Técnica baseada na especificação

**Método**: int[] encomendaCombustivel (int quantidade, boolean emergencia)

Se emergencia = **TRUE**

A gasolina só é vendida se:

* 5% da quantidade subtraído do valor do tanque de aditivo, o restante do tanque for >= 0
* 70% da quantidade subtraído do valor do tanque de gasolina, o restante do tanque for >= 0
* 25% da quantidade subtraído dos valores dos tanques de álcool somados, o restante for >= 0

Se emergencia = **FALSE**

A gasolina só é vendida se:

* 5% da quantidade subtraído do valor do tanque de aditivo, o restante do tanque for >= 125
* 70% da quantidade subtraído do valor do tanque de gasolina, o restante do tanque for >= 2500
* 25% da quantidade subtraído dos valores dos tanques de álcool somados, o restante for >= 625

**Classes de equivalência:**

Vamos considerar que todos os tanques estejam cheios (aditivo = **500lts**; gasolina = **10000lts**; alcool1 = **1250lts**; alcool2 = **1250lts**) e que estamos somando os valores dos dois tanques de álcool.

|  |  |  |
| --- | --- | --- |
| Emergência | Quantidade | Resultado esperado |
| TRUE | 5000 | [250, 6500, 625, 625] |
| TRUE | 15000 | [-1, -1, -1] |
| FALSE | 1500 | [425, 8950, 1062, 1062] |
| FALSE | 8250 | [-1, 0, -1] |
| FALSE | -1 | [-2] |

Já para criar as classes de equivalência dos métodos que enchem os tanques, segundo a especificação do problema, existe um caso onde será abastecido o quanto der e outro caso no qual se passa uma quantidade inválida por parâmetro.

**Método**: int recebeAditivo (int quantidade)

Quantidade prévia do tanque de aditivo: **200lts**

|  |  |
| --- | --- |
| Quantidade | Resultado esperado |
| 500 | 300 |
| -1 | -1 |

**Método**: int recebeGasolina (int quantidade)

Quantidade prévia do tanque de gasolina: **7000lts**

|  |  |
| --- | --- |
| Quantidade | Resultado esperado |
| 10000 | 3000 |
| -10 | -1 |

**Método**: int recebeAlcool (int quantidade)

Quantidade prévia de cada tanque de álcool: **50lts**

|  |  |
| --- | --- |
| Quantidade | Resultado esperado |
| 2500 | 2400 |
| -100 | -1 |

Técnica do valor limite

**Método**: encomendaCombustivel (int quantidade, boolean emergencia)

**Matriz para valor limite:**

Vamos considerar que todos os tanques estejam pela metade (aditivo = **250lts**; gasolina = **5000lts**; alcool1 = **625lts**; alcool2 = **625lts**).

Se estivermos tratando de **in-points**, vamos usar dois casos:

- os respectivos tanques vão estar 100% cheios;

- os respectivos tanques vão estar 90% cheios.

O único parâmetro avaliado na matriz é a quantidade, além disso continuamos somando os valores dos dois tanques de álcool.

Se emergencia = **TRUE**

A gasolina só é vendida se:

t.aditivo >= 0 && t.gasolina >= 0 && t.alcool >= 0

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Condition | Type | t1 | t2 | t3 | t4 | t5 | t6 |
| t.aditivo - (e\*0.05)  >= 0 | on | 5000 |  |  |  |  |  |
| off |  | 5001 |  |  |  |  |
| 100% cheio = 500lts | in |  |  |  |  |  |  |
| t.gasolina - (e\*0.7)  >= 0 | on |  |  | 7142 |  |  |  |
| off |  |  |  | 7143 |  |  |
| 100% cheio = 10000lts | in |  |  |  |  |  |  |
| t.alcool - (e\*0.25)  >= 0 | on |  |  |  |  | 5000 |  |
| off |  |  |  |  |  | 5001 |
| 100% cheio = 2500lts | in |  |  |  |  |  |  |

t1: entrada {5000, TRUE}, saída {0, 6500, 625, 625}

t2: entrada {5001, TRUE}, saída {-1, 0, 0}

t3: entrada {7142, TRUE}, saída {142, 0, 357, 357}

t4: entrada {7143, TRUE}, saída {0, -1, 0}

t5: entrada {5000, TRUE}, saída {250, 6500, 0, 0}

t6: entrada {5001, TRUE}, saída {0, 0, -1}

Se emergencia = **FALSE**

A gasolina só é vendida se:

t.aditivo >=125 && t.gasolina >=2500 && t.alcool >= 625

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Condition | Type | t1 | t2 | t3 | t4 | t5 | t6 |
| t.aditivo - (e\*0.05)  >= 125 | on | 2500 |  |  |  |  |  |
| off |  | 2501 |  |  |  |  |
| 90% cheio = 450lts | in |  |  |  |  |  |  |
| t.gasolina - (e\*0.7)  >= 2500 | on |  |  | 3571 |  |  |  |
| off |  |  |  | 3572 |  |  |
| 90% cheio = 9000lts | in |  |  |  |  |  |  |
| t.alcool - (e\*0.25)  >= 625 | on |  |  |  |  | 2500 |  |
| off |  |  |  |  |  | 2501 |
| 90% cheio = 2250lts | in |  |  |  |  |  |  |

t1: entrada {2500, FALSE}, saída {125, 7250, 812, 812}

t2: entrada {2501, FALSE}, saída {-1, 0, 0}

t3: entrada {3571, FALSE}, saída {271, 2500, 678, 678}

t4: entrada {3572, FALSE}, saída {0, -1, 0}

t5: entrada {2500, FALSE}, saída {325, 7250, 312, 312}

t6: entrada {2501, FALSE}, saída {0, 0, -1}

Teste estrutural

Devemos garantir que nossos casos de testes baseados na especificação cubram todas as linhas dos nossos métodos.

Havia uma condição em cada método de encher um tanque na qual nenhum dos casos acima passou por. Apenas conseguimos atingir essas condições quando passamos por parâmetro uma quantidade, que quando somada com a atual do tanque, é insuficiente para enchê-lo.

**Método**: int recebeAditivo (int quantidade)

Quantidade prévia do tanque de aditivo: **200lts**

|  |  |
| --- | --- |
| Quantidade | Resultado esperado |
| 100 | 100 |

**Método**: int recebeGasolina (int quantidade)

Quantidade prévia do tanque de gasolina: **7000lts**

|  |  |
| --- | --- |
| Quantidade | Resultado esperado |
| 2950 | 2950 |

**Método**: int recebeAlcool (int quantidade)

Quantidade prévia de cada tanque de álcool: **50lts**

|  |  |
| --- | --- |
| Quantidade | Resultado esperado |
| 50 | 50 |

Relatorio

Em uma linguagem orientada a objetos, além de usarmos os parâmetros para explorar classes de equivalência, devemos considerar também o estado dos atributos do objeto.

Pensando nisso, resolvemos usar valores diferentes no estado dos tanques antes de rodar cada técnica de teste.

data= 3.5;

inteiro = (int)Math.round(data); //4

os valores da tabela são assim pq tamo usando esse metodo

Já para criar as classes de equivalência nos métodos que enchem os tanques devemos analisar a quantidade que cada tanque já tinha para retornar o quanto foi possível ser abastecido. Segundo a especificação do problema, existe um caso onde será abastecido o quanto der e outro caso em que se passa uma quantidade inválida por parâmetro.

como temos três condições no método de vender gasolina, o numero de combinações de teste pode crescer rapidamente, logo devemos usar a estratégia do domínio simplicado onde utilizamos uma matriz para facilitar a criação de casos de teste.

A fim de seguir a estratégia proposta decidimos variar os in-points nos diferentes testes(quando é true ou false na emergencia). Isso permite verificar se o programa fornece resultados corretos para vários in-points.

Já para os métodos que enchem os tanques, foi deduzido que não haveria testes de valor limite já que as rotinas sempre tentarão encher até o máximo permitido dos tanques, diferentemente de casos onde se dá para traduzir in e off points como true e false em certas condições.

Como o teste estrutural pode chegar a testar todas as combinações de decisões de um determinado script, devemos ser muito criteriosos pois a quantidade de casos de teste cresce exponencialmente.

Pensando nisso, decidimos primeiramente garantir que os casos já criados com base na especificação cubram todas as linhas do nosso código. Após isso devemos focar nossa atenção nos pontos mais críticos, analisando suas condições.

Tbm pq não tínhamos loops

Usar ferramenta de analise de cobertura