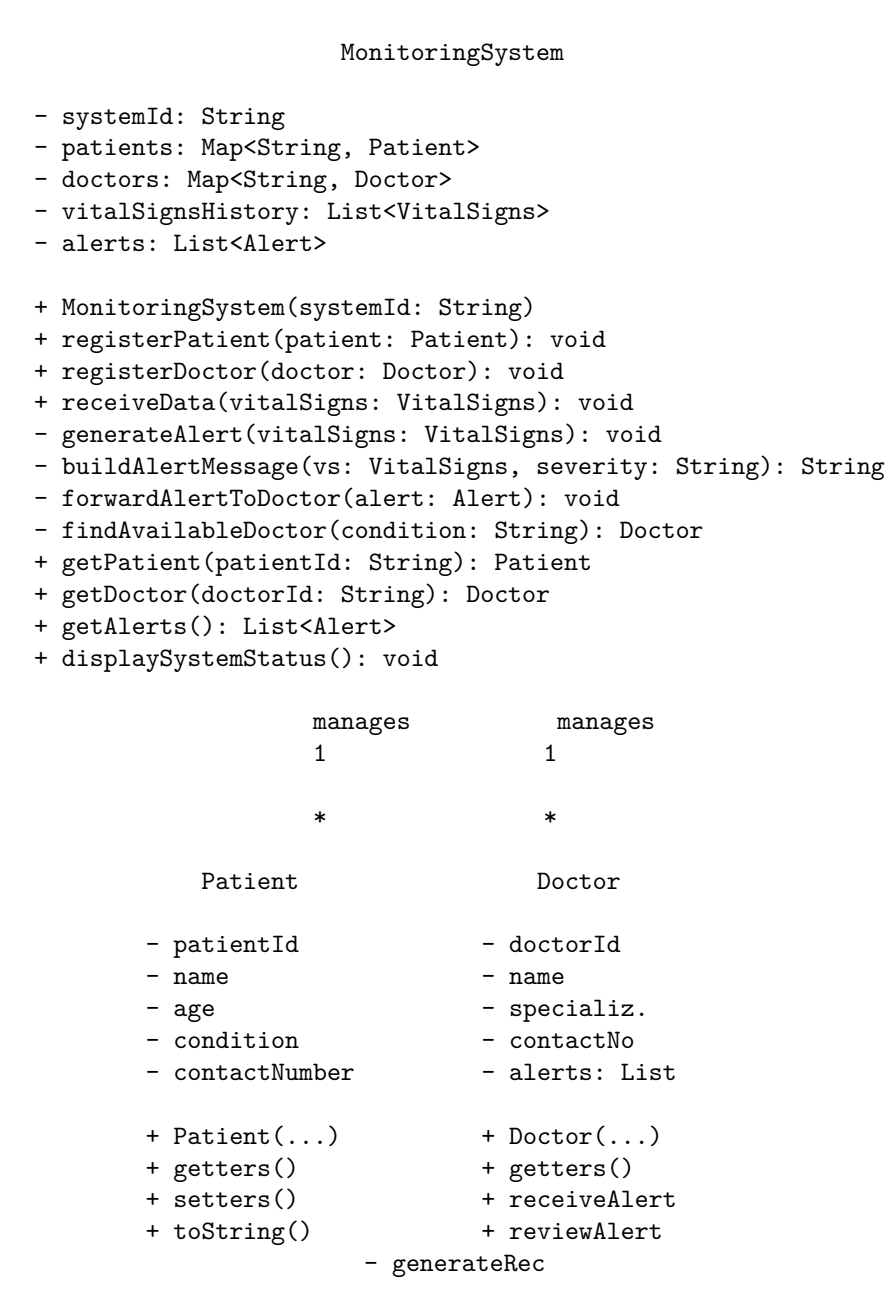
UML Class Diagram - Remote Health Monitoring System

Complete Class Diagram



Relationships Explained

1. MonitoringSystem Patient (1 to *)

- **Type**: Aggregation

- **Description**: The system manages multiple patients
- **Implementation**: `Map<String, Patient> patients`
- **Direction**: Bidirectional (system knows patients, patients registered to system)

2. MonitoringSystem Doctor (1 to *)

- **Type**: Aggregation
- **Description**: The system manages multiple doctors
- **Implementation**: `Map<String, Doctor> doctors`
- **Direction**: Bidirectional

3. Patient WearableDevice (1 to 1)

- **Type**: Association
- **Description**: Each patient uses one wearable device
- **Implementation**: Device stores `patientId` reference
- **Direction**: Unidirectional (device knows patient)

4. WearableDevice → VitalSigns (1 to *)

- **Type**: Dependency (generates)
- **Description**: Device generates multiple vital sign readings over time
- **Implementation**: `recordVitalSigns()` returns new VitalSigns object
- **Direction**: Unidirectional

5. VitalSigns → Alert (1 to 0..1)

- **Type**: Dependency
- **Description**: Abnormal vital signs trigger alert creation
- **Implementation**: System checks `isAbnormal()` and creates Alert
- **Direction**: Unidirectional

6. Alert ← VitalSigns (1 to 1)

- **Type**: Composition
- **Description**: Alert contains the vital signs data that triggered it
- **Implementation**: `private VitalSigns vitalSigns`
- **Direction**: Unidirectional (alert owns vital signs reference)

7. Doctor Alert (* to *)

- **Type**: Association
- **Description**: Doctors review multiple alerts; alerts can be reviewed by doctors
- **Implementation**: Doctor has `List<Alert> assignedAlerts`
- **Direction**: Bidirectional

Class Responsibilities

MonitoringSystem (Controller)

Purpose: Central coordinator managing all system operations **Key Responsibilities**: - Register and manage patients and doctors - Receive data from wearable devices - Analyze vital signs for abnormalities - Generate and dispatch alerts - Maintain system history and status

Patient (Entity)

Purpose: Represents a patient being monitored **Key Responsibilities**: - Store patient demographic information - Track chronic condition - Provide patient identification

WearableDevice (Sensor Interface)

Purpose: Simulates IoT health monitoring device **Key Responsibilities**: - Record vital signs from patient - Transmit data to monitoring system - Manage device activation status

VitalSigns (Value Object)

Purpose: Encapsulates health metrics at a point in time **Key Responsibilities**: - Store vital sign measurements - Determine if readings are abnormal - Provide timestamp for readings

Alert (Entity)

Purpose: Represents health warnings requiring attention **Key Responsibilities**: - Contain abnormal vital signs information - Categorize severity level - Track resolution status - Provide descriptive alert message

Doctor (Actor)

Purpose: Medical professional reviewing alerts **Key Responsibilities**: - Receive and manage alerts - Review patient health data - Generate medical recommendations - Schedule patient follow-ups

Design Patterns Applied

1. MVC Pattern

- **Model**: Patient, Doctor, VitalSigns, Alert
- **View**: Main class console output
- **Controller**: MonitoringSystem

2. Observer Pattern (Implicit)

- WearableDevice generates data
- MonitoringSystem observes and reacts to data
- Doctors are notified of alerts

3. Strategy Pattern

- Doctor's `generateRecommendation()` applies different strategies based on:
  - Patient's chronic condition
  - Alert severity
  - Specific vital sign abnormalities

4. Factory Pattern (Partial)

- Alert severity determination (`determineSeverity()`)
- Alert message building (`buildAlertMessage()`)

Data Flow Sequence

```
1. WearableDevice.recordVitalSigns()
            ↓
2. WearableDevice.transmitData(system, vitalSigns)
            ↓
3. MonitoringSystem.receiveData(vitalSigns)
            ↓
4. VitalSigns.isAbnormal() → true?
            ↓ YES
5. MonitoringSystem.generateAlert(vitalSigns)
            ↓
6. Alert.determineSeverity(vitalSigns)
            ↓
7. MonitoringSystem.forwardAlertToDoctor(alert)
            ↓
8. Doctor.receiveAlert(alert)
            ↓
9. Doctor.reviewAlert(alert, patient)
            ↓
10. Doctor.generateRecommendation(alert, patient)
            ↓
11. Alert.resolve()
```

Multiplicity Notation

- `1` : Exactly one
- `*` : Zero or more

- `0..1` : Zero or one
- `1..*` : One or more

Class Stereotypes

- `<<entity>>` : Patient, Doctor, Alert
- `<<value object>>` : VitalSigns
- `<<controller>>` : MonitoringSystem
- `<<boundary>>` : WearableDevice
- `<<utility>>` : Main (demo/test class)