MFES

January 1, 2018

Contents

1 Agenda

```
class Agenda
instance variables

private healthProfessional : HealthProfessional;
private agenda : set of (Schedule);

inv card agenda >= 0;

operations
 -- Agenda constructor

 public Agenda : HealthProfessional ==> Agenda
  Agenda(h) == (healthProfessional := h; agenda := {}; return self)
 post healthProfessional = h and agenda = {};

 -- Returns the agenda's health professional

 pure public getHealthProfessional : () ==> HealthProfessional
  getHealthProfessional() == (return healthProfessional);

 -- Returns the agenda's schedules

 pure public getAgenda : () ==> set of (Schedule)
  getAgenda() == (return agenda);

 -- Adds a schedule to the agenda

 public addSchedule : Schedule ==> ()
  addSchedule(s) == (agenda := agenda union {s})
 pre s not in set agenda and forall sch in set agenda & not overlap(s, sch)
 post s in set agenda;

 -- Removes a schedule from the agenda

 public removeSchedule : Schedule ==> ()
  removeSchedule(s) == (agenda := agenda \ {s})
 pre s in set agenda
 post s not in set agenda;

 -- Checks if a schedule overlaps with other schedule

 pure public overlap: Schedule * Schedule ==> bool
  overlap(t1, t2) == (return t1.overlap(t1, t2));
end Agenda
```

Function or operation	Line	Coverage	Calls
Agenda	11	100.0%	44
addSchedule	24	100.0%	84
getAgenda	20	100.0%	208
getHealthProfessional	16	100.0%	1175
overlap	36	100.0%	40
removeSchedule	30	100.0%	64
Agenda.vdmpp		100.0%	1615

2 Appointment

```
class Appointment is subclass of Task

instance variables
  private priority : Types`Priority;

  inv priority <> nil;
  inv medicalAssoc.getType() = <Doctor>;
operations
 -- Appointment constructor
 public Appointment: HealthProfessional * Schedule * Patient * Hospital==> Appointment
  Appointment(d, s, p, h) == (medicalAssoc := d; priority := <Medium>; Task(d, s, p, h, <
      Appointment>))

 post medicalAssoc = d and priority = <Medium>;

 -- Urgency Apppointment constructor
 public Appointment: HealthProfessional * Types`Priority * Schedule * Patient * Hospital ==>
     Appointment
  Appointment(d, p, s, pat, h) == (medicalAssoc := d; priority := p; Task(d, s, pat, h, <
      Urgencies>))
 pre p <> nil
 post medicalAssoc = d and priority = p;

 -- Returns the appointment's priority
 pure public getPriority : () ==> Types`Priority
  getPriority() == (return priority);


  -- Sets the appointment's priority
  public setPriority : Types`Priority ==> ()
   setPriority(p) == (priority := p)

  pre taskType = <Urgencies>;

end Appointment
```

Function or operation	Line	Coverage	Calls
Appointment	12	100.0%	16
addPrescription	36	0.0%	0
getPrescriptions	27	100.0%	4
getPriority	23	100.0%	12
removePrescription	42	0.0%	0
setPriority	31	100.0%	4
Appointment.vdmpp		100.0%	36

3 HealthProfessional

```
class HealthProfessional is subclass of Person

instance variables
  private medicalNumber: Types`String;
  private specialties:set of (Specialty);
  private patients : set of(Patient);
 private type : Types`Type;

 inv card patients >= 0;
  inv card specialties < 5;
 inv type <> nil;
 inv len medicalNumber > 5;
operations
 -- Health Professional constructor

 public HealthProfessional: Types`String * Types`String * Types`String * Types`String * Types`
     String * Types`String * Types`Type ==> HealthProfessional
  HealthProfessional(a, fn, ln, c, pn, s, t) == (medicalNumber := s; type := t; specialties :=
      {}; patients := {}; Person(a, fn, ln, c, pn))
 pre t <> nil and len s > 5
 post medicalNumber = s and type = t and specialties = {} and patients = {};

 -- Returns the health professional's number

 pure public getMedicalNumber: () ==> Types`String
  getMedicalNumber() == (return medicalNumber);

 -- Returns the health professional's specialties

 pure public getSpecialties: () ==> set of (Specialty)
  getSpecialties() == (return specialties);

 -- Returns all the health professional's patients

 pure public getPatients: () ==> set of (Patient)
  getPatients() == (return patients);

 -- Returns the health professional's type

 pure public getType : () ==> Types`Type
  getType() == (return type);

 -- Removes a specialty from the health professional's specialties

 public removeSpecialty: Specialty ==> ()
  removeSpecialty(s) == (specialties := specialties \ {s})
 pre s in set specialties
 post s not in set specialties;

 -- Adds a specialty to the health professional's specialties

 public addSpecialty: Specialty ==> ()
  addSpecialty(s) == (specialties := specialties union {s})
 pre s not in set specialties
 post s in set specialties;

 -- Adds a patient to the health professional's patients

 public addPatient : Patient ==> ()
  addPatient(p) == (patients :=  patients union {p})
 pre p not in set patients
 post p in set patients;

 -- Removes a patient from the health professional's patients

 public removePatient : Patient ==> ()
  removePatient(p) == (patients := patients \ {p})
 pre p in set patients
 post p not in set patients;
```

```
end HealthProfessional
```

Function or operation	Line	Coverage	Calls
HealthProfessional	15	100.0%	136
addPatient	49	100.0%	48
addSpecialty	43	100.0%	8
getMedicalNumber	21	100.0%	16
getPatients	29	100.0%	144
getSpecialties	25	100.0%	44
getType	33	100.0%	692
removePatient	55	100.0%	4
removeSpecialty	37	100.0%	8
HealthProfessional.vdmpp		100.0%	1100

4 Hospital

```
class Hospital
instance variables
  private medicalAssociated: set of (HealthProfessional);
  private agenda : set of (Agenda);
  private name: Types`String;
  private address: Types`String;
  private tasks: set of(Task);
  private trainings: set of(Training);
  private safetyNet: SafetyNetHospital;

 inv card medicalAssociated >= 0;
 inv card agenda <= card medicalAssociated;
 inv card tasks >= 0;
 inv card trainings >= 0;
operations
 -- Hospital constructor

 public Hospital: Types`String * Types`String * SafetyNetHospital ==> Hospital
  Hospital(n, a, s) == (name := n; address := a; safetyNet := s; medicalAssociated := {}; tasks
      := {}; trainings := {}; agenda := {};
  safetyNet.addHospital(self); return self)
 post name = n and address = a and safetyNet = s and medicalAssociated = {} and tasks = {} and
     trainings = {} and agenda = {};

 -- Returns the hospital's name

 pure public getName: () ==> Types`String
  getName() == (return name);

 -- Returns the hospital's address

 pure public getAddress: () ==> Types`String
  getAddress() == (return address);

 -- Returns the tasks created on this hospital

 pure public getTasks: () ==> set of (Task)
  getTasks() == (return tasks);

 -- Returns the trainings created on this hospital

 pure public getTrainings : () ==> set of (Training)
  getTrainings() == (return trainings);

 -- Returns all the agendas of the health professionals of this hospital

 pure public getAgendas : () ==> set of(Agenda)
  getAgendas() == (return agenda);

 -- Returns the schedules of a health professional's agenda

 pure public getAgenda : HealthProfessional ==> Agenda
  getAgenda(h) == (
   dcl a1 : Agenda;
   for all a2 in set agenda do
    if(a2.getHealthProfessional() = h)
     then a1 := a2;
   return a1);

 -- Removes an agenda

 public removeAgenda : Agenda ==> ()
  removeAgenda(a) == (agenda := agenda \ {a})
 pre a in set agenda
 post a not in set agenda;

 -- Adds a health professional to the hospital's health professionals

 public addMedAssociated: HealthProfessional ==> ()
  addMedAssociated(d) == (
   dcl agendaNew : Agenda;
   agendaNew := new Agenda(d);
   medicalAssociated := {d} union medicalAssociated;
   agenda := agenda union {agendaNew})
 pre d not in set medicalAssociated
 post d in set medicalAssociated;

 -- Removes a health professional from the hospital's health professionals

 public removeMedAssociated: HealthProfessional ==> ()
  removeMedAssociated(d) == (
   for all t in set tasks do
    if(d = t.getMedAssoc())
     then removeTask(t);
   for all t in set trainings do
    if(d = t.getMedAssoc())
     then removeTraining(t);
   for all a in set agenda do
    if(a.getHealthProfessional().getCC() = d.getCC())
     then removeAgenda(a);
   medicalAssociated := medicalAssociated \ {d})
 pre d in set medicalAssociated
 post d not in set medicalAssociated;

 -- Adds a task to the hospital

 public addTask: Task ==> ()
  addTask(d) == (
   if(d.getPatient() not in set d.getMedAssoc().getPatients())
    then d.getMedAssoc().addPatient(d.getPatient());
   tasks := {d} union tasks;
   for all a in set agenda do
```

```
   if(a.getHealthProfessional().getCC() = d.getMedAssoc().getCC())
     then a.removeSchedule(d.getSchedule());)
pre d not in set tasks and d.getSchedule() in set getAgenda(d.getMedAssoc()).getAgenda()
post d in set tasks and d.getPatient() in set d.getMedAssoc().getPatients() and d.getSchedule()
    not in set getAgenda(d.getMedAssoc()).getAgenda();

-- Removes a task from the hospital

public removeTask: Task ==> ()
 removeTask(d) == (
  for all a in set agenda do
   if(a.getHealthProfessional() = d.getMedAssoc())
     then a.addSchedule(d.getSchedule());
  tasks := tasks \ {d})
pre d in set tasks and d.getSchedule() not in set getAgenda(d.getMedAssoc()).getAgenda()
post d not in set tasks and d.getSchedule() in set getAgenda(d.getMedAssoc()).getAgenda();

-- Adds a training to the hospital

public addTraining: Training ==> ()
 addTraining(d) == (
  for all a in set agenda do
   if(a.getHealthProfessional() = d.getMedAssoc())
     then a.removeSchedule(d.getSchedule());
  trainings := {d} union trainings)
pre d not in set trainings and d.getSchedule() in set getAgenda(d.getMedAssoc()).getAgenda()
post d in set trainings and d.getSchedule() not in set getAgenda(d.getMedAssoc()).getAgenda();

-- Removes a training from the hospital

public removeTraining: Training ==> ()
 removeTraining(d) == (
  for all a in set agenda do
   if(a.getHealthProfessional() = d.getMedAssoc())
     then a.addSchedule(d.getSchedule());
  trainings := trainings \ {d})
pre d in set trainings and d.getSchedule() not in set getAgenda(d.getMedAssoc()).getAgenda()
post d not in set trainings and d.getSchedule() in set getAgenda(d.getMedAssoc()).getAgenda();

-- Returns the tasks of a hospital by its type

pure public getTasksByType: Types`TaskType ==> set of (Task)
 getTasksByType(s) == (
  dcl tasksTotal: set of (Task);
  tasksTotal := {};
  for all t in set tasks do
   if(t.getType() = s)
    then tasksTotal := tasksTotal union {t};
  return tasksTotal);

-- Returns the trainings of a hospital by its type

pure public getTrainingsByType: Types`Purpose ==> set of (Training)
 getTrainingsByType(s) == (
  dcl train: set of (Training);
  train := {};
  for all t in set trainings do
   if(t.getPurpose() = s)
    then train := train union {t};
  return train);

-- Returns the health professionals of a hospital

pure public getMedicalAssociated: () ==> set of (HealthProfessional)
 getMedicalAssociated() == (
    return medicalAssociated);

 -- Returns the health professionals of a hospital by its type

 pure public getMedicalAssociatedByType: Types`Type ==> set of (HealthProfessional)
  getMedicalAssociatedByType(type) == (
   dcl med: set of(HealthProfessional);
   med := {};
   for all d in set medicalAssociated do
    if(d.getType() = type)
     then med := med union {d};
   return med);
end Hospital
```

Function or operation	Line	Coverage	Calls
Hospital	17	100.0%	28
addMedAssociated	58	100.0%	44
addTask	84	100.0%	48
addTraining	106	100.0%	8
getAddress	27	100.0%	12
getAgenda	43	100.0%	123
getAgendas	39	100.0%	16
getMedicalAssociated	146	100.0%	3
getMedicalAssociatedByType	151	100.0%	176
getName	23	100.0%	40
getTasks	31	100.0%	8
getTasksByType	126	100.0%	96
getTrainings	35	100.0%	4
getTrainingsByType	136	100.0%	48
removeAgenda	52	100.0%	4
removeMedAssociated	68	100.0%	4
removeTask	96	100.0%	8
removeTraining	116	100.0%	8
Hospital.vdmpp		100.0%	678

5 Patient

```
class Patient is subclass of Person
instance variables
  private healthNumber: Types`String;

  inv len healthNumber > 5;
operations
 -- Patient constructor

 public Patient: Types`String * Types`String * Types`String * Types`String * Types`String * Types
     `String ==> Patient
  Patient(a, fn, ln, c, pn, n) == ( healthNumber := n; Person(a, fn, ln, c, pn))
 pre len n > 5
 post healthNumber = n;
 -- Returns the patient's health number

 pure public getHealthNumber : () ==> Types`String
  getHealthNumber() == (return healthNumber);

end Patient
```

Function or operation	Line	Coverage	Calls
Patient	8	100.0%	72
getHealthNumber	14	100.0%	4
Patient.vdmpp		100.0%	76

6 Person

```
class Person

instance variables
  protected address: Types`String;
  protected firstName: Types`String;
  protected lastName: Types`String;
  protected cc : Types`String;
  protected phoneNumber: Types`String;

operations
 -- Person constructor

 public Person: Types`String * Types`String * Types`String * Types`String * Types`String ==>
     Person
  Person(a, fn, ln, c, pn) == ( address := a; firstName := fn; lastName := ln; cc := c;
      phoneNumber := pn; return self)
 post address = a and firstName = fn and lastName = ln and cc = c and phoneNumber = pn;

 -- Returns the person's cc number

 pure public getCC : () ==> Types`String
  getCC() == (return cc);

 -- Returns all the person's information

 pure public getInfo: () ==> Types`String
  getInfo() == (return "Name: " ^ firstName ^ " " ^ lastName ^ "\nAddress: " ^ address ^ "\nPhone
       Number: " ^ phoneNumber ^ "\nCC: " ^ cc);

end Person
```

Function or operation	Line	Coverage	Calls
Person	12	100.0%	208
getCC	17	100.0%	780
getInfo	21	100.0%	20
Person.vdmpp		100.0%	1008

7 SafetyNetHospital

```
class SafetyNetHospital
instance variables
 private hospitals: set of (Hospital);

 inv card hospitals >= 0;
operations
 -- Safety Net constructor

 public SafetyNetHospital : () ==> SafetyNetHospital
  SafetyNetHospital() == (hospitals := {}; return self)
 post hospitals = {};

 -- Adds an hospital to the safety net's hospitals

 public addHospital : Hospital ==> ()
  addHospital(h) == (hospitals := hospitals union {h})
 pre h not in set hospitals
 post h in set hospitals;

 -- Removes an hospital from the safety net's hospitals

 public removeHospital : Hospital ==> ()
  removeHospital(h) == (hospitals := hospitals \ {h})
 pre h in set hospitals
 post h not in set hospitals;

 -- Returns all the hospitals registered

 pure public getHospitals : () ==> set of (Hospital)
  getHospitals() == (return hospitals);

 -- Returns the hospital with more tasks (by type) created

 pure public getHospitalsMoreAppointments : Types`TaskType ==> Hospital
  getHospitalsMoreAppointments(t) == (
                    dcl max: int, hosp: Hospital;
                    max := -1;
                    for all h in set hospitals do
                     if((card h.getTasksByType(t)) > max)
                      then (max := (card h.getTasksByType(t)); hosp := h);
                    return hosp);
 -- Returns the health professionals (by type) that works on more than one hospital

 pure public getMedMoreHospitals : Types`Type ==> set of(HealthProfessional)
  getMedMoreHospitals(t) == (
                  dcl doctors: set of(HealthProfessional);
                  doctors := {};
                  for all h in set hospitals do (
                   dcl med: set of (HealthProfessional), list: set of(Hospital);
                   med := h.getMedicalAssociatedByType(t);

                   list := hospitals \ {h};
                   for all m in set med do(
                    for all l in set list do
                     if(m.getType() = t and m in set l.getMedicalAssociatedByType(t) and m not in
                          set doctors)
                      then doctors := doctors union {m};
                   );
                  );

                  return doctors;
                 );
 -- Returns the health professionals by patient and type

 pure public getMedAssociatedByPatient: Patient * Types`Type ==> map Hospital to set of(
     HealthProfessional)
  getMedAssociatedByPatient(p, t) == (
                     dcl maps: map Hospital to set of(HealthProfessional), med : set of (
                         HealthProfessional);
                     maps := { |-> };
                     med := {};
                     for all h in set hospitals do (
                      for all m in set h.getMedicalAssociatedByType(t) do
                       if(p in set m.getPatients())
                        then med := med union {m};

                      maps := maps munion {h |-> med};
                      med := {};);
                      return maps);
 -- Returns the health professionals by type and hospital

 pure public getMedByHospital: Types`Type ==> map Hospital to set of(HealthProfessional)
  getMedByHospital(t) == (
                     dcl maps: map Hospital to set of(HealthProfessional);
                     maps := { |-> };
                     for all h in set hospitals do
                      maps := maps munion {h |-> h.getMedicalAssociatedByType(t)};
                     return maps);
end SafetyNetHospital
```

Function or operation	Line	Coverage	Calls
SafetyNetHospital	8	100.0%	16
addHospital	13	100.0%	28
getHospitals	25	100.0%	24
getHospitalsMoreAppointments	29	100.0%	16
getMedAssociatedByPatient	57	100.0%	8
getMedByHospital	71	100.0%	8
getMedMoreHospitals	38	100.0%	24
removeHospital	19	100.0%	8
SafetyNetHospital.vdmpp		100.0%	132

8 Schedule

```
class Schedule

instance variables
  private startHour: Types`Date;
  private endHour: Types`Date;

  inv lessThan(startHour, endHour);
operations
 -- Schedule constructor

 public Schedule: Types`Date * Types`Date ==> Schedule
  Schedule(d, d2) == (startHour := d; endHour := d2; return self)
 pre lessThan(d, d2)
```

```
post startHour = d and endHour = d2;

-- Sets the schedule start hour and endHour's values

public setSchedule : Types`Date * Types`Date ==> ()
 setSchedule(d1, d2) == (startHour := d1; endHour := d2;)
pre lessThan(d1, d2)
post startHour = d1 and endHour = d2;

-- Returns the startHour's value

pure public getScheduleStart : () ==> Types`Date
 getScheduleStart() == (return startHour);

-- Returns the endHour's value

pure public getScheduleEnd : () ==> Types`Date
 getScheduleEnd() == (return endHour);

-- Checks if two schedules overlap

pure public overlap : Schedule * Schedule ==> bool
 overlap(d1, d2) == (
         if((lessThan(d1.startHour, d2.startHour) and greaterThan(d1.endHour, d2.startHour)) or
         (not lessThan(d1.startHour, d2.startHour) and lessThan(d1.startHour, d2.endHour)))
          then return true;
         return false;);
-- Checks if a date is lower than other

pure static public lessThan : Types`Date * Types`Date ==> bool
 lessThan(d1, d2) == (
         if(d1.year < d2.year)
          then return true
         else if(d1.year > d2.year)
          then return false;
         if(d1.month < d2.month)
          then return true
         else if(d1.month > d2.month)
          then return false;
         if(d1.day < d2.day)
          then return true
         else if(d1.day > d2.day)
          then return false;
         if(d1.time.hour < d2.time.hour)
          then return true
         else if(d1.time.hour > d2.time.hour)
          then return false;
         return (d1.time.min < d2.time.min););
-- Checks if a date is greater than other

pure static public greaterThan : Types`Date * Types`Date ==> bool
 greaterThan(d1, d2) == (
         if(d1.year < d2.year)
          then return false
         else if(d1.year > d2.year)
          then return true;
         if(d1.month < d2.month)
          then return false
         else if(d1.month > d2.month)
          then return true;
         if(d1.day < d2.day)
          then return false
         else if(d1.day > d2.day)
          then return true;
         if(d1.time.hour < d2.time.hour)
```

```
           then return false
          else if(d1.time.hour > d2.time.hour)
           then return true;
          return (d1.time.min > d2.time.min););
end Schedule
```

Function or operation	Line	Coverage	Calls
Schedule	10	100.0%	108
getScheduleEnd	26	100.0%	92
getScheduleStart	22	100.0%	188
greaterThan	57	100.0%	88
lessThan	37	100.0%	936
overlap	30	100.0%	40
setSchedule	16	100.0%	4
Schedule.vdmpp		100.0%	1456

9 Specialty

```
class Specialty

instance variables
  private name: Types`String;
operations
 -- Specialty constructor

 public Specialty : Types`String ==> Specialty
  Specialty(n) == (name := n; return self)
 post name = n;

 -- Returns the specialty's name

 pure public getName : () ==> Types`String
  getName() == (return name);

end Specialty
```

Function or operation	Line	Coverage	Calls
Specialty	7	100.0%	8
getName	12	100.0%	8
Specialty.vdmpp		100.0%	16

10 Surgery

```
class Surgery is subclass of Task
instance variables
  private secondaryDoctors:set of (HealthProfessional);
  private other:set of (HealthProfessional);

  inv card secondaryDoctors >= 0;
  inv card other >= 0;
  inv medicalAssoc.getType() = <Surgeon>;
operations
 -- Surgery constructor

 public Surgery: HealthProfessional * Schedule * Patient * Hospital ==> Surgery
  Surgery(s, sch, p, h) == (medicalAssoc := s ; other := {}; secondaryDoctors := {}; Task(s, sch,
       p, h, <Surgery>))
 post medicalAssoc = s and other = {} and secondaryDoctors = {};

 -- Adds an auxiliary surgeon to the surgery

 public addSecondaryDoctor : HealthProfessional ==> ()
  addSecondaryDoctor(s) == (secondaryDoctors := secondaryDoctors union {s})
 pre s <> medicalAssoc and s.getType() = <Surgeon> and  s not in set secondaryDoctors
 post s in set secondaryDoctors;

 -- Removes an auxiliary surgeon from the surgery

 public removeSecondaryDoctor : HealthProfessional ==> ()
  removeSecondaryDoctor(s) == (secondaryDoctors := secondaryDoctors \ {s})
 pre s.getType() = <Surgeon> and s in set secondaryDoctors
 post s not in set secondaryDoctors;

 -- Adds a nurse to the surgery

 public addOther : HealthProfessional ==> ()
  addOther(s) == (other := other union {s})
 pre s.getType() = <Nurse> and s not in set other
 post s in set other;

 -- Removes a nurse from the surgery

 public removeOther : HealthProfessional ==> ()
  removeOther(s) == (other := other \ {s})
 pre s.getType() = <Nurse> and s in set other
 post s not in set other;

 -- Sets the surgery's main surgeon

 public setMainDoctor : HealthProfessional ==> ()
  setMainDoctor(s) == (medicalAssoc := s)
 pre s.getType() = <Surgeon> and s not in set secondaryDoctors;

 -- Returns the auxiliary staff of a surgery by type

 pure public getSurgeryPersons : Types`Type ==> set of (HealthProfessional)
  getSurgeryPersons(t) == (
              dcl med : set of (HealthProfessional);
              if(t = <Surgeon>)
               then med := secondaryDoctors
              else
               med := other;
              return med);
end Surgery
```

Function or operation	Line	Coverage	Calls

Surgery	11	100.0%	16
addOther	28	100.0%	4
addSecondaryDoctor	16	100.0%	4
getSurgeryPersons	45	100.0%	24
removeOther	34	100.0%	4
removeSecondaryDoctor	22	100.0%	4
setMainDoctor	40	100.0%	4
Surgery.vdmpp		100.0%	60

11 Task

```
class Task
instance variables
  protected schedule: Schedule;
  protected patient: Patient;
  protected hospital: Hospital;
  protected medicalAssoc: HealthProfessional;
  protected taskType : Types`TaskType;

  inv taskType <> nil;
operations
 -- Task constructor

 public Task: HealthProfessional * Schedule * Patient * Hospital * Types`TaskType ==> Task
  Task(med, s, p, h, t) == (schedule := s; patient := p; hospital := h; taskType := t;
      medicalAssoc := med; return self)
 pre med.getCC() <> p.getCC()
 post schedule = s and patient = p and hospital = h and medicalAssoc = med;

 -- Returns the task's schedule

 pure public getSchedule: () ==> Schedule
  getSchedule() == (return schedule);

 -- Returns the task's patient

 pure public getPatient: () ==> Patient
  getPatient() == (return patient);

 -- Returns the task's hospital

 pure public getHospital: () ==> Hospital
  getHospital() == (return hospital);

 -- Returns the task's type

 pure public getType: () ==> Types`TaskType
  getType() == (return taskType);

 -- Returns the task's health professional associated

 pure public getMedAssoc : () ==> HealthProfessional
  getMedAssoc() == (return medicalAssoc);

 -- Sets the task's schedule

 public setSchedule : Schedule ==> ()
  setSchedule(s) == (
   for all a in set hospital.getAgendas() do
    if(a.getHealthProfessional().getCC() = medicalAssoc.getCC())
     then (a.addSchedule(schedule); a.removeSchedule(s));
   schedule := s)
 pre s in set hospital.getAgenda(medicalAssoc).getAgenda()
 post s not in set hospital.getAgenda(medicalAssoc).getAgenda();

 -- Returns the surgery's associated

 pure public getSurgeryPersons : Types`Type ==> set of (HealthProfessional)
  getSurgeryPersons(t) == ( return {}; );

end Task
```

Function or operation	Line	Coverage	Calls
Task	12	100.0%	80
getHospital	26	100.0%	4
getMedAssoc	34	100.0%	536
getPatient	22	100.0%	140
getSchedule	18	100.0%	348
getSurgeryPersons	48	100.0%	4
getType	30	100.0%	488
setSchedule	38	100.0%	4
Task.vdmpp		100.0%	1604

12 Training

```
class Training

instance variables
 private medicalAssociated: HealthProfessional;
 private purpose: Types`Purpose;
 private schedule: Schedule;

 inv purpose <> nil;
operations
 -- Training constructor

 public Training: Types`Purpose * Schedule * HealthProfessional ==> Training
   Training(p, s, h) == (purpose := p; schedule := s; medicalAssociated := h; return self)
 post purpose = p and schedule = s and medicalAssociated = h;

 -- Returns the training's schedule

 pure public getSchedule : () ==> Schedule
   getSchedule() == (return schedule);

 -- Returns the training's purpose

  pure public getPurpose : () ==> Types`Purpose
  getPurpose() == (return purpose);

 -- Returns the health professional associated to the training

 pure public getMedAssoc : () ==> HealthProfessional
```

```
  getMedAssoc() == (return medicalAssociated);

 -- Sets the training's purpose

 public setPurpose : Types`Purpose ==> ()
   setPurpose(p) == (purpose := p);

 end Training
```

Function or operation	Line	Coverage	Calls
Training	11	100.0%	24
getMedAssoc	24	100.0%	120
getPurpose	20	100.0%	28
getSchedule	16	100.0%	88
setPurpose	28	100.0%	4
Training.vdmpp		100.0%	264

13 Treatment

```
class Treatment is subclass of Task
instance variables
  public med: HealthProfessional;
  public name: Types`String;

  inv med.getType() = <Nurse> or med.getType() = <Technician>;
operations
 -- Treatment constructor

 public Treatment: HealthProfessional * Types`String * Schedule * Patient * Hospital ==>
     Treatment
  Treatment(m, n, s, p, h) == (name := n; med := m; Task(m, s, p, h, <Other>))
 post name = n and med = m;

 -- Returns the treatment's name

 pure public getName: () ==> Types`String
  getName() == (return name);

 -- Returns the health professional associated to the treatment

 pure public getMed : () ==> HealthProfessional
  getMed() == (return med);

end Treatment
```

Function or operation	Line	Coverage	Calls
Treatment	9	100.0%	16
getMed	18	100.0%	4
getName	14	100.0%	4
Treatment.vdmpp		100.0%	24

14 Types

```
class Types
types
 public String = seq1 of (char);
 public Priority = <High> | <Medium> | <Low>;
 public Type = <Doctor> | <Surgeon> | <Nurse> | <Technician>;
 public TaskType = <Appointment> | <Urgencies> | <Surgery> | <Other>;
 public Purpose = <Training> | <AddSkills>;
 public Time :: hour : nat
         min: nat
 inv t == t.hour >= 0 and t.hour < 24 and t.min >= 0 and t.min < 60;
 public Date ::  year: nat1
         month: nat1
         day: nat1
         time: Time
 inv d == d.month <= 12 and d.day <= daysOfMonth(d.month, d.year);

operations
 -- Gets the days of a month

 public static pure daysOfMonth : nat1 * nat1 ==> nat1
  daysOfMonth(month, year) == (
              if(month = 1 or month = 3 or month = 5 or month = 7 or month = 8 or month = 10 or
                  month = 12)
               then return 31
              else if(month = 4 or month = 6 or month = 9 or month = 11)
               then return 30
              else if(month = 2)
               then if ((year mod 4) = 0 and (year mod 100) <> 0 or (year mod 400) = 0)
                then return 29;
              return 28;);

end Types
```

Function or operation	Line	Coverage	Calls
daysOfMonth	19	100.0%	92
Types.vdmpp		100.0%	92

15 PersonTest

```
class PersonTest
instance variables
 private patient: Patient := new Patient("Rua 1 Maio", "Rui", "Andrade", "123456789", "223456111"
     , "0987654321");
 private doctor: HealthProfessional := new HealthProfessional("Rua de Cima", "Ana", "Marques", "
     123432156", "921349076", "111111222", <Doctor>);
 private surgeon: HealthProfessional := new HealthProfessional("Rua 2", "Diogo", "Viana", "
     234512389", "921349134", "111111232", <Surgeon>);
 private nurse: HealthProfessional := new HealthProfessional("Rua de Baixo", "Lisete", "Antunes",
      "123444654", "921378643", "111222333", <Nurse>);
 private technician: HealthProfessional := new HealthProfessional("Rua Antero Marques", "Inłs",
     "Pinto", "123432151", "921348765", "123432578", <Technician>);
operations
```

```
private assertTrue: bool ==> ()
 assertTrue(cond) == return
pre cond;


public testGetInformation: () ==> ()
 testGetInformation() == (
  IO`print("\n Getting patient informations \n");
  assertTrue(patient.getHealthNumber() = "0987654321");
  assertTrue(patient.getCC() = "123456789");
  assertTrue(patient.getInfo() = "Name: " ^ "Rui" ^ " " ^ "Andrade" ^ "\nAddress: " ^ "Rua 1
      Maio" ^ "\nPhone Number: " ^ "223456111" ^ "\nCC: " ^ "123456789");

  IO`print("\n Getting doctor informations \n");
  assertTrue(doctor.getMedicalNumber() = "111111222");
  assertTrue(doctor.getCC() = "123432156");
  assertTrue(doctor.getInfo() = "Name: " ^ "Ana" ^ " " ^ "Marques" ^ "\nAddress: " ^ "Rua de
      Cima" ^ "\nPhone Number: " ^ "921349076" ^ "\nCC: " ^ "123432156");
  assertTrue(doctor.getType() = <Doctor>);

  IO`print("\n Getting surgeon informations \n");
  assertTrue(surgeon.getMedicalNumber() = "111111232");
  assertTrue(surgeon.getCC() = "234512389");
  assertTrue(surgeon.getInfo() = "Name: " ^ "Diogo" ^ " " ^ "Viana" ^ "\nAddress: " ^ "Rua 2" ^
      "\nPhone Number: " ^ "921349134" ^ "\nCC: " ^ "234512389");
  assertTrue(surgeon.getType() = <Surgeon>);

  IO`print("\n Getting nurse informations \n");
  assertTrue(nurse.getMedicalNumber() = "111222333");
  assertTrue(nurse.getCC() = "123444654");
  assertTrue(nurse.getInfo() = "Name: " ^ "Lisete" ^ " " ^ "Antunes" ^ "\nAddress: " ^ "Rua de
      Baixo" ^ "\nPhone Number: " ^ "921378643" ^ "\nCC: " ^ "123444654");
  assertTrue(nurse.getType() = <Nurse>);

  IO`print("\n Getting technician informations \n");
  assertTrue(technician.getMedicalNumber() = "123432578");
  assertTrue(technician.getCC() = "123432151");
  assertTrue(technician.getInfo() = "Name: " ^ "Inłs" ^ " " ^ "Pinto" ^ "\nAddress: " ^ "Rua
      Antero Marques" ^ "\nPhone Number: " ^ "921348765" ^ "\nCC: " ^ "123432151");
  assertTrue(technician.getType() = <Technician>);
 );


public testAddRemovePatient : () ==> ()
 testAddRemovePatient() == (
  IO`print("\n Number of patients: ");
  IO`print(card doctor.getPatients());
  assertTrue(card doctor.getPatients() = 0);

  IO`print("\n Adding a patient \n");
  doctor.addPatient(patient);

  IO`print("\n Number of patients: ");
  IO`print(card doctor.getPatients());
  assertTrue(card doctor.getPatients() = 1);

  IO`print("\n Removing a patient \n");
  doctor.removePatient(patient);

  IO`print("\n Number of patients: ");
  IO`print(card doctor.getPatients());
  assertTrue(card doctor.getPatients() = 0);

  IO`print("\n Adding a patient \n");
  assertTrue(card surgeon.getPatients() = 0);
```

```
   surgeon.addPatient(patient);
   IO`print("\n Number of patients: ");
   IO`print(card surgeon.getPatients());
   assertTrue(card surgeon.getPatients() = 1);
  );


 public testAddRemoveSpecialty : () ==> ()
  testAddRemoveSpecialty() == (
   dcl specialty1: Specialty := new Specialty("General"), specialty2: Specialty := new Specialty(
       "Cardio");

   IO`print("\n Number of specialties: ");
   IO`print(card doctor.getSpecialties());
   assertTrue(card doctor.getSpecialties() = 0);

   IO`print("\n Adding a specialty \n");
   doctor.addSpecialty(specialty1);

   IO`print("\n Number of specialties: ");
   IO`print(card doctor.getSpecialties());

   assertTrue(specialty1.getName() = "General");
   assertTrue(card doctor.getSpecialties() = 1);
   assertTrue(doctor.getSpecialties() = {specialty1});

   IO`print("\n Adding a specialty \n");
   doctor.addSpecialty(specialty2);

   IO`print("\n Number of specialties: ");
   IO`print(card doctor.getSpecialties());

   assertTrue(specialty2.getName() = "Cardio");
   assertTrue(card doctor.getSpecialties() = 2);
   assertTrue(doctor.getSpecialties() = {specialty1, specialty2});

   IO`print("\n Removing a specialty \n");
   doctor.removeSpecialty(specialty1);

   IO`print("\n Number of specialties: ");
   IO`print(card doctor.getSpecialties());

   assertTrue(card doctor.getSpecialties() = 1);
   assertTrue(doctor.getSpecialties() = {specialty2});
  );


 public static main: () ==> ()
   main() == (
    dcl personTest: PersonTest := new PersonTest();
    IO`print("\n *****Running PersonTest***** \n");
    personTest.testGetInformation();
    personTest.testAddRemovePatient();
    personTest.testAddRemoveSpecialty();
   );

end PersonTest
```

Function or operation	Line	Coverage	Calls

assertTrue	9	100.0%	264
main	112	100.0%	4
testAddRemovePatient	45	100.0%	4
testAddRemoveSpecialty	74	100.0%	4
testGetInformation	13	100.0%	4
PersonTest.vdmpp		100.0%	280

16 RunTests

```
class RunTests

operations

 public static main: () ==> ()
   main() == (
    dcl taskTest: TaskTest := new TaskTest(), personTest: PersonTest := new PersonTest(),
     trainingTest: TrainingTest := new TrainingTest(), safetyNetTest: SafetyNetHospitalTest :=
         new SafetyNetHospitalTest();

    personTest.main();
    taskTest.main();
    trainingTest.main();
    safetyNetTest.main();
    IO`print("\n\n ====== All TaskTest run successfully ====== \n\n");
    IO`print("\n\n ====== All TrainingTesr run successfully ====== \n\n");
    IO`print("\n\n ====== All PersonTest run successfully ====== \n\n");
   IO`print("\n\n ====== All SafetyNetHospitalTest run successfully ====== \n\n");
   );
end RunTests
```

Function or operation	Line	Coverage	Calls
main	4	100.0%	4
RunTests.vdmpp		100.0%	4

17 SafetyNetHospitalTest

```
class SafetyNetHospitalTest
instance variables
 private safetyNet: SafetyNetHospital := new SafetyNetHospital();

 private time1: Types`Time := mk_Types`Time(12, 10);
 private date1: Types`Date := mk_Types`Date(2017, 12, 25, time1);
 private time2: Types`Time := mk_Types`Time(12, 30);
 private date2: Types`Date := mk_Types`Date(2017, 12, 25, time2);
 private schedule: Schedule := new Schedule(date1, date2);

 private time3: Types`Time := mk_Types`Time(12, 15);
 private date3: Types`Date := mk_Types`Date(2017, 12, 25, time3);
 private time4: Types`Time := mk_Types`Time(12, 35);
 private date4: Types`Date := mk_Types`Date(2017, 12, 25, time4);
 private schedule2: Schedule := new Schedule(date3, date4);
 private time5: Types`Time := mk_Types`Time(12, 40);
 private date5: Types`Date := mk_Types`Date(2017, 12, 25, time5);
 private time6: Types`Time := mk_Types`Time(12, 50);
 private date6: Types`Date := mk_Types`Date(2017, 12, 25, time6);
 private schedule3: Schedule := new Schedule(date5, date6);

 private time7: Types`Time := mk_Types`Time(12, 10);
 private date7: Types`Date := mk_Types`Date(2017, 11, 22, time7);
 private time8: Types`Time := mk_Types`Time(12, 30);
 private date8: Types`Date := mk_Types`Date(2017, 11, 22, time8);
 private schedule4: Schedule := new Schedule(date7, date8);

 private time9: Types`Time := mk_Types`Time(12, 35);
 private date9: Types`Date := mk_Types`Date(2017, 11, 23, time9);
 private time10: Types`Time := mk_Types`Time(12, 45);
 private date10: Types`Date := mk_Types`Date(2017, 11, 23, time10);
 private schedule5: Schedule := new Schedule(date9, date10);

 private patient: Patient := new Patient("Rua 1 Maio", "Rui", "Andrade", "123456789", "223456111"
     , "0987654321");
 private patient2: Patient := new Patient("Rua 1 Maio", "Diogo", "Andrade", "123321123", "
     911112345", "908765123");
 private patient3: Patient := new Patient("Rua 1 Maio", "Vitor", "Andrade", "135790864", "
     912345334", "123432130");
 private patient4: Patient := new Patient("Rua 1 Maio", "Simone", "Andrade", "234123765", "
     931238654", "0987654143");

  private hospital: Hospital := new Hospital("Hospital das Cam lias", "Rua de Cima", safetyNet);

  private doctor: HealthProfessional := new HealthProfessional("Rua de Cima", "Ana", "Marques", "
      123432156", "921349076", "111111222", <Doctor>);
  private doctor2: HealthProfessional := new HealthProfessional("Rua de Cima", "Anabela", "
      Marques", "123432157", "921349077", "111111223", <Doctor>);
  private surgeon: HealthProfessional := new HealthProfessional("Rua 2", "Diogo", "Viana", "
      234512389", "921349134", "111111232", <Surgeon>);
  private secSurgeon: HealthProfessional := new HealthProfessional("Rua 2", "Diana", "Viana", "
      234512390", "921349135", "111111235", <Surgeon>);
 private nurse: HealthProfessional := new HealthProfessional("Rua de Baixo", "Lisete", "Antunes",
      "123444654", "921378643", "111222333", <Nurse>);
 private technician: HealthProfessional := new HealthProfessional("Rua de Baixo", "Lu s", "
     Antunes", "123444655", "921377654", "111222345", <Technician>);

 private appointment: Appointment := new Appointment(doctor, schedule, patient, hospital);
 private appointment2: Appointment := new Appointment(doctor, schedule3, patient4, hospital);
 private appointment3: Appointment := new Appointment(doctor2, schedule3, patient, hospital);
 private urgencies: Appointment := new Appointment(doctor2, <High>, schedule, patient2, hospital)
     ;
 private surgery: Surgery := new Surgery(surgeon, schedule, patient3, hospital);
 private treatment: Treatment := new Treatment(technician, "Fisioterapia", schedule, patient4,
     hospital);

 private purpose: Types`Purpose := <Training>;
 private training : Training := new Training(purpose, schedule3, nurse);
 private train : Training := new Training(purpose, schedule4, doctor);
operations

 private assertTrue: bool ==> ()
  assertTrue(cond) == return
 pre cond;


 public testAddRemoveHospitals: () ==> ()
  testAddRemoveHospitals() == (
      dcl h1: Hospital, h2: Hospital, h3: Hospital;
```

```
    h1 := new Hospital("Hospital dos Lusadas", "Rua de Cima", safetyNet);
    h2 := new Hospital("Hospital Novo", "Rua 1 de Maio", safetyNet);
    h3 := new Hospital("Hospital da Trofa", "Rua da Trofa", safetyNet);
    IO`print("\n Number of hospitals: ");
    IO`print(card safetyNet.getHospitals());

    IO`print("\n\n Getting hospitals information \n");
    assertTrue(h1.getName() = "Hospital dos Lusadas");
    assertTrue(h2.getName() = "Hospital Novo");
    assertTrue(h3.getName() = "Hospital da Trofa");

    assertTrue(h1.getAddress() = "Rua de Cima");
    assertTrue(h2.getAddress() = "Rua 1 de Maio");
    assertTrue(h3.getAddress() = "Rua da Trofa");

    IO`print("\n Removing hospitals \n");
    assertTrue(card safetyNet.getHospitals() = 4);
    safetyNet.removeHospital(h1);

    IO`print("\n Removing hospitals \n");
    assertTrue(card safetyNet.getHospitals() = 3);
    safetyNet.removeHospital(h2);
    assertTrue(card safetyNet.getHospitals() = 2);

    IO`print("\n Number of hospitals: ");
    IO`print(card safetyNet.getHospitals());
);


public testAddRemoveMedHospital : () ==> ()
testAddRemoveMedHospital() == (
  dcl agenda1 : Agenda, agenda2 : Agenda, agenda3 : Agenda, agenda4: Agenda, agenda5: Agenda;

  IO`print("\n Adding health professionals \n");
  hospital.addMedAssociated(doctor);
  hospital.addMedAssociated(doctor2);
  hospital.addMedAssociated(surgeon);
  hospital.addMedAssociated(nurse);
  hospital.addMedAssociated(technician);

  IO`print("\n Adding agendas to health professionals \n");
  for all a in set hospital.getAgendas() do(
   if(a.getHealthProfessional() = doctor)
    then agenda1 := a
   else if(a.getHealthProfessional() = doctor2)
    then agenda2 := a
   else if(a.getHealthProfessional() = surgeon)
    then agenda3 := a
   else if(a.getHealthProfessional() = nurse)
    then agenda4 := a
   else
    agenda5 := a;);

   IO`print("\n Checking agenda \n");
   assertTrue(hospital.getAgenda(doctor) = agenda1);

   IO`print("\n Adding schedules to agendas \n");
   agenda1.addSchedule(schedule);
   agenda1.addSchedule(schedule3);
   agenda1.addSchedule(schedule4);

   agenda2.addSchedule(schedule);
   agenda2.addSchedule(schedule3);

   agenda3.addSchedule(schedule);
```

```
    agenda4.addSchedule(schedule3);

    agenda5.addSchedule(schedule);

    IO`print("\n Checking agendas \n");
    assertTrue(card agenda1.getAgenda() = 3);
    assertTrue(card agenda2.getAgenda() = 2);
    assertTrue(card agenda3.getAgenda() = 1);
    assertTrue(card agenda4.getAgenda() = 1);
    assertTrue(card agenda5.getAgenda() = 1);

    IO`print("\n Total number of doctors: ");
    IO`print(card hospital.getMedicalAssociatedByType(<Doctor>));
    IO`print("\n Total number of surgeons: ");
    IO`print(card hospital.getMedicalAssociatedByType(<Surgeon>));
    IO`print("\n Total number of nurses: ");
    IO`print(card hospital.getMedicalAssociatedByType(<Nurse>));
    IO`print("\n Total number of technicians: ");
    IO`print(card hospital.getMedicalAssociatedByType(<Technician>));

    assertTrue(card hospital.getMedicalAssociatedByType(<Doctor>) = 2);
    assertTrue(card hospital.getMedicalAssociatedByType(<Surgeon>) = 1);
    assertTrue(card hospital.getMedicalAssociatedByType(<Nurse>) = 1);
    assertTrue(card hospital.getMedicalAssociatedByType(<Technician>) = 1);

    IO`print("\n Total number of doctors: ");
    IO`print(card hospital.getMedicalAssociatedByType(<Doctor>));

    assertTrue(card hospital.getMedicalAssociatedByType(<Doctor>) = 2);

    assertTrue(card hospital.getMedicalAssociated() = 5);

    IO`print("\n Removing a doctor \n");
    hospital.addTask(appointment);
    hospital.addTraining(train);
    hospital.removeMedAssociated(doctor);
    assertTrue(card hospital.getMedicalAssociatedByType(<Doctor>) = 1);

    IO`print("\n Total number of doctors: ");
    IO`print(card hospital.getMedicalAssociatedByType(<Doctor>));

    hospital.addMedAssociated(doctor);

    for all a in set hospital.getAgendas() do
     if(a.getHealthProfessional().getCC() = doctor.getCC())

      then agenda1 := a;

    agenda1.addSchedule(schedule);
    agenda1.addSchedule(schedule3);
    assertTrue(card agenda1.getAgenda() = 2);

 );

   public testAddRemoveTaskHospital : () ==> ()
   testAddRemoveTaskHospital() == (
    IO`print("\n Adding tasks \n");
     hospital.addTask(appointment);
     hospital.addTask(appointment2);
     hospital.addTask(appointment3);
     hospital.addTask(urgencies);
     hospital.addTask(surgery);
     hospital.addTask(treatment);
     IO`print("\n\n Total number of appointments: ");
     IO`print(card hospital.getTasksByType(<Appointment>));
     IO`print("\n Total number of urgencies: ");
     IO`print(card hospital.getTasksByType(<Urgencies>));
     IO`print("\n Total number of surgeries: ");
     IO`print(card hospital.getTasksByType(<Surgery>));
     IO`print("\n Total number of other treatments: ");
     IO`print(card hospital.getTasksByType(<Other>));

     IO`print("\n\n Total number of tasks: ");
     IO`print(card hospital.getTasks());

     assertTrue(card hospital.getTasks() = 6);

     assertTrue(card hospital.getTasksByType(<Appointment>) = 3);
     assertTrue(card hospital.getTasksByType(<Urgencies>) = 1);
     assertTrue(card hospital.getTasksByType(<Surgery>) = 1);
     assertTrue(card hospital.getTasksByType(<Other>) = 1);

     IO`print("\n\n Removing an appointment \n");
     hospital.removeTask(appointment);
     assertTrue(card hospital.getTasksByType(<Appointment>) = 2);

     IO`print("\n Total number of appointments: ");
     IO`print(card hospital.getTasksByType(<Appointment>));


     IO`print("\n Adding an appointment \n");
     hospital.addTask(appointment);
     assertTrue(card hospital.getTasksByType(<Appointment>) = 3);

     IO`print("\n Total number of appointments: ");
     IO`print(card hospital.getTasksByType(<Appointment>));
 );

 public testAddRemoveTrainingHospital : () ==> ()
  testAddRemoveTrainingHospital() == (
   IO`print("\n\n Total number of trainings: ");
   IO`print(card hospital.getTrainingsByType(<Training>) + card hospital.getTrainingsByType(<
       AddSkills>));

   assertTrue(card hospital.getTrainingsByType(<Training>) = 0);
   assertTrue(card hospital.getTrainingsByType(<AddSkills>) = 0);

   IO`print("\n Adding a training \n");
   hospital.addTraining(training);
   assertTrue(card hospital.getTrainingsByType(<Training>) = 1);

   IO`print("\n Total number of trainings: ");
   IO`print(card hospital.getTrainingsByType(<Training>) + card hospital.getTrainingsByType(<
       AddSkills>));

   assertTrue(card hospital.getTrainings() = (card hospital.getTrainingsByType(<Training>) +
       card hospital.getTrainingsByType(<AddSkills>)));


   IO`print("\n Removing a training \n");
   hospital.removeTraining(training);
   assertTrue(card hospital.getTrainingsByType(<Training>) = 0);

   IO`print("\n\n Total number of trainings: ");
   IO`print(card hospital.getTrainingsByType(<Training>) + card hospital.getTrainingsByType(<
       AddSkills>));
 );
```

```
public testGetHospitalsMoreAppointments : () ==> ()
 testGetHospitalsMoreAppointments() == (
  IO`print("\n Checking Safety Net Hospitals with more appointments, etc \n");
  assertTrue(safetyNet.getHospitalsMoreAppointments(<Appointment>).getName() = "Hospital das
      Camlias");
  assertTrue(safetyNet.getHospitalsMoreAppointments(<Urgencies>).getName() = "Hospital das
      Camlias");
  assertTrue(safetyNet.getHospitalsMoreAppointments(<Surgery>).getName() = "Hospital das
      Camlias");
  assertTrue(safetyNet.getHospitalsMoreAppointments(<Other>).getName() = "Hospital das Camlias
      ");
 );

public testGetMedMoreHospitals : () ==> ()
 testGetMedMoreHospitals() == (
  for all t in set safetyNet.getHospitals() do
   if(t.getName() <> "Hospital das Camlias")

    then t.addMedAssociated(doctor);

  IO`print("\n Checking Safety Net Doctors that works in more than 1 hospital \n");
  IO`print("\n Number of Doctors: ");
  IO`print(card safetyNet.getMedMoreHospitals(<Doctor>));
  assertTrue(card safetyNet.getMedMoreHospitals(<Doctor>) = 1);
  assertTrue(safetyNet.getMedMoreHospitals(<Doctor>) = {doctor});
 );

public testGetMedAssociatedByPatient : () ==> ()

 testGetMedAssociatedByPatient() == (
  dcl mapTest : map Hospital to set of (HealthProfessional);
  IO`print("\n\n Getting Doctors associated by patient by hospital \n");
  mapTest := safetyNet.getMedAssociatedByPatient(patient, <Doctor>);

  assertTrue(card mapTest(hospital) = 2);
  assertTrue(mapTest(hospital) = {doctor, doctor2});
 );

public testGetMedByHospital : () ==> ()
 testGetMedByHospital() == (
  dcl mapTest : map Hospital to set of (HealthProfessional);
  IO`print("\n\n Getting Doctors associated by hospital \n");
  mapTest := safetyNet.getMedByHospital(<Doctor>);


  assertTrue(card mapTest(hospital) = 2);
  assertTrue(mapTest(hospital) = {doctor, doctor2});

  mapTest := safetyNet.getMedByHospital(<Surgeon>);

  assertTrue(card mapTest(hospital) = 1);
  assertTrue(mapTest(hospital) = {surgeon});
 );

public static main: () ==> ()
 main() == (
  dcl safetyNetTest: SafetyNetHospitalTest := new SafetyNetHospitalTest();
  IO`print("\n *****Running SafetyNetHospitalTest***** \n");
  safetyNetTest.testAddRemoveHospitals();
  safetyNetTest.testAddRemoveMedHospital();
  safetyNetTest.testAddRemoveTaskHospital();
  safetyNetTest.testAddRemoveTrainingHospital();
  safetyNetTest.testGetHospitalsMoreAppointments();
  safetyNetTest.testGetMedMoreHospitals();
```

```
    safetyNetTest.testGetMedAssociatedByPatient();
    safetyNetTest.testGetMedByHospital();
   );

end SafetyNetHospitalTest
```

Function or operation	Line	Coverage	Calls
assertTrue	60	100.0%	186
main	292	100.0%	3
testAddRemoveHospitals	64	100.0%	4
testAddRemoveMedHospital	95	100.0%	3
testAddRemoveTaskHospital	177	100.0%	3
testAddRemoveTrainingHospital	220	100.0%	4
testGetHospitalsMoreAppointments	245	100.0%	3
testGetMedAssociatedByPatient	267	100.0%	3
testGetMedByHospital	277	100.0%	3
testGetMedMoreHospitals	254	100.0%	3
SafetyNetHospitalTest.vdmpp		100.0%	215

18 TaskTest

```
class TaskTest

instance variables
 private safetyNet: SafetyNetHospital := new SafetyNetHospital();

 private time1: Types`Time := mk_Types`Time(12, 10);
 private date: Types`Date := mk_Types`Date(2017, 11, 25, time1);
 private d: Types`Date := mk_Types`Date(2017, 2, 25, time1);
 private d2: Types`Date := mk_Types`Date(2016, 2, 25, time1);
 private date1: Types`Date := mk_Types`Date(2017, 12, 25, time1);
 private time2: Types`Time := mk_Types`Time(12, 30);
 private date2: Types`Date := mk_Types`Date(2017, 12, 25, time2);
 private schedule: Schedule := new Schedule(date1, date2);

 private time3: Types`Time := mk_Types`Time(12, 15);
 private date3: Types`Date := mk_Types`Date(2017, 12, 25, time3);
 private time4: Types`Time := mk_Types`Time(12, 35);
 private date4: Types`Date := mk_Types`Date(2017, 12, 25, time4);
 private schedule2: Schedule := new Schedule(date3, date4);

 private time5: Types`Time := mk_Types`Time(12, 40);
 private date5: Types`Date := mk_Types`Date(2017, 12, 25, time5);
 private time6: Types`Time := mk_Types`Time(12, 50);
 private date6: Types`Date := mk_Types`Date(2017, 12, 25, time6);
 private schedule3: Schedule := new Schedule(date5, date6);

 private time7: Types`Time := mk_Types`Time(12, 10);
 private date7: Types`Date := mk_Types`Date(2018, 11, 22, time7);
 private time8: Types`Time := mk_Types`Time(12, 30);
 private date8: Types`Date := mk_Types`Date(2018, 11, 22, time8);
 private schedule4: Schedule := new Schedule(date7, date8);

 private time9: Types`Time := mk_Types`Time(12, 35);
 private date9: Types`Date := mk_Types`Date(2017, 11, 22, time9);
 private time10: Types`Time := mk_Types`Time(12, 45);
 private date10: Types`Date := mk_Types`Date(2017, 11, 22, time10);
 private schedule5: Schedule := new Schedule(date9, date10);

 private patient: Patient := new Patient("Rua 1 Maio", "Rui", "Andrade", "123456789", "223456111"
     , "0987654321");
 private patient2: Patient := new Patient("Rua 1 Maio", "Diogo", "Andrade", "123321123", "
     911112345", "908765123");
 private patient3: Patient := new Patient("Rua 1 Maio", "Vitor", "Andrade", "135790864", "
     912345334", "123432130");
 private patient4: Patient := new Patient("Rua 1 Maio", "Simone", "Andrade", "234123765", "
     931238654", "0987654143");

  private hospital: Hospital := new Hospital("Hospital dos Lus adas", "Rua de Cima", safetyNet);
  private doctor: HealthProfessional := new HealthProfessional("Rua de Cima", "Ana", "Marques", "
      123432156", "921349076", "111111222", <Doctor>);
  private doctor2: HealthProfessional := new HealthProfessional("Rua de Cima", "Anabela", "
      Marques", "123432157", "921349077", "111111223", <Doctor>);
  private surgeon: HealthProfessional := new HealthProfessional("Rua 2", "Diogo", "Viana", "
      234512389", "921349134", "111111232", <Surgeon>);
  private secSurgeon: HealthProfessional := new HealthProfessional("Rua 2", "Diana", "Viana", "
      234512390", "921349135", "111111235", <Surgeon>);
 private nurse: HealthProfessional := new HealthProfessional("Rua de Baixo", "Lisete", "Antunes",
      "123444654", "921378643", "111222333", <Nurse>);
 private technician: HealthProfessional := new HealthProfessional("Rua de Baixo", "Lu s", "
     Antunes", "123444655", "921377654", "111222345", <Technician>);

 private appointment: Appointment := new Appointment(doctor, schedule, patient, hospital);
 private urgencies: Appointment := new Appointment(doctor2, <High>, schedule, patient2, hospital)
     ;
 private surgery: Surgery := new Surgery(surgeon, schedule3, patient3, hospital);
 private treatment: Treatment := new Treatment(technician, "Fisioterapia", schedule, patient4,
     hospital);
operations
 private assertTrue: bool ==> ()
  assertTrue(cond) == return

 pre cond;

 public testGetsSetsTask : () ==> ()
  testGetsSetsTask() == (

   dcl agenda1 : Agenda, agenda2 : Agenda, agenda3 : Agenda, agenda4: Agenda;
   hospital.addMedAssociated(doctor);
   hospital.addMedAssociated(doctor2);
   hospital.addMedAssociated(surgeon);
   hospital.addMedAssociated(technician);

   for all a in set hospital.getAgendas() do(
    if(a.getHealthProfessional() = doctor)
     then agenda1 := a
    else if(a.getHealthProfessional() = doctor2)
     then agenda2 := a
    else if(a.getHealthProfessional() = surgeon)
     then agenda3 := a
    else
     agenda4 := a;);

   assertTrue(hospital.getAgenda(doctor).getAgenda() = {});
   assertTrue(card hospital.getAgenda(doctor).getAgenda() = 0);

   agenda1.addSchedule(schedule);
   agenda1.addSchedule(schedule3);
```

```
assertTrue(agenda1.getHealthProfessional().getCC() = doctor.getCC());

assertTrue(card agenda1.getAgenda() = 2);

agenda2.addSchedule(schedule);

assertTrue(card agenda2.getAgenda() = 1);

agenda3.addSchedule(schedule3);

assertTrue(card agenda3.getAgenda() = 1);

agenda4.addSchedule(schedule);

assertTrue(card agenda4.getAgenda() = 1);

agenda4.removeSchedule(schedule);

assertTrue(card agenda4.getAgenda() = 0);

agenda4.addSchedule(schedule);

assertTrue(card agenda4.getAgenda() = 1);

hospital.addTask(appointment);
hospital.addTask(urgencies);
hospital.addTask(surgery);
hospital.addTask(treatment);

IO`print("\n Getting appointment informations \n");
assertTrue(appointment.getPatient().getCC() = "123456789");
assertTrue(appointment.getHospital().getName() = "Hospital dos Lus adas");

assertTrue(appointment.getType() = <Appointment>);
assertTrue(urgencies.getType() = <Urgencies>);
assertTrue(surgery.getType() = <Surgery>);
assertTrue(treatment.getType() = <Other>);

IO`print("\n Getting tasks informations \n");
assertTrue(appointment.getMedAssoc().getCC() = "123432156");
assertTrue(card appointment.getSurgeryPersons(<Nurse>) = 0);
assertTrue(urgencies.getMedAssoc().getCC() = "123432157");
assertTrue(surgery.getMedAssoc().getCC() = "234512389");

IO`print("\n Checking schedules \n");
assertTrue(appointment.getSchedule().getScheduleStart().year = 2017);
assertTrue(appointment.getSchedule().getScheduleStart().month = 12);
assertTrue(appointment.getSchedule().getScheduleStart().day = 25);
assertTrue(appointment.getSchedule().getScheduleStart().time.hour = 12);
assertTrue(appointment.getSchedule().getScheduleStart().time.min = 10);

assertTrue(appointment.getSchedule().getScheduleEnd().year = 2017);
assertTrue(appointment.getSchedule().getScheduleEnd().month = 12);
assertTrue(appointment.getSchedule().getScheduleEnd().day = 25);
assertTrue(appointment.getSchedule().getScheduleEnd().time.hour = 12);
assertTrue(appointment.getSchedule().getScheduleEnd().time.min = 30);

assertTrue(appointment.getSchedule().lessThan(appointment.getSchedule().getScheduleStart(),
    appointment.getSchedule().getScheduleEnd()) = true);

appointment.getSchedule().setSchedule(date3, date4);
assertTrue(appointment.getSchedule().getScheduleStart().year = 2017);
assertTrue(appointment.getSchedule().getScheduleStart().month = 12);
assertTrue(appointment.getSchedule().getScheduleStart().day = 25);
assertTrue(appointment.getSchedule().getScheduleStart().time.hour = 12);
```

```
  assertTrue(appointment.getSchedule().getScheduleStart().time.min = 15);

  assertTrue(appointment.getSchedule().getScheduleEnd().year = 2017);
  assertTrue(appointment.getSchedule().getScheduleEnd().month = 12);
  assertTrue(appointment.getSchedule().getScheduleEnd().day = 25);
  assertTrue(appointment.getSchedule().getScheduleEnd().time.hour = 12);
  assertTrue(appointment.getSchedule().getScheduleEnd().time.min = 35);

  appointment.setSchedule(schedule3);

  assertTrue(appointment.getSchedule().getScheduleStart().year = 2017);
  assertTrue(appointment.getSchedule().getScheduleStart().month = 12);
  assertTrue(appointment.getSchedule().getScheduleStart().day = 25);
  assertTrue(appointment.getSchedule().getScheduleStart().time.hour = 12);
  assertTrue(appointment.getSchedule().getScheduleStart().time.min = 40);

  assertTrue(appointment.getSchedule().getScheduleEnd().year = 2017);
  assertTrue(appointment.getSchedule().getScheduleEnd().month = 12);
  assertTrue(appointment.getSchedule().getScheduleEnd().day = 25);
  assertTrue(appointment.getSchedule().getScheduleEnd().time.hour = 12);
  assertTrue(appointment.getSchedule().getScheduleEnd().time.min = 50);
 );

public testAppointment : () ==> ()
 testAppointment() == (

  IO`print("\n Checking appointment priority \n");
  assertTrue(appointment.getPriority() = <Medium>);
  assertTrue(urgencies.getPriority() = <High>);

  urgencies.setPriority(<Low>);
  assertTrue(urgencies.getPriority() = <Low>);
 );

public testSurgery: () ==> ()
 testSurgery() == (
  IO`print("\n Checking surgery informations \n");
  assertTrue(card surgery.getSurgeryPersons(<Surgeon>) = 0);

  surgery.addSecondaryDoctor(secSurgeon);
  assertTrue(card surgery.getSurgeryPersons(<Surgeon>) = 1);

  surgery.removeSecondaryDoctor(secSurgeon);
  assertTrue(card surgery.getSurgeryPersons(<Surgeon>) = 0);

  assertTrue(card surgery.getSurgeryPersons(<Nurse>) = 0);
  surgery.addOther(nurse);
  assertTrue(card surgery.getSurgeryPersons(<Nurse>) = 1);

  surgery.removeOther(nurse);
  assertTrue(card surgery.getSurgeryPersons(<Nurse>) = 0);

  assertTrue(surgery.getMedAssoc().getCC() = "234512389");
  surgery.setMainDoctor(secSurgeon);
  assertTrue(surgery.getMedAssoc().getCC() = "234512390");
);

public testTreatment : () ==> ()
 testTreatment() == (
  IO`print("\n Checking treatment informations \n");
  IO`print("\n Checking schedule functions \n");
  assertTrue(treatment.getName() = "Fisioterapia");
  assertTrue(treatment.getMed().getCC() = "123444655");
 );
```

```
  public testScheduleFunctions: () ==> ()
  testScheduleFunctions() == (
   dcl sch : Schedule, sch1 : Schedule, sch2 : Schedule, dateNew: Types`Date, dateNew2 : Types`
       Date;
   dateNew := mk_Types`Date(2017, 10, 25, time1);
   dateNew2 := mk_Types`Date(2017, 10, 25, time2);
   sch := new Schedule(dateNew, dateNew2);

   dateNew := mk_Types`Date(2017, 10, 26, time1);
   dateNew2 := mk_Types`Date(2017, 10, 26, time2);
   sch1 := new Schedule(dateNew, dateNew2);

   dateNew := mk_Types`Date(2017, 11, 26, time1);

   dateNew2 := mk_Types`Date(2017, 11, 26, time2);
   sch2 := new Schedule(dateNew, dateNew2);

   IO`print("\n Checking schedule functions \n");
   assertTrue(appointment.getSchedule().lessThan(appointment.getSchedule().getScheduleStart(),
       appointment.getSchedule().getScheduleEnd()));
   assertTrue(appointment.getSchedule().greaterThan(appointment.getSchedule().getScheduleEnd(),
       appointment.getSchedule().getScheduleStart()));

   assertTrue(appointment.getSchedule().lessThan(appointment.getSchedule().getScheduleStart(),
       schedule4.getScheduleStart()));
   assertTrue(not(schedule4.lessThan(schedule4.getScheduleStart(), schedule5.getScheduleStart()))
       );
   assertTrue(sch.lessThan(sch.getScheduleStart(), schedule.getScheduleStart()));
   assertTrue(not(sch1.lessThan(sch1.getScheduleStart(), sch.getScheduleStart())));
   assertTrue(not(schedule3.lessThan(schedule3.getScheduleStart(), sch2.getScheduleStart())));
   assertTrue(sch.lessThan(sch.getScheduleStart(), sch1.getScheduleStart()));

   assertTrue(appointment.getSchedule().greaterThan(schedule4.getScheduleStart(), schedule5.
       getScheduleStart()));
   assertTrue(not(appointment.getSchedule().greaterThan(appointment.getSchedule().
       getScheduleStart(), schedule4.getScheduleStart())));
   assertTrue(not(sch.greaterThan(sch.getScheduleStart(), schedule.getScheduleStart())));
   assertTrue(sch1.greaterThan(sch1.getScheduleStart(), sch.getScheduleStart()));
   assertTrue(schedule.greaterThan(schedule.getScheduleStart(), sch.getScheduleStart()));
   assertTrue(not(sch.greaterThan(sch.getScheduleStart(), sch1.getScheduleStart())));

   IO`print("\n Checking overlap \n");
   assertTrue(schedule.overlap(schedule, schedule2));

  );

  public static main: () ==> ()
   main() == (
    dcl taskTest: TaskTest := new TaskTest();
    IO`print("\n\n *****Running TaskTest***** \n");
    taskTest.testGetsSetsTask();
    taskTest.testAppointment();

    taskTest.testSurgery();
    taskTest.testTreatment();
    taskTest.testScheduleFunctions();
   );

end TaskTest
```

Function or operation	Line	Coverage	Calls

assertTrue	59	100.0%	156
main	293	100.0%	2
testAppointment	174	100.0%	2
testGetsSetsTask	63	100.0%	2
testScheduleFunctions	256	100.0%	2
testSurgery	225	100.0%	2
testTreatment	248	100.0%	2
TaskTest.vdmpp		100.0%	168

19 TrainingTest

```
class TrainingTest
instance variables
 private doctor: HealthProfessional := new HealthProfessional("Rua de Cima", "Ana", "Marques", "
     123432156", "921349076", "111111222", <Doctor>);
  private purpose: Types`Purpose := <Training>;

 private time1: Types`Time := mk_Types`Time(12, 10);
 private date1: Types`Date := mk_Types`Date(2017, 12, 25, time1);
 private time2: Types`Time := mk_Types`Time(12, 30);
 private date2: Types`Date := mk_Types`Date(2017, 12, 25, time2);
 private schedule: Schedule := new Schedule(date1, date2);

 private time3: Types`Time := mk_Types`Time(12, 15);
 private date3: Types`Date := mk_Types`Date(2017, 12, 25, time3);
 private time4: Types`Time := mk_Types`Time(12, 35);
 private date4: Types`Date := mk_Types`Date(2017, 12, 25, time4);
 private schedule2: Schedule := new Schedule(date3, date4);

 private training : Training := new Training(purpose, schedule, doctor);
operations


 private assertTrue: bool ==> ()
  assertTrue(cond) == return
 pre cond;


 public testGetsSets : () ==> ()
  testGetsSets() == (
   IO`print("\n Testing Training gets and sets \n");
   assertTrue(training.getPurpose() = <Training>);
   assertTrue(training.getMedAssoc().getCC() = "123432156");

   training.setPurpose(<AddSkills>);
   assertTrue(training.getPurpose() = <AddSkills>);

   assertTrue(training.getSchedule().getScheduleStart().year = 2017);
   assertTrue(training.getSchedule().getScheduleStart().month = 12);
   assertTrue(training.getSchedule().getScheduleStart().day = 25);
   assertTrue(training.getSchedule().getScheduleStart().time.hour = 12);
   assertTrue(training.getSchedule().getScheduleStart().time.min = 10);

   assertTrue(training.getSchedule().getScheduleEnd().year = 2017);
   assertTrue(training.getSchedule().getScheduleEnd().month = 12);
   assertTrue(training.getSchedule().getScheduleEnd().day = 25);
   assertTrue(training.getSchedule().getScheduleEnd().time.hour = 12);
   assertTrue(training.getSchedule().getScheduleEnd().time.min = 30);
  );


 public static main: () ==> ()
   main() == (
    dcl trainingTest: TrainingTest := new TrainingTest();
    IO`print("\n *****Running TrainingTest***** \n");
    trainingTest.testGetsSets();
   );
end TrainingTest
```

Function or operation	Line	Coverage	Calls
assertTrue	21	100.0%	52
main	47	100.0%	2
testGetsSets	25	100.0%	2
TrainingTest.vdmpp		100.0%	56