MFES-FEUP

January 2, 2018

Contents

1 Agricultor

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

types
 public NomeProduto = Produto`NomeProduto;

operations
```

```
  public  Agricultor : seq of (char) * 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 = {|->};


  public  adicionaProduto : Produto ==> ()
   adicionaProduto(produto) == stock := stock munion {produto.nome |-> produto}
  pre produto <> undefined

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

  public removeProduto : NomeProduto ==> ()
   removeProduto(nomeProduto) == stock := {nomeProduto} <-: stock

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

  public  adicionaPesoProduto : 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 and nomeProduto in set dom stock;

  public  removePesoProduto : 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 and nomeProduto in set dom stock;


end Agricultor
```

2 CentroDistribuicao

```
class CentroDistribuicao
instance variables
 public clientes : set of Cliente;
  public localizacao:seq of (char);

operations
  public  CentroDistribuicao : seq of (char) ==> CentroDistribuicao
  CentroDistribuicao(local) == (clientes := {}; localizacao := local; return self;)

  pre len local > 0
  post localizacao = local and clientes = {};




end CentroDistribuicao
```

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>;


  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);


  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~ );


  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); return true;);
 return false;)
 pre len nomeProduto > 0 and p > 0;


 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;


  public alterarTamanho: Tamanho ==> ()
  alterarTamanho(tam)  ==
    tamanho := tam
  pre tam <> undefined
```

```
  post tamanho = tam;


  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;
end Cesta
```

4 Cliente

```
class Cliente

types
 public Genero = <HOMEM> | <MULHER>;
 public EncomendaStatus = <SEM_ENC> | <COM_ENC> | <ENTREGUE> | <CANCELADA> | <PRONTA>

instance variables
  public encomenda:Cesta;
  public estadoEnc: EncomendaStatus;
  public nome:seq1 of (char);
  public genero: Genero;

operations

  public  Cliente : seq1 of (char) * 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>;


  public  associaCesta : Cesta ==> ()
  associaCesta(cesta) ==
    (
     encomenda := cesta;
     estadoEnc := <COM_ENC>;
    )
  pre cesta <> undefined
  post encomenda = cesta;


  public  removeCesta : ()  ==> ()
  removeCesta() == estadoEnc := <CANCELADA>
  post estadoEnc = <CANCELADA>;


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


 public encomendaPronta: () ==> ()
```

```
 encomendaPronta() == estadoEnc := <PRONTA>
 pre estadoEnc = <COM_ENC>
 post estadoEnc = <PRONTA>;


 public levantaCesta: () ==> ()
 levantaCesta() == estadoEnc := <ENTREGUE>
 pre estadoEnc = <PRONTA>
 post estadoEnc = <ENTREGUE>;

end Cliente
```

5 FrutaFeia

```
class FrutaFeia

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

operations
 public FrutaFeia: () ==> FrutaFeia

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

 --public adicionaCliente: Cliente ==> () --REDO

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

 --public removeCliente: Cliente ==> ()  --REDO

 --removeCliente(cliente) ==
 --  clientes := clientes \ {cliente}
 --pre cliente <> undefined and cliente in set clientes
 --post card clientes~ - 1 = card clientes;

 public adicionaCentro: CentroDistribuicao  ==> ()

 adicionaCentro(centro) ==
   centros := centros union {centro}
 pre centro <> undefined and centro not in set centros
 post card centros = card centros~ +1 ;

 public removeCentro: CentroDistribuicao  ==> ()

 removeCentro(centro) ==
   centros := centros \ {centro}
 pre centro <> undefined and centro in set centros
 post card centros = card centros~ - 1 ;

 public adicionaAgricultor: Agricultor ==> ()

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

```
 public removeAgricultor: Agricultor  ==> ()

 removeAgricultor(agricultor) ==
   agricultores := agricultores \ {agricultor}
 pre agricultor <> undefined and agricultor in set agricultores
 post card agricultores = card agricultores~ - 1 ;



 public geraCestas: () ==> ()
 geraCestas() ==
 return;


 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;



 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;


 public geraCestaPequena: () ==> Cesta
 geraCestaPequena () ==
 (
  dcl cesta : Cesta := new Cesta();
  dcl produtos : set of Produto := getTodosProdutos();
  dcl totalNaCesta : nat := 0;
  dcl pesoAretirar : real := 0.5;
  for all produto in set produtos
   do
    if(cesta.produtoNaCesta(produto.nome) = false and produto.peso >= 0.5)
     then
      (
       produto.removePeso(pesoAretirar);
       cesta.adicionaProduto(new Produto(produto.nome, produto.origem, pesoAretirar));
       totalNaCesta := totalNaCesta + 1;
       if(totalNaCesta = 7)
        then return cesta;
      );
   return cesta;
 ); -- TODO peso minimo em cada produto? numero minimo de produtos

 --TODO update total peso
 --TODO update total produtos
end FrutaFeia
```

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;


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



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


end Produto
```

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
```

8 SuiteTestCase

```
-- Main class of Tests
class SuiteTestCase

operations

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


  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
```

9 TestAgricultor

```
class TestAgricultor is subclass of SuiteTestCase
instance variables

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);
  assertTrue(agricultor.adicionaPesoProduto(produto.nome, 5));
  assertEqual(10, produto.peso);
  assertTrue(agricultor.removePesoProduto(produto.nome, 5));
  assertEqual(5, produto.peso);
  );
```

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

end TestAgricultor
```

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 produto: Produto := new Produto("Maca" ,"Golden", 5);
 --  dcl produto2: Produto := new Produto("Maracuja","Rose", 1);

   IO`println("\t\t TODO");
);

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

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);
   );


   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
```

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 testAssociaMudaRemoveCesta() == (
    dcl cliente: Cliente := new Cliente("Fernando", <HOMEM>);
    dcl cestaA: Cesta := new Cesta();
    dcl cestaB: Cesta := new Cesta();
```

```
   IO`println("\t\t Testing AssociaCesta");
   assertEqual(<SEM_ENC>, cliente.estadoEnc);
   cliente.associaCesta(cestaA);
   cliente.mudaCesta(cestaB);
   assertEqual(<COM_ENC>, cliente.estadoEnc);
   cliente.removeCesta();
   assertEqual(<CANCELADA>, cliente.estadoEnc);
  );


  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.associaCesta(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();
    testAssociaMudaRemoveCesta();
    testUseCaseCesta();
  );

end TestCliente
```

13 TestFrutaFeia

```
class TestFrutaFeia is subclass of SuiteTestCase

operations




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 cesta: Cesta;

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

```
 agricultor.adicionaProduto(ananas);

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

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

 IO`println("\t\t Cesta:" );
 cesta := frutaFeia.geraCestaPequena();
 IO`println(cesta);
 assertEqual(7, card cesta.produtos);

);



public testAll() == (
  testGeraCestaPequena();
);

end TestFrutaFeia
```

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
```