**Verificação e Validação de Software I**

**Prof. Bernardo Copstein**

**Técnicas de Geração de Casos de Teste**

**Exercício I**

## Enunciado do problema

Em uma diminuta ilha tropical a gasolina vendida nos postos é resultado de uma mistura de 3 componentes: 5% de aditivo, 25% de álcool e 70% de gasolina pura. A preparação do produto é feita por encomenda dos postos em um único centro de distribuição. Esse centro possui estoques adequados ao número de veículos da ilha. A figura 1 apresenta os tanques disponíveis nesse centro.

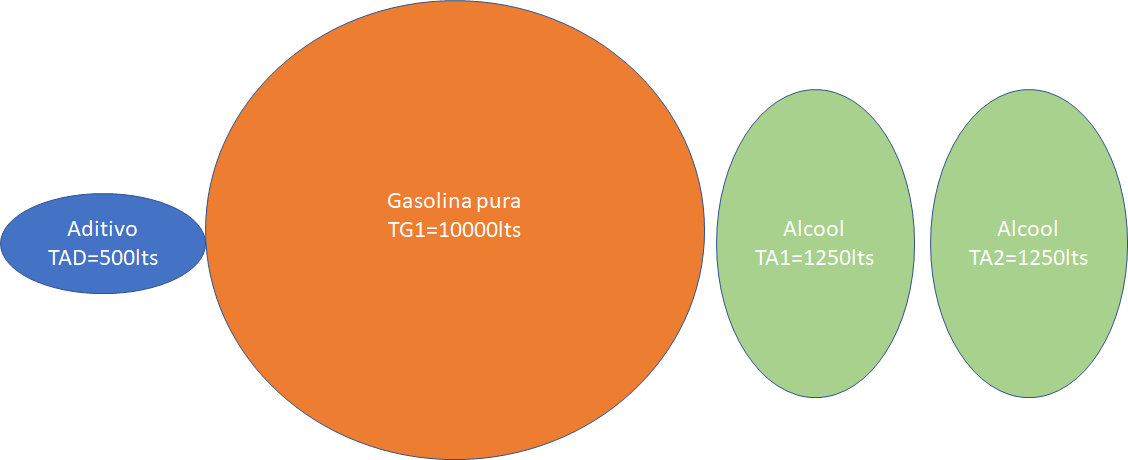


Figura 1 – Tanques do centro de distribuição de combustível.

A produção da mistura que será entregue nos postos deve-se obedecer aos percentuais já definidos e considerar as seguintes regras:

1. Se houver falta de qualquer um dos componentes na quantidade adequada a encomenda não pode ser entregue.
2. Os tanques de álcool devem ter sempre a mesma quantidade de combustível de maneira a manter o equilíbrio da estrutura devido a forma como foram construídos. Isso vale tanto para o armazenamento como para a retirada.
3. Deve-se manter uma reserva técnica de 25% da capacidade de cada tanque. Tal reserva só pode ser gasta em caso de “emergência nacional”. Isso significa que, em situação normal, um tanque com apenas 25% de combustível deve ser considerado vazio.

## Estrutura da classe

A estrutura da classe “DepComb” pode ser vista na figura 2.

|  |
| --- |
| package com.bcopstein;  public class DepComb {  public static final int MAX\_ADITIVO = 500;  public static final int MAX\_ALCOOL = 2500;  public static final int MAX\_GASOLINA = 10000;  private int tGasolina;  private int tAditivo;  private int tAlcool1;  private int tAlcool2;  public DepComb(int tGasolina,int tAditivo,int tAlcool1,int tAlcool2){  this.tGasolina = tGasolina;  this.tAditivo = tAditivo;  this.tAlcool1 = tAlcool1;  this.tAlcool2 = tAlcool2;  }  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){**  **return 0;**  **}**  **public int recebeGasolina(int qtdade){**  **return 0;**  **}**  **public int recebeAlcool(int qtdade){**  **return 0;**  **}**  **public int[] encomendaCombustivel(int qtdade,boolean emerg){**  **return null;**  **}**    @Override  public String toString() {  return "DepComb [tAditivo=" + tAditivo +  ", tAlcool1=" + tAlcool1 +  ", tAlcool2=" + tAlcool2 +  ", tGasolina="+ tGasolina + "]";  }  } |

Figura 2 – Estrutura da classe “DepComb”

Os métodos “recebeAditivo”, “recebeGasolina” e “recebeAlcool” são usados quando o centro de distribuição recebe carga dos componentes. Todos recebem por parâmetro a quantidade de combustível recebida e retornam à quantidade que puderam armazenar devido a limitação do tamanho dos tanques e de quanto ainda tinham armazenado. Devem retornar “-1” caso a quantidade recebida por parâmetro seja inválida. O método “encomendaCombustivel” é usado quando o centro de distribuição recebe o pedido de um posto. Este método recebe a quantidade solicitada pelo posto e um booleano que indica se é um pedido de emergência nacional (true) ou não (false). Se o pedido puder ser atendido, o método retorna um arranjo com a quantidade de combustível remanescente em cada tanque, depois do pedido atendido. As quantidades devem ser retornadas pela ordem: aditivo, gasolina, álcool T1 e álcool T2. No caso de ser recebido um valor inválido por parâmetro deve-se retornar -2 na primeira posição do arranjo. Se o pedido não puder ser atendido retorna-se -1 na posição correspondente ao elemento que tem quantidade insuficiente e 0 nas demais (no caso do álcool deve retornar -1 na posição do tanque 1). Por simplicidade trabalha-se apenas com números inteiros. Os cálculos devem ser feitos com números reais e convertidos para inteiro após a última operação.

## Tarefas da primeira etapa

As tarefas que seguem referem-se aos métodos destacados em negrito. Ao final deverá ser entregue um relatório contendo o relato da execução dos passos que seguem. O relatório deve ser elaborado no arquivo “readme.md”, mesmo padrão usado no GitHub. A classe alvo e a classe driver devem ser entregues como anexo do relatório.

1. Projetar os casos de teste. Apresentar uma tabela com os casos de teste e resultados esperados para cada teste indicando as técnicas utilizadas para a geração dos casos de teste. Detalhar a aplicação das estratégias (ex: partições definidas, onpoint, offpoint etc).
2. Implementar a classe driver.
3. Implementar a classe alvo
4. Executar os testes. Anotar os defeitos detectados (se houver); corrigir os defeitos e repetir o processo até que não restem mais defeitos.
5. Completar os casos de teste usando a técnica de teste estrutural e a ferramenta ....
6. Repetir o processo de execução e correção incluindo os novos casos de teste se houverem.
7. Escrever o relatório final

## Tarefas da segunda etapa

A segunda etapa consiste em trocar seu driver de teste com o de outro grupo. O objetivo é identificar as dificuldades em se usar um driver de teste desenvolvido por outra equipe e a capacidade dos dois drivers em encontrar falhas no código alheio. Toda essa experiência deve ser acrescentada no relatório final.

## Cronograma:

|  |  |
| --- | --- |
| Aula | Atividade |
| 14/04 | Apresentação do enunciado do trabalho, ajuste dos grupos etc |
| 16/04 | Acompanhamento da evolução do trabalho |
| 21/04 | Troca dos drivers de teste |
| 23/04 | Entrega dos relatórios/discussão dos resultados |