Hokify

December 26, 2017

Contents

1 Hokify

```
class Hokify
types
-- TODO Define types here
 public String = seq of char;
 public Utilizadores = set of Utilizador;
 public Trabalhos = set of Trabalho;

values
-- TODO Define values here
instance variables
-- TODO Define instance variables here
 private utilizadores: Utilizadores := {};
 private trabalhos: Trabalhos := {};

operations
-- TODO Define operations here


 --Construtor
 public Hokify: () ==> Hokify
 Hokify()==(return self);



 --Adicionar Utilizadores
 public addUtilizadores: Utilizador ==> ()
 addUtilizadores(utilizador) == utilizadores := utilizadores union {utilizador}
 pre sameUser(utilizador)
 post utilizadores = utilizadores~ union {utilizador};


 --Adicionar Trabalhos
```

```
 public addTrabalhos: Trabalho ==> ()
 addTrabalhos(trabalho) == trabalhos := trabalhos union {trabalho}
 pre sameTrabalho(trabalho)
 post trabalhos = trabalhos~ union {trabalho};



 -- Retorna os utilizadores
 public pure getUtilizadores : () ==> Utilizadores
 getUtilizadores() ==
 (
  return utilizadores;
 );



 -- Retorna os trabalhos
 public pure getTrabalhos : () ==> Trabalhos
 getTrabalhos() ==
 (
  return trabalhos;
 );

 -- Retorna os trabalhos por nome
 public pure getTrabalhosPorNome: String ==> Trabalhos
 getTrabalhosPorNome(name) == (return {trabalhos | trabalhos in set trabalhos & trabalhos.
     nomeSemelhante(name)})
 pre len name > 0;

 -- Verifica se o utilizador existe por email ou telefone
 public pure sameUser: Utilizador ==> bool
 sameUser(user) ==(
  for all u in set utilizadores do
   if (u.getEmail() = user.getEmail() or
     u.getTelefone() = user.getTelefone()) then


    return false;
  return true;
 );

 -- Verifica se o trabalho existe por email ou nome ou entidade
 public pure sameTrabalho: Trabalho ==> bool
 sameTrabalho(trab) ==(
  for all u in set trabalhos do
   if (u.getEmail() = trab.getEmail() or
     u.getNome() = trab.getNome() or
     u.getEntidade() = trab.getEntidade()) then
    return false;
  return true;
 );

functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end Hokify
```

Function or operation	Line	Coverage	Calls

Hokify	18	100.0%	7
addSkills	22	100.0%	14
addTrabalhos	28	100.0%	14
addUtilizadores	22	100.0%	14
getSkills	34	100.0%	14
getTrabalhos	41	100.0%	14
getTrabalhosPorNome	59	100.0%	3
getUtilizadores	34	100.0%	14
sameTrabalho	41	100.0%	3
sameUser	33	100.0%	1
searchByName	59	100.0%	2
Hokify.vdmpp		100.0%	100

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	100.0%	4
assertTrue	12	100.0%	17

MyTestCase.vdmpp		100.0%	21

3 TestAll

```
class TestAll 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","email@email.com",<Masculino>,<Licenciatura>,"
     Portugal","Porto",27,123456789);
 user.addInteresse("Informatica");
 user.addInteresse("Cinema");
 user.addSkills("Java");
 user.addSkills("VDM");

 assertEqual(user.getNome(),"Pedro");
 assertEqual(user.getEmail(),"email@email.com");
 assertEqual(user.getSexo(),<Masculino>);
 assertEqual(user.getEscolaridade(),<Licenciatura>);
 assertEqual(user.getPais(),"Portugal");
 assertEqual(user.getLocalidade(),"Porto");
 assertEqual(user.getIdade(),27);
 assertEqual(user.getTelefone(),123456789);
 assertEqual(user.getInteresses(),{"Informatica","Cinema"});
 assertEqual(user.getSkills(),{"Java","VDM"});

 user.removeInteresse("Cinema");
 assertEqual(user.getInteresses(),{"Informatica"});

 user.removeSkills("VDM");
 assertEqual(user.getSkills(),{"Java"});

 user.setNome("Pedro Faria");

 user.setEmail("PedroFaria@gmail.com");
 user.setPais("Espanha");
 user.setLocalidade("Madrid");
 user.setEscolaridade(<Mestrado>);
 user.setTelefone(987654321);
 user.setEscolaridade(<Mestrado>);
 assertEqual(user.getNome(),"Pedro Faria");
 assertEqual(user.getEmail(),"PedroFaria@gmail.com");
 assertEqual(user.getPais(),"Espanha");
 assertEqual(user.getLocalidade(),"Madrid");
 assertEqual(user.getEscolaridade(),<Mestrado>);
 assertEqual(user.getTelefone(),987654321);
 assertEqual(user.getEscolaridade(),<Mestrado>);


 --deve falhar pois estamos adicionando um duplicado
 --user.addInteresse("Informatica");
```

```
 --user.addSkills("Java");

 return;
);



public TestTrabalhoFirst :() ==> ()
TestTrabalhoFirst() ==
(
 dcl trabalho : Trabalho := new Trabalho("Programador de java","Google","google@google.pt",<
     Licenciatura>,123456789,"Portugal","Porto");
 trabalho.addInteresse("Informatica");
 trabalho.addInteresse("Cinema");
 trabalho.addSkills("Java");
 trabalho.addSkills("VDM");

 assertEqual(trabalho.getNome(),"Programador de java");
 assertEqual(trabalho.getEntidade(), "Google");
 assertEqual(trabalho.getEmail(),"google@google.pt");
 assertEqual(trabalho.getEscolaridade(),<Licenciatura>);
 assertEqual(trabalho.getTelefone(),123456789);
 assertEqual(trabalho.getPais(),"Portugal");
 assertEqual(trabalho.getLocalidade(),"Porto");
 assertEqual(trabalho.getInteresses(),{"Informatica","Cinema"});
 assertEqual(trabalho.getSkills(),{"Java","VDM"});

 trabalho.removeInteresse("Cinema");
 assertEqual(trabalho.getInteresses(),{"Informatica"});

 trabalho.removeSkills("VDM");
 assertEqual(trabalho.getSkills(),{"Java"});

 trabalho.setNome("Programador de C++");
 trabalho.setEntidade("apple");
 trabalho.setEmail("apple@apple.pt");
 trabalho.setPais("Espanha");
 trabalho.setLocalidade("Madrid");
 trabalho.setTelefone(987654321);
 trabalho.setEscolaridade(<Mestrado>);
 assertEqual(trabalho.getNome(),"Programador de C++");
 assertEqual(trabalho.getEmail(),"apple@apple.pt");
 assertEqual(trabalho.getPais(),"Espanha");
 assertEqual(trabalho.getLocalidade(),"Madrid");
 assertEqual(trabalho.getTelefone(),987654321);
 assertEqual(trabalho.getEscolaridade(),<Mestrado>);

 return;
);



public TestHokify :() ==> ()
TestHokify() ==
(
 dcl hokify : Hokify := new Hokify();
 dcl utilizador : Utilizador := new Utilizador("Pedro","email@email.com",<Masculino>,<
     Licenciatura>,"Portugal","Porto",27,123456789);
 dcl trabalho : Trabalho := new Trabalho("Programador de java","Google","google@google.pt",<
     Licenciatura>,123456789,"Portugal","Porto");
 dcl sameuser : Utilizador := new Utilizador("Pedro","email@email.com",<Masculino>,<Licenciatura
     >,"Portugal","Porto",27,123456789);
 dcl sametrabalho : Trabalho := new Trabalho("Programador de java","Google","google@google.pt",<
     Licenciatura>,123456789,"Portugal","Porto");
```

```
 dcl utilizador2 : Utilizador := new Utilizador("Fabiola","gmail@gmail.com",<Feminino>,<Mestrado
     >,"Portugal","Lisboa",26,123123123);
 dcl trabalho2 : Trabalho := new Trabalho("Programador de c++","Apple","apple@apple.pt",<
     Licenciatura>,4562343434,"Portugal","Lisboa");

 hokify.addUtilizadores(utilizador);
 hokify.addTrabalhos(trabalho);
 assertEqual(card hokify.getUtilizadores(),1);
 assertEqual(card hokify.getTrabalhos(),1);

 hokify.addUtilizadores(utilizador2);
 hokify.addTrabalhos(trabalho2);
 assertEqual(card hokify.getUtilizadores(),2);
 assertEqual(card hokify.getTrabalhos(),2);

 --Deve falhar pois estamos adicionando um utilizador igual
 --hokify.addUtilizadores(sameuser);
 --hokify.addTrabalhos(sametrabalho);
 assertTrue(not hokify.sameUser(sameuser));
 assertTrue(not hokify.sameTrabalho(sametrabalho));

 assertEqual(hokify.getTrabalhosPorNome("Programador"),{trabalho,trabalho2});
 assertEqual(hokify.getTrabalhosPorNome("java"),{trabalho});
 assertEqual(hokify.getTrabalhosPorNome("c++"),{trabalho2});
 return;
);


public static main: () ==> ()
main() ==
(
 IO`print("TestPlayerFirst -> ");
 new TestAll().TestPlayerFirst();
 IO`println("Passed");

 IO`print("TestTrabalhoFirst -> ");
 new TestAll().TestTrabalhoFirst();
 IO`println("Passed");

 IO`print("TestHokify -> ");
 new TestAll().TestHokify();
 IO`println("Passed");
);

functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end TestAll
```

Function or operation	Line	Coverage	Calls
TestHokify	102	100.0%	19
TestPlayerFirst	10	100.0%	21
TestPlayerFirst	10	100.0%	21
TestTrabalhoFirst	59	100.0%	21
TestTrabalhoFirst	50	100.0%	21
main	37	100.0%	19
main	101	100.0%	14

TestAll.vdmpp		100.0%	136

4 Trabalho

```
class Trabalho
types
-- TODO Define types here
 public String = seq of char;
 public Escolaridade = <Secundario> | <Licenciatura> | <Mestrado> | <Doutoramento>;
 public Interesses = set of String;
 public Skills = set of String;

values
-- TODO Define values here
instance variables
-- TODO Define instance variables here
 private nome: String;
 private entidade: String;
 private email: String;
 private escolaridade: Escolaridade;
 private telefone: nat1;
 private pais: String;
 private localidade: String;
 private interesses: Interesses := {};
 private skills: Skills := {};

operations
-- TODO Define operations here

 --Construtor

 public Trabalho: String * String * String * Escolaridade * nat1 * String * String ==> Trabalho
 Trabalho(nomeC,entidadeC,emailC,escolaridadeC,telefoneC,paisC,localidadeC) == (
  nome := nomeC;
  entidade := entidadeC;
  email := emailC;
  escolaridade := escolaridadeC;
  telefone := telefoneC;
  pais := paisC;
  localidade := localidadeC;
  return self;
 )
 post interesses = {} and
   skills = {} and
   nome = nomeC and
   entidade = entidadeC and
   email = emailC and
   escolaridade = escolaridadeC and
   telefone = telefoneC and
   pais = paisC and
   localidade = localidadeC;

 -- Adicionar interesses

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

 -- Remover interesses
```

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

-- Retorna os interesses

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

-- Adicionar skills

public addSkills: String ==> ()
addSkills(String) == skills := skills union {String}
pre String not in set skills
post skills = skills~ union {String};

-- Remover skills

public removeSkills: String ==> ()
removeSkills(String) == skills := skills \ {String}
pre String in set skills
post skills = skills~ \ {String};

-- Retorna as skills

public pure getSkills : () ==> Skills
getSkills() ==
(
 return skills;
);

-- Retorna o nome

public pure getNome : () ==> String
getNome() ==
(
 return nome;
);
-- Retorna o entidade

public pure getEntidade : () ==> String
getEntidade() ==
(
 return entidade;
);
-- Retorna o email

public pure getEmail : () ==> String
getEmail() ==
(
 return email;
);
-- Retorna o escolaridade

public pure getEscolaridade : () ==> Escolaridade
getEscolaridade() ==
(
 return escolaridade;
);
-- Retorna o telefone
```

```
public pure getTelefone : () ==> nat1
getTelefone() ==
(
 return telefone;
);
-- Retorna o pais

public pure getPais : () ==> String
getPais() ==
(
 return pais;
);
-- Retorna o localidade

public pure getLocalidade : () ==> String
getLocalidade() ==
(
 return localidade;
);


-- Editar Nome

public setNome: String ==> ()
setNome(newName) == nome := newName
pre newName <> undefined
post nome = newName;

-- Editar Entidade

public setEntidade: String ==> ()
setEntidade(newEntidade) == entidade := newEntidade
pre newEntidade <> undefined
post entidade = newEntidade;

-- Editar Email

public setEmail: String ==> ()
setEmail(newEmail) == email := newEmail
pre newEmail <> undefined
post email = newEmail;

-- Editar Telefone

public setTelefone: nat1 ==> ()
setTelefone(newTelefone) == telefone := newTelefone
pre newTelefone <> undefined
post telefone = newTelefone;

-- Editar Pais

public setPais: String ==> ()
setPais(newPais) == pais := newPais
pre newPais <> undefined
post pais = newPais;

-- Editar Localidade

public setLocalidade: String ==> ()
setLocalidade(newLocalidade) == localidade := newLocalidade
pre newLocalidade <> undefined
post localidade = newLocalidade;

-- Editar Escolaridade
```

```
 public setEscolaridade: Escolaridade ==> ()
 setEscolaridade(newEscolaridade) == escolaridade := newEscolaridade
 pre newEscolaridade <> undefined
 post escolaridade = newEscolaridade;



 public pure nomeSemelhante: String ==> bool
 nomeSemelhante(n) == (
  dcl nameS: seq of char := nome;
  dcl found: bool := false;

  while len nameS >= len n and not found do (
   found := true;

   for index = 1 to len n do
    if found and n(index) <> nameS(index) then (
     found := false;
    );

   if found then
    return true
   else (
    nameS := tl nameS;
    found := false;
   );
  );

  return false;
 )
 pre len n > 0;

functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end Trabalho
```

Function or operation	Line	Coverage	Calls
Trabalho	27	100.0%	75
addInteresse	49	100.0%	42
addSkills	68	100.0%	42
getEmail	99	100.0%	88
getEntidade	93	100.0%	49
getEscolaridade	105	100.0%	42
getInteresses	61	100.0%	42
getLocalidade	123	100.0%	42
getNome	87	100.0%	70
getPais	117	100.0%	42
getSkills	80	100.0%	42
getTelefone	111	100.0%	42
nomeSemelhante	172	100.0%	295
removeInteresse	55	100.0%	21
removeSkills	74	100.0%	21
setEmail	143	100.0%	21
setEntidade	137	100.0%	21

setEscolaridade	167	100.0%	21
setLocalidade	161	100.0%	21
setNome	131	100.0%	21
setPais	155	100.0%	21
setTelefone	149	100.0%	21
similarName	172	100.0%	295
Trabalho.vdmpp		100.0%	1397

5 Utilizador

```
class Utilizador
types
-- TODO Define types here
 public String = seq of char;
 public Sexo = <Masculino> | <Feminino>;
 public Escolaridade = <Secundario> | <Licenciatura> | <Mestrado> | <Doutoramento>;
 public Interesses = set of String;
 public Skills = set of String;

values
-- TODO Define values here

instance variables
-- TODO Define instance variables here
 private nome: String;
 private email: String;
 private sexo: Sexo;
 private escolaridade: Escolaridade;
 private idade: nat1;
 private telefone: nat1;
 private pais: String;
 private localidade: String;
 private interesses: Interesses := {};

 private skills: Skills := {};

operations
-- TODO Define operations here

 --Construtor
 public Utilizador: String * String * Sexo * Escolaridade * String * String * nat1 * nat1 ==>
     Utilizador
 Utilizador(nm,emailC,sexoC,escolaridadeC,paisC,localidadeC,idadeC,telefoneC) == (
  nome := nm;
  email := emailC;
  sexo := sexoC;
  escolaridade := escolaridadeC;
  idade := idadeC;
  telefone := telefoneC;

  pais := paisC;
  localidade := localidadeC;
  return self
 )
 post interesses = {} and
   skills = {} and

   nome = nm and
   email = emailC and
```

```
  sexo = sexoC and
  idade = idadeC and
  telefone = telefoneC and
  pais = paisC and

  localidade = localidadeC;

-- Adicionar interesses
public addInteresse: String ==> ()
addInteresse(String) == interesses := interesses union {String}
pre String not in set interesses
post interesses = interesses~ union {String};


-- Remover interesses
public removeInteresse: String ==> ()
removeInteresse(String) == interesses := interesses \ {String}
pre String in set interesses
post interesses = interesses~ \ {String};


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

);

-- Adicionar skills
public addSkills: String ==> ()
addSkills(String) == skills := skills union {String}
pre String not in set skills
post skills = skills~ union {String};


-- Remover skills
public removeSkills: String ==> ()
removeSkills(String) == skills := skills \ {String}
pre String in set skills
post skills = skills~ \ {String};

-- Retorna as skills
public pure getSkills : () ==> Skills
getSkills() ==

(
 return skills;
);

-- Retorna o nome
public pure getNome : () ==> String
getNome() ==

(
 return nome;
);

-- Retorna a idade
public pure getIdade : () ==> nat1
getIdade() ==

(
 return idade;
);
```

```
-- Retorna o telefone
public pure getTelefone : () ==> nat1
getTelefone() ==
(
 return telefone;
);

-- Retorna o email
public pure getEmail : () ==> String
getEmail() ==
(
 return email;
);


-- Retorna o sexo
public pure getSexo : () ==> Sexo
getSexo() ==
(
 return sexo;


);

-- Retorna o pais
public pure getPais : () ==> String
getPais() ==
(

 return pais;

);

-- Retorna a localidade
public pure getLocalidade : () ==> String
getLocalidade() ==
(
 return localidade;
);

-- Retorna a escolaridade

public pure getEscolaridade : () ==> Escolaridade
getEscolaridade() ==
(
 return escolaridade;
);

-- Editar Nome
public setNome: String ==> ()
setNome(newName) == nome := newName
pre newName <> undefined
post nome = newName;

-- Editar Email
public setEmail: String ==> ()
setEmail(newEmail) == email := newEmail
pre newEmail <> undefined
post email = newEmail;


-- Editar Telefone
public setTelefone: nat1 ==> ()
```

```
 setTelefone(newTelefone) == telefone := newTelefone
 pre newTelefone <> undefined
 post telefone = newTelefone;


 -- Editar Pais
 public setPais: String ==> ()
 setPais(newPais) == pais := newPais
 pre newPais <> undefined
 post pais = newPais;


 -- Editar Localidade
 public setLocalidade: String ==> ()
 setLocalidade(newLocalidade) == localidade := newLocalidade
 pre newLocalidade <> undefined
 post localidade = newLocalidade;

 -- Editar Escolaridade
 public setEscolaridade: Escolaridade ==> ()
 setEscolaridade(newEscolaridade) == escolaridade := newEscolaridade
 pre newEscolaridade <> undefined
 post escolaridade = newEscolaridade;

functions
-- TODO Define functiones here

traces
-- TODO Define Combinatorial Test Traces here

end Utilizador
```

Function or operation	Line	Coverage	Calls
Utilizador	24	100.0%	77
addInteresse	38	100.0%	42
addSkills	57	100.0%	42
getEmail	103	100.0%	94
getEscolaridade	140	100.0%	63
getIdade	86	100.0%	21
getInteresses	50	100.0%	42
getLocalidade	130	100.0%	42
getNome	76	100.0%	42
getPais	123	100.0%	42
getSexo	100	100.0%	21
getSkills	69	100.0%	42
getTelefone	93	100.0%	70
removeInteresse	44	100.0%	21
removeSkills	63	100.0%	21
setEmail	123	100.0%	21
setEscolaridade	170	100.0%	42
setLocalidade	164	100.0%	21
setNome	117	100.0%	21
setPais	158	100.0%	21
setTelefone	129	100.0%	21

Utilizador.vdmpp		100.0%	829