MFES

December 23, 2017

Contents

1 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", "I n s", "
     Pinto", "123432151", "921348765", "123432578", <Technician>);
operations

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


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

   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>);

   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>);

   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>);

   assertTrue(technician.getMedicalNumber() = "123432578");
   assertTrue(technician.getCC() = "123432151");
   assertTrue(technician.getInfo() = "Name: " ^ "I n s" ^ " " ^ "Pinto" ^ "\nAddress: " ^ "Rua
       Antero Marques" ^ "\nPhone Number: " ^ "921348765" ^ "\nCC: " ^ "123432151");
   assertTrue(technician.getType() = <Technician>);
  );


 public testAddRemovePatient : () ==> ()
  testAddRemovePatient() == (
   assertTrue(card doctor.getPatients() = 0);

   doctor.addPatient(patient);
   assertTrue(card doctor.getPatients() = 1);

   doctor.removePatient(patient);
   assertTrue(card doctor.getPatients() = 0);
   assertTrue(card surgeon.getPatients() = 0);

   surgeon.addPatient(patient);
   assertTrue(card surgeon.getPatients() = 1);
  );


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

   assertTrue(card doctor.getSpecialties() = 0);

   doctor.addSpecialty(specialty1);
   assertTrue(specialty1.getName() = "General");

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

   doctor.addSpecialty(specialty2);
   assertTrue(specialty2.getName() = "Cardio");

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

   doctor.removeSpecialty(specialty1);
   assertTrue(card doctor.getSpecialties() = 1);
   assertTrue(doctor.getSpecialties() = {specialty2});
  );


 public static main: () ==> ()
   main() == (
    dcl personTest: PersonTest := new PersonTest();
    personTest.testGetInformation();
    personTest.testAddRemovePatient();
    personTest.testAddRemoveSpecialty();
   );

end PersonTest
```

Function or operation	Line	Coverage	Calls
assertTrue	9	100.0%	297
main	79	100.0%	9
testAddRemovePatient	40	100.0%	9
testAddRemoveSpecialty	56	100.0%	9
testGetInformation	13	100.0%	9
PersonTest.vdmpp		100.0%	333

2 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();
   );
end RunTests
```

Function or operation	Line	Coverage	Calls
main	4	100.0%	9
RunTests.vdmpp		100.0%	9

3 SafetyNetHospitalTest

```
class SafetyNetHospitalTest
types
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 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 Camlias", "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, schedule, 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 Lus adas", "Rua de Cima", safetyNet);
      h2 := new Hospital("Hospital Novo", "Rua 1 de Maio", safetyNet);
      h3 := new Hospital("Hospital da Trofa", "Rua da Trofa", safetyNet);

      assertTrue(h1.getName() = "Hospital dos Lus adas");
      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");


      assertTrue(card safetyNet.getHospitals() = 4);
      safetyNet.removeHospital(h1);
      assertTrue(card safetyNet.getHospitals() = 3);
      safetyNet.removeHospital(h2);
      assertTrue(card safetyNet.getHospitals() = 2);
  );

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


      hospital.removeTask(appointment);
      assertTrue(card hospital.getTasksByType(<Appointment>) = 2);

      hospital.addTask(appointment);
      assertTrue(card hospital.getTasksByType(<Appointment>) = 3);
```

```
);

public testAddRemoveMedHospital : () ==> ()
testAddRemoveMedHospital() == (
    assertTrue(card hospital.getMedicalAssociatedByType(<Doctor>) = 0);
    assertTrue(card hospital.getMedicalAssociatedByType(<Surgeon>) = 0);
    assertTrue(card hospital.getMedicalAssociatedByType(<Nurse>) = 0);
    assertTrue(card hospital.getMedicalAssociatedByType(<Technician>) = 0);

    hospital.addMedAssociated(doctor);
    assertTrue(card hospital.getMedicalAssociatedByType(<Doctor>) = 1);
    hospital.addTraining(training);

    hospital.removeMedAssociated(doctor);
    assertTrue(card hospital.getMedicalAssociatedByType(<Doctor>) = 0);


    hospital.addMedAssociated(doctor);
    hospital.addMedAssociated(surgeon);
    hospital.addMedAssociated(nurse);
    hospital.addMedAssociated(technician);
);

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


  hospital.addTraining(training);
  assertTrue(card hospital.getTrainingsByType(<Training>) = 1);

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


public testOverlap : () ==> ()
 testOverlap() == (
  assertTrue(hospital.overlap(schedule, schedule2));
  assertTrue(hospital.overlap(schedule, schedule3) = false);
 );

public testGetHospitalsMoreAppointments : () ==> ()

 testGetHospitalsMoreAppointments() == (
  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);

  assertTrue(card safetyNet.getMedMoreHospitals(<Doctor>) = 1);
  assertTrue(safetyNet.getMedMoreHospitals(<Doctor>) = {doctor});
 );

 public testGetMedAssociatedByPatient : () ==> ()
  testGetMedAssociatedByPatient() == (
   dcl mapTest : map Hospital to set of (HealthProfessional);


   mapTest := safetyNet.getMedAssociatedByPatient(patient, <Doctor>);

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

 public testGetMedByHospital : () ==> ()
  testGetMedByHospital() == (
   dcl mapTest : map Hospital to set of (HealthProfessional);
   mapTest := safetyNet.getMedByHospital(<Doctor>);

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


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

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

 public static main: () ==> ()
  main() == (
   dcl safetyNetTest: SafetyNetHospitalTest := new SafetyNetHospitalTest();
   safetyNetTest.testAddRemoveHospitals();
   safetyNetTest.testAddRemoveTaskHospital();
   safetyNetTest.testAddRemoveTrainingHospital();
   safetyNetTest.testAddRemoveMedHospital();
   safetyNetTest.testOverlap();
   safetyNetTest.testGetHospitalsMoreAppointments();
   safetyNetTest.testGetMedMoreHospitals();
   safetyNetTest.testGetMedAssociatedByPatient();
   safetyNetTest.testGetMedByHospital();
  );

end SafetyNetHospitalTest
```

Function or operation	Line	Coverage	Calls
assertTrue	41	100.0%	117
main	161	100.0%	8
testAddRemoveHospitals	45	100.0%	7
testAddRemoveMedHospital	81	100.0%	3
testAddRemoveTaskHospital	67	100.0%	3
testAddRemoveTrainingHospital	101	100.0%	3
testGetHospitalsMoreAppointments	119	100.0%	3
testGetMedAssociatedByPatient	137	100.0%	3
testGetMedByHospital	147	100.0%	3
testGetMedMoreHospitals	127	100.0%	3
testOverlap	113	100.0%	4
SafetyNetHospitalTest.vdmpp		100.0%	157

4 TaskTest

```
class TaskTest

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 patient: Patient := new Patient("Rua 1 Maio", "Rui", "Andrade", "123456789", "223456111
      ", "0987654321");
  private hospital: Hospital := new Hospital("Hospital dos Lusadas", "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, patient, hospital);
 private surgery: Surgery := new Surgery(surgeon, schedule, patient, hospital);
 private treatment: Treatment := new Treatment(technician, "Fisioterapia", schedule, patient,
     hospital);

 private medicament: Medicament := new Medicament("Brufen");
 private prescription: Prescription := new Prescription("123");
operations

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


 public testGetsSetsTask : () ==> ()
  testGetsSetsTask() == (
   assertTrue(appointment.getPatient().getCC() = "123456789");
   assertTrue(appointment.getHospital().getName() = "Hospital dos Lusadas");

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

   assertTrue(appointment.getMedAssoc().getCC() = "123432156");
   assertTrue(urgencies.getMedAssoc().getCC() = "123432157");
   assertTrue(surgery.getMedAssoc().getCC() = "234512389");
   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().compareDate(appointment.getSchedule().getScheduleStart(),
        appointment.getSchedule().getScheduleEnd()) = false);
   assertTrue(appointment.getSchedule().compareDateLess(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(schedule2);

   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);
  );


 public testAppointment : () ==> ()
  testAppointment() == (
   assertTrue(appointment.getPriority() = <Medium>);
   assertTrue(urgencies.getPriority() = <High>);

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

   assertTrue(card appointment.getPrescriptions() = 0);
   assertTrue(card urgencies.getPrescriptions() = 0);

   assertTrue(medicament.getName() = "Brufen");
   assertTrue(prescription.getCode() = "123");
   assertTrue(card prescription.getMedicaments() = 0);

   prescription.addMedicament(medicament);
   assertTrue(card prescription.getMedicaments() = 1);
   assertTrue(prescription.getMedicaments() = {medicament});
```

```
   prescription.removeMedicament(medicament);
   assertTrue(card prescription.getMedicaments() = 0);
   assertTrue(prescription.getMedicaments() = {});

   appointment.addPrescription(prescription);
   urgencies.addPrescription(prescription);
   assertTrue(card appointment.getPrescriptions() = 1);
   assertTrue(card urgencies.getPrescriptions() = 1);

   appointment.removePrescription(prescription);
   urgencies.removePrescription(prescription);
   assertTrue(card appointment.getPrescriptions() = 0);
   assertTrue(card urgencies.getPrescriptions() = 0);
  );


 public testSurgery: () ==> ()
  testSurgery() == (
   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.getMainDoctor().getCC() = "234512389");
   surgery.setMainDoctor(secSurgeon);
   assertTrue(surgery.getMainDoctor().getCC() = "234512390");
 );


 public testTreatment: () ==> ()
  testTreatment() == (
   assertTrue(treatment.getName() = "Fisioterapia");
   assertTrue(treatment.getMed().getCC() = "123444655");
  );


  public static main: () ==> ()
   main() == (
    dcl taskTest: TaskTest := new TaskTest();
    taskTest.testGetsSetsTask();
    taskTest.testAppointment();
    taskTest.testSurgery();
    taskTest.testTreatment();
   );

end TaskTest
```

Function or operation	Line	Coverage	Calls
assertTrue	34	100.0%	603
main	157	100.0%	9

testAppointment	95	100.0%	9
testGetsSetsTask	38	100.0%	27
testSurgery	129	100.0%	9
testTreatment	151	100.0%	45
TaskTest.vdmpp		100.0%	702

5 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() == (
   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);

   training.setSchedule(schedule2);

   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 = 15);

   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 = 35);
  );


 public static main: () ==> ()
   main() == (
    dcl trainingTest: TrainingTest := new TrainingTest();
    trainingTest.testGetsSets();
   );
end TrainingTest
```

Function or operation	Line	Coverage	Calls
assertTrue	21	100.0%	414
main	60	100.0%	9
testGetsSets	25	100.0%	9
TrainingTest.vdmpp		100.0%	432

6 Appointment

```
class Appointment is subclass of Task

instance variables
  private prescriptions:set of (Prescription);
  private priority : Types`Priority;

  inv priority <> nil;
  inv card prescriptions >= 0;
  inv medicalAssoc.getType() = <Doctor>;
operations

 public Appointment: HealthProfessional * Schedule * Patient * Hospital==> Appointment
  Appointment(d, s, p, h) == (medicalAssoc := d; priority := <Medium>; prescriptions := {}; Task(
      d, s, p, h, <Appointment>))
 post medicalAssoc = d and prescriptions = {} and priority = <Medium>;

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


 pure public getPriority : () ==> Types`Priority
  getPriority() == (return priority);


  pure public getPrescriptions : () ==> set of (Prescription)
```

```
   getPrescriptions() == (return prescriptions);


  public setPriority : Types`Priority ==> ()
   setPriority(p) == (priority := p)
  pre type = <Urgencies>;


  public addPrescription : Prescription ==> ()
   addPrescription(p) == (prescriptions := prescriptions union {p})
  pre p not in set prescriptions
  post p in set prescriptions;


 public removePrescription : Prescription ==> ()
   removePrescription(p) == (prescriptions := prescriptions \ {p})
  pre p in set prescriptions
  post p not in set prescriptions;

end Appointment
```

Function or operation	Line	Coverage	Calls
Appointment	11	100.0%	36
addPrescription	30	100.0%	18
getPrescriptions	23	100.0%	54
getPriority	20	100.0%	27
removePrescription	35	100.0%	18
setPriority	26	100.0%	18
Appointment.vdmpp		100.0%	171

7 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;
operations

 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
 post medicalNumber = s and type = t and specialties = {} and patients = {};


 pure public getMedicalNumber: () ==> Types`String
  getMedicalNumber() == (return medicalNumber);
 pure public getSpecialties: () ==> set of (Specialty)
  getSpecialties() == (return specialties);


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


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


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


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


 public addPatient : Patient ==> ()
  addPatient(p) == (patients :=  patients union {p})
 pre p not in set patients
 post p in set 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	13	100.0%	312
addPatient	40	100.0%	179
addSpecialty	35	100.0%	18
getMedicalNumber	18	100.0%	36
getPatients	24	100.0%	399
getSpecialties	21	100.0%	63
getType	27	100.0%	617
removePatient	45	100.0%	9
removeSpecialty	30	100.0%	9
HealthProfessional.vdmpp		100.0%	1642

8 Hospital

```
class Hospital
```

```
instance variables
  private medicalAssociated: set of (HealthProfessional);
  private name: Types`String;
  private address: Types`String;
  private tasks: set of(Task);
  private trainings: set of(Training);
  private safetyNet: [SafetyNetHospital];

 inv safetyNet <> nil;
 inv card medicalAssociated >= 0;
 inv card tasks >= 0;
operations


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


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


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


 public addMedAssociated: HealthProfessional ==> ()
  addMedAssociated(d) == (medicalAssociated := {d} union medicalAssociated)
 pre d not in set medicalAssociated
 post d in set medicalAssociated;


 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);
               medicalAssociated := medicalAssociated \ {d})
 pre d in set medicalAssociated
 post d not in set medicalAssociated;


 public addTask: Task ==> ()
  addTask(d) == (
         if(d.getPatient() not in set d.getMedAssoc().getPatients())
          then d.getMedAssoc().addPatient(d.getPatient());
         tasks := {d} union tasks)
 pre d not in set tasks and forall t in set tasks &
  not (overlap(d.getSchedule(), t.getSchedule()) and (d.getMedAssoc().getCC() = t.getMedAssoc().
      getCC()
  and d.getPatient().getCC() = t.getPatient().getCC() and d.getMedAssoc().getCC() = t.getPatient
      ().getCC() and d.getPatient().getCC() = t.getMedAssoc().getCC()))
  and forall tr in set trainings & not (overlap(d.getSchedule(), tr.getSchedule()) and (d.
      getMedAssoc().getCC() = tr.getMedAssoc().getCC()))
 post d in set tasks and d.getPatient() in set d.getMedAssoc().getPatients();
 public removeTask: Task ==> ()
  removeTask(d) == (tasks := tasks \ {d})
 pre d in set tasks
 post d not in set tasks;


 public addTraining: Training ==> ()
  addTraining(d) == (trainings := {d} union trainings)
 pre d not in set trainings and forall t in set trainings & not (overlap(d.getSchedule(), t.
     getSchedule()))
 and forall tr in set tasks & not (overlap(d.getSchedule(), tr.getSchedule()) and (d.getMedAssoc
     ().getCC() = tr.getMedAssoc().getCC()

 or d.getMedAssoc().getCC() = tr.getPatient().getCC()))
 post d in set trainings;

 public removeTraining: Training ==> ()
  removeTraining(d) == (trainings := trainings \ {d})

 pre d in set trainings
 post d not in set trainings;

 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);

 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);

 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);

 pure public overlap: Schedule * Schedule ==> bool
  overlap(t1, t2) == (
             if(t1.compareDate(t1.getScheduleStart(), t2.getScheduleStart())
              or (not t1.compareDateLess(t1.getScheduleStart(), t2.getScheduleStart())
              or not t1.compareDateLess(t1.getScheduleEnd(), t2.getScheduleStart())))
              then return true
             else
              return false);

end Hospital
```

Function or operation	Line	Coverage	Calls
Hospital	15	100.0%	64
addMedAssociated	27	100.0%	48
addTask	44	66.6%	0
addTraining	59	32.6%	0
getAddress	24	100.0%	27
getMedicalAssociatedByType	89	100.0%	40
getName	21	100.0%	84
getTasksByType	69	100.0%	36
getTrainingsByType	79	100.0%	2
overlap	99	100.0%	32
removeMedAssociated	32	100.0%	8
removeTask	54	100.0%	6
removeTraining	64	100.0%	4
Hospital.vdmpp		82.2%	351

9 Medicament

```
class Medicament

instance variables
  private name:Types`String;
operations

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


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

end Medicament
```

Function or operation	Line	Coverage	Calls
Medicament	6	100.0%	18
getName	10	100.0%	9
Medicament.vdmpp		100.0%	27

10 Patient

```
class Patient is subclass of Person
instance variables
```

```
  private healthNumber: Types`String;
operations

 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))
 post healthNumber = n;


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

end Patient
```

Function or operation	Line	Coverage	Calls
Patient	5	100.0%	109
getHealthNumber	9	100.0%	9
Patient.vdmpp		100.0%	118

11 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

 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;


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


 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	10	100.0%	421
getCC	14	100.0%	989
getInfo	17	100.0%	45

Person.vdmpp		100.0%	1455

12 Prescription

```
class Prescription

instance variables
  private medicaments:set of (Medicament);
  private code:Types`String;

operations

 public Prescription: Types`String ==> Prescription
  Prescription(c) == (code := c; medicaments := {}; return self)
 post code = c and medicaments = {};


 pure public getCode : () ==> Types`String
  getCode() == (return code);


 public addMedicament: Medicament ==> ()
  addMedicament(m) == (medicaments := {m} union medicaments)
 pre m not in set medicaments
 post m in set medicaments;


 public removeMedicament: Medicament ==> ()
  removeMedicament(m) == (medicaments := medicaments \ {m})
 pre m in set medicaments
 post m not in set medicaments;


 pure public getMedicaments: () ==> set of (Medicament)
  getMedicaments() == (return medicaments);

end Prescription
```

Function or operation	Line	Coverage	Calls
Prescription	8	100.0%	18
addMedicament	15	100.0%	9
getCode	12	100.0%	9
getMedicaments	25	100.0%	45
removeMedicament	20	100.0%	9
Prescription.vdmpp		100.0%	90

13 SafetyNetHospital

```
class SafetyNetHospital
instance variables
 private hospitals: set of (Hospital);
 inv card hospitals >= 0;
operations


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


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


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


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

 -- Mudar --

 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);


 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;
                );


 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);


 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%	37
addHospital	12	100.0%	64
getHospitals	22	100.0%	35
getHospitalsMoreAppointments	26	100.0%	32
getMedAssociatedByPatient	54	100.0%	16
getMedByHospital	68	100.0%	16
getMedMoreHospitals	35	100.0%	32
removeHospital	17	100.0%	36
SafetyNetHospital.vdmpp		100.0%	268

14 Schedule

```
class Schedule

types
instance variables
  private startHour: Types'Date;
  private endHour: Types'Date;

  inv compareDateLess(startHour, endHour) = true and startHour.year = endHour.year and startHour.
      month = endHour.month and startHour.day = endHour.day;
operations

 public Schedule: Types'Date * Types'Date ==> Schedule
  Schedule(d, d2) == (startHour := d; endHour := d2; return self)
 pre compareDateLess(d, d2)
 post startHour = d and endHour = d2;


 public setSchedule : Types'Date * Types'Date ==> ()
  setSchedule(d1, d2) == (startHour := d1; endHour := d2;)
 pre compareDateLess(d1, d2);


 pure public getScheduleStart : () ==> Types'Date
```

```
  getScheduleStart() == (return startHour);


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


 pure public compareDateLess : Types`Date * Types`Date ==> bool
  compareDateLess(d1, d2) == (return (d1.year <= d2.year and d1.month <= d2.month and d1.day <=
      d2.day and d1.time.hour <= d2.time.hour and d1.time.min < d2.time.min));


 pure public compareDate : Types`Date * Types`Date ==> bool
  compareDate(d1, d2) == (return (d1.year = d2.year and d1.month = d2.month and d1.day = d2.day
      and d1.time.hour = d2.time.hour and d1.time.min = d2.time.min));

end Schedule
```

Function or operation	Line	Coverage	Calls
Schedule	10	100.0%	128
compareDate	28	100.0%	343
compareDateLess	25	100.0%	450
getScheduleEnd	22	100.0%	314
getScheduleStart	19	100.0%	1156
setSchedule	15	100.0%	9
Schedule.vdmpp		100.0%	2400

15 Specialty

```
class Specialty

instance variables
  private name: Types`String;
operations

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


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

end Specialty
```

Function or operation	Line	Coverage	Calls
Specialty	6	100.0%	18
getName	10	100.0%	18
Specialty.vdmpp		100.0%	36

16 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;
operations

 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 = {};


 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;


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


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


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


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


 public getMainDoctor : () ==> HealthProfessional
  getMainDoctor() == (return medicalAssoc);


 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	9	100.0%	36
addOther	23	100.0%	9
addSecondaryDoctor	13	100.0%	9
getMainDoctor	37	100.0%	18
getSurgeryPersons	40	100.0%	54
removeOther	28	100.0%	9
removeSecondaryDoctor	18	100.0%	9
setMainDoctor	33	100.0%	9
Surgery.vdmpp		100.0%	153

17 Task

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

  inv patient <> nil;
  inv hospital <> nil;
  inv type <> nil;
operations

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


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


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


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


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


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


 public setSchedule : Schedule ==> ()
  setSchedule(s) == (schedule := s);

end Task
```

Function or operation	Line	Coverage	Calls
Task	13	100.0%	161
getHospital	25	100.0%	9
getMedAssoc	31	100.0%	1091
getPatient	22	100.0%	530
getSchedule	19	100.0%	969
getType	28	100.0%	483
setSchedule	34	100.0%	9
Task.vdmpp		100.0%	3252

18 Training

```
class Training

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

 inv medicalAssociated <> nil;
 inv purpose <> nil;
 inv schedule <> nil;

operations

 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;


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


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


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


 public setSchedule : Schedule ==> ()
   setSchedule(s) == (schedule := s);


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

 end Training
```

Function or operation	Line	Coverage	Calls
Training	13	100.0%	36
getMedAssoc	23	100.0%	17
getPurpose	20	100.0%	26
getSchedule	17	100.0%	180
setPurpose	29	100.0%	9
setSchedule	26	100.0%	9
Training.vdmpp		100.0%	277

19 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


 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;


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


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

end Treatment
```

Function or operation	Line	Coverage	Calls
Treatment	9	100.0%	36
getMed	16	100.0%	9
getName	13	100.0%	9
Treatment.vdmpp		100.0%	54

20 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 <= 31;
end Types
```

Function or operation	Line	Coverage	Calls
Types.vdmpp		100.0%	0