Hokify

December 27, 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;
 public Escolaridade = <Secundario> | <Licenciatura> | <Mestrado> | <Doutoramento>;

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(nome) == (return {trabalhos | trabalhos in set trabalhos & trabalhos.
    nomeSemelhante(nome)})
pre len nome > 0;


--Retorna os trabalhos por interesses
public pure getTrabalhosPorInteresses: String ==> Trabalhos
getTrabalhosPorInteresses(nome) == (
 dcl results: Trabalhos := {};


 for all tr in set trabalhos do
  if nome in set tr.getInteresses() then
   results := results union {tr};
 return results;
);


--Retorna os trabalhos por skills
public pure getTrabalhosPorSkills: String ==> Trabalhos
getTrabalhosPorSkills(nome) == (
 dcl results: Trabalhos := {};
 for all tr in set trabalhos do
  if nome in set tr.getSkills() then
   results := results union {tr};
 return results;
);



--Retorna os trabalhos por Escolaridade

public pure getTrabalhosPorEscolaridade: Escolaridade ==> Trabalhos
getTrabalhosPorEscolaridade(nome) == (
 dcl results: Trabalhos := {};
 for all tr in set trabalhos do
```

```
  if nome in set tr.getlistaEscolaridades() then
   results := results union {tr};
 return results;
);

--Retorna os trabalhos por Utilizador (Escolaridade,Skills,interesses)

public pure getTrabalhosPorUtilizador: Utilizador ==> Trabalhos
getTrabalhosPorUtilizador(usr) == (
 dcl results_escolaridade: Trabalhos := {};
 dcl results_skills: Trabalhos := {};
 dcl results_interesses: Trabalhos := {};
 dcl trabalhos_temp: Trabalhos := {};
 results_escolaridade := getTrabalhosPorEscolaridade(usr.getEscolaridade());

 for all skill in set usr.getSkills() do
  trabalhos_temp := getTrabalhosPorSkills(skill);
  for all skill_temp in set trabalhos_temp do
   if skill_temp not in set results_skills then
    results_skills := results_skills union {skill_temp};

 for all interesse in set usr.getInteresses() do
  trabalhos_temp := getTrabalhosPorInteresses(interesse);
  for all interesse_temp in set trabalhos_temp do
   if interesse_temp not in set results_interesses then
    results_interesses := results_interesses union {interesse_temp};
 return (results_escolaridade inter results_skills inter results_interesses);
);

--Retorna os trabalhos por pais
public pure getTrabalhosPorPais: String ==> Trabalhos
getTrabalhosPorPais(nome) == (
 dcl results: Trabalhos := {};
 for all tr in set trabalhos do
  if nome = tr.getPais() then
   results := results union {tr};
 return results;
);

--Retorna os trabalhos por localidade
public pure getTrabalhosPorLocalidade: String ==> Trabalhos
getTrabalhosPorLocalidade(nome) == (
 dcl results: Trabalhos := {};
 for all tr in set trabalhos do
  if nome = tr.getLocalidade() then
   results := results union {tr};
 return results;
);


--Retorna os utilizadres por interesses
public pure getUtilizadoresPorInteresses: String ==> Utilizadores
getUtilizadoresPorInteresses(nome) == (
 dcl results: Utilizadores := {};
 for all tr in set utilizadores do
  if nome in set tr.getInteresses() then
   results := results union {tr};
 return results;
);


--Retorna os utilizadres por skills
public pure getUtilizadoresPorSkills: String ==> Utilizadores
getUtilizadoresPorSkills(nome) == (
 dcl results: Utilizadores := {};
```

```
  for all tr in set utilizadores do
   if nome in set tr.getSkills() then
    results := results union {tr};
  return results;
 );


 --Retorna os utilizadres por Escolaridade
 public pure getUtilizadoresPorEscolaridade: Escolaridade ==> Utilizadores
 getUtilizadoresPorEscolaridade(nome) == (
  dcl results: Utilizadores := {};
  for all tr in set utilizadores do
   if nome in set tr.getlistaEscolaridades() then
    results := results union {tr};
  return results;
 );

 --Retorna os utilizadores por Trabalhos (Escolaridade,Skills,interesses)
 public pure getTrabalhosPorUtilizador: Trabalho ==> Utilizadores
 getTrabalhosPorUtilizador(trab) == (
  dcl results_escolaridade: Utilizadores := {};
  dcl results_skills: Utilizadores := {};
  dcl results_interesses: Utilizadores := {};
  dcl utilizadores_temp: Utilizadores := {};
  results_escolaridade := getUtilizadoresPorEscolaridade(trab.getEscolaridade());

  for all skill in set trab.getSkills() do
   utilizadores_temp := getUtilizadoresPorSkills(skill);
   for all skill_temp in set utilizadores_temp do
    if skill_temp not in set results_skills then
     results_skills := results_skills union {skill_temp};

  for all interesse in set trab.getInteresses() do
   utilizadores_temp := getUtilizadoresPorInteresses(interesse);
   for all interesse_temp in set utilizadores_temp do
    if interesse_temp not in set results_interesses then
     results_interesses := results_interesses union {interesse_temp};

  return (results_escolaridade inter results_skills inter results_interesses);
 );

 -- 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%	37
addSkills	22	100.0%	81
addTrabalhos	28	100.0%	243
addUtilizadores	22	100.0%	81
getSkills	34	100.0%	74
getTrabalhos	41	100.0%	74
getTrabalhosPorEscolaridade	76	100.0%	241
getTrabalhosPorInteresses	54	100.0%	213
getTrabalhosPorLocalidade	74	100.0%	24
getTrabalhosPorNome	59	100.0%	212
getTrabalhosPorPais	74	100.0%	29
getTrabalhosPorSkills	64	100.0%	213
getTrabalhosPorUtilizador	86	100.0%	56
getUtilizadores	34	100.0%	74
getUtilizadoresPorEscolaridade	147	100.0%	43
getUtilizadoresPorInteresses	127	100.0%	81
getUtilizadoresPorSkills	137	100.0%	52
mapEscolaridades	26	100.0%	243
sameTrabalho	41	100.0%	28
sameUser	33	100.0%	41
searchByName	59	100.0%	1347
Hokify.vdmpp		100.0%	3487

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%	19
assertTrue	12	100.0%	224
MyTestCase.vdmpp		100.0%	243

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 utilizador2 : Utilizador := new Utilizador("Fabiola","gmail@gmail.com",<Feminino>,<
     Doutoramento>,"Portugal","Lisboa",26,123123123);
 dcl utilizador3 : Utilizador := new Utilizador("Francisca","asd@asd.com",<Feminino>,<Secundario
     >,"Portugal","Lisboa",26,123132323);
 dcl sameuser : 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 trabalho2 : Trabalho := new Trabalho("Programador de c++","Apple","apple@apple.pt",<
     Doutoramento>,4562343434,"Portugal","Lisboa");
 dcl trabalho3 : Trabalho := new Trabalho("Programador de php","Apple2","apple2@apple.pt",<
     Secundario>,357864,"Portugal","Funchal");
 dcl sametrabalho : Trabalho := new Trabalho("Programador de java","Google","google@google.pt",<
     Licenciatura>,123456789,"Portugal","Porto");

 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);

 hokify.addUtilizadores(utilizador3);
 hokify.addTrabalhos(trabalho3);

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

 -- Testes para a pesquisa por nome do trabalho em que retorne os trabalhos
 assertEqual(hokify.getTrabalhosPorNome("Programador"),{trabalho,trabalho2,trabalho3});
 assertEqual(hokify.getTrabalhosPorNome("java"),{trabalho});
 assertEqual(hokify.getTrabalhosPorNome("c++"),{trabalho2});

 -- Testes para a pesquisa por interesses em que retorne os trabalhos
 trabalho.addInteresse("Informatica");
 trabalho.addInteresse("Cinema");
 assertEqual(hokify.getTrabalhosPorInteresses("Informatica"),{trabalho});
 assertEqual(hokify.getTrabalhosPorInteresses("Cinema"),{trabalho});
 assertTrue(not (hokify.getTrabalhosPorInteresses("Cinema")={trabalho2}));

 -- Testes para a pesquisa por skills em que retorne os trabalhos
 trabalho.addSkills("Java");
 trabalho.addSkills("VDM");
 assertEqual(hokify.getTrabalhosPorSkills("Java"),{trabalho});
 assertEqual(hokify.getTrabalhosPorSkills("VDM"),{trabalho});
```

```
 assertTrue(not (hokify.getTrabalhosPorSkills("Java")={trabalho2}));

 -- Testes para a pesquisa por localidade em que retorne os trabalhos
 assertEqual(hokify.getTrabalhosPorLocalidade("Porto"),{trabalho});
 assertEqual(hokify.getTrabalhosPorLocalidade("Lisboa"),{trabalho2});

 -- Testes para a pesquisa por pais em que retorne os trabalhos
 assertEqual(hokify.getTrabalhosPorPais("Portugal"),{trabalho,trabalho2,trabalho3});

 -- Testes para a escolaridade em que retorne os trabalhos
 assertEqual(hokify.getTrabalhosPorEscolaridade(<Secundario>),{trabalho3});
 assertEqual(hokify.getTrabalhosPorEscolaridade(<Licenciatura>),{trabalho,trabalho3});
 assertEqual(hokify.getTrabalhosPorEscolaridade(<Mestrado>),{trabalho,trabalho3});
 assertEqual(hokify.getTrabalhosPorEscolaridade(<Doutoramento>),{trabalho,trabalho2,trabalho3});

 -- Testes para procurar por trabalhos para um utilizador
 utilizador.addInteresse("Informatica");
 utilizador.addInteresse("Cinema");
 utilizador.addSkills("Java");
 utilizador.addSkills("VDM");
 assertEqual(hokify.getTrabalhosPorUtilizador(utilizador),{trabalho});

 utilizador2.addInteresse("Informatica");
 utilizador2.addInteresse("Cinema");
 utilizador2.addSkills("Java");
 utilizador2.addSkills("VDM");
 assertEqual(hokify.getTrabalhosPorUtilizador(utilizador2),{trabalho});

 -- Testes para a pesquisa por interesses em que retorne os utilizadores
 assertEqual(hokify.getUtilizadoresPorInteresses("Informatica"),{utilizador,utilizador2});
 utilizador2.removeInteresse("Cinema");
 assertEqual(hokify.getUtilizadoresPorInteresses("Cinema"),{utilizador});
 assertTrue(not (hokify.getUtilizadoresPorInteresses("Cinema")={}));

 -- Testes para a pesquisa por skills em que retorne os utilizadores
 assertEqual(hokify.getUtilizadoresPorSkills("Java"),{utilizador,utilizador2});
 utilizador2.removeSkills("VDM");
 assertEqual(hokify.getUtilizadoresPorSkills("VDM"),{utilizador});
 assertTrue(not (hokify.getUtilizadoresPorSkills("VDM")={}));

 -- Testes para a escolaridade em que retorne os utilizadores
 assertEqual(hokify.getUtilizadoresPorEscolaridade(<Secundario>),{utilizador,utilizador2,
     utilizador3});
 assertEqual(hokify.getUtilizadoresPorEscolaridade(<Licenciatura>),{utilizador,utilizador2});
 assertEqual(hokify.getUtilizadoresPorEscolaridade(<Mestrado>),{utilizador2});
 assertEqual(hokify.getUtilizadoresPorEscolaridade(<Doutoramento>),{utilizador2});

 -- Testes para procurar por utilizadores para um trabalho
 assertEqual(hokify.getTrabalhosPorUtilizador(trabalho),{utilizador});
 utilizador2.addInteresse("Cinema");
 utilizador2.addSkills("VDM");
 assertEqual(hokify.getTrabalhosPorUtilizador(trabalho),{utilizador,utilizador2});
 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%	7
TestPlayerFirst	10	100.0%	66
TestPlayerFirst	10	100.0%	66
TestTrabalhoFirst	59	100.0%	66
TestTrabalhoFirst	50	100.0%	66
main	37	100.0%	64
main	101	100.0%	7
TestAll.vdmpp		100.0%	342

4 Trabalho

```
class Trabalho
types
-- TODO Define types here
 public String = seq of char;
 public Escolaridade = <Secundario> | <Licenciatura> | <Mestrado> | <Doutoramento>;
 public ListaEscolaridade = set of Escolaridade;
 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 := {};
 private listaEscolaridade: ListaEscolaridade := {};

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;
 listaEscolaridades(escolaridade);
 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;



public listaEscolaridades: Escolaridade ==> ()
listaEscolaridades(tipo)==(
if tipo = <Secundario> then (
 listaEscolaridade := listaEscolaridade union {<Secundario>};

 listaEscolaridade := listaEscolaridade union {<Licenciatura>};
 listaEscolaridade := listaEscolaridade union {<Mestrado>};
 listaEscolaridade := listaEscolaridade union {<Doutoramento>};
)elseif tipo = <Licenciatura> then(
 listaEscolaridade := listaEscolaridade union {<Licenciatura>};
 listaEscolaridade := listaEscolaridade union {<Mestrado>};

 listaEscolaridade := listaEscolaridade union {<Doutoramento>};
)elseif tipo = <Mestrado> then(
 listaEscolaridade := listaEscolaridade union {<Mestrado>};
 listaEscolaridade := listaEscolaridade union {<Doutoramento>};
)else(
 listaEscolaridade := listaEscolaridade union {<Doutoramento>};
););


-- 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;
);

-- Retorna o listaEscolaridades
public pure getlistaEscolaridades : () ==> ListaEscolaridade

getlistaEscolaridades() ==

(
 return listaEscolaridade;
);


-- 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%	152
addInteresse	49	100.0%	32
addSkills	68	100.0%	32
getEmail	99	100.0%	80
getEntidade	93	100.0%	55
getEscolaridade	105	100.0%	24
getInteresses	61	100.0%	121
getLocalidade	123	100.0%	56
getMapEscolaridades	152	100.0%	69
getNome	87	100.0%	64
getPais	117	100.0%	37
getSkills	80	100.0%	121
getTelefone	111	100.0%	18
getlistaEscolaridades	151	100.0%	69
listaEscolaridades	51	100.0%	106
mapEscolaridades	51	100.0%	106
nomeSemelhante	172	100.0%	60
removeInteresse	55	100.0%	9
removeSkills	74	100.0%	9
setEmail	143	100.0%	9
setEntidade	137	100.0%	9
setEscolaridade	167	100.0%	9
setLocalidade	161	100.0%	9

setNome	131	100.0%	9
setPais	155	100.0%	9
setTelefone	149	100.0%	9
similarName	172	100.0%	9
Trabalho.vdmpp		100.0%	1292

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 ListaEscolaridade = set of Escolaridade;
 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 := {};
 private listaEscolaridade: ListaEscolaridade := {};

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;

  listaEscolaridades(escolaridade);
  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;


public listaEscolaridades: Escolaridade ==> ()
listaEscolaridades(tipo)==(

if tipo = <Doutoramento> then (
 listaEscolaridade := listaEscolaridade union {<Secundario>};
 listaEscolaridade := listaEscolaridade union {<Licenciatura>};
 listaEscolaridade := listaEscolaridade union {<Mestrado>};
 listaEscolaridade := listaEscolaridade union {<Doutoramento>};
)elseif tipo = <Mestrado> then(

 listaEscolaridade := listaEscolaridade union {<Secundario>};
 listaEscolaridade := listaEscolaridade union {<Licenciatura>};
 listaEscolaridade := listaEscolaridade union {<Mestrado>};
)elseif tipo = <Licenciatura> then(
 listaEscolaridade := listaEscolaridade union {<Secundario>};
 listaEscolaridade := listaEscolaridade union {<Licenciatura>};

)else(
 listaEscolaridade := listaEscolaridade union {<Secundario>};
);
);

-- 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;
 );

 -- Retorna o listaEscolaridades

 public pure getlistaEscolaridades : () ==> ListaEscolaridade

 getlistaEscolaridades() ==
 (
  return listaEscolaridade;
 );

 -- 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%	83
addInteresse	38	100.0%	102
addSkills	57	100.0%	102
getEmail	103	100.0%	140
getEscolaridade	140	100.0%	85
getIdade	86	100.0%	19
getInteresses	50	100.0%	226
getLocalidade	130	100.0%	38
getNome	76	100.0%	38
getPais	123	100.0%	38
getSexo	100	100.0%	19
getSkills	69	100.0%	226
getTelefone	93	100.0%	104
getlistaEscolaridades	169	100.0%	141
listaEscolaridades	55	100.0%	8
removeInteresse	44	100.0%	64
removeSkills	63	100.0%	32
setEmail	123	100.0%	19
setEscolaridade	170	100.0%	38
setLocalidade	164	100.0%	19
setNome	117	100.0%	19
setPais	158	100.0%	19
setTelefone	129	100.0%	19
Utilizador.vdmpp		100.0%	1598