FashionShow

January 1, 2018

Contents

1 ClothDisplayed

```
 class ClothDisplayed
types
-- TODO Define types here
values
-- TODO Define values here
instance variables
 public model : Person;
 public cloth : PieceOfCloth;
 public runway : RunwayShow;

 inv model.clothSize = cloth.size;

operations
-- TODO Define operations here
-- 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;

functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end ClothDisplayed
```

Function or operation	Line	Coverage	Calls
ClothDisplayed	16	100.0%	1
setCloth	34	100.0%	2
setModel	27	100.0%	2
setRunwayShow	42	100.0%	1
ClothDisplayed.vdmpp		100.0%	6

2 Event

```
class Event
instance variables
 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	19	100.0%	2
endEvent	44	100.0%	2
setEndDate	39	100.0%	2
setName	29	100.0%	2
setOrganizers	14	100.0%	2
setPlace	24	100.0%	2
setStartDate	34	100.0%	2
Event.vdmpp		100.0%	14

3 FashionShow

```
class FashionShow
instance variables
 public name : Utils`string;
  public place : Utils`string;
  public startDate : Utils`date;
  public endDate : Utils`date;
  public events : seq of (Event);

-- 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 := [];
   return self;
  );

-- 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];

end FashionShow
```

Function or operation	Line	Coverage	Calls
FashionShow	15	100.0%	13
addEvent	51	100.0%	11
setEndDate	41	100.0%	1
setEvents	46	100.0%	1
setName	26	100.0%	1
setPlace	31	100.0%	1
setStartDate	36	100.0%	1
FashionShow.vdmpp		100.0%	29

4 Notification

```
class Notification
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	12	100.0%	1
setEvent	28	100.0%	2
setMinToNotify	36	100.0%	1
setPerson	23	100.0%	1
Notification.vdmpp		100.0%	5

5 Person

```
class Person
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]
   pre not Utils`existsInSeq[Event](e,eventsAttending) and
      not exists te in set elems eventsAttending & Utils`coincDate(te.startDate,te.endDate,e.
          startDate,e.endDate)
   post eventsAttending = eventsAttending~ ^ [e];

--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 = t or ticketToShow(tt) = s)
   post ticketToShow = ticketToShow~ munion {t |-> s};

end Person
```

Function or operation	Line	Coverage	Calls
Person	14	100.0%	22
addEvent	52	100.0%	2
addTicketShow	66	100.0%	4
setBirthdate	32	100.0%	1
setClothSize	42	100.0%	1
setGender	37	100.0%	1
setIsDesigner	47	100.0%	1
setName	27	100.0%	1
setTicketToShow	60	100.0%	1
Person.vdmpp		100.0%	34

6 PieceOfCloth

```
class PieceOfCloth
types
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	13	100.0%	4
setClothType	32	100.0%	1
setCreator	22	100.0%	1
setSize	27	100.0%	1
PieceOfCloth.vdmpp		100.0%	7

7 Presentation

```
class Presentation is subclass of Event
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) == (
  organizers := sP;
  place := r;
  name := n;
  atomic(startDate := sD;
  endDate := eD;);
  subject := s;
  fS.addEvent(self);
  return self;
 );

-- set subject

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

end Presentation
```

Function or operation	Line	Coverage	Calls
Presentation	7	100.0%	4
setSubject	20	100.0%	1
Presentation.vdmpp		100.0%	5

8 Room

```
class Room
```

```
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	13	100.0%	11

addOccupant	39	100.0%	3
emptyTheRoom	47	100.0%	3
setCapacity	33	100.0%	1
setLocalization	28	100.0%	1
setName	23	100.0%	1
Room.vdmpp		100.0%	20

9 RunwayShow

```
class RunwayShow is subclass of Event
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) == (
   organizers := sP;
    place := r;
    name := n;
    atomic(startDate := sD;
    endDate := eD;);
   theme := t;
   fS.addEvent(self);
   return self;
  );

-- set name

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

end RunwayShow
```

Function or operation	Line	Coverage	Calls
RunwayShow	7	100.0%	7
setTheme	20	100.0%	1
RunwayShow.vdmpp		100.0%	8

10 Ticket

```
class Ticket
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;
   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	17	100.0%	3
setEndDate	44	100.0%	1
setHolder	29	100.0%	1
setShow	34	100.0%	1
setStartDate	39	100.0%	1

setType	49	100.0%	1
Ticket.vdmpp		100.0%	8

11 Utils

```
class Utils
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>;
values
-- TODO Define values here
instance variables
-- TODO Define instance variables here
operations
-- TODO Define operations here


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

functions
-- TODO Define functiones here


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

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

 public DaysOfMonth(year:nat,month:nat1) res:nat1 ==
  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(year: nat1) res : bool ==
  year mod 4 = 0 and year mod 100 <> 0 or
  year mod 400 = 0;

-- checks if d1 is older then d2

 public static isOldestDate(d1:date, d2:date) res : bool ==
  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(sd1:date, ed1:date, sd2:date, ed2:date) res : bool ==
  if isOldestDate(ed1,sd2) or ed1 = sd2 or isOldestDate(ed2,sd1) or ed2 = sd1 then false
  else true;


traces
-- TODO Define Combinatorial Test Traces here
end Utils
```

Function or operation	Line	Coverage	Calls
DaysOfMonth	35	100.0%	1
IsLeapYear	42	100.0%	36
assertTrue	24	100.0%	254
coincDate	56	100.0%	17
existsInSeq	31	100.0%	7
isOldestDate	47	100.0%	135
Utils.vdmpp		100.0%	450

12 ClothDisplayedTest

```
class ClothDisplayedTest
types
-- TODO Define types here
values
-- TODO Define values here
instance variables
-- TODO Define instance variables here
 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();
  );
functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end ClothDisplayedTest
```

Function or operation	Line	Coverage	Calls
main	41	100.0%	1
testClothDisplayed	21	100.0%	1
ClothDisplayedTest.vdmpp		100.0%	2

13 FashionShowTest

```
class FashionShowTest
types
-- TODO Define types here
values
-- TODO Define values here
instance variables
-- TODO Define instance variables here
 perTest1:Person := new Person("Test Person1", mk_Utils`date(1996,12,15,16,00), <Male>, <L>,
     false);
 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");
```

```
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]);
  );

 public static main: () ==> ()
  main() == (
   new FashionShowTest().testFashionShow();
  );
functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end FashionShowTest
```

Function or operation	Line	Coverage	Calls
main	39	100.0%	2
testFashionShow	18	100.0%	1
FashionShowTest.vdmpp		100.0%	3

14 NotificationTest

```
class NotificationTest
types
-- TODO Define types here
values
```

```
-- TODO Define values here
instance variables
-- TODO Define instance variables here
 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();
  );
functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end NotificationTest
```

Function or operation	Line	Coverage	Calls

main	46	100.0%	1
testNotification	22	100.0%	1
NotificationTest.vdmpp		100.0%	2

15 PersonTest

```
class PersonTest
types
-- TODO Define types here
values
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
     (2017,12,15,18,00),  mk_Utils`date(2017,12,15,19,30), "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
-- TODO Define operations here

 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 addEvent()
   pTest.addEvent(preTest);
```

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

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

  --test setTicketToShow
   pTest.setTicketToShow({ticket1 |-> fasTest1});
   Utils`assertTrue(pTest.ticketToShow = {ticket1 |-> fasTest1});

  --test addTicketShow()
   pTest.addTicketShow(ticket2, fasTest2);
   Utils`assertTrue(pTest.ticketToShow = {ticket1 |-> fasTest1, ticket2 |-> fasTest2});

   --pTest.addTicketShow(ticket1, fasTest1);
  );


 public static main: () ==> ()
  main() == (
   new PersonTest().testPerson();
  );
functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end PersonTest
```

Function or operation	Line	Coverage	Calls
main	66	100.0%	2
testPerson	20	100.0%	1
PersonTest.vdmpp		100.0%	3

16 PieceOfClothTest

```
class PieceOfClothTest
types
-- TODO Define types here
values
-- TODO Define values here
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
-- TODO Define operations here

 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();
  );
functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end PieceOfClothTest
```

Function or operation	Line	Coverage	Calls
main	34	100.0%	2
testPieceOfCloth	14	100.0%	1
PieceOfClothTest.vdmpp		100.0%	3

17 PresentationTest

```
class PresentationTest
types
-- TODO Define types here
values
-- TODO Define values here
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();
  );
functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end PresentationTest
```

Function or operation	Line	Coverage	Calls
main	62	100.0%	2

testPresentation	17	100.0%	1
PresentationTest.vdmpp		100.0%	3

18 RoomTest

```
class RoomTest
types
-- TODO Define types here
values
-- TODO Define values here
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();
  );
functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end RoomTest
```

Function or operation	Line	Coverage	Calls
main	28	100.0%	2
testRoom	10	100.0%	1
RoomTest.vdmpp		100.0%	3

19 RunwayShowTest

```
class RunwayShowTest
```

```
types
-- TODO Define types here
values
-- TODO Define values here
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();
  );functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end RunwayShowTest
```

Function or operation	Line	Coverage	Calls
main	63	100.0%	2
testRunwayShow	18	100.0%	1
RunwayShowTest.vdmpp		100.0%	3

20 Tests

```
class Tests
types
-- TODO Define types here
values
-- TODO Define values here
instance variables
-- TODO Define instance variables here
operations

  public static main: () ==> ()
   main() == (
   RoomTest`main();
   UtilsTest`main();
   PresentationTest`main();
   PersonTest`main();
   NotificationTest`main();
   RunwayShowTest`main();
   PieceOfClothTest`main();
   FashionShowTest`main();
   ClothDisplayedTest`main();
   TicketTest`main();
  );
functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end Tests
```

Function or operation	Line	Coverage	Calls
main	9	100.0%	1
Tests.vdmpp		100.0%	1

21 TicketTest

```
class TicketTest
types
-- TODO Define types here
values
-- TODO Define values here
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
-- TODO Define operations here

 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();
  );

functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end TicketTest
```

Function or operation	Line	Coverage	Calls
main	45	100.0%	2
testTicket	15	100.0%	1
TicketTest.vdmpp		100.0%	3

22 UtilsTest

```
class UtilsTest
types
-- TODO Define types here
values
-- TODO Define values here
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();
  );


functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end UtilsTest
```

Function or operation	Line	Coverage	Calls
main	75	100.0%	2
testDate	21	100.0%	1
testExistInSeq	70	100.0%	1

testString	15	100.0%	1
UtilsTest.vdmpp		100.0%	5