MFES

December 21, 2017

Contents

1 Appointment 1

2 Hospital 2

3 MedicalAssociated 5

4 Medicament 6

5 Patient 6

6 Person 6

7 Prescription 7

8 SafetyNetHospital 8

9 Schedule 9

10 Specialty 10

11 Surgery 10

12 Task 11

13 Training 12

14 Treatment 13

15 Types 14

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


 public Appointment: MedicalAssociated * Types`Priority * Schedule * Patient * Hospital ==>
     Appointment
  Appointment(d, p, s, pat, h) == (medicalAssoc := d; priority := p; prescriptions := {}; Task(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);


  pure public getPrescription : seq of (char) ==> Prescription
   getPrescription(code) == (
               dcl prescription: Prescription;

                for all p in set prescriptions do
                 if(p.compare(code))
                  then prescription := p;


                return prescription;
               )
  pre code <> [];

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

  pure public addPrescription : Prescription ==> set of (Prescription)
   addPrescription(p) == (return prescriptions union {p})

  pre p not in set prescriptions
  post p in set prescriptions;

 pure public removePrescription : Prescription ==> set of (Prescription)

   removePrescription(p) == (return prescriptions \ {p})
  pre p in set prescriptions
  post p not in set prescriptions;

end Appointment
```

2 Hospital

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

 inv name <> [] and address <> [];
 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 := {}; return self)
 pre n <> [] and a <> [] and 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);

 pure public getSafetyNet: () ==> SafetyNetHospital

  getSafetyNet() == (return safetyNet);

 pure public addMedAssociated: MedicalAssociated ==> set of (MedicalAssociated)

  addMedAssociated(d) == (return ({d} union medicalAssociated))
 pre d not in set medicalAssociated
 post d in set medicalAssociated;

 pure public removeMedAssociated: MedicalAssociated ==> set of (MedicalAssociated)

  removeMedAssociated(d) == (return (medicalAssociated \ {d}))
 pre d in set medicalAssociated
 post d not in set medicalAssociated;

 public addTask: Task ==> set of (Task)

  addTask(d) == (return ({d} union tasks))
 pre d not in set tasks and forall t in set tasks &
  not (overlap(d, t) and not (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()))
 post d in set tasks;

 pure public removeTask: Task ==> set of (Task)

  removeTask(d) == (return (tasks \ {d}))
 pre d in set tasks
 post d not in set tasks;


 public addTraining: Training ==> set of (Training)

  addTraining(d) == (return ({d} union trainings))
 pre d not in set trainings and forall t in set trainings & not (overlapTraining(d, t))
 post d in set trainings;
```

```
pure public removeTraining: Training ==> set of (Training)
 removeTraining(d) == (return (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);
             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);
             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 (MedicalAssociated)
 getMedicalAssociatedByType(type) == (
          dcl med: set of(MedicalAssociated);
          for all d in set medicalAssociated do
           if(d.getType() = type)
            then med := med union {d};


          return med);

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


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

end Hospital
```

3 MedicalAssociated

```
class MedicalAssociated 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 medicalNumber <> [];
 inv type <> nil;
operations
 public MedicalAssociated: Types`String * Types`Type ==> MedicalAssociated
  MedicalAssociated(s, t) == (medicalNumber := s; type := t; specialties := {}; patients := {};
      return self)
 pre s <> [] and 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);


 pure public removeSpecialty: Specialty ==> set of(Specialty)
  removeSpecialty(s) == (return specialties \ {s})
 pre s in set specialties

 post s not in set specialties;

 pure public addSpecialty: Specialty ==> set of(Specialty)
  addSpecialty(s) == (return specialties union {s})
 pre s not in set specialties

 post s in set specialties;

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

 post p in set patients;

 public removePatient : Patient ==> set of(Patient)
  removePatient(p) == (return patients \ {p})
 pre p in set patients

 post p not in set patients;

end MedicalAssociated
```

4 Medicament

```
class Medicament

instance variables
  private name:Types`String;
  inv name <> [];
operations
 public Medicament: Types`String ==> Medicament
  Medicament(n) == (name := n; return self)

 pre n <> []
 post name = n;

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


end Medicament
```

5 Patient

```
class Patient is subclass of Person
instance variables
  private healthNumber: Types`String;
  inv healthNumber <> [];
operations
 public Patient: Types`String ==> Patient
  Patient(n) == ( healthNumber := n; return self)
 pre n <> []

 post healthNumber = n;

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


end Patient
```

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;

  inv address <> [] and firstName <> [] and lastName <> [] and cc <> [] and len cc = 9 and
      phoneNumber <> [] and len phoneNumber = 9;
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)

 pre a <> [] and fn <> [] and ln <> [] and c <> [] and pn <> []
 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
```

7 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)

 pre c <> []
 post code = c and medicaments = {};

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


 pure public addMedicament: Medicament ==> set of (Medicament)
  addMedicament(m) == (return ({m} union medicaments))

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

 pre m in set medicaments
 post m not in set medicaments;

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


 pure public compare: Types`String ==> bool
  compare(c) == (return c = code);


end Prescription
```

8 SafetyNetHospital

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

 inv card hospitals >= 0;
operations

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

 post hospitals = {};

 pure public addHospital : Hospital ==> set of (Hospital)
  addHospital(h) == (return hospitals union {h})

 pre h not in set hospitals
 post h in set hospitals;

 pure public removeHospital : Hospital ==> set of (Hospital)
  removeHospital(h) == (return hospitals \ {h})

 pre h in set hospitals
 post h not in set hospitals;

 pure public numHospitals : () ==> nat
  numHospitals() == (return card hospitals);


 -- Mudar --
 pure public getHospitalsMoreAppointments : () ==> Hospital
  getHospitalsMoreAppointments() == (

                    dcl max: nat, hosp: Hospital;
                    max := 0;
                    for all h in set hospitals do
                     if(card (h.getTasksByType(<Appointment>)) > max)
                      then (max := card (h.getTasksByType(<Appointment>)); hosp := h);
                    return hosp);

 pure public getDoctorsMoreHospitals : () ==> set of(MedicalAssociated)
  getDoctorsMoreHospitals() == (
                  dcl doctors: set of(MedicalAssociated);
                  for all h in set hospitals do (
                   dcl med: set of (MedicalAssociated), list: set of(Hospital);
                   med := h.getMedicalAssociatedByType(<Doctor>);

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

                  return doctors;
                 );

end SafetyNetHospital
```

9 Schedule

```
class Schedule

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

  inv compareDateLess(startHour, endHour) = true;
operations

 public Schedule: Types`Date ==> Schedule
  Schedule(d) == (startHour := d; return self)
 post startHour = d;

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


 public setEndHour : Types`Date ==> Types`Date
  setEndHour(d) == (endHour := d; return endHour)
 pre compareDateLess(startHour, d);

 public setStartHour : Types`Date ==> Types`Date
  setStartHour(d) == (startHour := d; return startHour)
 pre compareDateLess(d, endHour);


 public setSchedule : Types`Date * Types`Date ==> Schedule
  setSchedule(d1, d2) == (startHour := d1; endHour := d2; return self)
 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.hour < d2.hour and d1.min < d2.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.hour = d2.hour and d1.min = d2.min));

end Schedule
```

10 Specialty

```
class Specialty

instance variables
  private name: Types`String;
  inv name <> [];
operations
 public Specialty : Types`String ==> Specialty
  Specialty(n) == (name := n; return self)

 pre n <> []
 post name = n;

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


end Specialty
```

11 Surgery

```
class Surgery is subclass of Task

instance variables
  private secondaryDoctors:set of (MedicalAssociated);
  private other:set of (MedicalAssociated);

  inv card secondaryDoctors >= 0;
  inv card other >= 0;
operations

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

 post medicalAssoc = s and other = {} and secondaryDoctors = {};

 pure public addSecondaryDoctor : MedicalAssociated ==> set of (MedicalAssociated)
  addSecondaryDoctor(s) == (return secondaryDoctors union {s})

 pre s <> medicalAssoc and s.getType() = <Surgeon> and  s not in set secondaryDoctors
 post s in set secondaryDoctors;

 pure public removeSecondaryDoctor : MedicalAssociated ==> set of (MedicalAssociated)
  removeSecondaryDoctor(s) == (return secondaryDoctors \ {s})

 pre s.getType() = <Surgeon> and s in set secondaryDoctors
 post s not in set secondaryDoctors;

 pure public addOther : MedicalAssociated ==> set of (MedicalAssociated)
  addOther(s) == (return other union {s})

 pre s.getType() = <Nurse> and s not in set other
 post s in set other;

 pure public removeOther : MedicalAssociated ==> set of (MedicalAssociated)
  removeOther(s) == (return other \ {s})

 pre s.getType() = <Nurse> and s in set other
 post s not in set other;

 public setMainDoctor : MedicalAssociated ==> ()
  setMainDoctor(s) == (medicalAssoc := s)

 pre s.getType() = <Surgeon> and s not in set secondaryDoctors;

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


 -- Mudar --
 public getSurgeryPersons : () ==> seq of (set of (MedicalAssociated))
  getSurgeryPersons() == (

              dcl med : seq of (set of (MedicalAssociated));
              med := med ^ [secondaryDoctors] ^ [other];
              return med);

end Surgery
```

12 Task

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

  inv schedule <> nil;
  inv patient <> nil;
  inv hospital <> nil;
  inv type <> nil;
  inv medicalAssoc.getCC() <> patient.getCC();


operations
 public Task: Schedule * Patient * Hospital * Types`TaskType ==> Task
  Task(s, p, h, t) == (schedule := s; patient := p; hospital := h; type := t; medicalAssoc := nil
      ; return self)
 post schedule = s and patient = p and hospital = h and medicalAssoc = nil;

 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 : () ==> MedicalAssociated

  getMedAssoc() == (return medicalAssoc);

 public setSchedule : Schedule ==> ()

  setSchedule(s) == (schedule := s);

 public setPatient : Patient ==> ()

  setPatient(s) == (patient := s);

 public setHospital : Hospital ==> ()

  setHospital(s) == (hospital := s);

 public setMedAssoc : MedicalAssociated ==> ()
  setMedAssoc(s) == (medicalAssoc := s);

end Task
```

13 Training

```
class Training

instance variables
 public medicalAssociated:set of (MedicalAssociated);
 public purpose:[Types`Purpose];
 public schedule:[Schedule];

 inv card medicalAssociated > 1 and card medicalAssociated < 10;
 inv purpose <> nil;
 inv schedule <> nil;

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


 pure public getSchedule : () ==> Schedule
```

```
   getSchedule() == (return schedule);


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


  pure public addMedicalAssociated: MedicalAssociated ==> set of (MedicalAssociated)
   addMedicalAssociated(m) == (return medicalAssociated union {m})
  pre m not in set medicalAssociated

 post m in set medicalAssociated;

  pure public removeMedicalAssociated: MedicalAssociated ==> set of (MedicalAssociated)
   removeMedicalAssociated(m) == (return medicalAssociated \ {m})
  pre m in set medicalAssociated

  post m not in set medicalAssociated;

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


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


 end Training
```

14 Treatment

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

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

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

 pre n <> []
 post name = n;

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


 public setMed: MedicalAssociated ==> ()
  setMed(t) == (med := t; return);


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


end Treatment
```

15 Types

```
class Types
types
 public String = seq of (char);
 public Priority = <High> | <Medium> | <Low>;
 public Type = <Doctor> | <Surgeon> | <Nurse> | <Technician>;
 public TaskType = <Appointment> | <Urgencies> | <Surgery> | <Other>;
 public Purpose = <Training> | <AddSkills>;
 public Date ::  year: nat1
         month: nat1
         day: nat1
         hour: nat
         min: nat
 inv d == d.month <= 12 and d.day <= 31 and d.hour >= 0 and d.hour < 24 and d.min >= 0 and d.min
     < 60;
end Types
```