FrutaFeia

January 3, 2018

Contents

1 Agricultor

```
class Agricultor

types
 public Nome = seq of (char);

instance variables
  public stock: map Produto`NomeProduto to Produto := { |-> };
  public nome:Nome;
  public localizacao:seq of (char);
```

```
operations
  public  Agricultor : Nome * seq of (char) ==> Agricultor
   Agricultor(n, l) == ( nome := n; localizacao := l; stock := {|->}; return self)
  pre len n > 0 and len l > 0
  post nome = n and localizacao = l and stock = {|->};


 --adiciona um produto ao agricultor
  public  adicionaProduto : Produto ==> ()
   adicionaProduto(produto) == stock := stock munion {produto.nome |-> produto}
  pre produto <> undefined and produto.nome not in set dom stock

  post  stock = stock~ munion {produto.nome |-> produto};

 --remove um produto do agricultor
  public removeProduto : Produto`NomeProduto ==> ()
   removeProduto(nomeProduto) == stock := {nomeProduto} <-: stock

  pre len nomeProduto > 0 and nomeProduto in set dom stock
  post nomeProduto not in set dom stock;

  --adicona peso a um produto no agricultor se existir
  public  adicionaPesoProduto : Produto`NomeProduto * real ==> bool
  adicionaPesoProduto(nomeProduto, peso) ==
   (
    if ( nomeProduto in set dom stock )
     then  ( stock(nomeProduto).adicionaPeso(peso); return true;);

    return false
   )
  pre len nomeProduto > 0 and peso > 0
  post (nomeProduto in set dom stock and RESULT=true ) or (nomeProduto not in set dom stock and
      RESULT = false);


  -- remove peso de um produto se existir
  public  removePesoProduto : Produto`NomeProduto * real ==> bool
  removePesoProduto(nomeProduto, peso) ==
    (
    if ( nomeProduto in set dom stock )
     then  ( stock(nomeProduto).removePeso(peso); return true;);
    return false
   )
  pre len nomeProduto > 0 and peso > 0
  post (nomeProduto in set dom stock and RESULT=true ) or (nomeProduto not in set dom stock and
      RESULT = false);


end Agricultor
```

Function or operation	Line	Coverage	Calls
Agricultor	11	100.0%	14
adicionaPesoProduto	26	100.0%	4
adicionaProduto	16	100.0%	34
removePesoProduto	35	100.0%	2
removeProduto	21	100.0%	2
Agricultor.vdmpp		100.0%	56

2 CentroDistribuicao

```
class CentroDistribuicao

types
 public Local = seq of (char);

instance variables
 public clientes : set of Cliente;
  public localizacao: Local;

operations


  public  CentroDistribuicao : Local ==> CentroDistribuicao
  CentroDistribuicao(local) == (clientes := {}; localizacao := local; return self;)
  pre len local > 0
  post localizacao = local and clientes = {};


 --Adiciona um cliente ao centro
 public adicionaCliente: Cliente ==> ()
 adicionaCliente( cliente ) ==
   clientes := clientes union {cliente}
 pre cliente <> undefined and cliente not in set clientes
 post card clientes~ + 1 = card clientes and cliente in set clientes;



 --Remove um cliente do centro
 public removeCliente: Cliente`Nome ==> ()
 removeCliente(clienteNome) ==
 (
  for all cliente in set clientes
   do
    if(cliente.nome = clienteNome)
     then ( clientes := clientes \ {cliente}; return;)
 )
 pre exists1 cliente in set clientes & cliente.nome = clienteNome
 post card clientes~ - 1 = card clientes and forall cliente in set clientes & cliente.nome <>
     clienteNome;

end CentroDistribuicao
```

Function or operation	Line	Coverage	Calls
CentroDistribuicao	12	100.0%	97
adicionaCliente	17	100.0%	71
removeCliente	23	88.5%	4
CentroDistribuicao.vdmpp		94.7%	172

3 Cesta

```
class Cesta
```

```
types
 public Tamanho = <PEQUENA> | <GRANDE>

instance variables
 public produtos:set of Produto;
  public peso: real;
 public tamanho: Tamanho;

operations

  public  Cesta : () ==> Cesta
  Cesta() == (peso := 0; produtos := {}; tamanho:= <PEQUENA>; return self;)
  post peso = 0 and produtos = {} and tamanho = <PEQUENA>;



  --adiciona um produto ao agricultor
  public  adicionaProduto : Produto ==> ()
  adicionaProduto(produto) ==
  (
   produtos := produtos union {produto};
    peso := peso + produto.peso
  )
  pre produto <> undefined and produto not in set produtos

  post peso = produto.peso + peso~ and (card produtos = card produtos~ + 1) and produto in set
      produtos;


 --remove um produto da cesta
  public  removeProduto : Produto ==> ()
  removeProduto(produto) ==
  (
   produtos := produtos \ {produto};
   peso := peso - produto.peso

  )
  pre produto <> undefined and produto in set produtos and peso - produto.peso >= 0
  post peso = peso~ - produto.peso and (card produtos + 1  = card produtos~ ) and produto not in
      set produtos;


 -- adiciona um determinado peso a um produto
  public  adicionaPesoProduto : Produto`NomeProduto * Produto`Peso ==> bool
  adicionaPesoProduto(nomeProduto, p) ==
 ( for all elem in set produtos

   do
   if nomeProduto = elem.nome
    then (elem.adicionaPeso(p); peso:= peso + p; return true;);
 return false;)
 pre len nomeProduto > 0 and p >= 0
 post (exists1 produto in set produtos & produto.nome = nomeProduto and RESULT = true) or --
     produto exite retorna true ou nao existe e retorna false
     (forall produto in set produtos & produto.nome <> nomeProduto and RESULT = false);

 --remove um determinado peso de um produto

 public  removePesoProduto : Produto`NomeProduto * Produto`Peso ==> bool
  removePesoProduto(nomeProduto, p) ==
  ( for all elem in set produtos
    do
     if nomeProduto = elem.nome
      then (elem.removePeso(p); return true;);
```

```
  return false)
  pre len nomeProduto > 0 and p >= 0 and peso - p >= 0
  post (exists1 produto in set produtos & produto.nome = nomeProduto and RESULT = true) or --
      produto exite retorna true ou nao existe e retorna false
     (forall produto in set produtos & produto.nome <> nomeProduto and RESULT = false);

  -- altera o tamanho da cesta  para que existem em Cesta`Tamanho
  public alterarTamanho: Tamanho ==> ()
  alterarTamanho(tam)  ==
    tamanho := tam
  pre tam <> undefined
  post tamanho = tam;

  --Verifica se o produto com um certo nome esta no set
  public produtoNaCesta : Produto`NomeProduto ==> bool
  produtoNaCesta (prodNome) ==
   (for all produto in set produtos
    do
     if prodNome = produto.nome
       then return true;
    return false
   )
   pre produtos <> undefined
   post ((exists1 produto in set produtos & produto.nome = prodNome)  and   RESULT = true) or  --
         produto exite retorna true ou nao existe e retorna false
      ((forall produto in set produtos & produto.nome <> prodNome) and RESULT=false);

  ---Obtem o peso de um produto


   public produtoNaCestaPeso : Produto`NomeProduto ==> real
   produtoNaCestaPeso (prodNome) ==
   (for all produto in set produtos
    do
     if prodNome = produto.nome
       then return produto.peso;
    return 99999;
   )
   pre produtos <> undefined
   post RESULT >= 0;

end Cesta
```

Function or operation	Line	Coverage	Calls
Cesta	12	100.0%	501
adicionaPesoProduto	34	100.0%	87
adicionaProduto	16	100.0%	1070
alterarTamanho	52	100.0%	97
produtoNaCesta	58	100.0%	392
produtoNaCestaPeso	85	100.0%	302
removePesoProduto	43	100.0%	42
removeProduto	25	100.0%	21
Cesta.vdmpp		100.0%	2512

4 Cliente

```
class Cliente

types
 public Genero = <HOMEM> | <MULHER>;
 public EncomendaStatus = <SEM_ENC> | <COM_ENC> | <ENTREGUE> | <CANCELADA> | <PRONTA> | <
     POR_CONCLUIR>;
 public Nome = seq1 of (char);
instance variables
  public encomenda:Cesta;
  public estadoEnc: EncomendaStatus;
  public nome:Nome;
  public genero: Genero;

operations

  public  Cliente : Nome * Genero ==> Cliente
  Cliente(n, gen) == (nome:= n; genero:= gen; estadoEnc := <SEM_ENC>; encomenda:= new Cesta();
      return self;)
  pre len n > 0 and gen <> undefined
  post nome = n and genero = gen and estadoEnc = <SEM_ENC>;



  --mete o estado atual da encomenda para cancelada
  public  removeCesta : ()  ==> ()
  removeCesta() == estadoEnc := <CANCELADA>

  pre encomenda <> undefined
  post estadoEnc = <CANCELADA>;

 --associa uma nova cesta ao cliente
  public  mudaCesta : Cesta ==> ()
  mudaCesta(cesta) ==

    (encomenda := cesta; estadoEnc := <COM_ENC>)
  pre cesta <> undefined
  post encomenda = cesta and encomenda <> encomenda~ and estadoEnc = <COM_ENC>;



 --altera  o estado da encomenda do cliente para pronto
 public encomendaPronta: () ==> ()
 encomendaPronta() == estadoEnc := <PRONTA>
 pre estadoEnc = <COM_ENC>
 post estadoEnc = <PRONTA>;


  --cliente levanta a cesta
 public levantaCesta: () ==> ()
 levantaCesta() == estadoEnc := <ENTREGUE>
 pre estadoEnc = <PRONTA>
 post estadoEnc = <ENTREGUE>;


 -- verifica se os requesitos da encomenda sao correspondidos
 public verificaCestaParametros: () ==> ()
 verificaCestaParametros () ==
  if(encomenda.tamanho = <PEQUENA>)
   then (
    if(card encomenda.produtos = 7 and encomenda.peso >= 3) --check requesitos encomenda pequena
    then estadoEnc := <PRONTA>
    else  estadoEnc := <POR_CONCLUIR>
   )
   else if(encomenda.tamanho = <GRANDE>)
```

```
   then (
    if(card encomenda.produtos = 8 and encomenda.peso >= 6) --check requesitos encomenda grande
     then estadoEnc := <PRONTA>
     else estadoEnc := <POR_CONCLUIR>
   )
 pre estadoEnc = <COM_ENC> or estadoEnc = <POR_CONCLUIR>
 post estadoEnc = <PRONTA> or estadoEnc = <POR_CONCLUIR>; -- se a encomenda esta completa passa
     para pronta se imcompleta para <POR_CONCLUIR>

end Cliente
```

Function or operation	Line	Coverage	Calls
Cliente	14	100.0%	187
encomendaPronta	29	100.0%	19
levantaCesta	34	100.0%	19
mudaCesta	23	100.0%	139
removeCesta	19	100.0%	21
verificaCestaParametros	40	100.0%	90
Cliente.vdmpp		100.0%	475

5 FrutaFeia

```
class FrutaFeia

instance variables
 public centros : set of CentroDistribuicao;
 public agricultores: set of Agricultor;

 inv forall centro in set centros & exists1 cent in set centros & centro.localizacao = cent.
     localizacao;

 inv forall agricultor in set agricultores & exists1 agro in set agricultores & agro.nome =
     agricultor.nome;
 inv forall centro in set centros & forall cli in set centro.clientes & exists1 cliente in set
     centro.clientes & cli.nome = cliente.nome;



operations
 public FrutaFeia: () ==> FrutaFeia
 FrutaFeia() == (centros := {}; agricultores := {})
 post centros = {} and agricultores = {};

 --Adiciona cliente ao centro respectivo
 public adicionaCliente: Cliente * CentroDistribuicao`Local ==> ()

 adicionaCliente( cliente, centroLocal ) ==
 for all centro in set centros
 do
   if ( centro.localizacao = centroLocal)
    then  centro.adicionaCliente(cliente)
 pre forall centro in set centros & (forall cli in set centro.clientes & cli.nome <> cliente.nome
     )
 post exists1 centro in set centros & cliente in set centro.clientes;
```

```
-- Remove cliente do seu centro respectivo
public removeCliente:  Cliente`Nome * CentroDistribuicao`Local ==> ()
removeCliente(clienteNome, centroLocal) ==
 for all centro in set centros
do

  if ( centro.localizacao = centroLocal)
   then  centro.removeCliente(clienteNome)
pre exists1 centro in set centros & (exists1 cliente in set centro.clientes & cliente.nome =
    clienteNome)
post forall  centro in set centros & (forall cliente in set centro.clientes & cliente.nome <>
    clienteNome);

-- adiciona um centro de Distribuicao do registo

public adicionaCentro: CentroDistribuicao  ==> ()
adicionaCentro(centro) ==
  centros := centros union {centro}
pre centros <> undefined and forall cent in set centros & centro.localizacao <> cent.localizacao
post card centros = card centros~ +1 and exists1 cent in set centros &  centro.localizacao= cent
    .localizacao;


-- remove um Centro de Distribuiao do registo
public removeCentro: CentroDistribuicao`Local  ==> ()
removeCentro(centroLocal) ==
 for all centro in set centros
 do
  if(centroLocal = centro.localizacao)
   then (centros := centros \ {centro}; return)

pre centros <> undefined and exists1 centro in set centros & centro.localizacao = centroLocal
post card centros = card centros~ - 1  and forall centro in set centros & centro.localizacao <>
    centroLocal;

-- adiciona um agricultor do registo

public adicionaAgricultor: Agricultor ==> ()
adicionaAgricultor(agricultor) ==
  agricultores := agricultores union {agricultor}
pre agricultor <> undefined and agricultor not in set agricultores
post card agricultores = card agricultores~ +1 ;

 -- retira um agricultor registo
public removeAgricultor: Agricultor`Nome  ==> ()
removeAgricultor(agricultorNome) ==
for all agricultor in set agricultores
 do
  if( agricultor.nome = agricultorNome)

  then(agricultores := agricultores \ {agricultor}; return;)
pre agricultores <> undefined and exists1 agricultor in set agricultores & agricultor.nome =
    agricultorNome
post card agricultores = card agricultores~ - 1 and forall agricultor in set agricultores &
    agricultor.nome <> agricultorNome ;

-- Retorna todos os clientes distribuidos pelos diferentes centros
public getTodosClientes: () ==> set of Cliente
getTodosClientes() ==
(
 dcl  clientes: set of Cliente := {};
 for all centro in set centros
  do
   clientes := clientes union centro.clientes;
```

```
 return clientes;
)
pre centros <> undefined;


-- Retorna todos os produtos de todos os agricultores ( os produtos com o mesmo nome de
    diferentes agricultores sao indepedentes)
public getTodosProdutos: () ==> set of Produto

getTodosProdutos () ==
(
 dcl produtos: set of Produto := {};

 for all agricultor in set agricultores
  do
   produtos := produtos union rng agricultor.stock;

 return produtos;
)
pre agricultores <> undefined;

--Gera uma cesta grande
public geraCestaGrande: () ==> Cesta
geraCestaGrande() ==
 (
  dcl cesta : Cesta := geraCesta(6, 0.75, 8, <GRANDE> );
  return cesta;
 );

--Gera uma cesta pequena
public geraCestaPequena: () ==> Cesta
geraCestaPequena() ==
 (
  dcl cesta : Cesta := geraCesta(3, 0.430, 7, <PEQUENA>);
  return cesta
 );

--Cria uma cesta com determinados parametros
public geraCesta: real * real * real * Cesta`Tamanho==> Cesta
geraCesta(pesoMinimo, pesoMinimoProduto, totalProdutos, tipoCesta) ==
(
 dcl cesta : Cesta := new Cesta();
 dcl produtos : set of Produto := getTodosProdutos();
 dcl totalNaCesta : nat := 0;
 dcl pesoAretirar : real := pesoMinimoProduto;
 dcl minimo: real := pesoMinimo;
 dcl runs: nat := 0;

 cesta.alterarTamanho(tipoCesta);

 --Adiciona produtos   cesta
 while(minimo > 0 and runs < 3)
 do
  (
   for all produto in set produtos
   do
    if(produto.peso >= pesoAretirar)
     then
     (
     if(cesta.produtoNaCesta(produto.nome) = false and totalNaCesta < totalProdutos) --Se o
         produto nao esta na cesta e o numero minimo de produtos nao foi alcanado
      then
       ( produto.removePeso(pesoAretirar);
         cesta.adicionaProduto(new Produto(produto.nome, produto.origem, pesoAretirar));
```

```
          totalNaCesta := totalNaCesta + 1;

          minimo := minimo - pesoAretirar;
        )
       else
       (
        if(totalNaCesta = totalProdutos  and minimo <= 0) -- Se a cesta esta completa
          then return cesta
          else
          ( --Adiciona mais X peso a um produto numa cesta
           if(cesta.produtoNaCestaPeso(produto.nome) + pesoAretirar < 0.8) -- limite de +- 50%
                que um produto pode ocupar
           then
            ( produto.removePeso(pesoAretirar);
             if(cesta.adicionaPesoProduto(produto.nome, pesoAretirar))
              then minimo := minimo - pesoAretirar
             else produto.adicionaPeso(pesoAretirar)
            );
          );
       );
      );
    pesoAretirar := pesoAretirar / (runs + 1);
    runs := runs + 1;
   );
   return cesta;
);

public preencheCesta: Cesta ==> () -- procura produtos e tenta responder as necessidades da
     cesta se houver stock
preencheCesta(cesta) ==
(
 dcl produtos : set of Produto := getTodosProdutos();
 dcl totalNaCesta : nat := card cesta.produtos;
 dcl pesoAretirar : real := 0.3;
 dcl minimo: real := 0;
 dcl runs: nat := 0;
 dcl totalProdutos: nat := 0;

 --- setup com os requesitos das cestas
 if(cesta.tamanho = <PEQUENA>)
   then (
      totalProdutos := 7;
      minimo := 3 - cesta.peso;
     )
 else
  if(cesta.tamanho = <GRANDE>)
   then ( totalProdutos := 8;
       minimo := 6 - cesta.peso;
      );

 --Adicionar produtos   cesta
  while(minimo > 0 and runs < 3)
 do
  (
   for all produto in set produtos
   do
    if(produto.peso >= pesoAretirar)
     then

     (
     if(cesta.produtoNaCesta(produto.nome) = false and totalNaCesta < totalProdutos) -- Se nao
          existir na cesta e a cesta nao tem o numero de produtos minimo
      then
       (
         -- adicionar produto   cesta
```

```
          produto.removePeso(pesoAretirar);
          cesta.adicionaProduto(new Produto(produto.nome, produto.origem, pesoAretirar));
          totalNaCesta := totalNaCesta + 1;
          minimo := minimo - pesoAretirar;
        )
       else
       (
        if(totalNaCesta = totalProdutos  and minimo <= 0) -- Se a encomenda ja esta completa
          then return
          else
          ( -- tenta adicionar X peso de um produto que ja se encontra na cesta
           if(cesta.produtoNaCestaPeso(produto.nome) + pesoAretirar < 0.8)
           then (
            produto.removePeso(pesoAretirar);
            if(cesta.adicionaPesoProduto(produto.nome, pesoAretirar))
              then minimo := minimo - pesoAretirar
             else produto.adicionaPeso(pesoAretirar);
         );
          );
       );
      );
    pesoAretirar := pesoAretirar / (runs + 1); -- tenta adicionar quantidades mais pequenas
    runs := runs + 1;
   );
);

public geraCestaTodosClientes: () ==> ()
geraCestaTodosClientes() ==
 for all centro in set centros
 do (
  for all cliente in set centro.clientes -- todos os clientes em todos os centros
  do (
   IO`println(cliente);
   if cliente.estadoEnc = <COM_ENC> -- tenta corresponder a uma encomenda feita pelo cliente
    then (
        if (cliente.encomenda.tamanho = <PEQUENA>)
         then ( cliente.mudaCesta(geraCestaPequena());
             cliente.verificaCestaParametros(); -- avalia se os requesitos da encomenda estao
                 cumpridoss
           )
       else
        if(cliente.encomenda.tamanho = <GRANDE>)
         then( cliente.mudaCesta(geraCestaGrande());
            cliente.verificaCestaParametros()
           )
       )
   else ( if cliente.estadoEnc = <POR_CONCLUIR>  -- tenta completar uma encomenda pendente
       then (
          preencheCesta(cliente.encomenda);
          cliente.verificaCestaParametros();
           )
   );
   );
  );
end FrutaFeia
```

Function or operation	Line	Coverage	Calls
FrutaFeia	8	100.0%	6
adicionaAgricultor	38	100.0%	10
adicionaCentro	26	83.8%	4

adicionaCliente	12	100.0%	8
geraCesta	93	98.0%	10
geraCestaGrande	79	100.0%	4
geraCestaPequena	86	100.0%	6
geraCestaTodosClientes	194	100.0%	2
geraCestas	51	82.6%	2
getTodosClientes	55	100.0%	6
getTodosProdutos	67	100.0%	16
preencheCesta	140	98.4%	4
removeAgricultor	44	89.4%	0
removeCentro	32	89.4%	2
removeCliente	19	85.1%	4
FrutaFeia.vdmpp		96.1%	84

6 Produto

```
class Produto

types
 public NomeProduto = seq of char;
 public Origem = seq of char;
 public Peso = real;
instance variables
  public nome: NomeProduto;
  public origem:seq of (char);
  public peso: Peso;

  inv peso >= 0;

operations

  public  Produto : NomeProduto * Origem * Peso ==> Produto
  Produto(n, o, p) == (nome := n; origem := o; peso := p;return self;)
  pre len n > 0 and len o > 0 and p >= 0
  post nome = n and origem = o and peso = p;


  --Adiciona peso ao produto
  public  adicionaPeso : Peso ==> ()
  adicionaPeso(p) == peso := peso + p
  pre p > 0
  post peso = peso~ + p;


 --Remove peso do produto
  public  removePeso : Peso ==> ()
  removePeso(p) == peso := peso - p
  pre p > 0 and peso - p >= 0
  post peso = peso~ - p;


end Produto
```

Function or operation	Line	Coverage	Calls
Produto	15	100.0%	986
adicionaPeso	20	100.0%	25
removePeso	26	100.0%	125
Produto.vdmpp		100.0%	1136

7 Runner

```
class Runner
  operations

    public static main() == (
      IO`println("Running tests...");
      new TestAgricultor().testAll();
      new TestCentroDistribuicao().testAll();
      new TestCesta().testAll();
      new TestCliente().testAll();
      new TestProduto().testAll();
      new TestFrutaFeia().testAll();
      IO`println("Tests completed!");
    );

end Runner
```

Function or operation	Line	Coverage	Calls
main	3	91.6%	25
Runner.vdmpp		91.6%	25

8 SuiteTestCase

```
-- Main class of Tests
class SuiteTestCase

operations

  protected assertTrue: bool ==> ()
  assertTrue(arg) ==
    return
  pre arg;



  protected assertFalse: bool ==> ()
  assertFalse(arg) ==
    return
  pre arg = false;

  protected assertEqual: ? * ? ==> ()
  assertEqual(expected, actual) ==
    if expected <> actual then (
      IO`print("Actual value (");
```

```
      IO`print(actual);
      IO`print(") different from expected (");
      IO`print(expected);
      IO`println(")\n")
    )
  post expected = actual

end SuiteTestCase
```

Function or operation	Line	Coverage	Calls
assertEqual	10	100.0%	2
assertFalse	10	100.0%	61
assertTrue	5	100.0%	147
SuiteTestCase.vdmpp		100.0%	210

9 TestAgricultor

```
class TestAgricultor is subclass of SuiteTestCase

operations
  public testConstructor() == (

   dcl agricultor: Agricultor := new Agricultor("Pedro", "Mirandela");
    IO`println("\t\t Testing Constructor");
    assertEqual("Pedro", agricultor.nome);
    assertEqual("Mirandela", agricultor.localizacao);
  );

 public testAdicionaAndRemovePesoProduto() == (

  dcl agricultor: Agricultor := new Agricultor("Pedro", "Mirandela");
   dcl produto: Produto := new Produto("Maca", "Golden", 5);
  IO`println("\t\t Testing AdicionaAndRemovePesoProduto");
  agricultor.adicionaProduto(produto);
  assertEqual(5, produto.peso);
  assertFalse(agricultor.adicionaPesoProduto("Banana", 5));
  assertTrue(agricultor.adicionaPesoProduto(produto.nome, 5));
  assertEqual(10, produto.peso);
  assertFalse(agricultor.removePesoProduto("Banana", 5));
  assertTrue(agricultor.removePesoProduto(produto.nome, 5));
  assertEqual(5, produto.peso);
  assertEqual(1, card dom agricultor.stock);

  agricultor.removeProduto("Maca");
  assertEqual(0, card dom agricultor.stock);
  );

  public testAll() == (
    IO`println("\t Testing Agricultor class");
    testConstructor();
    testAdicionaAndRemovePesoProduto();
  );

end TestAgricultor
```

Function or operation	Line	Coverage	Calls
testAdicionaAndRemovePesoProduto	12	98.4%	17
testAll	24	83.3%	15
testConstructor	5	93.7%	16
TestAgricultor.vdmpp		96.4%	48

10 TestCentroDistribuicao

```
class TestCentroDistribuicao is subclass of SuiteTestCase

operations

  public testConstructor() == (
    dcl centro: CentroDistribuicao := new CentroDistribuicao("Porto");
    IO`println("\t\t Testing Constructor");
    assertEqual("Porto", centro.localizacao);
  );


 public testManageCenter()  == (
  dcl centro: CentroDistribuicao := new CentroDistribuicao("Porto");
  dcl cliente: Cliente := new Cliente("Fernando", <HOMEM>);
  IO`println("\t Testing ManageCenter");
  assertEqual(0, card centro.clientes);
  centro.adicionaCliente(cliente);
  assertEqual(1, card centro.clientes);
  centro.removeCliente(cliente.nome);

  assertEqual(0, card centro.clientes);
);

  public testAll() == (
    IO`println("\t Testing CentroDistribuicao class");
    testConstructor();
    testManageCenter();
  );
end TestCentroDistribuicao
```

Function or operation	Line	Coverage	Calls
testAll	18	83.3%	28
testConstructor	4	90.9%	32
testManageCenter	10	96.7%	0
TestCentroDistribuicao.vdmpp		93.7%	60

11 TestCesta

```
class TestCesta is subclass of SuiteTestCase

operations
```

```
  public testConstructor() == (
   dcl cesta: Cesta := new Cesta();
    IO`println("\t\t Testing Constructor");
    assertEqual(<PEQUENA>, cesta.tamanho);
    assertEqual(0, cesta.peso);
  );


  public testAdicionaAndRemoveProduto() == (
   dcl cesta: Cesta := new Cesta();
   dcl produto: Produto := new Produto("Maca", "Golden", 5);
   IO`println("\t\t Testing AdicionaAndRemoveProduto");
   assertEqual(0, card cesta.produtos);
   cesta.adicionaProduto(produto);
   assertEqual(1, card cesta.produtos);
   cesta.removeProduto(produto);
   assertEqual(0, card cesta.produtos);
  );


  public testAdicionaAndRemovePesoProduto() == (
   dcl cesta: Cesta := new Cesta();
   dcl produto: Produto := new Produto("Maca", "Golden", 5);
  IO`println("\t\t Testing AdicionaAndRemovePesoProduto");
  cesta.adicionaProduto(produto);
  assertEqual(5, produto.peso);
  assertTrue(cesta.adicionaPesoProduto(produto.nome, 5));
  assertEqual(10, produto.peso);
  assertTrue(cesta.removePesoProduto(produto.nome, 5));
  assertEqual(5, produto.peso);
  assertFalse(cesta.removePesoProduto("Morango", 5));
  assertFalse(cesta.adicionaPesoProduto("Banana", 4));

  );

  public testAlterarTamanho() == (
   dcl cesta: Cesta := new Cesta();
  IO`println("\t\t Testing AdicionaAndRemovePesoProduto");
  assertEqual(<PEQUENA>, cesta.tamanho);
  cesta.alterarTamanho(<GRANDE>);
  assertEqual(<GRANDE>, cesta.tamanho);

  );

  public testAll() == (
    IO`println("\t Testing Cesta class");
    testConstructor();
    testAdicionaAndRemoveProduto();
    testAdicionaAndRemovePesoProduto();
    testAlterarTamanho();
  );

end TestCesta
```

Function or operation	Line	Coverage	Calls
testAdicionaAndRemovePesoProduto	22	97.8%	6
testAdicionaAndRemoveProduto	11	96.6%	30
testAll	42	87.5%	2
testAlterarTamanho	34	93.7%	2

testConstructor	4	92.8%	60
TestCesta.vdmpp		95.6%	100

12 TestCliente

```
class TestCliente is subclass of SuiteTestCase

operations

  public testConstructor() == (
   dcl cliente: Cliente := new Cliente("Fernando", <HOMEM>);
    IO`println("\t\t Testing Constructor");
    assertEqual("Fernando", cliente.nome);
    assertEqual(<HOMEM>, cliente.genero);
    assertEqual(<SEM_ENC>, cliente.estadoEnc);
  );


  public testMudaRemoveCesta() == (
   dcl cliente: Cliente := new Cliente("Fernando", <HOMEM>);
   dcl cestaA: Cesta := new Cesta();
   IO`println("\t\t Testing AssociaCesta");
   assertEqual(<SEM_ENC>, cliente.estadoEnc);
   cliente.mudaCesta(cestaA);
   assertEqual(<COM_ENC>, cliente.estadoEnc);
   cliente.removeCesta();
   assertEqual(<CANCELADA>, cliente.estadoEnc);
  );



  public testVerificaEstadoEncomenda() == (
   dcl cliente: Cliente := new Cliente("Fernando", <HOMEM>);
   dcl cesta: Cesta := new Cesta();
   dcl cesta1: Cesta := new Cesta();
   dcl produto1: Produto := new Produto("Maca1", "Golden", 5);
   dcl produto2: Produto := new Produto("Maca2", "Golden", 5);
   dcl produto3: Produto := new Produto("Maca3", "Golden", 5);
   dcl produto4: Produto := new Produto("Maca4", "Golden", 5);
   dcl produto5: Produto := new Produto("Maca5", "Golden", 5);
   dcl produto6: Produto := new Produto("Maca6", "Golden", 5);
   dcl produto7: Produto := new Produto("Maca7", "Golden", 5);
   dcl produto8: Produto := new Produto("Maca8", "Golden", 5);
   cesta1.alterarTamanho(<GRANDE>);

   IO`println("\t\t Testing AssociaCesta");
   assertEqual(<SEM_ENC>, cliente.estadoEnc);
   cliente.mudaCesta(cesta);
   cliente.verificaCestaParametros();
   cliente.mudaCesta(cesta1);
   cliente.verificaCestaParametros();
   cliente.encomenda.adicionaProduto(produto1);
   cliente.encomenda.adicionaProduto(produto2);
   cliente.encomenda.adicionaProduto(produto3);
   cliente.encomenda.adicionaProduto(produto4);
   cliente.encomenda.adicionaProduto(produto5);
   cliente.encomenda.adicionaProduto(produto6);
   cliente.encomenda.adicionaProduto(produto7);
   cliente.encomenda.adicionaProduto(produto8);
   cliente.verificaCestaParametros();
```

```
  );

  public testUseCaseCesta() == (
   dcl cliente: Cliente := new Cliente("Fernando", <HOMEM>);
   dcl cesta: Cesta := new Cesta();
   IO`println("\t\t Testing AssociaCesta");
   assertEqual(<SEM_ENC>, cliente.estadoEnc);
   cliente.mudaCesta(cesta);
   assertEqual(<COM_ENC>, cliente.estadoEnc);
   cliente.encomendaPronta();
   assertEqual(<PRONTA>, cliente.estadoEnc);
   cliente.levantaCesta();
   assertEqual(<ENTREGUE>, cliente.estadoEnc);
  );

  public testAll() == (
    IO`println("\t Testing Cliente class");
    testConstructor();
    testMudaRemoveCesta();
    testVerificaEstadoEncomenda();
    testUseCaseCesta();
  );

end TestCliente
```

Function or operation	Line	Coverage	Calls
testAll	36	87.5%	16
testConstructor	4	95.0%	30
testMudaRemoveCesta	12	96.0%	30
testUseCaseCesta	23	96.6%	16
testVerificaEstadoEncomenda	23	98.7%	16
TestCliente.vdmpp		96.9%	108

13 TestFrutaFeia

```
class TestFrutaFeia is subclass of SuiteTestCase

operations



public testAdicionaRemove() == (
 dcl frutaFeia : FrutaFeia := new FrutaFeia();
 dcl agricultor: Agricultor := new Agricultor("Pedro", "Mirandela");
 dcl cliente: Cliente := new Cliente("Fernando", <HOMEM>);
 dcl centro: CentroDistribuicao := new CentroDistribuicao("Porto");
 IO`println("\t Testing AdicionaRemove");
 assertEqual(0, card frutaFeia.agricultores);
 frutaFeia.adicionaAgricultor(agricultor);
 assertEqual(1, card frutaFeia.agricultores);
 frutaFeia.removeAgricultor(agricultor.nome);
 assertEqual(0, card frutaFeia.agricultores);
 assertEqual(0, card frutaFeia.centros);
 frutaFeia.adicionaCentro(centro);
 assertEqual(1, card frutaFeia.centros);
```

```
 assertEqual(0, card frutaFeia.getTodosClientes());
 frutaFeia.adicionaCliente(cliente, "Porto");
 assertEqual(1, card frutaFeia.getTodosClientes());
 frutaFeia.removeCliente(cliente.nome, "Porto");
 assertEqual(0, card frutaFeia.getTodosClientes());
 frutaFeia.removeCentro(centro.localizacao);
 assertEqual(0, card frutaFeia.centros);
);

public testGeraCestaPequena() == (
 dcl frutaFeia : FrutaFeia := new FrutaFeia();
 dcl agricultor: Agricultor := new Agricultor("Pedro", "Mirandela");
 dcl agricultor2: Agricultor := new Agricultor("Manuel", "Porto");
 dcl pera: Produto := new Produto("Pera", "Mate", 2);
 dcl manga: Produto := new Produto("Manga", "Rosa", 3);
 dcl ananas: Produto := new Produto("Ananas", "Preto", 4);
 dcl maca: Produto := new Produto("Maca", "Golden", 5);
 dcl maracuja: Produto := new Produto("Maracuja", "Bambuin", 6);
  dcl alface: Produto := new Produto("Alface", "Alfacinha", 7);
  dcl pimento: Produto := new Produto("Pimento", "Verde", 8);
  dcl laranja: Produto := new Produto("Laranja", "Algarve", 8);
 dcl cesta: Cesta;
 IO`println("\t Testing GeraCestaPequena");


 agricultor.adicionaProduto(pera);
 agricultor.adicionaProduto(manga);
 agricultor.adicionaProduto(ananas);

 agricultor2.adicionaProduto(maca);
 agricultor2.adicionaProduto(maracuja);
 agricultor2.adicionaProduto(alface);
 agricultor2.adicionaProduto(pimento);
 agricultor2.adicionaProduto(laranja);

 frutaFeia.adicionaAgricultor(agricultor);
 frutaFeia.adicionaAgricultor(agricultor2);

 IO`println("\t\t Cesta:" );
 cesta := frutaFeia.geraCestaPequena();
 frutaFeia.preencheCesta(cesta);
 assertTrue(cesta.peso > 3);
 IO`println(cesta);
 assertEqual(7, card cesta.produtos);
 cesta := frutaFeia.geraCestaGrande();
 assertEqual(<GRANDE>, cesta.tamanho);
 frutaFeia.preencheCesta(cesta);
 assertTrue(cesta.peso >= 6);
 assertEqual(card cesta.produtos , 8);
);

public testeGereTodasEncomendas() == (
 dcl frutaFeia : FrutaFeia := new FrutaFeia();
 dcl agricultor: Agricultor := new Agricultor("Pedro", "Mirandela");
 dcl agricultor2: Agricultor := new Agricultor("Manuel", "Porto");
 dcl pera: Produto := new Produto("Pera", "Mate", 0.8);
 dcl manga: Produto := new Produto("Manga", "Rosa", 0.8);
 dcl ananas: Produto := new Produto("Ananas", "Preto", 4);
 dcl maca: Produto := new Produto("Maca", "Golden", 5);
 dcl maracuja: Produto := new Produto("Maracuja", "Bambuin", 6);
  dcl alface: Produto := new Produto("Alface", "Alfacinha", 7);
  dcl pimento: Produto := new Produto("Pimento", "Verde", 8);
  dcl laranja: Produto := new Produto("Laranja", "Algarve", 8);
  dcl centro: CentroDistribuicao := new CentroDistribuicao("Porto");
  dcl cliente: Cliente := new Cliente("Fernando", <HOMEM>);
```

```
  dcl cliente1: Cliente := new Cliente("Alfredo", <HOMEM>);
  dcl cliente2: Cliente := new Cliente("Joana", <MULHER>);
  dcl cesta: Cesta := new Cesta();
  dcl cesta1: Cesta := new Cesta();
 IO`println("\t Testing GereTodasEncomendas:" );

 agricultor.adicionaProduto(pera);
 agricultor.adicionaProduto(manga);
 agricultor.adicionaProduto(ananas);

 agricultor2.adicionaProduto(maca);
 agricultor2.adicionaProduto(maracuja);

 agricultor2.adicionaProduto(alface);
 agricultor2.adicionaProduto(pimento);
 agricultor2.adicionaProduto(laranja);

 frutaFeia.adicionaAgricultor(agricultor);
 frutaFeia.adicionaAgricultor(agricultor2);

 frutaFeia.adicionaCentro(centro);
 frutaFeia.adicionaCliente(cliente, "Porto");
 frutaFeia.adicionaCliente(cliente1, "Porto");
 frutaFeia.adicionaCliente(cliente2, "Porto");

 cliente.mudaCesta(new Cesta());
 cesta.alterarTamanho(<GRANDE>);
 cliente1.mudaCesta(cesta);
 cliente2.mudaCesta(cesta1);
 frutaFeia.geraCestaTodosClientes();

 IO`println("\t\t Cliente Status:" );
 IO`println(cliente);
 IO`println(cliente1);
 IO`println(cliente2);


 pera.adicionaPeso(4);
 manga.adicionaPeso(4);
  frutaFeia.geraCestaTodosClientes();
  IO`println("\t\t Cliente Status depois de reabastecer e distribuir:" );
  IO`println(cliente);
 IO`println(cliente1);
 IO`println(cliente2);

 assertEqual(<PRONTA>, cliente.estadoEnc);
 assertEqual(<PRONTA>, cliente1.estadoEnc);
 assertEqual(<PRONTA>, cliente2.estadoEnc);

);

public testAll() == (
 testAdicionaRemove();
 testGeraCestaPequena();
 testeGereTodasEncomendas();
);

end TestFrutaFeia
```

Function or operation	Line	Coverage	Calls

testAdicionaRemove	5	98.7%	12
testAll	95	80.0%	2
testGeraCestaPequena	5	99.1%	10
testeGereTodasEncomendas	42	99.3%	16
TestFrutaFeia.vdmpp		98.8%	40

14 TestProduto

```
class TestProduto is subclass of SuiteTestCase

operations

  public testConstructor() == (
   dcl produto: Produto := new Produto("Maca", "Golden", 5);
    IO`println("\t\t Testing Constructor");
    assertEqual("Maca", produto.nome);
    assertEqual("Golden", produto.origem);
    assertEqual(5, produto.peso);
  );


  public testAdicionaAndRemovePeso() == (
   dcl produto: Produto := new Produto("Maca", "Golden", 5);
  IO`println("\t\t Testing AdicionaAndRemovePeso");
  assertEqual(5, produto.peso);
  produto.adicionaPeso(5);
  assertEqual(10, produto.peso);
  produto.removePeso(5);
  assertEqual(5, produto.peso);

  );


  public testAll() == (
    IO`println("\t Testing Produto class");
    testConstructor();
    testAdicionaAndRemovePeso();
  );

end TestProduto
```

Function or operation	Line	Coverage	Calls
testAdicionaAndRemovePeso	12	96.0%	11
testAll	23	83.3%	11
testConstructor	4	95.2%	11
TestProduto.vdmpp		94.2%	33