Verificação e Validação de Software
Turma 31, Grupo 2
Arthur Foltz - 20102486
Caroline Lewandowski - 20102626
Laura Caetano - 20180670
Letícia Flores - 19180300

Geração de Casos de Teste para Teste de Unidade

O exercício consiste em aplicar as técnicas vistas em aula para geração de casos de teste para Teste Unitário, a partir de um problema que propõe a estrutura de uma classe "CentroDistribuicao", e define os requisitos e como cada método deve funcionar.

A partir do entendimento da situação proposta, foram definidas as seguintes tarefas pelo grupo:

1. Definição dos casos de teste
2. Implementação da classe driver de teste
3. Implementação da classe "CentroDistribuicao"
4. Análise dos resultados

Apresentamos neste relatório a descrição de cada uma das tarefas executadas e os resultados obtidos e o código implementado no github[1]

1. Definição dos casos de teste

Para definição dos casos de teste o grupo utilizou as seguintes técnicas: (1) Elaboração de um diagrama de estados, (2) Definição de partições, (3) Definição dos limites

Diagrama de estados:

[1] Lauramrcae/t1VeriVal (github.com)

O diagrama de estados permite visualizar melhor as partes do problema, tendo em vista que são possíveis três estados diferentes. Assim, podemos identificar visualmente em quais pontos determinado estado é alterado.

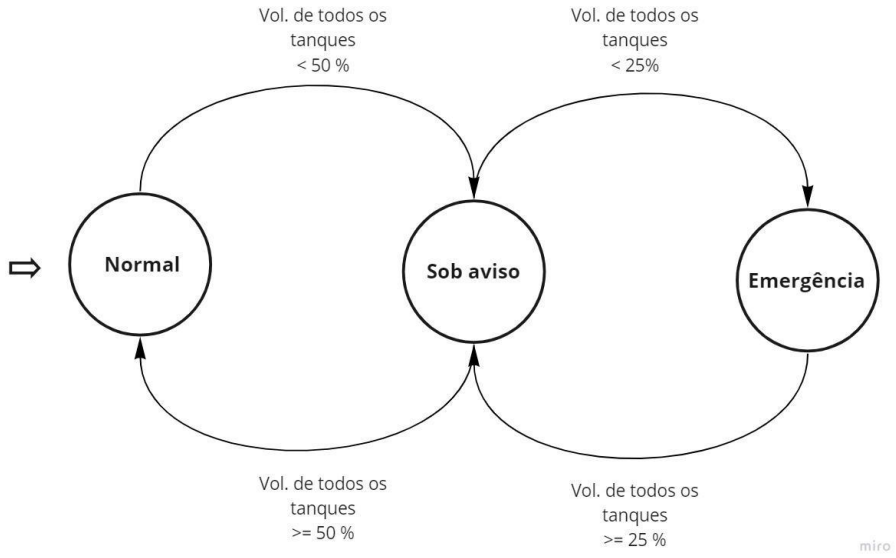


Figura 1 - Diagrama de estados

O seguinte caso de teste diz respeito aos valores correspondentes aos percentuais dos tanques, definidos na proposta da atividade, e correspondem ao volume que as misturas devem atingir para trocar entre as situações “Emergência”, “Sobreaviso” e “Normal”.

Para essa situação, primeiramente foram criadas partições para cada tanque, sendo AD correspondente ao tanque aditivo, GP para Gasolina Pura e AL para álcool:

AD1 [-inf, 0]

AD2 [1, 125]

AD3 [126,249]

AD4 [250,500]

AD5 [5001, inf+]

GP1 [-inf, 0]

GP2 [1, 2500]

GP3 [2501, 4999]

GP4 [5000, 10000]

GP5 [10001, inf+]

AL1 [-inf, 0]

AL2 [1, 312.5]

AL3 [312.6, 624]

AL4 [625, 1250]

AL5 [1251, inf+]

Levando em conta a quantidade de condicionais que a implementação da troca de estados deverá conter em sua implementação, foi decidido criar casos de teste de valor limite:

Tanque	Situação	Condição	Tipo	Teste	Resultado Esperado:
ADITIVO		≤0%	On	**T1** = 0	Erro
			Off	**T2=** 1	Emergência
	Emergência	0% > x < 25%	On	**T3** = 0 **T5** = 125	T3 = Erro **T5** = Sobraviso
			Off	**T4** = 1 **T6** = 124	T4 = Emergência **T6** = Emergência
	Sobraviso	25% ≤ x < 50%	On	**T11** = 125 **T13** = 250	**T11** = Sobraviso**T13** = Normal
			Off	**T12 =** 124 **T14** = 249	**T12** = Emergência **T14** = Sobraviso
	Normal	50% ≤ x ≤ 100%	On	**T19** = 250 **T21** = 500	**T19** = Normal **T21** = Normal
			Off	**T20 =** 249 **T22** = 501	**T20** = Sobraviso **T22** = Erro
		> 100%	On	**T23** = 500	Normal
			Off	**T24** = 501	Erro
GASOLINA PURA		≤0%	On	**T25** = 0	Erro
			Off	**T26** = 1	Emergência
	Emergência	0% > x < 25%	On	**T27** = 0 **T29** = 2500	**T27** = Erro **T29** = Sobraviso
			Off	**T28 =** 1 **T30** = 2499	**T28** = Emergência **T30** = Emergência
	Sobraviso	25% ≤ x < 50%	On	**T35** = 2500 **T37** = 5000	**T35** = Sobraviso**T38** = Normal
			Off	**T36 =** 2499 **T38** = 4999	**T36** = Emergência **T38** = Sobraviso

	Normal	50% ≥ x ≤ 100%	On	**T43** = 5000 **T45** = 10000	**T43** = Normal **T45** = Normal
			Off	**T44 =** 4999 **T46** = 10001	**T44** = Sobraviso**T46** = Erro
		> 100%	On	**T47** = 10000	Normal
			Off	**T48** = 10001	Erro
ÁLCOOL		≤0%	On	**T49** = 0	Erro
			Off	**T50** = 1	Emergência
	Emergência	0% > x < 25%	On	**T51** = 0 **T52** = 312	**T51** = Erro **T52** = Sobraviso
			Off	**T53 =** 1 **T54** = 311	**T53** = Emergência **T54** = Emergência
	Sobraviso	25% ≤ x < 50%	On	**T55** = 312 **T57** = 625	**T55** = Sobraviso**T57** = Normal
			Off	**T56 =** 311 **T58** = 624	**T56** = Emergência **T58** = Sobraviso
	Normal	50% ≥ x ≤ 100%	On	**T59** = 625 **T60** = 1250	T59 = Normal T60 = Normal
			Off	**T61 =** 624 **T62** = 1251	T61 = SobravisoT62 = Erro
		> 100%	On	**T63** = 1250	Normal
			Off	**T64** = 1251	Erro

Após a criação de testes para a situação dos tanques, segue-se para as verificações do comportamento das classes para cada situação. Considerando a espera de diferentes resultados entre postos comuns e estratégicos.

A partir dos requisitos do sistema, outras partições foram definidas para gerar combinações de valores para cada caso:

T1: Volume dos tanques maior ou igual a 50%

T2: Volume dos tanques maior que 50% e menor que 25%

T3: Volume dos tanques menor ou igual a 25%
Q1: quantidade solicitada menor que capacidade
Q2: quantidade solicitada maior que capacidade
Q3: quantidade solicitada inválida
P1: Posto do tipo comum
P2: Posto do tipo estratégico

Desta forma, foi possível gerar casos de teste baseados em especificação:

- *T1, Q1, P1; T1, Q1, P2;*

Aditivo = 500, Gasolina 10000, Álcool TA1 = 1250, Álcool TA2 = 1250, Quantidade solic. = 100, P1 = COMUM; ***(testeEncomendaPostoComumSituacaoNormalComMisturaSuficiente())***

Aditivo = 500, Gasolina 10000, Álcool TA1 = 1250, Álcool TA2 = 1250, Quantidade solic. = 100, P1 = ESTRATÉGICO; ***(testeEncomendaPostoEstrategicoSituacaoNormalComMisturaSuficiente())***

- *T2, Q1, P1; T2, Q1, P2;*

Aditivo = 244, Gasolina 4930, Álcool TA1 = 612, Álcool TA2 = 612, Quantidade solic. = 200, P1 = COMUM; ***(testeEncomendaPostoComumSituacaoSobreavisoComMisturaSuficiente())***

Aditivo = 244, Gasolina 4929, Álcool TA1 = 636, Álcool TA2 = 636, Quantidade solic. = 100, P1 = ESTRATÉGICO; ***(testeEncomendaPostoEstrategicoSituacaoSobreavisoComMisturaSuficiente())***

- *T3, Q1, P1; T3, Q1, P2;*

Aditivo = 0, Gasolina 10000, Álcool TA1 = 1250, Álcool TA2 = 1250, Quantidade solic. = 250, P1 = COMUM; ***(testeEncomendaPostoComumSituacaoEmergencia())***

Aditivo = 124, Gasolina 2599, Álcool TA1 = 324, Álcool TA2 = 324, Quantidade solic. = 200, P1 = ESTRATÉGICO; ***(testeEncomendaPostoEstrategicoSituacaoEmergenciaComMisturaSuficiente())***

- *T1, Q2, P1; T1, Q2, P2;*

Aditivo = 500, Gasolina 10000, Álcool TA1 = 1250, Álcool TA2 = 1250, Quantidade solic. = 15000, P1 = COMUM; ***(testeEncomendaPostoComumSemMisturaSuficiente())***

Aditivo = 500, Gasolina 10000, Álcool TA1 = 1250, Álcool TA2 = 1250, Quantidade solic. = 15000, P1 = ESTRATÉGICO; ***(testeEncomendaPostoEstrategicoSituacaoSobreavisoSemMisturaSuficiente())***

- *T2, Q2, P1; T2, Q2, P2;*

Aditivo = 249, Gasolina 4999, Álcool TA1 = 624, Álcool TA2 = 624, Quantidade solic. = 2000, P1 = COMUM; ***(testeEncomendaPostoComumSituacaoSobreavisoSemMisturaSuficiente())***

Aditivo = 249, Gasolina 4999, Álcool TA1 = 624, Álcool TA2 = 624, Quantidade solic. = 2000, P1 = ESTRATÉGICO; ***(testeEncomendaPostoEstrategicoSituacaoSobreavisoSemMisturaSuficiente())***

- *T3, Q2, P1; T3, Q2, P2;*

Aditivo = 129, Gasolina 2599, Álcool TA1 = 324, Álcool TA2 = 324, Quantidade solic. = 1000, P1 = COMUM; ***(testeEncomendaPostoComumSituacaoNormalSemMisturaSuficiente())***

Aditivo = 129, Gasolina 2599, Álcool TA1 = 324, Álcool TA2 = 324, Quantidade solic. = 1000, P1 = ESTRATÉGICO; ***(testeEncomendaPostoEstrategicoSituacaoEmergenciaSemMisturaSuficiente())***

- *T1, Q3, P1;*

Aditivo = 500, Gasolina 10000, Álcool TA1 = 1250, Álcool TA2 = 1250, Quantidade solic. = -10, P1 = COMUM; ***(testeEncomendaComValorInvalido())***

2. Implementação da classe driver

Foram definidas as seguintes classes de teste para a atividade:

Classes de teste:	Objetivo:
testeControllerValoresInvalidos	Testar valores inválidos
testeSituacaoNormal testeSituacaoSobraviso testeSituacaoEmergencia	Testar caso de valor limite da classe getSituacao()
testeRecebeAditivoPositivo testeRecebeGasolinaPositivo testeRecebeAlcoolPositivo testeRecebeAditivoNegativo testeRecebeGasolinanegativo testeRecebeAlcoolNegativo	Testar os métodos referente a recebimento de combustível
testeEncomendaComValorInvalido testeEncomendaPostoComumSituacaoEmergencia testeEncomendaPostoComumSituacaoNormalComMisturaSuficiente testeEncomendaPostoComumSituacaoSobreavisoComMisturaSuficiente testeEncomendaPostoComumSituacaoSobreavisoSemMisturaSuficiente testeEncomendaPostoComumSituacaoEmergenciaSemMisturaSuficiente testeEncomendaPostoComumSemMisturaSuficiente testeEncomendaPostoEstrategicoSituacaoEmergenciaComMisturaSuficiente testeEncomendaPostoEstrategicoSituacaoNormalComMisturaSuficiente testeEncomendaPostoEstrategicoSituacaoSobreavisoComMisturaSuficiente testeEncomendaPostoEstrategicoSituacaoSobreavisoSemMisturaSuficiente testeEncomendaPostoEstrategicoSituacaoEmergenciaSemMisturaSuficiente	Testar método encomendaCombustivel, considerandos variações tipo de posto, situação e quantidade de mistura

Para facilitar a entrada de valores, ao longo da implementação da classe, teremos o uso de testes parametrizados, indicado pela anotação ***@ParameterizedTest***

```java
@ParameterizedTest
@CsvSource({ "-10,10,10,10",
        "10,-10,10,10",
        "10,10,10,-10",
        "10,10,-10,10",
        "10, 10, 10, 5",
        "10, 10, 5, 10",
        "0, 10, 5, 10",
        "10, 0, 5, 10",
        "10, 10, 0, 10",
        "0, 10, 5, 0",
        "501,10,10,10",
        "10,10001,10,10",
        "10,10,1251,10",
        "10,10,10,1251"
})
public void testeControllerValoresInvalidos(int adt, int gas, int al1, int al2) {
    Assertions.assertThrows(IllegalArgumentException.class, () -> {
        new CentroDistribuicao(adt, gas, al1, al2);
    });
}
```

Figura 2 - Teste de valores inválidos

Método que testa a inserção de valores inválidos como números negativos, acima do limite dos reservatórios ou "zeros". Este método testa o instanciamento da classe "CentroDistribuicao".

Para a implementação dos testes de valor límite, que testam o método "getSituacao", foram gerados alguns métodos de teste com base no caso de teste criado, como exemplo:

```java
    @ParameterizedTest
    @CsvSource({ "500,10000,1250,1250",
            "250,5000,625,625"
    })
    public void testeSituacaoNormal(int adt, int gas, int al1, int al2) {
        cd = new CentroDistribuicao(adt, gas, al1, al2);
        SITUACAO expectedResult = cd.getSituacao();
        Assertions.assertEquals(expectedResult, SITUACAO.NORMAL);
    }
```

Figura 3 - Teste Situação Normal do tanque

Para realizar o teste das classes que realizam o recebimento de gasolina, álcool ou aditivo realizamos a seguinte estrutura de teste:

```
67      @Test
68      public void testeRecebeAditivoPositivo() {
69          cd = new CentroDistribuicao(1, 10000, 1250, 1250);
70          int result = cd.recebeAditivo(500);
71          Assertions.assertEquals(result, 499);
72      }
73
```

Figura 6 - Teste Recebe Aditivo Positivo

Para testar entradas inválidas de álcool, combustível e aditivo temos a seguinte estrutura de teste, todos eles esperam um retorno -1 pois os métodos que estão sendo testados devem retornar -1 em caso de erro. Os métodos que são testados são ***recebeGasolina***, ***recebeAlcool*** e ***recebeAditivo*** da classe "***CentroDistribuicao***".

```
88      @ParameterizedTest
89      @CsvSource({ "501",
90      "0",
91      "-2"
92      })
93      public void testeRecebeAditivoValoresInvalidos(int adt) {
94          cd = new CentroDistribuicao(1, 10000, 1250, 1250);
95          int result = cd.recebeAditivo(adt);
96          Assertions.assertEquals(result, -1);
97      }
```

Figura 9 - Teste Recebe Aditivo com valor inválido

Na classe "***CentroDistribuicao***" há um método que permite a encomenda de combustível pelos postos, porém o sucesso dessa operação depende de vários fatores, para testar o funcionamento de acordo com as especificações entregues, temos os métodos de teste abaixo:

```
121     @ParameterizedTest
122     @CsvSource({ "500,10000,1250,1250,0",
123     "500,10000,1250,1250,-10"
124     })
125     public void testeEncomendaComValorInvalido(int adt, int gas, int al1, int al2, int qtd) {
126         cd = new CentroDistribuicao(adt, gas, al1, al2);
127         int expectedResult[] = new int[4];
128         expectedResult[0] = -7;
129         Assertions.assertArrayEquals(cd.encomendaCombustivel(qtd, TIPOPOSTO.COMUM), expectedResult);
130     }
```

Figura 12 - Teste do método encomendaCombustivel com valores inválidos

3. Implementação da classe "CentroDistribuicao"

A classe "CentroDistribuicao" foi implementada considerando a estrutura apresentada no enunciado do problema.

Métodos:	Objetivo:
defineSituacao	Contêm a lógica que irá verificar a situação dos tanques do Centro de distribuição. Esse método é importante, pois para cada situação o programa deverá se comportar de maneira diferente.
getSituacao	
gettGasolina	Retornam a quantidade de cada componente nos tanques.
gettAditivo	
gettAlcool1	
gettAlcool2	
recebeAditivo	Retornam as mudanças no tanque ao adicionar, por parâmetro, certa quantidade de componente.
recebeGasolina	
recebeAlcool	
encomendaCombustivel	Principal método do programa, que recebe por parâmetro a quantidade solicitada pelos postos e o tipo de posto. O método foi implementado visando contemplar cada um dos requisitos. Inicialmente verifica se a quantidade é válida, e depois realiza uma lógica específica para cada

	tipo de posto.

4. Resultados

Defeitos encontrados:

Recebemos a implementação de outros grupo e obtivemos os seguintes resultados ao rodar nossos casos de teste:

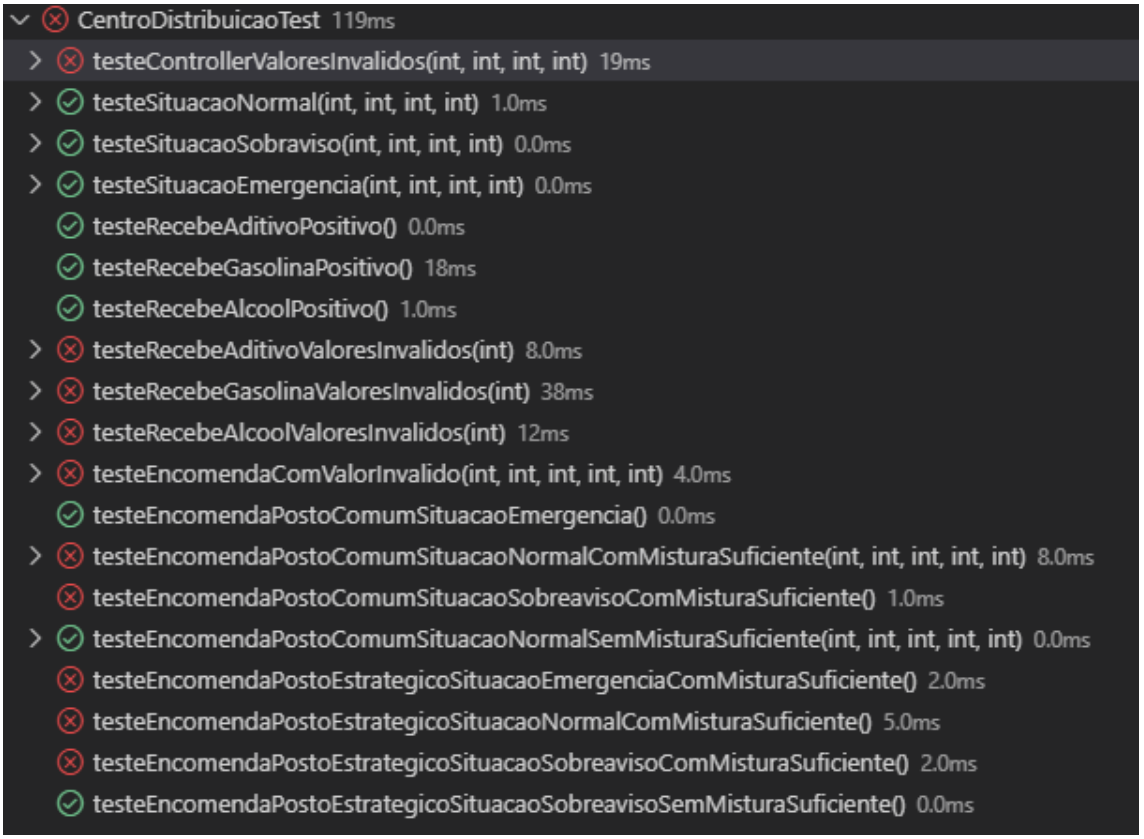


Figura 13 - Resultado dos testes aplicados

Após analisarmos os casos de erro do primeiro teste, que é da Controller, tivemos que alterar o código para quando os valores 0 ou acima do valor máx do tanque fossem recebidos o código retornasse um IllegalArgumentException.

O teste seguinte que gerou erro foi o de recebimento de gasolina, aditivo e álcool. O erro foi similar ao anterior, quando recebíamos valores 0 ou acima do max do tanque não retornava o erro -1. Após a correção deste defeito, os testes seguintes que

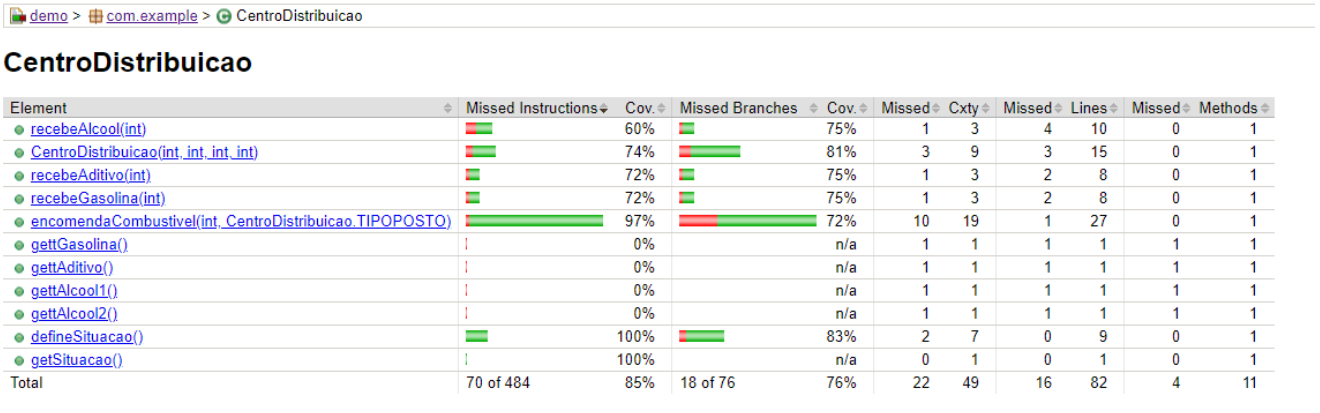
geraram erro foram para o método de encomendaCombustivel(). Os seguintes testes falharam:

- testeEncomendaComValorInvalido()
- testeEncomendaPostoComumSituacaoNormalComMisturaSuficiente
- testeEncomendaPostoComumSituacaoSobreavisoComMisturaSuficiente
- testeEncomendaPostoEstrategicoSituacaoEmergenciaComMisturaSuficiente
- testeEncomendaPostoEstrategicoSituacaoNormalComMisturaSuficiente
- testeEncomendaPostoEstrategicoSituacaoSobreavisoComMisturaSuficiente
- testeEncomendaPostoEstrategicoSituacaoSobreavisoSemMisturaSuficiente

O com valores inválidos não gerava erro quando solicitado quantidade igual a 0. Já os testes seguintes com mistura suficiente estavam retornando a quantidade abastecida e não a quantidade restante nos tanques. Após o ajuste do retorno das quantidades, foi descoberto o erro de retorno na quantidade esperada quando o posto era estratégico e em situação de emergência. Era necessário que retornasse somente 50% do valor solicitado e estava retornando 100%. Abaixo está o código antes da alteração e o código após. Após aplicar as correções, foi rodado o teste sobre as classes novamente, e todos obtiveram sucesso.

- **Resultados Code Coverage:**

Por fim, foi feita uma análise de cobertura de código na implementação de outro grupo. Onde o total de cobertura da classe centroDistribuição foi de 85%.



demo > com.example > CentroDistribuicao

CentroDistribuicao

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
recebeAlcool(int)		60%		75%	1	3	4	10	0	1
CentroDistribuicao(int, int, int, int)		74%		81%	3	9	3	15	0	1
recebeAditivo(int)		72%		75%	1	3	2	8	0	1
recebeGasolina(int)		72%		75%	1	3	2	8	0	1
encomendaCombustivel(int, CentroDistribuicao.TIPOPOSTO)		97%		72%	10	19	1	27	0	1
gettGasolina()		0%		n/a	1	1	1	1	1	1
gettAditivo()		0%		n/a	1	1	1	1	1	1
gettAlcool1()		0%		n/a	1	1	1	1	1	1
gettAlcool2()		0%		n/a	1	1	1	1	1	1
defineSituacao()		100%		83%	2	7	0	9	0	1
getSituacao()		100%		n/a	0	1	0	1	0	1
Total	70 of 484	85%	18 of 76	76%	22	49	16	82	4	11

Verificamos que alguns métodos ficaram com o percentual relativamente baixo, como por exemplo o método recebeAlcool():

```
    public int recebeAlcool(int qtdade) {
        if (qtdade <= 0 )
            return -1;
        if (qtdade + tAlcool1 + tAlcool2 > MAX_ALCOOL) {
            int aux = tAlcool1 + tAlcool2;
            tAlcool1 = MAX_ALCOOL / 2;
            tAlcool2 = MAX_ALCOOL / 2;
            return ((tAlcool1 + tAlcool2) - aux);
        }
        tAlcool1 = tAlcool1 + (qtdade / 2);
        tAlcool2 = tAlcool2 + (qtdade / 2);
        return qtdade;
    }
```

Ao investigar, identificamos que o teste implementado não testou por completo cada condição, visto que esse método possui mais de uma condição.

Neste caso, poderíamos implementar mais um teste para aumentar a cobertura ou definir que o percentual atual é aceitável.

5. Conclusão

Tendo em vista os pontos abordados ao longo do relatório, pontua-se que a definição de testes auxilia significativamente na implementação de programas, até mesmo simples como o da atividade proposta, onde sua utilização permitiu identificar rapidamente erros na implementação.

Pode-se pontuar também que combinar diferentes testes e a utilização de ferramentas como code coverage permitiram uma cobertura mais completa do código. Desta forma, a atividade proporcionou um melhor entendimento sobre a disciplina tal como a prática para elaboração e execução de testes.

ANEXOS

```
public class CentroDistribuicao {
    public enum SITUACAO { NORMAL, SOBRAVISO, EMERGENCIA }
    public enum TIPOPOSTO { COMUM, ESTRATEGICO }

    public static final int MAX_ADITIVO = 500;
    public static final int MAX_ALCOOL = 2500;
    public static final int MAX_GASOLINA = 10000;
```

```
   // tanques e situação
   private int tAditivo;
   private int tGasolina;
   private int tAlcool1;
   private int tAlcool2;
   private SITUACAO situacao;

   public CentroDistribuicao(int tAditivo, int tGasolina, int tAlcool1, int tAlcool2) throws
IllegalArgumentException{
     if( tAditivo < 0 || tGasolina < 0 || tAlcool1 < 0 || tAlcool2 < 0 || tAlcool1 != tAlcool2 )
       throw new IllegalArgumentException("ILLEGAL_ARGUMENT_EXCEPTION");
     if(tAditivo > MAX_ADITIVO) this.tAditivo = MAX_ADITIVO;
       else this.tAditivo = tAditivo;

     if(tGasolina > MAX_GASOLINA) this.tGasolina = MAX_GASOLINA;
     else this.tGasolina = tGasolina;

     if(tAlcool1 + tAlcool2 > MAX_ALCOOL){
       this.tAlcool1 = MAX_ALCOOL / 2;
       this.tAlcool2 = MAX_ALCOOL / 2;
     } else{
       this.tAlcool1 = tAlcool1;
       this.tAlcool2 = tAlcool2;
     }

     defineSituacao();
   }

   public void defineSituacao(){
     if(tGasolina >= (MAX_GASOLINA/2) && tAditivo >= (MAX_ADITIVO/2)
&& (tAlcool1+tAlcool2) >= (MAX_ALCOOL/2)){
       situacao = SITUACAO.NORMAL;
     }
     else if(tGasolina >= (MAX_GASOLINA/4) && tAditivo >= (MAX_ADITIVO/4)
&& (tAlcool1+tAlcool2) >= (MAX_ALCOOL/4)){
       situacao = SITUACAO.SOBRAVISO;
     }
     else situacao = SITUACAO.EMERGENCIA;
   }

   public SITUACAO getSituacao(){
     return situacao;
   }

   public int gettGasolina(){
     return tGasolina;
   }

   public int gettAditivo(){
     return tAditivo;
   }

   public int gettAlcool1(){
     return tAlcool1;
   }
```

```
public int gettAlcool2(){
  return tAlcool2;
}

public int recebeAditivo(int qtdade) {
  if(qtdade < 0) return -1;
  if(qtdade+tAditivo > MAX_ADITIVO){
    int aux = tAditivo;
    tAditivo = MAX_ADITIVO;
    return (tAditivo - aux);
  }
  tAditivo+= qtdade;
  return qtdade;
}

public int recebeGasolina(int qtdade) {
  if(qtdade < 0) return -1;
  if(qtdade+tGasolina > MAX_GASOLINA){
    int aux = tGasolina;
    tGasolina = MAX_GASOLINA;
    return (tGasolina - aux);
  }
  tGasolina+= qtdade;
  return qtdade;
}

public int recebeAlcool(int qtdade) {
  if(qtdade < 0) return -1;
  if(qtdade+tAlcool1+tAlcool2 > MAX_ALCOOL){
    int aux = tAlcool1 + tAlcool2;
    tAlcool1 = MAX_ALCOOL / 2;
    tAlcool2 = MAX_ALCOOL / 2;
    return ((tAlcool1+tAlcool2) - aux);
  }
  tAlcool1 = tAlcool1 + (qtdade/2);
  tAlcool2 = tAlcool2 + (qtdade/2);
  return qtdade;
}

public int[] encomendaCombustivel(int qtdade, TIPOPOSTO tipoPosto) {
  if(qtdade < 0) return new int[] {-7,0,0,0};

  double gasolina = qtdade * 0.7;
  double aditivo = qtdade * 0.05;
  double alcool1 = qtdade * 0.25 * 0.5;
  double alcool2 = qtdade * 0.25 * 0.5;
```

```
//Se a situacao for normal ou a situacao for sobreaviso e o posto estrategico(tem o mesmo
comportamento)
    if(getSituacao() == SITUACAO.NORMAL || (getSituacao() == SITUACAO.SOBRAVISO &&
tipoPosto == TIPOPOSTO.ESTRATEGICO)) {
      if( tGasolina >= gasolina && tAditivo >= aditivo && tAlcool1 >= alcool1 && tAlcool2 >=
alcool2 ){
        tGasolina-= gasolina;
        tAditivo-= aditivo;
        tAlcool1-= alcool1;
        tAlcool2-= alcool2;
        return new int[] {(int)aditivo, (int)gasolina, (int)alcool1, (int)alcool2};
      } else return new int[] {-21,0,0,0};
    }

    if(getSituacao() == SITUACAO.SOBRAVISO){
     if( tGasolina >= gasolina && tAditivo >= aditivo && tAlcool1 >= alcool1 && tAlcool2 >=
alcool2 ){
       if(tipoPosto == TIPOPOSTO.COMUM){
         tGasolina-= (gasolina/2);
         tAditivo-= (aditivo/2);
         tAlcool1-= (alcool1/2);
         tAlcool2-= (alcool2/2);
         return new int[] {(int)(aditivo/2), (int)(gasolina/2), (int)(alcool1/2), (int)(alcool2/2)};
       }
     } else return new int[] {-21,0,0,0};
    }

    if(tipoPosto == TIPOPOSTO.COMUM) return new int[] {-14,0,0,0};

    if( tGasolina >= gasolina && tAditivo >= aditivo && tAlcool1 >= alcool1 && tAlcool2 >=
alcool2 ){
      tGasolina-= gasolina;
      tAditivo-= aditivo;
      tAlcool1-= alcool1;
      tAlcool2-= alcool2;
      return new int[] {(int)aditivo,(int)gasolina,(int)alcool1,(int)alcool2};
    }

    return new int[] {-21,0,0,0};
  }
}
```

Classe Driver:

```
package
com.examp
le;
```

```
import com.example.CentroDistribuicao.SITUACAO;
import com.example.CentroDistribuicao.TIPOPOSTO;


import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;


public class CentroDistribuicaoTest {


  private CentroDistribuicao cd = null;


  @ParameterizedTest
  @CsvSource({ "-10,10,10,10",
        "10,-10,10,10",
        "10,10,10,-10",
        "10,10,-10,10",
        "10, 10, 10, 5",
        "10, 10, 5, 10",
        "0, 10, 5, 10",
        "10, 0, 5, 10",
        "10, 10, 0, 10",
        "0, 10, 5, 0",
        "501,10,10,10",
        "10,10001,10,10",
        "10,10,1251,10",
        "10,10,10,1251"
  })
  public void testeControllerValoresInvalidos(int adt, int gas, int al1, int al2) {
     Assertions.assertThrows(IllegalArgumentException.class, () -> {
        new CentroDistribuicao(adt, gas, al1, al2);
     });
  }


  @ParameterizedTest
  @CsvSource({ "500,10000,1250,1250",
        "250,5000,625,625"
  })
  public void testeSituacaoNormal(int adt, int gas, int al1, int al2) {
     cd = new CentroDistribuicao(adt, gas, al1, al2);
     SITUACAO expectedResult = cd.getSituacao();
```

```
    Assertions.assertEquals(expectedResult, SITUACAO.NORMAL);
}


@ParameterizedTest
@CsvSource({ "249,4999,624,624",
        "125,2500,313,313"
})
public void testeSituacaoSobraviso(int adt, int gas, int al1, int al2) {
    cd = new CentroDistribuicao(adt, gas, al1, al2);
    SITUACAO expectedResult = cd.getSituacao();
    Assertions.assertEquals(expectedResult, SITUACAO.SOBRAVISO);
}


@ParameterizedTest
@CsvSource({ "124,2499,312,312",
        "1,2,4,4"
})
public void testeSituacaoEmergencia(int adt, int gas, int al1, int al2) {
    cd = new CentroDistribuicao(adt, gas, al1, al2);
    SITUACAO expectedResult = cd.getSituacao();
    Assertions.assertEquals(expectedResult, SITUACAO.EMERGENCIA);
}


@Test
public void testeRecebeAditivoPositivo() {
    cd = new CentroDistribuicao(1, 10000, 1250, 1250);
    int result = cd.recebeAditivo(500);
    Assertions.assertEquals(result, 499);
}


@Test
public void testeRecebeGasolinaPositivo() {
    cd = new CentroDistribuicao(1, 1, 1250, 1250);
    int result = cd.recebeGasolina(10000);
    Assertions.assertEquals(result, 9999);
}


@Test
public void testeRecebeAlcoolPositivo() {
    cd = new CentroDistribuicao(1, 1, 1, 1);
    int result = cd.recebeAlcool(900);
```

```
    Assertions.assertEquals(result, 900);
  }


  @ParameterizedTest
  @CsvSource({ "501",
  "0",
  "-2"
  })
  public void testeRecebeAditivoValoresInvalidos(int adt) {
    cd = new CentroDistribuicao(1, 10000, 1250, 1250);
    int result = cd.recebeAditivo(adt);
    Assertions.assertEquals(result, -1);
  }


  @ParameterizedTest
  @CsvSource({ "10001",
  "0",
  "-2"
  })
  public void testeRecebeGasolinaValoresInvalidos(int gas) {
    cd = new CentroDistribuicao(1, 1,1, 1);
    int result = cd.recebeGasolina(gas);
    Assertions.assertEquals(result, -1);
  }


  @ParameterizedTest
  @CsvSource({ "2501",
  "0",
  "-2"
  })
  public void testeRecebeAlcoolValoresInvalidos(int al) {
    cd = new CentroDistribuicao(1, 1, 1, 1);
    int result = cd.recebeAlcool(al);
    Assertions.assertEquals(result, -1);
  }


  @ParameterizedTest
  @CsvSource({ "500,10000,1250,1250,0",
  "500,10000,1250,1250,-10"
  })
  public void testeEncomendaComValorInvalido(int adt, int gas, int al1, int al2, int
qtd) {
```

```
        cd = new CentroDistribuicao(adt, gas, al1, al2);
        int expectedResult[] = new int[4];
        expectedResult[0] = -7;
        Assertions.assertArrayEquals(cd.encomendaCombustivel(qtd,
TIPOPOSTO.COMUM), expectedResult);
    }


    @Test
    public void testeEncomendaPostoComumSituacaoEmergencia() {
        cd = new CentroDistribuicao(1, 10000, 1250, 1250);
        int expectedResult[] = new int[4];
        expectedResult[0] = -14;
        Assertions.assertArrayEquals(cd.encomendaCombustivel(250,
TIPOPOSTO.COMUM), expectedResult);
    }


    @ParameterizedTest
    @CsvSource({ "500,10000,1250,1250,100"
    })
    public void
testeEncomendaPostoComumSituacaoNormalComMisturaSuficiente(int adt, int
gas, int al1, int al2, int qtd) {
        cd = new CentroDistribuicao(adt, gas, al1, al2);
        double qtdAd = adt - qtd * 0.05;
        double qtdG = gas - qtd * 0.7;
        double qtdAl = al1 - (qtd / 2) * 0.25;


        double expectedResult[] = new double[] { qtdAd, qtdG, qtdAl, qtdAl };
        int[] result = new int[4];
        result = cd.encomendaCombustivel(qtd, TIPOPOSTO.COMUM);
        double[] resultConverted = new double[4];
        for (int i = 0; i < 4; i++) {
            resultConverted[i] = result[i];
        }
        Assertions.assertArrayEquals(resultConverted, expectedResult, 1);
    }


    @Test
    public void
testeEncomendaPostoComumSituacaoSobreavisoComMisturaSuficiente() {
        cd = new CentroDistribuicao(249, 5000, 625, 625);


        double expectedResult[] = new double[] { 244, 4930, 613, 613 };
```

```
        int[] result = new int[4];
        result = cd.encomendaCombustivel(200, TIPOPOSTO.COMUM);
        double[] resultConverted = new double[4];
        for (int i = 0; i < 4; i++) {
            resultConverted[i] = result[i];
        }
        Assertions.assertArrayEquals(resultConverted, expectedResult, 1);
    }


    @ParameterizedTest
    @CsvSource({ "250,5000,625,625,7500",
            "230,2500,330,330,7200"
    })
    public void
testeEncomendaPostoComumSituacaoNormalSemMisturaSuficiente(int adt, int
gas, int al1, int al2, int qtd) {
        cd = new CentroDistribuicao(adt, gas, al1, al2);
        int expectedResult[] = new int[4];
        expectedResult[0] = -21;
        Assertions.assertArrayEquals(cd.encomendaCombustivel(qtd,
TIPOPOSTO.COMUM), expectedResult);
    }


    @Test
    public void
testeEncomendaPostoEstrategicoSituacaoEmergenciaComMisturaSuficiente() {
        cd = new CentroDistribuicao(124, 2599, 324, 324);
        int expectedResult[] = new int[] { 119, 2529, 311, 311 };
        Assertions.assertArrayEquals(cd.encomendaCombustivel(200,
TIPOPOSTO.ESTRATEGICO), expectedResult);
    }


    @Test
    public void
testeEncomendaPostoEstrategicoSituacaoNormalComMisturaSuficiente() {
        cd = new CentroDistribuicao(500, 10000, 1250, 1250);
        int expectedResult[] = new int[] { 495, 9930, 1237, 1237 };
        Assertions.assertArrayEquals(cd.encomendaCombustivel(100,
TIPOPOSTO.ESTRATEGICO), expectedResult);
    }


    @Test
    public void
testeEncomendaPostoEstrategicoSituacaoSobreavisoComMisturaSuficiente() {
        cd = new CentroDistribuicao(249, 4999, 649, 649);
```

```
        int expectedResult[] = new int[] { 244, 4929, 636, 636 };
        Assertions.assertArrayEquals(cd.encomendaCombustivel(100,
TIPOPOSTO.ESTRATEGICO), expectedResult);
    }


    @Test
    public void
testeEncomendaPostoEstrategicoSituacaoSobreavisoSemMisturaSuficiente() {
        cd = new CentroDistribuicao(249, 4999, 649, 649);
        int expectedResult[] = new int[4];
        expectedResult[0] = -21;
        Assertions.assertArrayEquals(cd.encomendaCombustivel(20000,
TIPOPOSTO.ESTRATEGICO), expectedResult);
    }


}
```