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 type <> nil and priority <> nil;
  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)
  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;

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

end Appointment
```

2 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 name <> [] and address <> [];
 inv safetyNet <> nil;
 inv card medicalAssociated >= 0;

 inv card tasks >= 0;
operations

 public Hospital: String * String * SafetyNetHospital ==> Hospital
  Hospital(n, a, s) == (name := n; address := a; safetyNet := s; medicalAssociated := {}; tasks
      := {}; return self)

 pre n <> [] and a <> [] and safetyNet <> nil
 post name = n and address = a and safetyNet = s and medicalAssociated = {} and tasks = {};


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


 pure public getAddress: () ==> 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;

 pure public getTasksByType: () ==> seq of(set of (Task))
  getTasksByType() == (
              dcl tasksTotal: seq of(set of (Task)), tasks2: set of (Task), tasks3: set of(Task),
                   tasks4: set of (Task), tasks5: set of(Task);

              for all t in set tasks do(
               if(t.getType() = "Appointment")
                then tasks2 := tasks2 union {t}
               else if(t.getType() = "Urgencies")
                then tasks3 := tasks3 union {t}

               else if(t.getType() = "Surgery")

                then tasks4 := tasks4 union {t}
               else
                tasks5 := tasks5 union {t}
               );

              tasksTotal := tasksTotal ^ [tasks2] ^ [tasks3] ^ [tasks4] ^ [tasks5];
              return tasksTotal);


 pure public getMedicalAssociatedByType: () ==> seq of(set of (MedicalAssociated))

  getMedicalAssociatedByType() == (
           dcl med: seq of(set of(MedicalAssociated)), doctors: set of (MedicalAssociated),
               surgeons: set of (MedicalAssociated), other: set of (MedicalAssociated);
           for all d in set medicalAssociated do(
            if(d.getType() = <Doctor>)
             then doctors := doctors union {d}
            else if(d.getType() = <Surgeon>)

             then surgeons := surgeons union {d}
            else
             other := other union {d}
            );

            med := med ^ [doctors] ^ [surgeons] ^ [other];
           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);

end Hospital
```

3 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;
 inv medicalNumber <> [];
 inv type <> nil;
operations

 public MedicalAssociated: String * 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: () ==> 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
```

4 Medicament

```
class Medicament

types
 public String = seq of (char);
instance variables
  public name:String;
  inv name <> [];
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
```

5 Patient

```
class Patient is subclass of Person

types
 public String = seq of (char);
instance variables
  public healthNumber: String;
  inv healthNumber <> [];
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
```

6 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;


  inv address <> [] and firstName <> [] and lastName <> [] and cc <> [] and len cc = 9 and
      phoneNumber <> [] and len phoneNumber = 9;
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 getCC : () ==> String
  getCC() == (return cc);


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



end Person
```

7 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
```

8 SafetyNetHospital

```
class SafetyNetHospital
types

instance variables
 public 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);


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

                   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 hd l.getMedicalAssociatedByType() and
                          m not in set doctors)
                      then doctors := doctors union {m};
                   );
                  );

                  return doctors;
                 );

end SafetyNetHospital
```

9 Schedule

```
class Schedule

types
 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;

instance variables
  public startHour: Date;
  public endHour: Date;


  inv compareDateLess(startHour, endHour) = true;
operations


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


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


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

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

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

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


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


 pure public compareDateLess : Date * 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 : Date * 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

types
 public String = seq of (char);
instance variables
  public name: String;
  inv name <> [];
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
```

11 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 <> 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);

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

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

              return med);

 pure public getType : () ==> seq of (char)

  getType() == (return "Surgery");

end Surgery
```

12 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.getCC() <> patient.getCC();


operations
 public Task: Schedule * Patient * Hospital ==> Task
  Task(s, p, h) == (schedule := s; patient := p; hospital := h; 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 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
```

13 Treatment

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

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



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



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




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




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



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



end Treatment
```