Hokify

December 25, 2017

Contents

1 Interesse

```
class Interesse
types
-- TODO Define types here
 public String = seq of char;

values
-- TODO Define values here

instance variables
-- TODO Define instance variables here
 private nome: String;

operations
-- TODO Define operations here




 public Interesse: String ==> Interesse
 Interesse(nm) == (
  nome := nm;
  return self
 );

 public pure getNome : () ==> String
 getNome() ==
 (
  return nome;
 );

functions
-- TODO Define functiones here

traces
```

```
-- TODO Define Combinatorial Test Traces here

end Interesse
```

Function or operation	Line	Coverage	Calls
Interesse	16	0.0%	0
Utilizador	16	0.0%	0
getNome	15	0.0%	0
Interesse.vdmpp		0.0%	0

2 MyTestCase

```
class MyTestCase
/*
  Superclass for test classes, simpler but more practical than VDMUnit`TestCase.
  For proper use, you have to do: New -> Add VDM Library -> IO.
  JPF, FEUP, MFES, 2014/15.
*/

operations

 -- Simulates assertion checking by reducing it to pre-condition checking.
 -- If 'arg' does not hold, a pre-condition violation will be signaled.

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

 -- Simulates assertion checking by reducing it to post-condition checking.
 -- If values are not equal, prints a message in the console and generates
 -- a post-conditions violation.

 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 MyTestCase
```

Function or operation	Line	Coverage	Calls
assertEqual	20	38.8%	1
assertTrue	12	0.0%	0
MyTestCase.vdmpp		35.0%	1

3 TestUtilizador

```
class TestUtilizador is subclass of MyTestCase
types
-- TODO Define types here
values
-- TODO Define values here
instance variables
-- TODO Define instance variables here
operations
-- TODO Define operations here


public TestPlayerFirst :() ==> ()
TestPlayerFirst() ==
(
 dcl user : Utilizador := new Utilizador("pedro",<Masculino>,27);
 assertEqual(user.getNome(),"pedro");
 return;
);


public static main: () ==> ()
main() ==
(
   new TestUtilizador().TestPlayerFirst();
);

functions

-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end TestUtilizador
```

Function or operation	Line	Coverage	Calls
TestPlayerFirst	10	100.0%	1
TestPlayerStart	10	100.0%	1
TestUpdateRemovepieces	25	0.0%	0
main	45	0.0%	0
main	18	100.0%	1
TestUtilizador.vdmpp		100.0%	3

4 Utilizador

```
class Utilizador
types
-- TODO Define types here
 public String = seq of char;
 public Sexo = <Masculino> | <Feminino>;
 public Interesses = set of Interesse;

values
-- TODO Define values here
```

```
instance variables
-- TODO Define instance variables here
 private nome: String;

 private sexo: Sexo;
 private idade: nat1;
 private interesses: Interesses := {};

operations
-- TODO Define operations here

 public Utilizador: String * Sexo * nat1 ==> Utilizador
 Utilizador(nm,sexoC,idadeC) == (
  nome := nm;
  sexo := sexoC;
  idade := idadeC;
  return self
 )
 post interesses = {} and


   nome = nm;

 public addInteresse: Interesse ==> ()

  addInteresse(interesse) == interesses := interesses union {interesse}
  pre interesse not in set interesses
  post interesses = interesses~ union {interesse};


  public removeInteresse: Interesse ==> ()
  removeInteresse(interesse) == interesses := interesses \ {interesse}
  pre interesse in set interesses
  post interesses = interesses~ \ {interesse};

  public pure getInteresses : () ==> Interesses
 getInteresses() ==
 (
  return interesses;
 );

 public pure getNome : () ==> String
 getNome() ==
 (
  return nome;
 );

functions
-- TODO Define functiones here

traces
-- TODO Define Combinatorial Test Traces here

end Utilizador
```

Function or operation	Line	Coverage	Calls
Utilizador	14	100.0%	1
addInteresse	28	0.0%	0
getInteresses	31	0.0%	0

getNome	20	100.0%	1
push	28	100.0%	1
removeInteresse	35	0.0%	0
Utilizador.vdmpp		39.2%	3