Remote Patient Monitoring System (RPMS)

A Comprehensive Healthcare Management Application Using OOP and JavaFX

Table of Contents

1. Background

In today's rapidly evolving healthcare landscape, traditional medical service delivery faces numerous challenges including overcrowded clinics, delayed diagnoses, and limited access to care, particularly in rural or remote areas. These issues became increasingly evident during the COVID-19 pandemic, which catalyzed a paradigm shift in healthcare delivery models away from physical consultations toward remote monitoring solutions.

Remote Patient Monitoring (RPM) technologies have emerged as a critical solution to these challenges. By enabling healthcare providers to monitor vital signs such as heart rate, blood pressure, temperature, and oxygen saturation in real-time without requiring patients to be physically present in medical facilities, RPM systems facilitate early detection of health anomalies, reduce hospital readmissions, optimize healthcare resource allocation, and ultimately improve patient outcomes.

The digital transformation of healthcare has accelerated, with global RPM market valued at USD 1.2 billion in 2022 and projected to grow at a CAGR of 19.4% from 2023 to 2030. This growth reflects the increasing recognition of remote monitoring as an essential component of modern healthcare infrastructure, especially for managing chronic conditions and elderly care.

To bridge the gap between theoretical understanding and practical application, our Remote Patient Monitoring System (RPMS) has been developed as an educational model that simulates real-world healthcare monitoring applications. Built with Object-Oriented Programming (OOP) principles in Java and JavaFX, this system provides a comprehensive platform where:

- Patients can upload health vitals through CSV files, view personalized reports, track health trends, and consult with healthcare providers remotely
- Doctors can monitor patient data across time, provide feedback, prescribe medications, and schedule virtual or in-person appointments
- Administrators can manage user accounts, system analytics, and generate comprehensive reports on system usage and health trends

This project serves dual purposes: demonstrating the application of fundamental OOP principles—encapsulation, inheritance, polymorphism, composition, and interfaces—in a complex system, while also addressing the pressing social need for accessible, efficient healthcare monitoring solutions that can function effectively even during public health crises.

2. Introduction

The Remote Patient Monitoring System (RPMS) represents a sophisticated Java desktop application developed to simulate a comprehensive healthcare platform through rigorous application of Object-Oriented Programming (OOP) methodologies. The system facilitates seamless interaction between patients, doctors, and administrators through a secure, role-specific interface developed with JavaFX.

This project was conceptualized and implemented as part of the Object-Oriented Programming curriculum, with the primary goal of translating theoretical OOP concepts into a modular, practical solution addressing real-world healthcare challenges. The system encompasses a wide range of functionalities including patient data management, doctor feedback mechanisms, emergency alert systems, telemedicine capabilities, health analytics visualization, and comprehensive reporting tools.

2.1 Project Objectives

The development of the RPMS was guided by the following core objectives:

- **Demonstrate OOP Application**: Implement a comprehensive system that showcases practical applications of abstraction, encapsulation, inheritance, polymorphism, and composition in solving complex domain problems.
- **Design for Modularity**: Create a scalable, extensible system architecture with clearly defined modules that can be independently developed, tested, and maintained.
- **Ensure Role-Based Access**: Implement distinct user interfaces and functionalities for patients, doctors, and administrators, with appropriate access controls.
- **Develop a Complete Application**: Build a fully functional Java application incorporating standard development tools, design patterns, and industry practices.
- **Simulate Real-World Interactions**: Create authentic healthcare workflows that mirror actual patient-doctor interactions in remote monitoring scenarios.
- **Implement Data Visualization**: Provide intuitive visual representations of health data to facilitate rapid comprehension of patient status and trends.
- **Ensure System Security**: Implement robust authentication, authorization, and data protection measures appropriate for healthcare applications.

2.2 Key Features

The RPMS includes the following comprehensive feature set, each designed to address specific healthcare monitoring requirements:

- **Health Data Management**:
  - Patient-driven vital signs upload through CSV files (heart rate, oxygen saturation, temperature, blood pressure)
  - Data validation to ensure accuracy within medically acceptable ranges
  - Historical data storage and retrieval for trend analysis
  - Structured data organization for efficient querying
- **Clinical Feedback System**:
  - Doctor review interface for patient vitals assessment
  - Structured feedback forms for medical recommendations
  - Medication prescription capabilities with dosage, frequency, and duration parameters
  - Diagnostic test recommendation functionality
  - Treatment plan documentation tools
- **Emergency Alert Framework**:
  - Automated critical vital signs detection algorithms
  - Configurable threshold settings for different vital parameters
  - Immediate alert dispatch to medical personnel
  - Notification cascade to patient and emergency contacts
  - Manual emergency trigger option for patients experiencing distress
  - Alert severity classification system
- **Telemedicine Suite**:
  - Text-based real-time chat using Java sockets
  - End-to-end message encryption
  - Video consultation capability via generated meeting links
  - Consultation scheduling and reminder system
  - Chat history preservation for medical documentation
  - File sharing capabilities for relevant medical documents
- **Comprehensive Reporting**:
  - Customizable health report generation
  - Historical vitals trend documentation
  - Medication tracking and compliance reporting
  - Clinical feedback compilation
  - Exportable formats for external use (PDF)

    - Secure report storage and retrieval
- **Interactive Data Visualization**:
    - Temporal trend analysis through interactive line graphs
    - Multi-parameter correlation charts
    - Color-coded anomaly highlighting
    - Zoom and filter capabilities for detailed examination
    - Customizable date range selection
    - Statistical summary generation
- **Authentication and User Management**:
    - Role-based access control (RBAC) implementation
    - Secure credential management
    - Password encryption and security protocols
    - User profile management
    - Activity logging and audit trail creation
    - Session management and timeout features
- **Appointment System**:
    - Patient-initiated appointment requests
    - Doctor approval workflow
    - Scheduling conflict detection
    - Calendar integration and visualization
    - Appointment change and cancellation management
    - Automated reminder generation
- **Notification Center**:
    - Appointment reminders via simulated email
    - Medication administration alerts
    - Critical vitals notifications
    - System announcements and updates
    - User-configurable notification preferences
    - Multi-channel notification delivery (within-app, email simulation)

2.3 Applied OOP Concepts

The RPMS demonstrates sophisticated application of core Object-Oriented Programming principles:

- **Encapsulation**:
    - Data hiding through private attributes with controlled access via getters and setters

    - Implementation of business logic within relevant classes
    - Information hiding to protect sensitive medical data
    - Bundling of related functionality within cohesive classes
    - Creation of immutable objects where appropriate (e.g., medical records)
- **Inheritance**:
    - Hierarchical class structure with common functionality in base classes
    - Specialization through derived classes for specific user roles
    - Method overriding to customize behavior for different user types
    - Abstract base classes to define common interfaces
    - Protected members for controlled access in derived classes
- **Polymorphism**:
    - Dynamic method dispatch for role-specific dashboard rendering
    - Interface implementation for notification systems
    - Method overloading for flexible parameter handling
    - Operator overloading where appropriate
    - Runtime type identification for appropriate handling of different object types
- **Composition**:
    - Complex object construction through component relationships
    - "Has-a" relationships between entities (e.g., Patient has VitalSigns)
    - Delegation of responsibilities to composed objects
    - Loose coupling between components for improved maintainability
    - Favoring composition over inheritance where appropriate
- **Interface Implementation**:
    - Behavior contracts through interface definitions
    - Multiple interface implementation for diverse functionality
    - Dependency injection via interfaces
    - Strategy pattern implementation
    - Service abstraction through interface definitions

3. Literature Review

3.1 Comparative Analysis of Existing Systems

A comprehensive analysis of leading healthcare monitoring systems reveals their relative strengths and limitations, providing context for our RPMS development:

System	Features	Limitations	RPMS Improvements
Apple HealthKit	Advanced vitals tracking, seamless wearable integration, developer API, sophisticated data visualization	Limited clinical interaction, no emergency response system, proprietary ecosystem lock-in	Enhanced doctor-patient interaction interface, real-time emergency alert system, platform independence
MyChart (Epic)	Comprehensive medical record access, appointment scheduling, messaging, bill payment	Limited real-time monitoring, passive data repository, minimal analytical capabilities	Active vital monitoring with threshold alerts, trend analysis visualization, proactive health management
Teladoc	High-quality video consultations, international provider network, 24/7 availability	Minimal health data integration, standalone service rather than integrated platform	Unified data-consultation platform, contextual health data during consultations, comprehensive patient history access
Fitbit Health Solutions	Continuous activity tracking, sleep analysis, heart rate monitoring, extensive historical data	Consumer-grade (non-medical) data, limited clinical integration, minimal provider interaction	Medical-grade data validation, formal clinical feedback loop, integrated provider communication

Omron Healthcare	Specialized cardiovascular monitoring, clinical-grade blood pressure data, trend tracking	Single-system focus, limited integration with broader health metrics, minimal telemedicine	Multi-system health monitoring, integrated consultation platform, comprehensive health dashboard
Medtronic CareLink	Implantable device monitoring, detailed cardiac data, physician alerts	Device-specific, requires specialized hardware, closed ecosystem	Hardware-agnostic design, open data format (CSV), extensible architecture

3.2 Technological Landscape

Current remote monitoring technologies can be categorized into several key domains:

1. **Data Acquisition Systems**:
    - Wearable sensors (smartwatches, patches)
    - Implantable monitors
    - Smart home sensors (weight scales, blood pressure cuffs)
    - Smartphone-based monitoring apps
2. **Data Transmission Technologies**:
    - Bluetooth Low Energy (BLE)
    - Wi-Fi direct connections
    - Cellular data transmission
    - Near Field Communication (NFC)
    - LoRaWAN for remote deployments
3. **Backend Processing Systems**:
    - Cloud-based health data platforms
    - Edge computing solutions for real-time processing
    - Machine learning systems for anomaly detection
    - Rules-based alert systems
4. **User Interface Technologies**:
    - Mobile applications
    - Web portals
    - Desktop applications
    - Voice-assisted interfaces

    - Ambient computing solutions
5. **Integration Standards**:
    - HL7 FHIR for health data exchange
    - DICOM for imaging
    - SNOMED CT for clinical terminology
    - IEEE 11073 for device communication

3.3 Challenges in Current Healthcare Monitoring Solutions

Existing remote monitoring solutions frequently encounter several limitations:

1. **Infrastructure Dependencies**: Many commercial systems rely heavily on proprietary hardware and cloud infrastructure, creating significant cost barriers and accessibility issues, particularly for smaller healthcare providers or resource-constrained environments.
2. **Architectural Rigidity**: Current systems often lack modular design principles, making them difficult to extend or adapt for emerging technologies such as advanced AI/ML capabilities or novel IoT devices without substantial redevelopment efforts.
3. **Interoperability Challenges**: The healthcare technology ecosystem suffers from fragmentation, with many solutions operating in isolated data silos rather than participating in a connected health information exchange.
4. **Usability Constraints**: Many existing systems prioritize technical capabilities over user experience, creating adoption barriers particularly among elderly patients or those with limited technological literacy.
5. **Cost Considerations**: Enterprise healthcare solutions typically involve significant licensing costs, hardware investments, and maintenance expenses, limiting their adoption in resource-constrained settings.

The RPMS addresses these challenges through its Java-based, OOP-focused architecture. By employing standardized CSV formats for data input, the system maintains hardware agnosticism, allowing functionality without specialized devices. The JavaFX-based user interface provides a cost-effective yet visually appealing and intuitive experience. The modular OOP design ensures extensibility and adaptability for future enhancements without architectural overhauls.

Moreover, the educational focus on OOP principles ensures that the system serves not only as a functional tool but also as a learning platform for understanding fundamental software design principles that can inform future healthcare technology development.

4. Methodology – Design and Implementation

4.1 System Architecture

The RPMS architecture was developed with a strong emphasis on modularity, scalability, maintainability, and clear separation of concerns. This approach facilitates system evolution, component reuse, and collaborative development through well-defined interfaces between architectural elements.

4.1.1 Class Structure and OOP Concepts

The system architecture is organized into several distinct modules, each implementing specific OOP principles:

User Management Module:

1. **User (Abstract Class)**: Serves as the foundation for the user hierarchy, encapsulating universal attributes such as username, password, name, contact information, and authentication state. This class demonstrates abstraction by defining common user behaviors while deferring specific implementations to concrete subclasses.

   Key Methods:

   - authenticate(String username, String password): boolean
   - displayDashboard(): void [abstract]
   - updateProfile(UserProfile profile): boolean
   - getRole(): UserRole

2. **Patient (extends User)**: Extends the base User class with patient-specific attributes such as medical history, emergency contacts, allergies, and a collection of vital signs readings. This class demonstrates inheritance by reusing User functionality while adding specialized patient capabilities.

   Key Attributes:

   - vitalsList: List<VitalSigns>
   - appointments: List<Appointment>
   - medicalHistory: MedicalHistory
   - emergencyContact: Contact

Key Methods:

- uploadVitals(File csvFile): boolean
- viewFeedback(): List<Feedback>
- requestAppointment(Doctor doctor, DateTime dateTime): boolean
- displayDashboard(): void [overridden]

3. **Doctor (extends User)**: Inherits from User and adds doctor-specific attributes like specialization, license number, and patient management capabilities. Implements specialized methods for reviewing patient data and providing clinical feedback.

   Key Attributes:

   - specialization: String
   - licenseNumber: String
   - patients: List<Patient>
   - schedule: DoctorSchedule
   - feedbackGiven: List<Feedback>

   Key Methods:

   - reviewPatientData(Patient patient): PatientSummary
   - provideFeedback(Patient patient, String feedback): boolean
   - manageAppointments(): List<Appointment>
   - displayDashboard(): void [overridden]

4. **Administrator (extends User)**: Extends User with system management capabilities, allowing user administration, system monitoring, and report generation.

   Key Attributes:

   - accessLevel: AdminAccessLevel
   - managedSystems: List<SystemModule>

   Key Methods:

   - createUser(User user): boolean
   - generateSystemReport(): Report
   - monitorSystemStatus(): SystemStatus
   - displayDashboard(): void [overridden]

Health Data Module:

1. **VitalSigns (Class)**: Encapsulates individual health measurement readings with appropriate validation and metadata. Demonstrates encapsulation by restricting direct attribute access and providing controlled state changes.

   #### Key Attributes:

   - recordId: String
   - patientId: String
   - dateRecorded: LocalDateTime
   - heartRate: int
   - bloodPressureSystolic: int
   - bloodPressureDiastolic: int
   - oxygenLevel: double
   - temperature: double
   - respiratoryRate: int
   - height: double
   - weight: double

   #### Key Methods:

   - isWithinNormalRange(): boolean
   - requiresImmediate Attention(): boolean
   - getFormattedReading(VitalType type): String

2. **MedicalRecord (Class)**: Maintains a patient's comprehensive medical information, including vitals history, diagnoses, and treatments. Uses composition to incorporate various health data components.

Key Attributes:

- patientId: String
- vitalHistory: List<VitalSigns>
- diagnoses: List<Diagnosis>
- prescriptions: List<Prescription>

Key Methods:

- addVitalReading(VitalSigns vitals): void
- getVitalsForPeriod(LocalDate start, LocalDate end): List<VitalSigns>
- getLatestReading(VitalType type): VitalSigns

Notification System:

1. **Notifiable (Interface)**: Defines the contract for classes capable of sending notifications. Demonstrates interface-based programming for flexible component integration.

Key Methods:

- sendNotification(String recipient, String message): void
- isAvailable(): boolean

2. **EmailNotification (implements Notifiable)**: Implements the Notifiable interface for email-based alerts. Shows interface implementation and the Strategy pattern in notification delivery.

Key Methods:

- sendNotification(String recipient, String message): void [implemented]
- formatEmailBody(String message): String
- validateEmailAddress(String email): boolean

3. **SMSNotification (implements Notifiable)**: Implements Notifiable for SMS-based alerts, demonstrating polymorphism through different notification implementations.

Key Methods:

- sendNotification(String recipient, String message): void [implemented]
- formatSMSBody(String message): String
- validatePhoneNumber(String phone): boolean

4. **NotificationService (Class)**: Orchestrates notification delivery using appropriate channels based on message urgency and user preferences. Uses composition to incorporate different notification methods.

Key Attributes:

- emailNotifier: EmailNotification
- smsNotifier: SMSNotification
- notificationHistory: List<NotificationRecord>

Key Methods:

- notifyUser(User user, String message, Priority priority): boolean
- scheduleNotification(User user, String message, LocalDateTime time): boolean
- broadcastEmergencyAlert(String message, List<User> recipients): void

Appointment System:

1. **Appointment (Class)**: Encapsulates appointment details including participants, scheduling, and status. Shows proper encapsulation of a business entity.

Key Attributes:

- appointmentId: String
- doctor: Doctor
- patient: Patient
- dateTime: LocalDateTime
- status: AppointmentStatus
- notes: String

Key Methods:

- reschedule(LocalDateTime newDateTime): boolean
- cancel(): boolean
- confirm(): boolean
- generateSummary(): String

2. **AppointmentService (Class)**: Manages appointment creation, modification, and querying. Demonstrates the Service pattern for business logic encapsulation.

Key Methods:

- createAppointment(Doctor doctor, Patient patient, LocalDateTime dateTime): Appointment
- getDoctorAppointments(Doctor doctor): List<Appointment>
- getPatientAppointments(Patient patient): List<Appointment>
- updateAppointmentStatus(String appointmentId, AppointmentStatus status): boolean

Communication Module:

1. **ChatSystem (Class)**: Implements real-time text communication between users using Java sockets. Shows application of networking principles within OOP.

Key Attributes:

- activeConnections: Map<String, Socket>
- messageHistory: Map<String, List<Message>>

Key Methods:

- initializeConnection(): boolean
- sendMessage(String sender, String recipient, String message): boolean
- getChatHistory(String user1, String user2): List<Message>
- closeConnection(String userId): void

2. **VideoConsultation (Class)**: Generates and manages video conference links for remote consultations.

Key Methods:

- generateMeetingLink(): String
- sendInvitation(String[] participants, LocalDateTime time): boolean
- recordMetadata(String meetingId, String purpose): void

Data Visualization Module:

1. **TrendVisualizer (Class)**: Creates JavaFX chart representations of patient health data. Demonstrates separation of data processing from presentation.

Key Methods:

- createVitalsLineChart(List<VitalSigns> data, VitalType type): LineChart
- highlightAbnormalReadings(Chart chart, List<VitalSigns> data): void
- exportChartAsImage(Chart chart, File destination): boolean

2. **ReportGenerator (Class)**: Produces comprehensive health reports based on patient data and clinical feedback.

Key Methods:

- generatePatientReport(Patient patient, LocalDate startDate, LocalDate endDate): Report
- generateSystemUsageReport(LocalDate startDate, LocalDate endDate): Report
- exportToPDF(Report report, File destination): boolean

4.1.2 System Workflow

The RPMS implements a comprehensive workflow that facilitates a complete lifecycle of healthcare monitoring:

1. **Authentication and Session Initialization**:
    - Users access the system through a secure login screen
    - Credentials are verified against the user database
    - Role-specific session parameters are loaded
    - Appropriate dashboard is displayed based on user role
2. **Patient Workflow**:
    - Health data upload via CSV file selection
    - File validation for format and value ranges
    - Data visualization of historical trends
    - Appointment request submission
    - Doctor feedback and prescription review
    - Document upload for medical records
    - Emergency alert triggering if needed
3. **Doctor Workflow**:
    - Patient list review with vital status indicators
    - Detailed patient data examination
    - Historical trend analysis for selected patients
    - Feedback and prescription creation
    - Appointment scheduling and management
    - Emergency alert response and documentation
    - Communication with patients via chat/video
4. **Administrator Workflow**:
    - User account management (creation, modification, deactivation)
    - System usage monitoring and analytics
    - Report generation and export
    - Configuration management
    - Audit log review and security monitoring
5. **Emergency Alert Process**:
    - Automatic monitoring of uploaded vital signs
    - Threshold comparison against medical standards
    - Immediate notification generation for critical values
    - Alert escalation based on severity
    - Response documentation and follow-up triggers

6. **Appointment Management Process**:
    - Patient initiates appointment request
    - Doctor receives notification and reviews request
    - Appointment confirmation or rescheduling
    - Calendar update for both parties
    - Reminder generation at configured intervals
    - Post-appointment feedback collection

4.2 Implementation Details

4.2.1 JavaFX GUI Development

The RPMS user interface was developed using JavaFX, providing a responsive, intuitive, and visually appealing experience across different user roles:

JavaFX Configuration and Setup:

- The application utilizes JavaFX 17, configured directly through the IDE without Maven dependency management
- Scene Builder was used for initial layout design, with programmatic refinements
- Custom CSS styling was applied to create a consistent, professional healthcare interface
- The application follows a Scene-Controller architecture with FXML definitions where appropriate

Scene Hierarchy:

- Root BorderPane layout for primary application structure
- SplitPane for dashboard organization with collapsible navigation
- ScrollPane containers for content areas to accommodate varying content sizes
- Custom dialog implementations for interactive processes

Custom Controls:

- Development of specialized patient vitals display components
- Custom chart extensions for medical data visualization
- Tailored TableView implementations with medical status indicators
- Role-specific data entry forms with validation

Key UI Components:

1. **Login Screen**:
    - Clean, minimalist design with healthcare branding
    - Role selection through ToggleButtons
    - Secure password field with visibility toggle
    - Input validation with real-time feedback
    - Animated transitions to role-specific dashboards
2. **Patient Dashboard**:
    - Prominent vital signs summary with status indicators
    - Intuitive navigation through tabbed interface
    - Visual alerts for abnormal readings
    - Quick-access appointment scheduling
    - Clear feedback display with collapsible details
    - CSV upload area with drag-and-drop support
3. **Doctor Dashboard**:
    - Patient list with vital status indicators
    - Comprehensive patient history viewer
    - Multi-parameter charting for trend analysis
    - Structured feedback and prescription forms
    - Integrated communication tools
    - Appointment calendar with availability management
4. **Admin Dashboard**:
    - System statistics overview with key metrics
    - User management interface with role-based filtering
    - Report generation tools with parameter selection
    - Configuration panel for system settings
    - Audit log viewer with search and filter capabilities
5. **Visualization Screens**:
    - Interactive line charts for temporal analysis
    - Multi-series comparison capabilities
    - Zoom and pan functionality for detailed exploration
    - Data point inspection on hover/click
    - Export options for sharing and documentation
6. **Communication Interfaces**:
    - Text chat with message history
    - File attachment capabilities
    - Video call initiation interface

    - Meeting link generation and sharing
    - Contact selection and availability indication

4.2.2 Event Handling and User Interaction

The RPMS implements a comprehensive event handling architecture to create responsive, intuitive user interactions:

EventBus Implementation:

- Custom EventBus for system-wide event distribution
- Event subscription and publication mechanisms
- Type-safe event handling with generics
- Priority-based event processing

User Input Handling:

- Real-time form validation using ChangeListener and binding
- Input formatting with TextFormatter for specialized medical values
- Contextual help tooltips triggered by focus events
- Error highlighting with CSS pseudoclass state management

Asynchronous Operations:

- Background processing using Task and Service abstractions
- Progress indication for long-running operations
- Non-blocking UI updates with Platform.runLater()
- Cancellable operations where appropriate

Observable Collections:

- LiveData implementation for automatically updated views
- ObservableList bindings for TableView and ListView components
- Real-time filtering using FilteredList and Predicates
- Sorting capabilities with SortedList wrappers

Animation and Transitions:

- Subtle feedback animations for user actions
- Smooth scene transitions between application states
- Alert notifications with fade effects
- Progress indicators for background operations

4.2.3 File Handling for CSV Processing

The system implements robust CSV processing for vital signs data:

CSV File Structure:

- Standardized header format with vital sign parameter names
- Required columns for patient ID, timestamp, and key measurements
- Optional metadata columns for contextual information
- Support for different decimal separators (locale awareness)

CSV Processing Pipeline:

1. File selection through JavaFX FileChooser with CSV extension filtering
2. Preliminary format validation (headers, column count)
3. Row-by-row parsing with detailed error collection
4. Data type conversion with robust error handling
5. Medical range validation for each vital parameter
6. Batch processing for efficient handling of large files

Validation Logic:

- Heart rate range verification (40-200 bpm)
- Blood pressure validation (systolic 70-200 mmHg, diastolic 40-120 mmHg)
- Temperature bounds checking (35-43°C)
- Oxygen saturation validation (70-100%)
- Respiratory rate verification (8-40 breaths per minute)
- Height and weight plausibility checks

Error Handling:

- Detailed error reporting with line numbers and column identifiers
- Graceful handling of format inconsistencies
- Partial file processing with warning flags
- Error aggregation for comprehensive feedback
- Suggestions for corrective actions

Sample CSV Validation Code Structure:

```
private List<ValidationError> validateVitalsCSV(File csvFile) {

        List<ValidationError> errors = new ArrayList<>();
```

```
        try (BufferedReader reader = new BufferedReader(new FileReader(csvFile))) {
        String headerLine = reader.readLine();
        // Validate header structure
        if (!validateHeader(headerLine)) {
        errors.add(new ValidationError(-1, "Invalid header format"));
        return errors;
        }
        String line;
        int lineNumber = 1;
        while ((line = reader.readLine()) != null) {
        lineNumber++;
        String[] values = line.split(",");
        // Validate line structure
        if (values.length != EXPECTED_COLUMN_COUNT) {
                errors.add(new ValidationError(lineNumber,
                "Expected " + EXPECTED_COLUMN_COUNT + " columns, found " +
values.length));
                continue;
        }
        // Validate individual measurements
        try {
                double heartRate = Double.parseDouble(values[HEART_RATE_INDEX]);
                if (heartRate < 40 || heartRate > 200) {
                errors.add(new ValidationError(lineNumber, HEART_RATE_INDEX,
                "Heart rate out of range: " + heartRate));
                }
```

```
                // Additional validations for other vital parameters
                // ...
            } catch (NumberFormatException e) {
                errors.add(new ValidationError(lineNumber,
                "Invalid number format in line"));
            }
            }
            } catch (IOException e) {
            errors.add(new ValidationError(-1, "Error reading file: " + e.getMessage()));
            }
            return errors;
}
```

4.2.4 Data Visualization Techniques

The RPMS implements sophisticated data visualization using JavaFX's charting capabilities:

Chart Types and Implementation:

- **LineChart**: For temporal vital signs trend visualization
- **BarChart**: For comparative analysis across time periods
- **PieChart**: For distribution analysis of appointment types, patient conditions, etc.
- **AreaChart**: For stacked visualization of related parameters
- **ScatterChart**: For correlation analysis between different vital parameters

Customization and Extensions:

- Custom chart styling through CSS and programmatic customization
- Extended chart classes with healthcare-specific functionality
- Interactive data point inspection with tooltips
- Threshold line overlays for normal range indication

- Anomaly highlighting with different colors/markers

Advanced Features:

- Multi-series visualization for comparing different vital signs
- Dynamic time range adjustment through DatePicker integration
- Zoom and pan capabilities for detailed examination
- Annotations for significant medical events
- Statistical overlay options (mean, median, trend lines)

Data Preparation:

- Time-series aggregation for different granularities (hourly, daily, weekly)
- Moving averages calculation for trend smoothing
- Outlier detection and optional filtering
- Missing data interpolation where appropriate
- Unit conversion for consistent visualization

Chart Interaction:

- Selection of data points for detailed information
- Click-through navigation to related records
- Legend toggling for series visibility control
- Export functionality (PNG, PDF)
- Customizable view settings

4.2.5 Chat and Video Consultation Implementation

The communication module provides essential telemedicine capabilities:

Text Chat Implementation:

- Java Socket programming for real-time bidirectional communication
- Client-server architecture with multithreading for concurrent chats
- Message queuing for reliable delivery
- Read receipts and typing indicators
- Chat history persistence in the database
- Message encryption for privacy

4.2.6 Database Integration:

project connects to MySQL through JDBC (Java Database Connectivity), allowing JavaFX application to store and retrieve hospital data like patient records,

appointments, and medical histories. The database maintains data integrity through proper relationships between tables (patients, doctors, appointments, medications, etc.). JavaFX controllers contain methods that execute SQL queries to perform operations like registering patients, scheduling appointments, and generating medical reports, with prepared statements ensuring security against SQL injection.

5- Outcome Images

Outcome samples, including screenshots, charts, or other visual outputs, are available in the supplementary materials provided with the original submission. These demonstrate the systems functionality and user interface.

6- CSV samples

Sample CSV files used for vital signs data input are included in the supplementary materials provided with the original submission.

7. Future Extensions of the Project

The RPMS is designed with scalability in mind, allowing for several future enhancements to improve its functionality and impact:

- **AI/ML Integration**: Machine learning models can be incorporated to analyze historical vitals data and predict health risks, such as the likelihood of a heart attack based on trends in heart rate and blood pressure. For example, a logistic regression model could be trained on patient data to classify risk levels, with predictions displayed on the doctor’s dashboard.
- **IoT Integration**: The system can be extended to connect with wearable devices like smartwatches or pulse oximeters, enabling real-time data collection. This would replace manual CSV uploads with automated data streaming, using protocols like Bluetooth or MQTT to interface with IoT devices.
- **Cloud Storage and Scalability**: Transitioning to a cloud-based database (e.g., using MySQL on AWS) would enhance data accessibility, allowing patients and doctors to access the system from anywhere. This would also improve scalability, supporting a larger user base without performance degradation.

- **Advanced Analytics and Visualization**: Incorporating more sophisticated data visualization, such as 3D graphs or heatmaps, would provide deeper insights into patient health trends. Predictive analytics could be added to forecast future health metrics based on current data, using time-series analysis techniques like ARIMA.
- **Mobile Application Support**: Developing a mobile app version of the RPMS using JavaFX or Android SDK would improve accessibility, allowing patients to upload vitals and receive alerts on their smartphones.
- **Integration with Electronic Health Records (EHRs)**: The system could be integrated with existing EHR systems like Epic or Cerner, enabling seamless data sharing with hospitals and clinics. This would require implementing HL7 FHIR standards for interoperability.

These extensions would transform the RPMS into a more comprehensive healthcare solution, addressing both current limitations and future needs in remote patient monitoring.

8. Conclusion

The Remote Patient Monitoring System (RPMS) successfully meets all requirements of the CS-212 Object-Oriented Programming semester project, delivering a functional, modular, and user-friendly application. The system leverages OOP principles Encapsulation, Inheritance, Polymorphism, Composition, and Interfaces to create a scalable architecture that addresses real-world healthcare challenges. Key features, including health data monitoring, doctor-patient interaction, emergency alerts, and report generation, were implemented using Java, JavaFX, and Java APIs, ensuring compliance with the project guidelines.

The development process adhered to professional ethics and software engineering standards, using a modern IDE (IntelliJ IDEA) and including comprehensive documentation via JavaDocs and comments. The system's design allows for future enhancements, such as AI/ML integration and IoT connectivity, positioning it as a foundation for advanced healthcare solutions. This project not only fulfills the course learning outcomes but also demonstrates the practical application of OOP in solving real-world problems, contributing to the broader field of healthcare technology.