MFES

December 19, 2017

Contents

1 Appointment

```
class Appointment is subclass of Task

instance variables
  public doctor: [Doctor];
  public prescriptions:set of (Prescription);



  inv doctor <> nil;
  inv card prescriptions >= 0;
operations
 public Appointment: Doctor ==> Appointment
  Appointment(d) == (doctor := d; prescriptions := {}; return self)
 post doctor = d and prescriptions = {};


 pure public getDoctorAppointment : () ==> Doctor
  getDoctorAppointment() == (return doctor);


  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 <> [];


  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() == (return "Appointment");

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 Doctor

```
class Doctor is subclass of MedicalAssociated
types

instance variables

operations

 public Doctor: () ==> Doctor
```

```
  Doctor() == (return self);


 pure public getType : () ==> String
  getType() == (return "Doctor");


end Doctor
```

4 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);

 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;



pure public addTask: Task ==> set of (Task)
 addTask(d) == (return ({d} union tasks))
pre d not in set tasks
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 (Doctor)
 getDoctors() == (
```

```
          dcl doctors: set of (Doctor);
          for all d in set medicalAssociated do
           if(d.getType() = "Doctor")
            then doctors := doctors union {d};
          return doctors);


  pure public geNormalDoctors: () ==> set of (NormalDoctor)
  geNormalDoctors() == (
          dcl doctors: set of (Doctor);
          for all d in set medicalAssociated do
           if(d.getType() = "Normal Doctor")
            then doctors := doctors union {d};
          return doctors);
end Hospital
```

5 HospitalTreatment

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

  inv technician <> nil;
operations



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

 post name = n;

 pure public getName: () ==> String

  getName() == (return name);

 public setTechnician: OtherMedicalAssociated ==> ()

  setTechnician(t) == (technician := t; return);

 pure public getTechnician : () ==> OtherMedicalAssociated

  getTechnician() == (return technician);

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


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

end HospitalTreatment
```

6 MedicalAssociated

```
class MedicalAssociated is subclass of Person

types
 public String = seq of (char);
instance variables
  public medicalNumber: String;
  public specialties:set of (Specialty);



  inv card specialties < 5;

operations
 public MedicalAssociated: String ==> MedicalAssociated


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

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

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

 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;


 pure public getType : () ==> String
  getType() == (return "");

end MedicalAssociated
```

7 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
```

8 NormalDoctor

```
class NormalDoctor is subclass of Doctor

instance variables


 public patients : set of(Patient);

 inv card patients > 0;
operations
 public NormalDoctor: () ==> NormalDoctor
  NormalDoctor() == (patients := {}; return self)

 post patients = {};

 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;


 pure public getType : () ==> String
  getType() == (return "Normal Doctor");

end NormalDoctor
```

9 OtherMedicalAssociated

```
class OtherMedicalAssociated is subclass of MedicalAssociated

types

instance variables
operations
```

```
 public OtherMedicalAssociated: () ==> OtherMedicalAssociated
  OtherMedicalAssociated() == (return self);




 pure public getType : () ==> String
  getType() == (return "Other");

end OtherMedicalAssociated
```

10 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
```

11 Person

```
class Person

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

operations


 public Person: String * String * String ==> Person
```

```
  Person(a, fn, ln) == ( address := a; firstName := fn; lastName := ln; return self)
 pre a <> [] and fn <> [] and ln <> []
 post address = a and firstName = fn and lastName = ln;



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


 pure public getName : () ==> String
  getName() == (return firstName ^ " " ^ lastName);



 pure public getFirstName : () ==> String
  getFirstName() == (return firstName);


 pure public getLastName : () ==> String
  getLastName() == (return lastName);

end Person
```

12 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
```

13 SafetyNetHospital

```
class SafetyNetHospital
types

instance variables
 public hospitals: set of (Hospital);

 inv card hospitals > 1;
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(Doctor)
  getDoctorsMoreHospitals() == (
                  dcl doctors: set of(Doctor);
                  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
```

14 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);


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

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

end Schedule
```

15 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
```

16 Surgeon

```
class Surgeon is subclass of Doctor

operations

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


 pure public getType : () ==> String
  getType() == (return "Surgeon");
end Surgeon
```

17 Surgery

```
class Surgery is subclass of Task

types

instance variables
  public mainDoctor: [Surgeon];
  public secondaryDoctors:set of (Surgeon);
  public other:set of (OtherMedicalAssociated);

  inv mainDoctor <> nil;
  inv card secondaryDoctors >= 0;

  inv card other >= 0;
operations

 public Surgery: Surgeon ==> Surgery

  Surgery(s) == (mainDoctor := s; other := {}; secondaryDoctors := {}; return self)
```

```
 post mainDoctor = s and other = {} and secondaryDoctors = {};


 pure public addSecondaryDoctor : Surgeon ==> set of (Surgeon)
  addSecondaryDoctor(s) == (return secondaryDoctors union {s})
 pre s not in set secondaryDoctors

 post s in set secondaryDoctors;

 pure public removeSecondaryDoctor : Surgeon ==> set of (Surgeon)

  removeSecondaryDoctor(s) == (return secondaryDoctors \ {s})
 pre s in set secondaryDoctors
 post s not in set secondaryDoctors;



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

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




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

 post s not in set other;

 public setMainDoctor : Surgeon ==> ()
  setMainDoctor(s) == (mainDoctor := s);

 public getMainDoctor : () ==> Surgeon
  getMainDoctor() == (return mainDoctor);

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


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


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

end Surgery
```

18 Task

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

  inv schedule <> nil;
```

```
  inv patient <> nil;
  inv hospital <> nil;


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);

 public setSchedule : Schedule ==> ()

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

 public setPatient : Patient ==> ()
  setPatient(s) == (patient := s);

 public setHospital : Hospital ==> ()
  setHospital(s) == (hospital := s);


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

end Task
```

19 Training

```
class Training

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

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

  inv card medicalAssociated > 1 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
```

20 Urgencies

```
class Urgencies is subclass of Task

types
 public Priority = <High> | <Medium> | <Low>
instance variables
  public priority : [Priority];

  inv priority <> nil;

operations
 public Urgencies : Priority ==> Urgencies
  Urgencies(p) == (priority := p; return self)
 post priority = p;


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


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


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

end Urgencies
```