ExhibitionCenter

January 4, 2019

Contents

```
class Auditorium is subclass of Installation
instance variables
 public foyers: set of Foyer := {};
operations
-- Constructor (must have telephones and movingWalls always false)

 public Auditorium: Utils_vdm`String * real * nat * real * real * real * bool * bool * bool *
     bool * bool * bool ==> Auditorium
  Auditorium(i, pr, c, h, w, l, airC, natL, ceilL, blackC, compN, soundP) == (
   --Measures
   id := i;
   price := pr;
   capacity := c;
   heigth := h;
   atomic(width := w; lenght := l; area := w * l);
   --Conditions
   airCondition := airC;
   naturalLigth := natL;
   ceilingLighting := ceilL;
   blackOutCurtains := blackC;
   computerNetwork := compN;
   soundproofWalls := soundP;
  )
  pre c > 0 and h > 0 and w > 0 and l > 0 and pr > 0
  post not (telephones or movingWalls);

--Impossible to create a installation with out parameters
  public Auditorium: () ==> Auditorium
  Auditorium() == return self
  pre false;

--Change installation's conditions

 public setConditions: bool * bool * bool * bool * bool * bool * bool * bool ==> ()
  setConditions(airC, natL, ceilL, blackC, -, compN, soundW, -) == (
   airCondition := airC;
   naturalLigth := natL;
   ceilingLighting := ceilL;
   blackOutCurtains := blackC;
   computerNetwork := compN;
   soundproofWalls := soundW
  )
  post not (telephones or movingWalls);

--Shows all installation information

 public showDetails: () ==> map Utils_vdm`String to Utils_vdm`String
```

```
 showDetails() == (
  dcl res: map Utils_vdm`String to Utils_vdm`String := {|->};
  res := res ++ {
    "ID" |-> id, "Price" |-> Utils_vdm`toStringVDM(price), "Capacity" |-> Utils_vdm`toStringVDM(
        capacity),
    "Area" |-> Utils_vdm`toStringVDM(area), "Heigth" |-> Utils_vdm`toStringVDM(heigth),
    "Width" |-> Utils_vdm`toStringVDM(width), "Lenght" |-> Utils_vdm`toStringVDM(lenght),
    "Air Condition" |-> Utils_vdm`toStringVDM(airCondition),
    "Natural Ligth" |-> Utils_vdm`toStringVDM(naturalLigth),
    "Ceiling Lighting" |-> Utils_vdm`toStringVDM(ceilingLighting),
    "Black Out Curtains" |-> Utils_vdm`toStringVDM(blackOutCurtains),
    "Computer Network" |-> Utils_vdm`toStringVDM(computerNetwork),
    "Soundproof Walls" |-> Utils_vdm`toStringVDM(soundproofWalls)
  };
  return res
 );

-- Add foyer to foyers set

 public addFoyer : Foyer ==> ()
  addFoyer(f) == foyers := foyers union {f}
  pre f not in set foyers
  post card foyers = card foyers~ + 1 and f in set foyers;

-- Remove foyer from foyers set

 public removeFoyer : Foyer ==> ()
  removeFoyer(f) == foyers := foyers \ {f}
  pre f in set foyers
  post card foyers = card foyers~ - 1 and f not in set foyers;


 public addRoom : Room ==> ()
  addRoom(-) == return
  pre false;

 public removeRoom : Room ==> ()
  removeRoom(-) == return
  pre false;

--Verify if has installation

 public hasInstallation: Installation ==> bool
  hasInstallation(inst) == (
   if(isofclass(Foyer, inst)) then return (let f = narrow_(inst, Foyer) in f in set foyers)
   else return false
  );
end Auditorium
```

Function or operation	Line	Coverage	Calls
Auditorium	6	0.0%	0
addFoyer	61	100.0%	6
addRoom	72	0.0%	0
hasInstallation	80	100.0%	90
removeFoyer	67	100.0%	6
removeRoom	75	0.0%	0
setConditions	31	100.0%	6
showDetails	43	100.0%	6

Auditorium.vdmpp		96.2%	114

```
class CarParking is subclass of Installation

operations
-- Constructor (must have all conditions always false)

 public CarParking: Utils_vdm`String * real * nat * real * real * real ==> CarParking
  CarParking(i, pr, c, h, w, l) == (
   --Measures
   id := i;
   price := pr;
   capacity := c;
   heigth := h;
   atomic(width := w; lenght := l; area := w * l);
  )
  pre c > 0 and h > 0 and w > 0 and l > 0
  post not (airCondition or naturalLigth or ceilingLighting or
  blackOutCurtains or telephones or computerNetwork or
  soundproofWalls or movingWalls);
--Impossible to create a installation with out parameters
 public CarParking: () ==> CarParking
  CarParking() == return self
  pre false;

--Change installation's conditions

 public setConditions: bool * bool * bool * bool * bool * bool * bool * bool ==> ()
  setConditions(-, -, -, -, -, -, -, -) == (
   skip;
  );

--Shows all installation information

 public showDetails: () ==> map Utils_vdm`String to Utils_vdm`String
 showDetails() == (
  dcl res: map Utils_vdm`String to Utils_vdm`String := {|->};
  res := res ++ {
    "ID" |-> id, "Price" |-> Utils_vdm`toStringVDM(price), "Capacity" |-> Utils_vdm`toStringVDM(
        capacity),
    "Area" |-> Utils_vdm`toStringVDM(area), "Heigth" |-> Utils_vdm`toStringVDM(heigth),
    "Width" |-> Utils_vdm`toStringVDM(width), "Lenght" |-> Utils_vdm`toStringVDM(lenght)
  };
  return res
 );


 public addFoyer : Foyer ==> ()
  addFoyer(-) == return
  pre false;

 public removeFoyer : Foyer ==> ()
  removeFoyer(-) == return
  pre false;

 public addRoom : Room ==> ()
  addRoom(-) == return
  pre false;

 public removeRoom : Room ==> ()
  removeRoom(-) == return
  pre false;
```

```
--Verify if has installation

 public hasInstallation: Installation ==> bool
  hasInstallation(-) == return false;
end CarParking
```

Function or operation	Line	Coverage	Calls
CarParking	5	0.0%	0
addFoyer	41	0.0%	0
addRoom	47	0.0%	0
hasInstallation	54	100.0%	54
removeFoyer	44	0.0%	0
removeRoom	50	0.0%	0
setConditions	24	100.0%	6
showDetails	30	100.0%	6
CarParking.vdmpp		88.7%	66

```
class Center
types
 public DateType = <begin> | <ending>;
 public UserPosition = <attendee> | <staff> | <host>;
values
 private adminName: Utils_vdm`String = "admin";
 private adminPass: Utils_vdm`String = "admin1234";

instance variables
 public name: Utils_vdm`String;
 public installations: map Utils_vdm`String to Installation;
 inv card dom installations > 0 and
  forall id in set dom installations &  installations(id).id = id;
 public services: map Service`ServiceType to Service := {|->};
 inv forall type in set dom services & services(type).type = type;
 public events: map Utils_vdm`String to Event := {|->};
 public users: map Utils_vdm`String to User := {|->};

operations

 public Center:  Utils_vdm`String * Installation ==> Center
  Center(n, inst) == (
   let instID = inst.getID() in
   installations := {instID |-> inst};
   name := n;
  )
  post card dom installations = 1 and name = n;

--INSTALLATION OPERATIONS
--Show one installation information

 public showInstallationDetails: Utils_vdm`String ==> map Utils_vdm`String to Utils_vdm`String
 showInstallationDetails(inst) == return installations(inst).showDetails()
 pre inst in set dom installations;

--Show all intallations information

 public showInstallationsDetails: () ==> set of (map Utils_vdm`String to Utils_vdm`String)
 showInstallationsDetails() == (
```

```
 dcl res: set of (map Utils_vdm`String to Utils_vdm`String) := {};
 for all inst in set dom installations do res := res union {showInstallationDetails(inst)};
 return res
 )
 post card RESULT = card dom installations;

-- Add one installation to installations set

 public addInstallations: Utils_vdm`String * Installation ==> ()
  addInstallations(uName, inst) == let instID = inst.getID() in
  installations := installations ++ {instID |-> inst}
  pre uName in set dom users and uName = adminName and
   inst.id not in set dom installations and isofbaseclass(Installation, inst)
  post card dom installations = card dom installations~ + 1 and inst in set rng installations;

-- Add installation, foyer or room, to other installation, auditorium or pavilion\

 public addInstallationToInstallation: Utils_vdm`String * Utils_vdm`String * Installation ==> ()
 addInstallationToInstallation(uName, instID, inst1) == (
  installations := installations ++ {inst1.id |-> inst1};
  let inst2 = installations(instID) in (
   if(isofclass(Foyer, inst1)) then let foyer = narrow_(inst1, Foyer) in inst2.addFoyer(foyer)
   elseif(isofclass(Room, inst1)) then let room = narrow_(inst1, Room) in inst2.addRoom(room)
  )
 )
 pre uName in set dom users and uName = adminName and
  instID in set dom installations and
  (isofclass(Foyer, inst1) or isofclass(Room, inst1)) and
   (
   (isofclass(Foyer, inst1) and (isofclass(Auditorium, installations(instID)) or isofclass(
       Pavilion, installations(instID)))) or
   (isofclass(Room, inst1) and isofclass(Pavilion, installations(instID)))
  ) and
  not exists inst in set rng installations & (
   (isofclass(Auditorium, inst) or isofclass(Pavilion, inst)) and
   hasInstallation(inst1, inst)
  )
 post inst1 in set rng installations
  and hasInstallation(inst1,installations(instID));

-- Remove installation, foyer or room, from other installation, auditorium or pavilion\

 public removeInstallationFromInstallation: Utils_vdm`String * Utils_vdm`String * Installation
     ==> ()
 removeInstallationFromInstallation(uName, instID, inst1) == (
  let inst2 = installations(instID) in (
   if(isofclass(Foyer, inst1)) then let foyer = narrow_(inst1, Foyer) in inst2.removeFoyer(foyer)
   elseif(isofclass(Room, inst1)) then let room = narrow_(inst1, Room) in inst2.removeRoom(room)
  )
 )
 pre uName in set dom users and uName = adminName and
  instID in set dom installations and
  inst1 in set rng installations and
  (isofclass(Foyer, inst1) or isofclass(Room, inst1)) and
  let inst = installations(instID) in hasInstallation(inst1, inst) and (
   (isofclass(Foyer, inst1) and (isofclass(Auditorium, installations(instID)) or isofclass(
       Pavilion, installations(instID)))) or
   (isofclass(Room, inst1) and isofclass(Pavilion, installations(instID)))
  )
 post inst1 in set rng installations and
 not hasInstallation(inst1, installations(instID));

--Remove installation from installations map

 public removeInstallation: Utils_vdm`String * Utils_vdm`String ==> ()
```

```
 removeInstallation(uName, instID) == (
  dcl instToRemove: set of Utils_vdm`String := {instID};
  for all inst in set rng installations \ {installations(instID)} do (
   if(hasInstallation(inst, installations(instID))) then (instToRemove := instToRemove union {
       inst.id});
   if(inst.hasInstallation(installations(instID))) then (removeInstallationFromInstallation(uName
       , inst.id, installations(instID)))
  );
  installations := instToRemove <-: installations
 )
 pre uName in set dom users and uName = adminName and
  instID in set dom installations
 post instID not in set dom installations and
 not exists inst in set rng installations & hasInstallation(installations~(instID), inst);

--Edit installation measures

 public changeInstallationMeasures: Utils_vdm`String * Utils_vdm`String * nat * real * real *
     real ==> ()
 changeInstallationMeasures(uName, inst, c, h, w, l) == installations(inst).setMeasures(c, h, w,
     l)
 pre uName in set dom users and uName = adminName and
  inst in set dom installations;

--Edit installation price per day

 public changeInstallationRent: Utils_vdm`String * Utils_vdm`String * real ==> ()
 changeInstallationRent(uName, inst, pr) == installations(inst).setPrice(pr)
 pre uName in set dom users and uName = adminName and
  inst in set dom installations;

--Edit installation's conditions

 public changeInstallationConditions: Utils_vdm`String * Utils_vdm`String * bool * bool * bool *
     bool * bool * bool * bool ==> ()
 changeInstallationConditions(uName, inst, airC, natL, ceilL, blackC, tele, compN, soundW, movW)
     == installations(inst).setConditions(airC, natL, ceilL, blackC, tele, compN, soundW, movW)
 pre uName in set dom users and uName = adminName and
  inst in set dom installations;

--Gets all the installations available in the given period

 public getAvailableInstallations: Utils_vdm`Date * Utils_vdm`Date ==> map Utils_vdm`String to
     Installation
 getAvailableInstallations(b, e) == (
  dcl tmpUsedInsts: set of Installation := {};
  dcl usedInsts: set of Installation := {};
  dcl tmpFreeInsts: map Utils_vdm`String to Installation := {|->};
  for all ev in set rng Utils_vdm`fMap[Utils_vdm`String, bool, Event](lambda x: Event &
      hasDatesConflict(x, b, e))(Utils_vdm`setTOseq[Utils_vdm`String](dom events), events) do (
   tmpUsedInsts :=  tmpUsedInsts union {ev.installation}
  );
  tmpFreeInsts := installations :-> usedInsts;
  usedInsts := tmpUsedInsts;
  for all inst1 in set rng tmpFreeInsts do (
   for all inst2 in set tmpUsedInsts do (
    if(inst1.hasInstallation(inst2)) then usedInsts :=  usedInsts union {inst1};
    if(inst2.hasInstallation(inst1)) then usedInsts :=  usedInsts union {inst1}
   )
  );
  return tmpFreeInsts :-> usedInsts;
 )
 pre Utils_vdm`before(b, e) or b = e
 post rng RESULT subset rng installations and
 forall ev in set rng Utils_vdm`fMap[Utils_vdm`String, bool, Event](lambda x: Event &
```

```
     hasDatesConflict(x, b, e))(Utils_vdm`setTOseq[Utils_vdm`String](dom events), events) &
 ev.installation not in set rng RESULT;

--Checks if this two installations are the same or associated
-- (if inst2 is part of inst1)

 public associatedInstallations: Installation * Installation ==> bool
 associatedInstallations(inst1, inst2) == (
  if(inst1 = inst2) then return true
  else inst1.hasInstallation(inst2)
 )
 pre {inst1, inst2} subset rng installations;

/** SERVICE OPERATIONS **/
--Show all services information

 public showServicesDetails: () ==> set of (map Utils_vdm`String to Utils_vdm`String)
 showServicesDetails() == (
 dcl res: set of (map Utils_vdm`String to Utils_vdm`String) := {};
 for all service in set rng services do res := res union {service.showDetails()};
 return res
 )
 post card RESULT = card dom services;

-- Add one service to services set

 public addService: Utils_vdm`String * Service ==> ()
  addService(uName, serv) == let type = serv.type in
  services := services ++ {type |-> serv}
  pre uName in set dom users and uName = adminName and
   serv not in set rng services
  post card dom services = card dom services~ + 1 and serv in set rng services;

--Remove service from services map

 public removeService: Utils_vdm`String * Service`ServiceType ==> ()
  removeService(uName, service) == (
   for all e in set rng events do (if(service in set elems e.services) then e.removeService(
       service));
   services := {service} <-: services
  )
  pre uName in set dom users and uName = adminName and
   service in set dom services
  post service not in set dom services and
   forall e in set rng events & service not in set elems e.services;

-- Changes a service price

 public changeServicePrice: Utils_vdm`String * Service`ServiceType * real ==> ()
 changeServicePrice(uName, serviceType, price) == let service = services(serviceType) in
 service.setPrice(price)
 pre uName in set dom users and uName = adminName and
  price > 0 and serviceType in set dom services
 post services(serviceType).price = price;

/** EVENT OPERATIONS **/
--Create an event and add it to the events map

 public createEvent:  Utils_vdm`String * nat * real * Utils_vdm`Date * Utils_vdm`Date * bool *
     Event`EventType * Installation * Utils_vdm`String ==> Event
 createEvent(n, tTickets, tPrice, b, e, p, t, inst, h) == (
  dcl event: Event := new Event(n, tTickets, tPrice, b, e, p, t, inst, h);
  users(h).addEvent(n);
  events := events ++ {n |-> event};
  return event
```

```
 )
 pre h in set dom users and n not in set dom events and
  (Utils_vdm`before(b, e) or b = e) and
  forall ev in set rng Utils_vdm`fMap[Utils_vdm`String, bool, Event](lambda x: Event &
      hasDatesConflict(x, b, e))(Utils_vdm`setTOseq[Utils_vdm`String](dom events), events) &
   ev.installation <> inst
 post n in set dom events;

--Show event information

 public showEventDetails: Utils_vdm`String * Utils_vdm`String ==> map Utils_vdm`String to
     Utils_vdm`String
 showEventDetails(uName, evName) == return events(evName).showDetails()
 pre uName in set dom users and
 (if(events(evName).privacy) then (uName = adminName or uName in set
  events(evName).guests union events(evName).staff union {events(evName).host})
  else true);

--Show the name of all events, but not the private events if uName is not a guest, staff or host

 public listEvents: Utils_vdm`String ==> set of Event
 listEvents(uName) == (
  dcl res: set of Event := {};
  for all event in set rng events do (
   if(event.privacy and uName in set (event.guests union {event.host, adminName} union event.
       staff))
    then res := res union {event}
   elseif(not event.privacy) then res := res union {event}
  );
  return res
 )
 pre uName in set dom users
 post RESULT subset rng events and forall e in set RESULT &
  not (e.privacy and uName not in set (e.guests union {e.host} union e.staff union {adminName}));

--Return the services used in one event

 public listEventServices: Utils_vdm`String ==> set of Service
 listEventServices(evName) == return rng (elems events(evName).services <: services)
 pre evName in set dom events
 post forall s in set RESULT & s.type in set dom services and s.type in set elems events(evName).
     services;

--Return the services available to add to the event

 public availableServicesForEvent: Utils_vdm`String ==> set of Service
 availableServicesForEvent(evName) == return rng (elems events(evName).services <-: services)
 pre evName in set dom events
 post forall s in set RESULT & s.type in set dom services and s.type not in set elems events(
     evName).services;

--Return how much an event spent in services

 public moneySpentWithServices: Utils_vdm`String * Utils_vdm`String ==> real
 moneySpentWithServices(uName, evName) == (
  dcl ev: Event := events(evName);
  dcl days: int := Utils_vdm`getDatesDifference(events(evName).begin, events(evName).ending) + 1;
  dcl res: real := 0;
  for all service in set elems ev.services do res := res + services(service).price * days;
  return res
 )
 pre uName in set dom users and (uName = events(evName).host or uName = adminName) and
  evName in set dom events;

--Return how much an event spent with installation rent
```

```
 public moneySpentWithInstallation: Utils_vdm`String * Utils_vdm`String ==> real
 moneySpentWithInstallation(uName, evName) == (
  dcl days: int := Utils_vdm`getDatesDifference(events(evName).begin, events(evName).ending) + 1;
  return days * events(evName).installation.price
 )
 pre uName in set dom users and (uName = events(evName).host or uName = adminName) and
  evName in set dom events;

--In case of a private event, host can invite users (so they will be
-- the only ones capable of seeing and buying a ticket to the event)

 public inviteToEvent: Utils_vdm`String * Utils_vdm`String * Utils_vdm`String ==> ()
 inviteToEvent(evName, hName, uName) == events(evName).inviteUser(hName, uName)
 pre evName in set dom events and uName in set dom users and
  hName in set dom users and hName <> uName and
  events(evName).privacy = true
 post uName in set events(evName).guests;

--Shows all the available events

 public showAvailableEvents: Utils_vdm`String ==> [set of Event]
 showAvailableEvents(uName) == (
  dcl res: set of Event := {};
  for all e in set rng events do (
   if(card e.attendees = e.totalTickets) then skip else (
    if(e.privacy and uName in set e.guests union e.staff union {e.host, adminName}) then res :=
        res union {e};
    if(not e.privacy) then res := res union {e}
   )
  );
  return res
 )
 pre uName in set dom users
 post RESULT subset rng events;

--Shows all the available events in a period

 public showAvailableEventsBetweenDates: Utils_vdm`String * Utils_vdm`Date * Utils_vdm`Date ==> [
     set of Event]
 showAvailableEventsBetweenDates(uName, b, e) == (
  dcl res: set of Event := {};
  for all ev in set rng events do
   if(hasDatesConflict(ev, b, e)) then (
    if(card ev.attendees = ev.totalTickets) then skip else (
     if(ev.privacy and uName in set ev.guests union ev.staff union {ev.host, adminName}) then res
         := res union {ev};
     if(not ev.privacy) then res := res union {ev}
    )
   );
  return res
 )
 pre uName in set dom users and (Utils_vdm`before(b, e) or b = e)
 post RESULT subset rng events and
  forall event in set RESULT & hasDatesConflict(event, b, e);

/** OPERATIONS TO CHANGE EVENTS ATTRIBUTES **/
--Change event name

 public changeEventName: Utils_vdm`String * Utils_vdm`String * Utils_vdm`String ==> ()
 changeEventName(evName, hName, value) == (
  dcl ev: Event := events(evName);
  ev.setName(value);
  for all user in set (ev.attendees union ev.staff union {ev.host}) do (
   users(user).events := (users(user).events \ {evName}) union {value}
```

```
  );
  events := ({evName} <-: events) ++ {value |-> ev}
 )
 pre evName in set dom events and value not in set dom events and
  hName in set dom users and events(evName).host = hName
 post evName not in set dom events and value in set dom events;

--Change event number of tickets

 public changeEventTotalTickets: Utils_vdm`String * Utils_vdm`String * nat ==> ()
 changeEventTotalTickets(evName, hName, value) == (
  dcl ev: Event := events(evName);
  ev.setTotalTickets(value)
  --events := events ++ {evName |-> ev} Se ev for uma referencia, nao e preciso atualizar
 )
 pre evName in set dom events and value > 0 and
  hName in set dom users and events(evName).host = hName
 post events(evName).totalTickets = value;

--Change event ticket price

 public changeEventTicketPrice: Utils_vdm`String * Utils_vdm`String * real ==> ()
 changeEventTicketPrice(evName, hName, value) == (
  dcl ev: Event := events(evName);
  ev.setTicketPrice(value)
  --events := events ++ {evName |-> ev}
 )
 pre evName in set dom events and value >= 0 and
  hName in set dom users and events(evName).host = hName
 post events(evName).ticketPrice = value;

--Change event date (begin or ending)

 public changeEventDate: Utils_vdm`String * Utils_vdm`String * DateType * Utils_vdm`Date ==> ()
 changeEventDate(evName, hName, dateType, value) == (
  dcl ev: Event := events(evName);
  cases dateType:
   <begin> -> ev.setBeginDate(value),
   <ending> -> ev.setEndingDate(value)
   --others -> error
   end
 )
 pre evName in set dom events and (dateType = <begin> or dateType = <ending>) and
  hName in set dom users and events(evName).host = hName and
  if(dateType = <begin>) then (forall e in set rng events \ {events(evName)} &
   not ( e.installation = events(evName).installation and hasDatesConflict(e, value, events(
       evName).ending)))
  else (forall e in set rng events \ {events(evName)} &
   not (e.installation = events(evName).installation and hasDatesConflict(e, events(evName).begin
       , value)))
 post if(dateType = <begin>) then events(evName).begin = value
  else events(evName).ending = value;

--Change event privacy condition

 public changeEventPrivacy: Utils_vdm`String * Utils_vdm`String * bool ==> ()
 changeEventPrivacy(evName, hName, value) == (
  dcl ev: Event := events(evName);
  ev.setPrivacy(value)
 )
 pre evName in set dom events and
  hName in set dom users and events(evName).host = hName
 post events(evName).privacy = value;

--Change event type
```

```
 public changeEventType: Utils_vdm`String * Utils_vdm`String * Event`EventType ==> ()
 changeEventType(evName, hName, value) == (
  dcl ev: Event := events(evName);
  ev.setType(value)
 )
 pre evName in set dom events and
  hName in set dom users and events(evName).host = hName
 post events(evName).type = value;

--Change where event will occur

 public changeEventInstallation: Utils_vdm`String * Utils_vdm`String * Utils_vdm`String ==> ()
 changeEventInstallation(evName, hName, instName) == (
  dcl ev: Event := events(evName);
  ev.changeInstallation(installations(instName))
 )
 pre evName in set dom events and
  let b = events(evName).begin, e = events(evName).ending in (
   forall ev in set rng Utils_vdm`fMap[Utils_vdm`String, bool, Event](lambda x: Event &
       hasDatesConflict(x, b, e))(Utils_vdm`setTOseq[Utils_vdm`String](dom events), events) &
   instName <> ev.installation.id
  ) and
  hName in set dom users and events(evName).host = hName
 post events(evName).installation.id = instName;

--Add service to event

 public addServiceToEvent: Utils_vdm`String * Utils_vdm`String * Service`ServiceType ==> ()
 addServiceToEvent(evName, hName, servType) == events(evName).addService(servType)
 pre evName in set dom events and servType in set dom services and
  hName in set dom users and events(evName).host = hName
 post servType in set elems events(evName).services;

--Remove service from event

 public removeServiceFromEvent: Utils_vdm`String * Utils_vdm`String * Service`ServiceType ==> ()
 removeServiceFromEvent(evName, hName, servType) == events(evName).removeService(servType)
 pre evName in set dom events and servType in set dom services and
  servType in set elems events(evName).services and
  hName in set dom users and events(evName).host = hName
 post servType not in set elems events(evName).services;

/** USER FUNCTIONS **/
--Show one user information

 public showUserDetails: Utils_vdm`String * Utils_vdm`String ==> map Utils_vdm`String to
     Utils_vdm`String
 showUserDetails(uName1, uName2) == (
  dcl res: map Utils_vdm`String to Utils_vdm`String := users(uName2).showDetails();
  dcl money: real := 0;
  for all evName in set users(uName2).events do if(uName2 in set events(evName).attendees) then
      money := money + events(evName).ticketPrice;
  res := res ++ {"Money spent"|->Utils_vdm`toStringVDM(money)};
  return res
 )
 pre uName1 in set dom users and uName2 in set dom users and
  (uName1 = adminName or uName1 = uName2)
 post  card dom RESULT = 3;

--Show all users information

 public showUsersDetails: Utils_vdm`String ==> set of (map Utils_vdm`String to Utils_vdm`String)
 showUsersDetails(uName) == (
  dcl res: set of (map Utils_vdm`String to Utils_vdm`String) := {};
```

```
  for all user in set dom users do res := res union {showUserDetails(uName, user)};
  return res
 )
 pre uName in set dom users and uName = adminName
 post  card RESULT = card dom users;

--Add user to our center

 public addUser: User ==> ()
 addUser(user) == users := users ++ {user.name |-> user}
 pre user.name not in set dom users
 post user in set rng users;

--Add user to event

 public addUserToEvent: Utils_vdm`String * Utils_vdm`String * UserPosition ==> ()
 addUserToEvent(evName, uName, uPos) == (
  dcl ev: Event := events(evName);
  dcl user: User := users(uName);
  cases uPos:
   <attendee> -> (ev.addAttendee(uName); user.addEvent(evName)),
   <staff> -> (ev.addStaff(uName); user.addEvent(evName)),
   <host> -> (users(ev.host).removeEvent(evName); ev.setHost(uName); user.addEvent(evName))
  end
 )
 pre evName in set dom events and uName in set dom users and
  uPos in set {<attendee>, <staff>, <host>}
 post evName in set users(uName).events;
 --Remove user from event

 public removeUserFromEvent: Utils_vdm`String * Utils_vdm`String * UserPosition ==> ()
 removeUserFromEvent(evName, uName, uPos) == (
  dcl ev: Event := events(evName);
  dcl user: User := users(uName);
  cases uPos:
   <attendee> -> (ev.removeAttendee(uName); user.removeEvent(evName)),
   <staff> -> (ev.removeStaff(uName); user.removeEvent(evName))
  end
 )
 pre evName in set dom events and uName in set dom users and
  uPos in set {<attendee>, <staff>}
 post evName not in set users(uName).events;

functions
-- Verify if an installation has a specific installation (room or foyer)

 public hasInstallation: Installation * Installation -> bool
 hasInstallation(inst1, inst2) == (
  if (isofclass(Auditorium, inst2)) then (
   let inst = narrow_(inst2, Auditorium) in
   if(isofclass(Foyer, inst1)) then let foyer = narrow_(inst1, Foyer) in foyer in set inst.foyers
   else false)
   elseif(isofclass(Pavilion, inst2)) then (
   let inst = narrow_(inst2, Pavilion) in
   if(isofclass(Foyer, inst1)) then let foyer = narrow_(inst1, Foyer) in foyer in set inst.foyers
   else(if(isofclass(Room, inst1)) then (let room = narrow_(inst1, Room) in room in set inst.
       rooms)else false)
  ) else (
   false )
 )
 pre true;--(isofclass(Foyer, inst1) or isofclass(Room, inst1)) and
 --(isofclass(Auditorium, inst2) or isofclass(Pavilion, inst2));

--Checks if this event has conflicts with the period given in the arguments
```

```
 public hasDatesConflict: Event * Utils_vdm`Date * Utils_vdm`Date -> bool
 hasDatesConflict(ev, b2, e2) == (
  if(Utils_vdm`before(b2, ev.begin)) then (Utils_vdm`before(ev.begin, e2) or ev.begin = e2)
  else (Utils_vdm`before(b2, ev.ending) or b2 = ev.ending or b2 = ev.begin)
 )
 pre (Utils_vdm`before(ev.begin, ev.ending) or ev.begin = ev.ending)
  and (Utils_vdm`before(b2, e2) or b2 = e2);

end Center
```

Function or operation	Line	Coverage	Calls
Center	20	100.0%	6
addInstallationToInstallation	52	100.0%	30
addInstallations	44	100.0%	54
addService	170	100.0%	24
addServiceToEvent	399	100.0%	24
addUser	438	100.0%	30
addUserToEvent	444	100.0%	18
associatedInstallations	152	100.0%	18
availableServicesForEvent	241	100.0%	12
changeEventDate	346	100.0%	5
changeEventInstallation	385	100.0%	6
changeEventName	310	100.0%	6
changeEventPrivacy	365	100.0%	12
changeEventTicketPrice	335	100.0%	6
changeEventTotalTickets	324	100.0%	11
changeEventType	375	100.0%	6
changeInstallationConditions	121	100.0%	30
changeInstallationMeasures	109	100.0%	30
changeInstallationRent	115	100.0%	30
changeServicePrice	189	100.0%	12
createEvent	198	100.0%	12
getAvailableInstallations	127	100.0%	1260
hasDatesConflict	490	100.0%	812
hasInstallation	473	100.0%	432
inviteToEvent	269	100.0%	8
listEventServices	235	100.0%	12
listEvents	220	100.0%	60
moneySpentWithInstallation	259	100.0%	12
moneySpentWithServices	247	100.0%	18
removeInstallation	94	100.0%	24
removeInstallationFromInstallation	75	100.0%	24
removeService	178	100.0%	12
removeServiceFromEvent	406	100.0%	6
removeUserFromEvent	458	100.0%	6
showAvailableEvents	277	100.0%	5
showAvailableEventsBetweenDates	292	100.0%	23
showEventDetails	212	100.0%	16
showInstallationDetails	30	100.0%	60

showInstallationsDetails	35	100.0%	6
showServicesDetails	161	100.0%	6
showUserDetails	415	100.0%	35
showUsersDetails	428	100.0%	6
Center.vdmpp		100.0%	3225

```
class Event
types
 public EventType=<Conference> | <TradeFair> | <Party> | <Musical> | <TeamBuilding>;
instance variables
 public name: Utils_vdm`String := "untitled";
 public totalTickets: nat := 1;
 inv totalTickets > 0;
 public ticketPrice: real := 0;
 public begin: Utils_vdm`Date := mk_Utils_vdm`Date(2018, 1, 1);
 public ending: Utils_vdm`Date := mk_Utils_vdm`Date(2018, 1, 2);
 public privacy: bool := false;
 public type: [EventType] := nil;
 inv begin = ending or Utils_vdm`before(begin, ending);
 public installation: Installation;
 inv installation <> undefined;
 public services: [seq of Service`ServiceType] := [];
 public attendees: [set of Utils_vdm`String] := {};
 inv card attendees <= totalTickets;
 public staff: [set of Utils_vdm`String] := {};
 public host: Utils_vdm`String := "undefined";
 inv host not in set attendees and host not in set staff and host <> "undefined";
 inv forall attendee in set attendees & attendee not in set staff;
 public guests: [set of Utils_vdm`String] := {};
 inv if(not privacy) then guests = {} else true;
operations

--Constructor

 public Event: Utils_vdm`String * nat * real * Utils_vdm`Date * Utils_vdm`Date * bool * EventType
      * Installation * Utils_vdm`String ==> Event
 Event(n, tTickets, tPrice, b, e, p, t, inst, h) == (
  atomic(installation := inst; host := h);
  name := n;
  totalTickets := tTickets;
  ticketPrice := tPrice;
  atomic(begin := b; ending := e);
  privacy := p;
  type := t;
 )
 pre tTickets > 0 and (b = e or Utils_vdm`before(b, e));

--Set event name

 public setName: Utils_vdm`String ==> ()
 setName(n) == name := n
 post name = n;
--Set event number of tickets

 public setTotalTickets: nat ==> ()
 setTotalTickets(tTickets) == totalTickets := tTickets
 pre tTickets > 0 and tTickets >= card attendees
 post totalTickets = tTickets;
--Set event price of tickets

 public setTicketPrice: nat ==> ()
```

```
 setTicketPrice(tPrice) == ticketPrice := tPrice
 pre tPrice >= 0
 post ticketPrice = tPrice;
--Set event begin date

 public setBeginDate: Utils_vdm`Date ==> ()
 setBeginDate(b) == begin := b
 pre Utils_vdm`before(b, ending)
 post begin = b;
--Set event ending date

 public setEndingDate: Utils_vdm`Date ==> ()
 setEndingDate(e) == ending := e
 pre Utils_vdm`before(begin, e)
 post ending = e;
--Set event privacy condition

 public setPrivacy: bool ==> ()
 setPrivacy(p) == (
  if(p) then guests := attendees
   else guests := {};
  privacy := p
 )
 post privacy = p;
--Set event type

 public setType: EventType ==> ()
 setType(t) == type := t
 pre t in set {<Conferences>, <TradeFair>, <Party>, <Musical>, <TeamBuilding>}
 post type = t;

--Shows all event information

 public showDetails: () ==> map Utils_vdm`String to Utils_vdm`String
 showDetails() == (
  dcl res: map Utils_vdm`String to Utils_vdm`String := {|->};
  res := res ++ {
    "Name" |-> name, "Total number of tickets" |-> Utils_vdm`toStringVDM(totalTickets), "Sold
        tickets" |-> Utils_vdm`toStringVDM(card attendees),
    "Ticket's price" |-> Utils_vdm`toStringVDM(ticketPrice), "Is private" |-> Utils_vdm`
        toStringVDM(privacy),
    "Starting date" |-> Utils_vdm`toStringVDM(begin), "Ending date" |-> Utils_vdm`toStringVDM(
        ending),
    "Installation" |-> installation.id,
    "Event type" |-> typetoStringVDM(type),
    "Services" |-> servicestoStringVDM(),
    "Attendees" |-> usersSettoStringVDM(attendees),
    "Staff" |-> usersSettoStringVDM(staff),
    "Host" |-> host
  };
  if(privacy) then res := res ++ {"Guests" |-> usersSettoStringVDM(guests)};
  return res
 );

--Changes the installation where the event will occur

 public changeInstallation: Installation ==> ()
 changeInstallation(inst) == installation := inst
 pre inst <> installation
 post installation = inst;

--Add one service to services sequence

 public addService: Service`ServiceType ==> ()
 addService(service) == services := services ^ [service]
```

```
 pre service not in set elems services
 post len services = len services~ + 1 and service in set elems services;

--Remove one service from services sequence

 public removeService: Service`ServiceType ==> ()
 removeService(service) == (
  dcl tmpServices: seq of Service`ServiceType := services;
  services := [];
  while(tmpServices <> []) do (
   if(hd tmpServices <> service) then services := services ^ [hd tmpServices];
   tmpServices := tl tmpServices
  )
 )
 pre service in set elems services
 post service not in set elems services;

--Returns how much did the event earned

 public earnedMoney: () ==> real
 earnedMoney() == return card attendees * ticketPrice
 post RESULT = card attendees * ticketPrice;

--Return the number of tickets that are left

 public remainingTickets: () ==> nat
 remainingTickets() == return totalTickets - card attendees
 post RESULT = totalTickets - card attendees;

--Change event's host

 public setHost: Utils_vdm`String ==> ()
 setHost(user) == (
  if(user in set attendees) then attendees := attendees \ {user}
  elseif(user in set staff) then staff := staff \ {user};
  host := user
 )
 post host = user;

--Add attendee to the event

 public addAttendee: Utils_vdm`String ==> ()
 addAttendee(user) == attendees := attendees union {user}
 pre user not in set attendees and card attendees < totalTickets and
  user not in set staff and user <> host and (not privacy or user in set guests)
 post user in set attendees;

--Add user to staff set

 public addStaff: Utils_vdm`String ==> ()
 addStaff(user) == staff := staff union {user}
 pre user not in set staff and
  user not in set attendees and user <> host
 post user in set staff;

--Remove attendee

 public removeAttendee: Utils_vdm`String ==> ()
  removeAttendee(user) == attendees := attendees \ {user}
  pre user in set attendees
  post user not in set attendees;

--Remove user from staff set

 public removeStaff: Utils_vdm`String ==> ()
```

```
  removeStaff(user) == staff := staff \ {user}
  pre user in set staff
  post user not in set staff;

--Invite user to private event

 public inviteUser: Utils_vdm`String * Utils_vdm`String ==> ()
 inviteUser(h, user) == guests := guests union {user}
 pre h = host and user <> host and
  privacy and user not in set staff and
  card guests < totalTickets
 post user in set guests;

--Converts service set to string

 public servicestoStringVDM: () ==> Utils_vdm`String
 servicestoStringVDM() == (
  dcl res: Utils_vdm`String := "";
  if(len services = 0) then return "";
  for all service in set elems services do res := res ^ Service`typetoStringVDM(service) ^ ", ";
  res := res(1, ..., len res - 2);
  return res
 );

--Converts set of users to string

 public usersSettoStringVDM: [set of Utils_vdm`String] ==> Utils_vdm`String
 usersSettoStringVDM(users) == (
  dcl res: Utils_vdm`String := "";
  if(card users = 0) then return "";
  for all user in set users do res := res ^ user ^ ", ";
  res := res(1, ..., len res - 2);
  return res
 );

functions

--Converts event type to string

 public typetoStringVDM: EventType -> Utils_vdm`String
 typetoStringVDM(t) == (
  cases t:
   <Conference> -> "Conference",
   <TradeFair> -> "Trade Fair",
   <Party> -> "Party",
   <Musical> -> "Musical",
   <TeamBuilding> -> "Team Building"
  end
 )
 pre t in set {<Conference>, <TradeFair>, <Party>, <Musical>, <TeamBuilding>};

end Event
```

Function or operation	Line	Coverage	Calls
Event	28	100.0%	30
addAttendee	142	100.0%	30
addService	104	100.0%	24
addStaff	149	100.0%	18
changeInstallation	98	100.0%	12

earnedMoney	123	100.0%	5
inviteUser	168	100.0%	8
remainingTickets	128	100.0%	5
removeAttendee	156	100.0%	6
removeService	110	100.0%	12
removeStaff	162	100.0%	6
servicestoStringVDM	176	100.0%	6
setBeginDate	55	100.0%	10
setEndingDate	60	100.0%	10
setHost	133	100.0%	6
setName	41	100.0%	6
setPrivacy	65	100.0%	12
setTicketPrice	50	100.0%	6
setTotalTickets	45	100.0%	11
setType	73	100.0%	6
showDetails	79	100.0%	16
typetoStringVDM	198	100.0%	31
usersSettoStringVDM	186	100.0%	12
Event.vdmpp		100.0%	288

```
class Foyer is subclass of Installation
operations
-- Constructor (must have blackOutCurtains, telephones, soundproofWalls and movingWalls always
    false)

 public Foyer: Utils_vdm`String * real * nat * real * real * real * bool * bool * bool * bool ==>
      Foyer
  Foyer(i, pr, c, h, w, l, airC, natL, ceilL, compN) == (
   --Measures
   id := i;
   price := pr;
   capacity := c;
   heigth := h;
   atomic(width := w; lenght := l; area := w * l);
   --Conditions
   airCondition := airC;
   naturalLigth := natL;
   ceilingLighting := ceilL;
   computerNetwork := compN;
  )
  pre c > 0 and h > 0 and w > 0 and l > 0
  post not (blackOutCurtains or telephones or soundproofWalls or movingWalls);
--Impossible to create a installation with out parameters
 public Foyer: () ==> Foyer
  Foyer() == return self
  pre false;

--Change installation's conditions

 public setConditions: bool * bool * bool * bool * bool * bool * bool * bool ==> ()
  setConditions(airC, natL, ceilL, -, -, compN, -, -) == (
   airCondition := airC;
   naturalLigth := natL;
   ceilingLighting := ceilL;
   computerNetwork := compN
  )
  post not (blackOutCurtains or telephones or soundproofWalls or movingWalls);
```

```
--Shows all installation information

 public showDetails: () ==> map Utils_vdm`String to Utils_vdm`String
 showDetails() == (
  dcl res: map Utils_vdm`String to Utils_vdm`String := {|->};
  res := res ++ {
    "ID" |-> id, "Price" |-> Utils_vdm`toStringVDM(price), "Capacity" |-> Utils_vdm`toStringVDM(
        capacity),
    "Area" |-> Utils_vdm`toStringVDM(area), "Heigth" |-> Utils_vdm`toStringVDM(heigth),
    "Width" |-> Utils_vdm`toStringVDM(width), "Lenght" |-> Utils_vdm`toStringVDM(lenght),
    "Air Condition" |-> Utils_vdm`toStringVDM(airCondition),
    "Natural Ligth" |-> Utils_vdm`toStringVDM(naturalLigth),
    "Ceiling Lighting" |-> Utils_vdm`toStringVDM(ceilingLighting),
    "Computer Network" |-> Utils_vdm`toStringVDM(computerNetwork)
  };
  return res
 );


 public addFoyer : Foyer ==> ()
  addFoyer(-) == return
  pre false;

 public removeFoyer : Foyer ==> ()
  removeFoyer(-) == return
  pre false;

 public addRoom : Room ==> ()
  addRoom(-) == return
  pre false;

 public removeRoom : Room ==> ()
  removeRoom(-) == return
  pre false;
--Verify if has installation

 public hasInstallation: Installation ==> bool
  hasInstallation(-) == return false;
end Foyer
```

Function or operation	Line	Coverage	Calls
Foyer	4	0.0%	0
addFoyer	51	0.0%	0
addRoom	57	0.0%	0
hasInstallation	64	100.0%	468
removeFoyer	54	0.0%	0
removeRoom	60	0.0%	0
setConditions	26	100.0%	6
showDetails	36	100.0%	12
Foyer.vdmpp		91.4%	486

```
class Installation
instance variables
-- Measures
 public id: Utils_vdm`String := "untitled";
```

```
 public price: real := 0;
 public capacity: nat := 0;
 public area: real := 0;
 public heigth: real := 0;
 public width: real := 0;
 public lenght: real := 0;
 inv area = width * lenght;
-- Conditions
 public airCondition: bool := false;
 public naturalLigth: bool := false;
 public ceilingLighting: bool := false;
 public blackOutCurtains: bool := false;
 public telephones: bool := false;
 public computerNetwork: bool := false;
 public soundproofWalls: bool := false;
 public movingWalls: bool := false;
operations
-- Return installation id

 public getID: () ==> Utils_vdm`String
  getID() == return id;


 public addFoyer : Foyer ==> ()
  addFoyer(f) == is subclass responsibility;

 public removeFoyer : Foyer ==> ()
  removeFoyer(f) == is subclass responsibility;

 public addRoom : Room ==> ()
  addRoom(r) == is subclass responsibility;

 public removeRoom : Room ==> ()
  removeRoom(r) == is subclass responsibility;

 public hasInstallation: Installation ==> bool
  hasInstallation(inst) == is subclass responsibility;

 public setConditions: bool * bool * bool * bool * bool * bool * bool * bool ==> ()
  setConditions(airC, natL, ceilL, blackC, tele, compN, soundW, movW) ==
   is subclass responsibility;

 public showDetails: () ==> map Utils_vdm`String to Utils_vdm`String
  showDetails() == is subclass responsibility;
--Change Installation price per day

 public setPrice: real ==> ()
  setPrice(pr) == price := pr
  pre pr > 0
  post price = pr;
 --Change installation measures

 public setMeasures: nat * real * real * real ==> ()
   setMeasures(c, h, w, l) == (
    atomic(
     capacity := c;
     heigth := h;
     width := w;
     lenght := l;
     area := w * l;
    )
   )
   pre c > 0 and h > 0 and w > 0 and l > 0
   post capacity = c and heigth = h and width = w and lenght = l;
end Installation
```

Function or operation	Line	Coverage	Calls
addFoyer	26	100.0%	7
addRoom	30	100.0%	7
getID	23	100.0%	60
hasInstallation	34	100.0%	7
removeFoyer	28	100.0%	7
removeRoom	32	100.0%	7
setConditions	36	100.0%	7
setMeasures	47	100.0%	30
setPrice	42	100.0%	30
showDetails	39	100.0%	7
Installation.vdmpp		100.0%	169

```
class Pavilion is subclass of Installation
instance variables
 public rooms: set of Room := {};
 public foyers: set of Foyer := {};
operations
-- Constructor (must have telephones, soundproofWalls and movingWalls always false)

 public Pavilion: Utils_vdm`String * real * nat * real * real * real * bool * bool * bool * bool
     * bool ==> Pavilion
  Pavilion(i, pr, c, h, w, l, airC, natL, ceilL, blackC, compN) == (
   --Measures
   id := i;
   price := pr;
   capacity := c;
   heigth := h;
   atomic(width := w; lenght := l; area := w * l);
   --Conditions
   airCondition := airC;
   naturalLigth := natL;
   ceilingLighting := ceilL;
   blackOutCurtains := blackC;
   computerNetwork := compN;
  )
  pre c > 0 and h > 0 and w > 0 and l > 0
  post not (telephones or soundproofWalls or movingWalls);
--Impossible to create a installation with out parameters
  public Pavilion: () ==> Pavilion
  Pavilion() == return self
  pre false;

--Change installation's conditions

 public setConditions: bool * bool * bool * bool * bool * bool * bool * bool ==> ()
  setConditions(airC, natL, ceilL, blackC, -, compN, -, -) == (
   airCondition := airC;
   naturalLigth := natL;
   ceilingLighting := ceilL;
   blackOutCurtains := blackC;
   computerNetwork := compN
  )
  post not (telephones or soundproofWalls or movingWalls);
```

```
--Shows all installation information

 public showDetails: () ==> map Utils_vdm`String to Utils_vdm`String
 showDetails() == (
  dcl res: map Utils_vdm`String to Utils_vdm`String := {|->};
  res := res ++ {
    "ID" |-> id, "Price" |-> Utils_vdm`toStringVDM(price), "Capacity" |-> Utils_vdm`toStringVDM(
        capacity),
    "Area" |-> Utils_vdm`toStringVDM(area), "Heigth" |-> Utils_vdm`toStringVDM(heigth),
    "Width" |-> Utils_vdm`toStringVDM(width), "Lenght" |-> Utils_vdm`toStringVDM(lenght),
    "Air Condition" |-> Utils_vdm`toStringVDM(airCondition),
    "Natural Ligth" |-> Utils_vdm`toStringVDM(naturalLigth),
    "Ceiling Lighting" |-> Utils_vdm`toStringVDM(ceilingLighting),
    "Black Out Curtains" |-> Utils_vdm`toStringVDM(blackOutCurtains),
    "Computer Network" |-> Utils_vdm`toStringVDM(computerNetwork)
  };
  return res
 );

-- Add room to rooms set

 public addRoom : Room ==> ()
  addRoom(r) == rooms := rooms union {r}
  pre r not in set rooms and r.area <= (area - sumElems(Utils_vdm`setTOseq[real]({x.area| x in
      set rooms})))
  post card rooms = card rooms~ + 1 and r in set rooms;

-- Add foyer to foyers set

 public addFoyer : Foyer ==> ()
  addFoyer(f) == foyers := foyers union {f}
  pre f not in set foyers
  post card foyers = card foyers~ + 1 and f in set foyers;

-- Remove room from rooms set

 public removeRoom : Room ==> ()
  removeRoom(r) == rooms := rooms \ {r}
  pre r in set rooms
  post card rooms = card rooms~ - 1 and r not in set rooms;

-- Remove foyer from foyers set

 public removeFoyer : Foyer ==> ()
  removeFoyer(f) == foyers := foyers \ {f}
  pre f in set foyers
  post card foyers = card foyers~ - 1 and f not in set foyers;

--Verify if has installation

 public hasInstallation: Installation ==> bool
  hasInstallation(inst) == (
   if(isofclass(Foyer, inst)) then return (let f = narrow_(inst, Foyer) in f in set foyers)
   elseif(isofclass(Room, inst)) then return (let r = narrow_(inst, Room) in r in set rooms)
   else return false
  );

functions

--sum the elements of a seq

 public sumElems: seq of real -> real
 sumElems(list) == (
  if(len list = 0) then 0 else (
  if(len list = 1) then hd list
```

```
  else hd list + sumElems(tl list)
 ))
 measure len list;

end Pavilion
```

Function or operation	Line	Coverage	Calls
Pavilion	7	0.0%	0
addFoyer	63	100.0%	24
addRoom	57	100.0%	6
hasInstallation	81	100.0%	78
removeFoyer	75	100.0%	12
removeRoom	69	100.0%	6
setConditions	30	100.0%	6
showDetails	40	100.0%	12
sumElems	91	100.0%	22
Pavilion.vdmpp		98.8%	166

```
class Room is subclass of Installation

operations
-- Constructor

 public Room: Utils_vdm`String * real * nat * real * real * real * bool * bool * bool * bool *
     bool * bool * bool * bool ==> Room
  Room(i, pr, c, h, w, l, airC, natL, ceilL, blackC, tele, compN, soundP, movW) == (
   --Measures
   id := i;
   price := pr;
   capacity := c;
   heigth := h;
   atomic(width := w; lenght := l; area := w * l);
   --Conditions
   airCondition := airC;
   naturalLigth := natL;
   ceilingLighting := ceilL;
   blackOutCurtains := blackC;
   telephones := tele;
   computerNetwork := compN;
   soundproofWalls := soundP;
   movingWalls := movW;
  )
  pre c > 0 and h > 0 and w > 0 and l > 0;
--Impossible to create a installation with out parameters
 public Room: () ==> Room
  Room() == return self
  pre false;

--Change installation's conditions

 public setConditions: bool * bool * bool * bool * bool * bool * bool * bool ==> ()
  setConditions(airC, natL, ceilL, blackC, tele, compN, soundW, movW) == (
   airCondition := airC;
   naturalLigth := natL;
   ceilingLighting := ceilL;
   blackOutCurtains := blackC;
```

```
   telephones := tele;
   computerNetwork := compN;
   soundproofWalls := soundW;
   movingWalls := movW
  );

--Shows all installation information

 public showDetails: () ==> map Utils_vdm`String to Utils_vdm`String
 showDetails() == (
  dcl res: map Utils_vdm`String to Utils_vdm`String := {|->};
  res := res ++ {
    "ID" |-> id, "Price" |-> Utils_vdm`toStringVDM(price), "Capacity" |-> Utils_vdm`toStringVDM(
        capacity),
    "Area" |-> Utils_vdm`toStringVDM(area), "Heigth" |-> Utils_vdm`toStringVDM(heigth),
    "Width" |-> Utils_vdm`toStringVDM(width), "Lenght" |-> Utils_vdm`toStringVDM(lenght),
    "Air Condition" |-> Utils_vdm`toStringVDM(airCondition),
    "Natural Ligth" |-> Utils_vdm`toStringVDM(naturalLigth),
    "Ceiling Lighting" |-> Utils_vdm`toStringVDM(ceilingLighting),
    "Black Out Curtains" |-> Utils_vdm`toStringVDM(blackOutCurtains),
    "Telephones" |-> Utils_vdm`toStringVDM(telephones),
    "Computer Network" |-> Utils_vdm`toStringVDM(computerNetwork),
    "Soundproof Walls" |-> Utils_vdm`toStringVDM(soundproofWalls),
    "Moving Walls" |-> Utils_vdm`toStringVDM(movingWalls)
  };
  return res
 );

 public addFoyer : Foyer ==> ()
  addFoyer(-) == return
  pre false;

 public removeFoyer : Foyer ==> ()
  removeFoyer(-) == return
  pre false;

 public addRoom : Room ==> ()
  addRoom(-) == return
  pre false;

 public removeRoom : Room ==> ()
  removeRoom(-) == return
  pre false;
--Verify if has installation

 public hasInstallation: Installation ==> bool
  hasInstallation(-) == return false;
end Room
```

Function or operation	Line	Coverage	Calls
Room	5	0.0%	0
addFoyer	61	0.0%	0
addRoom	67	0.0%	0
hasInstallation	74	100.0%	126
removeFoyer	64	0.0%	0
removeRoom	70	0.0%	0
setConditions	30	100.0%	6
showDetails	43	100.0%	6

Room.vdmpp		92.4%	138

```
class Service
types
 public ServiceType = <AudioVisual> | <Catering> | <IT> | <Cleaning> | <Security> | <Decoration>;
values

instance variables
 public price: real;
 inv price > 0;
 public type: ServiceType;
operations
--Constructor

 public Service: real * ServiceType ==> Service
  Service(p, t) == (
   price := p;
   type := t;
  )
  pre p > 0
  post price = p and type <> nil;

-- Changes service price

 public setPrice: real ==> ()
 setPrice(p) ==
  price := p
 pre p > 0
 post price = p;

--Shows all service information

 public showDetails: () ==> map Utils_vdm`String to Utils_vdm`String
 showDetails() == (
  dcl res: map Utils_vdm`String to Utils_vdm`String := {|->};
  res := res ++ {
    "Type" |-> typetoStringVDM(type), "Price (per day)" |-> Utils_vdm`toStringVDM(price)
  };
  return res
 );

functions

--Converys event type to string

 public typetoStringVDM: ServiceType -> Utils_vdm`String
 typetoStringVDM(t) == (
  cases t:
   <AudioVisual> -> "Audio Visual",
   <Catering> -> "Catering",
   <IT> -> "IT",
   <Cleaning> -> "Cleaning",
   <Security> -> "Security",
   <Decoration> -> "Decoration"
  end
 )
 pre t in set {<AudioVisual>, <Catering>, <IT>, <Cleaning>, <Security>, <Decoration>};


end Service
```

Function or operation	Line	Coverage	Calls
Service	12	100.0%	12
setPrice	21	100.0%	6
showDetails	28	100.0%	6
typetoStringVDM	40	100.0%	5
Service.vdmpp		100.0%	29

```
class User
instance variables
 public name: Utils_vdm`String := "default_name";
 public password: Utils_vdm`String := "pass1234";
 public events: [set of Utils_vdm`String] := {};

operations
--Constructor

 public User: Utils_vdm`String * Utils_vdm`String ==> User
 User(n, p) == (
  name := n;
  password := p
 )
 pre n <> "" and p <> ""
 post name = n and password = p;

--Shows all user information

 public showDetails: () ==> map Utils_vdm`String to Utils_vdm`String
 showDetails() == (
  dcl res: map Utils_vdm`String to Utils_vdm`String := {|->};
  res := res ++ {
    "Name" |-> name, "Events Attended" |-> eventsSettoStringVDM(events)
  };
  return res
 );

--Add event

 public addEvent: Utils_vdm`String ==> ()
 addEvent(evName) == events := events union {evName}
 post evName in set events;

--Remove event

 public removeEvent: Utils_vdm`String ==> ()
 removeEvent(evName) == events := events \ {evName}
 pre evName in set events
 post evName not in set events;

--Checks if this user attend the event

 public attendEvent: Utils_vdm`String ==> bool
 attendEvent(evName) == return evName in set events;

--Converts set of events to string

 public eventsSettoStringVDM: [set of Utils_vdm`String] ==> Utils_vdm`String
 eventsSettoStringVDM(events_var) == (
  dcl res: Utils_vdm`String := "";
  if(card events_var = 0) then return "";
  for all event in set events_var do res := res ^ event ^ ", ";
```

```
  res := res(1, ..., len res - 2);
  return res
 );
end User
```

Function or operation	Line	Coverage	Calls
User	9	100.0%	30
addEvent	28	100.0%	95
attendEvent	39	100.0%	5
eventsSettoStringVDM	43	100.0%	29
removeEvent	33	100.0%	30
showDetails	18	100.0%	35
User.vdmpp		100.0%	224

```
class Utils_vdm
types
 public Date ::  year : nat
         month: nat
         day : nat
    inv d == d.month <= 12 and d.day <= DaysOfMonth(d.year, d.month);
 public String = seq of char
 ord a < b == a = b;
values
functions

--Checks if date b is before date a

 public before: Date * Date -> bool
 before(b, a) ==
  b.year < a.year or
  (b.year = a.year and b.month < a.month) or
  (b.year = a.year and b.month = a.month and b.day < a.day)
 pre true
 post RESULT = (b.year < a.year or (b.year = a.year and b.month < a.month) or (b.year = a.year
     and b.month = a.month and b.day < a.day));

--Get the number os days per month

 public DaysOfMonth: nat * nat -> nat
  DaysOfMonth(y, m) ==
     if (m = 2)
     then
       (if (isLeapYear(y))
       then 29
       else 28)
         else (31 - (m - 1) mod 7 mod 2)
  pre y > 0 and m > 0 and m < 13
  post (m = 2 and isLeapYear(y) and RESULT = 29) or
   (m = 2 and not isLeapYear(y) and RESULT = 28) or
   (RESULT = (31 - (m - 1) mod 7 mod 2));

--Checks if this year is a leap year (with 366 days)

 public isLeapYear: nat -> bool
 isLeapYear(y) ==
  y mod 4 = 0 and y mod 100 <> 0 or y mod 400 = 0
 pre y > 0
```

```
 post RESULT = (y mod 4 = 0 and y mod 100 <> 0 or y mod 400 = 0);

--Calculates the difference between two dates

 public getDatesDifference: Date * Date -> int
 getDatesDifference(date1, date2) == (
  dateCount(date2.year, date2.month, 1, date2.year * 365 + date2.day) -
  dateCount(date1.year, date1.month, 1, date1.year * 365 + date1.day)
 )
 pre date1 = date2 or before(date1, date2)
 post RESULT >= 0;


 public dateCount: nat * nat * int * nat -> int
 dateCount(year, month, i, res) == (
    if(i = month) then res
    else dateCount(year, month, i + 1, res + DaysOfMonth(year, i))
 )
 pre year > 0 and month > 0 and i > 0 and res > 0;

-- Curried Function
-- Received function can return a boolean or an element of the same map value type
-- Apply a generic function to all map elements and change them (when function returns an element
    ) or
-- filter the ones that verify the condition (when function returns a boolen)

 public fMap[@keyType, @returnType, @valueType]: (@valueType -> @returnType) -> seq of @keyType *
      (map @keyType to @valueType) -> (map @keyType to @valueType)
 fMap (f)(keys, m) ==
 if keys = [] then {|->}
 else let key = hd keys in (
  let res = f(m(key)) in (
    if(res = true) then {key|->m(key)} ++ (fMap[@keyType, @returnType, @valueType] (f)(tl keys, m
        ))
    else fMap[@keyType, @returnType, @valueType] (f)(tl keys, m)
  )
 )
 pre elems keys subset dom m
 post dom RESULT subset elems keys
 measure mfMap[@keyType, @returnType, @valueType](f, keys, m);

-- fMap measure function

 public mfMap[@keyType, @returnType, @valueType]: (@valueType -> @returnType) * seq of @keyType *
      (map @keyType to @valueType) -> nat
 mfMap (-, keys, -) == len keys;

--Turns set into seq

 public setTOseq[@elem]: set of @elem -> seq of @elem
 setTOseq(tmpSet) == (
  [x | x in set tmpSet]
 )
 post elems RESULT subset tmpSet;



-- converts an element to string
-- public toStringVDM[@elem]: @elem -> String
-- toStringVDM(value) == (
--  cases true:
--   (is_String(value)) -> value,
--   (is_Date(value)) -> datetoStringVDM(value),
--   (is_nat(value)) -> nattoStringVDM(value),
--   (is_real(value)) -> nattoStringVDM(value) ^ [',', ' '] ^ nattoStringVDM(getRemainder(value,
```

```
    0)),
--    (is_bool(value)) -> booltoStringVDM(value)
--  end
-- )
-- measure 1;
-- converts an element to string

 public toStringVDM: String -> String
 toStringVDM(value) == (
  value
 )
 measure 1;
 public toStringVDM: bool -> String
 toStringVDM(value) == (
  booltoStringVDM(value)
 )
 measure 1;
 public toStringVDM: Date -> String
 toStringVDM(value) == (
  datetoStringVDM(value)
 )
 measure 1;
 public toStringVDM: real -> String
 toStringVDM(value) == (
  nattoStringVDM(value) ^ [',', ' '] ^ nattoStringVDM(getRemainder(value, 0))
 )
 measure 1;


 public booltoStringVDM: bool -> String
 booltoStringVDM(value) == if(value) then "yes" else "no";


 public datetoStringVDM: Date -> String
 datetoStringVDM(value) == nattoStringVDM(value.day) ^ ['/'] ^ nattoStringVDM(value.month) ^ ['/'
     ] ^ nattoStringVDM(value.year);


 public nattoStringVDM: real -> String
 nattoStringVDM(value) == (
  if(floor(value) < 10) then mapNat(floor(value))
  else nattoStringVDM(floor(value) div 10) ^ mapNat(floor(value) rem 10)
 );


 public getRemainder: real * int -> int
 getRemainder(value, n) == (
  if(is_int(value)) then value rem (10 ** n)
  else getRemainder(value * 10, n + 1)
 );


 public mapNat: nat -> String
 mapNat(n) == (
  cases n:
   0 -> "0",
   1 -> "1",
   2 -> "2",
   3 -> "3",
   4 -> "4",
   5 -> "5",
   6 -> "6",
   7 -> "7",
   8 -> "8",
   9 -> "9"
```

```
  end
 );


end Utils_vdm
```

Function or operation	Line	Coverage	Calls
DaysOfMonth	22	100.0%	66
before	13	100.0%	1859
booltoStringVDM	122	100.0%	6
dateCount	51	100.0%	60
datetoStringVDM	125	100.0%	32
fMap	62	100.0%	414
getDatesDifference	43	100.0%	60
getRemainder	134	100.0%	10
isLeapYear	36	100.0%	210
mapNat	140	100.0%	104
mfMap	76	100.0%	414
nattoStringVDM	128	100.0%	988
setTOseq	80	100.0%	108
toStringVDM	101	100.0%	538
Utils_vdm.vdmpp		99.5%	4869

```
class CenterTest  is subclass of Test_vdm
operations

 /***** USE CASE SCENARIOS ******/

 --Center Init

  public createCenter: Utils_vdm`String * Installation ==> Center
  createCenter(name, inst) == (
   dcl res: Center := new Center(name, inst);
   res.addUser(new User("admin", "admin1234"));
   return res
  )
  post "admin" in set dom RESULT.users and card dom RESULT.installations = 1;

--Changing installations
-- Here we add installations to the center, aggregate installations to other installations,
-- remove installations from other installations and from center and
-- change installations' attributes
--RF16, RF17, RF18, RF19, RF20, RF21

 public testInstallations: Center ==> ()
 testInstallations(center) == (
  dcl room1: Installation := new Room("Room1", 15, 10, 7, 20, 20, false, true, true, true, false,
       false, true, false);
  dcl room2: Installation := new Room("Room2", 15, 10, 7, 20, 20, false, true, true, true, false,
       false, true, false);
  dcl pavilion1: Installation := new Pavilion("Pavilion1", 150, 50, 10, 50, 70, false, false,
      true, false, false);
  dcl pavilion2: Installation := new Pavilion("Pavilion2", 150, 50, 10, 50, 70, false, false,
      true, false, false);
```

```
dcl foyer1: Installation := new Foyer("Foyer1", 10, 15, 4, 6, 6, false, true, false, true);
dcl foyer2: Installation := new Foyer("Foyer2", 10, 15, 4, 6, 6, false, true, false, true);
dcl foyer3: Installation := new Foyer("Foyer3", 10, 15, 4, 6, 6, false, true, false, true);
dcl parking1: Installation := new CarParking("Car Parking1", 30, 50, 7, 50, 50);
dcl auditorium1: Installation := new Auditorium("Auditorium1", 50, 120, 10, 40, 40, true, false
    , true, false, true, false);
--Add installations
center.addInstallations("admin", room1);
center.addInstallations("admin", pavilion1);
center.addInstallations("admin", pavilion2);
assertTrue({room1, pavilion1} subset rng center.installations);

center.addInstallations("admin", auditorium1);
center.addInstallationToInstallation("admin", "Auditorium1", foyer1);
assertTrue(center.associatedInstallations(auditorium1, foyer1));
assertTrue(not center.hasInstallation(auditorium1, room1));
assertTrue(not center.hasInstallation(room1, auditorium1));
center.addInstallationToInstallation("admin", "Pavilion1", room1);
center.addInstallationToInstallation("admin", "Pavilion1", room2);
assertTrue(center.associatedInstallations(pavilion1, pavilion1));
assertTrue(center.associatedInstallations(pavilion1, room1));
center.addInstallationToInstallation("admin", "Pavilion1",  foyer2);
center.addInstallationToInstallation("admin", "Pavilion2",  foyer3);
assertEqual(narrow_(center.installations("Pavilion1"), Pavilion).rooms, {room1,room2});
assertEqual(narrow_(center.installations("Pavilion1"), Pavilion).foyers, {foyer2});

assertTrue(foyer2 in set rng center.installations);
--Remove installations
center.removeInstallationFromInstallation("admin", "Auditorium1", foyer1);
assertTrue(not auditorium1.hasInstallation(foyer1));
center.removeInstallationFromInstallation("admin", "Pavilion2", foyer3);
assertTrue(not pavilion2.hasInstallation(foyer3));
center.removeInstallationFromInstallation("admin", "Pavilion1", room1);
assertEqual(narrow_(center.installations("Pavilion1"), Pavilion).rooms, {room2});
assertTrue(room1 in set rng center.installations);

center.addInstallationToInstallation("admin", "Pavilion1", foyer1);
center.removeInstallation("admin", "Foyer2");
center.removeInstallation("admin", "Pavilion1");
center.removeInstallation("admin", "Foyer3");
center.removeInstallation("admin", "Pavilion2");
assertTrue(rng center.installations inter {pavilion1, foyer2} = {});
--Change installation attributes
center.addInstallations("admin", pavilion1);
center.addInstallations("admin", foyer2);
center.addInstallations("admin", parking1);
center.changeInstallationMeasures("admin", "Room1", 20, 10, 10, 25);
center.changeInstallationMeasures("admin", "Pavilion1", 20, 10, 10, 25);
center.changeInstallationMeasures("admin", "Foyer2", 20, 10, 10, 25);
center.changeInstallationMeasures("admin", "Car Parking1", 20, 10, 10, 25);
center.changeInstallationMeasures("admin", "Auditorium1", 20, 10, 10, 25);
assertTrue(room1.capacity = 20 and room1.heigth = 10 and room1.width = 10 and room1.lenght =
    25);
center.changeInstallationRent("admin", "Room1", 32);
center.changeInstallationRent("admin", "Pavilion1", 32);
center.changeInstallationRent("admin", "Foyer2", 32);
center.changeInstallationRent("admin", "Car Parking1", 32);
center.changeInstallationRent("admin", "Auditorium1", 32);
assertEqual(room1.price, 32);
center.changeInstallationConditions("admin", "Room1", true, true, true, true, true, true, true,
     true);
center.changeInstallationConditions("admin", "Pavilion1", true, true, true, true, true, true,
    true, true);
center.changeInstallationConditions("admin", "Foyer2", true, true, true, true, true, true, true
    , true);
```

```
  center.changeInstallationConditions("admin", "Car Parking1", true, true, true, true, true, true
      , true, true);
  center.changeInstallationConditions("admin", "Auditorium1", true, true, true, true, true, true,
       true, true);
  assertTrue(room1.airCondition and room1.naturalLigth and room1.ceilingLighting and room1.
      blackOutCurtains and
   room1.telephones and room1.computerNetwork and room1.soundproofWalls and room1.movingWalls);
  assertTrue((parking1.airCondition or parking1.naturalLigth or parking1.ceilingLighting or
      parking1.blackOutCurtains and
   parking1.telephones or parking1.computerNetwork or parking1.soundproofWalls or parking1.
       movingWalls) = false);

  assertEqual(card center.showInstallationsDetails(), card dom center.installations);
  center.addInstallations("admin", foyer1);

 )
 pre card dom center.installations = 1
 post card dom center.installations = 6;

--Simple action to add and remove a service from the center
--RF22, RF23, RF24

 public testServices: Center ==> ()
 testServices(center) == (
  dcl service1: Service := new Service(10, <AudioVisual>);

  center.addService("admin", service1);
  assertEqual(card dom center.services, 1);
  center.removeService("admin", <AudioVisual>);
  assertEqual(card dom center.services, 0);
  center.addService("admin", service1);
  assertEqual(card center.showServicesDetails(), 1)
 )
 pre card dom center.services = 0
 post card dom center.services = 1;

--Simple action of adding users to our center
--RF1

 public testUsers: Center ==> ()
 testUsers(center) == (
  dcl user1: User := new User("User1", "1234");
  dcl user2: User := new User("User2", "1234");
  dcl user3: User := new User("User3", "1234");
  dcl user4: User := new User("User4", "1234");

  center.addUser(user1);center.addUser(user2);
  center.addUser(user3);center.addUser(user4);
  assertEqual(card dom center.users, 5);

 )
 pre card dom center.users = 1
 post card dom center.users = 5;

--RF2, RF5, RF8, RF9, RF14, RF15

 public testEdge: Center ==> ()
 testEdge(center) == (
 (
  dcl pavilion3: Installation := new Pavilion("Pavilion3", 150, 50, 10, 50, 70, false, false,
      true, false, false);
  dcl foyer4: Installation := new Foyer("Foyer4", 10, 15, 4, 6, 6, false, true, false, true);
  center.addInstallations("admin", pavilion3);
  center.addInstallationToInstallation("admin", "Pavilion3", foyer4);
 );
```

```
(

 dcl event2: Event := center.createEvent("Event2", 3, 10, mk_Utils_vdm`Date(2018, 12, 4),
     mk_Utils_vdm`Date(2018, 12, 4), false, <Conference>, center.installations("Pavilion3"), "
     User1");
 dcl event3: Event := center.createEvent("Event3", 3, 10, mk_Utils_vdm`Date(2018, 12, 5),
     mk_Utils_vdm`Date(2018, 12, 5), true, <Conference>, center.installations("Foyer4"), "User1"
     );
 dcl event4: Event := center.createEvent("Event4", 3, 10, mk_Utils_vdm`Date(2018, 12, 6),
     mk_Utils_vdm`Date(2018, 12, 6), true, <Conference>, center.installations("Room1"), "User1")
     ;
 dcl event5: Event := center.createEvent("Event5", 3, 10, mk_Utils_vdm`Date(2018, 12, 7),
     mk_Utils_vdm`Date(2018, 12, 7), true, <Conference>, center.installations("Room1"), "User1")
     ;
 dcl service1: Service := new Service(10, <IT>);
 dcl d1: Utils_vdm`Date := mk_Utils_vdm`Date(2020, 2, 4);
 dcl d2: Utils_vdm`Date := mk_Utils_vdm`Date(2100, 2, 4);

 assertTrue("Pavilion3" not in set dom center.getAvailableInstallations(mk_Utils_vdm`Date(2018,
     12, 4), mk_Utils_vdm`Date(2018, 12, 4)));
 assertTrue("Foyer4" not in set dom center.getAvailableInstallations(mk_Utils_vdm`Date(2018, 12,
      4), mk_Utils_vdm`Date(2018, 12, 4)));
 assertTrue("Pavilion3" not in set dom center.getAvailableInstallations(mk_Utils_vdm`Date(2018,
     12, 5), mk_Utils_vdm`Date(2018, 12, 5)));
 assertTrue("Foyer4" not in set dom center.getAvailableInstallations(mk_Utils_vdm`Date(2018,
     12,5), mk_Utils_vdm`Date(2018, 12, 5)));


 center.addUserToEvent("Event3", "User2", <staff>);
 assertTrue("User2" in set event3.staff);
 center.addUserToEvent("Event3", "User3", <staff>);
 assertTrue("User3" in set event3.staff);
 center.addUserToEvent("Event3", "User2", <host>);
 assertEqual(event3.host, "User2");
 assertTrue(not "User2" in set event3.staff);
 center.removeUserFromEvent("Event3", "User3", <staff>);
 assertTrue(not "User3" in set event3.staff);

 center.addService("admin", service1);
 center.changeEventPrivacy("Event2", "User1", true);
 assertEqual(event2.privacy, true);
 assertEqual(event2.guests, event2.attendees);
 center.addServiceToEvent("Event2", "User1", <AudioVisual>);
 center.addServiceToEvent("Event2", "User1", <IT>);
 assertTrue(len event2.services = 2);

 assertTrue(center.listEventServices("Event2") union center.availableServicesForEvent("Event2")
     = rng center.services);
 center.removeService("admin", <IT>);
 assertTrue(len event2.services = 1);

 center.addService("admin", service1);
 assertTrue(center.listEventServices("Event2") union center.availableServicesForEvent("Event2")
     = rng center.services);
 assertTrue(center.listEvents("admin") = rng center.events);
 assertTrue(center.showAvailableEventsBetweenDates("admin",mk_Utils_vdm`Date(2018, 12, 7),
     mk_Utils_vdm`Date(2018, 12, 7)) = {event5});
 center.inviteToEvent("Event5", "User1", "User2");
 center.addUserToEvent("Event5", "User2", <attendee>);
 center.changeEventTotalTickets("Event5", "User1", 1);
 assertTrue(card event5.attendees = event5.totalTickets);
 assertTrue(center.showAvailableEventsBetweenDates("admin",mk_Utils_vdm`Date(2018, 12, 7),
     mk_Utils_vdm`Date(2018, 12, 7)) = {});
 assertTrue(center.showAvailableEvents("admin") = rng center.events \ {event5});
```

```
  assertEqual(center.showEventDetails("admin", "Event3")("Name"), "Event3");
  assertEqual(center.showEventDetails("User2", "Event3")("Name"), "Event3");

  assertTrue(center.events("Event3").earnedMoney() = 0);
  assertTrue(center.events("Event3").remainingTickets() = 3);

  assertTrue(Event`typetoStringVDM(<TeamBuilding>) ="Team Building");
  assertTrue(Event`typetoStringVDM(<TradeFair>) ="Trade Fair");
  assertTrue(Event`typetoStringVDM(<Musical>) ="Musical");

  assertTrue(Service`typetoStringVDM(<Catering>) ="Catering");
  assertTrue(Service`typetoStringVDM(<IT>) ="IT");
  assertTrue(Service`typetoStringVDM(<Cleaning>) ="Cleaning");
  assertTrue(Service`typetoStringVDM(<Security>) ="Security");
  assertTrue(Service`typetoStringVDM(<Decoration>) ="Decoration");

  assertEqual(Utils_vdm`getRemainder(1.33,0),33);
  assertEqual(Utils_vdm`mapNat(4),"4");
  assertEqual(Utils_vdm`mapNat(7),"7");
  assertEqual(Utils_vdm`mapNat(9),"9");

  assertTrue(Pavilion`sumElems([1, 0.1]) = 1.1);

  assertTrue(not center.users("admin").attendEvent("Event3"));

  assertTrue(center.showUserDetails("User1","User1")("Name") = "User1");

  center.changeEventDate("Event5", "User1", <ending>, mk_Utils_vdm`Date(2019, 12, 5));
  assertEqual(center.events("Event5").ending, mk_Utils_vdm`Date(2019, 12, 5));
  center.changeEventDate("Event5", "User1", <begin>, mk_Utils_vdm`Date(2019, 12, 2));
  assertEqual(center.events("Event5").begin, mk_Utils_vdm`Date(2019, 12, 2));

 )
)

 pre card dom center.events = 1
 post card dom center.events = 5;

--Create an event and change its attributes (event details, services and installation)

--RF4, RF6, RF7, RF13
 public testEvent: Center ==> ()
 testEvent(center) == (
  dcl event1: Event := center.createEvent("Event1", 3, 10, mk_Utils_vdm`Date(2018, 12, 1),
      mk_Utils_vdm`Date(2018, 12, 3), true, <Conference>, center.installations("Room1"), "User1")
      ;

  --Check event and its availability to the users
  assertTrue("Room1" not in set dom center.getAvailableInstallations(mk_Utils_vdm`Date(2018, 12,
      1), mk_Utils_vdm`Date(2018, 12, 3)));
  assertEqual(center.showAvailableEvents("User2"), {});
  center.inviteToEvent("Event1", "User1", "User2");
  assertEqual(center.showAvailableEvents("User2"), {event1});
  assertEqual(center.showAvailableEventsBetweenDates("User3", mk_Utils_vdm`Date(2018, 12, 1),
      mk_Utils_vdm`Date(2018, 12, 3)), {});
  --Add and remove services from event
  center.addServiceToEvent("Event1", "User1", <AudioVisual>);
  assertTrue(len event1.services = 1);
  center.removeServiceFromEvent("Event1", "User1", <AudioVisual>);
  assertTrue(len event1.services = 0);
  center.addServiceToEvent("Event1", "User1", <AudioVisual>);
  center.changeEventInstallation("Event1", "User1", "Foyer1");
  assertEqual(event1.installation.id, "Foyer1");
  --Change event attributes
  center.changeEventName("Event1", "User1", "New Event Name");
```

```
  assertEqual(event1.name, "New Event Name");
  center.changeEventTotalTickets("New Event Name", "User1", 6);
  assertEqual(event1.totalTickets, 6);
  center.changeEventTicketPrice("New Event Name", "User1", 5);
  assertEqual(event1.ticketPrice, 5);
  center.changeEventDate("New Event Name", "User1", <begin>, mk_Utils_vdm`Date(2018, 12, 2));
  assertEqual(event1.begin, mk_Utils_vdm`Date(2018, 12, 2));
  center.changeEventDate("New Event Name", "User1", <ending>, mk_Utils_vdm`Date(2018, 12, 5));
  assertEqual(event1.ending, mk_Utils_vdm`Date(2018, 12, 5));
  center.changeEventPrivacy("New Event Name", "User1", false);
  assertEqual(event1.privacy, false);
  assertEqual(center.showAvailableEventsBetweenDates("User3", mk_Utils_vdm`Date(2018, 12, 1),
      mk_Utils_vdm`Date(2018, 12, 3)), {event1});
  center.changeEventType("New Event Name", "User1", <Party>);
)
 pre card dom center.events = 0
 post card dom center.events = 1;

--By changing differente external factors, like services' price,
-- installation's rent and number of attendees,
-- we will analise the money earned and lost with the event

--RF2, RF3, RF9, RF11, RF12, RF25
 public testEventProfit: Center ==> ()
 testEventProfit(center) == (
  dcl event1: Event := center.events("New Event Name");
  --Add users to the event and change their position in it (host, attendee or staff)
  center.addUserToEvent("New Event Name", "User2", <attendee>);
  assertTrue("User2" in set event1.attendees);
  center.addUserToEvent("New Event Name", "User3", <host>);
  assertEqual(event1.host, "User3");
  center.addUserToEvent("New Event Name", "User1", <staff>);
  assertTrue("User1" in set event1.staff);
  center.addUserToEvent("New Event Name", "User4", <attendee>);
  assertEqual(event1.attendees, {"User2", "User4"});
  center.addUserToEvent("New Event Name", "User4", <host>);
  center.addUserToEvent("New Event Name", "User3", <attendee>);
  assertEqual(event1.host, "User4");
  assertEqual(event1.attendees, {"User2", "User3"});
  center.removeUserFromEvent("New Event Name", "User3", <attendee>);
  assertEqual(event1.attendees, {"User2"});
  center.addUserToEvent("New Event Name", "User3", <attendee>);
  assertEqual(event1.attendees, {"User2", "User3"});
  assertEqual(center.showEventDetails("admin", "New Event Name")("Sold tickets"), "2, 0");
  let usersDetails = center.showUsersDetails("admin") in (
   for all userDetails in set usersDetails do (
    if(userDetails("Name") in set event1.attendees) then assertEqual(userDetails("Money spent"),
        "5, 0")
   )
  );
  --Check money spent with the event
  assertEqual(center.moneySpentWithServices("User4", "New Event Name"), 40);--since service (<
      AudioVisual>) price is 10 and event days are 4
  assertEqual(center.moneySpentWithServices("admin", "New Event Name"), 40);--since service (<
      AudioVisual>) price is 10 and event days are 4
  assertEqual(center.moneySpentWithInstallation("User4", "New Event Name"), 40);--since
      installation ("Foyer1") rent is 10 and event days are 4
  assertEqual(center.moneySpentWithInstallation("admin", "New Event Name"), 40);--since
      installation ("Foyer1") rent is 10 and event days are 4
  center.changeServicePrice("admin", <AudioVisual>, 30);
  assertEqual(center.services(<AudioVisual>).price, 30);
  assertEqual(center.moneySpentWithServices("User4", "New Event Name"), 120)--since service (<
      AudioVisual>) price is 10 and event days are 4

 )
```

```
pre card dom center.events = 1;
/***** TEST CASES WITH VALID INPUTS ******/


public static main: () ==> ()
main() ==
(
  dcl centerTest: CenterTest := new CenterTest();
 dcl center: Center := centerTest.createCenter("Super Center", new Foyer("Foyer1", 10, 15, 4, 6,
      6, false, true, false, true));
 centerTest.testInstallations(center);
 centerTest.testServices(center);
 centerTest.testUsers(center);
 centerTest.testEvent(center);
 centerTest.testEventProfit(center);
 centerTest.testEdge(center);
);



  /***** TEST CASES WITH INVALID INPUTS (EXECUTE ONE AT A TIME) ******/
public static testBuyTicketToPrivateWhileNotInvited: () ==> ()
testBuyTicketToPrivateWhileNotInvited() == (
 dcl center: Center := new CenterTest().createCenter("Super Center", new Foyer("Foyer1", 10, 15,
      4, 6, 6, false, true, false, true));
 dcl user1: User := new User("User1", "1234");

 dcl user2: User := new User("User2", "1234");
 (
 dcl e : Event := new Event("Event1", 3, 10, mk_Utils_vdm`Date(2018, 12, 4), mk_Utils_vdm`Date
     (2018, 12, 4), true, <Conference>, center.installations("Foyer1"), "User1");
 e.addAttendee("User2");
 )

);
public static testRoomToTwoPavilions: () ==> ()
testRoomToTwoPavilions() == (
 dcl center: Center := new CenterTest().createCenter("Super Center", new Foyer("Foyer1", 10, 15,
      4, 6, 6, false, true, false, true));
 dcl room1: Installation := new Room("Room1", 15, 10, 7, 20, 20, false, true, true, true, false,
      false, true, false);
 dcl pavilion1: Installation := new Pavilion("Pavilion1", 150, 50, 10, 50, 70, false, false,
     true, false, false);
 dcl pavilion2: Installation := new Pavilion("Pavilion2", 150, 50, 10, 50, 70, false, false,
     true, false, false);
 --Add installations
 center.addInstallations("admin", pavilion1);
 center.addInstallations("admin", pavilion2);
 center.addInstallationToInstallation("admin", "Pavilion1", room1);
 center.addInstallationToInstallation("admin", "Pavilion2", room1);
);
public static testCreateRoomWithoutParameters: () ==> ()
testCreateRoomWithoutParameters() == (
 dcl r : Room := new Room(); -- pre-condition fail
 skip;
);
public static testAddServicesWithoutAdmin: () ==> ()
testAddServicesWithoutAdmin() == (
 dcl center: Center := new CenterTest().createCenter("Super Center", new Foyer("Foyer1", 10, 15,
      4, 6, 6, false, true, false, true));
 center.addService("User1", new Service(10, <IT>)); -- pre-condition fail
);
public static testEventWithWrongDates: () ==> ()
testEventWithWrongDates() == (
 dcl event: Event := new Event("WRONG", 10, 9.99, mk_Utils_vdm`Date(2019,1,1), mk_Utils_vdm`Date
```

```
      (2018,1,1), false, <Party>, new Foyer("Foyer1", 10, 15, 4, 6, 6, false, true, false, true),
       "ME");
  skip; -- pre-condition fail
 );
end CenterTest
```

Function or operation	Line	Coverage	Calls
createCenter	7	100.0%	6
main	310	100.0%	2
testAddServicesWithoutAdmin	328	0.0%	0
testBuyTicketToPrivateWhileNotInvited	323	0.0%	0
testCreateRoomWithoutParameters	323	0.0%	0
testEdge	134	100.0%	18
testEvent	230	100.0%	2
testEventProfit	271	100.0%	2
testEventWithWrongDates	333	0.0%	0
testInstallations	20	99.5%	6
testRoomToTwoPavilions	323	0.0%	0
testServices	102	100.0%	6
testUsers	118	100.0%	6
CenterTest.vdmpp		89.1%	48

```
class Test_vdm

operations

protected assertTrue: bool ==> ()
assertTrue(arg) == return pre arg;

protected assertEqual: ? * ? ==> ()
assertEqual(expected, actual) == if expected <> actual then ( IO`print("Value ("); IO`print(
    actual); IO`print(") different from expected
    ("); IO`print(expected); IO`println(")\n")
) post expected = actual

end Test_vdm
```

Function or operation	Line	Coverage	Calls
assertEqual	6	38.8%	0
assertTrue	4	100.0%	147
Test_vdm.vdmpp		45.0%	147