FashionShow

January 3, 2018

Contents

1 ClothDisplayed

```
 class ClothDisplayed

/*

 Defines a cloth displayed by a model on a certain runway show.
  J. Oliveira, FEUP, MFES, 2017/18.

*/

instance variables
 public model : Person;
 public cloth : PieceOfCloth;
 public runway : RunwayShow;

 inv model.clothSize = cloth.size;

operations
-- constructor

 public ClothDisplayed: Person * PieceOfCloth * RunwayShow ==> ClothDisplayed
  ClothDisplayed(p, c, r) == (
   model := p;
   cloth := c;
   runway := r;
   return self;
  )
  pre c.creator in set elems r.organizers and
    c.size = p.clothSize;

-- set model

 public setModel : Person ==> ()
  setModel(p) == (
   model := p;
  )
  pre cloth.size = p.clothSize;

-- set cloth

 public setCloth : PieceOfCloth ==> ()
  setCloth(c) == (
   cloth := c;
  )
  pre c.creator in set elems runway.organizers and
    c.size = model.clothSize;

-- set runway show

 public setRunwayShow : RunwayShow ==> ()
  setRunwayShow(r) == (
   runway := r;
  )
  pre cloth.creator in set elems r.organizers;

end ClothDisplayed
```

Function or operation	Line	Coverage	Calls
ClothDisplayed	19	100.0%	2
setCloth	37	100.0%	1
setModel	30	100.0%	1
setRunwayShow	45	100.0%	1
ClothDisplayed.vdmpp		100.0%	5

2 Event

```
class Event

/*

 Defines a event of a fashion show, that will have 3 sub-types, Presentation, PrimpingSession and
      RunwayShow.
  J. Oliveira, FEUP, MFES, 2017/18.

*/


instance variables
 public show : FashionShow;
 public organizers : seq1 of (Person);
  public place : Room;
  public name : Utils`string;
  public startDate : Utils`date;
  public endDate : Utils`date;

  -- invariants
 inv Utils`isOldestDate(startDate, endDate);

operations

-- set organizers

  public setOrganizers : seq1 of (Person) ==> ()
   setOrganizers(o) ==
    organizers := o;

-- add organizer

  public addOrganizer : Person ==> ()
   addOrganizer(p) ==
    organizers := organizers ^ [p];

 -- set place

  public setPlace : Room ==> ()
   setPlace (r) ==
    place := r;
```

```
-- set name

  public setName : Utils`string ==> ()
   setName(n) ==
    name := n;

-- set start date

 public setStartDate : Utils`date ==> ()
  setStartDate(d) ==
   startDate := d;

-- set end date

 public setEndDate : Utils`date ==> ()
  setEndDate(d) ==
   endDate := d;

-- end event and empty the room

 public endEvent : () ==> ()
  endEvent() ==
   place.emptyTheRoom();

end Event
```

Function or operation	Line	Coverage	Calls
addOrganizer	30	100.0%	3
endEvent	55	100.0%	3
setEndDate	50	100.0%	3
setName	40	100.0%	3
setOrganizers	25	100.0%	3
setPlace	35	100.0%	3
setStartDate	45	100.0%	3
Event.vdmpp		100.0%	21

3 FashionShow

```
class FashionShow

/*

 Defines a fashion show, it's the main class of the project.
  J. Oliveira, FEUP, MFES, 2017/18.

*/


instance variables
 public name : Utils`string;
  public place : Utils`string;
  public startDate : Utils`date;
  public endDate : Utils`date;
  public events : seq of (Event);
  public peopleAttending : set of (Person);

-- invariants
inv Utils`isOldestDate(startDate, endDate);
inv not exists e1, e2 in set elems events & (e1 <> e2 and e1.place = e2.place and Utils`coincDate
    (e1.startDate,e1.endDate, e2.startDate, e2.endDate));

operations
-- constructor

 public FashionShow : Utils`string * Utils`string * Utils`date * Utils`date ==> FashionShow
  FashionShow(n, p, sD, eD) == (
   name := n;
   place := p;
   startDate := sD;
   endDate := eD;
   events := [];
   peopleAttending := {};
   return self;
  )
  post events = [] and peopleAttending = {};

-- set name

  public setName : Utils`string ==> ()
   setName(n) ==
    name := n;

-- set place

  public setPlace : Utils`string ==> ()
   setPlace(p) ==
    place := p;

-- set start date

 public setStartDate : Utils`date ==> ()
  setStartDate(d) ==
   startDate := d;

-- set end date

 public setEndDate : Utils`date ==> ()
  setEndDate(d) ==
   endDate := d;

-- set events

 public setEvents : seq of (Event) ==> ()
  setEvents(e) ==
   events := e;

-- add event

 public addEvent : Event ==> ()
  addEvent(e) ==
   events := events ^ [e]
   post events = events~ ^ [e];

-- add person to people attending event

 public addPersonAttending : Person ==> ()
  addPersonAttending(p) ==
   peopleAttending := peopleAttending union {p}
   pre p not in set peopleAttending
   post peopleAttending = peopleAttending~ union {p};

-- get designers attending the event

 public getDesigners : () ==> set of (Person)
  getDesigners() == (
   dcl ret : set of Person := {};
   for all p in set peopleAttending do(
    if p.isDesigner then ret := ret union {p};
   );
   return ret;
  );

end FashionShow
```

Function or operation	Line	Coverage	Calls
FashionShow	25	100.0%	16
addEvent	63	100.0%	17
addPersonAttending	69	100.0%	10
getDesigners	76	100.0%	7
setEndDate	53	100.0%	1
setEvents	58	100.0%	1
setName	38	100.0%	1
setPlace	43	100.0%	1
setStartDate	48	100.0%	1
FashionShow.vdmpp		100.0%	55

4 Notification

```
class Notification

/*

 Defines a notification that a person can create regarding a certain event.
  J. Oliveira, FEUP, MFES, 2017/18.

*/


instance variables
 public person:Person;
 public event:Event;
  public startTime:Utils`date;
  public minToNotify:nat;

 inv event.startDate = startTime;

 operations
-- constructor

 public Notification : Person * Event * nat ==> Notification
   Notification(p,e,m) == (
    person := p;
    event := e;
    startTime := event.startDate;
    minToNotify := m;
    return self;
   )
   post event.startDate = startTime;

 -- set person

  public setPerson : Person ==> ()
   setPerson(p) ==
    person := p;

 -- set event

  public setEvent : Event ==> ()
   setEvent(e) == (
    atomic(event := e;
    startTime := e.startDate;);
   )
   post event.startDate = startTime;

-- set minutes to notification

  public setMinToNotify : nat ==> ()
   setMinToNotify(m) ==
    minToNotify := m;

end Notification
```

Function or operation	Line	Coverage	Calls
Notification	21	100.0%	1
setEvent	37	100.0%	1
setMinToNotify	45	100.0%	1
setPerson	32	100.0%	1
Notification.vdmpp		100.0%	4

5 Person

```
class Person

/*

 Defines a person that will attend a fashion show.
  J. Oliveira, FEUP, MFES, 2017/18.

*/


instance variables
  public name : Utils`string;
  public birthdate : Utils`date;
  public gender : Utils`gender;
  public clothSize : Utils`clothSize;
  public isDesigner : bool;
  public eventsAttending : seq of (Event);
  public ticketToShow : map Ticket to FashionShow;
```

```
operations
-- constructor

 public Person : Utils`string * Utils`date * Utils`gender * Utils`clothSize * bool ==> Person
  Person(n, bD, g, cS, iD) == (
   name := n;
   birthdate := bD;
   gender := g;
   clothSize := cS;
   isDesigner := iD;
   eventsAttending := [];
   ticketToShow := { |-> };
   return self;
  );

--set name

 public setName : Utils`string ==> ()
  setName(n) ==
   name := n;

--set birthdate

 public setBirthdate : Utils`date ==> ()
  setBirthdate(bD) ==
   birthdate := bD;

--set gender

 public setGender : Utils`gender ==> ()
  setGender(g) ==
   gender := g;

-- set cloth size

 public setClothSize : Utils`clothSize ==> ()
  setClothSize(cS) ==
   clothSize := cS;

--set is designer

 public setIsDesigner : bool ==> ()
  setIsDesigner(iD) ==
   isDesigner := iD;

--add event to eventsAttending

 public addEvent : Event ==> ()
  addEvent(e) == (
   eventsAttending := eventsAttending ^ [e];
   e.place.addOccupant(self);
   )
   pre not Utils`existsInSeq[Event](e,eventsAttending) and
      e.show in set rng ticketToShow and
      exists t in set dom ticketToShow & (Utils`isOldestDate(ticketToShow(t).startDate, e.
          startDate) and Utils`isOldestDate(e.endDate,ticketToShow(t).endDate)) and
      not exists te in set elems eventsAttending & Utils`coincDate(te.startDate,te.endDate,e.
          startDate,e.endDate)
   post eventsAttending = eventsAttending~ ^ [e] and self in set elems e.place.occupants;

--set ticketToShow

 public setTicketToShow : map Ticket to FashionShow ==> ()
  setTicketToShow(e) ==
   ticketToShow := e
   post ticketToShow = e;

--add ticket and show to ticketToShow

 public addTicketShow : Ticket * FashionShow ==> ()
  addTicketShow(t,s) ==
   ticketToShow := ticketToShow munion {t |-> s}
   pre not exists tt in set dom ticketToShow & tt.holder = self and (tt = t or ticketToShow(tt) =
        s)
   post ticketToShow = ticketToShow~ munion {t |-> s};

end Person
```

Function or operation	Line	Coverage	Calls
Person	22	100.0%	31
addEvent	60	100.0%	14
addTicketShow	78	100.0%	10
setBirthdate	40	100.0%	1
setClothSize	50	100.0%	1
setGender	45	100.0%	1
setIsDesigner	55	100.0%	1
setName	35	100.0%	1
setTicketToShow	72	100.0%	1
Person.vdmpp		100.0%	61

6 PieceOfCloth

```
class PieceOfCloth

/*

 Defines a piece of cloth designed by a certain designer.
  J. Oliveira, FEUP, MFES, 2017/18.

*/

instance variables
 public creator:Person;
  public size:Utils`clothSize;
  public clothType:Utils`clothType;

--invariants
 inv creator.isDesigner;

operations
-- constructor

 public PieceOfCloth : Person * Utils`clothSize * Utils`clothType ==> PieceOfCloth
  PieceOfCloth(p,s,t) == (
   creator := p;
   size := s;
   clothType := t;
   return self;
  );

-- set creator

 public setCreator : Person ==> ()
  setCreator(p) ==
   creator := p;

-- set size

 public setSize : Utils`clothSize ==> ()
  setSize(s) ==
   size := s;

-- set clothType

 public setClothType : Utils`clothType ==> ()
  setClothType(t) ==
   clothType := t;

end PieceOfCloth
```

Function or operation	Line	Coverage	Calls
PieceOfCloth	20	100.0%	6
setClothType	39	100.0%	1
setCreator	29	100.0%	1
setSize	34	100.0%	1
PieceOfCloth.vdmpp		100.0%	9

7 Presentation

```
class Presentation is subclass of Event

/*

 Defines a presentation, event of a fashion show.
  J. Oliveira, FEUP, MFES, 2017/18.

*/


instance variables
  public subject:Utils`string;

operations
-- constructor

 public Presentation : FashionShow * seq1 of Person * Room * Utils`string * Utils`date * Utils`
     date * Utils`string ==> Presentation
 Presentation(fS, sP, r, n, sD, eD, s) == (
  show := fS;
  organizers := sP;
  place := r;
  name := n;
  atomic(startDate := sD;
  endDate := eD;);
  subject := s;
  fS.addEvent(self);
  return self;
 )
 post Utils`existsInSeq[Event](self,fS.events);

-- set subject

 public setSubject : Utils`string ==> ()
  setSubject(s) ==
   subject := s;

end Presentation
```

Function or operation	Line	Coverage	Calls
Presentation	16	100.0%	5
setSubject	31	100.0%	1
Presentation.vdmpp		100.0%	6

8 PrimpingSession

```
class PrimpingSession is subclass of Event

/*

 Defines a primping session, event of a fashion show.
  J. Oliveira, FEUP, MFES, 2017/18.

*/


instance variables
  public subject:Utils`string;

operations
-- constructor

 public PrimpingSession : FashionShow * seq1 of Person * Room * Utils`string * Utils`date * Utils
     `date * Utils`string ==> PrimpingSession
 PrimpingSession(fS, sP, r, n, sD, eD, s) == (
  show := fS;
  organizers := sP;
  place := r;
  name := n;
  atomic(startDate := sD;
  endDate := eD;);
  subject := s;
  fS.addEvent(self);
  return self;
 )
  post Utils`existsInSeq[Event](self,fS.events);

-- set subject

 public setSubject : Utils`string ==> ()
  setSubject(s) ==
   subject := s;

end PrimpingSession
```

Function or operation	Line	Coverage	Calls
PrimpingSession	16	100.0%	2
setSubject	31	100.0%	1
PrimpingSession.vdmpp		100.0%	3

9 Room

```
class Room

/*

 Defines a room, that will host an event of a fashion show.
  J. Oliveira, FEUP, MFES, 2017/18.

*/


instance variables
  public name : Utils`string;
  public localization : Utils`string;
 public capacity : nat1;
 public occupants : seq of Person;

-- invariants
  inv len occupants <= capacity;

operations
-- constructor

 public Room : Utils`string * Utils`string * nat1 ==> Room
  Room(n, local, cap) == (
   name := n;
   localization := local;
   capacity := cap;
   occupants := [];
   return self;
  );

-- set name

  public setName : Utils`string ==> ()
   setName(n) ==
    name := n;

-- set localization

  public setLocalization : Utils`string ==> ()
   setLocalization(l) ==
    localization := l;

-- set capacity
```

```
  public setCapacity : nat1 ==> ()
   setCapacity(c) ==
    capacity := c
    pre c >= len occupants;

-- add a person to the occupants list

 public addOccupant : Person ==> ()
  addOccupant(p) ==
   occupants := occupants ^ [p]
   pre len occupants < capacity
   post len occupants <= capacity and
        occupants = occupants~ ^ [p];

-- make the room empty

 public emptyTheRoom : () ==> ()
  emptyTheRoom() ==
   occupants := []
   post occupants = [];

end Room
```

Function or operation	Line	Coverage	Calls
Room	22	100.0%	16
addOccupant	48	100.0%	11
emptyTheRoom	56	100.0%	4
setCapacity	42	100.0%	1
setLocalization	37	100.0%	1
setName	32	100.0%	1
Room.vdmpp		100.0%	34

10 RunwayShow

```
class RunwayShow is subclass of Event

/*

 Defines a runway show, event of a fashion show.
  J. Oliveira, FEUP, MFES, 2017/18.

*/


instance variables
  public theme:Utils`string;

operations
-- constructor

 public RunwayShow : FashionShow * seq1 of Person * Room * Utils`string * Utils`date * Utils`date
      * Utils`string ==> RunwayShow
  RunwayShow(fS, sP, r, n, sD, eD, t) == (
   show := fS;
   organizers := sP;
```

```
    place := r;
    name := n;
    atomic(startDate := sD;
    endDate := eD;);
   theme := t;
   fS.addEvent(self);
   return self;
  )
  post Utils`existsInSeq[Event](self,fS.events);

-- set name

  public setTheme : Utils`string ==> ()
   setTheme(t) ==
    theme := t;

end RunwayShow
```

Function or operation	Line	Coverage	Calls
RunwayShow	16	100.0%	10
setTheme	31	100.0%	1
RunwayShow.vdmpp		100.0%	11

11 Ticket

```
class Ticket

/*

 Defines a ticket for a fashion show.
  J. Oliveira, FEUP, MFES, 2017/18.

*/

instance variables
 public holder : Person;
 public show : FashionShow;
  public startDate : Utils`date;
  public endDate : Utils`date;
  public type : Utils`ticketType;

-- invariants
inv Utils`isOldestDate(startDate, endDate);
inv Utils`isOldestDate(show.startDate, startDate) or show.startDate = startDate;
inv Utils`isOldestDate(endDate, show.endDate) or endDate = show.endDate;
inv if type = <Designer> then holder.isDesigner else true;

operations
-- constructor

 public Ticket : Person * FashionShow * Utils`date * Utils`date * Utils`ticketType ==> Ticket
  Ticket(p, s, sD, eD, t) == (
   holder := p;
    atomic(show := s;
    startDate := sD;
    endDate := eD;);
   type := t;
   s.addPersonAttending(p);
   p.addTicketShow(self,s);
   return self;
  );

-- set ticket holder

 public setHolder : Person ==> ()
  setHolder(p) ==
   holder := p;

--set fashion show

 public setShow : FashionShow ==> ()
  setShow(s) ==
   show := s;

-- set start date

 public setStartDate : Utils`date ==> ()
  setStartDate(d) ==
   startDate := d;

-- set end date

 public setEndDate : Utils`date ==> ()
  setEndDate(d) ==
   endDate := d;

-- set ticket type

 public setType : Utils`ticketType ==> ()
  setType(t) ==
   type := t;

end Ticket
```

Function or operation	Line	Coverage	Calls
Ticket	25	100.0%	10
setEndDate	53	100.0%	1
setHolder	38	100.0%	1
setShow	43	100.0%	1
setStartDate	48	100.0%	1
setType	58	100.0%	1
Ticket.vdmpp		100.0%	15

12 Utils

```
class Utils

/*

 Class that contains some useful code common to the other classes, this includes:
 Class-like types:
```

```
 - string;
 - date.

 Enumerations-like types:
 - gender;
 - clothSize;
 - clothType;
 - tickeType.

 Operations:
 - assertTrue().

 Functions:
 - existsInSeq();
 - daysOfMonth();
 - isLeapYear();
 - isOldestDate();
 - coincDate();

*/

types
 public string = seq of char;
 public date :: year : nat
             month: nat1
          day : nat1
          hour : nat
          minute : nat
     inv d == d.month <= 12 and
          d.day <= daysOfMonth(d.year, d.month) and
          d.hour <= 23 and
          d.minute <= 59;

 public gender = <Male> | <Female>;
 public clothSize = <XL> | <L> | <M> | <S> | <XS>;
 public clothType = <Shirt> | <Jacket> | <Pants> | <Shoes> | <Hat>;
 public ticketType = <Designer> | <Worker> | <Volunteer> | <Guest> | <Sponsor> | <Attendee>;

operations


 public static assertTrue: bool ==> ()
  assertTrue(cond) == return
  pre cond;

functions

-- function that checks if e exists in s

 public static existsInSeq[@T] : @T * seq of @T -> bool
 existsInSeq(e,s)==
  exists t in set elems s & t = e;

-- function that returns the number of days in a month

 public daysOfMonth : nat * nat -> nat1
  daysOfMonth(year,month) ==
  if month in set {1,3,5,7,8,10,12} then 31
  elseif month in set {4,6,9,11} then 30
  elseif isLeapYear(year) and month = 2 then 29
  else 28;

-- function that says if a given year is a leap year or not

 public static isLeapYear : nat1 -> bool
```

```
  isLeapYear(year)==
   year mod 4 = 0 and year mod 100 <> 0 or
   year mod 400 = 0;

-- checks if d1 is older then d2

 public static isOldestDate : date * date -> bool
  isOldestDate(d1, d2) ==
   if d1.year <> d2.year then d1.year < d2.year
   else if d1.month <> d2.month then d1.month < d2.month
   else if d1.day <> d2.day then d1.day < d2.day
   else if d1.hour <> d2.hour then d1.hour < d2.hour
   else if d1.minute <> d2.minute then d1.minute < d2.minute
   else false;

-- checks if pair (sd1, ed1) is coincident with (sd2, ed2) are coincident

 public static coincDate : date * date * date * date -> bool
  coincDate(sd1, ed1, sd2, ed2) ==
   if isOldestDate(ed1,sd2) or ed1 = sd2 or isOldestDate(ed2,sd1) or ed2 = sd1 then false
   else true;

end Utils
```

Function or operation	Line	Coverage	Calls
assertTrue	47	100.0%	318
coincDate	83	100.0%	66
daysOfMonth	59	100.0%	1
existsInSeq	54	100.0%	41
isLeapYear	67	100.0%	37
isOldestDate	73	100.0%	250
Utils.vdmpp		100.0%	713

13 ClothDisplayedTest

```
class ClothDisplayedTest

/*

 Defines the test scenarios and test cases for the ClothDisplayed class.
  J. Oliveira, FEUP, MFES, 2017/18.

*/

instance variables
 pTest1:Person := new Person("Test Person1", mk_Utils`date(1996,12,15,16,00), <Female>, <S>,
     false);
 pTest2:Person := new Person("Test Person2", mk_Utils`date(1996,12,15,16,00), <Male>, <L>, true);
 pTest3:Person := new Person("Test Person3", mk_Utils`date(1994,10,15,16,00), <Female>, <S>,
     false);
 cTest1:PieceOfCloth := new PieceOfCloth(pTest2,<S>,<Shirt>);
 cTest2:PieceOfCloth := new PieceOfCloth(pTest2,<S>,<Pants>);
 rTest1:Room := new Room("Sala1","Edificio A, Piso 2",50);
 fasTest1:FashionShow  := new FashionShow("1234Show", "MEO Arena", mk_Utils`date(2017,12,15,8,00)
     ,  mk_Utils`date(2017,12,20,00,30));
 fasTest2:FashionShow  := new FashionShow("5678Show", "MEO Arena", mk_Utils`date(2017,12,15,8,00)
     ,  mk_Utils`date(2017,12,20,00,30));
 runTest1:RunwayShow := new RunwayShow(fasTest1, [pTest2],rTest1, "NameTest",  mk_Utils`date
     (2017,12,15,16,00),  mk_Utils`date(2017,12,15,17,30), "testTheme");
 runTest2:RunwayShow := new RunwayShow(fasTest1, [pTest2],rTest1, "NameTest",  mk_Utils`date
     (2017,12,15,14,00),  mk_Utils`date(2017,12,15,15,30), "testTheme");
 cDTest1:ClothDisplayed := new ClothDisplayed(pTest1,cTest1,runTest1);

operations

 private testClothDisplayed : () ==> ()
  testClothDisplayed() == (
   --test contructor
   Utils`assertTrue(cDTest1.model = pTest1);
   Utils`assertTrue(cDTest1.cloth = cTest1);
   Utils`assertTrue(cDTest1.runway = runTest1);

   --test setModel
   cDTest1.setModel(pTest3);
   Utils`assertTrue(cDTest1.model = pTest3);

   --test setCloth
   cDTest1.setCloth(cTest2);
   Utils`assertTrue(cDTest1.cloth = cTest2);

   --test setCloth
   cDTest1.setRunwayShow(runTest2);
   Utils`assertTrue(cDTest1.runway = runTest2);
  );


 public static main: () ==> ()
  main() == (
   new ClothDisplayedTest().testClothDisplayed();
  );

end ClothDisplayedTest
```

Function or operation	Line	Coverage	Calls
main	44	100.0%	1
testClothDisplayed	24	100.0%	1
ClothDisplayedTest.vdmpp		100.0%	2

14 FashionShowTest

```
class FashionShowTest

/*

 Defines the test scenarios and test cases for the FashionShow class.
  J. Oliveira, FEUP, MFES, 2017/18.

*/

instance variables
```

```
 perTest1:Person := new Person("Test Person1", mk_Utils`date(1996,12,15,16,00), <Male>, <L>,
     false);
 perTest2:Person := new Person("Test Person2", mk_Utils`date(1995,12,15,16,00), <Male>, <L>, true
     );
 perTest3:Person := new Person("Test Person3", mk_Utils`date(1994,12,15,16,00), <Male>, <L>, true
     );
 fasTest1:FashionShow  := new FashionShow("1234Show", "MEO Arena", mk_Utils`date(2017,12,15,8,00)
     ,  mk_Utils`date(2017,12,20,00,30));
 rTest1:Room := new Room("Sala1","Edificio A, Piso 2",50);
 rTest2:Room := new Room("Sala2","Edificio B, Piso 3",5);
 orgTest: seq of (Person) := [perTest1];
 preTest1:Presentation := new Presentation(fasTest1,orgTest,rTest1, "NameTest",  mk_Utils`date
     (2017,12,15,16,00),  mk_Utils`date(2017,12,15,17,30), "testSubject");
 preTest2:Presentation := new Presentation(fasTest1,orgTest,rTest2, "NameTest2",  mk_Utils`date
     (2017,12,15,16,00),  mk_Utils`date(2017,12,15,17,30), "testSubject");
 runTest1:RunwayShow := new RunwayShow(fasTest1,orgTest,rTest1, "PreTest",  mk_Utils`date
     (2017,12,15,18,00),  mk_Utils`date(2017,12,15,19,30), "testTheme");
 tTest1:Ticket := new Ticket(perTest1, fasTest1, mk_Utils`date(2017,12,15,8,00),  mk_Utils`date
     (2017,12,20,00,30), <Worker>);
 tTest2:Ticket := new Ticket(perTest2, fasTest1, mk_Utils`date(2017,12,15,8,00),  mk_Utils`date
     (2017,12,20,00,30), <Designer>);
 tTest3:Ticket := new Ticket(perTest3, fasTest1, mk_Utils`date(2017,12,15,8,00),  mk_Utils`date
     (2017,12,20,00,30), <Attendee>);


operations

 private testFashionShow : () ==> ()
  testFashionShow() == (
   --test contructor
   Utils`assertTrue(fasTest1.name = "1234Show");
   Utils`assertTrue(fasTest1.place = "MEO Arena");
   Utils`assertTrue(fasTest1.startDate = mk_Utils`date(2017,12,15,8,00));
   Utils`assertTrue(fasTest1.endDate = mk_Utils`date(2017,12,20,00,30));
   Utils`assertTrue(fasTest1.events = [preTest1,preTest2,runTest1]);

   --test setName()
   fasTest1.setName("5678Show");
   Utils`assertTrue(fasTest1.name = "5678Show");

   --test setPlace()
   fasTest1.setPlace("5678 Street");
   Utils`assertTrue(fasTest1.place = "5678 Street");

   --test setStartDate()
   fasTest1.setStartDate(mk_Utils`date(2017,12,15,7,00));
   Utils`assertTrue(fasTest1.startDate = mk_Utils`date(2017,12,15,7,00));

   --test setEndDate()
   fasTest1.setEndDate(mk_Utils`date(2017,12,20,01,00));
   Utils`assertTrue(fasTest1.endDate = mk_Utils`date(2017,12,20,01,00));

   --test setEvents()
   fasTest1.setEvents([preTest1,runTest1]);
   Utils`assertTrue(fasTest1.events = [preTest1,runTest1]);

   --test getDesigners()
   Utils`assertTrue(fasTest1.getDesigners() = {perTest2,perTest3});

  );


 public static main: () ==> ()
  main() == (
   new FashionShowTest().testFashionShow();
  );

end FashionShowTest
```

Function or operation	Line	Coverage	Calls
main	61	100.0%	2
testFashionShow	27	100.0%	1
FashionShowTest.vdmpp		100.0%	3

15 NotificationTest

```
class NotificationTest

/*

 Defines the test scenarios and test cases for the Notification class.
  J. Oliveira, FEUP, MFES, 2017/18.

*/

instance variables
 perTest1:Person := new Person("Test Person1", mk_Utils`date(1996,12,15,16,00), <Male>, <L>,
     false);
 perTest2:Person := new Person("Test Person2", mk_Utils`date(1991,12,15,16,00), <Female>, <S>,
     false);
 perTest3:Person := new Person("Test Person3", mk_Utils`date(1986,12,15,16,00), <Male>, <XL>,
     true);
 orgTest1: seq of (Person) := [perTest1, perTest2, perTest3];
 roomTest1:Room := new Room("Sala1","Edificio A, Piso 2",50);
 fasTest1:FashionShow  := new FashionShow("1234Show", "MEO Arena", mk_Utils`date(2017,12,15,8,00)
     ,  mk_Utils`date(2017,12,20,00,30));
 fasTest2:FashionShow  := new FashionShow("5678Show", "MEO Arena", mk_Utils`date(2017,12,15,8,00)
     ,  mk_Utils`date(2017,12,20,00,30));
 rTest1:RunwayShow := new RunwayShow(fasTest1, [perTest1],roomTest1, "NameTest",  mk_Utils`date
     (2017,12,15,16,00),  mk_Utils`date(2017,12,15,17,30), "testTheme");
 rTest2:RunwayShow := new RunwayShow(fasTest1, [perTest1],roomTest1, "NameTest",  mk_Utils`date
     (2017,12,15,8,00),  mk_Utils`date(2017,12,15,15,30), "testTheme");
 nTest:Notification := new Notification(perTest1,rTest1,15);

operations
 --test constructor

 private testNotification : () ==> ()
  testNotification() == (
   --test contructor
    Utils`assertTrue(nTest.person = perTest1);
    Utils`assertTrue(nTest.event = rTest1);
    Utils`assertTrue(nTest.startTime = mk_Utils`date(2017,12,15,16,00));
    Utils`assertTrue(nTest.minToNotify = 15);

   --test setPerson()
    nTest.setPerson(perTest2);
    Utils`assertTrue(nTest.person = perTest2);

   --test setEvent()
    nTest.setEvent(rTest2);
    Utils`assertTrue(nTest.event = rTest2);
    Utils`assertTrue(nTest.startTime = mk_Utils`date(2017,12,15,8,00));

   --test setMinToNotify()
    nTest.setMinToNotify(20);
    Utils`assertTrue(nTest.minToNotify = 20);


  );


 public static main: () ==> ()
  main() == (
   new NotificationTest().testNotification();
  );

end NotificationTest
```

Function or operation	Line	Coverage	Calls
main	48	100.0%	2
testNotification	24	100.0%	1
NotificationTest.vdmpp		100.0%	3

16 PersonTest

```
class PersonTest

/*

 Defines the test scenarios and test cases for the Person class.
  J. Oliveira, FEUP, MFES, 2017/18.

*/

instance variables
 pTest :Person := new Person("Test Person", mk_Utils`date(1996,12,15,16,00), <Male>, <L>, false);
 pTest1:Person := new Person("Test Person1", mk_Utils`date(1996,12,15,16,00), <Male>, <L>, false)
     ;
 rTest1:Room := new Room("Sala1","Edificio A, Piso 2",50);
 rTest2:Room := new Room("Sala2","Edificio B, Piso 3",5);
 orgTest: seq of (Person) := [pTest1];
 fasTest1:FashionShow  := new FashionShow("1234Show", "MEO Arena", mk_Utils`date(2017,12,15,8,00)
     ,  mk_Utils`date(2017,12,20,00,30));
 fasTest2:FashionShow  := new FashionShow("5678Show", "MEO Arena", mk_Utils`date(2017,12,15,8,00)
     ,  mk_Utils`date(2017,12,20,00,30));
 preTest:Presentation := new Presentation(fasTest1,orgTest,rTest1, "NameTest",  mk_Utils`date
     (2017,12,15,16,00),  mk_Utils`date(2017,12,15,17,30), "testSubject");
 runTest:RunwayShow := new RunwayShow(fasTest1,orgTest,rTest2, "PreTest",  mk_Utils`date
     (2017,12,15,12,00),  mk_Utils`date(2017,12,15,15,30), "testTheme");
 ticket1:Ticket := new Ticket(pTest, fasTest1, mk_Utils`date(2017,12,15,8,00),  mk_Utils`date
     (2017,12,20,00,30), <Worker>);
 ticket2:Ticket := new Ticket(pTest, fasTest2, mk_Utils`date(2017,12,15,8,00),  mk_Utils`date
     (2017,12,20,00,30), <Worker>);

operations
```

```
 private testPerson : () ==> ()
  testPerson() == (
  --test constructor
   Utils`assertTrue(pTest.name = "Test Person");
    Utils`assertTrue(pTest.birthdate = mk_Utils`date(1996,12,15,16,00));
   Utils`assertTrue(pTest.gender = <Male>);
   Utils`assertTrue(pTest.clothSize = <L>);
   Utils`assertTrue(pTest.isDesigner = false);

  --test setName()
   pTest.setName("Test Person2");
   Utils`assertTrue(pTest.name = "Test Person2");

  --test setBirthdate()
   pTest.setBirthdate(mk_Utils`date(1995,12,15,16,00));
   Utils`assertTrue(pTest.birthdate = mk_Utils`date(1995,12,15,16,00));

  --test setGender()
   pTest.setGender(<Female>);
   Utils`assertTrue(pTest.gender = <Female>);

  --test setClothsize()
   pTest.setClothSize(<XS>);
   Utils`assertTrue(pTest.clothSize = <XS>);

  --test setIsDesigner
   pTest.setIsDesigner(true);
   Utils`assertTrue(pTest.isDesigner = true);


  --test addEvent()
   pTest.addEvent(preTest);
   pTest.addEvent(runTest);
   Utils`assertTrue(pTest.eventsAttending = [preTest, runTest]);
   Utils`assertTrue(len pTest.eventsAttending = 2);

  --test setTicketToShow
   pTest.setTicketToShow({ticket2 |-> fasTest2});
   Utils`assertTrue(pTest.ticketToShow = {ticket2 |-> fasTest2});
  );


 public static main: () ==> ()
  main() == (
   new PersonTest().testPerson();
  );

end PersonTest
```

Function or operation	Line	Coverage	Calls
main	65	100.0%	2
testPerson	24	100.0%	1
PersonTest.vdmpp		100.0%	3

17 PieceOfClothTest

```
class PieceOfClothTest

/*

 Defines the test scenarios and test cases for the PieceOfCloth class.
  J. Oliveira, FEUP, MFES, 2017/18.

*/

instance variables
 pTest1:Person := new Person("Test Person1", mk_Utils`date(1996,12,15,16,00), <Female>, <S>, true
     );
 pTest2:Person := new Person("Test Person2", mk_Utils`date(1996,12,15,16,00), <Male>, <L>, true);
 pTest3:Person := new Person("Test Person3", mk_Utils`date(1994,10,15,16,00), <Female>, <S>,
     false);
 cTest1:PieceOfCloth := new PieceOfCloth(pTest1,<S>,<Shirt>);
 cTest2:PieceOfCloth := new PieceOfCloth(pTest2,<S>,<Pants>);

operations

 private testPieceOfCloth : () ==> ()
  testPieceOfCloth() == (
   --test contructor
   Utils`assertTrue(cTest1.creator = pTest1);
   Utils`assertTrue(cTest1.size = <S>);
   Utils`assertTrue(cTest1.clothType = <Shirt>);

   --test setCreator
   cTest1.setCreator(pTest2);
   Utils`assertTrue(cTest1.creator = pTest2);

   --test setClothSize
   cTest1.setSize(<M>);
   Utils`assertTrue(cTest1.size = <M>);

   --test setClothType
   cTest1.setClothType(<Pants>);
   Utils`assertTrue(cTest1.clothType = <Pants>);
  );


 public static main: () ==> ()
  main() == (
   new PieceOfClothTest().testPieceOfCloth();
  );

end PieceOfClothTest
```

Function or operation	Line	Coverage	Calls
main	38	100.0%	1
testPieceOfCloth	18	100.0%	1
PieceOfClothTest.vdmpp		100.0%	2

18 PresentationTest

```
class PresentationTest
```

```
/*

 Defines the test scenarios and test cases for the Presentation class.
  J. Oliveira, FEUP, MFES, 2017/18.

*/

instance variables
 perTest1:Person := new Person("Test Person1", mk_Utils`date(1996,12,15,16,00), <Male>, <L>,
     false);
 perTest2:Person := new Person("Test Person2", mk_Utils`date(1991,12,15,16,00), <Female>, <S>,
     false);
 perTest3:Person := new Person("Test Person3", mk_Utils`date(1986,12,15,16,00), <Male>, <XL>,
     true);
 orgTest1: seq of (Person) := [perTest1, perTest2, perTest3];
 rTest1:Room := new Room("Sala1","Edificio A, Piso 2",50);
 rTest2:Room := new Room("Sala2","Edificio B, Piso 3",5);
 fasTest1:FashionShow  := new FashionShow("1234Show", "MEO Arena", mk_Utils`date(2017,12,15,8,00)
     ,  mk_Utils`date(2017,12,20,00,30));
 fasTest2:FashionShow  := new FashionShow("5678Show", "MEO Arena", mk_Utils`date(2017,12,15,8,00)
     ,  mk_Utils`date(2017,12,20,00,30));
 pTest:Presentation := new Presentation(fasTest1, [perTest1],rTest1, "NameTest",  mk_Utils`date
     (2017,12,15,16,00),  mk_Utils`date(2017,12,15,17,30), "testSubject");

operations

 private testPresentation : () ==> ()
  testPresentation() == (
  --test constructor
   Utils`assertTrue(pTest.organizers = [perTest1]);
   Utils`assertTrue(pTest.place = rTest1);
   Utils`assertTrue(pTest.name = "NameTest");
   Utils`assertTrue(pTest.startDate = mk_Utils`date(2017,12,15,16,00));
   Utils`assertTrue(pTest.endDate = mk_Utils`date(2017,12,15,17,30));
   Utils`assertTrue(pTest.subject = "testSubject");

  --test setPlace()
   pTest.setPlace(rTest2);
   Utils`assertTrue(pTest.place = rTest2);

  --test setName()
   pTest.setName("nameTest1");
   Utils`assertTrue(pTest.name = "nameTest1");

  --test setStartDate()
   pTest.setStartDate(mk_Utils`date(2017,12,14,16,00));
   Utils`assertTrue(pTest.startDate = mk_Utils`date(2017,12,14,16,00));

  --test setEndDate()
   pTest.setEndDate(mk_Utils`date(2017,12,14,17,00));
   Utils`assertTrue(pTest.endDate = mk_Utils`date(2017,12,14,17,00));

  --test addOrganizer()
   --pTest.addOrganizer(perTest1);
   pTest.addOrganizer(perTest2);
   Utils`assertTrue(pTest.organizers = [perTest1, perTest2]);

  --test setOrganizers()
   pTest.setOrganizers(orgTest1);
   Utils`assertTrue(pTest.organizers = [perTest1, perTest2, perTest3]);

  --test endEvent()
   pTest.place.addOccupant(perTest1);
   pTest.endEvent();
   Utils`assertTrue(pTest.place.occupants = []);

  --test setSubject()
   pTest.setSubject("testSubject1");
   Utils`assertTrue(pTest.subject = "testSubject1");
  );


 public static main: () ==> ()
  main() == (
   new PresentationTest().testPresentation();
  );

end PresentationTest
```

Function or operation	Line	Coverage	Calls
main	67	100.0%	2
testPresentation	22	100.0%	1
PresentationTest.vdmpp		100.0%	3

19 PrimpingSessionTest

```
class PrimpingSessionTest

/*

 Defines the test scenarios and test cases for the PrimpingSession class.
  J. Oliveira, FEUP, MFES, 2017/18.

*/

instance variables
 perTest1:Person := new Person("Test Person1", mk_Utils`date(1996,12,15,16,00), <Male>, <L>,
     false);
 perTest2:Person := new Person("Test Person2", mk_Utils`date(1991,12,15,16,00), <Female>, <S>,
     false);
 perTest3:Person := new Person("Test Person3", mk_Utils`date(1986,12,15,16,00), <Male>, <XL>,
     true);
 orgTest1: seq of (Person) := [perTest1, perTest2, perTest3];
 rTest1:Room := new Room("Sala1","Edificio A, Piso 2",50);
 rTest2:Room := new Room("Sala2","Edificio B, Piso 3",5);
 fasTest1:FashionShow  := new FashionShow("1234Show", "MEO Arena", mk_Utils`date(2017,12,15,8,00)
     ,  mk_Utils`date(2017,12,20,00,30));
 fasTest2:FashionShow  := new FashionShow("5678Show", "MEO Arena", mk_Utils`date(2017,12,15,8,00)
     ,  mk_Utils`date(2017,12,20,00,30));
 pTest:PrimpingSession := new PrimpingSession(fasTest1, [perTest1],rTest1, "NameTest",  mk_Utils`
     date(2017,12,15,16,00),  mk_Utils`date(2017,12,15,17,30), "testSubject");

operations

 private testPrimpingSession : () ==> ()
  testPrimpingSession() == (
  --test constructor
   Utils`assertTrue(pTest.organizers = [perTest1]);
   Utils`assertTrue(pTest.place = rTest1);
   Utils`assertTrue(pTest.name = "NameTest");
   Utils`assertTrue(pTest.startDate = mk_Utils`date(2017,12,15,16,00));
   Utils`assertTrue(pTest.endDate = mk_Utils`date(2017,12,15,17,30));
   Utils`assertTrue(pTest.subject = "testSubject");

  --test setPlace()
   pTest.setPlace(rTest2);
   Utils`assertTrue(pTest.place = rTest2);

  --test setName()
   pTest.setName("nameTest1");
   Utils`assertTrue(pTest.name = "nameTest1");

  --test setStartDate()
   pTest.setStartDate(mk_Utils`date(2017,12,14,16,00));
   Utils`assertTrue(pTest.startDate = mk_Utils`date(2017,12,14,16,00));

  --test setEndDate()
   pTest.setEndDate(mk_Utils`date(2017,12,14,17,00));
   Utils`assertTrue(pTest.endDate = mk_Utils`date(2017,12,14,17,00));

  --test addOrganizer()
   --pTest.addOrganizer(perTest1);
   pTest.addOrganizer(perTest2);
   Utils`assertTrue(pTest.organizers = [perTest1, perTest2]);

  --test setOrganizers()
   pTest.setOrganizers(orgTest1);
   Utils`assertTrue(pTest.organizers = [perTest1, perTest2, perTest3]);

  --test endEvent()
   pTest.place.addOccupant(perTest1);
   pTest.endEvent();
   Utils`assertTrue(pTest.place.occupants = []);

  --test setSubject()
   pTest.setSubject("testSubject1");
   Utils`assertTrue(pTest.subject = "testSubject1");
  );


 public static main: () ==> ()
  main() == (
   new PrimpingSessionTest().testPrimpingSession();
  );

end PrimpingSessionTest
```

Function or operation	Line	Coverage	Calls
main	67	100.0%	1
testPrimpingSession	22	100.0%	1
PrimpingSessionTest.vdmpp		100.0%	2

20 RoomTest

```
class RoomTest
```

```
/*

 Defines the test scenarios and test cases for the Room class.
  J. Oliveira, FEUP, MFES, 2017/18.

*/

instance variables
 pTest : Person := new Person("TestPerson", mk_Utils`date(1996,12,15,16,00), <Male>, <L>, false);
 rTest:Room := new Room("Sala1","Edificio A, Piso 2",50);
operations

 private testRoom : () ==> ()
  testRoom() == (
   Utils`assertTrue(rTest.name = "Sala1");
    Utils`assertTrue(rTest.localization = "Edificio A, Piso 2");
   Utils`assertTrue(rTest.capacity = 50);
   rTest.setName("Sala11");
   Utils`assertTrue(rTest.name = "Sala11");
   rTest.setLocalization("Edificio B, Piso 1");
   Utils`assertTrue(rTest.localization = "Edificio B, Piso 1");
   rTest.setCapacity(5);
   Utils`assertTrue(rTest.capacity = 5);
   rTest.addOccupant(pTest);
   Utils`assertTrue(len rTest.occupants = 1);
   Utils`assertTrue(rTest.occupants =  [pTest]);
   rTest.emptyTheRoom();
   Utils`assertTrue(rTest.occupants =  []);
  );


 public static main: () ==> ()
  main() == (
   new RoomTest().testRoom();
  );

end RoomTest
```

Function or operation	Line	Coverage	Calls
main	32	100.0%	2
testRoom	14	100.0%	1
RoomTest.vdmpp		100.0%	3

21 RunwayShowTest

```
class RunwayShowTest

/*

 Defines the test scenarios and test cases for the Utils class.
  J. Oliveira, FEUP, MFES, 2017/18.

*/

instance variables
```

```
 perTest1:Person := new Person("Test Person1", mk_Utils`date(1996,12,15,16,00), <Male>, <L>,
     false);
 perTest2:Person := new Person("Test Person2", mk_Utils`date(1991,12,15,16,00), <Female>, <S>,
     false);
 perTest3:Person := new Person("Test Person3", mk_Utils`date(1986,12,15,16,00), <Male>, <XL>,
     true);
 orgTest1: seq of (Person) := [perTest1, perTest2, perTest3];
 rTest1:Room := new Room("Sala1","Edificio A, Piso 2",50);
 rTest2:Room := new Room("Sala2","Edificio B, Piso 3",5);
 fasTest1:FashionShow  := new FashionShow("1234Show", "MEO Arena", mk_Utils`date(2017,12,15,8,00)
     ,  mk_Utils`date(2017,12,20,00,30));
 fasTest2:FashionShow  := new FashionShow("5678Show", "MEO Arena", mk_Utils`date(2017,12,15,8,00)
     ,  mk_Utils`date(2017,12,20,00,30));
 rTest:RunwayShow := new RunwayShow(fasTest1, [perTest1],rTest1, "NameTest",  mk_Utils`date
     (2017,12,15,16,00),  mk_Utils`date(2017,12,15,17,30), "testTheme");

operations

 private testRunwayShow : () ==> ()
  testRunwayShow() == (
  --test constructor
   Utils`assertTrue(rTest.organizers = [perTest1]);
   Utils`assertTrue(rTest.place = rTest1);
   Utils`assertTrue(rTest.name = "NameTest");
   Utils`assertTrue(rTest.startDate = mk_Utils`date(2017,12,15,16,00));
   Utils`assertTrue(rTest.endDate = mk_Utils`date(2017,12,15,17,30));
   Utils`assertTrue(rTest.theme = "testTheme");

  --test setPlace()
   rTest.setPlace(rTest2);
   Utils`assertTrue(rTest.place = rTest2);

  --test setName()
   rTest.setName("nameTest1");
   Utils`assertTrue(rTest.name = "nameTest1");

  --test setStartDate()
   rTest.setStartDate(mk_Utils`date(2017,12,14,16,00));
   Utils`assertTrue(rTest.startDate = mk_Utils`date(2017,12,14,16,00));

  --test setEndDate()
   rTest.setEndDate(mk_Utils`date(2017,12,14,17,00));
   Utils`assertTrue(rTest.endDate = mk_Utils`date(2017,12,14,17,00));

  --test addOrganizer()
   --rTest.addOrganizer(perTest1);
   rTest.addOrganizer(perTest2);
   Utils`assertTrue(rTest.organizers = [perTest1, perTest2]);

  --test setOrganizers()
   rTest.setOrganizers(orgTest1);
   Utils`assertTrue(rTest.organizers = [perTest1, perTest2, perTest3]);

  --test endEvent()
   rTest.place.addOccupant(perTest1);
   rTest.endEvent();
   Utils`assertTrue(rTest.place.occupants = []);

  --test setTheme()
   rTest.setTheme("testTheme1");
   Utils`assertTrue(rTest.theme = "testTheme1");
  );


 public static main: () ==> ()
```

```
  main() == (
   new RunwayShowTest().testRunwayShow();
  );

end RunwayShowTest
```

Function or operation	Line	Coverage	Calls
main	67	100.0%	1
testRunwayShow	22	100.0%	1
RunwayShowTest.vdmpp		100.0%	2

22 TestScenario

```
class TestScenario

/*

 Tests the scenario 1 explained in the report
  J. Oliveira, FEUP, MFES, 2017/18.

*/

instance variables
 fasTest1 : FashionShow;
 pTest1 : Person;
 pTest2 : Person;
 pTest3 : Person;
 pTest4 : Person;
 presTest1 : Presentation;
 primpTest1 : PrimpingSession;
 runTest1 : RunwayShow;
 runTest2 : RunwayShow;
 runTest3 : RunwayShow;
 roomTest1 : Room;
 roomTest2 : Room;
 roomTest3 : Room;
 ticTest1 : Ticket;
 ticTest2 : Ticket;
 ticTest3 : Ticket;
 ticTest4 : Ticket;
 cTest1 : PieceOfCloth;
 cTest2 : PieceOfCloth;
 cdTest1 : ClothDisplayed;
 cdTest2 : ClothDisplayed;




operations

 private test : () ==> ()
  test() == (

  --test scenario 1 -> creating fashion show 1234Show
   fasTest1 := new FashionShow("1234Show", "MEO Arena", mk_Utils`date(2017,12,15,8,00),  mk_Utils
       `date(2017,12,20,00,30));
   Utils`assertTrue(fasTest1.name = "1234Show");
   Utils`assertTrue(fasTest1.place = "MEO Arena");
   Utils`assertTrue(fasTest1.startDate = mk_Utils`date(2017,12,15,8,00));
   Utils`assertTrue(fasTest1.endDate = mk_Utils`date(2017,12,20,00,30));


  --fasTest1 := new FashionShow("1234Show", "MEO Arena", mk_Utils`date(2017,12,20,8,00),
      mk_Utils`date(2017,12,15,00,30));
   /*Would fail because end date is previous to start date*/


  --test scenario 2 -> creating one event of each type
   pTest1 := new Person("Test Person1", mk_Utils`date(1996,12,15,16,00), <Male>, <L>, false);
   pTest2 := new Person("Test Person2", mk_Utils`date(1995,10,15,16,00), <Female>, <S>, false);
   pTest3 := new Person("Test Person3", mk_Utils`date(1995,7,15,16,00), <Female>, <M>, true);
   pTest4 := new Person("Test Person4", mk_Utils`date(1994,10,5,16,00), <Male>, <XL>, true);

   roomTest1 := new Room("Sala1","Edificio A, Piso 1",50);
   roomTest2 := new Room("Sala2","Edificio A, Piso 2",50);
   roomTest3 := new Room("Sala3","Edificio A, Piso 2",3);

   presTest1 := new Presentation(fasTest1, [pTest1], roomTest1, "TestPresentation1",  mk_Utils`
       date(2017,12,15,16,00),  mk_Utils`date(2017,12,15,17,30), "testSubject");
   primpTest1 := new PrimpingSession(fasTest1, [pTest1], roomTest2, "TestPrimping1",  mk_Utils`
       date(2017,12,15,16,00),  mk_Utils`date(2017,12,15,17,30), "testSubject");
   runTest1 := new RunwayShow(fasTest1, [pTest3, pTest4], roomTest1, "TestRunway1",  mk_Utils`
       date(2017,12,15,17,30),  mk_Utils`date(2017,12,15,18,30), "testTheme");
   runTest2 := new RunwayShow(fasTest1, [pTest3], roomTest3, "TestRunway1",  mk_Utils`date
       (2017,12,15,17,30),  mk_Utils`date(2017,12,15,18,30), "testTheme");
   runTest3 := new RunwayShow(fasTest1, [pTest3], roomTest3, "TestRunway1",  mk_Utils`date
       (2017,12,15,18,30),  mk_Utils`date(2017,12,15,19,30), "testTheme");

   --runTest1 := new RunwayShow(fasTest1, [pTest1], roomTest1, "TestPresentation1",  mk_Utils`
       date(2017,12,15,16,00),  mk_Utils`date(2017,12,15,17,30), "testSubject");
   /* would fail because there is already an event  in roomTest1 at the same time and pTest1 isn'
       t a designer*/

   --runTest1 := new RunwayShow(fasTest1, [], roomTest1, "TestPresentation1",  mk_Utils`date
       (2017,12,15,16,00),  mk_Utils`date(2017,12,15,17,30), "testSubject");
   /* would fail because there is already an event  in roomTest1 at the same time and there is no
        organizers*/

   --runTest1 := new RunwayShow(fasTest1, [pTest1], roomTest1, "TestPresentation1",  mk_Utils`
       date(2017,12,15,16,00),  mk_Utils`date(2017,12,25,17,30), "testSubject");
   /* would fail because there is already an event  in roomTest1 at the same time, pTest1 isn't a
        designer and the end date is after the end date of the main event*/

   Utils`assertTrue(len presTest1.organizers >= 1);
   Utils`assertTrue(len primpTest1.organizers >= 1);
   Utils`assertTrue(len runTest1.organizers >= 1);
   Utils`assertTrue(not exists e1, e2 in set elems fasTest1.events & (e1 <> e2 and e1.place = e2.
       place and Utils`coincDate(e1.startDate, e1.endDate, e2.startDate, e2.endDate)));

  --test scenario 3
   ticTest1 := new Ticket(pTest1,fasTest1,mk_Utils`date(2017,12,15,16,00),mk_Utils`date
       (2017,12,15,16,30),<Guest>);
   ticTest2 := new Ticket(pTest2,fasTest1,mk_Utils`date(2017,12,15,16,00),mk_Utils`date
       (2017,12,15,18,30),<Worker>);
   ticTest3 := new Ticket(pTest3,fasTest1,mk_Utils`date(2017,12,15,16,00),mk_Utils`date
       (2017,12,17,16,30),<Designer>);
   ticTest4 := new Ticket(pTest4,fasTest1,mk_Utils`date(2017,12,15,16,00),mk_Utils`date
       (2017,12,15,16,30),<Guest>);
```

```
   --ticTest3 := new Ticket(pTest2,fasTest1,mk_Utils`date(2016,10,15,16,00),mk_Utils`date
       (2016,10,15,16,30),<Designer>);
   /*would fail because pTest2 isn't a designer, the dates are before the date of the show and
       because pTest2 already as an ticket for this event*/

   Utils`assertTrue(fasTest1.peopleAttending = {pTest1,pTest2,pTest3,pTest4});
   Utils`assertTrue(fasTest1.getDesigners() = {pTest3, pTest4});
   Utils`assertTrue(exists t in set dom pTest1.ticketToShow & (pTest1.ticketToShow(t) = fasTest1)
       );
   Utils`assertTrue(exists t in set dom pTest2.ticketToShow & (pTest2.ticketToShow(t) = fasTest1)
       );
   Utils`assertTrue(exists t in set dom pTest3.ticketToShow & (pTest3.ticketToShow(t) = fasTest1)
       );
   Utils`assertTrue(exists t in set dom pTest4.ticketToShow & (pTest4.ticketToShow(t) = fasTest1)
       );
   Utils`assertTrue(pTest1 in set fasTest1.peopleAttending);
   Utils`assertTrue(pTest2 in set fasTest1.peopleAttending);
   Utils`assertTrue(pTest3 in set fasTest1.peopleAttending);
   Utils`assertTrue(pTest4 in set fasTest1.peopleAttending);

  --test scenario 4

   cTest1 := new PieceOfCloth(pTest3,<S>,<Shirt>);
   cTest2 := new PieceOfCloth(pTest4,<M>,<Pants>);

   --Test2 := new PieceOfCloth(pTest2,<M>,<Pants>);
   /* Would fail because pTest2 isn't a designer*/

  --test scenario 5
   pTest1.addEvent(presTest1);
   pTest2.addEvent(primpTest1);

   pTest1.addEvent(runTest2);
   pTest2.addEvent(runTest2);
   pTest3.addEvent(runTest2);

   --pTest4.addEvent(runTest2);
   /*would fail because the room that hosts the event runTest2 is full*/

   --pTest3.addEvent(runTest1);
   /*would fail because pTest3 is already attending runTest2 at the same time*/


   --pTest1.addEvent(presTest1);
   /*would fail because the pTest1 is already attending runTest1*/

  --test scenario 6
   cdTest1 := new ClothDisplayed(pTest2,cTest1,runTest1);

   --cdTest1 := new ClothDisplayed(pTest3,cTest2,runTest2);
   /*would fail because the creator of cTest2 isn't a organizer of runTest2*/

   --cdTest2 := new ClothDisplayed(pTest2,cTest2,runTest1)
   /*would fail because pTest2 wears size S and cTest2 is of size M*/
  );

 public static main: () ==> ()
  main() == (
   new TestScenario().test();
  );

end TestScenario
```

Function or operation	Line	Coverage	Calls
main	125	100.0%	1
test	34	100.0%	1
TestScenario.vdmpp		100.0%	2

23 Tests

```
class Tests

/*

 Superclass for test classes.
  J. Oliveira, FEUP, MFES, 2017/18.

*/

operations

  public static main: () ==> ()
   main() == (
   FashionShowTest`main();
   TicketTest`main();
   PersonTest`main();
   PresentationTest`main();
   PrimpingSessionTest`main();
   RunwayShowTest`main();
   PieceOfClothTest`main();
   ClothDisplayedTest`main();
   NotificationTest`main();
   RoomTest`main();
   UtilsTest`main();
   TestScenario`main();
  );

end Tests
```

Function or operation	Line	Coverage	Calls
main	11	100.0%	1
Tests.vdmpp		100.0%	1

24 TicketTest

```
class TicketTest

/*

 Defines the test scenarios and test cases for the Ticket class.
  J. Oliveira, FEUP, MFES, 2017/18.

*/
```

```
instance variables
 pTest1:Person := new Person("Test Person1", mk_Utils`date(1996,12,15,16,00), <Female>, <S>,
     false);
 pTest2:Person := new Person("Test Person2", mk_Utils`date(1996,12,15,16,00), <Male>, <L>, true);
 pTest3:Person := new Person("Test Person3", mk_Utils`date(1994,10,15,16,00), <Female>, <S>,
     false);
 fasTest1:FashionShow  := new FashionShow("1234Show", "MEO Arena", mk_Utils`date(2014,12,15,8,00)
     ,  mk_Utils`date(2017,12,20,00,30));
 fasTest2:FashionShow  := new FashionShow("5678Show", "MEO Arena", mk_Utils`date(2014,12,15,8,00)
     ,  mk_Utils`date(2017,12,20,00,30));
 tTest:Ticket := new Ticket(pTest1,fasTest1,mk_Utils`date(2016,10,15,16,00),mk_Utils`date
     (2016,10,15,16,30),<Guest>);

operations


 private testTicket : () ==> ()
  testTicket() == (
   --test contructor
    Utils`assertTrue(tTest.holder = pTest1);
    Utils`assertTrue(tTest.show = fasTest1);
    Utils`assertTrue(tTest.startDate = mk_Utils`date(2016,10,15,16,00));
    Utils`assertTrue(tTest.endDate = mk_Utils`date(2016,10,15,16,30));
    Utils`assertTrue(tTest.type = <Guest>);

   --test setHolder()
    tTest.setHolder(pTest2);
    Utils`assertTrue(tTest.holder = pTest2);

   --test setFashionShow()
    tTest.setShow(fasTest2);
    Utils`assertTrue(tTest.show = fasTest2);

   --test setStartDate()
    tTest.setStartDate(mk_Utils`date(2016,10,14,16,00));
    Utils`assertTrue(tTest.startDate = mk_Utils`date(2016,10,14,16,00));

   --test setEndDate()
    tTest.setEndDate(mk_Utils`date(2016,10,14,16,30));
    Utils`assertTrue(tTest.endDate = mk_Utils`date(2016,10,14,16,30));

   --test setTicketType()
    tTest.setType(<Designer>);
    Utils`assertTrue(tTest.type = <Designer>);
  );


 public static main: () ==> ()
  main() == (
   new TicketTest().testTicket();
  );


end TicketTest
```

Function or operation	Line	Coverage	Calls
main	50	100.0%	2
testTicket	20	100.0%	1
TicketTest.vdmpp		100.0%	3

25 UtilsTest

```
class UtilsTest

/*

 Defines the test scenarios and test cases for the Utils class.
  J. Oliveira, FEUP, MFES, 2017/18.

*/

instance variables
 str : Utils`string;
 dat : Utils`date;
 gen : Utils`string;
 cSize : Utils`string;
 cType : Utils`string;
 tType : Utils`string;
 testSeq : seq of nat := [1,2,3,4,5];

operations

 public testString : () ==> ()
  testString() == (
   str := "Teste";
   Utils`assertTrue(str = "Teste");
  );


 public testDate : () ==> ()
  testDate() == (
   str := "Teste";
   Utils`assertTrue(str = "Teste");
   dat := mk_Utils`date(2017,12,29,17,15);
   Utils`assertTrue(dat.year = 2017);
   Utils`assertTrue(dat.month = 12);
   Utils`assertTrue(dat.day = 29);
   Utils`assertTrue(dat.hour = 17);
   Utils`assertTrue(dat.minute = 15);

   dat := mk_Utils`date(2017,11,29,17,15);
   Utils`assertTrue(dat.year = 2017);
   Utils`assertTrue(dat.month = 11);
   Utils`assertTrue(dat.day = 29);
   Utils`assertTrue(dat.hour = 17);
   Utils`assertTrue(dat.minute = 15);

   dat := mk_Utils`date(2016,2,29,17,15);
   Utils`assertTrue(dat.year = 2016);
   Utils`assertTrue(dat.month = 2);
   Utils`assertTrue(dat.day = 29);
   Utils`assertTrue(dat.hour = 17);
   Utils`assertTrue(dat.minute = 15);

   dat := mk_Utils`date(2015,2,28,17,15);
   Utils`assertTrue(dat.year = 2015);
   Utils`assertTrue(dat.month = 2);
   Utils`assertTrue(dat.day = 28);
   Utils`assertTrue(dat.hour = 17);
   Utils`assertTrue(dat.minute = 15);

   Utils`assertTrue(not Utils`isOldestDate(mk_Utils`date(2015,2,28,17,15),mk_Utils`date
        (2015,2,28,17,15)));
   Utils`assertTrue(Utils`isOldestDate(mk_Utils`date(2014,2,28,17,15),mk_Utils`date
       (2015,2,28,17,15)));
   Utils`assertTrue(Utils`isOldestDate(mk_Utils`date(2015,1,28,17,15),mk_Utils`date
       (2015,2,28,17,15)));
   Utils`assertTrue(Utils`isOldestDate(mk_Utils`date(2015,2,27,17,15),mk_Utils`date
       (2015,2,28,17,15)));
   Utils`assertTrue(Utils`isOldestDate(mk_Utils`date(2015,2,28,16,15),mk_Utils`date
       (2015,2,28,17,15)));
   Utils`assertTrue(Utils`isOldestDate(mk_Utils`date(2015,2,28,17,14),mk_Utils`date
       (2015,2,28,17,15)));

   Utils`assertTrue(Utils`coincDate(mk_Utils`date(2015,2,28,17,14),mk_Utils`date(2015,2,28,17,15)
       ,mk_Utils`date(2015,2,28,17,14),mk_Utils`date(2015,2,28,17,15)));
   Utils`assertTrue(not Utils`coincDate(mk_Utils`date(2015,2,28,17,14),mk_Utils`date
       (2015,2,28,17,15),mk_Utils`date(2015,2,28,17,15),mk_Utils`date(2015,2,28,17,17)));
   Utils`assertTrue(not Utils`coincDate(mk_Utils`date(2015,2,28,17,14),mk_Utils`date
       (2015,2,28,17,15),mk_Utils`date(2015,2,28,17,16),mk_Utils`date(2015,2,28,17,17)));

   Utils`assertTrue(Utils`coincDate(mk_Utils`date(2015,2,28,17,14),mk_Utils`date(2015,2,28,17,15)
       ,mk_Utils`date(2015,2,28,17,14),mk_Utils`date(2015,2,28,17,15)));
   Utils`assertTrue(not Utils`coincDate(mk_Utils`date(2015,2,28,17,15),mk_Utils`date
       (2015,2,28,17,17),mk_Utils`date(2015,2,28,17,14),mk_Utils`date(2015,2,28,17,15)));
   Utils`assertTrue(not Utils`coincDate(mk_Utils`date(2015,2,28,17,16),mk_Utils`date
       (2015,2,28,17,17),mk_Utils`date(2015,2,28,17,14),mk_Utils`date(2015,2,28,17,15)));

  );


 public testExistInSeq : () ==> ()
  testExistInSeq() == (
   Utils`assertTrue(Utils`existsInSeq[nat](1,testSeq) = true);
   Utils`assertTrue(Utils`existsInSeq[nat](10,testSeq) = false);
  );


 public static main: () ==> ()
  main() == (
   new UtilsTest().testString();
   new UtilsTest().testDate();
   new UtilsTest().testExistInSeq();
  );

end UtilsTest
```

Function or operation	Line	Coverage	Calls
main	81	100.0%	2
testDate	26	100.0%	1
testExistInSeq	75	100.0%	1
testString	20	100.0%	1
UtilsTest.vdmpp		100.0%	5