
    else if ((salarioBruto>3751.05)&(salarioBruto<=4664.68))  
        valorIRRF = 600.00;  
    else // acima de 4.664,68  
        valorIRRF = 900.00;  
    return valorIRRF;  
}
```

...

Volte a classe de teste, implemente os casos de teste no método "deveriaCalcularIRRF"

@Test

```
public void deveriaCalcularIRRF() {  
    assertEquals("CASO 5: ", 0, s.calcularOImpostoDeRendaRetidoNaFonte(1000),0);  
    assertEquals("CASO 6: ", 150, s.calcularOImpostoDeRendaRetidoNaFonte(2200),0);  
    assertEquals("CASO 7: ", 350, s.calcularOImpostoDeRendaRetidoNaFonte(3000),0);  
    assertEquals("CASO 8: ", 600, s.calcularOImpostoDeRendaRetidoNaFonte(4000),0);  
    assertEquals("CASO 9: ", 900, s.calcularOImpostoDeRendaRetidoNaFonte(5000),0);  
}
```

(12) **REFATORE!** verifique nomes de variáveis, métodos, se há necessidade de criar constantes etc.

Método da classe base refatorada. => Refatore e salve o projeto!

Foi feita refatoração de mudança de nome de: **calcularOImpostoDerendaRetidoNaFonte**,

para: **calcularIRRF**



# Testes De Software

Professor Gilmar Luiz de Borba

...

```
public double calcularIRRF(double salarioBruto) {  
    double valorIRRF=0;  
    if (salarioBruto <= 1903.98)  
        valorIRRF = 0;  
    else if ((salarioBruto > 1903.98)&(salarioBruto<=2826.65))  
        valorIRRF = 150.00;  
    else if ((salarioBruto > 2826.65)&(salarioBruto<=3751.05))  
        valorIRRF = 350.00;  
    else if ((salarioBruto>3751.05)&(salarioBruto<=4664.68))  
        valorIRRF = 600.00;  
    else // acima de 4.664,68  
        valorIRRF = 900.00;  
    return valorIRRF;  
}
```

...

Após refatorar a classe base, salve o projeto para que as alterações façam efeito na classe de teste.

Método da classe de teste após a refatoração (automático)

...

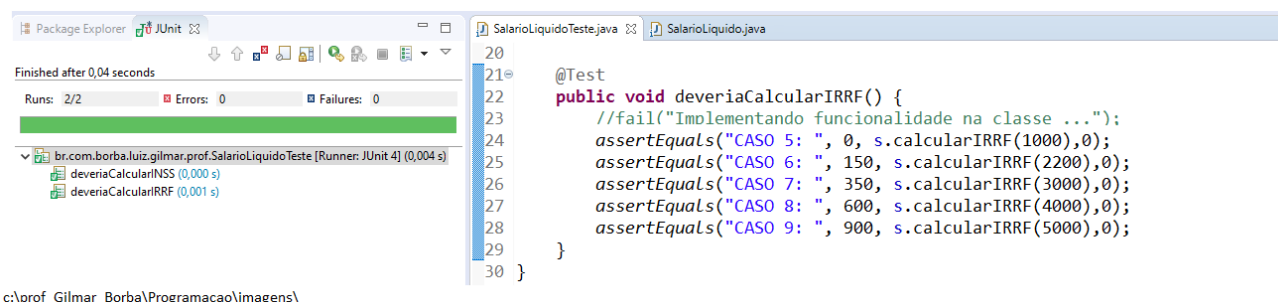
@Test

```
public void deveriaCalcularIRRF() {  
    assertEquals("CASO 5: ", 0, s.calcularIRRF(1000),0);  
    assertEquals("CASO 6: ", 150, s.calcularIRRF(2200),0);  
    assertEquals("CASO 7: ", 350, s.calcularIRRF(3000),0);  
    assertEquals("CASO 8: ", 600, s.calcularIRRF(4000),0);  
    assertEquals("CASO 9: ", 900, s.calcularIRRF(5000),0);  
}
```

...

c:\prof\_Gilmar\_Borba\Programacao\imagens\

Imagem dos testes do método “calcularIRRF” aprovados:



# Testes De Software

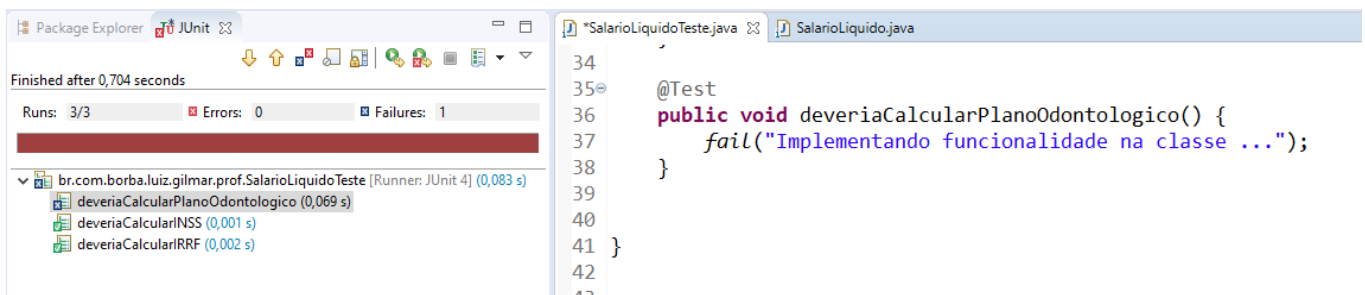
Professor Gilmar Luiz de Borba

## Cálculo do Valor do Plano Odontológico

(13) Criar o método de teste: `deveriaCalcularPlanoOdontologico` e o método `calcularPlanoOdontologico` na classe base.

Comentário/História: considerar somente o valor a ser cobrado do funcionário, desconto (exemplo): 5% independente do salário.

`valorOdontologico = (salarioBruto * (5/100));`



Criar o método na classe base:

```
public double calcularPlanoOdontologico(double salarioBruto) {  
    double valorOdontologico=0;  
    valorOdontologico = (salarioBruto * 5/100);  
    return valorOdontologico;  
}
```

Implemente os casos de teste:

(CASO 10: 3.000, desconto: 150,00)

(CASO 11: 2.000, desconto: 100,00)

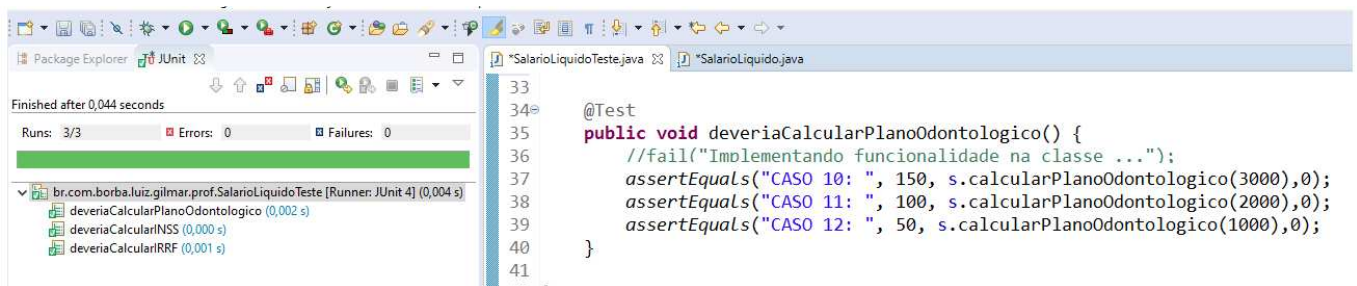
(CASO 12: 1.000, desconto: 50,00)

# Testes De Software

Professor Gilmar Luiz de Borba

```
@Test
public void deveriaCalcularPlanoOdontologico() {
    //fail("Implementando funcionalidade na classe ...");
    assertEquals("CASO 10: ", 150, s.calcularPlanoOdontologico(3000),0);
    assertEquals("CASO 11: ", 100, s.calcularPlanoOdontologico(2000),0);
    assertEquals("CASO 12: ", 50, s.calcularPlanoOdontologico(1000),0);
}
```

Imagem dos testes do método “deveriaCalcularPlanoOdontologico” aprovados:



Siga os passos anteriores ...

(14) Criar o método de teste: `deveriaCalcularPlanoSaude` e o método `calcularPlanoSaude` na classe base. Comentário/História: considerar somente o valor a ser cobrado do funcionário, desconto (exemplo): 10% independente do salário.

Implemente os casos de teste:

(CASO 13: 3.000, desconto: 300,00)

(CASO 14: 2.000, desconto: 200,00)

(CASO 15: 1.000, desconto: 100,00)

Siga os passos anteriores ...

# Testes De Software

Professor Gilmar Luiz de Borba

(15) Criar o método de teste: deveriaCalcularValorHora e o método calcularValorHora na classe base. Esse método servirá de base para calcular a hora extra.

Comentário/História: considerar valor único para diurno/noturno, 44 horas semanais, mês de 5 semanas (220 horas), esse método auxiliará o cálculo de proventos.

Implemente os casos de teste:

(CASO 16: para salário bruto de 3000, valor da hora: 13,64)

(CASO 17: para salário bruto de 2000, valor da hora: 9,09)

(CASO 18: para salário bruto de 1000, valor da hora: 4,55)

Siga os passos anteriores ...

(16) Criar o método de teste deveriaCalcularHoraExtra e o método calcularHoraExtra na classe base. A expressão para o cálculo da hora extra será:

$\text{valorHoraExtra} = \text{quantidadeHoras} * \text{valorDaHoraTrabalhada}$

Implemente os casos de teste:

(CASO 19: salário bruto: 3000, 30 horas extras, valor do provento:  $30 * 13,64 = 409,09$ )

(CASO 20: salário bruto: 2000, 30 horas extras, valor do provento:  $30 * 9,09 = 272,73$ )

(CASO 21: salário bruto: 1000, 30 horas extras, valor do provento:  $30 * 4,55 = 136,36$ )

Siga os passos anteriores ...

(17) Criar o método de teste para deveriaCalcularSalarioLiquido e o método calcularSalarioLiquido na classe base.

Comentário/História: calcular salário líquido com base nos cálculos acima.

Salário Líquido = salario bruto - calcularINSS

- calcularIRRF

- calcularPlanoOdontologico

- calcularPlanoSaude

+ calcularHoraExtra

# Testes De Software

Professor Gilmar Luiz de Borba

Implemente os casos de teste:

CASO 22:

para salário bruto de 3.000

30 horas extras (valor da hora extra = 13,64)

salário líquido = 2.279,09

CASO 23: 1642.73

para salário bruto de 2.000

30 horas extras (valor da hora extra = 9,09)

salário líquido = 1642,09

CASO 24:

para salário bruto de 1.000

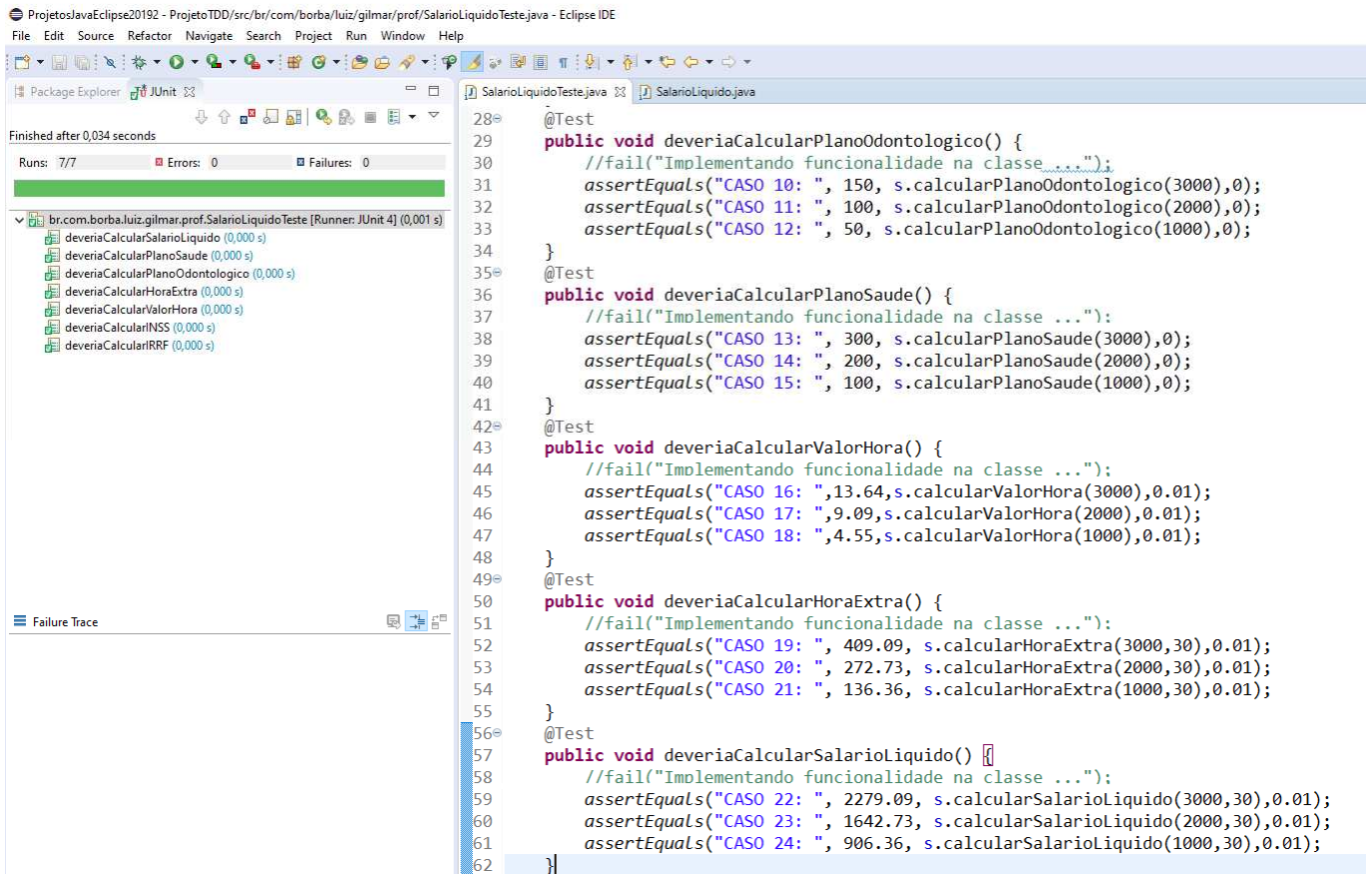
30 horas extras (valor da hora extra = 4,54)

salário líquido = 906,36

Siga os passos anteriores ...

# Testes De Software

Professor Gilmar Luiz de Borba



The screenshot shows the Eclipse IDE interface. The top menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The Package Explorer on the left shows a project structure with a package 'br.com.borba.luiiz.gilmar.prof.SalarioLiquidoTeste' containing several test classes. The JUnit runner shows 'Finished after 0,034 seconds' with 'Runs: 7/7', 'Errors: 0', and 'Failures: 0'. The main editor displays the source code for 'SalarioLiquidoTeste.java'.

```
28 @Test
29 public void deveriaCalcularPlanoOdontologico() {
30     //fail("Implementando funcionalidade na classe...");
31     assertEquals("CASO 10: ", 150, s.calcularPlanoOdontologico(3000),0);
32     assertEquals("CASO 11: ", 100, s.calcularPlanoOdontologico(2000),0);
33     assertEquals("CASO 12: ", 50, s.calcularPlanoOdontologico(1000),0);
34 }
35 @Test
36 public void deveriaCalcularPlanoSaude() {
37     //fail("Implementando funcionalidade na classe ...");
38     assertEquals("CASO 13: ", 300, s.calcularPlanoSaude(3000),0);
39     assertEquals("CASO 14: ", 200, s.calcularPlanoSaude(2000),0);
40     assertEquals("CASO 15: ", 100, s.calcularPlanoSaude(1000),0);
41 }
42 @Test
43 public void deveriaCalcularValorHora() {
44     //fail("Implementando funcionalidade na classe ...");
45     assertEquals("CASO 16: ", 13.64, s.calcularValorHora(3000),0.01);
46     assertEquals("CASO 17: ", 9.09, s.calcularValorHora(2000),0.01);
47     assertEquals("CASO 18: ", 4.55, s.calcularValorHora(1000),0.01);
48 }
49 @Test
50 public void deveriaCalcularHoraExtra() {
51     //fail("Implementando funcionalidade na classe ...");
52     assertEquals("CASO 19: ", 409.09, s.calcularHoraExtra(3000,30),0.01);
53     assertEquals("CASO 20: ", 272.73, s.calcularHoraExtra(2000,30),0.01);
54     assertEquals("CASO 21: ", 136.36, s.calcularHoraExtra(1000,30),0.01);
55 }
56 @Test
57 public void deveriaCalcularSalarioLiquido() {
58     //fail("Implementando funcionalidade na classe ...");
59     assertEquals("CASO 22: ", 2279.09, s.calcularSalarioLiquido(3000,30),0.01);
60     assertEquals("CASO 23: ", 1642.73, s.calcularSalarioLiquido(2000,30),0.01);
61     assertEquals("CASO 24: ", 906.36, s.calcularSalarioLiquido(1000,30),0.01);
62 }
```

::fim