MFES

December 20, 2017

Contents

1 Appointment

```
class Appointment is subclass of Task

types
 public Type = <Normal> | <Urgencies>;
 public Priority = <High> | <Medium> | <Low>;
instance variables
  public prescriptions:set of (Prescription);
  public type : Type;
  public priority : Priority;


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

 public Appointment: MedicalAssociated * Type==> Appointment
  Appointment(d, t) == (medicalAssoc := d; type := t; priority := <Medium>; prescriptions := {};
      return self)
 post medicalAssoc = d and type = t and prescriptions = {} and priority = <Medium>;


 public Appointment: MedicalAssociated * Type * Priority ==> Appointment
  Appointment(d, t, p) == (medicalAssoc := d; type := t; priority := p; prescriptions := {};
      return self)

 post medicalAssoc = d and type = t and prescriptions = {} and priority = p;


 pure public getTypeAppointment : () ==> Type
  getTypeAppointment() == (return type);


 pure public getPriority : () ==> 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 : Priority ==> ()
   setPriority(p) == (priority := p);

  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;

 pure public getType : () ==> seq of (char)
  getType() == (
          if type = <Normal>
           then return "Appointment"
          else
           return "Urgencies");
```

```
end Appointment
```

2 Date

```
class Date
types

instance variables
  private year: nat;
  private month: nat;
  private day: nat;
  private hour: nat;
  private minutes : nat;

operations


 public Date: nat * nat * nat * nat * nat ==> Date
 Date(y, m, d, h, min) == (year := y; month := m; day := d; hour := h; minutes := min; return
     self)
 pre y > 0 and m > 0 and m <= 12 and d > 0 and d <= 31 and h >= 0 and h <= 23 and min >= 0 and
     min <= 59;


 pure public getYear : () ==> nat
  getYear() == (return year);


 pure public getMonth : () ==> nat
  getMonth() == (return month);


 pure public getDay : () ==> nat
  getDay() == (return day);


 pure public getHour : () ==> nat
  getHour() == (return hour);


 pure public getMin : () ==> nat
  getMin() == (return minutes);


 pure public compareDateLess : Date ==> bool
  compareDateLess(date) == (return (year < date.getYear() and month < date.getMonth() and day <
      date.getDay() and hour < date.getHour() and minutes < date.getMin()));


 pure public compareDate : Date ==> bool
  compareDate(date) == (return (date.getYear() = year and date.getMonth() = month and date.getDay
      () = day and hour = date.getHour() and minutes = date.getMin()));
end Date
```

3 Hospital

```
class Hospital
types
 public String = seq of(char);
instance variables
  public medicalAssociated: set of (MedicalAssociated);
  public name: String;
  public address: String;
  public tasks: set of(Task);
  public safetyNet: [SafetyNetHospital];

 inv card medicalAssociated >= 0;
 inv card tasks >= 0;
operations


 public Hospital: String * String ==> Hospital
  Hospital(n, a) == (name := n; address := a; medicalAssociated := {}; tasks := {}; return self)
 pre n <> [] and a <> []
 post name = n and address = a and medicalAssociated = {} and tasks = {};


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


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


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


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


 pure public getMedAssociated: String ==> MedicalAssociated
  getMedAssociated(n) == (
                  dcl medical: MedicalAssociated;
                  for all m in set medicalAssociated do
                   if(m.getName() = n)
                    then medical := m;

                  return medical;
                 )
 pre n <> [];


 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;

 pure public getAppointments: () ==> set of (Task)
  getAppointments() == (
              dcl tasks2: set of (Task);
              for all t in set tasks do

               if(t.getType() = "Appointment")
                then tasks2 := tasks2 union {t};
              return tasks2);

 pure public getSurgeries: () ==> set of (Task)
  getSurgeries() == (
              dcl tasks2: set of (Task);
              for all t in set tasks do

               if(t.getType() = "Surgery")
                then tasks2 := tasks2 union {t};
              return tasks2);

 pure public getOther: () ==> set of (Task)
  getOther() == (
              dcl tasks2: set of (Task);
              for all t in set tasks do

               if(t.getType() = "Other")
                then tasks2 := tasks2 union {t};
              return tasks2);

 pure public getDoctors: () ==> set of (MedicalAssociated)
  getDoctors() == (
           dcl doctors: set of (MedicalAssociated);
           for all d in set medicalAssociated do

            if(d.getType() = <Doctor>)
             then doctors := doctors union {d};
           return doctors);


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

```
              return false);

end Hospital
```

4 MedicalAssociated

```
class MedicalAssociated is subclass of Person

types
 public String = seq of (char);
 public Type = <Doctor> | <Surgeon> | <Nurse> | <Technician>;
instance variables
  public medicalNumber: String;
  public specialties:set of (Specialty);
  public patients : set of(Patient);
 public type : Type;


 inv card patients >= 0;
  inv card specialties < 5;

operations
 public MedicalAssociated: String * Type ==> MedicalAssociated

  MedicalAssociated(s, t) == (medicalNumber := s; type := t; specialties := {}; patients := {};
      return self)
 pre s <> []
 post medicalNumber = s and type = t and specialties = {} and patients = {};


 pure public getMedicalNumber: () ==> 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 : () ==> 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
```

5 Medicament

```
class Medicament

types
 public String = seq of (char);
instance variables
  public name:String;

operations

 public Medicament: String ==> Medicament
  Medicament(n) == (name := n; return self)
 pre n <> []
 post name = n;


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

end Medicament
```

6 Patient

```
class Patient is subclass of Person

types
 public String = seq of (char);
instance variables
  healthNumber: String;

operations

 public Patient: String ==> Patient
  Patient(n) == ( healthNumber := n; return self)
 pre n <> []
 post healthNumber = n;


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


end Patient
```

7 Person

```
class Person

types
 public String = seq of (char);
instance variables
  public address: String;
  public firstName: String;
  public lastName: String;
  public cc : String;
  public phoneNumber: String;


operations
 public Person: String * String * String * String * 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 getAddress : () ==> String

  getAddress() == (return address);

 pure public getName : () ==> String

  getName() == (return firstName ^ " " ^ lastName);


 pure public getCC : () ==> String

  getCC() == (return cc);

end Person
```

8 Prescription

```
class Prescription

types

instance variables
  public medicaments:set of (Medicament);
  public code:seq of (char);

operations

 public Prescription: seq of (char) ==> Prescription
  Prescription(c) == (code := c; medicaments := {}; return self)
 pre c <> []
 post code = c and medicaments = {};


 pure public getCode : () ==> seq of (char)
  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: seq of (char) ==> bool
  compare(c) == (return c = code);

end Prescription
```

9 SafetyNetHospital

```
class SafetyNetHospital
types

instance variables
 public hospitals: set of (Hospital);

 inv card hospitals >= 0;
operations


 public SafetyNetHospital : () ==> SafetyNetHospital
  SafetyNetHospital() == (return self);


 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);


 pure public getHospitalsMoreAppointments : () ==> Hospital
  getHospitalsMoreAppointments() == (
                    dcl max: nat, hosp: Hospital;
                    max := 0;
                    for all h in set hospitals do
                     if(card (h.getAppointments()) > max)
```

```
                      then (max := card (h.getAppointments()); 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.getMedicalAssociated();

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

                  return doctors;
                 );

end SafetyNetHospital
```

10 Schedule

```
class Schedule
instance variables
  public startHour: Date;
  public endHour: Date;

  inv startHour.compareDateLess(endHour) = true;

operations


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


 public setEndHour : Date ==> Date
  setEndHour(d) == (endHour := d; return endHour)
 pre startHour.compareDateLess(endHour);


 public setStartHour : Date ==> Date
  setStartHour(d) == (startHour := d; return startHour)
 pre d.compareDateLess(endHour);


 public setSchedule : Date * Date ==> Schedule
  setSchedule(d1, d2) == (startHour := d1; endHour := d2; return self)
 pre d1.compareDateLess(d2);


 pure public getScheduleStart : () ==> Date
  getScheduleStart() == (return startHour);
 pure public getScheduleEnd : () ==> Date
  getScheduleEnd() == (return endHour);

end Schedule
```

11 Specialty

```
class Specialty

types
 public String = seq of (char);
instance variables
  public name: String;

operations

 public Specialty : String ==> Specialty
  Specialty(n) == (name := n; return self)
 pre n <> []
 post name = n;


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

end Specialty
```

12 Surgery

```
class Surgery is subclass of Task

types

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

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

 public Surgery: MedicalAssociated ==> Surgery
  Surgery(s) == (medicalAssoc := s ; other := {}; secondaryDoctors := {}; return self)

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

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

 pre s not in set secondaryDoctors
 post s in set secondaryDoctors;

 pure public removeSecondaryDoctor : MedicalAssociated ==> set of (MedicalAssociated)
  removeSecondaryDoctor(s) == (return secondaryDoctors \ {s})
 pre 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 not in set other
 post s in set other;

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

 pre s in set other
 post s not in set other;

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


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


 public getSecondaryDoctors : () ==> set of (MedicalAssociated)
  getSecondaryDoctors() == (return secondaryDoctors);


 public getOthers : () ==> set of (MedicalAssociated)
  getOthers() == (return other);


 pure public getType : () ==> seq of (char)
  getType() == (return "Surgery");


end Surgery
```

13 Task

```
class Task
instance variables
  public schedule:[Schedule];
  public patient:[Patient];
  public hospital:[Hospital];
  public medicalAssoc:[MedicalAssociated];

  inv schedule <> nil;
  inv patient <> nil;
  inv hospital <> nil;
  inv medicalAssoc <> nil;

  inv medicalAssoc.getCC() <> patient.getCC();

operations
 public Task: Schedule * Patient * Hospital ==> Task

  Task(s, p, h) == (schedule := s; patient := p; hospital := h; return self)
 post schedule = s and patient = p and hospital = h;
 pure public getSchedule: () ==> Schedule
  getSchedule() == (return schedule);


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


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



 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);

 pure public getType : () ==> seq of (char)
  getType() == (return "");

end Task
```

14 Training

```
class Training

types
 public Purpose = <Training> | <AddSkills>;

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

  inv card medicalAssociated >= 0 and card medicalAssociated < 10;
  inv purpose <> nil;
  inv schedule <> nil;
operations

 public Training: 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 : () ==> 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 : Purpose ==> ()
  setPurpose(p) == (purpose := p);

end Training
```

15 Treatment

```
class Treatment is subclass of Task
types
 public String = seq of (char);
instance variables
  public technician: [MedicalAssociated];
  public name: String;

  inv technician <> nil;
operations



 public Treatment: String ==> Treatment
  Treatment(n) == (name := n; return self)
 pre n <> []
 post name = n;



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



 public setTechnician: MedicalAssociated ==> ()
  setTechnician(t) == (technician := t; return);




 pure public getTechnician : () ==> MedicalAssociated
```

```
  getTechnician() == (return technician);



 pure public getTechnicianName : () ==> String
  getTechnicianName() == (return technician.getName());



 pure public getType : () ==> seq of (char)
  getType() == (return "Hospital Treatment");

end Treatment
```