6.Design Patterns

6.1 Abstract Factory-Antea Koxherri & Flavia Koco

The class diagram represents a diagnostic testing module within a healthcare system. This module is responsible for creating and processing two types of diagnostic services:

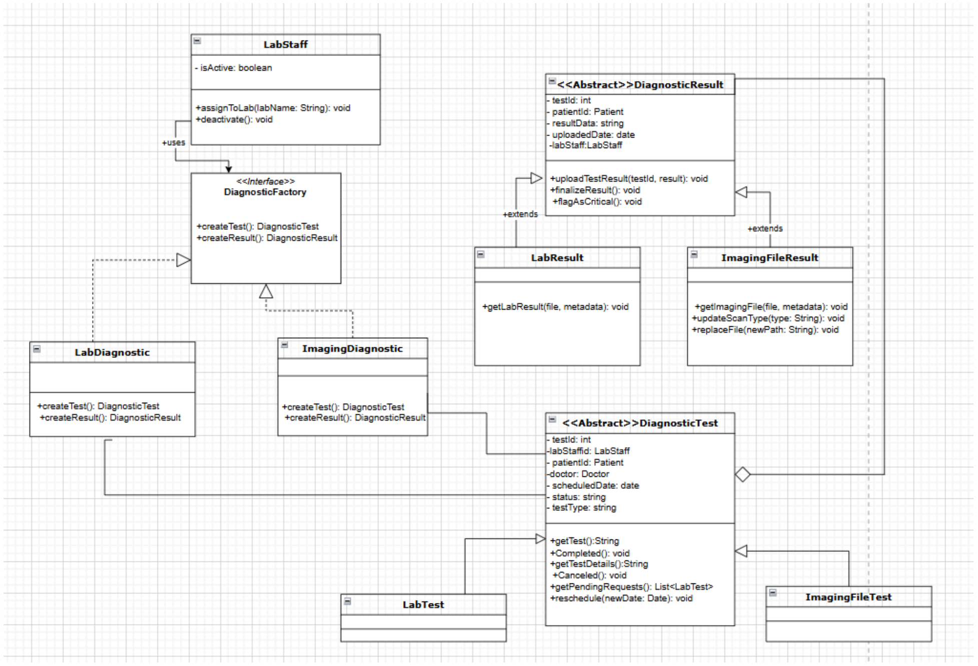
- **Lab Diagnostics** (e.g., blood tests, urinalysis)
- **Imaging Diagnostics** (e.g., X-rays, MRIs)

These services differ in how they produce and handle test results, yet share common behaviors and structures. To manage this complexity and enforce consistency, the **Abstract Factory Pattern** is used.

Why Abstract Factory?

The **Abstract Factory Pattern** is used here to:

1. **Encapsulate the creation logic** of diagnostic tests and results based on diagnostic type (Lab or Imaging).
2. **Separate system concerns** — the LabStaff class only knows how to use a DiagnosticFactory, not the details of each diagnostic type.
3. **Promote consistency** — each factory ensures the right type of test/result pair is created.
4. **Make the system scalable** — new diagnostic types (e.g., GeneticTestingDiagnostic) can be added with minimal changes by introducing new factories.
5. **Reduce tight coupling** — LabStaff doesn't need to know the exact concrete classes like LabTest or ImagingFileTest. It works through abstraction.


LabStaff
- isActive: boolean
+assignToLab(labName: String): void
+deactivate(): void
+uses
<<Interface>>
DiagnosticFactory
+createTest(): DiagnosticTest
+createResult(): DiagnosticResult
LabDiagnostic
+createTest(): DiagnosticTest
+createResult(): DiagnosticResult
ImagingDiagnostic
+createTest(): DiagnosticTest
+createResult(): DiagnosticResult
<<Abstract>>DiagnosticResult
- testId: int
- patientId: Patient
- resultData: string
- uploadedDate: date
-labStaff:LabStaff
+uploadTestResult(testId, result): void
+finalizeResult(): void
+flagAsCritical(): void
+extends
+extends
LabResult
+getLabResult(file, metadata): void
ImagingFileResult
+getImagingFile(file, metadata): void
+updateScanType(type: String): void
+replaceFile(newPath: String): void
<<Abstract>>DiagnosticTest
- testId: int
-labStaffid: LabStaff
- patientId: Patient
-doctor: Doctor
- scheduledDate: date
- status: string
- testType: string
+getTest():String
+Completed(): void
+getTestDetails():String
+Canceled(): void
+getPendingRequests(): List<LabTest>
+reschedule(newDate: Date): void
LabTest
ImagingFileTest

6.2 Prototype Pattern- Belina Durmishi

Justification for Using the Prototype Design Pattern in Selected Classes
The Prototype Design Pattern allows for efficient object creation by duplicating existing instances rather than constructing them from scratch. This is particularly beneficial in systems where object configuration is repetitive, resource-intensive, or involves minimal variation across instances. The following classes within the healthcare system architecture are well-suited for applying this pattern:

1. Report
Rationale: Reports maintain a consistent format across different patients and time intervals. Often, generating a new report involves reusing the structure of a previous report with minimal changes.

Example Scenario: A clinician generates a new monthly progress report for a patient by duplicating a previous report and modifying fields such as patientId, dateGenerated, and progressScore.

2. Consultation
Rationale: Consultations, especially recurring or follow-up sessions, typically involve similar content and structure with minor updates.

Example Scenario: A nutritionist creates a follow-up consultation by cloning an earlier session record, updating the date, and appending new notes.

3. EducationalMaterial
Rationale: Educational content is frequently reused across patients, with occasional modifications to tailor the material for specific needs.

Example Scenario: A standard dietary guide is cloned and personalized by a nutritionist before assigning it to a new patient.

4. Medication
Rationale: Medications often share the same core properties (e.g., name, manufacturer) while differing in attributes like dosage or expiry.

Example Scenario: A previously defined medication is duplicated, and only the dosage and expiry date are updated for a new prescription.

5. DietaryRecord
Rationale: Dietary records for patients often follow predictable and repetitive structures, especially in planned meal tracking.

Example Scenario: A nutritionist duplicates a dietary record from the previous week and updates it with new meal entries and revised BMI values.

6. Meal

Rationale: Standardized meals (e.g., heart-healthy breakfast) are reused across patient plans, typically with slight modifications.

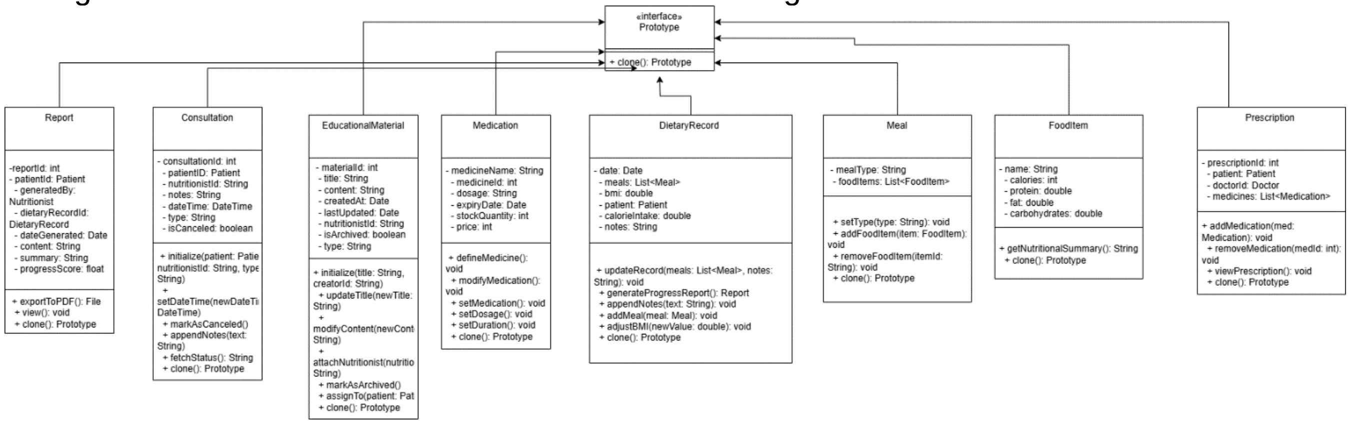
Example Scenario: A nutritionist clones a template meal and adjusts its contents by substituting one or two food items.

7. FoodItem
Rationale: Nutritional attributes of food items such as calories, protein, or fat remain constant, making them ideal for reuse through cloning.

Example Scenario: A common food item like “boiled egg” is cloned and included in multiple different meals without re-entering its nutritional data.

8. Prescription
Rationale: Prescriptions for chronic conditions often consist of recurring medications with occasional adjustments.

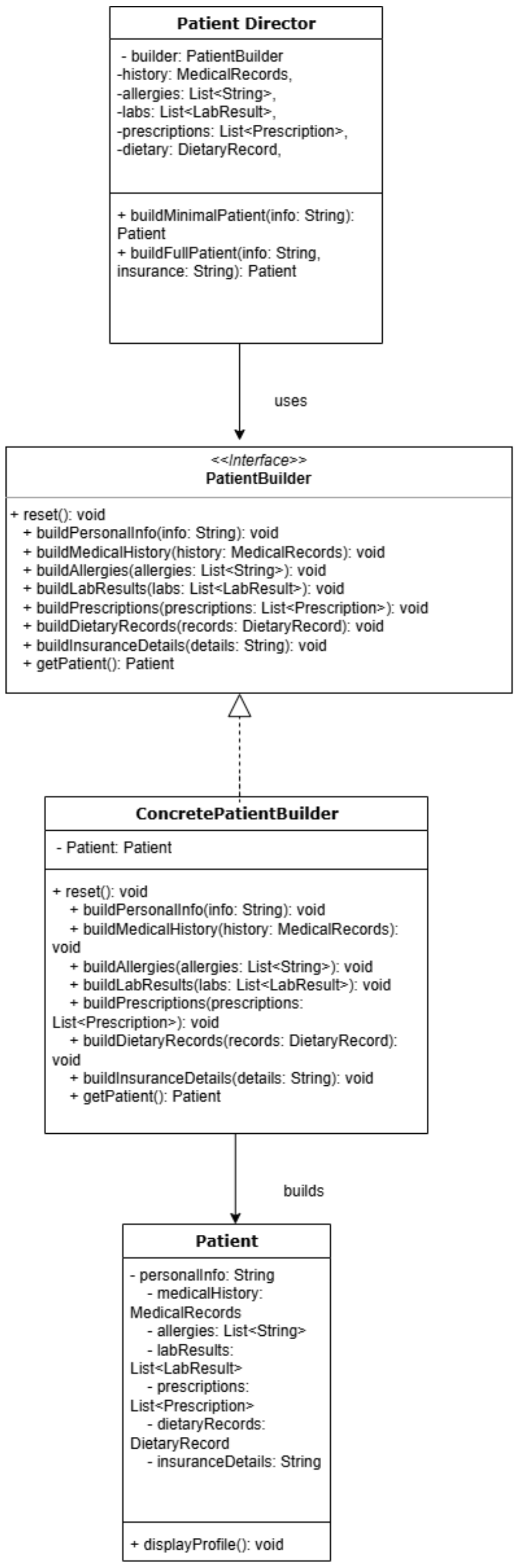
Example Scenario: A doctor replicates a patient’s previous prescription and modifies the dosage of one medication to reflect recent clinical changes.



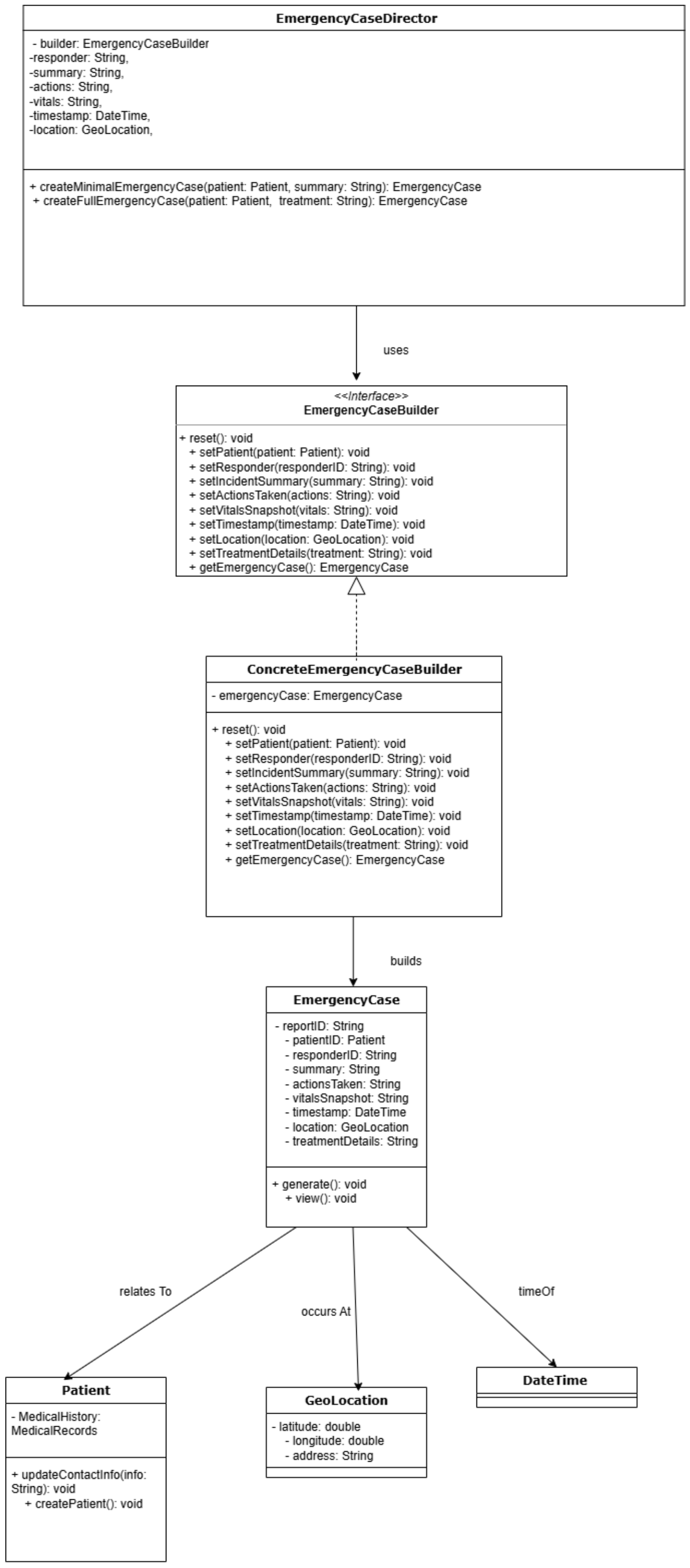
6.3 Builder Pattern

Evelina Gace

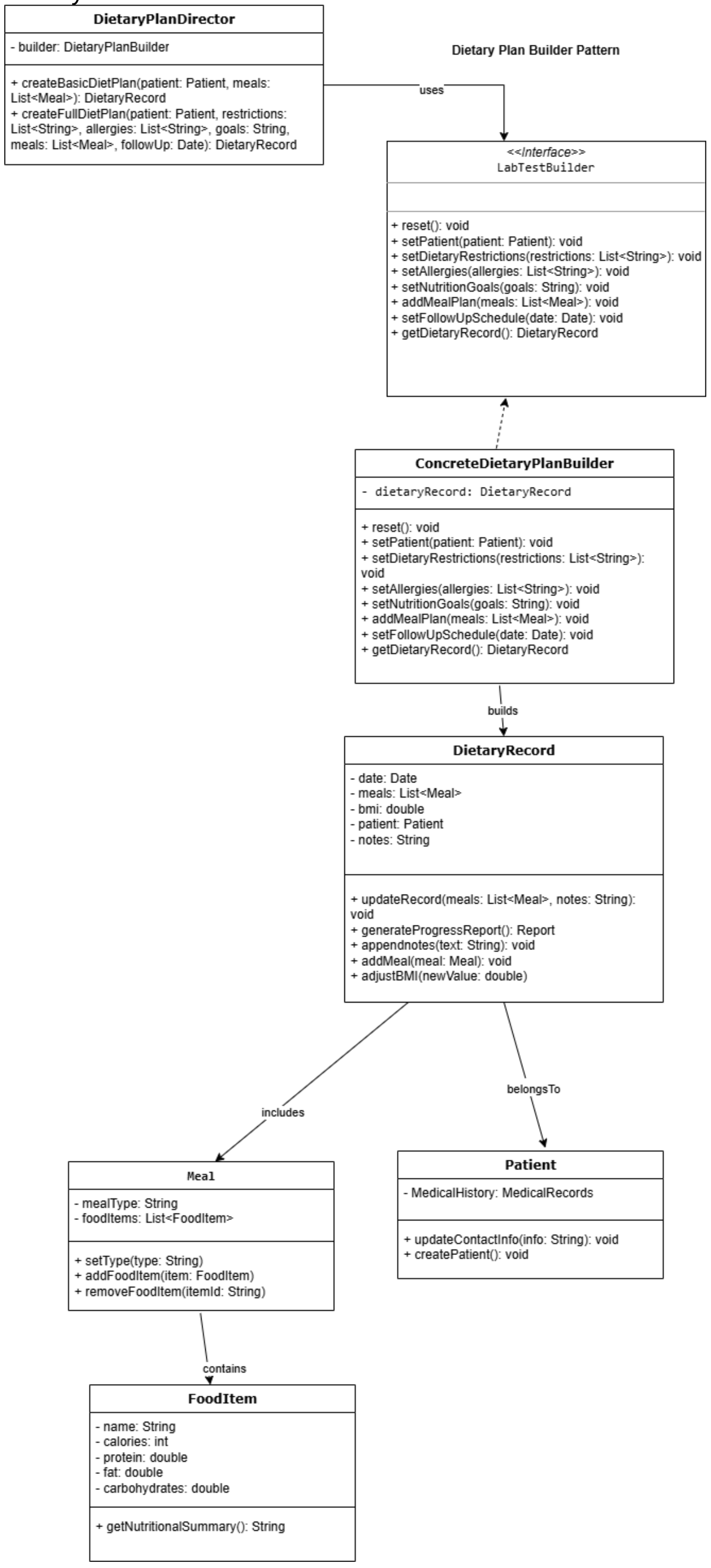
Patient Builder Pattern


Patient Director
- builder: PatientBuilder
-history: MedicalRecords,
-allergies: List<String>,
-labs: List<LabResult>,
-prescriptions: List<Prescription>,
-dietary: DietaryRecord,
+ buildMinimalPatient(info: String): Patient
+ buildFullPatient(info: String, insurance: String): Patient
uses
<<Interface>>
PatientBuilder
+ reset(): void
+ buildPersonalInfo(info: String): void
+ buildMedicalHistory(history: MedicalRecords): void
+ buildAllergies(allergies: List<String>): void
+ buildLabResults(labs: List<LabResult>): void
+ buildPrescriptions(prescriptions: List<Prescription>): void
+ buildDietaryRecords(records: DietaryRecord): void
+ buildInsuranceDetails(details: String): void
+ getPatient(): Patient
ConcretePatientBuilder
- Patient: Patient
+ reset(): void
+ buildPersonalInfo(info: String): void
+ buildMedicalHistory(history: MedicalRecords): void
+ buildAllergies(allergies: List<String>): void
+ buildLabResults(labs: List<LabResult>): void
+ buildPrescriptions(prescriptions: List<Prescription>): void
+ buildDietaryRecords(records: DietaryRecord): void
+ buildInsuranceDetails(details: String): void
+ getPatient(): Patient
builds
Patient
- personalInfo: String
- medicalHistory: MedicalRecords
- allergies: List<String>
- labResults: List<LabResult>
- prescriptions: List<Prescription>
- dietaryRecords: DietaryRecord
- insuranceDetails: String
+ displayProfile(): void

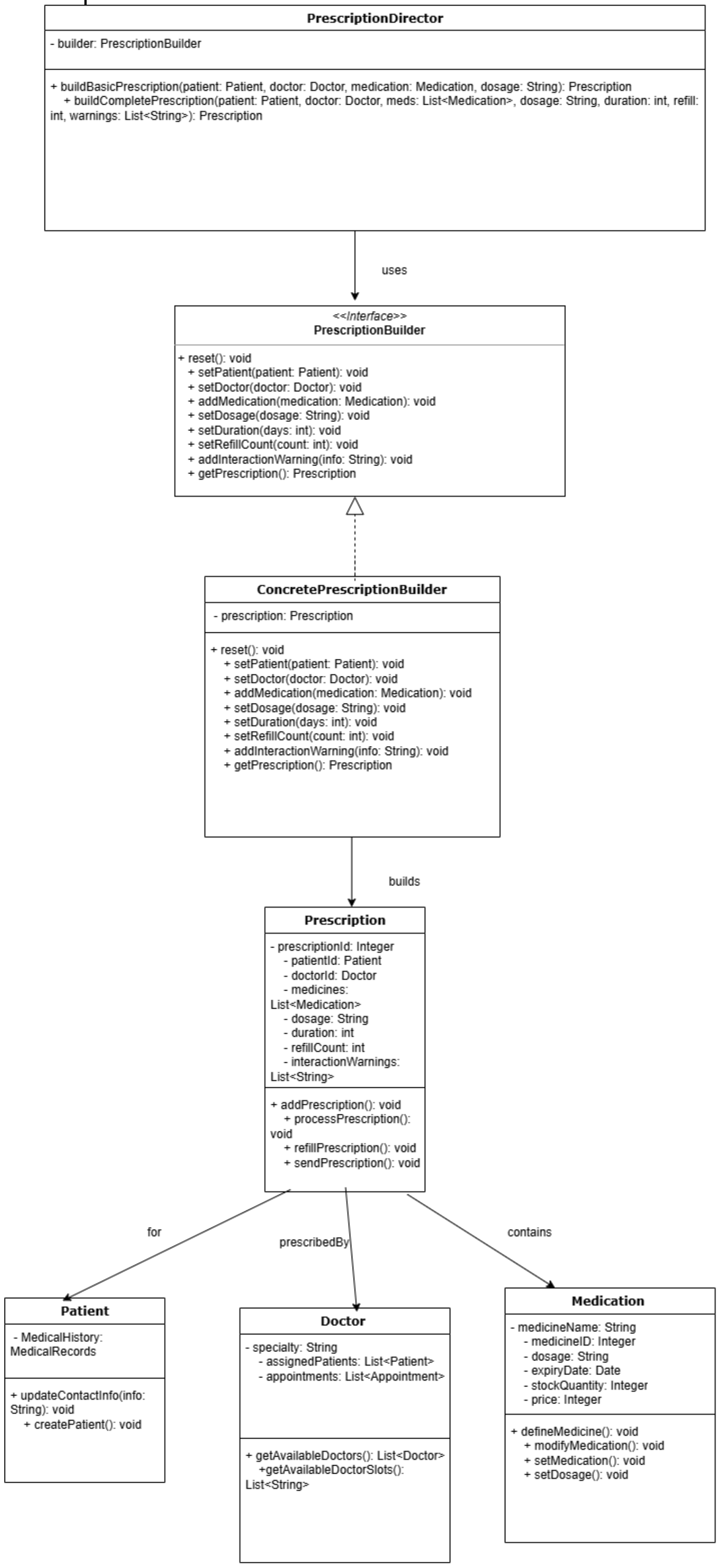
Emergency Case Report Builder Pattern


EmergencyCaseDirector
- builder: EmergencyCaseBuilder
-responder: String,
-summary: String,
-actions: String,
-vitals: String,
-timestamp: DateTime,
-location: GeoLocation,
+ createMinimalEmergencyCase(patient: Patient, summary: String): EmergencyCase
+ createFullEmergencyCase(patient: Patient, treatment: String): EmergencyCase
uses
<<Interface>>
EmergencyCaseBuilder
+ reset(): void
+ setPatient(patient: Patient): void
+ setResponder(responderID: String): void
+ setIncidentSummary(summary: String): void
+ setActionsTaken(actions: String): void
+ setVitalsSnapshot(vitals: String): void
+ setTimestamp(timestamp: DateTime): void
+ setLocation(location: GeoLocation): void
+ setTreatmentDetails(treatment: String): void
+ getEmergencyCase(): EmergencyCase
ConcreteEmergencyCaseBuilder
- emergencyCase: EmergencyCase
+ reset(): void
+ setPatient(patient: Patient): void
+ setResponder(responderID: String): void
+ setIncidentSummary(summary: String): void
+ setActionsTaken(actions: String): void
+ setVitalsSnapshot(vitals: String): void
+ setTimestamp(timestamp: DateTime): void
+ setLocation(location: GeoLocation): void
+ setTreatmentDetails(treatment: String): void
+ getEmergencyCase(): EmergencyCase
builds
EmergencyCase
- reportID: String
- patientID: Patient
- responderID: String
- summary: String
- actionsTaken: String
- vitalsSnapshot: String
- timestamp: DateTime
- location: GeoLocation
- treatmentDetails: String
+ generate(): void
+ view(): void
relates To
occurs At
timeOf
Patient
- MedicalHistory: MedicalRecords
+ updateContactInfo(info: String): void
+ createPatient(): void
GeoLocation
- latitude: double
- longitude: double
- address: String
DateTime

Dietary Plan Builder Pattern

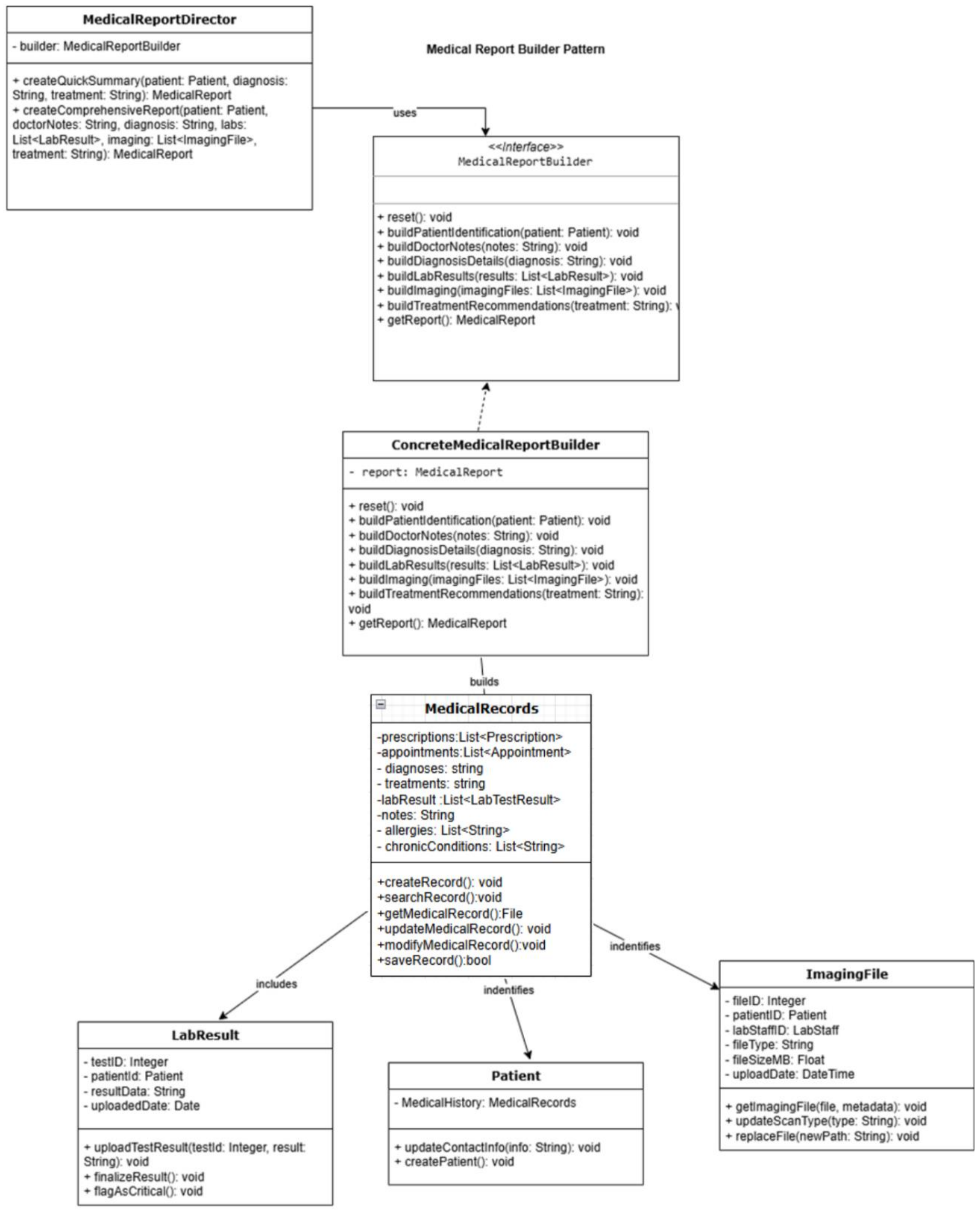

Dietary Plan Builder Pattern
DietaryPlanDirector
- builder: DietaryPlanBuilder
+ createBasicDietPlan(patient: Patient, meals: List<Meal>): DietaryRecord
+ createFullDietPlan(patient: Patient, restrictions: List<String>, allergies: List<String>, goals: String, meals: List<Meal>, followUp: Date): DietaryRecord
uses
<<Interface>>
LabTestBuilder
+ reset(): void
+ setPatient(patient: Patient): void
+ setDietaryRestrictions(restrictions: List<String>): void
+ setAllergies(allergies: List<String>): void
+ setNutritionGoals(goals: String): void
+ addMealPlan(meals: List<Meal>): void
+ setFollowUpSchedule(date: Date): void
+ getDietaryRecord(): DietaryRecord
ConcreteDietaryPlanBuilder
- dietaryRecord: DietaryRecord
+ reset(): void
+ setPatient(patient: Patient): void
+ setDietaryRestrictions(restrictions: List<String>): void
+ setAllergies(allergies: List<String>): void
+ setNutritionGoals(goals: String): void
+ addMealPlan(meals: List<Meal>): void
+ setFollowUpSchedule(date: Date): void
+ getDietaryRecord(): DietaryRecord
builds
DietaryRecord
- date: Date
- meals: List<Meal>
- bmi: double
- patient: Patient
- notes: String
+ updateRecord(meals: List<Meal>, notes: String): void
+ generateProgressReport(): Report
+ appendnotes(text: String): void
+ addMeal(meal: Meal): void
+ adjustBMI(newValue: double)
includes
belongsTo
Meal
- mealType: String
- foodItems: List<FoodItem>
+ setType(type: String)
+ addFoodItem(item: FoodItem)
+ removeFoodItem(itemId: String)
Patient
- MedicalHistory: MedicalRecords
+ updateContactInfo(info: String): void
+ createPatient(): void
contains
FoodItem
- name: String
- calories: int
- protein: double
- fat: double
- carbohydrates: double
+ getNutritionalSummary(): String

Prescription Builder Pattern

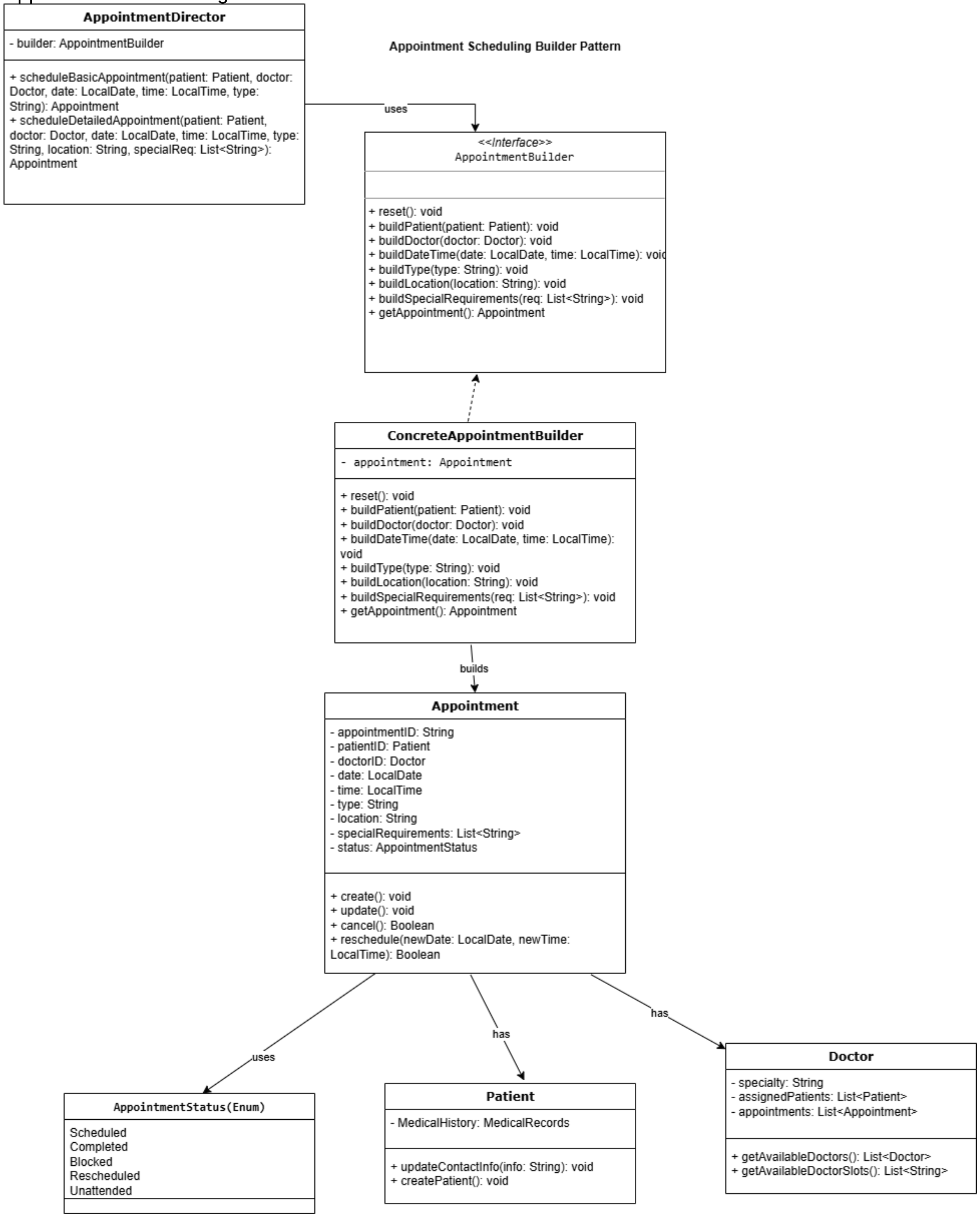

PrescriptionDirector
- builder: PrescriptionBuilder
+ buildBasicPrescription(patient: Patient, doctor: Doctor, medication: Medication, dosage: String): Prescription
+ buildCompletePrescription(patient: Patient, doctor: Doctor, meds: List<Medication>, dosage: String, duration: int, refill: int, warnings: List<String>): Prescription
uses
<<Interface>>
PrescriptionBuilder
+ reset(): void
+ setPatient(patient: Patient): void
+ setDoctor(doctor: Doctor): void
+ addMedication(medication: Medication): void
+ setDosage(dosage: String): void
+ setDuration(days: int): void
+ setRefillCount(count: int): void
+ addInteractionWarning(info: String): void
+ getPrescription(): Prescription
ConcretePrescriptionBuilder
- prescription: Prescription
+ reset(): void
+ setPatient(patient: Patient): void
+ setDoctor(doctor: Doctor): void
+ addMedication(medication: Medication): void
+ setDosage(dosage: String): void
+ setDuration(days: int): void
+ setRefillCount(count: int): void
+ addInteractionWarning(info: String): void
+ getPrescription(): Prescription
builds
Prescription
- prescriptionId: Integer
- patientId: Patient
- doctorId: Doctor
- medicines: List<Medication>
- dosage: String
- duration: int
- refillCount: int
- interactionWarnings: List<String>
+ addPrescription(): void
+ processPrescription(): void
+ refillPrescription(): void
+ sendPrescription(): void
for
prescribedBy
contains
Patient
- MedicalHistory: MedicalRecords
+ updateContactInfo(info: String): void
+ createPatient(): void
Doctor
- specialty: String
- assignedPatients: List<Patient>
- appointments: List<Appointment>
+ getAvailableDoctors(): List<Doctor>
+getAvailableDoctorSlots(): List<String>
Medication
- medicineName: String
- medicineID: Integer
- dosage: String
- expiryDate: Date
- stockQuantity: Integer
- price: Integer
+ defineMedicine(): void
+ modifyMedication(): void
+ setMedication(): void
+ setDosage(): void

Elisona Doku

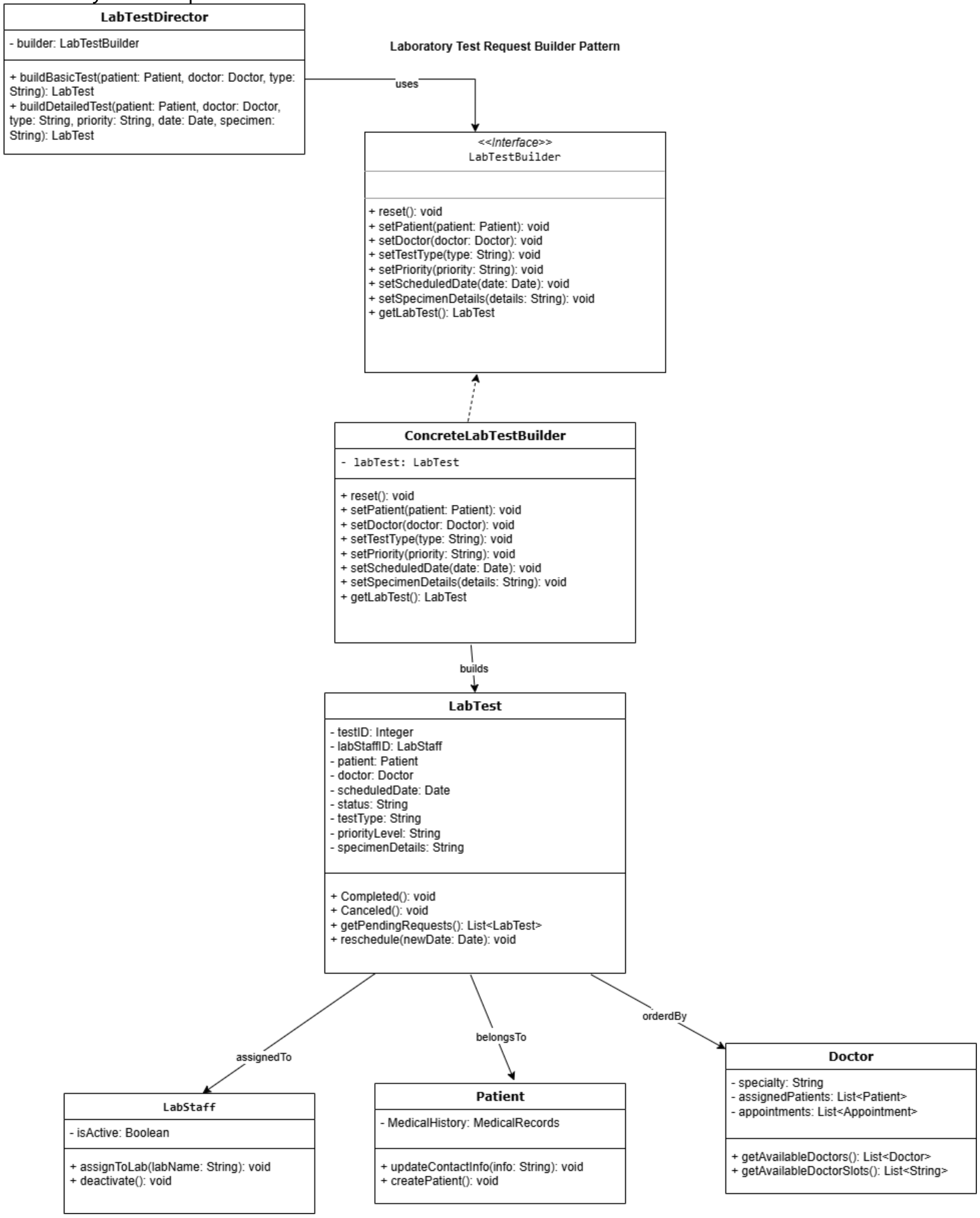
Medical Report Builder Pattern:


Medical Report Builder Pattern
MedicalReportDirector
- builder: MedicalReportBuilder
+ createQuickSummary(patient: Patient, diagnosis: String, treatment: String): MedicalReport
+ createComprehensiveReport(patient: Patient, doctorNotes: String, diagnosis: String, labs: List<LabResult>, imaging: List<ImagingFile>, treatment: String): MedicalReport
uses
<<Interface>>
MedicalReportBuilder
+ reset(): void
+ buildPatientIdentification(patient: Patient): void
+ buildDoctorNotes(notes: String): void
+ buildDiagnosisDetails(diagnosis: String): void
+ buildLabResults(results: List<LabResult>): void
+ buildImaging(imagingFiles: List<ImagingFile>): void
+ buildTreatmentRecommendations(treatment: String):
+ getReport(): MedicalReport
ConcreteMedicalReportBuilder
- report: MedicalReport
+ reset(): void
+ buildPatientIdentification(patient: Patient): void
+ buildDoctorNotes(notes: String): void
+ buildDiagnosisDetails(diagnosis: String): void
+ buildLabResults(results: List<LabResult>): void
+ buildImaging(imagingFiles: List<ImagingFile>): void
+ buildTreatmentRecommendations(treatment: String): void
+ getReport(): MedicalReport
builds
MedicalRecords
-prescriptions:List<Prescription>
-appointments:List<Appointment>
- diagnoses: string
- treatments: string
-labResult :List<LabTestResult>
-notes: String
- allergies: List<String>
- chronicConditions: List<String>
+createRecord(): void
+searchRecord():void
+getMedicalRecord():File
+updateMedicalRecord(): void
+modifyMedicalRecord():void
+saveRecord():bool
includes
indentifies
indentifies
LabResult
- testID: Integer
- patientId: Patient
- resultData: String
- uploadedDate: Date
+ uploadTestResult(testId: Integer, result: String): void
+ finalizeResult(): void
+ flagAsCritical(): void
Patient
- MedicalHistory: MedicalRecords
+ updateContactInfo(info: String): void
+ createPatient(): void
ImagingFile
- fileID: Integer
- patientID: Patient
- labStaffID: LabStaff
- fileType: String
- fileSizeMB: Float
- uploadDate: DateTime
+ getImagingFile(file, metadata): void
+ updateScanType(type: String): void
+ replaceFile(newPath: String): void

Appointment Scheduling Builder Pattern:


Appointment Scheduling Builder Pattern
AppointmentDirector
- builder: AppointmentBuilder
+ scheduleBasicAppointment(patient: Patient, doctor: Doctor, date: LocalDate, time: LocalTime, type: String): Appointment
+ scheduleDetailedAppointment(patient: Patient, doctor: Doctor, date: LocalDate, time: LocalTime, type: String, location: String, specialReq: List<String>): Appointment
uses
<<Interface>>
AppointmentBuilder
+ reset(): void
+ buildPatient(patient: Patient): void
+ buildDoctor(doctor: Doctor): void
+ buildDateTime(date: LocalDate, time: LocalTime): void
+ buildType(type: String): void
+ buildLocation(location: String): void
+ buildSpecialRequirements(req: List<String>): void
+ getAppointment(): Appointment
ConcreteAppointmentBuilder
- appointment: Appointment
+ reset(): void
+ buildPatient(patient: Patient): void
+ buildDoctor(doctor: Doctor): void
+ buildDateTime(date: LocalDate, time: LocalTime): void
+ buildType(type: String): void
+ buildLocation(location: String): void
+ buildSpecialRequirements(req: List<String>): void
+ getAppointment(): Appointment
builds
Appointment
- appointmentID: String
- patientID: Patient
- doctorID: Doctor
- date: LocalDate
- time: LocalTime
- type: String
- location: String
- specialRequirements: List<String>
- status: AppointmentStatus
+ create(): void
+ update(): void
+ cancel(): Boolean
+ reschedule(newDate: LocalDate, newTime: LocalTime): Boolean
uses
has
has
AppointmentStatus(Enum)
Scheduled
Completed
Blocked
Rescheduled
Unattended
Patient
- MedicalHistory: MedicalRecords
+ updateContactInfo(info: String): void
+ createPatient(): void
Doctor
- specialty: String
- assignedPatients: List<Patient>
- appointments: List<Appointment>
+ getAvailableDoctors(): List<Doctor>
+ getAvailableDoctorSlots(): List<String>

Laboratory Test Request Builder Pattern:

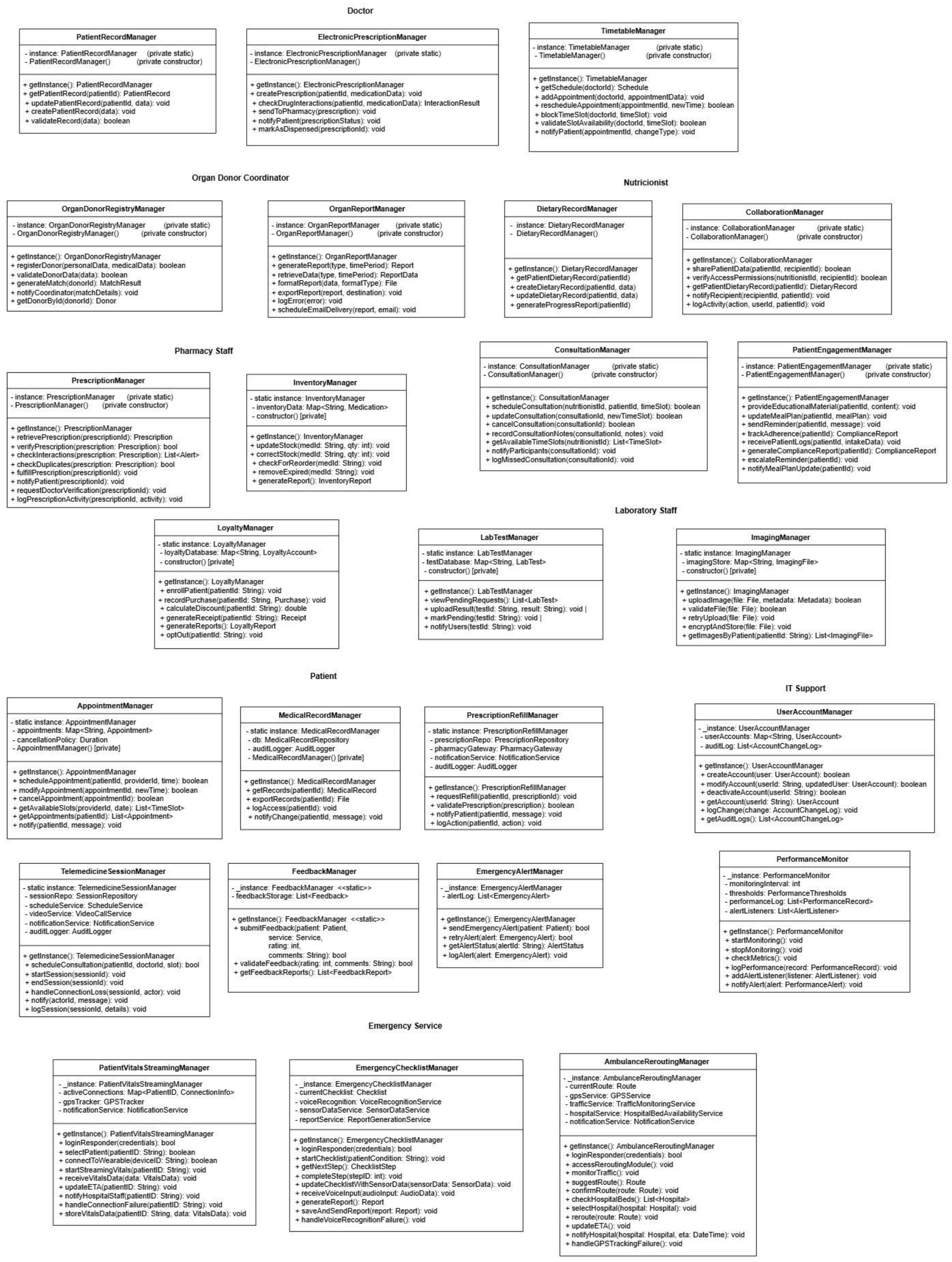

Laboratory Test Request Builder Pattern
LabTestDirector
- builder: LabTestBuilder
+ buildBasicTest(patient: Patient, doctor: Doctor, type: String): LabTest
+ buildDetailedTest(patient: Patient, doctor: Doctor, type: String, priority: String, date: Date, specimen: String): LabTest
uses
<<Interface>>
LabTestBuilder
+ reset(): void
+ setPatient(patient: Patient): void
+ setDoctor(doctor: Doctor): void
+ setTestType(type: String): void
+ setPriority(priority: String): void
+ setScheduledDate(date: Date): void
+ setSpecimenDetails(details: String): void
+ getLabTest(): LabTest
ConcreteLabTestBuilder
- labTest: LabTest
+ reset(): void
+ setPatient(patient: Patient): void
+ setDoctor(doctor: Doctor): void
+ setTestType(type: String): void
+ setPriority(priority: String): void
+ setScheduledDate(date: Date): void
+ setSpecimenDetails(details: String): void
+ getLabTest(): LabTest
builds
LabTest
- testID: Integer
- labStaffID: LabStaff
- patient: Patient
- doctor: Doctor
- scheduledDate: Date
- status: String
- testType: String
- priorityLevel: String
- specimenDetails: String
+ Completed(): void
+ Canceled(): void
+ getPendingRequests(): List<LabTest>
+ reschedule(newDate: Date): void
assignedTo
belongsTo
orderdBy
LabStaff
- isActive: Boolean
+ assignToLab(labName: String): void
+ deactivate(): void
Patient
- MedicalHistory: MedicalRecords
+ updateContactInfo(info: String): void
+ createPatient(): void
Doctor
- specialty: String
- assignedPatients: List<Patient>
- appointments: List<Appointment>
+ getAvailableDoctors(): List<Doctor>
+ getAvailableDoctorSlots(): List<String>

6.4 Factory Method

Elkier Ago:

6.5 Singeleton

Liza Koliqi:


Doctor
PatientRecordManager
- instance: PatientRecordManager (private static)
- PatientRecordManager() (private constructor)
+ getInstance(): PatientRecordManager
+ getPatientRecord(patientId): PatientRecord
+ updatePatientRecord(patientId, data): void
+ createPatientRecord(data): void
+ validateRecord(data): boolean
ElectronicPrescriptionManager
- instance: ElectronicPrescriptionManager (private static)
- ElectronicPrescriptionManager()
+ getInstance(): ElectronicPrescriptionManager
+ createPrescription(patientId, medicationData): void
+ checkDrugInteractions(patientId, medicationData): InteractionResult
+ sendToPharmacy(prescription): void
+ notifyPatient(prescriptionStatus): void
+ markAsDispensed(prescriptionId): void
TimetableManager
- instance: TimetableManager (private static)
- TimetableManager() (private constructor)
+ getInstance(): TimetableManager
+ getSchedule(doctorId): Schedule
+ addAppointment(doctorId, appointmentData): void
+ rescheduleAppointment(appointmentId, newTime): boolean
+ blockTimeSlot(doctorId, timeSlot): void
+ validateSlotAvailability(doctorId, timeSlot): boolean
+ notifyPatient(appointmentId, changeType): void
Organ Donor Coordinator
OrganDonorRegistryManager
- instance: OrganDonorRegistryManager (private static)
- OrganDonorRegistryManager() (private constructor)
+ getInstance(): OrganDonorRegistryManager
+ registerDonor(personalData, medicalData): boolean
+ validateDonorData(data): boolean
+ generateMatch(donorId): MatchResult
+ notifyCoordinator(matchDetails): void
+ getDonorById(donorId): Donor
OrganReportManager
- instance: OrganReportManager (private static)
- OrganReportManager() (private constructor)
+ getInstance(): OrganReportManager
+ generateReport(type, timePeriod): Report
+ retrieveData(type, timePeriod): ReportData
+ formatReport(data, formatType): File
+ exportReport(report, destination): void
+ logError(error): void
+ scheduleEmailDelivery(report, email): void
Nutricionist
DietaryRecordManager
- instance: DietaryRecordManager
- DietaryRecordManager()
+ getInstance(): DietaryRecordManager
+ getPatientDietaryRecord(patientId)
+ createDietaryRecord(patientId, data)
+ updateDietaryRecord(patientId, data)
+ generateProgressReport(patientId)
CollaborationManager
- instance: CollaborationManager (private static)
- CollaborationManager() (private constructor)
+ getInstance(): CollaborationManager
+ sharePatientData(patientId, recipientId): boolean
+ verifyAccessPermissions(nutritionistId, recipientId): boolean
+ getPatientDietaryRecord(patientId): DietaryRecord
+ notifyRecipient(recipientId, patientId): void
+ logActivity(action, userId, patientId): void
ConsultationManager
- instance: ConsultationManager (private static)
- ConsultationManager() (private constructor)
+ getInstance(): ConsultationManager
+ scheduleConsultation(nutritionistId, patientId, timeSlot): boolean
+ updateConsultation(consultationId, newTimeSlot): boolean
+ cancelConsultation(consultationId): boolean
+ recordConsultationNotes(consultationId, notes): void
+ getAvailableTimeSlots(nutritionistId): List<TimeSlot>
+ notifyParticipants(consultationId): void
+ logMissedConsultation(consultationId): void
PatientEngagementManager
- instance: PatientEngagementManager (private static)
- PatientEngagementManager() (private constructor)
+ getInstance(): PatientEngagementManager
+ provideEducationalMaterial(patientId, content): void
+ updateMealPlan(patientId, mealPlan): void
+ sendReminder(patientId, message): void
+ trackAdherence(patientId): ComplianceReport
+ receivePatientLogs(patientId, intakeData): void
+ generateComplianceReport(patientId): ComplianceReport
+ escalateReminder(patientId): void
+ notifyMealPlanUpdate(patientId): void
Pharmacy Staff
PrescriptionManager
- instance: PrescriptionManager (private static)
- PrescriptionManager() (private constructor)
+ getInstance(): PrescriptionManager
+ retrievePrescription(prescriptionId): Prescription
+ verifyPrescription(prescription: Prescription): bool
+ checkInteractions(prescription: Prescription): List<Alert>
+ checkDuplicates(prescription: Prescription): bool
+ fulfillPrescription(prescriptionId): void
+ notifyPatient(prescriptionId): void
+ requestDoctorVerification(prescriptionId): void
+ logPrescriptionActivity(prescriptionId, activity): void
InventoryManager
- static instance: InventoryManager
- inventoryData: Map<String, Medication>
- constructor() [private]
+ getInstance(): InventoryManager
+ updateStock(medId: String, qty: int): void
+ correctStock(medId: String, qty: int): void
+ checkForReorder(medId: String): void
+ removeExpired(medId: String): void
+ generateReport(): InventoryReport
LoyaltyManager
- static instance: LoyaltyManager
- loyaltyDatabase: Map<String, LoyaltyAccount>
- constructor() [private]
+ getInstance(): LoyaltyManager
+ enrollPatient(patientId: String): void
+ recordPurchase(patientId: String, Purchase): void
+ calculateDiscount(patientId: String): double
+ generateReceipt(patientId: String): Receipt
+ generateReports(): LoyaltyReport
+ optOut(patientId: String): void
Laboratory Staff
LabTestManager
- static instance: LabTestManager
- testDatabase: Map<String, LabTest>
- constructor() [private]
+ getInstance(): LabTestManager
+ viewPendingRequests(): List<LabTest>
+ uploadResult(testId: String, result: String): void |
+ markPending(testId: String): void |
+ notifyUsers(testId: String): void
ImagingManager
- static instance: ImagingManager
- imagingStore: Map<String, ImagingFile>
- constructor() [private]
+ getInstance(): ImagingManager
+ uploadImage(file: File, metadata: Metadata): boolean
+ validateFile(file: File): boolean
+ retryUpload(file: File): void
+ encryptAndStore(file: File): void
+ getImagesByPatient(patientId: String): List<ImagingFile>
Patient
AppointmentManager
- static instance: AppointmentManager
- appointments: Map<String, Appointment>
- cancellationPolicy: Duration
- AppointmentManager() [private]
+ getInstance(): AppointmentManager
+ scheduleAppointment(patientId, providerId, time): boolean
+ modifyAppointment(appointmentId, newTime): boolean
+ cancelAppointment(appointmentId): boolean
+ getAvailableSlots(providerId, date): List<TimeSlot>
+ getAppointments(patientId): List<Appointment>
+ notify(patientId, message): void
MedicalRecordManager
- static instance: MedicalRecordManager
- db: MedicalRecordRepository
- auditLogger: AuditLogger
- MedicalRecordManager() [private]
+ getInstance(): MedicalRecordManager
+ getRecords(patientId): MedicalRecord
+ exportRecords(patientId): File
+ logAccess(patientId): void
+ notifyChange(patientId, message): void
PrescriptionRefillManager
- static instance: PrescriptionRefillManager
- prescriptionRepo: PrescriptionRepository
- pharmacyGateway: PharmacyGateway
- notificationService: NotificationService
- auditLogger: AuditLogger
+ getInstance(): PrescriptionRefillManager
+ requestRefill(patientId, prescriptionId): void
+ validatePrescription(prescription): boolean
+ notifyPatient(patientId, message): void
+ logAction(patientId, action): void
TelemedicineSessionManager
- static instance: TelemedicineSessionManager
- sessionRepo: SessionRepository
- scheduleService: ScheduleService
- videoService: VideoCallService
- notificationService: NotificationService
- auditLogger: AuditLogger
+ getInstance(): TelemedicineSessionManager
+ scheduleConsultation(patientId, doctorId, slot): bool
+ startSession(sessionId): void
+ endSession(sessionId): void
+ handleConnectionLoss(sessionId, actor): void
+ notify(actorId, message): void
+ logSession(sessionId, details): void
FeedbackManager
- _instance: FeedbackManager <<static>>
- feedbackStorage: List<Feedback>
+ getInstance(): FeedbackManager <<static>>
+ submitFeedback(patient: Patient,
service: Service,
rating: int,
comments: String): bool
+ validateFeedback(rating: int, comments: String): bool
+ getFeedbackReports(): List<FeedbackReport>
EmergencyAlertManager
- _instance: EmergencyAlertManager
- alertLog: List<EmergencyAlert>
+ getInstance(): EmergencyAlertManager
+ sendEmergencyAlert(patient: Patient): bool
+ retryAlert(alert: EmergencyAlert): bool
+ getAlertStatus(alertId: String): AlertStatus
+ logAlert(alert: EmergencyAlert): void
IT Support
UserAccountManager
- _instance: UserAccountManager
- userAccounts: Map<String, UserAccount>
- auditLog: List<AccountChangeLog>
+ getInstance(): UserAccountManager
+ createAccount(user: UserAccount): boolean
+ modifyAccount(userId: String, updatedUser: UserAccount): boolean
+ deactivateAccount(userId: String): boolean
+ getAccount(userId: String): UserAccount
+ logChange(change: AccountChangeLog): void
+ getAuditLogs(): List<AccountChangeLog>
PerformanceMonitor
- _instance: PerformanceMonitor
- monitoringInterval: int
- thresholds: PerformanceThresholds
- performanceLog: List<PerformanceRecord>
- alertListeners: List<AlertListener>
+ getInstance(): PerformanceMonitor
+ startMonitoring(): void
+ stopMonitoring(): void
+ checkMetrics(): void
+ logPerformance(record: PerformanceRecord): void
+ addAlertListener(listener: AlertListener): void
+ notifyAlert(alert: PerformanceAlert): void
Emergency Service
PatientVitalsStreamingManager
- _instance: PatientVitalsStreamingManager
- activeConnections: Map<PatientID, ConnectionInfo>
- gpsTracker: GPSTracker
- notificationService: NotificationService
+ getInstance(): PatientVitalsStreamingManager
+ loginResponder(credentials): bool
+ selectPatient(patientID: String): boolean
+ connectToWearable(deviceID: String): boolean
+ startStreamingVitals(patientID: String): void
+ receiveVitalsData(data: VitalsData): void
+ updateETA(patientID: String): void
+ notifyHospitalStaff(patientID: String): void
+ handleConnectionFailure(patientID: String): void
+ storeVitalsData(patientID: String, data: VitalsData): void
EmergencyChecklistManager
- _instance: EmergencyChecklistManager
- currentChecklist: Checklist
- voiceRecognition: VoiceRecognitionService
- sensorDataService: SensorDataService
- reportService: ReportGenerationService
+ getInstance(): EmergencyChecklistManager
+ loginResponder(credentials): bool
+ startChecklist(patientCondition: String): void
+ getNextStep(): ChecklistStep
+ completeStep(stepID: int): void
+ updateChecklistWithSensorData(sensorData: SensorData): void
+ receiveVoiceInput(audioInput: AudioData): void
+ generateReport(): Report
+ saveAndSendReport(report: Report): void
+ handleVoiceRecognitionFailure(): void
AmbulanceReroutingManager
- _instance: AmbulanceReroutingManager
- currentRoute: Route
- gpsService: GPSService
- trafficService: TrafficMonitoringService
- hospitalService: HospitalBedAvailabilityService
- notificationService: NotificationService
+ getInstance(): AmbulanceReroutingManager
+ loginResponder(credentials): bool
+ accessReroutingModule(): void
+ monitorTraffic(): void
+ suggestRoute(): Route
+ confirmRoute(route: Route): void
+ checkHospitalBeds(): List<Hospital>
+ selectHospital(hospital: Hospital): void
+ reroute(route: Route): void
+ updateETA(): void
+ notifyHospital(hospital: Hospital, eta: DateTime): void
+ handleGPSTrackingFailure(): void