LinkedIn

Relatório Final

Mestrado Integrado em
Engenharia Informática e Computação

Métodos Formais em Engenharia de Software

Turma 2, Tema 1:
Diogo Magalhães Moura - up201304068
Inês Alexandra Santos Carneiro - up201303501
Sérgio Augusto Pires Domingues - up201304367

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

28 de Dezembro de 2016

Índice

Descrição informal do projecto e lista de requisitos

Descrição informal do projecto

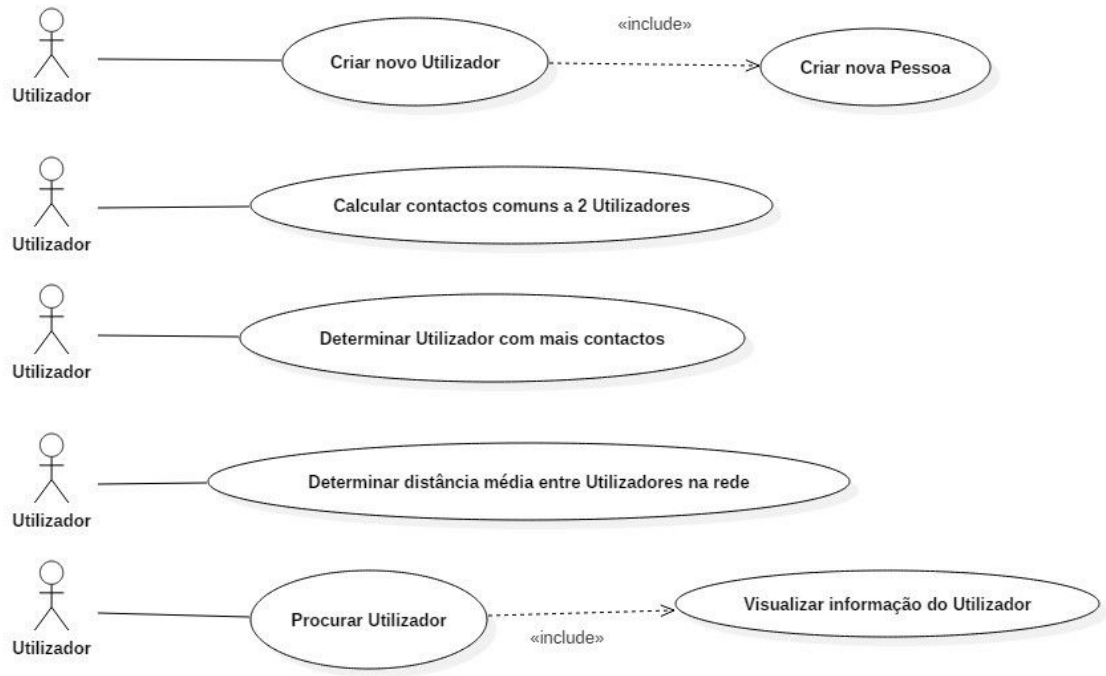
Neste projecto procurou-se modelar a plataforma LinkedIn mundialmente conhecida como sendo a maior rede social de profissionais de várias áreas. No modelo implementado à semelhança da plataforma, são representados profissionais os quais têm conexões com outros profissionais, possuem um registo académico, experiência laboral, entre outros parâmetros que poderão ser consultados nas secções seguintes. Ainda é permitido a estes profissionais atribuir recomendações em competências de outros profissionais.

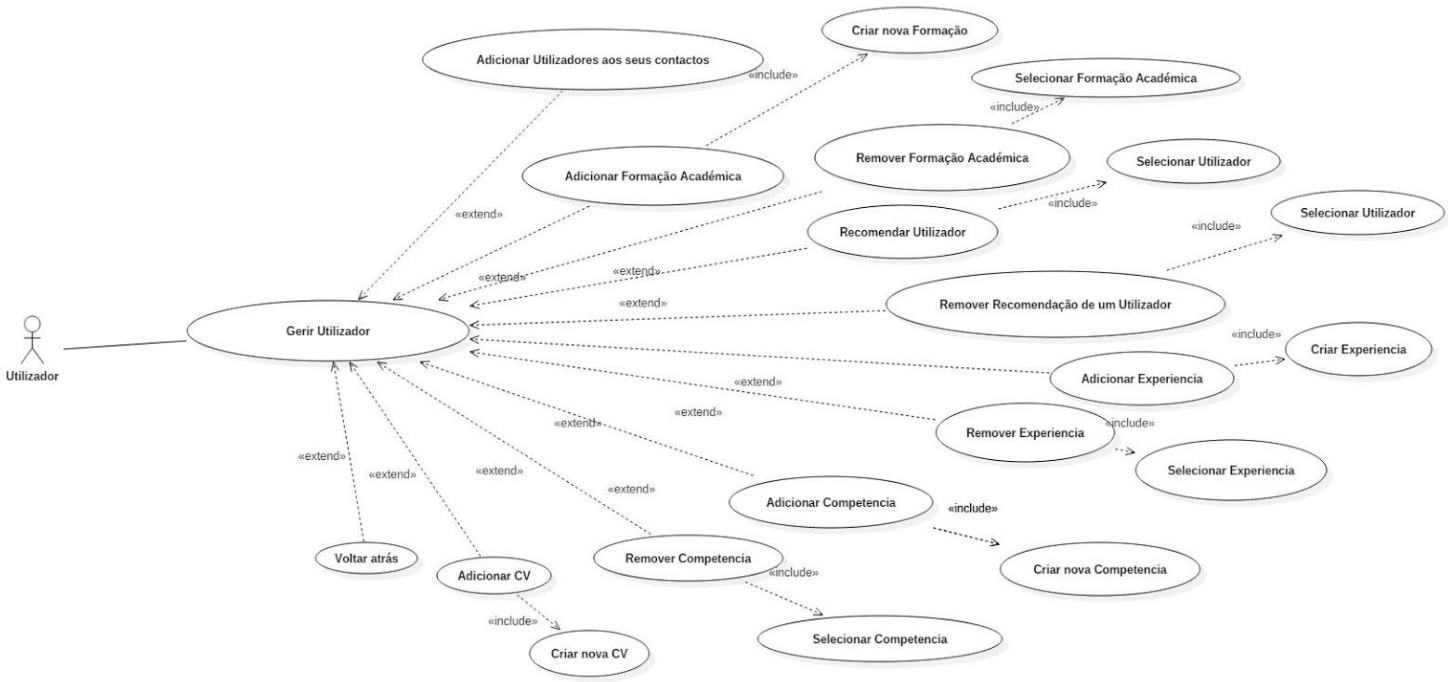
Lista de Requisitos

Id	**Prioridade**	**Descrição**
R1	Obrigatório	Adicionar pessoas à rede
R2	Obrigatório	Determinar a distância entre duas pessoas
R3	Obrigatório	Colocar CVs
R4	Obrigatório	Procurar pessoas
R5	Obrigatório	Calcular os contactos comuns entre duas pessoas
R6	Obrigatório	Determinar a pessoa com mais contactos
R7	Obrigatório	Calcular a distância média entre pessoas na rede
R8	Obrigatório	Adicionar uma pessoa como uma nova conexão
R9	Opcional	Adicionar e remover competências
R10	Opcional	Adicionar e remover formação académica
R11	Opcional	Adicionar e remover experiência de trabalho
R12	Opcional	Adicionar e remover recomendações de competências a uma pessoa

Visual UML model

Use Case model





Cenário	**Criar novo Utilizador**
Descrição	Cenário normal de criação de um novo utilizador na rede
Pré-condições	As pessoas são únicas
Pós-condições	A pessoa é criada como especificada
Etapas	1. O utilizador seleciona a opção de “criar novo utilizador” 2. O sistema pede ao utilizador para introduzir um nome 3. O utilizador introduz o nome do novo utilizador 4. O sistema retorna para um menu onde se pode gerir o utilizador
Excepções	Nenhuma

Cenário	**Pesquisa de pessoas na rede**
Descrição	Cenário normal de pesquisa de pessoas na rede
Pré-condições	Não especificado
Pós-condições	Não especificado
Etapas	1. O utilizador seleciona a opção de “Procurar Utilizador” 2. O sistema pede ao utilizador para introduzir o nome do utilizador a procurar 3. O utilizador introduz nome do Utilizador a procurar 4. O sistema retorna lista de utilizadores procurados para seleção
Excepções	Não é introduzido um nome válido

Cenário	**Calcular contatos comuns a duas pessoas**
Descrição	Cenário normal de pesquisa de contatos comuns entre duas pessoas
Pré-condições	As pessoas especificadas têm de estar contidas nos utilizadores da rede LinkedIn
Pós-condições	Não especificado
Etapas	1. O utilizador seleciona a opção de “Calcular contactos comuns a dois Utilizadores” 2. O sistema pede ao utilizador para introduzir selecionar os utilizadores 3. O utilizador os utilizadores ara os quais se pretende calcular

	os contactos comuns 4. O sistema retorna ao menu anterior após mostrar resposta ao utilizador
Excepções	Não são introduzidos nomes válidos

Cenário	**Determinar a pessoa com mais contatos**
Descrição	Cenário normal de pesquisa da pessoa com mais contatos
Pré-condições	O número total de pessoas na rede tem de ser maior que 0
Pós-condições	Não especificado
Etapas	1. O utilizador seleciona a opção de “Determinar Utilizador com mais contactos” 2. O sistema depois de mostrar o utilizador retorna ao menu anterior
Excepções	Nenhuma

Cenário	**Determinar distância entre duas pessoas na rede**
Descrição	Cenário normal de cálculo da distância entre duas pessoas
Pré-condições	Pessoas especificadas têm de estar contidas na rede
Pós-condições	Não especificado
Etapas	1. O utilizador seleciona a opção de “Determinar distância entre duas pessoas” 2. O sistema pede ao utilizador para selecionar os utilizadores 3. O utilizador seleciona os Utilizadores para os quais pretende calcular os contactos comuns 4. O sistema retorna ao menu anterior após mostrar resposta ao utilizador
Excepções	Não são introduzidos nomes válidos de utilizadores

Cenário	**Determinar distância média entre pessoas na rede**
Descrição	Cenário normal de cálculo da distância média entre duas pessoas na rede
Pré-condições	Não especificado

Pós-condições	Não especificado
Etapas	1. O utilizador seleciona a opção de “Determinar distância média entre pessoas na rede” 2. O sistema depois de mostrar o resultado retorna ao menu anterior
Excepções	Quando os utilizadores da rede não tem ligações Quando a rede não tem utilizadores

Cenário	**Adicionar utilizador à rede de contatos**
Descrição	Cenário normal para adicionar um utilizador à rede de contatos
Pré-condições	A pessoa especificada não pode estar contida na sua lista de contactos O utilizador não pode estar na lista de contactos da pessoa especificada A pessoa especificada não pode ser o próprio utilizador
Pós-condições	A pessoa especificada tem de estar adicionada à lista de contactos assim como o utilizador tem de estar adicionado à lista de contacto da pessoa especificada
Etapas	Isto pode ser feito após a criação de um novo utilizador ou após procurar um utilizador e selecionar o pretendido. 1. O utilizador seleciona a opção de “Adicionar contactos ao utilizador” 2. O sistema apresenta uma lista de possíveis utilizadores 3. O utilizador seleciona um utilizador ao qual pretende criar a conexão 4. O sistema retorna ao menu anterior
Excepções	Não é selecionado um utilizador na lista

Cenário	**Adicionar formação académica**
Descrição	Cenário normal para adicionar informação sobre a formação académica do utilizador
Pré-condições	Não especificado
Pós-condições	Não especificado
Etapas	1. O utilizador seleciona a opção de “Adicionar formação académica” 2. O sistema pede ao utilizador para introduzir o nome da instituição, a descrição da instituição e o ano em que a

	frequentou 3. O utilizador insere o nome da instituição, a descrição da instituição e o ano em que a frequentou 4. O sistema retorna ao menu anterior após adicionar a formação académica
Excepções	Nenhuma

Cenário	**Remover formação académica**
Descrição	Cenário normal para eliminar informação sobre a formação académica do utilizador
Pré-condições	A formação académica a eliminar tem de estar contida na lista de formações do utilizador
Pós-condições	A formação académica eliminada não pode estar contida na lista de formações do utilizador
Etapas	1. O utilizador seleciona a opção de "Remover formação académica" 2. O sistema retorna uma lista com todas as formações do utilizador 3. O utilizador seleciona a formação que pretende eliminar 4. O sistema retorna ao menu anterior após eliminar a formação selecionada
Excepções	O utilizador não seleciona uma formação académica válida

Cenário	**Adicionar recomendação a um utilizador**
Descrição	Cenário normal para adicionar uma recomendação a um utilizador selecionado
Pré-condições	O utilizador selecionado tem de estar na lista de conexões do utilizador A competência que quer recomendar tem de estar na lista de competências do utilizador O utilizador ainda não pode ter recomendado essa competência
Pós-condições	O utilizador tem uma recomendação na competência feita pelo utilizador
Etapas	1. O utilizador seleciona a opção de "Recomendar utilizador" 2. O sistema apresenta uma lista de possíveis utilizadores a recomendar 3. O utilizador seleciona um utilizador da lista 4. O sistema pede ao utilizador para inserir um nome para a

	competência a recomendar 5. O utilizador introduz um nome para a competência a recomendar 6. O sistema retorna ao menu anterior após adicionar a recomendação
Excepções	O utilizador selecionado não consta na lista

Cenário	**Remover recomendação a um utilizador**
Descrição	Cenário normal para remover uma recomendação a um utilizador selecionado
Pré-condições	O utilizador selecionado tem de estar na lista de conexões do utilizador O utilizador tem uma recomendação da competência feita pelo utilizador
Pós-condições	Não especificado
Etapas	1. O utilizador seleciona a opção de “Remover recomendação de utilizador” 2. O sistema apresenta uma lista com todas as recomendações 3. O utilizador seleciona a recomendação a eliminar 4. O sistema pede ao utilizador para introduzir o nome da competência a eliminar 5. O utilizador introduz o nome da competência 6. O sistema retorna ao menu anterior após eliminar a competência
Excepções	O utilizador selecionado não pertence á lista O nome da competência não coincide com nenhuma

Cenário	**Adicionar experiência a um utilizador**
Descrição	Cenário normal para adicionar uma experiência ao Utilizador
Pré-condições	Não especificado
Pós-condições	Não especificado
Etapas	1. O utilizador seleciona a opção de “Adicionar experiência” 2. O sistema pede ao utilizador para inserir o nome da experiência, o título da experiência e a descrição da mesma 3. O utilizador insere o nome da experiência, o título da experiência e a descrição da mesma 4. O sistema retorna ao menu anterior após adicionar a

	experiência ao utilizador
Excepções	Nenhuma

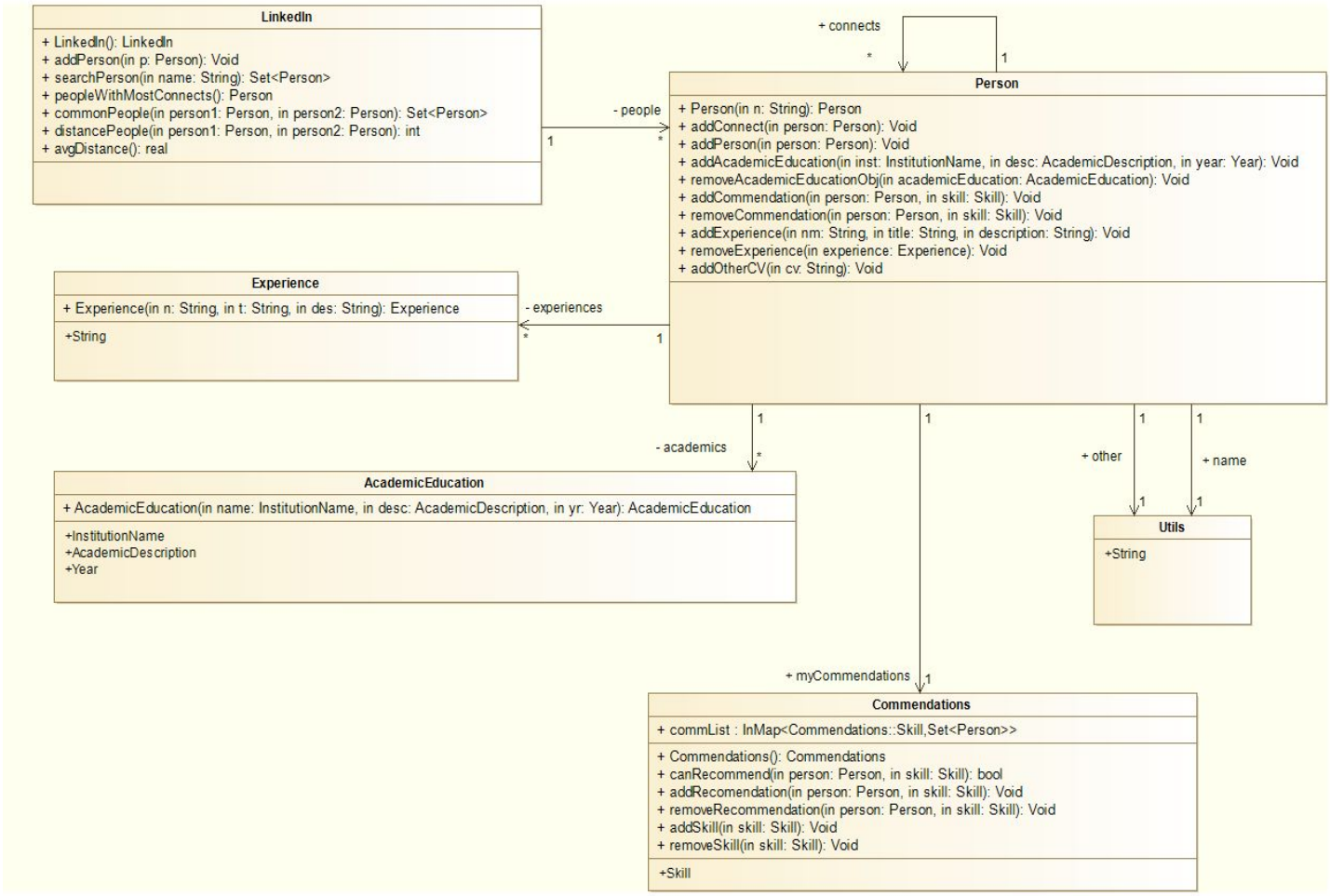
Cenário	**Remover experiência a um utilizador**
Descrição	Cenário normal para remover uma experiência selecionada ao Utilizador
Pré-condições	A experiência selecionada tem de estar presente na lista de experiências do utilizador
Pós-condições	A experiência selecionada não pdoe estar presente na lista de experiências do utilizador
Etapas	1. O utilizador seleciona a opção de "Remover Experiência" 2. O sistema retorna uma lista com todas as experiências do utilizador 3. O utilizador seleciona a experiência que pretende eliminar 4. O sistema retorna ao menu anterior após eliminar a experiência selecionada
Excepções	O utilizador seleciona um utilizador que não consta na lista

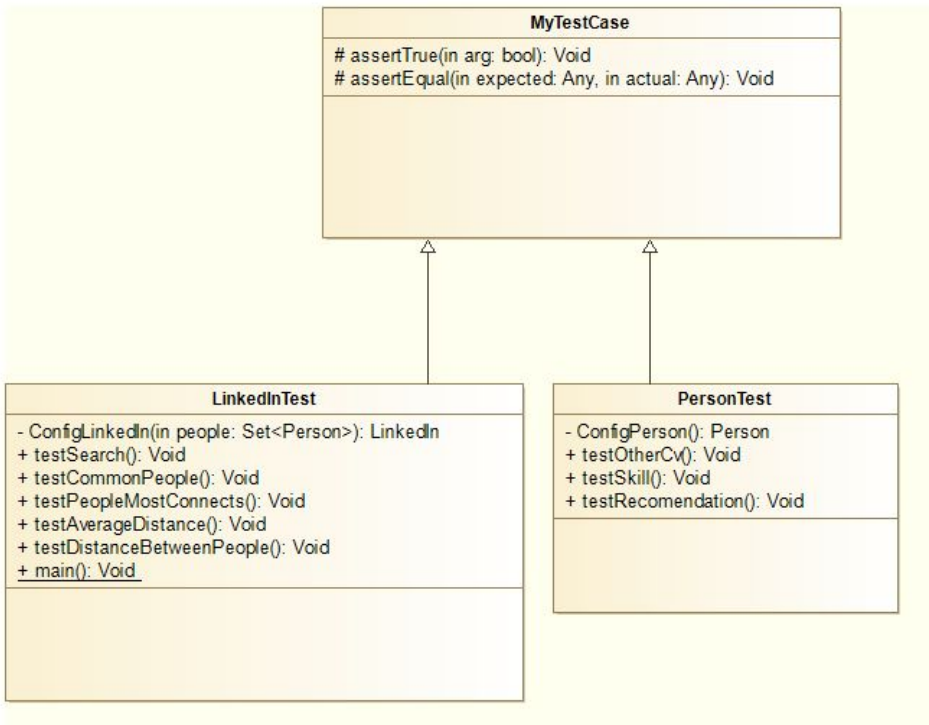
Cenário	**Adicionar competência a um utilizador**
Descrição	Cenário normal para adicionar uma competência a um utilizador selecionado
Pré-condições	O utilizador selecionado tem de estar na lista de conexões do utilizador
Pós-condições	Não especificado
Etapas	1. O utilizador seleciona a opção de "Adicionar competência" 2. O sistema pede ao utilizador para inserir um nome da competência 3. O utilizador insere nome da competência 4. O sistema retorna ao menu anterior após adicionar a competência ao utilizador
Excepções	Nenhuma

Cenário	**Remover competência a um utilizador**
Descrição	Cenário normal para adicionar uma recomendação a um utilizador selecionado
Pré-condições	O utilizador selecionado tem de estar na lista de conexões do utilizador
Pós-condições	Não especificado
Etapas	1. O utilizador seleciona a opção de “Remover Competência” 2. O sistema retorna uma lista com todas as competências do utilizador 3. O utilizador seleciona a competência que pretende eliminar 4. O sistema retorna ao menu anterior após eliminar a competência selecionada
Excepções	O utilizador seleciona uma competência que não está na lista

Cenário	**Adicionar um CV**
Descrição	Cenário normal para adicionar um CV
Pré-condições	Não especificado
Pós-condições	Não especificado
Etapas	1. O utilizador seleciona a opção de “Adicionar cv” 2. O sistema pede ao utilizador para introduzir uma texto com a cv que pretende adicionar 3. O Utilizador insere um texto com o cv que pretende adicionar 4. O sistema retorna ao menu anterior após adicionar a informação introduzida ao cv
Excepções	Nenhuma

Class model


LinkedIn
+ LinkedIn(): LinkedIn
+ addPerson(in p: Person): Void
+ searchPerson(in name: String): Set<Person>
+ peopleWithMostConnects(): Person
+ commonPeople(in person1: Person, in person2: Person): Set<Person>
+ distancePeople(in person1: Person, in person2: Person): int
+ avgDistance(): real
- people
1
*
+ connects
*
1
Person
+ Person(in n: String): Person
+ addConnect(in person: Person): Void
+ addPerson(in person: Person): Void
+ addAcademicEducation(in inst: InstitutionName, in desc: AcademicDescription, in year: Year): Void
+ removeAcademicEducationObj(in academicEducation: AcademicEducation): Void
+ addCommendation(in person: Person, in skill: Skill): Void
+ removeCommendation(in person: Person, in skill: Skill): Void
+ addExperience(in nm: String, in title: String, in description: String): Void
+ removeExperience(in experience: Experience): Void
+ addOtherCV(in cv: String): Void
Experience
+ Experience(in n: String, in t: String, in des: String): Experience
+String
- experiences
*
1
1
*
- academics
1
1
1
1
+ other
+ name
1
1
AcademicEducation
+ AcademicEducation(in name: InstitutionName, in desc: AcademicDescription, in yr: Year): AcademicEducation
+InstitutionName
+AcademicDescription
+Year
Utils
+String
+ myCommendations
1
Commendations
+ commList : InMap<Commendations::Skill,Set<Person>>
+ Commendations(): Commendations
+ canRecommend(in person: Person, in skill: Skill): bool
+ addRecomendation(in person: Person, in skill: Skill): Void
+ removeRecommendation(in person: Person, in skill: Skill): Void
+ addSkill(in skill: Skill): Void
+ removeSkill(in skill: Skill): Void
+Skill


MyTestCase
assertTrue(in arg: bool): Void
assertEqual(in expected: Any, in actual: Any): Void
LinkedInTest
- ConfigLinkedIn(in people: Set<Person>): LinkedIn
+ testSearch(): Void
+ testCommonPeople(): Void
+ testPeopleMostConnects(): Void
+ testAverageDistance(): Void
+ testDistanceBetweenPeople(): Void
+ main(): Void
PersonTest
- ConfigPerson(): Person
+ testOtherCv(): Void
+ testSkill(): Void
+ testRecomendation(): Void

Class	Descrição
LinkedIn	Classe principal que contém toda a lógica da rede social, é onde se encontram definidas as operações que estão disponíveis aos utilizadores
Person	Classe que define uma pessoa na rede social. Cada pessoa tem um nome, um conjunto de formações académicas, de experiências, de recomendações e de conexões. Possui também um CV.
Experience	Classe que define as experiências de um utilizador, cada experiência é formada por um nome, título e descrição
AcademicEducation	Classe que define a formação académica de um utilizador. A formação académica é definida por uma instituição, descrição e ano em que a frequentou.
Commendations	Classe que representa a lista de recomendações feita ao utilizador. Cada recomendação é um *map* formado por uma competência e um conjunto de pessoas
Utils	Classe com tipos definidos a serem usados noutras classes
LinkedInTest	Classe principal de testes. Testa todas as funcionalidades obrigatórias assim como as opcionais e as chama os testes para a classe pessoa definidos em *PersonTest*
PersonTest	Classe com testes relacionados com um utilizador da rede, como testes de recomendações, cv, experiências e formação académica.
MyTestCase	Classe retirada do exemplo *ReportVendingMachine* que implementa os métodos *assertTrue* e *assertEqual* usados nas classes-filhas (*LinkedInTest* e *PersonTest*)

Modelo Formal VDM++

Classe AcademicEducation

```
class AcademicEducation

types
	public InstitutionName = Utils`String;
	public AcademicDescription = Utils`String;
	public Year = nat1;

instance variables
	private institution : InstitutionName;
	private description : AcademicDescription;
	private date : Year;

operations
	-- Construtor Educacao Academica
	public AcademicEducation : InstitutionName * AcademicDescription * Year ==>
AcademicEducation
	AcademicEducation(name, desc, yr) == (
		institution := name;
		description := desc;
		date := yr;
	return self);

end AcademicEducation
```

Classe Commendations

```
class Commendations
types
	public Skill = Utils`String;
instance variables
	public commList: inmap Skill to set of Person := { |-> };
operations
	-- Construtor Recomendacoes
	public Commendations: () ==> Commendations
		Commendations() == (return self);

	--Verifica se uma pessoa pode recomendar uma competencia
	public pure canRecommend: Person * Skill ==> bool
```

```
		canRecommend(person,skill)==(
			return skill in set dom commList and
			person not in set commList(skill); --nao pode recomendar >1 vez
a mesma skill tal como no linkedin
		);

	--Adiciona uma recomendacao a uma competencia
	public addRecomendation: Person * Skill ==> ()
		addRecomendation(person,skill)==
		 commList:= commList ++ { skill |-> commList(skill) union {person}}
		pre dom commList <> {}	--o inmap não deve estar vazio
		post person in set commList(skill);

	--Remove uma recomendacao a uma competencia
	public removeRecommendation: Person * Skill ==> ()
		removeRecommendation(person,skill) ==
			commList(skill) := commList(skill) \ {person}
		pre skill in set dom commList
			and person in set commList(skill)
		post commList <> commList~
			and person not in set commList(skill);

	--Adiciona uma competencia
	public addSkill: Skill ==> ()
		addSkill(skill)==
			commList:= commList munion {skill |-> {} }
		pre skill not in set dom commList
		post commList <> commList~
		and skill in set dom commList;

	--Remove uma competencia
	public removeSkill: Skill ==> ()
		removeSkill(skill) ==
			commList := {skill} <-: commList
		pre skill in set dom commList
		post skill not in set dom commList;

end Commendations
```

Classe Experience

```
class Experience
types
	public String = Utils`String;
instance variables
	private name: String;
	private title: String;
	private description: String;
operations
```

```
	-- Construtor da experiencia
	public Experience: String * String * String ==> Experience
	Experience(n,t,des) == (name:=n;title:=t;description:=des;return self)
end Experience
```

Classe LinkedIn

```
class LinkedIn

instance variables
	public people: set of Person :={};
	inv people <> {} => verifyMaxConnect();
operations
	-- LinkedIn contrutor
	public LinkedIn: () ==> LinkedIn
	LinkedIn() == (return self);

	--Adiciona uma pessoa à rede do LinkedIn
	public addPerson: Person ==> ()
	addPerson(p) ==
		people := people union {p}
	pre p not in set people
	post p in set people;

	--Pesquisa de um utilizador pelo seu nome
	public searchPerson: Utils`String ==> set of Person   --TODO ranked search
and substring search
	searchPerson(name) ==
		return { person | person in set people & person.name = name};

	--Determinar a pessoa com mais contactos
	public pure peopleWithMostConnects: () ==> Person
	peopleWithMostConnects()==(
		dcl r:nat :=0;
		dcl mostConnectedPerson: [Person] :=nil;

		for all p in set people do
			(if card p.connects > r
			then( mostConnectedPerson := p;
				r := card p.connects;
				)
			);
		return mostConnectedPerson;  --help como conto as pessoas com mais
coneccoes
		)
		pre card people > 1;
```

```
		--Determinar os contactos em comum entre as duas pessoas passadas pelo
argumento
		public pure commonPeople: Person * Person ==> set of Person
		commonPeople(person1,person2) ==
			return person1.connects inter person2.connects
			pre person1 in set people and person2 in set people;

		--Calcular as distancias entre duas pessoas usando um pesquisa em
largura
		--Retorna -1 se as pessoas nao estiverem conectas em nenhum nivel
		--Retorna 0 se for a mesma pessoa
		--Retorna >0 caso contrário
		public distancePeople: Person * Person ==> int
		distancePeople(person1,person2) ==(
			dcl counter: nat1 := 1; --contador dos niveis que já passou
			dcl q: set of Person := person1.connects; --Pessoas a visitar
			dcl qVisited: set of Person := {}; --Pessoas já visitadas

			if person1 = person2 then return 0; --Se for a mesma pessoa

			-- enquanto a pessoa nao esta no set a pesquisar
			while person2 not in set q do (
				counter := counter + 1;

			-- Pesquisa do nivel
				for all person in set q do(
					qVisited:= qVisited union {person};
					q := q union person.connects \ qVisited;
				);

			-- evitar loop infinito
				if card q = 0 then
					return -1
			);

		--se for impossivel de alcancar retorna -1 caso contrario retorna o
numero de niveis
			if person2 not in set q then
				return -1
			else
				return counter
		)
		pre person1 in set people and person2 in set people;

		--Retorna a média das distancias na rede
		public avgDistance: () ==> real
			avgDistance() ==(
			dcl distTotal:nat:=0;
			dcl num:nat:=0;
			dcl temp:nat:=0;
			for all person in set people do(
```

```
                        for all connect in set people \ {person} do(
                                temp:=distancePeople(person,connect);
                                if temp >0 then(
                                distTotal:=temp+distTotal;
                                num:= num+1;
                                )
                        )
                );
                return distTotal/num;
        );

        public pure verifyMaxConnect:()==> bool
                verifyMaxConnect()==(
                        for all p in set people do (
                                if not p.connects subset people then return false
                        );
                        return true;
                );

end LinkedIn
```

Classe Person

```
class Person
instance variables
   public connects : set of Person := {};
   public name: Utils`String;
   public myCommendations : Commendations := new Commendations();
   public academics : set of AcademicEducation := {};
   public experiences : set of Experience := {};
   public other :Utils`String :="";
operations
       --Construtor Pessoa
       public Person: Utils`String ==> Person
       Person(n) == (name:=n; return self)
       post connects = {} and academics = {} and myCommendations.commList = {|->};

  --Adiciona uma conecao / Essa pessoa é adicionada as minhas coneccoes e
  --eu sou adicionado as coneccoes dela
  public addConnect:Person ==> ()
                addConnect(person)==(
                        addPerson(person);
                        person.addPerson(self);
                )
                pre person not in set connects and self not in set
person.connects
                post person in set connects and self in set person.connects;
```

```
  --Adiciona a pessoa as minhas coneccoes
  public addPerson:Person ==> ()
                    addPerson(person) ==
                    connects:= connects union {person}
                    pre person not in set connects
                    post person in set connects;

        --Adiciona educacao academica
        public addAcademicEducation: AcademicEducation`InstitutionName *
AcademicEducation`AcademicDescription * AcademicEducation`Year ==> AcademicEducation
                                    addAcademicEducation(inst,desc,year) ==(
                                    dcl aca : AcademicEducation :=new
AcademicEducation(inst,desc,year);
                                    academics:= academics union {aca};
                                    return aca;
                                    )
                                    post academics <> academics~ and card academics
>card academics~;

        --Remove educacao academica
        public removeAcademicEducationObj: AcademicEducation ==> ()
                removeAcademicEducationObj(academicEducation) ==
                        academics := academics \ {academicEducation}
                        pre academicEducation in set academics
                        post academics <> academics~
                                and academicEducation not in set academics;

        --Adiciona um recomendacao por parte da pessoa a um determinada competencia
        public addCommendation: Person * Commendations`Skill ==> ()
                                addCommendation(person,skill) ==
                                        myCommendations.addRecomendation(person,skill)
                                pre person in set connects
--so as conexoes podem recomendar
                                        and myCommendations.canRecommend(person,skill)
                                post not myCommendations.canRecommend(person,skill);

        --Remove uma recomendacao feita a uma pessoa
        public removeCommendation: Person * Commendations`Skill ==> ()
                removeCommendation(person,skill) ==
                        myCommendations.removeRecommendation(person,skill)
                        pre person in set connects;

        --Adiciona uma experiencia professional
        public addExperience: Experience`String * Experience`String *
Experience`String ==> Experience
                addExperience(nm,title,description) ==(
                        dcl exp :Experience := new Experience(nm,title,description);
                        experiences := experiences union {exp};
                        return exp;
                        )
```

```
                post experiences <> experiences~ and card experiences >card
experiences~;

        --Remove uma experiencia professional
        public removeExperience : Experience ==> ()
                removeExperience(experience) ==
                        experiences := experiences \ {experience}
                        pre experience in set experiences
                        post experience not in set experiences;

        --Adiciona outras carateristicas do curriculo
        public addOtherCV : Utils`String ==> ()
                addOtherCV(cv) ==
                        other:= other ^ cv
                        post len other > len other~;

end Person
```

Model validation

Classe 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
```

Classe LinkedInTest

```
class LinkedInTest is subclass of MyTestCase
	--linkedIn: LinkedIn := new LinkedIn();
operations
		--dcl diogo: Person := new Person("diogo");
		--dcl magalhaes: Person := new Person("magalhaes");
		--dcl moura: Person := new Person("moura");
```

```
        -- ines: Person := new Person("ines");
        --dcl alexandra: Person := new Person("alexandra");
        --dcl carneiro: Person := new Person("carneiro");
        --dcl sergio: Person := new Person("sergio");
        --dcl augusto: Person := new Person("augusto");
        -- domingues: Person := new Person("domingues");
private ConfigLinkedIn: set of Person ==> LinkedIn
ConfigLinkedIn(people) ==
(
        dcl linkedIn:LinkedIn := new LinkedIn();
        for all p in set people do
                linkedIn.addPerson(p);
        return linkedIn;
);

public testSearch : () ==> ()
testSearch() ==(
        dcl linkedIn : LinkedIn := ConfigLinkedIn({new Person("diogo")});
        dcl result:set of Person := linkedIn.searchPerson("diogo");
        assertTrue(card result = 1);
);

public testCommonPeople : () ==> ()
testCommonPeople() ==(
        dcl diogo: Person := new Person("diogo");
        dcl magalhaes: Person := new Person("magalhaes");
        dcl ines: Person := new Person("ines");
        dcl linkedIn : LinkedIn := ConfigLinkedIn({diogo,magalhaes,ines});
        diogo.addConnect(magalhaes);
        ines.addConnect(magalhaes);

        assertTrue(linkedIn.commonPeople(ines,diogo) = {magalhaes});
);

public testPeopleMostConnects : () ==> ()
testPeopleMostConnects() ==(
        dcl diogo: Person := new Person("diogo");
        dcl magalhaes: Person := new Person("magalhaes");
        dcl ines: Person := new Person("ines");
        dcl linkedIn : LinkedIn;
        linkedIn:=ConfigLinkedIn({diogo,magalhaes,ines});
        diogo.addConnect(magalhaes);
        diogo.addConnect(ines);

        assertEqual(diogo,linkedIn.peopleWithMostConnects());
);

public testAverageDistance : () ==> ()
testAverageDistance() ==(
        dcl diogo: Person := new Person("diogo");
        dcl magalhaes: Person := new Person("magalhaes");
```

```
            dcl ines: Person := new Person("ines");
            dcl linkedIn : LinkedIn := ConfigLinkedIn({diogo,magalhaes,ines});
            diogo.addConnect(magalhaes);
            diogo.addConnect(ines);
            magalhaes.addConnect(ines);

            assertEqual(1,linkedIn.avgDistance());
      );

      public testDistanceBetweenPeople : () ==> ()
      testDistanceBetweenPeople() ==(
            dcl diogo: Person := new Person("diogo");
            dcl magalhaes: Person := new Person("magalhaes");
            dcl ines: Person := new Person("ines");
            dcl moura: Person := new Person("moura");
            dcl linkedIn : LinkedIn :=
ConfigLinkedIn({diogo,magalhaes,ines,moura});
            diogo.addConnect(magalhaes);
            magalhaes.addConnect(ines);

            assertEqual(2,linkedIn.distancePeople(diogo,ines));
            assertEqual(0,linkedIn.distancePeople(diogo,diogo));
            assertEqual(1,linkedIn.distancePeople(diogo,magalhaes));
            assertEqual(-1,linkedIn.distancePeople(diogo,moura));
      );


  public static main: () ==> ()
  main() == (
            dcl liTest : LinkedInTest := new LinkedInTest();
            dcl personTest : PersonTest := new PersonTest();
            liTest.testSearch();
            liTest.testCommonPeople();
            liTest.testPeopleMostConnects();
            liTest.testDistanceBetweenPeople();
            liTest.testAverageDistance();
            personTest.testOtherCv();
            personTest.testSkill();
            personTest.testRecomendation();
            personTest.testAcadExp();
            personTest.testExp();
  );

  public static testPersonNotInNetwork: () ==> ()
      testPersonNotInNetwork()==(
            dcl diogo: Person := new Person("diogo");
                  dcl magalhaes: Person := new Person("magalhaes");
                  dcl linkedIn : LinkedIn := new LinkedIn();
                  diogo.addConnect(magalhaes);

                  linkedIn.addPerson(diogo);
```

```
	);

	public static testCommomPeopleNotInNetwork: () ==> ()
		testCommomPeopleNotInNetwork()==(
				dcl magalhaes: Person := new Person("magalhaes");
				dcl moura: Person := new Person("moura");
				dcl common : [set of Person] := nil;
				dcl linkedIn : LinkedIn := new LinkedIn();

				common:=linkedIn.commonPeople(magalhaes,moura);

		);

	public static testMostConnectSmallNetwork: () ==> ()
		testMostConnectSmallNetwork()==(
				dcl moura: Person := new Person("moura");
				dcl most : [Person] := nil;
				dcl linkedIn : LinkedIn := new LinkedIn();
				linkedIn.addPerson(moura);
				most:=linkedIn.peopleWithMostConnects();
		);

end LinkedInTest
```

Clase PersonTest

```
class PersonTest is subclass of MyTestCase

operations
	private ConfigPerson: () ==> Person
	ConfigPerson() ==
		return new Person("teste");

	public testOtherCv:() ==> ()
	testOtherCv()==(
		dcl p : Person := ConfigPerson();
		p.addOtherCV("ola");
		assertEqual("ola",p.other);
		p.addOtherCV("1");
		assertEqual("ola1",p.other);
	);

	public testSkill:() ==> ()
	testSkill() ==(
		dcl p : Person := ConfigPerson();
		p.myCommendations.addSkill("JS");
		assertTrue("JS" in set dom p.myCommendations.commList);
		p.myCommendations.removeSkill("JS");
		assertTrue("JS" not in set dom p.myCommendations.commList);
```

```
	);

	public testRecomendation:() ==> ()
	testRecomendation()==(
		dcl p : Person := ConfigPerson();
		dcl p1 : Person := new Person("x");
		p.addConnect(p1);

		p.myCommendations.addSkill("JS");
		p.addCommendation(p1,"JS");
		assertEqual(p.myCommendations.commList("JS"),{p1});

		p.removeCommendation(p1,"JS");
		assertEqual(p.myCommendations.commList("JS"),{});
	);

	public testAcadExp:() ==> ()
		testAcadExp()==(
			dcl p : Person := ConfigPerson();
			dcl aca : AcademicEducation:=
p.addAcademicEducation("FEUP","MIEIC",2018);
			assertEqual(p.academics,{aca});
			p.removeAcademicEducationObj(aca);
			assertEqual(p.academics,{});
		);

	public testExp:() ==> ()
		testExp()==(
			dcl p : Person := ConfigPerson();
			dcl exp : Experience:= p.addExperience("MUNDO","CEO","O BOSS");
			assertEqual(p.experiences,{exp});
			p.removeExperience(exp);
			assertEqual(p.experiences,{});
		);

end PersonTest
```

Verificação do Modelo

Exemplo de verificação de domínio

Apresenta-se de seguida um exemplo de uma *proof obligation* gerada pelo Overture:

Nº	Nome da PO	Tipo
2	Commendations`canRecomend	legal map application

O código analisado surge a seguir com a *map application* sombreada:

```
public pure canRecommend: Person * Skill ==> bool
	canRecommend(person,skill)==(
			return skill in set dom commList and
			person not in set commList(skill);
	);
```

O map **commlist** corresponde a um map **‘{skill |-> {person} }’** e a verificação **‘person not in set commList(skill)’** assegura que uma ***person*** não recomenda a mesma ***skill*** mais do que uma vez (à luz do que acontece na plataforma LinkedIn), isto verificando que a ***person*** não está mapeada para a ***skill*** em causa.

Example of invariant verification

Apresenta-se de seguida outro exemplo de uma *proof obligation* gerada pelo Overture:

Nº	Nome da PO	Tipo
2	LinkedIn`addPerson	state invariant holds

O código analisado surge a seguir com a mudanças de estado relevante sombreada:

```
public addPerson: Person ==> ()
	addPerson(p) ==
		people := people union {p}
	pre p not in set people
	post p in set people;
```

A invariante que se encontra a ser analisada apresenta-se a seguir:

```
inv verifyMaxConnect();
public pure verifyMaxConnect:()==> bool
      verifyMaxConnect()==(
            for all p in set people do (
                  if not p.connects subset people then return false);
                        return true;
                  );
```

Esta invariante permite garantir que durante a execução do programa se tenha de verificar sempre que uma pessoa é adicionada, que as conexões desta existem no LinkedIn ou seja as conexões de uma pessoa têm de ser um *subset* do '`set people`' (*set* correspondente à estrutura que armazena todas as pessoas presentes num determinado objecto LinkedIn ('`set people`')

Geração de Código

Para gerar o código apenas seguimos os passos indicados no tutorial sugerido no moodle.
O código gerado obtido foi executado sem erros. Para facilitar a apresentação do projecto apenas foi adicionado um ficheiro java com a lógica de todos os menus. Não foi alterada qualquer classe gerada pelo Overture. Foi adicionado o ficheiro de Menus.java em anexo para consulta, Anexo 1.

Conclusões

Todos os requisitos foram implementados no projecto.
Caso houvesse mais tempo para desenvolver o projecto seria interessante desenvolver mais a rede social e adicionar mais funcionalidades assim como aumentar o número dos testes feitos.
O projecto foi dividido igualmente por todos os elementos do grupo e na sua totalidade incluindo o relatório levou aproximadamente 50 horas a ser desenvolvido.

Referências

1. Slides disponibilizados nas aulas
2. Overture tool Quick start exercise,https://docs.google.com/document/d/1cf1cn2qbELCMaHZcBut__0dQL9HGnHlk-l7dmM5q5kw/pub
3. Validated Designs for Object-oriented Systems, J. Fitzgerald, P.G. Larsen, P. Mukherjee, N. Plat, M. Verhoef, Springer, 2005
4. VDM-10 Language Manual, Peter Gorm Larsen et al, Overture Technical Report Series No. TR-001, March 2014
5. Overture tool web site, http://overturetool.org

Anexo

Anexo 1:

```
package LinkedIn;

import com.sun.org.apache.xpath.internal.SourceTree;
import org.overture.codegen.runtime.Utils;
import org.overture.codegen.runtime.VDMSet;

import java.util.Iterator;
import java.util.Map;
import java.util.Scanner;

/**
* Created by inesa on 02/01/2017.
*/
public class Menus {

   private static int state = 0;
   static LinkedIn li = new LinkedIn();

   private static void mainMenu() {
       System.out.println(
               "Para continuar por favor insira o numero correspondente ao que
pretende fazer seguido de Enter: ");
       System.out.println("1. Criar novo utilizador");
       System.out.println("2. Procurar Utilizador");
       System.out.println("3. Calcular contactos comuns a dois Utilizadores");
       System.out.println("4. Determinar Utilizador com mais contactos");
       System.out.println("5. Determinar distância média entre pessoas na rede");
       System.out.println("6. Determinar distância entre duas pessoas");
       System.out.println("0. Exit");
   }
```

```
    private static String usingStringMenu() {
        Scanner scanner = new Scanner(System.in);
        String readString = scanner.nextLine();
        while (readString.isEmpty() && readString != null) {
            if (scanner.hasNextLine()) {
                readString = scanner.nextLine();
            } else {
                readString = null;
            }
        }
        return readString;
    }

    private static void secondMenu(Person p) {
        Scanner scan = new Scanner(System.in);

        System.out.println("Para continuar por favor insira o numero correspondente
ao que pretende fazer: ");
        System.out.println("0. Voltar ao menu principal");
        System.out.println("1. Adicionar contactos ao utilizador");
        System.out.println("2. Adicionar formação académica");
        System.out.println("3. Remover formação académica");
        System.out.println("4. Recomendar utilizador");
        System.out.println("5. Remover recomendação de utilizador");
        System.out.println("6. Adicionar experiência");
        System.out.println("7. Remover experiência");
        System.out.println("8. Adicionar competência");
        System.out.println("9. Remover competencia");
        System.out.println("10. Adicionar cv");


        int tempState = scan.nextInt();

        switch (tempState) {
            case 0:
                tempState = -1;
                break;
            case 1:
                if (li.people.size() <= 1) {
                    /*Person user = new Person(name);
                    li.addPerson(user);
                    p.addConnect(user);*/
                    System.out.println("Por favor crie mais utilizadores para poder
criar associações");
                } else {

                    System.out.println(
                            "Para continuar por favor insira o numero correspondente
ao utilizador a que pretende conectar:");
                    System.out.println(
                            "Caso pretenda um utilizador que não na lista introduza
-1:");

                    printPeopleIndexed(li.people);
```

```
                    int i = 1;
                    int index = scan.nextInt();

                    if (index != -1) {
                        for (Iterator<Person> it = li.people.iterator();
it.hasNext(); ) {
                            Person person = it.next();
                            if (i == index) {
                                if (person != p)
                                    p.addConnect(person);
                                else
                                    System.out.println("Não pode connectar o
utilizador a si proprio");
                                break;
                            }
                            i++;
                        }
                    }
                }

                secondMenu(p);
                break;
            case 2:
                System.out.println(
                        "Para continuar por favor insira o nome da instituição que
pretende adicionar seguido de Enter: ");
                String inst = usingStringMenu();
                System.out.println(
                        "Para continuar por favor insira uma descrição da
instituição que pretende adicionar seguido de Enter: ");
                String desc = usingStringMenu();
                System.out.println(
                        "Para continuar por favor insira o ano em que frequentou a
instituição que pretende adicionar seguido de Enter: ");
                int year = scan.nextInt();
                p.addAcademicEducation(inst, desc, year);

                secondMenu(p);
                break;
            case 3:
                if (p.academics.size() <= 0) {
                    System.out.println("Utilizador sem formação académica.");
                } else {
                    //TODO: implementar todos os eliminar
                    System.out.println(
                            "Para continuar por favor insira o numero correspondente
á formação que pretende eliminar:");
                    int i = 1;

                    for (Iterator<AcademicEducation> it = p.academics.iterator();
it.hasNext(); ) {
                        AcademicEducation acad = it.next();
                        System.out.println(i + ". " + acad);
```

```
                    i++;
                }

                i = 1;
                int academic = scan.nextInt();

                for (Iterator<AcademicEducation> it = p.academics.iterator();
it.hasNext(); ) {
                    AcademicEducation acad = it.next();
                    if (i == academic) {
                        p.removeAcademicEducationObj(acad);
                        break;
                    }
                    i++;
                }
            }

            secondMenu(p);
            break;
        case 4:
            if (li.people.size() <= 1) {
                System.out.println("Por favor crie mais utilizadores para poder
criar recomendações");
            } else {
                System.out.println(
                        "Para continuar por favor insira o numero correspondente
ao utilizador que pretende recomendar:");

                printPeopleIndexed(li.people);

                int i = 1;
                int index = scan.nextInt();

                for (Iterator<Person> it = li.people.iterator(); it.hasNext(); )
{
                    Person person = it.next();
                    if (i == index) {
                        System.out.println("Para continuar por favor insira o
nome da competência do utilizador, " + person + ", :");
                        String skill = usingStringMenu();
                        person.addCommendation(p, skill);

                        break;
                    }
                    i++;
                }
            }

            secondMenu(p);
            break;
        case 5:
            if (li.people.size() <= 1) {
                System.out.println("Por favor crie mais utilizadores para poder
remover recomendações");
```

```
                } else {
                    System.out.println(
                            "Para continuar por favor insira o numero correspondente
ao utilizador a que pretende remover a recomendaçao:");

                    printPeopleIndexed(li.people);

                    int i = 1;
                    int index = scan.nextInt();

                    for (Iterator<Person> it = li.people.iterator(); it.hasNext(); )
{
                        Person person = it.next();
                        if (i == index) {
                            System.out.println("Para continuar por favor insira o
nome da competência do utilizador, " + person + ", :");
                            String skill = usingStringMenu();
                            person.removeCommendation(p, skill);

                            break;
                        }
                        i++;
                    }
                }

                secondMenu(p);
                break;
            case 6:
                System.out.println(
                        "Para continuar por favor insira o nome da experiencia que
pretende adicionar seguido de Enter: ");
                String expName = usingStringMenu();
                System.out.println(
                        "Para continuar por favor insira um titulo para a
experiencia que pretende adicionar seguido de Enter: ");
                String title = usingStringMenu();
                System.out.println(
                        "Para continuar por favor insira uma descrição para a
experiencia que pretende adicionar seguido de Enter: ");
                String des = usingStringMenu();

                p.addExperience(expName, title, des);
                secondMenu(p);
                break;
            case 7:
                if (p.experiences.size() <= 0) {
                    System.out.println("Utilizador sem experiencias para remover.");
                } else {

                    System.out.println(
                            "Para continuar por favor insira o numero correspondente
á experiencia que pretende remover do Utilizador:");

                    int i=1;
```

```
                    for (Iterator<Experience> it = p.experiences.iterator();
it.hasNext(); ) {
                        Experience exp = it.next();
                        System.out.println(i + ". " + exp);
                        i++;
                    }

                    i = 1;
                    int index = scan.nextInt();

                    for (Iterator<Experience> it = p.experiences.iterator();
it.hasNext(); ) {
                        Experience exp = it.next();
                        if (i == index) {
                            p.removeExperience(exp);
                            break;
                        }
                        i++;
                    }
                }

                secondMenu(p);
                break;
            case 8:
                System.out.println(
                        "Para continuar por favor insira o nome da competência que
pretende adicionar seguido de Enter: ");
                String skill = usingStringMenu();
                p.myCommendations.addSkill(skill);
                secondMenu(p);
                break;
            case 9:
                if (p.myCommendations.commList.size() <= 0) {
                    System.out.println("Utilizador sem competencias para remover.");
                    secondMenu(p);
                } else {

                    Scanner sc = new Scanner(System.in);

                    System.out.println(
                            "Para continuar por favor insira o numero correspondente
a competencia que pretende remover :");
                    int i = 1;

                    for (Iterator it =
p.myCommendations.commList.entrySet().iterator(); it.hasNext(); ) {
                        Map.Entry entry = (Map.Entry) it.next();
                        System.out.println(i + ". " + entry);
                    }

                    i = 1;
                    int index = sc.nextInt();
```

```
                    for (Iterator it =
p.myCommendations.commList.entrySet().iterator(); it.hasNext(); ) {
                        Map.Entry entry = (Map.Entry) it.next();
                        if (i == index) {
                            p.myCommendations.commList.remove(entry.getKey());
                            secondMenu(p);
                            break;
                        }
                        i++;
                    }
                }

                break;
            case 10:
                System.out.println(
                        "Para continuar por favor insira um texto com a CV que
pretende adicionar seguido de Enter: ");
                String cv = usingStringMenu();
                p.addOtherCV(cv);

                secondMenu(p);
                break;
            default:
                break;
        }

    }


    private static void thirdMenu(VDMSet result) {

        Scanner scan = new Scanner(System.in);

        System.out.println(
                "Para continuar por favor insira o numero correspondente ao
Utilizador que pretende selecionar :");
        int i = 1;

        printPeopleIndexed(result);

        i = 1;
        int index = scan.nextInt();

        for (Iterator<Person> it = result.iterator(); it.hasNext(); ) {
            Person p = it.next();
            if (i == index) {
                secondMenu(p);
                break;
            }
            i++;
        }

    }
```

```
    public static void printPeopleIndexed(VDMSet list) {
        int i = 1;
        for (Iterator<Person> it = list.iterator(); it.hasNext(); ) {
            Person p = it.next();
            String user = "";
            user += i + ". " + "Name: " + p.name + " ";

            if (p.connects.size() > 0) {
                user += "Connects: ";

                for (Iterator<Person> j = p.connects.iterator(); j.hasNext(); ) {
                    Person person = j.next();
                    user += person.name + "; ";
                }
            }

            user += "Recomendations: " + p.myCommendations;

            if (p.academics.size() > 0) {
                user += Utils.toString(p.academics);
            }

            if (p.experiences.size() > 0) {
                user += Utils.toString(p.experiences);
            }

            if (p.other != "") {
                user += Utils.toString(p.other);
            }


            System.out.println(user + "\n");
            i++;
        }
    }

    public static void printPeople(VDMSet list) {
        for (Iterator<Person> it = list.iterator(); it.hasNext(); ) {
            Person person = it.next();
            System.out.println("Name: " + person.name);
            if (person.connects.size() > 0) {
                System.out.print("Connects: ");

                for (Iterator<Person> i = person.connects.iterator(); i.hasNext(); )
{

                    Person p = i.next();
                    System.out.print(p.name + "; ");
                }
                System.out.print("\n");
            }
            System.out.println("Recomendations: " + person.myCommendations);

            if (person.academics.size() > 0) {
```

```
                System.out.println(Utils.toString(person.academics));
            }

            if (person.experiences.size() > 0) {
                System.out.println(Utils.toString(person.experiences));
            }

            if (person.other != "") {
                System.out.println(Utils.toString(person.other));
            }


            System.out.println("\n");
        }
    }

    public static void main(String[] arg) {

        Scanner scan = new Scanner(System.in);

        System.out.println("Bem vindo ao LinkedIn! :D ");

        while (state != -1) {

            if (li.people.size() > 0) {
                System.out.println("A nossa rede é formada pelos utilizadores: ");
                printPeople(li.people);
            }

            mainMenu();
            state = scan.nextInt();

            switch (state) {
                case 0:
                    state = -1;
                    break;
                case 1:
                    System.out.println("Para continuar por favor insira o nome do
novo utilizador seguido de Enter: ");
                    String name = usingStringMenu();
                    Person p = new Person(name);
                    li.addPerson(p);
                    secondMenu(p);
                    break;
                case 2:
                    System.out.println(
                            "Para continuar por favor insira o nome do utilizador a
procurar seguido de Enter: ");
                    name = usingStringMenu();
                    VDMSet result = li.searchPerson(name);

                    System.out.println("Resultado da pesquisa para " + name);

                    printPeople(result);
```

```
                    thirdMenu(result);

                    break;
                case 3:
                    Person name1 =  new Person("");
                    Person name2 =  new Person("");

                    System.out.println("Para continuar por favor selecione o
utilizador seguido de Enter: ");

                    printPeopleIndexed(li.people);

                    int j = 1;
                    int index = scan.nextInt();

                    for (Iterator<Person> it = li.people.iterator(); it.hasNext(); )
{
                        Person person = it.next();
                        if (j == index) {
                            name1 = person;
                            break;
                        }
                        j++;
                    }

                    System.out.println("Para continuar por favor selecione outro
utilizador seguido de Enter: ");

                    printPeopleIndexed(li.people);

                    j = 1;
                    index = scan.nextInt();

                    for (Iterator<Person> it = li.people.iterator(); it.hasNext(); )
{
                        Person person = it.next();
                        if (j == index) {
                            name2 = person;
                            break;
                        }
                        j++;
                    }


                    VDMSet res = li.commonPeople(name1,name2);

                    System.out.println("Os contactos comuns aos utilizadores " +
name1.name + " e " + name2.name + " sao: ");

                    for (Iterator iterator_1 = res.iterator(); iterator_1.hasNext();
) {
                        Person per = ((Person) iterator_1.next());
                        System.out.println(per.name);
```

```
                    }

                    System.out.println("");
                    break;
                case 4:
                    Person person = li.peopleWithMostConnects();

                    if(person != null) {
                        System.out.println("A pessoa com mais contactos da rede é:
");


                        System.out.println("Name: " + person.name);
                        if (person.connects.size() > 0) {
                            System.out.print("Connects: ");

                            for (Iterator<Person> i = person.connects.iterator();
i.hasNext(); ) {
                                Person ps = i.next();
                                System.out.print(ps.name + "; ");
                            }
                            System.out.print("\n");
                        }
                        System.out.println("Recomendations: " +
person.myCommendations);

                        if (person.academics.size() > 0) {
                            System.out.println(Utils.toString(person.academics));
                        }

                        if (person.experiences.size() > 0) {
                            System.out.println(Utils.toString(person.experiences));
                        }

                        if (person.other != "") {
                            System.out.println(Utils.toString(person.other));
                        }

                        System.out.println("\n");
                    }
                    else {
                        if(li.people.size() > 0)
                            System.out.println("Não há nenhum utilizador com mais
contactos que outro");
                        else
                            System.out.println("Crie mais utilizadores para poder
continuar");
                    }
                    break;
                case 5:
                    Number dis = li.avgDistance();

                    System.out.println("A distância média entre os utilizadores na
rede é: ");
                    System.out.println(dis);
```

```
                    System.out.println("");
                    break;

                case 6:
                    name1 =  new Person("");
                    name2 =  new Person("");

                    System.out.println("Para continuar por favor selecione o
utilizador seguido de Enter: ");

                    printPeopleIndexed(li.people);

                    int i = 1;
                    index = scan.nextInt();

                    for (Iterator<Person> it = li.people.iterator(); it.hasNext(); )
{
                        person = it.next();
                        if (i == index) {
                            name1 = person;
                            break;
                        }
                        i++;
                    }

                    System.out.println("Para continuar por favor selecione outro
utilizador seguido de Enter: ");

                    printPeopleIndexed(li.people);

                    i = 1;
                    index = scan.nextInt();

                    for (Iterator<Person> it = li.people.iterator(); it.hasNext(); )
{
                        person = it.next();
                        if (i == index) {
                            name2 = person;
                            break;
                        }
                        i++;
                    }
                    dis = li.distancePeople(name1,name2);

                    System.out.println("A distância média entre os utilizadores " +
name1.name + " e " + name2.name + " é: ");
                    System.out.println(dis);

                    System.out.println("");
                    break;
            }
```

```
            }

            return;
        }
    }
```