```
model MTS

-- classes

class Hospital
attributes
  name : String
  phone : String
  address : String
end

class Patient
attributes
  firstName : String
  lastName : String
  id : String
  gender : Gender
  age : Integer
  obtain_patient_info : Boolean
  examined_airway : Boolean
  examined_breathing : Boolean
  checked_patient_for_shock : Boolean
  pain_measured : Boolean
  cardiac_pain_examined : Boolean
  breathing_rate_measured : Boolean
  pulse_checked : Boolean
  pleuritic_pain_examined : Boolean
  checked_for_persistent_vomiting : Boolean
  examine_patients_cardiac_history : Boolean
  pain_remeasured : Boolean
  vomiting_checked : Boolean
  mild_pain_examined : Boolean
  problems_checked : Boolean
  resuscitation_in_action : Boolean
  patient_stabilized : Boolean
  patient_supervised : Boolean
  patient_reassessed : Boolean
end

class TreatedPatient < Patient
attributes
  airwayCompromise : Boolean
  inadequateBreathing : Boolean
  shock : Boolean
  severePain : Boolean
  moderatePain : Boolean
  minorPain : Boolean
  cardiacPain : Boolean
  acutelyShortBreath : Boolean
  normalBreathing : Boolean
  abnormalPulse : Boolean
  normalPulse : Boolean
  pleuriticPain : Boolean
  persistentVomiting : Boolean
  recentVomiting : Boolean
  significantCardiacHistory : Boolean
  recentMildPain : Boolean
  recentProblem : Boolean
end

class Severity
attributes
  severity: SeverityLevel
operations
  examineAirway(p : TreatedPatient)
  checkForAdequateBreathing(p : TreatedPatient)
  checkForShock(p : TreatedPatient)
  measurePain(p : TreatedPatient)
  examineCardiacPain(p : TreatedPatient)
  measureBreathingRate(p : TreatedPatient)
  checkPulse(p : TreatedPatient)
  examinePleuriticPain(p : TreatedPatient)
  checkForPersistentVomiting(p : TreatedPatient)
  examineForCardiacHistory(p : TreatedPatient)
  reMeasureForPain(p : TreatedPatient)
  checkVomiting(p : TreatedPatient)
  examineForMildPain(p : TreatedPatient)
  checkForProblems(p : TreatedPatient)
end
```

```

class Immediate
operations
  resuscitationRequired(p : TreatedPatient)
  continuousObservation(p : TreatedPatient)
end

class VeryUrgent
operations
  stabilize(p : TreatedPatient)
  monitor(p : TreatedPatient)
end

class Urgent
operations
  supervision(p : TreatedPatient)
end

class Standard
operations
  reassess(p : TreatedPatient)
end

class NonUrgent
operations
  reassess(p : TreatedPatient)
  canWait(p : TreatedPatient)
end

-- associations

composition ConsistOf between
  Hospital[1] role hospital
  Patient[1..*] role patient
end

association WithTreatment between
  Patient[1] role patient
  TreatedPatient[1] role treatedPatient
end

association BasedOnSeverityLevel between
  TreatedPatient[1] role patientTreatment
  Severity[1] role severityLevel
end

association ImmediateSeverity between
  Severity[1] role severityLevel
  Immediate[1] role immediateCare
end

association VeryUrgentSeverity between
  Severity[1] role severityLevel
  VeryUrgent[1] role veryUrgentCare
end

association UrgentSeverity between
  Severity[1] role severityLevel
  Urgent[1] role urgentCare
end

association StandardSeverity between
  Severity[1] role severityLevel
  Standard[1] role standardCare
end

association NonUrgentSeverity between
  Severity[1] role severityLevel
  NonUrgent[1] role nonUrgentCare
end

-- enumeration

enum SeverityLevel {Immediate, VeryUrgent, Urgent, Standard, NonUrgent}
enum Gender {Male, Female}

-- constraints

constraints
```

```

context Hospital
inv AtLeastOnePatient: self.patient->size() >= 1
inv NonEmptyName: self.name <> ' ' or self.name.size > 1

context p : Patient
inv NumberOfPatients: Patient.allInstances()->size() >= 1
inv PatientsInHospital: p.hospital->size() = 1
inv NonEmptyFirstName: p.firstName <> ' ' or p.firstName.size > 1
inv NonEmptyLastName: p.lastName <> ' ' or p.lastName.size > 1
inv NonEmptyID: p.id <> ' ' or p.id.size > 1
inv UniquePatientID: Patient.allInstances()->forAll(p, p2 | p.id <> p2.id)

context Severity :: examineAirway(p : TreatedPatient)
pre: p.obtain_patient_info = true
post: p.examined_airway = true

context Severity:: checkForAdequateBreathing(p : TreatedPatient)
pre: p.examined_airway = true
post: p.examined_breathing = true

context Severity:: checkForShock(p : TreatedPatient)
pre: p.examined_airway = true and p.examined_breathing = true
post: p.checked_patient_for_shock = true

context Severity:: measurePain(p : TreatedPatient)
pre: p.checked_patient_for_shock = true
post: p.pain_measured = true

context Severity:: examineCardiacPain(p : TreatedPatient)
pre: p.pain_measured = true
post: p.cardiac_pain_examined = true

context Severity:: measureBreathingRate(p : TreatedPatient)
pre: p.pain_measured = true and p.cardiac_pain_examined = true
post: p.breathing_rate_measured = true

context Severity:: checkPulse(p : TreatedPatient)
pre: p.pain_measured = true and p.cardiac_pain_examined = true and p.breathing_rate_measured = true
post: p.pulse_checked = true

context Severity:: examinePleuriticPain(p : TreatedPatient)
pre: p.pulse_checked = true
post: p.pleuritic_pain_examined = true

context Severity:: checkForPersistentVomiting(p : TreatedPatient)
pre: p.pleuritic_pain_examined = true
post: p.checked_for_persistent_vomiting = true

context Severity:: examineForCardiacHistory(p : TreatedPatient)
pre: p.pleuritic_pain_examined=true and p.checked_for_persistent_vomiting = true
post: p.examine_patients_cardiac_history = true

context Severity:: reMeasureForPain(p : TreatedPatient)
pre: p.pleuritic_pain_examined = true and p.checked_for_persistent_vomiting = true and p.examine_patients_cardiac_history = true
post: p.pain_remeasured = true

context Severity:: checkVomiting(p : TreatedPatient)
pre: p.pain_remeasured = true
post: p.vomiting_checked = true

context Severity:: examineForMildPain(p : TreatedPatient)
pre: p.vomiting_checked = true
post: p.mild_pain_examined = true

context Severity:: checkForProblems(p : TreatedPatient)
pre: p.vomiting_checked = true and p.mild_pain_examined = true
post: p.problems_checked = true

context Immediate:: resuscitationRequired(p : TreatedPatient)
pre: p.examined_airway = true
pre: p.examined_breathing = true
pre: p.checked_patient_for_shock = true
post: p.resuscitation_in_action = true

context VeryUrgent:: stabilize(p : TreatedPatient)
pre: p.pain_measured = true
pre: p.cardiac_pain_examined = true
pre: p.breathing_rate_measured = true
```

```
pre: p.pulse_checked = true
post: p.patient_stabilized = true

context Urgent:: supervision(p : TreatedPatient)
pre: p.pleuritic_pain_examined = true
pre: p.checked_for_persistent_vomiting = true
pre: p.examine_patients_cardiac_history = true
pre: p.pain_remeasured = true
post: p.patient_supervised = true

context Standard:: reassess(p : TreatedPatient)
pre: p.vomiting_checked = true
pre: p.mild_pain_examined = true
pre: p.problems_checked = true
post: p.patient_reassessed = true

context NonUrgent:: reassess(p : TreatedPatient)
pre: p.vomiting_checked = true
pre: p.mild_pain_examined = true
pre: p.problems_checked = true
post: p.patient_reassessed = true

```

```
model ESI

-- classes

class Hospital
attributes
  name : String
  phone : String
  address : String
end

class Patient
attributes
  firstName : String
  lastName : String
  id : String
  gender : Gender
  age : Integer
  obtain_patient_info : Boolean
  checked_for_immediate_intervention : Boolean
  situation_assessed : Boolean
  examined_if_confused : Boolean
  examined_if_disoriented : Boolean
  examined_if_lethargic : Boolean
  checked_if_severe_pain : Boolean
  checked_if_distressed : Boolean
  number_resources_assessed : Boolean
  resources : Integer
  vitals_examined : Boolean
end

class TreatedPatient < Patient
attributes
  requiresLifeSavingIntervention : Boolean
  highRiskSituation : Boolean
  confused : Boolean
  lethargic : Boolean
  disoriented : Boolean
  severePain : Boolean
  distress : Boolean
  resourcesRequired : Integer
  heartRate : Integer
  respiratoryRate : Integer
  saO2 : Integer
end

class Severity
attributes
  severity : SeverityLevel
operations
  checkIfInterventionNeeded(p : TreatedPatient)
  assessSituation(p : TreatedPatient)
  examine(p : TreatedPatient)
  checkForPain(p : Patient)
  checkResourcesNeeded(p : TreatedPatient)
  measureVitalZones(p : TreatedPatient, resources : Integer)
end

class Immediate
operations
  resuscitationRequired(p : TreatedPatient)
  continuousObservation(p : TreatedPatient)
end

class VeryUrgent
operations
  stabilize(p : TreatedPatient)
  monitor(p : TreatedPatient)
end

class Urgent
operations
  supervision(p : TreatedPatient)
end

class Minor
operations
  reassess(p : TreatedPatient, resources : Integer)
end
```

```

class Delayed
operations
  canWait(p : TreatedPatient)
  reassess(p : TreatedPatient, resources : Integer)
end

-- associations

composition ConsistsOf between
  Hospital[1] role hospital
  Patient[1..*] role patient
end

association WithTreatment between
  Patient[1] role patient
  TreatedPatient[1] role treatedPatient
end

association BasedOnSeverityLevel between
  TreatedPatient[1] role patientTreatment
  Severity[1] role severityLevel
end

association ImmediateSeverity between
  Severity[1] role severityLevel
  Immediate[1] role immediateCare
end

association VeryUrgentSeverity between
  Severity[1] role severityLevel
  VeryUrgent[1] role veryUrgentCare
end

association UrgentSeverity between
  Severity[1] role severityLevel
  Urgent[1] role urgentCare
end

association MinorSeverity between
  Severity[1] role severityLevel
  Minor[1] role minorCare
end

association DelayedSeverity between
  Severity[1] role severityLevel
  Delayed[1] role delayedCare
end

-- enumeration

enum SeverityLevel {Immediate, VeryUrgent, Urgent, Minor, Delayed}
enum Gender {Male, Female}

-- constraints

constraints

context Hospital
inv AtLeastOnePatient: self.patient->size() >= 1
inv NonEmptyName: self.name <> ' ' or self.name.size > 1

context p : Patient
inv NumberOfPatients: Patient.allInstances()->size() >= 1
inv PatientsInHospital: p.hospital->size() = 1
inv NonEmptyFirstName: p.firstName <> ' ' or p.firstName.size > 1
inv NonEmptyLastName: p.lastName <> ' ' or p.lastName.size > 1
inv NonEmptyID: p.id <> ' ' or p.id.size > 1
inv UniquePatientID: Patient.allInstances()->forAll(p, p2 | p.id <> p2.id)

context Severity :: checkIfInterventionNeeded(p : TreatedPatient)
pre: p.obtain_patient_info = true
post: p.checked_for_immediate_intervention = true

context Severity :: assessSituation(p : TreatedPatient)
pre: p.checked_for_immediate_intervention = true
post: p.situation_assessed = true

context Severity :: examine(p : TreatedPatient)
```

```
pre: p.situation_assessed = true
post: p.examined_if_confused = true or p.examined_if_disoriented = true or p.examined_if_lethargic = true

context Severity :: checkForPain(p : TreatedPatient)
pre: p.situation_assessed = true
pre: p.examined_if_confused = true or p.examined_if_disoriented = true or p.examined_if_lethargic = true
post: p.checked_if_severe_pain = true or p.checked_if_distressed = true

context Severity:: checkResourcesNeeded(p : TreatedPatient)
pre: p.checked_if_severe_pain = true or p.checked_if_distressed = true
post: p.number_resources_assessed = true

context Severity :: measureVitalZones(p : TreatedPatient, resources: Integer)
pre: p.number_resources_assessed = true and resources > 1
post: p.vitals_examined = true

context Immediate :: resuscitationRequired (p : TreatedPatient)
pre: p.checked_for_immediate_intervention = true

context VeryUrgent :: stabilize(p : TreatedPatient)
pre: p.situation_assessed = true
pre: p.examined_if_confused = true or p.examined_if_disoriented = true or p.examined_if_lethargic = true
pre: p.checked_if_severe_pain = true or p.checked_if_distressed = true
pre: p.vitals_examined = true

context Urgent :: supervision(p : TreatedPatient)
pre: p.vitals_examined = true

context Minor :: reassess (p : TreatedPatient, resources: Integer)
pre: p.number_resources_assessed = true
pre: resources = 1

context Delayed:: reassess (p : TreatedPatient, resources: Integer)
pre: p.number_resources_assessed = true
pre: resources = 0

```