Magic Stay

Relatório Final

Mestrado Integrado em Engenharia Informática e Computação

Métodos Formais de Engenharia de Software

Turma 5 - Grupo 11
Luís Cruz - up201303248@fe.up.pt

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, s/n, 4200-465 Porto, Portugal

2 de Janeiro de 2018

Conteúdo

1 Descrição do Sistema e Lista de Requisitos

1.1 Descrição do Sistema

Baseado no já existente site *MagicStay.com* o projeto desenvolvido no âmbito na cadeira de Métodos Formais de Engenharia de Software foi criado com o intuito de facilitar a procura de estadias próximas da realização de eventos por parte dos utilizadores e dos organizadores dos próprios eventos.

A procura de estações hoteleiras pode ser realizada mediante diversos parâmetros tendo também sido criada uma forma de procurar o melhor quarto "atualmente"disponível nas proximidades de cada evento.

Na nossa versão do *MagicStay* é também possível a pesquisa de eventos consoante a sua categoria.

1.2 Lista de Requisitos

ID	Prioridade	Descrição
R1	Obrigatório	Registo do utilizador
R2	Obrigatório	Remover um utilizador
R3	Obrigatório	Utilizador pode arrendar uma propriedade
R4	Obrigatório	Utilizador pode adicionar uma propriedade
R5	Obrigatório	Utilizador pode remover uma propriedade
R6	Obrigatório	Utilizador pode adicionar um evento
R7	Obrigatório	Utilizador pode remover um evento
R8	Obrigatório	Utilizador pode procurar por uma propriedade por cidade
R9	Obrigatório	Utilizador pode procurar por uma propriedade por evento
R10	Obrigatório	Utilizador pode procurar por eventos por categoria
R11	Obrigatório	Utilizador pode procurar pela melhor propriedade de uma cidade

Tabela 1: Tabela de Requisitos

2 Modelo visual UML

2.1 Modelo Use Case

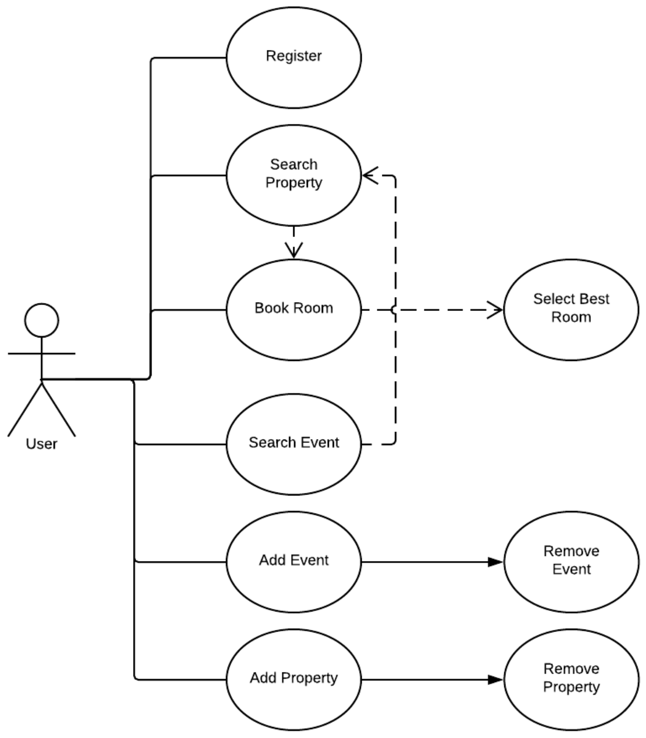


Figura 1: Modelo Use Case

2.2 Diagrama de Classes

Este diagrama de classes inclui todas as classes do projeto com a exceção das classes de teste das respetivas classes de modo a simplificar a visualização do diagrama.

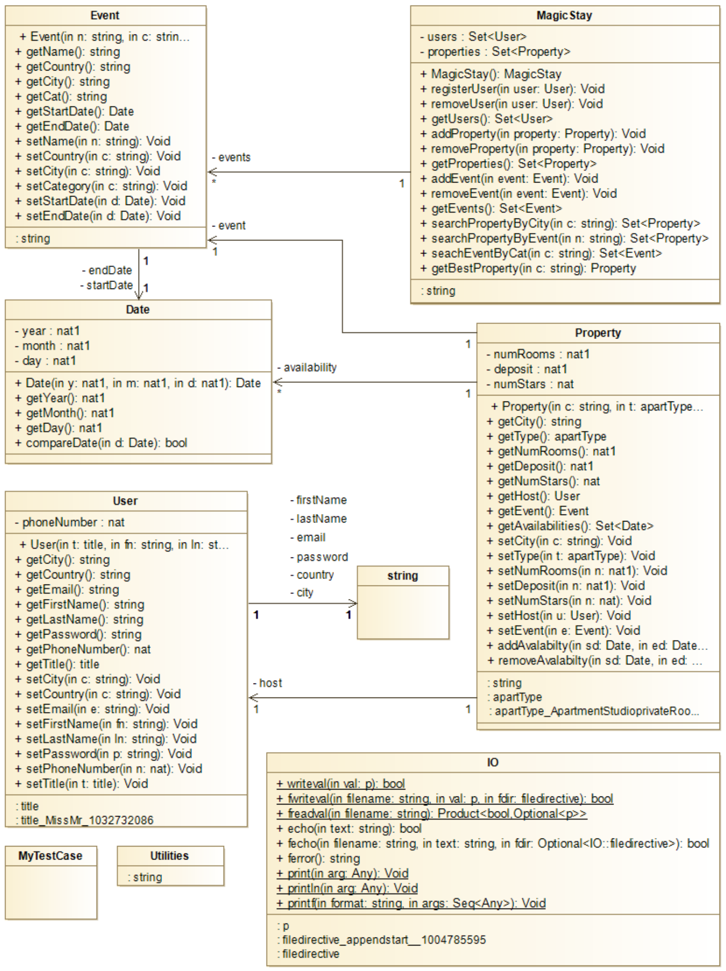

Event
+ Event(in n: string, in c: strin...
+ getName(): string
+ getCountry(): string
+ getCity(): string
+ getCat(): string
+ getStartDate(): Date
+ getEndDate(): Date
+ setName(in n: string): Void
+ setCountry(in c: string): Void
+ setCity(in c: string): Void
+ setCategory(in c: string): Void
+ setStartDate(in d: Date): Void
+ setEndDate(in d: Date): Void
: string
MagicStay
- users : Set<User>
- properties : Set<Property>
+ MagicStay(): MagicStay
+ registerUser(in user: User): Void
+ removeUser(in user: User): Void
+ getUsers(): Set<User>
+ addProperty(in property: Property): Void
+ removeProperty(in property: Property): Void
+ getProperties(): Set<Property>
+ addEvent(in event: Event): Void
+ removeEvent(in event: Event): Void
+ getEvents(): Set<Event>
+ searchPropertyByCity(in c: string): Set<Property>
+ searchPropertyByEvent(in n: string): Set<Property>
+ seachEventByCat(in c: string): Set<Event>
+ getBestProperty(in c: string): Property
: string
- events
*
1
- event
1
- endDate
- startDate
1
1
Date
- year : nat1
- month : nat1
- day : nat1
+ Date(in y: nat1, in m: nat1, in d: nat1): Date
+ getYear(): nat1
+ getMonth(): nat1
+ getDay(): nat1
+ compareDate(in d: Date): bool
- availability
*
1
1
Property
- numRooms : nat1
- deposit : nat1
- numStars : nat
+ Property(in c: string, in t: apartType...
+ getCity(): string
+ getType(): apartType
+ getNumRooms(): nat1
+ getDeposit(): nat1
+ getNumStars(): nat
+ getHost(): User
+ getEvent(): Event
+ getAvailabilities(): Set<Date>
+ setCity(in c: string): Void
+ setType(in t: apartType): Void
+ setNumRooms(in n: nat1): Void
+ setDeposit(in n: nat1): Void
+ setNumStars(in n: nat): Void
+ setHost(in u: User): Void
+ setEvent(in e: Event): Void
+ addAvalabilty(in sd: Date, in ed: Date...
+ removeAvalabilty(in sd: Date, in ed: ...
: string
: apartType
: apartType_ApartmentStudioprivateRoo...
User
- phoneNumber : nat
+ User(in t: title, in fn: string, in ln: st...
+ getCity(): string
+ getCountry(): string
+ getEmail(): string
+ getFirstName(): string
+ getLastName(): string
+ getPassword(): string
+ getPhoneNumber(): nat
+ getTitle(): title
+ setCity(in c: string): Void
+ setCountry(in c: string): Void
+ setEmail(in e: string): Void
+ setFirstName(in fn: string): Void
+ setLastName(in ln: string): Void
+ setPassword(in p: string): Void
+ setPhoneNumber(in n: nat): Void
+ setTitle(in t: title): Void
: title
: title_MissMr_1032732086
- firstName
- lastName
- email
- password
- country
- city
string
1
1
- host
1
1
IO
+ writeval(in val: p): bool
+ fwriteval(in filename: string, in val: p, in fdir: filedirective): bool
+ freadval(in filename: string): Product<bool,Optional<p>>
+ echo(in text: string): bool
+ fecho(in filename: string, in text: string, in fdir: Optional<IO::filedirective>): bool
+ ferror(): string
+ print(in arg: Any): Void
+ println(in arg: Any): Void
+ printf(in format: string, in args: Seq<Any>): Void
: p
: filedirective_appendstart__1004785595
: filedirective
MyTestCase
Utilities
: string

Figura 2: Diagrama de Classes

A tabela seguinte contém a informação sobre as classes apresentadas anteriormente no Diagrama de Classes da Figura 2.

Classe	Descrição
Event	Define um evento e todas as informações associadas
User	Define um utilizador e todas as informações associadas
Propoerty	Define uma estação hoteleira incluindo o número de quartos disponíveis assim como restante informação necessária
Date	Define uma data
MagicStay	Define o conteúdo do serviço MagicStay incluindo utilizadores, propriedades e eventos a decorrer. Inclui também as funções de pesquisa
IO	Biblioteca standard de Overture em Inputs/Outputs
Utilities	Define uma string
MyTestCase	Superclasse que define testes e casos de uso

Tabela 2: Descrição das classes

3 Formal VDM++ Model

3.1 Classe Date

```
class Date

instance variables
 private year : nat1;
 private month : nat1;
 private day : nat1;

operations
 -- Constructor

 public Date : nat1*nat1*nat1 ==> Date
  Date(y,m,d) ==
  (
   year := y;
   month := m;
   day := d;
   return self
  )
  pre y > 0 and m <= 12 and d <= 31;

 -- Returns the year of the date --

 public pure getYear : () ==> nat1
  getYear() == (return year);

 -- Returns the month of the date --

 public pure getMonth : () ==> nat1
  getMonth() == (return month);

 -- Returns the day of the date --

 pure public getDay : () ==> nat1
  getDay() == (return day);

 -- Compares two dates for example date1 and date2
 -- If the date1 is "bigger" than the date 2, returns true otherwise, false

 pure public compareDate : Date ==> bool
  compareDate(d) ==
  (
   return ((d.getYear() * d.getMonth() * d.getDay()) < (year * month * day))
  );

end Date
```

3.2 Classe Event

```
class Event
types
 public string = Utilities`string;

instance variables
 private name : string;
 private country : string;
 private city : string;
 private category : string;
 private startDate : Date;
 private endDate : Date;

operations
 --Constructor

 public Event : string*string*string*string*Date*Date ==> Event
  Event(n,c,ct,cat,sd,ed) ==
  (
   name := n;
   country := c;
   city := ct;
   category := cat;
   startDate := sd;
   endDate := ed;
   return self
  );

 --get event's name

 public getName : () ==> string
  getName() == return name;

 --get event's country where is going to happen

 public getCountry : () ==> string
  getCountry() == return country;

 --get event's country where is going to happen

 public getCity : () ==> string
  getCity() == return city;

 --get event's Category

 public getCat : () ==> string
  getCat() == return category;

 --get event's starting date

 public getStartDate : () ==> Date
  getStartDate() == return startDate;

 --get event's ending date
 public getEndDate : () ==> Date
  getEndDate() == return endDate;

 --set event's name

 public setName : string ==> ()
  setName(n) == (name := n;return);

 --set event's country

 public setCountry : string ==> ()
  setCountry(c) == (country := c;return);

 --set event's country

 public setCity : string ==> ()
  setCity(c) == (city := c;return);

 --set event's category

 public setCategory : string ==> ()
  setCategory(c) == (category := c;return);

 --set event's starting date

 public setStartDate : Date ==> ()
  setStartDate(d) == (startDate := d;return);

 --set event's starting date

 public setEndDate : Date ==> ()
  setEndDate(d) == (endDate := d;return);
end Event
```

3.3 Classe Property

```
class Property
types
 public string = Utilities`string;
 public apartType = <Studio> | <Apartment> | <privateRoom>;

instance variables
 private city : string;
 private type : apartType;
 private numRooms : nat1;
 private deposit : nat1;
 private numStars : nat;
 private host: User;
 private event: Event;
 private availability : set of Date := {};

 inv city <> "";

operations
 --Constructor

 public Property : string*apartType*nat1*nat1*nat*User*Event==> Property
  Property(c,t,nr,d,ns,h,e) ==
  (
   city := c;
   type := t;
   numRooms := nr;
   deposit := d;
   numStars := ns;
   host := h;
   event := e;
   availability := {};
   return self
  );

 -- get property's city

 public getCity : () ==> string
  getCity() == return city;

 -- get property's type

 public getType : () ==> apartType
  getType() == return type;

 -- get property's number of rooms

 public getNumRooms : () ==> nat1
  getNumRooms() == return numRooms;

 -- get property's rent value

 public getDeposit : () ==> nat1
  getDeposit() == return deposit;

 -- get property's number of stars

 public getNumStars : () ==> nat
  getNumStars() == return numStars;

 -- get property's owner

 public getHost : () ==> User
  getHost() == return host;

 -- get property's nearby events

 public getEvent : () ==> Event
  getEvent() == return event;

 -- get property's booking dates

 public getAvailabilities : () ==> set of Date
  getAvailabilities() == return availability;

 -- set property's city

 public setCity : string ==> ()
  setCity(c) == (city :=c; return);

 -- set property's type

 public setType : apartType ==> ()
  setType(t) == (type :=t; return);

 -- set property's number of rooms

 public setNumRooms : nat1 ==> ()
  setNumRooms(n) == (numRooms := n; return);
 -- set property's rent value

 public setDeposit : nat1 ==> ()
  setDeposit(n) == (deposit := n; return);

 -- set property's number of stars

 public setNumStars : nat ==> ()
  setNumStars(n) == (numStars := n; return);

 -- set property's owner

 public setHost : User ==> ()
  setHost(u) == (host := u; return);

 -- set property's event

 public setEvent : Event ==> ()
  setEvent(e) == (event := e; return);

 -- add a period in which property is booked

 public addAvalabilty : Date*Date ==> ()
  addAvalabilty(sd,ed) ==
  (
   availability := availability union {sd,ed}
  );

 -- remove a period in which property is booked

 public removeAvalabilty : Date*Date ==> ()
  removeAvalabilty(sd,ed) ==
  (
   availability := availability \ {sd,ed}
  );

end Property
```

3.4 Classe User

```
class User
types
 public string = Utilities`string;
 public title = <Mr> | <Miss>;
instance variables
 private userTitle: title;
  private firstName: string;
  private lastName: string;
  private email: string;
  private country: string;
  private city: string;
  private phoneNumber : nat;
  private password: string;

  inv firstName <> "";
  inv lastName <> "";
  inv email <> "";
  inv country <> "";
  inv city <> "";
  inv phoneNumber <> 0;
  inv password <> "";
```

```
operations

 -- User Constructor

 public User: title*string*string*string*string*string*nat*string ==> User
   User(t,fn,ln,e,ct,c,ph,pass) ==
   (
   userTitle := t;
    firstName := fn;
    lastName := ln;
    email := e;
    country := ct;
    city := c;
    phoneNumber := ph;
  password := pass;
  return self;
  );

 -- get user title

 public getTitle : () ==> title
  getTitle() == return userTitle;

 -- get user first Name

 public getFirstName : () ==> string
  getFirstName() == return firstName;

 -- get user last Name

 public getLastName : () ==> string
  getLastName() == return lastName;

 -- get user email

 public getEmail : () ==> string
  getEmail() == return  email;

 -- get user's country

 public getCountry : () ==> string
  getCountry() == return country;

 -- get user's city

 public getCity : () ==> string
  getCity() == return city;

 -- get user's phone number

 public getPhoneNumber : () ==> nat
  getPhoneNumber() == return phoneNumber;

 -- get user password

 public getPassword : () ==> string
  getPassword() == return password;

 -- set user title Mr or Miss

 public setTitle : title ==> ()
  setTitle(t) == (userTitle := t; return);

 -- set user first name
 public setFirstName : string ==> ()
  setFirstName(fn) == (firstName := fn; return);

 -- set user last name

 public setLastName : string ==> ()
  setLastName(ln) == (lastName := ln; return);

 -- set user email address

 public setEmail : string ==> ()
  setEmail(e) == (email := e; return);

 -- set user country of origin

 public setCountry : string ==> ()
  setCountry(c) == (country := c; return);

 -- set user city of origin

 public setCity : string ==> ()
  setCity(c) == (city := c; return);

 -- set user phone number

 public setPhoneNumber : nat ==> ()
  setPhoneNumber(n) == (phoneNumber := n; return);

 -- set user password

 public setPassword : string ==> ()
  setPassword(p) == (password := p; return);

end User
```

3.5 Classe MagisStay

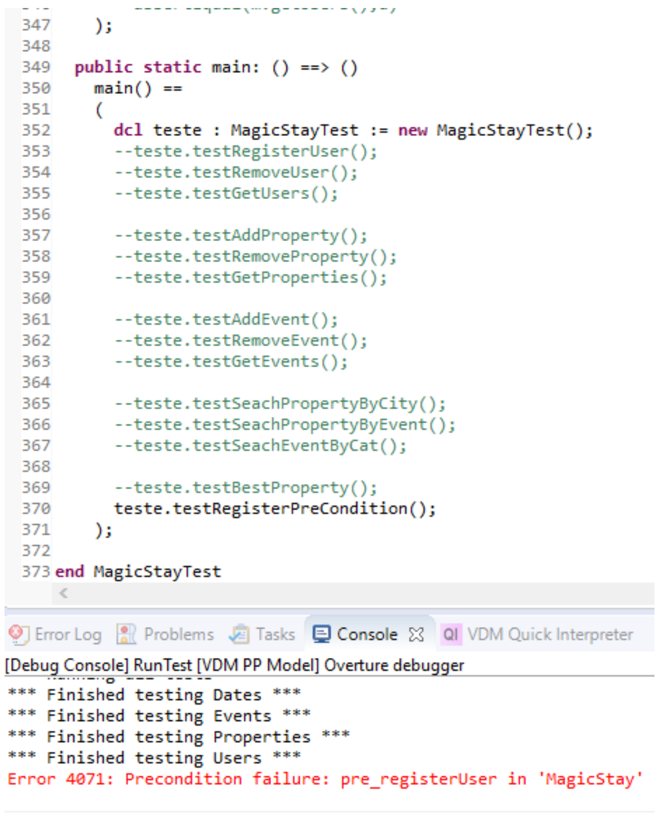
```
class MagicStay
types
 public string = Utilities`string;

instance variables
 private users : set of User := {};
 private properties : set of Property := {};
 private events : set of Event := {};

operations
 -- Contructor

 public MagicStay : () ==> MagicStay
  MagicStay() == return self
  post users = {} and properties = {} and events = {};

 --Register a User

 public registerUser : User ==> ()
  registerUser(user) == users := users union {user}
  pre user not in set users
  post users = users~ union {user};
```

```
--Remove a User

public removeUser : User ==> ()
 removeUser(user) == users := users \ {user}
 pre user in set users
 post users = users~ \ {user};

--get all the users registered in MagicStay

public getUsers :() ==> set of User
 getUsers() == (return users);

--Add a Property

public addProperty : Property ==> ()
 addProperty(property) == properties := properties union {property}
 pre property not in set properties
 post properties = properties~ union {property};

--remove a Property

public removeProperty : Property ==> ()
 removeProperty(property) == properties := properties \ {property}
 pre property in set properties
 post properties = properties~ \ {property};

--get all the properties available in MagicStay

public getProperties :() ==> set of Property
 getProperties() == (return properties);

--Add an Event

public addEvent : Event ==> ()
 addEvent(event) == events := events union {event}
 pre event not in set events
 post events = events~ union {event};

--remove a Event

public removeEvent : Event ==> ()
 removeEvent(event) == events := events \ {event}
 pre event in set events
 post events = events~ \ {event};

--get all the events available in MagicStay

public getEvents :() ==> set of Event
 getEvents() == (return events);


--Search properties by city

public searchPropertyByCity : string ==> set of Property
 searchPropertyByCity(c) ==
 (
  dcl res: set of Property := {};
   for all p in set properties do
    if (p.getCity() = c)
     then (res := {p} union res);
  return res
 )
 pre c <> "";

--Search for properties by event
```

```
 public searchPropertyByEvent : string ==> set of Property
  searchPropertyByEvent(n) ==
  (
   dcl res: set of Property := {};
    for all e in set events do
     if(e.getName() = n)
      then(
         dcl ct: string := e.getCity();
          for all p in set properties do
           if(p.getCity() = ct)
            then (res := {p} union res)
        );
   return res
  )
  pre n <> "";

 --search events by category

 public seachEventByCat : string ==> set of Event
  seachEventByCat(c) ==
  (
   dcl res: set of Event := {};
    for all e in set events do
     if(e.getCat() = c)
      then (res := {e} union res);
   return res
  )
  pre c <> "";

 --get the best property from a certain city

 public getBestProperty : string ==> Property
  getBestProperty(c) ==
  (
   dcl best : [Property] := nil;
   for all p in set properties do
    if (best = nil or p.getNumStars() > best.getNumStars() and p.getCity() = c) then
     best := p;
   return best
  )
  pre card properties > 0
  post RESULT in set properties;

end MagicStay
```

4 Validação do Modelo - Análise de Consistência

Na análise de consistência do programa, ou seja, a verificação das condições *pre*, *post* e *invariant* de cada função são cumpridas, foram feitos testes que vão contra essas mesmas condições. Segue abaixo alguns exemplos do insucesso desses testes.

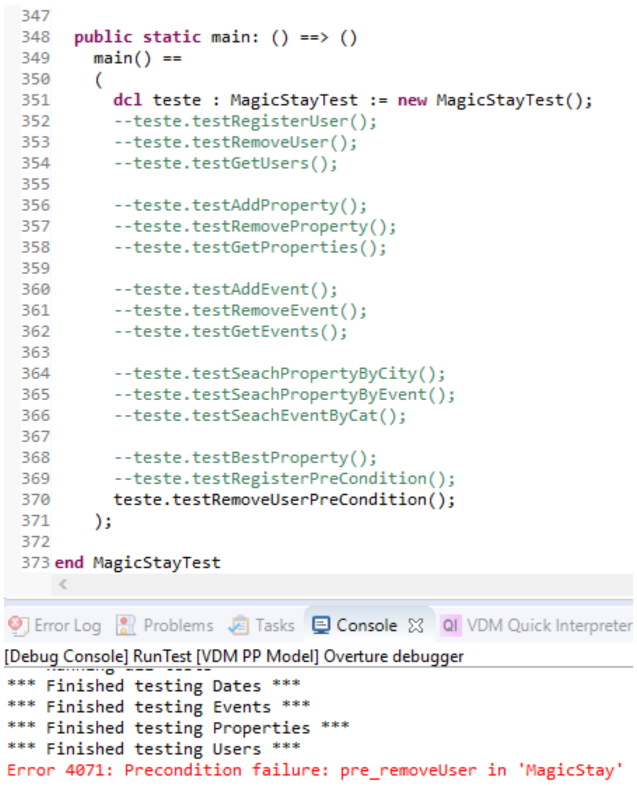


```
    );

  public static main: () ==> ()
    main() ==
    (
      dcl teste : MagicStayTest := new MagicStayTest();
      --teste.testRegisterUser();
      --teste.testRemoveUser();
      --teste.testGetUsers();

      --teste.testAddProperty();
      --teste.testRemoveProperty();
      --teste.testGetProperties();

      --teste.testAddEvent();
      --teste.testRemoveEvent();
      --teste.testGetEvents();

      --teste.testSeachPropertyByCity();
      --teste.testSeachPropertyByEvent();
      --teste.testSeachEventByCat();

      --teste.testBestProperty();
      teste.testRegisterPreCondition();
    );

end MagicStayTest
```

[Debug Console] RunTest [VDM PP Model] Overture debugger

```
*** Finished testing Dates ***
*** Finished testing Events ***
*** Finished testing Properties ***
*** Finished testing Users ***
Error 4071: Precondition failure: pre_registerUser in 'MagicStay'
```

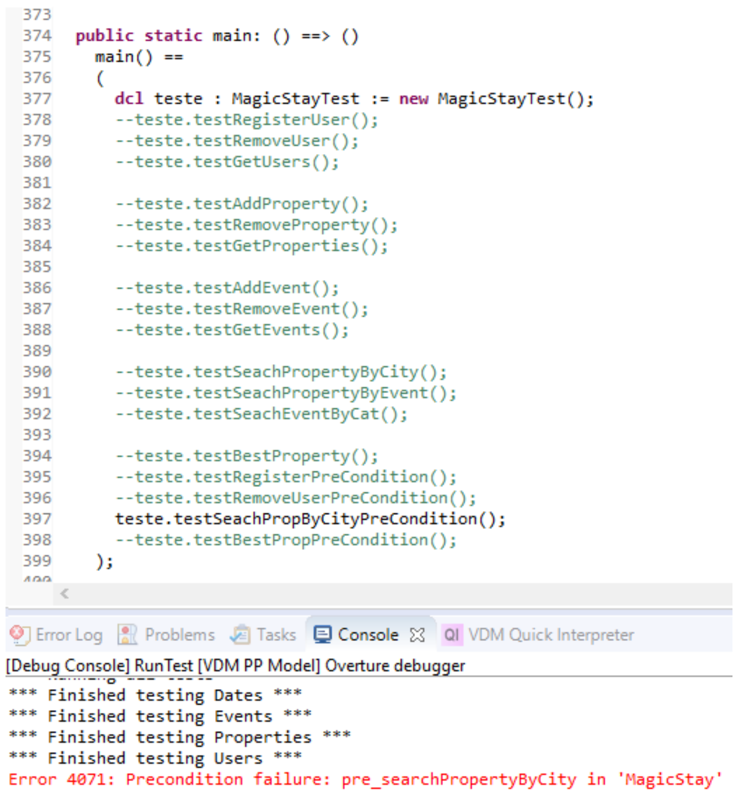
Figura 3: Tentativa de registar utilizador repetido



```
public static main: () ==> ()
  main() ==
  (
    dcl teste : MagicStayTest := new MagicStayTest();
    --teste.testRegisterUser();
    --teste.testRemoveUser();
    --teste.testGetUsers();

    --teste.testAddProperty();
    --teste.testRemoveProperty();
    --teste.testGetProperties();

    --teste.testAddEvent();
    --teste.testRemoveEvent();
    --teste.testGetEvents();

    --teste.testSeachPropertyByCity();
    --teste.testSeachPropertyByEvent();
    --teste.testSeachEventByCat();

    --teste.testBestProperty();
    --teste.testRegisterPreCondition();
    teste.testRemoveUserPreCondition();
  );

end MagicStayTest
```

```
[Debug Console] RunTest [VDM PP Model] Overture debugger
*** Finished testing Dates ***
*** Finished testing Events ***
*** Finished testing Properties ***
*** Finished testing Users ***
Error 4071: Precondition failure: pre_removeUser in 'MagicStay'
```

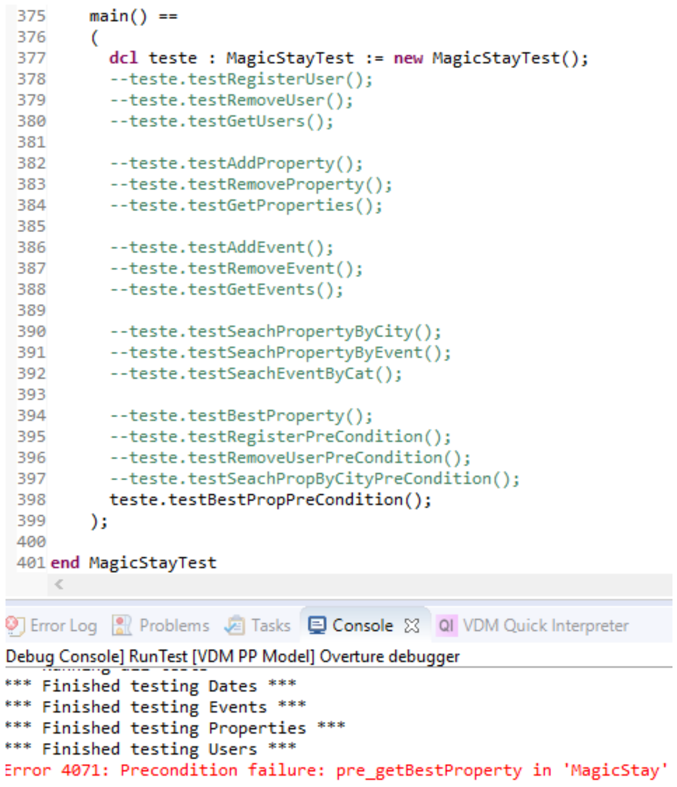
Figura 4: Tentativa de remover utilizador inexistente



```
public static main: () ==> ()
  main() ==
  (
    dcl teste : MagicStayTest := new MagicStayTest();
    --teste.testRegisterUser();
    --teste.testRemoveUser();
    --teste.testGetUsers();

    --teste.testAddProperty();
    --teste.testRemoveProperty();
    --teste.testGetProperties();

    --teste.testAddEvent();
    --teste.testRemoveEvent();
    --teste.testGetEvents();

    --teste.testSeachPropertyByCity();
    --teste.testSeachPropertyByEvent();
    --teste.testSeachEventByCat();

    --teste.testBestProperty();
    --teste.testRegisterPreCondition();
    --teste.testRemoveUserPreCondition();
    teste.testSeachPropByCityPreCondition();
    --teste.testBestPropPreCondition();
  );
```

```
[Debug Console] RunTest [VDM PP Model] Overture debugger
*** Finished testing Dates ***
*** Finished testing Events ***
*** Finished testing Properties ***
*** Finished testing Users ***
Error 4071: Precondition failure: pre_searchPropertyByCity in 'MagicStay'
```

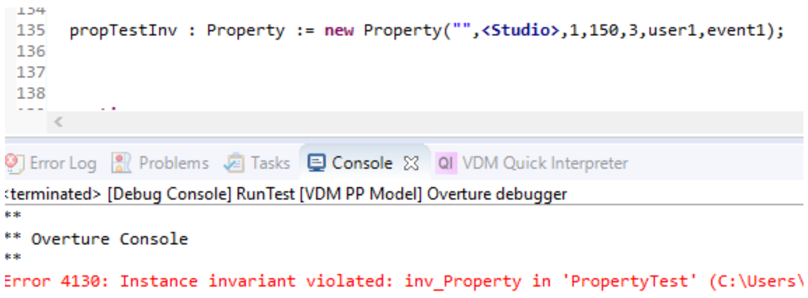
Figura 5: Tentativa de procura de uma propriedade sem introduzir cidade



```
    main() ==
    (
      dcl teste : MagicStayTest := new MagicStayTest();
      --teste.testRegisterUser();
      --teste.testRemoveUser();
      --teste.testGetUsers();

      --teste.testAddProperty();
      --teste.testRemoveProperty();
      --teste.testGetProperties();

      --teste.testAddEvent();
      --teste.testRemoveEvent();
      --teste.testGetEvents();

      --teste.testSeachPropertyByCity();
      --teste.testSeachPropertyByEvent();
      --teste.testSeachEventByCat();

      --teste.testBestProperty();
      --teste.testRegisterPreCondition();
      --teste.testRemoveUserPreCondition();
      --teste.testSeachPropByCityPreCondition();
      teste.testBestPropPreCondition();
    );

end MagicStayTest
```

```
Debug Console] RunTest [VDM PP Model] Overture debugger
*** Finished testing Dates ***
*** Finished testing Events ***
*** Finished testing Properties ***
*** Finished testing Users ***
Error 4071: Precondition failure: pre_getBestProperty in 'MagicStay'
```

Figura 6: Tentativa de procura de melhor propriedade sem introduzir cidade



```
propTestInv : Property := new Property("",<Studio>,1,150,3,user1,event1);
```

```
<terminated> [Debug Console] RunTest [VDM PP Model] Overture debugger
**
** Overture Console
**
Error 4130: Instance invariant violated: inv_Property in 'PropertyTest' (C:\Users\
```

Figura 7: Tentativa de introduzir propriedade sem cidade designada

5 Verificação do Modelo - Testes

Os testes apresentados a seguir, foram todos passados com sucesso. Apresentam-se também nesta secção as tabelas de cobertura dos testes, é de notar no entanto que as tabelas de cobertura não consideram os testes de análise de consistência de trabalho visto que estes testes tem como objetivo falhar.

5.1 Classe DateTest

```
class DateTest is subclass of MyTestCase
```

```
instance variables
 sdate : Date := new Date(2017, 3, 22);
 edate : Date := new Date(2017, 3, 24);

operations


 private testCompareDate: () ==> ()
  testCompareDate() ==
  (
   assertEqual(edate.compareDate(sdate), true);
     assertEqual(sdate.compareDate(edate), false)
  );


 private testGetYear: () ==> ()
  testGetYear() ==
  (
   assertEqual(sdate.getYear(), 2017);
     assertEqual(edate.getYear(), 2017)
  );


 private testGetMonth: () ==> ()
  testGetMonth() ==
  (
   assertEqual(sdate.getMonth(), 3);
     assertEqual(edate.getMonth(), 3)
  );


 private testGetDay: () ==> ()
  testGetDay() ==
  (
   assertEqual(sdate.getDay(), 22);
     assertEqual(edate.getDay(), 24)
  );


 public static main: () ==> ()
     main() ==
     (
      dcl teste : DateTest := new DateTest();
      teste.testCompareDate();
      teste.testGetYear();
      teste.testGetMonth();
      teste.testGetDay();
     );

end DateTest
```

Function or operation	Line	Coverage	Calls
Date	10	100.0%	44
compareDate	34	100.0%	2
getDay	29	100.0%	4
getMonth	25	100.0%	4
getYear	21	100.0%	4
main	79	100.0%	1

testCompareDate	51	100.0%	1
testGetDay	72	100.0%	1
testGetMonth	65	100.0%	1
testGetYear	58	100.0%	1
Date.vdmpp		100.0%	63

Tabela 3: Cobertura da classe Date

5.2 Classe EventTest

```
class EventTest is subclass of MyTestCase

instance variables
 user1 : User := new User(<Mr>,"luis","cruz","luis.aa@gmail.com","portugal","porto",912829288,
"feup");
 user2 : User := new User(<Miss>,"maria","alegre","m.a@hormail.com","portugal","porto",
922832238,"alegre");
 user3 : User := new User(<Mr>,"vito","corleone","v.godphaderPM@gmail.com","italia","rome",
987654321,"admin");
 user4 : User := new User(<Mr>,"Tim","Berners-Lee","timWWW@gmail.com","eua","cambridge",
123456789,"1234");

 date1 : Date := new Date(2017,2,2);
 date2 : Date := new Date(2017,2,3);
 date3 : Date := new Date(2017,3,22);
 date4 : Date := new Date(2017,3,24);
 date5 : Date := new Date(2017,12,30);

 event1 : Event := new Event("AI in 2020","portugal","porto","IT",date1,date2);
 event2 : Event := new Event("IT and security","italy","rome","IT",date3,date4);
 event3 : Event := new Event("W3C consvention","eua","cambridge","IT",date1,date2);
 event4 : Event := new Event("Talk a Bit","portugal","porto","IT",date1,date2);

 prop1 : Property := new Property("porto",<Studio>,1,150,3,user1,event1);
 prop2 : Property := new Property("porto",<Apartment>,2,300,4,user2,event1);
 prop3 : Property := new Property("rome",<Studio>,1,450,4,user3,event2);
 prop5 : Property := new Property("rome",<privateRoom>,1,100,2,user3,event2);
 prop4 : Property := new Property("cambeidge",<Apartment>,3,730,5,user4,event3);


operations
-- TEST GETS --

 private testGetName: () ==> ()
  testGetName() ==
  (
   assertEqual(event1.getName(), "AI in 2020");
   assertEqual(event2.getName(), "IT and security")
  );


 private testGetCountry : () ==> ()
  testGetCountry() ==
  (
   assertEqual(event1.getCountry(), "portugal");
   assertEqual(event2.getCountry(), "italy")
  );
```

```
private testGetCity : () ==> ()
 testGetCity() ==
 (
  assertEqual(event1.getCity(), "porto");
  assertEqual(event2.getCity(), "rome")
 );


private testGetCategory : () ==> ()
 testGetCategory() ==
 (
  assertEqual(event1.getCat(), "IT");
  assertEqual(event2.getCat(), "IT")
 );


private testGetStartDate : () ==> ()
 testGetStartDate() ==
 (
  assertEqual(event1.getStartDate(), date1);
  assertEqual(event2.getStartDate(), date3)
 );


private testGetEndDate : () ==> ()
 testGetEndDate() ==
 (
  assertEqual(event1.getEndDate(), date2);
  assertEqual(event2.getEndDate(), date4)
 );

-- TEST SETS --

private testSetName: () ==> ()
 testSetName() ==
 (
  event1.setName("Test Event name");
  event2.setName("Test2 Event name")
 );


private testSetCountry: () ==> ()
 testSetCountry() ==
 (
  event1.setCountry("france");
  event2.setCountry("eua")
 );


private testSetCity: () ==> ()
 testSetCity() ==
 (
  event1.setCity("paris");
  event2.setCity("lisbon")
 );


private testSetCategory: () ==> ()
 testSetCategory() ==
 (
  event1.setCategory("transport");
  event2.setCategory("media")
 );
 private testSetStartDate: () ==> ()
  testSetStartDate() ==
  (
   event1.setStartDate(date3);
   event2.setStartDate(date1)
  );


 private testSetEndDate: () ==> ()
  testSetEndDate() ==
  (
   event1.setEndDate(date4);
   event2.setEndDate(date2)
  );


 public static main: () ==> ()
  main() ==
  (
   dcl teste : EventTest := new EventTest();
   teste.testGetName();
   teste.testGetCountry();
   teste.testGetCity();
   teste.testGetCategory();
   teste.testGetStartDate();
   teste.testGetEndDate();
   teste.testGetStartDate();

   teste.testSetName();
   teste.testSetCountry();
   teste.testSetCity();
   teste.testSetCategory();
   teste.testSetStartDate();
   teste.testSetEndDate();
   teste.testSetStartDate()
  );
end EventTest
```

Function or operation	Line	Coverage	Calls
Event	15	100.0%	32
getCat	40	100.0%	6
getCity	36	100.0%	4
getCountry	32	100.0%	2
getEndDate	48	100.0%	2
getName	28	100.0%	10
getStartDate	44	100.0%	4
main	192	100.0%	1
setCategory	64	100.0%	2
setCity	60	100.0%	2
setCountry	56	100.0%	2
setEndDate	72	100.0%	2
setName	52	100.0%	2
setStartDate	68	100.0%	4
testGetCategory	128	100.0%	1
testGetCity	121	100.0%	1
testGetCountry	114	100.0%	3

testGetEndDate	142	100.0%	1
testGetName	107	100.0%	1
testGetStartDate	135	100.0%	2
testSetCategory	171	100.0%	1
testSetCity	164	100.0%	1
testSetCountry	157	100.0%	1
testSetEndDate	185	100.0%	1
testSetName	150	100.0%	1
testSetStartDate	178	100.0%	2
Event.vdmpp		100.0%	91

Tabela 4: Cobertura da classe Event

5.3 Classe PropertyTest

```
class PropertyTest is subclass of MyTestCase

instance variables
 user1 : User := new User(<Mr>,"luis","cruz","luis.aa@gmail.com","portugal","porto",912829288,
"feup");
 user2 : User := new User(<Miss>,"maria","alegre","m.a@hormail.com","portugal","porto",
922832238,"alegre");
 user3 : User := new User(<Mr>,"vito","corleone","v.godphaderPM@gmail.com","italia","rome",
987654321,"admin");
 user4 : User := new User(<Mr>,"Tim","Berners-Lee","timWWW@gmail.com","eua","cambridge",
123456789,"1234");

 date1 : Date := new Date(2017,2,2);
 date2 : Date := new Date(2017,2,3);
 date3 : Date := new Date(2017,3,22);
 date4 : Date := new Date(2017,3,24);
 date5 : Date := new Date(2017,12,30);

 event1 : Event := new Event("AI in 2020","portugal","porto","IT",date1,date2);
 event2 : Event := new Event("IT and security","italy","rome","IT",date3,date4);
 event3 : Event := new Event("W3C consvention","eua","cambridge","IT",date1,date2);
 event4 : Event := new Event("Talk a Bit","portugal","porto","IT",date1,date2);

 prop1 : Property := new Property("porto",<Studio>,1,150,3,user1,event1);
 prop2 : Property := new Property("porto",<Apartment>,2,300,4,user2,event1);
 prop3 : Property := new Property("rome",<Studio>,1,450,4,user3,event2);
 prop5 : Property := new Property("rome",<privateRoom>,1,100,2,user3,event2);
 prop4 : Property := new Property("cambeidge",<Apartment>,3,730,5,user4,event3);



operations
 -- TEST GETS --

 private testGetCity : () ==> ()
  testGetCity() ==
  (
   assertEqual(prop1.getCity(), "porto");
   assertEqual(prop3.getCity(), "rome")
  );


 private testGetType : () ==> ()
  testGetType() ==
  (
   assertEqual(prop1.getType(), <Studio>);
   assertEqual(prop2.getType(), <Apartment>)
  );


 private testGetNumRooms : () ==> ()
  testGetNumRooms() ==
  (
   assertEqual(prop1.getNumRooms(), 1);
   assertEqual(prop2.getNumRooms(), 2)
  );


 private testGetDeposit : () ==> ()
  testGetDeposit() ==
  (
   assertEqual(prop1.getDeposit(), 150);
   assertEqual(prop2.getDeposit(), 300)
  );


 private testGetNumStars : () ==> ()
  testGetNumStars() ==
  (
   assertEqual(prop1.getNumStars(), 3);
   assertEqual(prop2.getNumStars(), 4)
  );


 private testGetHost : () ==> ()
  testGetHost() ==
  (
   assertEqual(prop1.getHost(), user1);
   assertEqual(prop2.getHost(), user2)
  );


 private testGetEvent : () ==> ()
  testGetEvent() ==
  (
   assertEqual(prop1.getEvent(), event1);
   assertEqual(prop2.getEvent(), event1)
  );

 -- TEST SETS --


 private testSetCity: () ==> ()
  testSetCity() ==
  (
   prop1.setCity("paris");
   prop2.setCity("lisbon")
  );


 private testSetType: () ==> ()
  testSetType() ==
  (
   prop1.setType(<privateRoom>);
   prop2.setType(<privateRoom>)
  );


 private testSetNumRooms : () ==> ()
```

```
 testSetNumRooms() ==
 (
  prop1.setNumRooms(2);
  prop2.setNumRooms(3)
 );


private testSetDeposit : () ==> ()
 testSetDeposit() ==
 (
  prop1.setDeposit(400);
  prop2.setDeposit(500)
 );


private testSetNumStars : () ==> ()
 testSetNumStars() ==
 (
  prop1.setNumStars(1);
  prop2.setNumStars(1)
 );


private testSetHost : () ==> ()
 testSetHost() ==
 (
  prop1.setHost(user2);
  prop2.setHost(user1)
 );


private testSetEvent : () ==> ()
 testSetEvent() ==
 (
  prop1.setEvent(event4);
  prop2.setEvent(event4)
 );


-- TEST ADDS --


private testAddAvalilability : () ==> ()
 testAddAvalilability() ==
 (
  prop1.addAvalabilty(date1,date2);
  prop2.addAvalabilty(date1,date2);
  assertEqual(prop1.getAvailabilities(), {date1,date2});
  assertEqual(prop2.getAvailabilities(), {date1,date2})
 );


-- TEST REMOVES --

private testRemoveAvalilability : () ==> ()
 testRemoveAvalilability() ==
 (
  prop1.removeAvalabilty(date1,date2);
  prop2.removeAvalabilty(date1,date2);

  assertEqual(prop1.getAvailabilities(), {});
  assertEqual(prop2.getAvailabilities(), {})

 );
 public static main: () ==> ()
  main() ==
  (
   dcl teste : PropertyTest := new PropertyTest();
   teste.testGetCity();
   teste.testGetType();
   teste.testGetNumRooms();
   teste.testGetDeposit();
   teste.testGetNumStars();
   teste.testGetHost();
   teste.testGetEvent();

   teste.testSetCity();
   teste.testSetType();
   teste.testSetNumRooms();
   teste.testSetDeposit();
   teste.testSetNumStars();
   teste.testSetHost();
   teste.testSetEvent();

   teste.testAddAvalilability();
   teste.testRemoveAvalilability();

  );
end PropertyTest
```

Function or operation	Line	Coverage	Calls
Property	20	100.0%	40
addAvalabilty	95	100.0%	2
getAvailabilities	63	100.0%	4
getCity	35	100.0%	32
getDeposit	47	100.0%	2
getEvent	59	100.0%	2
getHost	55	100.0%	2
getNumRooms	43	100.0%	2
getNumStars	51	100.0%	18
getType	39	100.0%	2
main	261	100.0%	1
removeAvalabilty	102	100.0%	2
setCity	67	100.0%	2
setDeposit	79	100.0%	2
setEvent	91	100.0%	2
setHost	87	100.0%	2
setNumRooms	75	100.0%	2
setNumStars	83	100.0%	2
setType	71	100.0%	2
testAddAvalilability	242	100.0%	1
testGetCity	139	100.0%	3
testGetDeposit	160	100.0%	1
testGetEvent	181	100.0%	1
testGetHost	174	100.0%	3
testGetNumRooms	153	100.0%	3
testGetNumStars	167	100.0%	1

testGetType	146	100.0%	1
testRemoveAvalilability	251	100.0%	1
testSetCity	190	100.0%	1
testSetDeposit	211	100.0%	1
testSetEvent	232	100.0%	1
testSetHost	225	100.0%	1
testSetNumRooms	204	100.0%	1
testSetNumStars	218	100.0%	1
testSetType	197	100.0%	1
Property.vdmpp		100.0%	145

Tabela 5: Cobertura da classe Property

5.4 Classe UserTest

```
class UserTest is subclass of MyTestCase

instance variables
 user1 : User := new User(<Mr>,"luis","cruz","luis.aa@gmail.com","portugal","porto",912829288,
"feup");
 user2 : User := new User(<Miss>,"maria","alegre","m.a@hormail.com","portugal","porto",
922832238,"alegre");
 user3 : User := new User(<Mr>,"vito","corleone","v.godphaderPM@gmail.com","italia","rome",
987654321,"admin");
 user4 : User := new User(<Mr>,"Tim","Berners-Lee","timWWW@gmail.com","eua","cambridge",
123456789,"1234");

 date1 : Date := new Date(2017,2,2);
 date2 : Date := new Date(2017,2,3);
 date3 : Date := new Date(2017,3,22);
 date4 : Date := new Date(2017,3,24);
 date5 : Date := new Date(2017,12,30);

 event1 : Event := new Event("AI in 2020","portugal","porto","IT",date1,date2);
 event2 : Event := new Event("IT and security","italy","rome","IT",date3,date4);
 event3 : Event := new Event("W3C consvention","eua","cambridge","IT",date1,date2);
 event4 : Event := new Event("Talk a Bit","portugal","porto","IT",date1,date2);

 prop1 : Property := new Property("porto",<Studio>,1,150,3,user1,event1);
 prop2 : Property := new Property("porto",<Apartment>,2,300,4,user2,event1);
 prop3 : Property := new Property("rome",<Studio>,1,450,4,user3,event2);
 prop5 : Property := new Property("rome",<privateRoom>,1,100,2,user3,event2);
 prop4 : Property := new Property("cambeidge",<Apartment>,3,730,5,user4,event3);

operations


 private testGetTitle : () ==> ()
  testGetTitle() ==
  (
   assertEqual(user1.getTitle(), <Mr>);
   assertEqual(user2.getTitle(), <Miss>)
  );


 private testGetFirstName : () ==> ()
  testGetFirstName() ==
  (
   assertEqual(user1.getFirstName(), "luis");
   assertEqual(user2.getFirstName(), "maria")
  );


 private testGetLastName : () ==> ()
  testGetLastName() ==
  (
   assertEqual(user1.getLastName(), "cruz");
   assertEqual(user2.getLastName(), "alegre")
  );


 private testGetEmail : () ==> ()
  testGetEmail() ==
  (
   assertEqual(user1.getEmail(), "luis.aa@gmail.com");
   assertEqual(user2.getEmail(), "m.a@hormail.com")
  );


 private testGetCountry : () ==> ()
  testGetCountry() ==
  (
   assertEqual(user1.getCountry(), "portugal");
   assertEqual(user2.getCountry(), "portugal")
  );


 private testGetCity : () ==> ()
  testGetCity() ==
  (
   assertEqual(user1.getCity(), "porto");
   assertEqual(user3.getCity(), "rome")
  );


 private testGetPhoneNum : () ==> ()
  testGetPhoneNum() ==
  (
   assertEqual(user1.getPhoneNumber(), 912829288);
   assertEqual(user2.getPhoneNumber(), 922832238)
  );


 private testGetPassword : () ==> ()
  testGetPassword() ==
  (
   assertEqual(user1.getPassword(), "feup");
   assertEqual(user2.getPassword(), "alegre")
  );

 -- TEST SETS --


 private testSetTitle: () ==> ()
  testSetTitle() ==
  (
   user1.setTitle(<Mr>);
   user2.setTitle(<Miss>)
  );


 private testSetFirstName: () ==> ()
  testSetFirstName() ==
  (
   user1.setFirstName("joao");
   user2.setFirstName("joana")
  );


 private testSetLastName: () ==> ()
  testSetLastName() ==
  (
   user1.setLastName("batista");
   user2.setLastName("correia")
  );


 private testSetEmail: () ==> ()
  testSetEmail() ==
  (
   user1.setEmail("teste123@gmail.com");
   user2.setEmail("teste@gmail.com")
  );


 private testSetCountry: () ==> ()
  testSetCountry() ==
  (
   user1.setCountry("france");
   user2.setCountry("france")
  );


 private testSetCity: () ==> ()
  testSetCity() ==
  (
   user1.setCity("paris");
   user2.setCity("cannes")
  );


 private testSetPhoneNum: () ==> ()
  testSetPhoneNum() ==
  (
   user1.setPhoneNumber(912321456);
   user2.setPhoneNumber(933890765)
  );


 private testSetPassword: () ==> ()
  testSetPassword() ==
  (
   user1.setPassword("testPass");
   user2.setPassword("testPass")
  );


 public static main: () ==> ()
  main() ==
  (
   dcl teste : UserTest := new UserTest();
   teste.testGetTitle();
   teste.testGetFirstName();
   teste.testGetLastName();
   teste.testGetEmail();
   teste.testGetCountry();
   teste.testGetCity();
   teste.testGetPhoneNum();
```

```
    teste.testGetPassword();
    teste.testSetTitle();
    teste.testSetFirstName();
    teste.testSetLastName();
    teste.testSetEmail();
    teste.testSetCountry();
    teste.testSetCity();
    teste.testSetPhoneNum();
    teste.testSetPassword()
  );
end UserTest
```

Function or operation	Line	Coverage	Calls
User	26	100.0%	32
getCity	61	100.0%	2
getCountry	57	100.0%	2
getEmail	53	100.0%	2
getFirstName	45	100.0%	2
getLastName	49	100.0%	2
getPassword	69	100.0%	2
getPhoneNumber	65	100.0%	2
getTitle	41	100.0%	2
main	247	100.0%	1
setCity	93	100.0%	2
setCountry	89	100.0%	2
setEmail	85	100.0%	2
setFirstName	77	100.0%	2
setLastName	81	100.0%	2
setPassword	101	100.0%	2
setPhoneNumber	97	100.0%	2
setTitle	73	100.0%	2
testGetCity	168	100.0%	1
testGetCountry	161	100.0%	3
testGetEmail	154	100.0%	3
testGetFirstName	140	100.0%	1
testGetLastName	147	100.0%	1
testGetPassword	182	100.0%	1
testGetPhoneNum	175	100.0%	1
testGetTitle	133	100.0%	1
testSetCity	226	100.0%	1
testSetCountry	219	100.0%	1
testSetEmail	212	100.0%	1
testSetFirstName	198	100.0%	1
testSetLastName	205	100.0%	1
testSetPassword	240	100.0%	1
testSetPhoneNum	233	100.0%	1
testSetTitle	191	100.0%	1
User.vdmpp		100.0%	85

Tabela 6: Cobertura da classe User

5.5 Classe MagicStayTest

```
class MagicStayTest is subclass of MyTestCase

instance variables

 m : MagicStay := new MagicStay();
 user1 : User := new User(<Mr>,"luis","cruz","luis.aa@gmail.com","portugal","porto",912829288,
"feup");
 user2 : User := new User(<Miss>,"maria","alegre","m.a@hormail.com","portugal","porto",
922832238,"alegre");
 user3 : User := new User(<Mr>,"vito","corleone","v.godphaderPM@gmail.com","italia","rome",
987654321,"admin");
 user4 : User := new User(<Mr>,"Tim","Berners-Lee","timWWW@gmail.com","eua","cambridge",
123456789,"1234");

 date1 : Date := new Date(2017,2,2);
 date2 : Date := new Date(2017,2,3);
 date3 : Date := new Date(2017,3,22);
 date4 : Date := new Date(2017,3,24);
 date5 : Date := new Date(2017,12,30);

 event1 : Event := new Event("AI in 2020","portugal","porto","IT",date1,date2);
 event2 : Event := new Event("IT and security","italy","rome","IT",date3,date4);
 event3 : Event := new Event("W3C consvention","eua","cambridge","IT",date1,date2);
 event4 : Event := new Event("Talk a Bit","portugal","porto","IT",date1,date2);

 prop1 : Property := new Property("porto",<Studio>,1,150,3,user1,event1);
 prop2 : Property := new Property("porto",<Apartment>,2,300,4,user2,event1);
 prop3 : Property := new Property("rome",<Studio>,1,450,4,user3,event2);
 prop5 : Property := new Property("rome",<privateRoom>,1,100,2,user3,event2);
 prop4 : Property := new Property("cambridge",<Apartment>,3,730,5,user4,event3);

operations


 private testRegisterUser: () ==> ()
  testRegisterUser() ==
  (
   dcl u : set of User := {};
   u := u union {user1};
   u := u union {user2};
   u := u union {user3};
   u := u union {user4};

   m.registerUser(user1);
   m.registerUser(user2);
   m.registerUser(user3);
   m.registerUser(user4);

   assertEqual(m.getUsers(),u);

   m.removeUser(user1);
   m.removeUser(user2);
   m.removeUser(user3);
   m.removeUser(user4)

  );


 private testRemoveUser : () ==> ()
  testRemoveUser() ==
  (
   dcl ur : set of User := {user3,user4};
   m.registerUser(user1);
   m.registerUser(user2);
   m.registerUser(user3);
   m.registerUser(user4);

   m.removeUser(user1);
   m.removeUser(user2);


   assertEqual(m.getUsers(),ur);

   m.removeUser(user3);
   m.removeUser(user4);
  );


 private testGetUsers : () ==> ()
  testGetUsers() ==
  (
   dcl u : set of User := {user1,user2,user3,user4};
     m.registerUser(user1);
   m.registerUser(user2);
   m.registerUser(user3);
   m.registerUser(user4);

     assertEqual(m.getUsers(), u)
  );


 private testAddProperty : () ==> ()
  testAddProperty() ==
  (
   dcl p : set of Property := {prop1,prop2,prop3,prop4,prop5};

   m.addProperty(prop1);
   m.addProperty(prop2);
   m.addProperty(prop3);
   m.addProperty(prop4);
   m.addProperty(prop5);

   assertEqual(m.getProperties(),p)


  );


 private testRemoveProperty : () ==> ()
  testRemoveProperty() ==
  (
   dcl p : set of Property := {};
   p := p union {prop5};

   m.removeProperty(prop1);
   m.removeProperty(prop2);
   m.removeProperty(prop3);
   m.removeProperty(prop4);

   assertEqual(m.getProperties(),p);

   m.addProperty(prop1);
   m.addProperty(prop2);
   m.addProperty(prop3);
   m.addProperty(prop4)
  );
```

```
private testGetProperties : () ==> ()
 testGetProperties() ==
 (
  dcl p : set of Property := {};
    p := p union {prop1};
    p := p union {prop2};
    p := p union {prop3};
    p := p union {prop4};
    p := p union {prop5};

    assertEqual(m.getProperties(), p)
 );


private testAddEvent : () ==> ()
 testAddEvent() ==
 (
  dcl e : set of Event := {};
  e := e union {event1};
  e := e union {event2};
  e := e union {event3};
  e := e union {event4};

  m.addEvent(event1);
  m.addEvent(event2);
  m.addEvent(event3);
  m.addEvent(event4);

  assertEqual(m.getEvents(),e)
 );



private testRemoveEvent : () ==> ()
 testRemoveEvent() ==
 (
  dcl e : set of Event := {};
  e := e union {event2};
  e := e union {event3};


  m.removeEvent(event1);
  m.removeEvent(event4);

  assertEqual(m.getEvents(),e);



 );

private testGetEvents : () ==> ()
 testGetEvents() ==
 (
  dcl e : set of Event := {};
    e := e union {event2};
    e := e union {event3};

    assertEqual(m.getEvents(), e);


    m.addEvent(event1);
  m.addEvent(event4)
 );
```

```
private testSeachPropertyByCity: () ==> ()
 testSeachPropertyByCity() ==
 (

  assertEqual(m.searchPropertyByCity("porto"),{prop1,prop2});
  assertEqual(m.searchPropertyByCity("rome"),{prop3,prop5});
  assertEqual(m.searchPropertyByCity("cambridge"),{prop4})
 );

private testSeachPropertyByEvent: () ==> ()
 testSeachPropertyByEvent() ==

 (
  assertEqual(m.searchPropertyByEvent("IT and security"),{prop3,prop5});
  assertEqual(m.searchPropertyByEvent("Talk a Bit"),{prop1,prop2})
 );

private testSeachEventByCat: () ==> ()
 testSeachEventByCat() ==
 (
  dcl e : set of Event := {};
  e := e union {event1};
  e := e union {event2};
  e := e union {event3};

  e := e union {event4};

  assertEqual(m.seachEventByCat("IT"),e)
 );

private testBestProperty : () ==> ()
 testBestProperty() ==

 (
  assertEqual(m.getBestProperty("porto"),prop2);
  assertEqual(m.getBestProperty("cambridge"),prop4)
 );



private testRegisterPreCondition : () ==> ()
 testRegisterPreCondition() ==
 (
  m.registerUser(user1);
  m.registerUser(user1)
 );


private testRemoveUserPreCondition : () ==> ()
 testRemoveUserPreCondition() ==
 (
  m.removeUser(user1);
  m.removeUser(user1);
 );


private testSeachPropByCityPreCondition : () ==> ()
 testSeachPropByCityPreCondition() ==
 (
  assertEqual(m.searchPropertyByCity(""),{});
 );

private testBestPropPreCondition : () ==> ()
 testBestPropPreCondition() ==
  (

   m.addProperty(prop1);
   m.addProperty(prop2);
   m.addProperty(prop3);
   m.addProperty(prop4);
   m.addProperty(prop5);

   m.removeProperty(prop1);
   m.removeProperty(prop2);
   m.removeProperty(prop3);
   m.removeProperty(prop4);
   m.removeProperty(prop5);

   assertEqual(m.getBestProperty("porto"),prop2);
   assertEqual(m.getBestProperty("cambridge"),prop4)
  );

 public static main: () ==> ()
 main() ==
  (
   dcl teste : MagicStayTest := new MagicStayTest();
   teste.testRegisterUser();
   teste.testRemoveUser();
   teste.testGetUsers();

   teste.testAddProperty();
   teste.testRemoveProperty();
   teste.testGetProperties();

   teste.testAddEvent();
   teste.testRemoveEvent();
   teste.testGetEvents();

   teste.testSeachPropertyByCity();
   teste.testSeachPropertyByEvent();
   teste.testSeachEventByCat();

   teste.testBestProperty();
   --teste.testRegisterPreCondition();
   --teste.testRemoveUserPreCondition();
   --teste.testSeachPropByCityPreCondition();
   --teste.testBestPropPreCondition();
   );

end MagicStayTest
```

Function or operation	Line	Coverage	Calls
MagicStay	12	100.0%	2
addEvent	49	100.0%	12
addProperty	33	100.0%	9
getBestProperty	107	100.0%	2
getEvents	61	100.0%	3
getProperties	45	100.0%	3
getUsers	29	100.0%	3
main	329	100.0%	1
registerUser	17	100.0%	12

removeEvent	55	100.0%	2
removeProperty	39	100.0%	4
removeUser	23	100.0%	8
seachEventByCat	95	100.0%	1
searchPropertyByCity	66	100.0%	3
searchPropertyByEvent	78	100.0%	2
testAddEvent	253	100.0%	1
testAddProperty	204	100.0%	1
testBestProperty	322	100.0%	1
testGetEvents	285	100.0%	1
testGetProperties	240	100.0%	1
testGetUsers	192	100.0%	1
testRegisterUser	150	100.0%	1
testRemoveEvent	271	100.0%	1
testRemoveProperty	220	100.0%	1
testRemoveUser	173	100.0%	1
testSeachEventByCat	310	100.0%	1
testSeachPropertyByCity	295	100.0%	1
testSeachPropertyByEvent	303	100.0%	3
MagicStay.vdmpp		100.0%	82

Tabela 7: Cobertura da classe MagicStay

6 Geração de Código Java

A geração do código java foi possível através da funcionalidade do Overtune para esse efeito. Segue de seguida o código gerado.

7 Conclusões

Em suma, com este trabalho foi possível aprender a desenvolver e testar modelos formais de VDM++ através do IDE Overtune. No entanto, o trabalho podia ter sido mais aprofundado com o desenvolvimento de novas classes e outras funções presentes no *MagicStay*.

Relativamente à divisão de trabalho, o trabalho apresentado foi desenvolvido pelo membro Luís Cruz.

Concluindo, estamos satisfeitos pela oportunidade de realizar este trabalho e com o resultado obtido.

Referências

[1] Magic Stay `https://www.magicstay.com/`, 2 1 2018.

[2] Descrição do tema e estrutura do relatório `https://moodle.up.pt/pluginfile.php/165034/mod_resource/content/0/MFES-TrabVDM-1718.pdf`, 2 1 2018.

[3] Overtune Quick Start, `https://moodle.up.pt/pluginfile.php/164632/mod_resource/content/0/OverturQuickStartExercise.pdf`, 2 1 2018.

[4] Overture VDM-10 Language Manual, `https://moodle.up.pt/pluginfile.php/51409/mod_resource/content/2/VDM10_lang_man.pdf`, 2 1 2018.