FIL - Centro de Exposições de Lisboa

Relatório Final

Mestrado Integrado em Engenharia Informática e Computação

Métodos Formais em Engenharia de Software

Turma 4, Tópico 12:
José Manuel Faria Azevedo - 201506448
Rúben José da Silva Torres - 201405612

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

7 de Janeiro de 2019

Conteúdo

1 Descrição Informal do Sistema e Requisitos do Sistema

1.1 Descrição Informal do Sistema

O Centro de Exposições de Lisboa (FIL) é um centro onde são realizados diversos eventos. O centro é composto por várias salas, onde podem ser organizados os diferentes eventos. Para além do registo de eventos, é mantido o registo de visitantes e organizadores de eventos. Cada evento têm associado uma sala onde este é realizado, o horário do evento, o organizador que o organiza, a lista de participantes e o preço por cada participante. O horário contém a data inicial e final do respetivo evento e cada sala tem um limite de ocupantes que não pode ser excedido.

De seguida é apresentado um diagrama que representa a organização do centro de exposições:

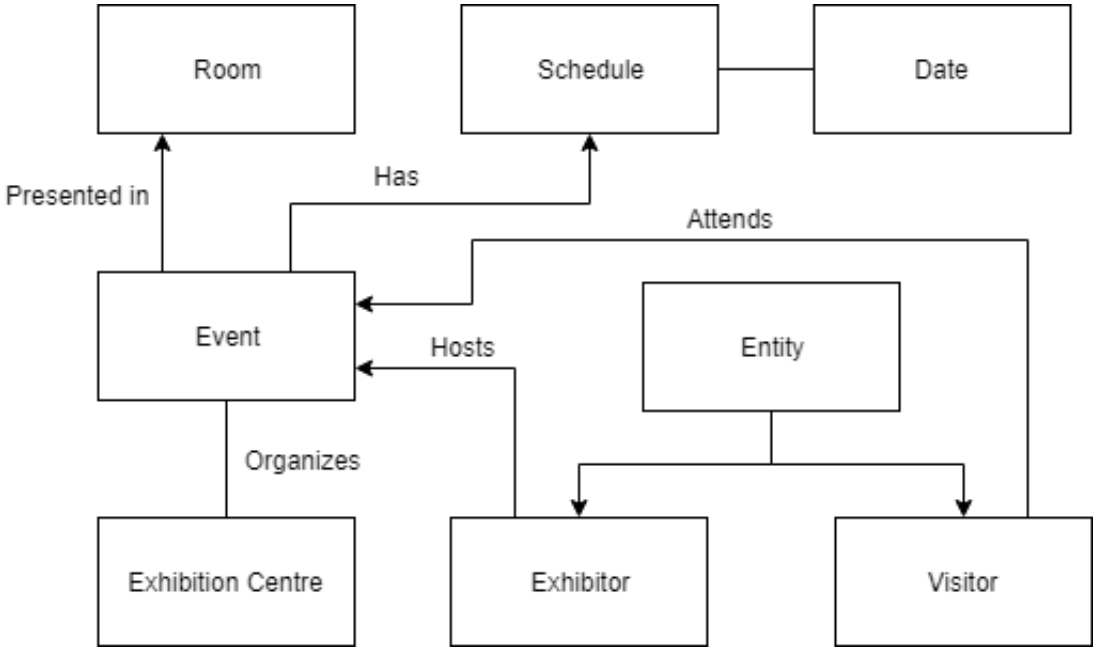

Figura 1: Organização do Centro de Exposições

1.2 Lista de Requisitos

ID	Descrição	Restrições
R1	Registar visitante	Visitante não pode estar já inscrito no centro.
R2	Registar organizador	Organizador não pode estar já inscrito no centro.
R3	Apagar visitante	Visitante tem que estar registado no centro, é removido a sua participação em todos os eventos que não começaram.
R4	Apagar organizador	Organizador tem que estar inscrito no centro e são removidos todos os eventos criados pelo mesmo que não começaram e que não se encontram fechados, ou seja são removidos todos os eventos futuros.
R5	Adicionar evento	O evento criado não pode ser organizado numa sala e num horário que se sobreponha a outro evento por ocorrer.
R6	Remover evento	O evento não pode ter começado e não pode estar dado como fechado
R7	Adicionar participante(s) a evento	O(s) participante(s) tem/têm que ser visitante(s) registados do centro e a sala reservada para o evento tem que ter espaço suficiente para todos os participantes
R8	Remover participante(s) de evento	O(s) participante(s) tem/têm que ser visitante(s) registados do centro e o evento tem de estar por ocorrer
R9	Adicionar Sala	Sala não pode estar já inscrita no centro
R10	Remover Sala	Sala tem que pertencer ao centro
R11	Fechar Eventos	Eventos são fechados quando data atual é maior que a data de fecho do evento
R12	Obter relatório com diferentes métricas, relativos a eventos/visitantes/organizadores	Evento/Visitante/Organizador tem que estar registado no centro
R13	Cancelar Evento	Evento tem de estar por ocorrer

2 Modelos UML

2.1 Modelo de Casos de Uso

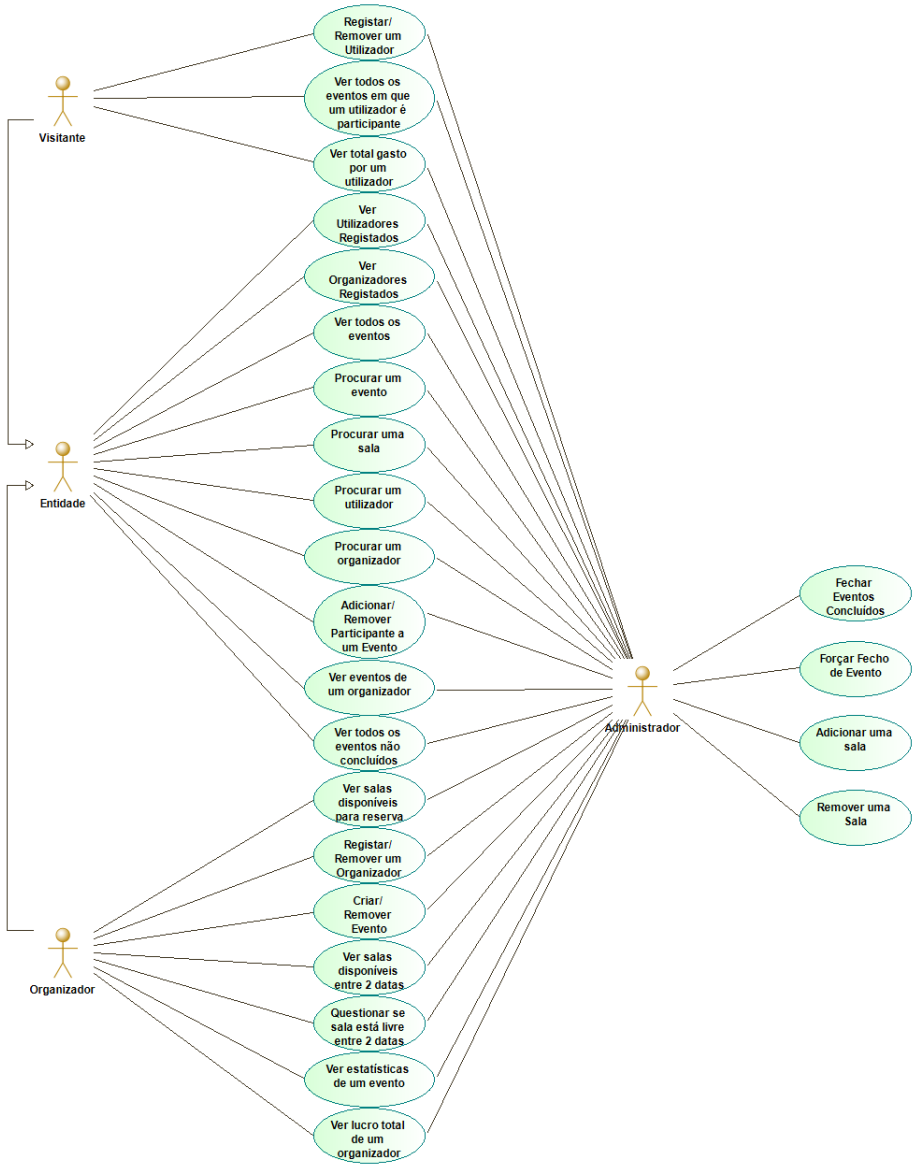

Figura 2: Modelo de Casos de Uso

Cenário	Registar um utilizador
Descrição	Adiciona um utilizador à lista de visitantes do centro.
Pré-Condições	N/A.
Pós-Condições	Utilizador pertence à lista de visitantes do centro.
Passos	N/A.
Exceções	N/A.

Cenário	Remover um utilizador
Descrição	Remove um utilizador da lista de visitantes do centro.
Pré-Condições	Utilizador pertence à lista de visitantes do centro.
Pós-Condições	Utilizador não pertence à lista de visitantes do centro e é removido de todos os eventos futuros em que esteja inscrito.
Passos	N/A.
Exceções	N/A.

Cenário	Ver todos os eventos em que um utilizador é participante
Descrição	Retorna todos os eventos que o utilizador participou ou tenciona participar.
Pré-Condições	Utilizador pertence à lista de visitantes do centro.
Pós-Condições	N/A.
Passos	N/A.
Exceções	N/A.

Cenário	Ver total gasto por um utilizador
Descrição	Retorna a quantia total que eu utilizador gastou pela participação em eventos do centro.
Pré-Condições	Utilizador pertence à lista de visitantes do centro.
Pós-Condições	N/A.
Passos	O valor retornado é obtido a partir do somatório do preço por evento de todos os eventos em que o visitante foi inscrito.
Exceções	N/A.

Cenário	Ver utilizadores registados
Descrição	Retorna todos os visitantes registados no centro.
Pré-Condições	N/A.
Pós-Condições	N/A.
Passos	N/A.
Exceções	N/A.

Cenário	Ver organizadores registados
Descrição	Retorna todos os organizadores registados no centro.
Pré-Condições	N/A.
Pós-Condições	N/A.
Passos	N/A.
Exceções	N/A.

Cenário	Ver todos os eventos
Descrição	Retorna todos os eventos organizados no centro.
Pré-Condições	N/A.
Pós-Condições	N/A.
Passos	N/A.
Exceções	N/A.

Cenário	Procurar um evento
Descrição	Retorna o evento pretendido.
Pré-Condições	Evento organizado no centro.
Pós-Condições	N/A.
Passos	Se argumento igual ao *id* de um dos eventos organizados pelo centro, este é retornado.
Exceções	N/A.

Cenário	Procurar uma sala
Descrição	Retorna a sala pretendida.
Pré-Condições	Sala pertence ao centro.
Pós-Condições	N/A.
Passos	Se argumento igual a *id* de uma sala do centro, esta é retornada.
Exceções	N/A.

Cenário	Procurar um Utilizador
Descrição	Retorna o visitante pretendido.
Pré-Condições	Visitante está registado no centro.
Pós-Condições	N/A.
Passos	Se argumento igual a *id* de um visitante, este é retornado.
Exceções	N/A.

Cenário	Procurar um Organizador
Descrição	Retorna o organizador pretendido.
Pré-Condições	Organizador está registado no centro.
Pós-Condições	N/A.
Passos	Se argumento igual a *id* de um organizador, este é retornado.
Exceções	N/A.

Cenário	Adicionar participante a um evento
Descrição	Adiciona um visitante à lista de participantes de um evento.
Pré-Condições	Evento e Visitante registados no centro e Evento não acabou e sala do evento não esgotada.
Pós-Condições	Visitante pertence à lista de participantes do evento.
Passos	Procura o evento passado como argumento e adiciona o visitante à lista de participantes.
Exceções	N/A.

Cenário	Remover participante a um evento
Descrição	Remove um visitante da lista de participantes de um evento.
Pré-Condições	Evento e visitante registados no centro e evento é um evento futuro.
Pós-Condições	Visitante não pertence à lista de participantes do evento.
Passos	Procura o evento passado como argumento e remove o visitante da lista de participantes.
Exceções	N/A.

Cenário	Ver eventos de um organizador
Descrição	Retorna todos os eventos organizados por um dado organizador.
Pré-Condições	Organizador está registado no centro.
Pós-Condições	N/A.
Passos	Procura eventos cujo organizador foi aquele passado como argumento.
Exceções	N/A.

Cenário	Ver todos os eventos não concluídos
Descrição	Retorna todos os eventos futuros e a decorrer.
Pré-Condições	N/A.
Pós-Condições	N/A.
Passos	Procura eventos organizados no centro cuja variável testitclosed seja *false*.
Exceções	N/A.

Cenário	Ver salas disponíveis para reserva
Descrição	Retorna salas disponíveis para reserva.
Pré-Condições	Sala pertence ao centro.
Pós-Condições	N/A.
Passos	N/A.
Exceções	N/A.

Cenário	Registar organizador
Descrição	Adiciona organizador à lista de organizadores do centro.
Pré-Condições	N/A.
Pós-Condições	Organizador pertence à lista de organizadores
Passos	N/A.
Exceções	N/A.

Cenário	Remove organizador
Descrição	Remove organizador da lista de organizadores do centro.
Pré-Condições	Organizador pertence à lista de organizadores do centro.
Pós-Condições	Organizador não pertence à lista de organizadores.
Passos	Remove-o da lista de organizadores e remove todos os eventos futuros organizados pelo mesmo.
Exceções	N/A.

Cenário	Criar evento
Descrição	Adiciona evento à lista de eventos do centro.
Pré-Condições	Sala e organizador fornecidos pertencem à lista de salas e organizadores, respetivamente, do centro. Sala fornecida não ocupada durante o período compreendido entre as duas datas fornecidas.
Pós-Condições	Evento pertence à lista de eventos do centro.
Passos	É criado um horário com as datas e sala fornecidas como argumentos e é criado o evento.
Exceções	N/A.

Cenário	Remover evento
Descrição	Remove evento da lista de eventos do centro.
Pré-Condições	Evento por ocorrer.
Pós-Condições	Event não pertence à lista de eventos do centro.
Passos	N/A.
Exceções	N/A.

Cenário	Ver salas disponíveis entre 2 datas
Descrição	Retorna salas disponíveis durante o período compreendido entre as duas datas dadas como argumentos.
Pré-Condições	Data dada como primeiro argumento menor ou igual à data dada como segundo argumento.
Pós-Condições	N/A.
Passos	Procura por todas as salas e todos os eventos registados aqueles que não coincidam com o período compreendido entre as 2 datas fornecidas.
Exceções	N/A.

Cenário	Questionar se sala está disponível entre 2 datas
Descrição	Verifica se sala dada como argumento está disponível no período compreendido entre as duas datas dadas como argumento.
Pré-Condições	Data dada como primeiro argumento menor ou igual à data dada como segundo argumento e sala pertence à lista de salas do centro.
Pós-Condições	N/A.
Passos	Procura todos os eventos registados cuja sala corresponda à sala fornecida e coincidam com o período compreendido entre as 2 datas fornecidas. Caso não existam retorna textittrue, caso contrário textitfalse.
Exceções	N/A.

Cenário	Ver estatísticas de um evento
Descrição	Devolve preço por participante, custo, valor de vendas, lucro e lista de participantes de um determinado evento.
Pré-Condições	Evento pertence à lista de eventos do centro.
Pós-Condições	N/A.
Passos	N/A.
Exceções	N/A.

Cenário	Ver lucro total de um organizador
Descrição	Devolve o lucro total de um organizador de eventos do centro.
Pré-Condições	Organizador pertence à lista de organizadores do centro.
Pós-Condições	N/A.
Passos	O lucro total é obtido a partir do somatório de lucro de todos os eventos cujo organizador seja aquele dado como argumento.
Exceções	N/A.

Cenário	Fechar Eventos Concluídos
Descrição	Fecha todos os eventos cuja data final tenha sido ultrapassada pela data atual.
Pré-Condições	Evento organizado no centro e não concluído.
Pós-Condições	Eventos cuja data final maior que a atual são fechados.
Passos	É obtida a data atual e é percorrida a lista de eventos, filtrando aqueles que já tenham sido concluídos, e é verificado se data final é maior que a atual. Caso se confirme, esse evento é fechado.
Exceções	N/A.

Cenário	Forçar fecho de evento
Descrição	Cancela evento dado como argumento.
Pré-Condições	Evento organizado no centro e não concluído.Password passada como argumento igual à password especial para fecho de evento.
Pós-Condições	Evento fechado e lista de participantes limpa.
Passos	É procurado o evento pretendido na lista de eventos e este é fechado caso password passada como argumento esteja correta.
Exceções	N/A.

Cenário	Adicionar uma sala
Descrição	Adiciona sala ao centro.
Pré-Condições	N/A.
Pós-Condições	N/A.
Passos	É criada uma sala com o espaço e preço por participante dados como argumentos.
Exceções	N/A.

Cenário	Remover uma sala
Descrição	Remove sala do centro.
Pré-Condições	Sala pertence ao centro.
Pós-Condições	Sala não pertence ao centro.
Passos	N/A.
Exceções	N/A.

2.2 Diagrama de Classes

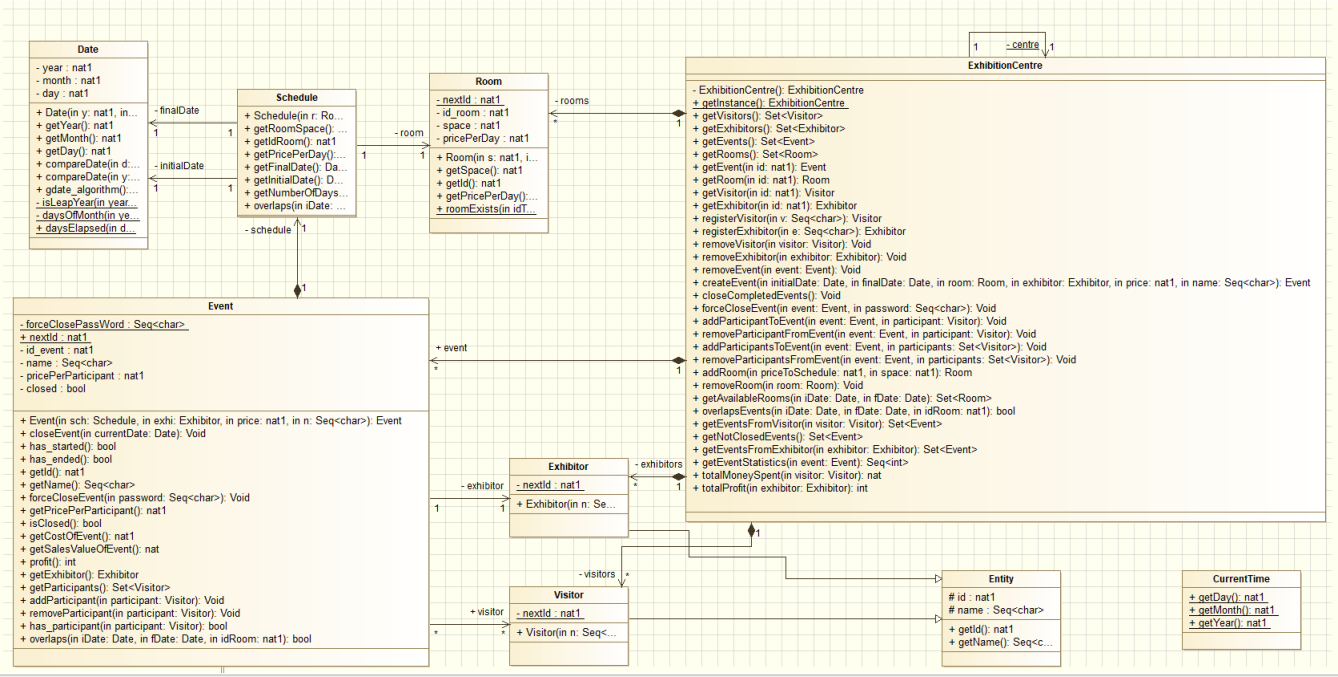

Figura 3: Diagrama de classes

Date: Representa uma data no formato yyyy/mm/dd.
Room: Representa uma sala a ser reservada, contém preço da reserva e o número máximo de pessoas que pode conter.
Schedule: Contém duas datas e uma sala, representa a reserva da sala entre estas duas datas.
Entity: Classe abstrata que representa uma entidade.
Exhibitor: Representa uma entidade, com o poder de organizar em eventos.
Visitor: Representa uma entidade, com o poder de participar em eventos.
Evento: Representa a reserva de uma sala dentro de duas datas, para um Exhibitor ter a possibilidade de mostrar, aos participantes interessados, a sua arte.
ExhibitionCentre: Contém todos os Eventos, Rooms, Visitor's, e Exhibitor's, pode ser vista como uma api, ou seja é a 1º camada de abstração de todo o código concebido.
CurrentTime: Uma biblioteca externa personalizada com o único objetivo de retornar a data atual.

3 Modelo Formal VDM++

3.1 Date

```
/*

  Class that represents a date;
  A date has a year, month, and day.

*/
class Date

instance variables
 private year : nat1;
 inv year>=2019;

 private month : nat1;
 inv month >= 1 and month <= 12;

 private day : nat1;
 inv day <= daysOfMonth(year,month);

operations
 -- Constructor

 public Date : nat1*nat1*nat1 ==> Date
  Date(y,m,d) ==
  (
   year := y;
   month := m;
   day := d;
   return self;
  );

 -- 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

 public pure compareDate : Date ==> bool
  compareDate(d) ==
  (
   if(d.getYear()<year)
    then return true
   else if(d.getYear()>year)
    then return false
   else if(d.getMonth()<month)
    then return true
   else if(d.getMonth()>month)
    then return false
```

```
   else if (d.getDay()<day)
    then return true
   else return false
  );

 public pure compareDate : nat1 * nat1 * nat1 ==> bool
  compareDate(y,m,d) ==
  (
   if(y<year)
    then return true
   else if(y>year)
    then return false
   else if(m<month)
    then return true
   else if(m>month)
    then return false
   else if (d<day)
    then return true
   else return false
  );

  --converts the gregorian date from yy/mm/dd to days

 pure public gdate_algorithm : () ==>nat1
  gdate_algorithm() == (
  (dcl m: nat := (month + 9) mod 12,
   y: nat := year - floor(m/10);

   return 365*y + floor(y/4) - floor(y/100) + floor(y/400) + floor((m*306 + 5)/10) + ( day - 1
        ))
  );

functions

 --checks if the given year is a leap year
 --returns true if it is a leap year, false otherwiser

 private isLeapYear : nat1 +> bool
  isLeapYear(year) == (year mod 4 = 0 and year mod 100 <> 0 or year mod 400 = 0)
 pre year>=2019;

 -- returns the number days of a given month

 private daysOfMonth : nat1 * nat1 +> nat1
  daysOfMonth(year, month) == (
   cases month : 1, 3, 5, 7, 8, 10, 12 -> 31, 4, 6, 9, 11 -> 30, 2 -> if isLeapYear(year) then
        29 else 28 end
 )
 pre year>=2019 and month >= 1 and month <= 12;

 --Counts the number of days between 2 dates, the first date also counts so that 2 equals
     dates returns 1;

 public daysElapsed : Date * Date +> nat1
 daysElapsed(date1, date2) == (
  date2.gdate_algorithm()-date1.gdate_algorithm()+1
 )
 pre date2.compareDate(date1) or ((date2.compareDate(date1) or date1.compareDate(date2)) =
     false);
 -- that date1 needs to be <= to the date2
end Date
```

Function or operation	Line	Coverage	Calls
Date	21	100.0%	24
compareDate	44	100.0%	44
daysElapsed	101	100.0%	183
daysOfMonth	94	100.0%	24
gdate_algorithm	77	100.0%	130
getDay	39	100.0%	730
getMonth	35	100.0%	1580
getYear	31	100.0%	1795
isLeapYear	89	100.0%	3
Date.vdmpp		100.0%	4513

3.2 Room

```
/*

  Class that represents a Room;
 A room has maximum number of people that can be present and a price associated to reserving
     it for a day;
 Also it has a unique set of Id's independent from other classes;

*/
class Room
instance variables

 private static nextId : nat1 := 1;
 private id_room : nat1;
 private space: nat1;
 private pricePerDay: nat1;

operations
 -- Constructor

 public Room : nat1 * nat1 ==> Room
  Room(s,ppd) == (
   space := s;
   pricePerDay := ppd;
   id_room := nextId;
   nextId := nextId + 1;
   return self;
  )
  post nextId>id_room;

 --returns the number of people that a room is allowed to have present at the same time

 public pure getSpace : () ==> nat1
  getSpace() == (return space);

 --returns the id of the room

 public pure getId : () ==> nat1
  getId() == (return id_room);

 --returns the price of a room reservation per day

 public pure getPricePerDay : () ==> nat1
  getPricePerDay() == (return pricePerDay);

 --returns true if a room Exists, false otherwise
```

```
 public pure static roomExists : nat1 ==> bool
  roomExists(idToCheck) == (return idToCheck<nextId);

end Room
```

Function or operation	Line	Coverage	Calls
Room	18	100.0%	15
getId	33	100.0%	1325
getPricePerDay	37	100.0%	36
getSpace	29	100.0%	66
roomExists	41	100.0%	304
Room.vdmpp		100.0%	1746

3.3 Schedule

```
/*

  Class that represents a Reservation of a room between 2 dates;

*/

class Schedule

instance variables
  private room:Room;
  private initialDate:Date;

  private finalDate:Date;
 inv (finalDate.compareDate(initialDate)
   or ((finalDate.compareDate(initialDate) or initialDate.compareDate(finalDate)) = false))
  and Date`daysElapsed(initialDate,finalDate)>0;
 -- The final date needs to be bigger or equal to the initial date

operations
 --Constructor

 public Schedule : Room*Date*Date ==> Schedule
  Schedule(r, iDate, fDate) == (
   room := r;
   initialDate := iDate;
   finalDate := fDate;
   return self;
  );

 --returns the maximum number of allowed people that is present in the room

 public pure getRoomSpace : () ==> nat1
  getRoomSpace() == (return room.getSpace());

 --returns the id of the reserved room

 public pure getIdRoom : () ==> nat1
  getIdRoom() == (return room.getId());

 --returns the price per day of the reserved room
```

```
 public pure getPricePerDay : () ==> nat1
  getPricePerDay()== (return room.getPricePerDay());

 --returns the final date

 public pure getFinalDate : () ==> Date
  getFinalDate()== (return finalDate);

 --returns the initial date

 public pure getInitialDate : () ==> Date
  getInitialDate()== (return initialDate);

 --returns the number of days beetween the initial date and the final date

 public pure getNumberOfDays : () ==>nat1
  getNumberOfDays() == (return Date`daysElapsed(initialDate,finalDate));

 --checks if 2 given dates overlap this schedule/reservation
 --returns true if overlaps this reservation, false otherwise

 public pure overlaps : Date * Date * nat1 ==> bool
  overlaps(iDate, fDate, idRoom) == (
   if idRoom<>room.getId()
    then return false
   else if iDate.compareDate(finalDate)
    then return false
   else if initialDate.compareDate(fDate)
    then return false
   else return true
  )
  pre (fDate.compareDate(iDate) or ((fDate.compareDate(iDate) or iDate.compareDate(fDate)) =
      false)) and Room`roomExists(idRoom);  -- that initial date needs to be <= to the final
      date

end Schedule
```

Function or operation	Line	Coverage	Calls
Schedule	21	100.0%	15
getFinalDate	42	100.0%	24
getIdRoom	34	100.0%	3
getInitialDate	46	100.0%	23
getNumberOfDays	50	100.0%	33
getPricePerDay	38	100.0%	33
getRoomSpace	30	100.0%	63
overlaps	55	100.0%	300
Schedule.vdmpp		100.0%	494

3.4 Entity

```
/*

  Abstract Class that represents an entity;
  An Entity has a id and a name.

*/
```

```
class Entity

instance variables
 protected id: nat1;

 protected name: seq1 of char;
 inv len name <= 50;

operations
  --Returns the id of an entity

 public pure getId : () ==> nat1
  getId() == (return id);

  --Returns the name of an entity

 public pure getName : () ==> seq1 of char
  getName() == (return name);

end Entity
```

Function or operation	Line	Coverage	Calls
getId	17	100.0%	6478
getName	21	100.0%	32
Entity.vdmpp		100.0%	6510

3.5 Exhibitor

```
/*

  Class that represents a Exhibitor;
  An Exhibitor is a subclass of an entity
  with only a constructor and a unique set of id's independent of other entity subclasses

*/
class Exhibitor is subclass of Entity

instance variables
 private static nextId : nat1 := 1;

operations
  --Constructor

 public Exhibitor : seq1 of char ==> Exhibitor
  Exhibitor(n) == (
   id := nextId;
   nextId := nextId + 1;
   name := n;
   return self;
  )
  post nextId>id;

end Exhibitor
```

Function or operation	Line	Coverage	Calls
Exhibitor	15	100.0%	11
Exhibitor.vdmpp		100.0%	11

3.6 Visitor

```
/*

  Class that represents a Visitor/User;
  An Visitor is a subclass of an entity
  with only a constructor and a unique set of id's independent of other entity subclasses

*/
class Visitor is subclass of Entity

instance variables
 private static nextId : nat1 := 1;

operations
 --Constructor

 public Visitor : seq1 of char ==> Visitor
  Visitor(n) == (
   id := nextId;
   nextId := nextId + 1;
   name := n;
   return self;
  )
  post nextId > id;

end Visitor
```

Function or operation	Line	Coverage	Calls
Visitor	15	100.0%	15
Visitor.vdmpp		100.0%	15

3.7 Event

```
/*

  Class that represents an Event;
  An Event needs to have a reservation of a room between 2 days (schedule),
  a name, and the exhibitor associated with it.
  Each Eventcontrols the flow of participants of itself.
 Also it has a unique set of Id's independent from other classes;
*/
class Event

values
 forceClosePassWord: seq1 of char = "password";

instance variables
 public static nextId : nat1 := 1;
 private id_event : nat1;
```

```
 private name: seq1 of char;
 inv len name <= 50;

  private schedule : Schedule;

  private participants : set of (Visitor);
  inv card participants <= schedule.getRoomSpace()
   and not exists p1, p2 in set participants & p1 <> p2 and p1.getId() = p2.getId() ;

  private exhibitor:Exhibitor;
 private pricePerParticipant : nat1;
 private closed : bool;

operations
 --Constructor

 public Event : Schedule * Exhibitor * nat1 * seq1 of char ==> Event
  Event (sch, exhi, price, n) == (
   schedule:=sch;
   exhibitor:=exhi;
   pricePerParticipant:=price;
   closed := false;
   participants := {};
   id_event := nextId;
   nextId := nextId + 1;
   name :=n;
  )
  post nextId>id_event;

 --Closes the event if Today's date is bigger then it's final date of exhibition

 public closeEvent : Date ==> ()
  closeEvent(currentDate) ==(
   closed:=true;
  )
  pre Date`daysElapsed(schedule.getFinalDate(),currentDate)>1 and
   CurrentTime`getDay()=currentDate.getDay() and
   CurrentTime`getMonth()=currentDate.getMonth() and
   CurrentTime`getYear()=currentDate.getYear() and not closed
  post closed;

 --Checks if today's date is bigger or equal then the event starting date
 --returns true if it is, false otherwise

 public pure has_started : () ==> bool
  has_started() == (
   return not schedule.getInitialDate().compareDate(CurrentTime`getYear(), CurrentTime`
       getMonth(), CurrentTime`getDay());
  );

 --Checks if today's date is bigger or equal then the event closing/final date
 --returns true if it is, false otherwise

 public pure has_ended : () ==> bool
  has_ended() == (
   return not schedule.getFinalDate().compareDate(CurrentTime`getYear(), CurrentTime`getMonth
       (), CurrentTime`getDay());
  );

 --returns the unique id of this event

 public pure getId : () ==> nat1
  getId() == (return id_event);
```

```
--returns the name of this event

public pure getName : () ==> seq1 of char
 getName() == (return name);

--Closes this event on special situations, the outcome is an event that won't happen but the
    exhibitor will still have to pay for it
--Needs a special password as an argument
--Participants are set to an empty set

public forceCloseEvent : (seq1 of char) ==> ()
 forceCloseEvent(password) ==(
  closed:=true;
  participants := {};
 )
 pre not closed and forceClosePassWord = password
 post closed and card participants = 0;

--returns price to participate in this event has a visitorS

public pure getPricePerParticipant : () ==> nat1
 getPricePerParticipant() == (return pricePerParticipant);

--checks if the event is closed (by date or forced)

public pure isClosed : () ==> bool
 isClosed() == (return closed);

--returns the exhibitor cost related to the reservation of the room for the starting and
    closing days

public pure getCostOfEvent : () ==> nat1
 getCostOfEvent() ==(
   return schedule.getPricePerDay()*schedule.getNumberOfDays();
 );

--returns the event sales value

public pure getSalesValueOfEvent : () ==> nat
 getSalesValueOfEvent() ==(
  return (card participants)*pricePerParticipant;
 );

--returns the profit, might be negative if costs related to reservation are bigger then sales

public pure profit : () ==> int
 profit () == (
  return getSalesValueOfEvent()-getCostOfEvent();
 );

--returns the exhibitor/organizer related to this event

public pure getExhibitor : () ==> Exhibitor
 getExhibitor() == (return exhibitor);

--returns all the enrolled participants of this event

public pure getParticipants : () ==> set of Visitor
 getParticipants() == (return participants);

--enrolls a given Visitor/Participant to the event

public addParticipant : Visitor ==> ()
 addParticipant(participant) == (
  participants := participants union {participant};
```

```
 )
 pre participant not in set participants and not closed
 post participant in set participants;

--Disenrolls a given Visitor/Participant from the event , if the latter isn't already closed

public removeParticipant : Visitor ==> ()
 removeParticipant(participant) == (
  participants := participants \ {participant};
 )
 pre participant  in set participants and not closed
 post participant not in set participants;

--Checks if a given visitor is already enrolled in this Event

public pure has_participant :  Visitor ==> bool
 has_participant(participant) == (
  return participant in set participants;
 );

--Checks if the reservation of this event is overlapped by two given dates and a room unique
    id

pure public overlaps : Date * Date * nat1 ==> bool
 overlaps(iDate, fDate, idRoom) == (
  return schedule.overlaps(iDate, fDate, idRoom);
 );

end Event
```

Function or operation	Line	Coverage	Calls
Event	33	100.0%	13
addParticipant	125	100.0%	29
closeEvent	47	100.0%	5
forceCloseEvent	82	100.0%	2
getCostOfEvent	99	100.0%	30
getExhibitor	117	100.0%	124
getId	72	100.0%	3859
getName	76	100.0%	4
getParticipants	121	100.0%	23
getPricePerParticipant	91	100.0%	27
getSalesValueOfEvent	105	100.0%	34
has_ended	66	100.0%	16
has_participant	141	100.0%	134
has_started	59	100.0%	20
isClosed	95	100.0%	331
overlaps	147	100.0%	288
profit	111	100.0%	20
removeParticipant	133	100.0%	11
Event.vdmpp		100.0%	4970

3.8 ExhibitionCentre

```
/*

  Class that represents The Exhibition Centre
  It follows the singleton pattern
 An exhibition Centre has a set of Events, Rooms, Registered Visitors and Exhibitors
 This Class controls all the flow related to the previous mentioned class sets, since it is
     designed to function more like an API
*/

class ExhibitionCentre
 instance variables
 private static centre : ExhibitionCentre := new ExhibitionCentre();

  private events: set of (Event);
  inv not exists ev2, ev3 in set events & ev2 <> ev3 and ev2.getId() = ev3.getId();

  private rooms: set of (Room);
  inv not exists r2, r3 in set rooms & r2 <> r3 and r2.getId() = r3.getId();

  private visitors: set of (Visitor);
  inv not exists v2, v3 in set visitors & v2 <> v3 and v2.getId() = v3.getId();

  private exhibitors: set of (Exhibitor);
  inv not exists e2, e3 in set exhibitors & e2 <> e3 and e2.getId() = e3.getId();


 operations
  -- PRIVATE Constructor

  private ExhibitionCentre : () ==> ExhibitionCentre
   ExhibitionCentre() == (
    events := {};
    rooms := {};
    visitors := {};
    exhibitors := {};
    return self;
   );

  --Returns the unique and only Exhibition Centre

  public static pure getInstance : () ==> ExhibitionCentre
   getInstance() == (
    return centre;
   );

  /*
  Getters
  */
  --Returns a set with All Registered Visitors

  public pure getVisitors : () ==> set of (Visitor)
   getVisitors() == (
    return visitors;
   );

  --Returns a set with All Registered Exhibitors

  public pure getExhibitors : () ==> set of (Exhibitor)
   getExhibitors() == (
    return exhibitors;
   );

  --Returns a set with All Registered Events closed or open

  public pure getEvents : () ==> set of (Event)
```

```
  getEvents() == (
   return events;
  );

 --Returns a set with All Rooms available for reservation

 public pure getRooms : () ==> set of (Room)
  getRooms() == (
   return rooms;
  );

 --Given an Id Returns a specific Event

 public pure  getEvent : nat1 ==> Event
 getEvent(id) == (
  return iota x in set events & x.getId()=id;
 )
 pre exists1 x in set events & x.getId()=id;

--Given an Id Returns a specific room

public pure  getRoom : nat1 ==> Room
 getRoom(id) == (
   return iota x in set rooms & x.getId()=id;
 )
 pre exists1 x in set rooms & x.getId()=id;

--Given an Id Returns a specific Visitor

public pure  getVisitor : nat1 ==> Visitor
 getVisitor(id) == (
   return iota x in set visitors & x.getId()=id;
 )
 pre exists1 x in set visitors & x.getId()=id;

--Given an Id Returns a specific Exhibitor

public pure  getExhibitor : nat1 ==> Exhibitor
 getExhibitor(id) == (
   return iota x in set exhibitors & x.getId()=id;
 )
 pre exists1 x in set exhibitors & x.getId()=id;

 /*
  Entity Related Methods
 */

 --Given a Name creates a new Visitor and adds it to the set of visitors
 --Returns the newly created Visitor

 public registerVisitor : seq1 of char ==> Visitor
  registerVisitor(v) == (
   (dcl visitor: Visitor := new Visitor(v);
    visitors := visitors union {visitor};
    return visitor;
   );
  )
  post exists v1 in set visitors & v1.getName()=v;

 --Given a Name creates a new Exhibitor and adds it to the set of exhibitors
 --Returns the newly created Exhibitor

 public registerExhibitor : seq1 of char ==> Exhibitor
  registerExhibitor(e) == (
   (dcl exhibitor: Exhibitor := new Exhibitor(e);
```

```
    exhibitors := exhibitors union { exhibitor };
    return exhibitor;
   );
  )
  post exists e1 in set exhibitors & e1.getName()=e;

 --Given a visitor deletes it from the set of visitors
 --removes all the future events that this visitor is enrolled on
 --closed and already started events are untouched

 public removeVisitor : Visitor ==> ()
 removeVisitor(visitor) == (
  visitors := visitors \ {visitor};
  for all event in set events do
   if event.has_participant(visitor)
    and not event.has_started()
    and not event.isClosed()
    then event.removeParticipant(visitor);
 )
 pre visitor in set visitors
 post visitor not in set visitors
  and not exists event in set events & event.has_participant(visitor)
   and not event.isClosed() and not event.has_started();

--Given a exhibitor deletes it from the set of exhibitors
 --removes all the future events that this exhibitor created
 --closed and already started events are untouched

public removeExhibitor: Exhibitor ==> ()
 removeExhibitor(exhibitor) == (
  exhibitors := exhibitors \ {exhibitor};
  for all event in set events do
   if not event.isClosed()
    and event.getExhibitor()=exhibitor
    and not event.has_started()
    then removeEvent(event);
 )
 pre exhibitor in set exhibitors
 post exhibitor not in set exhibitors
   and not exists event in set events & event.getExhibitor()=exhibitor and  not event.
       isClosed() and not event.has_started();

/*
  Event Related Methods
 */

 --If a given event is not closed and has not started, removes it from the event set

public removeEvent: Event ==> ()
 removeEvent (event) == (
  events := events \ {event};
 )
 pre not event.isClosed() and event in set events
  and not event.has_started();

--If it doens't overlap any of the already created events,
--creates a new event and adds it to the event set
--Returns the newly created Event

public createEvent: Date*Date*Room*Exhibitor*nat1*seq1 of char ==> Event
 createEvent(initialDate,finalDate,room,exhibitor, price, name) == (
  (dcl shcedule: Schedule := new Schedule(room,initialDate,finalDate),
   eventtoadd : Event := new Event (shcedule,exhibitor,price,name);
   events:=events union {eventtoadd};
   return eventtoadd;
```

```
   );
  )
  pre exhibitor in set exhibitors
   and room in set rooms
   and forall event in set events &
    not event.overlaps(initialDate, finalDate, room.getId()) or event.isClosed()
  post exists1 event in set events & event.overlaps(initialDate, finalDate, room.getId())
   and not event.isClosed();

 --Closes all the concluded events, that is, all the events that the final date is smaller
     than the current date

 public closeCompletedEvents : () ==> ()
  closeCompletedEvents() == (
  (dcl currentDate : Date := new Date(CurrentTime`getYear(), CurrentTime`getMonth(),
      CurrentTime`getDay());
   for all event in set events
    do if(event.has_ended() and not event.isClosed())
     then event.closeEvent(currentDate);
    );
  )
  post not exists event in set events & event.has_ended()
   and not event.isClosed();

 --Given a password and an event forces the closure of it

 public forceCloseEvent : Event*seq1 of char ==> ()
  forceCloseEvent(event,password) == (
   event.forceCloseEvent(password);
  )
  pre event in set events;

 --Adds a given Visitor to the set of the given event participants

 public addParticipantToEvent : Event * Visitor ==> ()
  addParticipantToEvent(event,participant) == (
   event.addParticipant(participant);
  )
  pre event in set events and participant in set visitors;

 --Removes a given Visitor From the set of the given event participants

 public removeParticipantFromEvent : Event * Visitor ==> ()
  removeParticipantFromEvent(event,participant) == (
   event.removeParticipant(participant);
  )
  pre event in set events and participant in set visitors;

 --Adds a given set of Visitor's to the set of the given event participants

 public addParticipantsToEvent : Event * set1 of Visitor ==> ()
  addParticipantsToEvent(event,participants) == (
   for all participant in set participants
    do event.addParticipant(participant);
  )
  pre event in set events and (participants subset visitors);

 --Removes a given set of Visitor's From the set of the given event participants

 public removeParticipantsFromEvent : Event * set1 of Visitor ==> ()
  removeParticipantsFromEvent(event,participants) == (
   for all participant in set participants
    do event.removeParticipant(participant);
  )
  pre event in set events and (participants subset visitors);
```

```
--Creates a new Room for this exhibition Centre
--Returns the newly created room

public addRoom : nat1 * nat1 ==> Room
 addRoom(priceToSchedule, space) == (
 (dcl room : Room := new Room(space,priceToSchedule);
   rooms := rooms union {room};
   return room;
   );
  );

--Removes a given room from the exhibition Centre
--All scheduled events are left untouched, it just precludes the reservation of the room to
    future created events

public removeRoom : Room ==> ()
 removeRoom (room) == (
   rooms := rooms \ {room};
 )
 pre room in set rooms
 post room not in set rooms;

/*
  Reports with several metrics
 */

 --Returns a set of Room that are available of reservation between the two given dates

public pure getAvailableRooms: Date*Date ==> set of Room
 getAvailableRooms(iDate, fDate) == (
  return {room | room in set rooms
   & forall event in set events &
    not event.overlaps(iDate, fDate, room.getId()) or event.isClosed()};
 )
 pre fDate.compareDate(iDate)
  or ((fDate.compareDate(iDate) or fDate.compareDate(iDate)) = false);

--Given a room id checks if it is for reservation between the two given dates

public pure overlapsEvents: Date*Date*nat1 ==> bool
 overlapsEvents(iDate,fDate,idRoom) == (
  return exists event in set events & event.overlaps(iDate, fDate, idRoom) and not event.
      isClosed()
 )
 pre fDate.compareDate(iDate)
  or ((fDate.compareDate(iDate) or fDate.compareDate(iDate)) = false)
  and exists1 x in set rooms & x.getId()=idRoom;

--Returns All the Events that a visitor is or was enrolled on

public pure getEventsFromVisitor : Visitor ==> set of Event
 getEventsFromVisitor(visitor) == (
  return {event | event in set events & event.has_participant(visitor)};
 )
 pre visitor in set visitors;

--Returns all the ongoing and future events

public pure getNotClosedEvents : () ==> set of Event
 getNotClosedEvents() == (
  return {event | event in set events & not event.isClosed()};
 );

--Returns All the Events that a Exhibitor created/organized
```

```
 public pure getEventsFromExhibitor : Exhibitor ==> set of Event
  getEventsFromExhibitor(exhibitor) == (
   return {event | event in set events & event.getExhibitor() = exhibitor};
  )
  pre exhibitor in set exhibitors;

 --Given an Event returns a set with the price per participant, cost of the event,
 --the event sales value, the profit for the exhibitor and the number of participants

 public pure getEventStatistics: Event ==> seq1 of int
  getEventStatistics(event) == (
    return [event.getPricePerParticipant(), event.getCostOfEvent(), event.getSalesValueOfEvent
        (), event.profit(), card event.getParticipants()];
  );

 --Returns the total money that a visitor spent on the exhibiton Centre

 public pure totalMoneySpent : Visitor ==> nat
  totalMoneySpent (visitor) == (
   (dcl total: nat := 0;
    for all event in set events
     do if event.has_participant(visitor)
      then total := total+event.getPricePerParticipant();
    return total;
   );
  )
  pre visitor in set visitors;

 --Returns the total profit that a Exhibitor gained on the exhibiton Centre

 public pure totalProfit : Exhibitor ==> int
  totalProfit (exhibitor) == (
   (dcl total: int := 0;
    for all event in set events
     do if event.getExhibitor() = exhibitor
      then total := total+event.profit();
    return total;
   );
  )
  pre exhibitor in set exhibitors;

 thread
   closeCompletedEvents();

end ExhibitionCentre
```

Function or operation	Line	Coverage	Calls
ExhibitionCentre	28	100.0%	1
addParticipantToEvent	222	100.0%	15
addParticipantsToEvent	236	100.0%	3
addRoom	253	100.0%	6
closeCompletedEvents	203	100.0%	1
createEvent	187	100.0%	9
forceCloseEvent	215	100.0%	1
getAvailableRooms	275	100.0%	9
getEvent	71	100.0%	15
getEventStatistics	315	100.0%	6

getEvents	59	100.0%	8
getEventsFromExhibitor	307	100.0%	7
getEventsFromVisitor	294	100.0%	7
getExhibitor	102	100.0%	2
getExhibitors	53	100.0%	1
getInstance	38	100.0%	2
getNotClosedEvents	301	100.0%	39
getRoom	78	100.0%	3
getRooms	65	100.0%	6
getVisitor	90	100.0%	2
getVisitors	47	100.0%	1
overlapsEvents	285	100.0%	18
registerExhibitor	130	100.0%	5
registerVisitor	119	100.0%	5
removeEvent	177	100.0%	4
removeExhibitor	159	100.0%	3
removeParticipantFromEvent	229	100.0%	1
removeParticipantsFromEvent	244	100.0%	2
removeRoom	263	100.0%	3
removeVisitor	142	100.0%	2
totalMoneySpent	321	100.0%	7
totalProfit	333	100.0%	5
ExhibitionCentre.vdmpp		100.0%	199

3.9 CurrentTime (Custom External Library)

Tendo em conta que VDM10 possui a limitação de não conter funções de sistema que retornem a data atual, o nosso grupo sentiu a necessidade de incluir uma biblioteca externa criada por nós. Esta biblioteca tem como objetivo apenas retornar o dia, mês e ano atual. Para criar esta biblioteca externa é apenas necessário gerar um jar partir do nosso código em Java (classe CurrentTime) e adicionar na pasta ./lib do projeto de vdm++, após este processo estar concluído criamos a classe vdm++ correspondente.

3.9.1 VDM++ CurrentTime

```
class CurrentTime

operations

public pure static getDay : () ==> nat1
getDay() == is not yet specified;


public pure static getMonth : () ==> nat1
getMonth() == is not yet specified;


public pure static getYear : () ==> nat1
getYear() == is not yet specified;

end CurrentTime
```

3.9.2 Código Java Utilizado para gerar biblioteca externa

```
import java.util.Calendar;
import java.util.Date;

import org.overture.interpreter.values.NaturalOneValue;
import org.overture.interpreter.values.Value;

public class CurrentTime {
  public static Value getDay() throws Exception {
    Date today = new Date();
    Calendar cal = Calendar.getInstance();
    cal.setTime(today); // don't forget this if date is arbitrary
    int day = cal.get(Calendar.DAY_OF_MONTH);

    return new NaturalOneValue(day);
  }

  public static Value getMonth() throws Exception {
    Date today = new Date();
    Calendar cal = Calendar.getInstance();
    cal.setTime(today); // don't forget this if date is arbitrary
    int month = cal.get(Calendar.MONTH);

    return new NaturalOneValue(month+1);
  }

  public static Value getYear() throws Exception {
    Date today = new Date();
    Calendar cal = Calendar.getInstance();
    cal.setTime(today); // don't forget this if date is arbitrary
    int year = cal.get(Calendar.YEAR);

    return new NaturalOneValue(year);
  }
}
```

4 Modelo de Validação

4.1 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
```

4.2 TestDate

```
class TestDate is subclass of MyTestCase

instance variables
 private date1: Date := new Date(2019,1,31);
 private date2: Date := new Date(2019,2,28);
 private date3: Date := new Date(2020,2,29);
 private date4: Date := new Date(2019,4,1);
 private date5: Date := new Date(2019,1,1);
 private date6: Date := new Date(2019,1,31);
 private date7: Date:= new Date(2019,3,1);
 private date8: Date:= new Date(2020,3,1);

operations

 --Checks if Date Objects are being created correctly and returning the correct values for the
      year month and day

 private testGets: () ==> ()
  testGets() == (
   assertEqual(2019,date1.getYear());
   assertEqual(1,date1.getMonth());
   assertEqual(31,date1.getDay());
   assertEqual(2019,date2.getYear());
   assertEqual(2,date2.getMonth());
   assertEqual(28,date2.getDay());
   assertEqual(2020,date3.getYear());
   assertEqual(2,date3.getMonth());
   assertEqual(29,date3.getDay());
   assertEqual(2019,date4.getYear());
   assertEqual(4,date4.getMonth());
   assertEqual(1,date4.getDay());

 );

 --Checks if operation to compare Dates works correctly (datex.compareDate(datey) is true when
      datex > date y)

 private testCompareDate: () ==> ()
  testCompareDate() == (
   assertTrue(not date1.compareDate(date3));
   assertTrue(date3.compareDate(date1));
   assertTrue(date1.compareDate(date5));
   assertTrue(not date5.compareDate(date1));
   assertTrue(date4.compareDate(date1));
   assertTrue(not date1.compareDate(date4));
   assertTrue(not date1.compareDate(date6));
   assertTrue(not date6.compareDate(date1));

   assertTrue(not date1.compareDate(date3.getYear(),date3.getMonth(),date3.getDay()));
   assertTrue(date3.compareDate(date1.getYear(),date1.getMonth(),date1.getDay()));

   assertTrue(date1.compareDate(date5.getYear(),date5.getMonth(),date5.getDay()));
   assertTrue(not date5.compareDate(date1.getYear(),date1.getMonth(),date1.getDay()));
   assertTrue(date4.compareDate(date1.getYear(),date1.getMonth(),date1.getDay()));
```

```
  assertTrue(not date1.compareDate(date4.getYear(),date4.getMonth(),date4.getDay()));
  assertTrue(not date1.compareDate(date6.getYear(),date6.getMonth(),date6.getDay()));
  assertTrue(not date6.compareDate(date1.getYear(),date1.getMonth(),date1.getDay()));
 );

--Test the conversion of a date to days
private testGDate: () ==> ()
 testGDate() == (
  assertEqual(737395,date1.gdate_algorithm());

  assertEqual(737423,date2.gdate_algorithm());
  assertEqual(737789,date3.gdate_algorithm());
  assertEqual(737455,date4.gdate_algorithm());
  assertEqual(737365,date5.gdate_algorithm());
  assertEqual(737395,date6.gdate_algorithm());
  assertEqual(737424,date7.gdate_algorithm());
  assertEqual(737790,date8.gdate_algorithm());
 );
-- Tests the number of days between two days
private testDaysElapsed: () ==> ()
 testDaysElapsed() == (
  assertEqual(29,Date`daysElapsed(date1,date2));
  assertEqual(1,Date`daysElapsed(date1,date6));
  assertEqual(31,Date`daysElapsed(date5,date1));
  assertEqual(367,Date`daysElapsed(date2,date3));

  assertEqual(91,Date`daysElapsed(date5,date4));
  assertEqual(33,Date`daysElapsed(date2,date4));
  assertEqual(2,Date`daysElapsed(date2,date7));
  assertEqual(2,Date`daysElapsed(date3,date8));

 );


--test Pre and Post conditions
  --Year must be >2018
private testInvYearDate: () ==> ()
 testInvYearDate() == (
  (dcl wrongYear: Date := new Date(2018,1,1);
  assertEqual(2018,wrongYear.getYear()));
 );


 --Month must be between 1 and 12
private testInvMonthDate: () ==> ()
 testInvMonthDate() == (
  (dcl wrongMonth: Date := new Date(2019,13,1);
  assertEqual(13,wrongMonth.getMonth()));

 );

--The day must respect the calender
private testInvDayDate: () ==> ()
 testInvDayDate() == (
  (dcl wrongDay: Date := new Date(2019,2,29);
  assertEqual(29,wrongDay.getDay()));

 );

private testPreConDitionDaysElapsed: () ==> ()
 testPreConDitionDaysElapsed() == (
  assertEqual(1,Date`daysElapsed(date2,date1));
 );
```

```
 public  test: () ==> ()
   test() ==
   (
    IO`println("\tDate Tests");
   testGets();
   testCompareDate();
   testDaysElapsed();
   testGDate();
    IO`println("\tTestes das datas terminados com sucesso!");

    --test Pre and Post conditions, leave the function calls commented

    --testInvYearDate();
    --testInvMonthDate();
    --testInvDayDate();
    --testPreConDitionDaysElapsed();
   );

end TestDate
```

4.3 TestRoom

```
class TestRoom is subclass of MyTestCase

instance variables
 private room1 : Room := new Room(10,100);
 private room2 : Room := new Room(10,100);
 private room3 : Room := new Room(10,100);


operations
 --Checks if Room Objects are being created correctly and returning the correct values

 private testGets: () ==> ()
  testGets() == (
   assertEqual(1,room1.getId());
   assertEqual(10,room1.getSpace());
   assertEqual(2,room2.getId());
   assertEqual(10,room2.getSpace());
   assertEqual(3,room3.getId());
   assertEqual(10,room3.getSpace());
   assertEqual(100,room1.getPricePerDay());
   assertEqual(100,room2.getPricePerDay());
   assertEqual(100,room3.getPricePerDay());
 );

 --Checks if a room exists

 private testRoomExists: () ==> ()
  testRoomExists() == (
   assertTrue(Room`roomExists(1));
   assertTrue(Room`roomExists(2));
   assertTrue(Room`roomExists(3));
   assertTrue(not Room`roomExists(4));
 );


 public  test: () ==> ()
   test() ==
```

```
    (
     IO`println("\tRoom Tests");
    testGets();
    testRoomExists();
     IO`println("\tTestes das salas terminados com sucesso!");

     --test Pre and Post conditions, leave the function calls commented
    );

end TestRoom
```

4.4 TestSchedule

```
class TestSchedule is subclass of MyTestCase

instance variables
 private room1 : Room := new Room(10,100);
 private room2 : Room := new Room(10,100);
 private room3 : Room := new Room(10,100);
 private date1 : Date := new Date(2019,1,1);
 private date2 : Date := new Date(2019,1,1);
 private date3 : Date := new Date(2019,1,2);
 private date4 : Date := new Date(2019,1,3);
 private schedule1 : Schedule := new Schedule(room1,date1,date2);
 private schedule2 : Schedule := new Schedule(room2,date1,date1);
 private schedule3 : Schedule := new Schedule(room3,date3,date3);

operations
 --Checks if Schedule Objects are being created correctly and returning the correct values

 private testGets: () ==> ()
  testGets() == (
     assertEqual(10,schedule1.getRoomSpace());
     assertEqual(room1.getId(),schedule1.getIdRoom());
     assertEqual(100,schedule1.getPricePerDay());
     assertEqual(date2,schedule1.getFinalDate());
     assertEqual(date1,schedule1.getInitialDate());
     assertEqual(1,schedule1.getNumberOfDays());

     assertEqual(10,schedule2.getRoomSpace());
     assertEqual(room2.getId(),schedule2.getIdRoom());
     assertEqual(100,schedule2.getPricePerDay());
     assertEqual(date1,schedule2.getFinalDate());
     assertEqual(date1,schedule2.getInitialDate());
     assertEqual(1,schedule1.getNumberOfDays());

     assertEqual(10,schedule3.getRoomSpace());
     assertEqual(room3.getId(),schedule3.getIdRoom());
     assertEqual(100,schedule3.getPricePerDay());
     assertEqual(date3,schedule3.getFinalDate());
     assertEqual(date3,schedule3.getInitialDate());
     assertEqual(1,schedule3.getNumberOfDays());
 );

 --tests if 2 dates overlaps a schedule/reservation of a room

 private testOverlaps: () ==> ()
  testOverlaps() == (
   assertTrue(not schedule1.overlaps(date3,date4,room1.getId()));
   assertTrue(schedule1.overlaps(date2,date3,room1.getId()));
```

```
    assertTrue(schedule1.overlaps(date1,date1,room1.getId()));
    assertTrue(not schedule1.overlaps(date1,date1,room2.getId()));
    assertTrue(not schedule2.overlaps(date3,date4,room2.getId()));
    assertTrue(schedule2.overlaps(date2,date3,room2.getId()));
    assertTrue(schedule2.overlaps(date1,date2,room2.getId()));
    assertTrue(not schedule2.overlaps(date2,date3,room1.getId()));
    assertTrue(schedule3.overlaps(date2,date3,room3.getId()));
    assertTrue(not schedule3.overlaps(date1,date2,room3.getId()));
    assertTrue(schedule3.overlaps(date3,date4,room3.getId()));
    assertTrue(not schedule3.overlaps(date4,date4,room3.getId()));

 );

 --test Pre and Post conditions
 --Final date need to be bigger or equal to initial Date

 private testFinalDateIsSmaller: () ==> ()
  testFinalDateIsSmaller() == (
    (dcl finalDateIsSmaller : Schedule :=new Schedule(room3,date4,date3);
    assertEqual(100,finalDateIsSmaller.getPricePerDay()));
  );

 --Final date need to be bigger or equal to initial Date

 private testFinalDateIsSmallerOverlaps: () ==> ()
  testFinalDateIsSmallerOverlaps() == (
    assertTrue(schedule1.overlaps(date4,date3,room1.getId()));
 );

 --A room needs to exist

 private testRoomExistsOverlaps: () ==> ()
  testRoomExistsOverlaps() == (
    assertTrue(schedule1.overlaps(date3,date3,1243243));
 );


 public  test: () ==> ()
    test() ==
    (
     IO`println("\tSchedule Tests");
    testGets();
    testOverlaps();
     IO`println("\tTestes das reservas terminados com sucesso!");

     --test Pre and Post conditions, leave the function calls commented
    --testFinalDateIsSmaller()
    --testFinalDateIsSmallerOverlaps();
    --testRoomExistsOverlaps();
    );

end TestSchedule
```

4.5 TestEntity

```
class TestEntity is subclass of MyTestCase

instance variables
 private visitor1 : Visitor := new Visitor(" J o o ");
 private visitor2 : Visitor := new Visitor(" J o s ");
```

```
 private visitor3 : Visitor := new Visitor("Maria");
 private exhibitor1 : Exhibitor := new Exhibitor("Paulo");
 private exhibitor2 : Exhibitor := new Exhibitor("Ana");
 private exhibitor3 : Exhibitor := new Exhibitor("Sofia");

operations
 --Checks if Entity Objects are being created correctly and returning the correct values

 private testGets: () ==> ()
  testGets() == (
   assertEqual(1,visitor1.getId());
   assertEqual(2,visitor2.getId());
   assertEqual(3,visitor3.getId());
   assertEqual(1,exhibitor1.getId());
   assertEqual(2,exhibitor2.getId());
   assertEqual(3,exhibitor3.getId());
   assertEqual(" J o o ",visitor1.getName());
   assertEqual(" J o s ",visitor2.getName());
   assertEqual("Maria",visitor3.getName());
   assertEqual("Paulo",exhibitor1.getName());
   assertEqual("Ana",exhibitor2.getName());
   assertEqual("Sofia",exhibitor3.getName());
  );

 -- A name can't have more than 50 characters

 private testNameTooBig: () ==> ()
  testNameTooBig() == (
  (dcl NameTooBig : Exhibitor :=new Exhibitor("O meu nome tem mais de cinquenta caracteres por
       isso  d   erro");
   assertEqual("O meu nome tem mais de cinquenta caracteres por isso  d   erro",NameTooBig.
       getName()););
  );


 public test: () ==> ()
  test() == (
   IO`println("\tEntity Tests");
   testGets();
   IO`println("\tTestes das entidades terminados com sucesso!");

   --test Pre and Post conditions, leave the function calls commented
   --testNameTooBig();

  );

end TestEntity
```

4.6 TestEvent

```
class TestEvent is subclass of MyTestCase

instance variables
 private room1 : Room := new Room(10,100);
 private room2 : Room := new Room(10,100);
 private room3 : Room := new Room(6,100);
 private date1 : Date := new Date(2019,1,1);
 private date2 : Date := new Date(2019,1,4);
 private date3 : Date := new Date(2019,1,2);
 private date4 : Date := new Date(2019,1,3);
```

```
 private schedule1 : Schedule := new Schedule(room1,date1,date2);
 private schedule2 : Schedule := new Schedule(room2,date1,date1);
 private schedule3 : Schedule := new Schedule(room3,date3,date3);
 private visitor1 : Visitor := new Visitor(" J o o ");
 private visitor2 : Visitor := new Visitor(" J o s ");
 private visitor3 : Visitor := new Visitor("Maria");
 private exhibitor1 : Exhibitor := new Exhibitor("Paulo");
 private exhibitor2 : Exhibitor := new Exhibitor("Ana");
 private exhibitor3 : Exhibitor := new Exhibitor("Sofia");
 private visitor4 : Visitor := new Visitor(" J o o ");
 private visitor5 : Visitor := new Visitor(" J o s ");
 private visitor6 : Visitor := new Visitor("Maria");
 private visitor7 : Visitor := new Visitor("Maria");
 private event1 : Event := new Event(schedule1, exhibitor1, 20,"evento1");
 private event2 : Event := new Event(schedule2, exhibitor2, 20,"evento2");
 private event3 : Event := new Event(schedule3, exhibitor3, 20,"evento3");
 private event4 : Event := new Event(schedule3, exhibitor3, 20,"evento4");

operations

 --Tests the closure and the force close of an event

 private testCloseandForceCloseEvent: () ==> ()
  testCloseandForceCloseEvent() == (
   event1.closeEvent(new Date(CurrentTime`getYear(), CurrentTime`getMonth(), CurrentTime`
       getDay()));
   event2.closeEvent(new Date(CurrentTime`getYear(), CurrentTime`getMonth(), CurrentTime`
       getDay()));
   event3.forceCloseEvent("password");

   assertTrue(event1.isClosed());
   assertTrue(event2.isClosed());
   assertTrue(event3.isClosed());
  );

 --Checks if Event Objects are being created correctly and returning the correct values

 private testGets: () ==> ()
  testGets() == (
   assertEqual(20,event1.getPricePerParticipant());
   assertEqual(true,event1.isClosed());
   assertEqual(400,event1.getCostOfEvent());
   assertEqual(0, event1.getSalesValueOfEvent());
   assertEqual(-400, event1.profit());
   assertEqual(exhibitor1,event1.getExhibitor());
   assertEqual({}, event1.getParticipants());
   assertTrue(event1.has_started());
   assertTrue(event1.has_ended());
   assertEqual(1, event1.getId());
   assertEqual("evento1", event1.getName());

   assertEqual(20,event2.getPricePerParticipant());
   assertEqual(true,event2.isClosed());
   assertEqual(100,event2.getCostOfEvent());
   assertEqual(0, event2.getSalesValueOfEvent());
   assertEqual(-100, event2.profit());
   assertEqual(exhibitor2,event2.getExhibitor());
   assertEqual({}, event2.getParticipants());
   assertTrue(event2.has_started());
   assertTrue(event2.has_ended());
   assertEqual(2, event2.getId());
   assertEqual("evento2", event2.getName());

   assertEqual(20,event3.getPricePerParticipant());
   assertEqual(true,event3.isClosed());
```

```
  assertEqual(100,event3.getCostOfEvent());
  assertEqual(0, event3.getSalesValueOfEvent());
  assertEqual(-100, event3.profit());
  assertEqual(exhibitor3,event3.getExhibitor());
  assertEqual({}, event3.getParticipants());
  assertTrue(event3.has_started());
  assertTrue(event3.has_ended());
  assertEqual(3, event3.getId());
  assertEqual("evento3", event3.getName());

  assertEqual(20,event4.getPricePerParticipant());
  assertEqual(false,event4.isClosed());
  assertEqual(100,event4.getCostOfEvent());
  assertEqual(0, event4.getSalesValueOfEvent());
  assertEqual(-100, event4.profit());
  assertEqual(exhibitor3,event4.getExhibitor());
  assertEqual({}, event4.getParticipants());
  assertTrue(event4.has_started());
  assertTrue(event4.has_ended());
  assertEqual(4, event4.getId());
  assertEqual("evento4", event4.getName());
 );

--tests if 2 dates overlaps the event

private testOverlaps: () ==> ()
 testOverlaps() == (
  assertTrue(event1.overlaps(date3,date4,room1.getId()));
  assertTrue(event1.overlaps(date1,date3,room1.getId()));
  assertTrue(event1.overlaps(date1,date1,room1.getId()));
  assertTrue(not event1.overlaps(date1,date1,room2.getId()));
  assertTrue(not event2.overlaps(date3,date4,room2.getId()));
  assertTrue(event2.overlaps(date1,date3,room2.getId()));
  assertTrue(event2.overlaps(date1,date1,room2.getId()));
  assertTrue(not event2.overlaps(date1,date3,room1.getId()));
  assertTrue(event3.overlaps(date1,date3,room3.getId()));
  assertTrue(not event3.overlaps(date1,date1,room3.getId()));
  assertTrue(event3.overlaps(date3,date4,room3.getId()));
  assertTrue(not event3.overlaps(date4,date4,room3.getId()));
);

--tests the addition and removal of participants

private testParticipants: () ==> ()
 testParticipants () == (
  event4.addParticipant(visitor1);
  event4.addParticipant(visitor2);
  assertEqual({visitor1,visitor2}, event4.getParticipants());
  assertEqual(2, card event4.getParticipants());
  assertEqual(40, event4.getSalesValueOfEvent());
  assertEqual(-60, event4.profit());

  event4.removeParticipant(visitor1);
  event4.removeParticipant(visitor2);
  assertEqual(0, event4.getSalesValueOfEvent());
  assertEqual(-100, event4.profit());

  event4.addParticipant(visitor1);
  event4.addParticipant(visitor2);
  event4.addParticipant(visitor3);
  event4.addParticipant(visitor4);
  event4.addParticipant(visitor5);
  event4.addParticipant(visitor6);

  assertEqual(6, card event4.getParticipants());
```

```
  assertEqual(120, event4.getSalesValueOfEvent());
  assertEqual(20, event4.profit());

  event4.removeParticipant(visitor1);
  event4.removeParticipant(visitor2);

  assertTrue(event4.has_participant(visitor3));
  assertTrue(not event4.has_participant(visitor1));

  assertEqual(4, card event4.getParticipants());
  assertEqual(80, event4.getSalesValueOfEvent());
  assertEqual(-20, event4.profit());

 );

 --test Pre and Post conditions
 -- A name can't have more than 50 characters

private testInvName: () ==> ()
 testInvName () == (
  (dcl NameTooBig : Event :=new Event(schedule1, exhibitor1, 20,"O meu nome tem mais de
      cinquenta caracteres por isso  d   erro");
   assertEqual("O meu nome tem mais de cinquenta caracteres por isso  d   erro",NameTooBig.
       getName());
  );
 );

-- An event can't have more participants than those that the room allows

private testLimitOfParticipants: () ==> ()
 testLimitOfParticipants () == (
  event4.addParticipant(visitor1);
  event4.addParticipant(visitor2);
  assertEqual(6, card event4.getParticipants());
  --fails
  event4.addParticipant(visitor7);
  assertEqual(7, card event4.getParticipants());
 );

--The Final date of the Event needs to be lower than the current date for an event to be
    closed

private testCloseEventPre: () ==> ()
 testCloseEventPre () == (
  (dcl currentDate : Date := new Date(CurrentTime`getYear(),CurrentTime`getMonth(),
      CurrentTime`getDay()),
    schedule : Schedule := new Schedule(room1, currentDate,currentDate),
   event : Event := new Event(schedule,exhibitor1,100,"event");
   event.closeEvent(currentDate);
   assertTrue(event.isClosed());
  );
 );

--Wrong Password

private testForceCloseEventWrongPassword: () ==> ()
 testForceCloseEventWrongPassword () == (
  event4.forceCloseEvent("testpassword");
 );

--IT's not possible to add a partipant to a closed event

private testaddParticipantToClosedEvent: () ==> ()
 testaddParticipantToClosedEvent () == (
```

```
   event4.closeEvent(new Date(CurrentTime`getYear(), CurrentTime`getMonth(), CurrentTime`
       getDay()));
   event4.addParticipant(visitor1);
  );

 --IT's not possible to remove a partipant from a closed event

 private testRemoveParticipantFromClosedEvent: () ==> ()
  testRemoveParticipantFromClosedEvent () == (
   event4.removeParticipant(visitor1);
  );



 public test: () ==> ()
  test() == (
   IO`println("\tEvent Tests");
    testCloseandForceCloseEvent();
    testGets();
    testParticipants();
    testOverlaps();
   IO`println("\tTestes a eventos terminados com sucesso!");

   --test Pre and Post conditions, leave the function calls commented~
   --testInvName();
   --testLimitOfParticipants();
   --testCloseEventPre();
   --testForceCloseEventWrongPassword();
   --testaddParticipantToClosedEvent();
   --testRemoveParticipantFromClosedEvent();
  );

end TestEvent
```

4.7 TestExhibitionCentre

Tendo em conta que o ExhibicionCentre é responsável pela organização de todo o nosso programa. Todos os requisitos são testados nesta classe.

```
class TestExhibitionCentre is subclass of MyTestCase

instance variables
 private centre : ExhibitionCentre := ExhibitionCentre`getInstance();
 private visitor1: Visitor := centre.registerVisitor("visitor1");
 private visitor2: Visitor := centre.registerVisitor("visitor2");
 private visitor3: Visitor := centre.registerVisitor("visitor3");
 private visitor4: Visitor := centre.registerVisitor("visitor4");
 private visitor5: Visitor := centre.registerVisitor("visitor5");
 private exhibitor1 : Exhibitor := centre.registerExhibitor("exhibitor1");
 private exhibitor3 : Exhibitor := centre.registerExhibitor("exhibitor2");
 private exhibitor2 : Exhibitor := centre.registerExhibitor("exhibitor3");
 private exhibitor4 : Exhibitor := centre.registerExhibitor("exhibitor4");
 private exhibitor5 : Exhibitor := centre.registerExhibitor("exhibitor5");
 private room1: Room := centre.addRoom(30,3);
 private room2: Room := centre.addRoom(20,3);
 private room3: Room := centre.addRoom(10,3);
 private date1: Date := new Date (2019,1,1);
 private date2: Date := new Date (2019,1,2);
 private date3: Date := new Date (2019,1,4);
 private date4: Date := new Date (3000,1,1);
```

```
 private date5: Date := new Date (3000,2,28);
 private event1 : Event;
 private event2 : Event;
 private event3 : Event;
 private event4 : Event;
 private event5 : Event;
 private event6 : Event;



operations

 --Tests the Entity flow for Exhibition Centre

 private testEntities: () ==> ()
  testEntities() == (
   assertEqual({visitor1,visitor2,visitor3,visitor4,visitor5},centre.getVisitors());
   assertEqual({exhibitor1,exhibitor2,exhibitor3,exhibitor4,exhibitor5},centre.getExhibitors()
       );

   assertEqual(visitor1, centre.getVisitor(visitor1.getId()));
   assertEqual(visitor2, centre.getVisitor(visitor2.getId()));
   assertEqual(exhibitor1, centre.getExhibitor(exhibitor1.getId()));
   assertEqual(exhibitor3, centre.getExhibitor(exhibitor3.getId()));

   centre.removeVisitor(visitor4);
   assertEqual({event2,event4},{event | event in set centre.getEvents() & event.
       has_participant(visitor4)});
   centre.removeVisitor(visitor2);
   assertEqual({event1},{event | event in set centre.getEvents() & event.has_participant(
       visitor2)});

   assertEqual({event1,event5}, centre.getEventsFromExhibitor(exhibitor1));
   centre.removeExhibitor(exhibitor1);
   assertEqual({event1,event5}, {event | event in set centre.getEvents() & event.getExhibitor
       () = exhibitor1});

   assertEqual(4, card centre.getEventsFromExhibitor(exhibitor2));
   centre.removeExhibitor(exhibitor2);
   assertEqual(1, card {event | event in set centre.getEvents() & event.getExhibitor() =
       exhibitor2});

   assertEqual({event4}, centre.getEventsFromExhibitor(exhibitor4));
   centre.removeExhibitor(exhibitor4);
   assertEqual({event4}, {event | event in set centre.getEvents() & event.getExhibitor() =
       exhibitor4});
   return;
  );

 --Tests the Room flow for Exhibition Centre

 private testRooms: () ==> ()
  testRooms() == (
   assertEqual(3, card centre.getRooms());

   for all room in set centre.getRooms()
    do assertEqual(room, centre.getRoom(room.getId()));

   for all room in set centre.getRooms()
    do centre.removeRoom(room);

   assertEqual(0, card centre.getRooms());

   room1 := centre.addRoom(30,3);
```

```
  room2 := centre.addRoom(20,3);
  room3 := centre.addRoom(10,3);

  assertEqual(3, card centre.getRooms());
 );

--Tests the Event flow for Exhibition Centre

private testEvents: () ==> ()
 testEvents() == (
  assertEqual(3, card centre.getRooms());
  event1 := centre.createEvent(date1,date2,room1,exhibitor1,20,"event1");
  event2 := centre.createEvent(date1,date2,room2,exhibitor2,20,"event2");
  event3 := centre.createEvent(date1,date2,room3,exhibitor3,20,"event3");
  event4 := centre.createEvent(date3,date4,room1,exhibitor4,20,"event4");
  event5 := centre.createEvent(date4,date5,room2,exhibitor1,20,"event5");
  event6 := centre.createEvent(date5,date5,room3,exhibitor2,20,"event6");


  assertEqual(6, card centre.getEvents());

  for all event in set centre.getEvents()
   do assertEqual(event, centre.getEvent(event.getId()));

  centre.addParticipantToEvent(event1,visitor1);
  centre.addParticipantsToEvent(event1, {visitor2,visitor3});

  assertEqual(3, card centre.getEvent(event1.getId()).getParticipants());
  assertEqual([20,60,60,0,3],centre.getEventStatistics(event1));

  centre.addParticipantToEvent(event2,visitor4);
  centre.addParticipantToEvent(event2,visitor5);

  assertEqual(2, card centre.getEvent(event2.getId()).getParticipants());
  assertEqual([20,40,40,0,2],centre.getEventStatistics(event2));

  centre.addParticipantToEvent(event3,visitor1);
  centre.addParticipantsToEvent(event3, {visitor2,visitor3});

  assertEqual(3, card centre.getEvent(event3.getId()).getParticipants());
  assertEqual([20,20,60,40,3],centre.getEventStatistics(event3));

  centre.addParticipantToEvent(event4,visitor4);
  centre.addParticipantToEvent(event4,visitor5);

  assertEqual(2, card centre.getEvent(event4.getId()).getParticipants());
  assertEqual([20, 10749030, 40, -10748990, 2],centre.getEventStatistics(event4));

  centre.addParticipantToEvent(event5,visitor1);
  centre.addParticipantsToEvent(event5, {visitor2,visitor3});

  assertEqual(3, card centre.getEvent(event5.getId()).getParticipants());
  assertEqual([20, 1180, 60, -1120, 3],centre.getEventStatistics(event5));

  centre.addParticipantToEvent(event6,visitor4);
  centre.addParticipantToEvent(event6,visitor5);

  assertEqual(2, card centre.getEvent(event6.getId()).getParticipants());
  assertEqual([20, 10, 40, 30, 2],centre.getEventStatistics(event6));

  assertEqual({event1,event3,event5},centre.getEventsFromVisitor(visitor1)
                   union centre.getEventsFromVisitor(visitor2)
                   union centre.getEventsFromVisitor(visitor3));
  assertEqual({event2,event4,event6},centre.getEventsFromVisitor(visitor4)
                   union centre.getEventsFromVisitor(visitor5));
```

```
assertEqual({event1,event5}, centre.getEventsFromExhibitor(exhibitor1));
assertEqual({event2,event6}, centre.getEventsFromExhibitor(exhibitor2));
assertEqual({event3}, centre.getEventsFromExhibitor(exhibitor3));
assertEqual({}, centre.getEventsFromExhibitor(exhibitor5));
assertEqual(60,centre.totalMoneySpent(visitor1));
assertEqual(60,centre.totalMoneySpent(visitor2));
assertEqual(60,centre.totalMoneySpent(visitor3));
assertEqual(60,centre.totalMoneySpent(visitor4));
assertEqual(60,centre.totalMoneySpent(visitor5));
assertEqual(-1120,centre.totalProfit(exhibitor1));
assertEqual(30,centre.totalProfit(exhibitor2));
assertEqual(40,centre.totalProfit(exhibitor3));
assertEqual(-10748990,centre.totalProfit(exhibitor4));
assertEqual(0,centre.totalProfit(exhibitor5));


centre.removeParticipantFromEvent(event1,visitor1);
assertEqual(2, card centre.getEvent(event1.getId()).getParticipants());
centre.removeParticipantsFromEvent(event3,{visitor1,visitor2});
assertEqual(1, card centre.getEvent(event3.getId()).getParticipants());
centre.removeParticipantsFromEvent(event5,{visitor1});
assertEqual(2, card centre.getEvent(event5.getId()).getParticipants());

assertEqual({},centre.getEventsFromVisitor(visitor1));
assertEqual({event1,event5},centre.getEventsFromVisitor(visitor2));
assertEqual(0,centre.totalMoneySpent(visitor1));
assertEqual(40,centre.totalMoneySpent(visitor2));

assertEqual({},centre.getAvailableRooms(date1,date2));
assertEqual({},centre.getAvailableRooms(date1,date3));
assertEqual({},centre.getAvailableRooms(date1,date1));
assertEqual({room3},centre.getAvailableRooms(date3,date4));
assertEqual({},centre.getAvailableRooms(date4,date5));
assertEqual({room1},centre.getAvailableRooms(date5,date5));

assertTrue(centre.overlapsEvents(date1,date2,room1.getId()));
assertTrue(centre.overlapsEvents(date1,date2,room2.getId()));
assertTrue(centre.overlapsEvents(date1,date2,room3.getId()));
assertTrue(centre.overlapsEvents(date3,date3,room1.getId()));
assertTrue(centre.overlapsEvents(date3,date4,room2.getId()));
assertTrue( not centre.overlapsEvents(date3,date4,room3.getId()));
assertTrue(centre.overlapsEvents(date4,date5,room1.getId()));
assertTrue(centre.overlapsEvents(date4,date5,room2.getId()));
assertTrue(centre.overlapsEvents(date4,date5,room3.getId()));
assertTrue(not centre.overlapsEvents(date5,date5,room1.getId()));
assertTrue(centre.overlapsEvents(date5,date5,room2.getId()));
assertTrue(centre.overlapsEvents(date5,date5,room3.getId()));

start(ExhibitionCentre`getInstance());
while centre.getNotClosedEvents()<>{event4,event5,event6}
 do IO`println("\t\tTesting thread");

assertEqual({event4,event5,event6},centre.getNotClosedEvents());
centre.forceCloseEvent(event5,"password");
assertEqual({event4,event6},centre.getNotClosedEvents());
assertEqual({room2},centre.getAvailableRooms(date4,date5));
assertTrue(centre.overlapsEvents(date4,date5,room1.getId()));
assertTrue(not centre.overlapsEvents(date4,date5,room2.getId()));
assertTrue(centre.overlapsEvents(date4,date5,room3.getId()));

assertEqual({room1,room2,room3},centre.getAvailableRooms(date1,date2));
assertTrue(not centre.overlapsEvents(date1,date2,room1.getId()));
assertTrue(not centre.overlapsEvents(date1,date2,room2.getId()));
assertTrue(not centre.overlapsEvents(date1,date2,room3.getId()));
```

```
  centre.removeEvent(event6);
  assertEqual(5, card centre.getEvents());
  assertEqual({event4},centre.getNotClosedEvents());
  assertEqual({room1,room2,room3},centre.getAvailableRooms(date5,date5));

  event6 := centre.createEvent(date5,date5,room3,exhibitor2,20,"event6");
  centre.addParticipantToEvent(event6,visitor4);
  centre.addParticipantToEvent(event6,visitor5);
  event6 := centre.createEvent(date5,date5,room1,exhibitor2,20,"event6");
  centre.addParticipantToEvent(event6,visitor4);
  centre.addParticipantToEvent(event6,visitor5);
  event6 := centre.createEvent(date5,date5,room2,exhibitor2,20,"event6");
  centre.addParticipantToEvent(event6,visitor4);
  centre.addParticipantToEvent(event6,visitor5);
 );

--The Event needs to be on the set

private testpreGetEvent : () ==> ()
 testpreGetEvent () == (
  event2 := centre.getEvent(9999999999999999);
 );

--The Room needs to be on the set

private testpreGetRoom : () ==> ()
 testpreGetRoom () == (
  room3 := centre.getRoom(9999999999999999);
 );

--The visitor need to be on the set

private testpreGetVisitor : () ==> ()
 testpreGetVisitor () == (
  visitor5 := centre.getVisitor(9999999999999999);
 );

--The exhibitor needs to be on the set

private testpreGetExhibitor : () ==> ()
 testpreGetExhibitor () == (
  exhibitor5 := centre.getExhibitor(9999999999999999);
 );

--It's not possible to remove closed events

private testpreRemoveEvent : () ==> ()
 testpreRemoveEvent () == (
  centre.removeEvent(event1);
 );

--Final date need to be bigger or equal than the initial date

private testpreGetAvailableRooms : () ==> ()
 testpreGetAvailableRooms () == (
  assertTrue(room1 in set centre.getAvailableRooms(new Date(2019,1,2),new Date(2019,1,1)));
 );

--The room needs to exist on the set

private testpreOverlapsEvents : () ==> ()
 testpreOverlapsEvents () == (
  assertTrue(not centre.overlapsEvents(new Date(2019,1,2),new Date(2019,1,2),999999999999));
 );
```

```
 --It's not possible to have 2 or more events that overlap eachother

 private testpreCreateEvent : () ==> ()
  testpreCreateEvent () == (
   event6 := centre.createEvent(date5,date5,room3,exhibitor5,20,"event6");
   event6 := centre.createEvent(date5,date5,room3,exhibitor5,20,"event6");
  );


 public test: () ==> ()
  test() == (
   IO`println("\tExhibition Centre Tests");
    testRooms();
    testEvents();
    testEntities();
   IO`println("\tTestes do Centro de  E x p o s i es  terminados com sucesso!");

   --test Pre and Post conditions, leave the function calls commented
   --testpreGetEvent();
   --testpreGetRoom();
   --testpreGetVisitor();
   --testpreGetExhibitor();
   --testpreRemoveEvent();
   --testpreCreateEvent();
   --testpreGetAvailableRooms();
   --testpreOverlapsEvents();
  );

end TestExhibitionCentre
```

4.8 TestMain

```
/*

  Class with the main function to run all test classes, simpler but more practical than
      VDMUnit`TestCase.
 To Add a Test: Create a new object of the test class and run tests
   R ben  Torres &  J o s  Azevedo, FEUP, MFES, 2014/15.

*/

class TestMain
operations


 public static main: () ==> ()
  main() ==
  (
   IO`println("Inicializar testes...");

 --to add a test use:  new TestClass.test()
   new TestDate().test();
   new TestRoom().test();
   new TestEntity().test();
  new TestSchedule().test();
  new TestEvent().test();
  new TestExhibitionCentre().test();

   IO`println("Testes terminados com sucesso!");
  );
```

```
end TestMain
```

5 Verificação do Modelo

Todas as *Proof Obligations* apresentadas foram geradas com o overture, gerando este 63 *Proof Obligations*, sendo estas apenas dos seguintes tipos:

- *state invariant holds*
- *type compatibility*
- *legal function application*
- *cases exhaustive*
- *state invariant initialized*
- *state invariant satisfiable*
- *operation establishes postcondition*
- *operation call*
- *legal function application*

5.1 Example of type compatibility verification

No.	Nome da PO	Tipo	Proof Obligation
2	Date‘gdate_algorithm(),m	type compatibility	(((month + 9) mod 12) >= 0)
3	Date‘gdate_algorithm(),y	type compatibility	((year - (floor (m / 10))) >= 0)

O código sobre análise é o seguinte:

```
--converts the gregorian date from yy/mm/dd to days

pure public gdate_algorithm : () ==>nat1
 gdate_algorithm() == (
 (dcl m: nat := (month + 9) mod 12,
  y: nat := year - floor(m/10);

  return 365*y + floor(y/4) - floor(y/100) + floor(y/400) + floor((m*306 + 5)/10) + ( day - 1
       ))
 );
```

Neste caso as duas provas são triviais, a primeira prova é trivial, pois o módulo de um qualquer número por outro número que seja maior ou igual a zero (neste caso 12) é sempre positivo, por consequência, a expressão ((month + 9) mod 12)) é impossível ter um resultado negativo, para qualquer que seja o valor de month. Em seguimento do resultado da primeira prova, podemos igualmente concluir que a segunda é trivial, sabendo que a invariante da variável year a impede de ser <2019, e o valor de m tem que estar compreendido entre 0 e 12, podemos então concluir que a expressão (year - floor(m/10)) é impossível resultar num valor negativo, pois tendo em conta que year será sempre maior que o floor(m/10), a subtração destes dois valores nunca será negativa.

5.2 Example of invariant verification

No.	Nome da PO	Tipo
18	Event‘addParticipant(Visitor)	state invariant holds

O código sobre análise é o seguinte:

```
--enrolls a given Visitor/Participant to the event

public addParticipant : Visitor ==> ()
 addParticipant(participant) == (
  participants := participants union {participant};
 )
 pre participant not in set participants and not closed
 post participant in set participants;
```

```
inv card participants <= schedule.getRoomSpace()
   and not exists p1, p2 in set participants & p1 <> p2 and p1.getId() = p2.getId() ;
```

Quando um utilizador compra um bilhete para um evento é verificado se esse evento já não atingiu o número máximo de pessoas a participar no evento, havendo assim uma verificação da invariante, sendo a restante expressão (not exists p1, p2 in set participants p1 < > p2 and p1.getId() = p2.getId()) impossível ser falsa, pois, por *design* os *id's* de participantes são gerados automaticamente e nunca são gerados dois *id's* repetidos.

6 Geração de Código

Para ser possível gerar código java, primeiramente é necessário substituir todas as expressões iota (‘iota’, bind, ‘’, expression) por expressões for all (for all pat in set setexpr do stmt), pois estas não são suportadas pelo gerador de código.
Após o código ser gerado é necessário comentar ou substituir todo o código relacionado com a iniciação de uma *thread*, isto acontece, pois o gerador de código java não gera classes do tipo runnable, como consequência todo o nosso programa passa ser *single-threaded*.
O passo seguinte é recriar a nossa livraria externa CurrentTime para uso em código java, um processo relativamente simples, mas que demonstra mais uma limitação do gerador de código.
De seguida adicionamos o parâmetro (String args[]) à função main da Classe TestMain.
Apesar de todas estas limitações, foi possível executar os nossos testes e constatar que nenhum deles falha. Por fim, foram feitas adições de pontuais correções e modificações de instruções de impressão para a linha de comandos para ajudar na construção da *interface*. Sendo este o resultado:

```
   --WELCOME TO FIL- Center of Exhibitions of Lisbon--

 1 - Ver Utilizadores Registados
 2 - Ver Organizadores Registados
 3 - Ver todos os Eventos
 4 - Ver salas disponíveis para reserva
 5 - Procurar um Evento
 6 - Procurar uma Sala
 7 - Procurar um Utilizador
 8 - Procurar um Organizador
 9 - Registar um Utilizador
10 - Registar um Organizador
11 - Remover um Utilizador
12 - Remover um Organizador
13 - Criar um Evento
14 - Remover um Evento
15 - Fechar Eventos Concluídos
16 - Forçar um Fecho de um evento (requer uma password especial, PASSWORD especial)
17 - Adicionar Participante a um evento
18 - Remover Participante de um evento
19 - Adicionar uma sala
20 - Remover uma sala
21 - Ver salas disponiveis para reserva entre duas datas
22 - Posso Reservar uma determinada sala entre duas datas
23 - Ver todos os eventos em que um utilizador é participante
24 - Ver todos os eventos de um organizador
25 - Ver estatísticas de um Evento
26 - Ver total de dinheiro gasto por um utilizador
27 - Ver total de lucro de um Organizador
28 - Ver todos os eventos não concluídos
0 - Fechar Programa

Escolha uma opção:
```

Figura 4: Interface

7 Conclusão

Concluindo o projeto, o grupo achou o resultado do produto final bastante satisfatório, cumprindo todos os objetivos estabelecidos.

As principais dificuldades encontradas no desenvolvimento foram a implementação de *threads*, usadas para fechar os eventos, devido à possibilidade de apenas serem implementadas *threads* procedimentais (o ideal seria usar *threads* periódicas). No entanto, o processo utilizado envolveu a criação de um *Thread.sleep* como biblioteca externa entre a chamada à função de verificação de fecho de eventos, sendo que esta resolução não tenha ficado perfeita, pois, o *sleep* era feito em todos as *threads*. Decidindo o grupo no final por retirar esta funcionalidade (*Thread.sleep*). Para além disso, a criação de uma biblioteca externa também criou bastante dificuldades bem como a falta de funções de sistema. Relativamente à dificuldade, o grupo a achou a sintaxe da linguagem acessível, sentindo por vezes alguma dificuldade em situações mais complexas devido à falta de informação *online* sobre a mesma relativamente a outras linguagens.

Quanto à divisão de tarefas, esta foi aproximadamente 60% por parte do membro Rúben Torres, sendo que este desenvolveu a maior parte do código, enquanto que o membro José Azevedo (40%) compensou no relatório, embora ambos tenham participado em ambas as partes.

8 Referências

8.1 Bibliografia

- *VDM-10 Language Manual* by Peter Gorm Larsen, Kenneth Laudahl, Nick Battle, John Fitzgerald, Sune Wolff, Shin Sahara & Marcel Verhoef;
- *Overture VDM-10 Tool Support: User Guide Version 2.6.0* by Peter Gorm Larsen, Kenneth Lausdahl, Peter Tran-Jørgensen, Joey Coleman, Sune Wolff, Luıs Diogo Couto and Victor Bandur Aarhus University, Department of Engineering Finlandsgade 22, DK-8000 Arhus C, Denmark;
- *Tutorial for Overture/VDM++* by Peter Gorm Larsen, John Fitzgerald, Sune Wolff, Nick Battle, Kenneth Lausdahl, Augusto Ribeiro, Kenneth Pierce, Victor Bandur;

8.2 Software

- Eclipse: `http://www.eclipse.org/`
- Overture: `http://overturetool.org/`
- Modelio: `https://www.modelio.org/`