**Sistema de Gestión Hospitalaria - Explicación General**

El sistema desarrollado es una aplicación orientada a objetos que modela la gestión integral de un hospital, implementando las operaciones fundamentales para administrar pacientes, personal médico, instalaciones y citas médicas.

**Arquitectura del Sistema**

**Estructura Jerárquica:** El sistema está organizado alrededor de una entidad central Hospital que contiene múltiples Departamentos especializados. Cada departamento agrupa médicos de la misma especialidad y gestiona sus propias salas médicas.

**Herencia y Polimorfismo**: Se implementa una clase abstracta Persona que sirve como base para Paciente y Medico, compartiendo atributos comunes como datos personales, tipo de sangre y validaciones. Esta arquitectura permite extensibilidad para agregar otros tipos de personal hospitalario (por ejemplo administrativos).

**Gestión de Relaciones:** El sistema maneja relaciones bidireccionales sincronizadas entre entidades. Por ejemplo, cuando se asigna un médico a un departamento, ambos objetos actualizan sus referencias mutuas automáticamente, manteniendo la consistencia de datos.

**Funcionalidades Principales**

**Gestión de Pacientes:** Cada paciente recibe automáticamente una historia clínica única con identificador generado (HC-DNI-AÑO). La historia permite registrar diagnósticos, tratamientos y alergias de forma incremental.

**Sistema de Citas Médicas:** La clase Cita actúa como entidad asociativa, resolviendo la relación muchos-a-muchos entre pacientes, médicos y salas. Incluye validaciones de negocio como verificación de disponibilidad y compatibilidad de especialidades. ¿Cómo seria la PK de esa tabla? No puede ser paciente-medico-sala, hay qye agregarle otro atributo mas como el Fecha de atención.

**Validaciones de Integridad**: El sistema implementa múltiples capas de validación:

* Formato de datos (DNI, matrícula médica, números de identificación)
* Reglas de negocio (fechas futuras, costos positivos, disponibilidad)
* Consistencia referencial (especialidad médico-departamento-sala)

**Persistencia y Recuperación**

**Serialización CSV:** Se implementa un sistema de persistencia personalizado que convierte objetos complejos a formato CSV, manejando caracteres especiales y referencias entre entidades. La carga reversa reconstruye el grafo de objetos manteniendo todas las relaciones.

Serializo para grabar y deserializo cuando lo traigo del disco para Java

**Gestión de Estado:** El CitaManager mantiene índices optimizados (ConcurrentHashMap) para búsquedas eficientes por paciente, médico o sala, evitando recorridos lineales en operaciones frecuentes.

**Aspectos de Diseño**

**Principios SOLID:**

* **Responsabilidad Única:** Cada clase tiene un propósito específico bien definido
* **Abierto/Cerrado:** Las enumeraciones permiten agregar especialidades sin modificar código existente
* **Sustitución de Liskov:** Los subtipos de Persona son intercambiables en contextos apropiados
* **Segregación de Interfaces:** CitaService define un contrato específico sin métodos innecesarios. La idea es crear un paquete que se llame “Servicio” y ahí se pone la interfaz y su implementación, por lo que se va a tener u los datos por un lado y los servicios por otro.

**Patrones Implementados:**

* **Service Layer:** Separación entre lógica de negocio (CitaManager) y modelo de datos
* **Builder implícito:** A través de constructores validados
* **Template Method:** En la clase abstracta Persona. Parecido a las clases abstractas, se tiene un elemento común arriba y las implemetaciones polimoficamente están abajo

**Manejo de Errores**

El sistema define excepciones específicas (CitaException) que proporcionan contexto detallado sobre fallos de validación. Los errores se manejan de forma granular, permitiendo recuperación parcial en operaciones batch.

**Escalabilidad y Extensibilidad**

La arquitectura permite expansiones naturales:

* Nuevas especialidades médicas mediante enumeraciones
* Tipos adicionales de personal extendiendo Persona
* Nuevos estados de cita sin impacto en lógica existente
* Integración con bases de datos reemplazando la capa de persistencia CSV

El sistema demuestra un diseño robusto que balancea simplicidad de uso con flexibilidad técnica, proporcionando una base sólida para un sistema de gestión hospitalaria más complejo.

**Sistema de Gestión Hospitalaria - Análisis Técnico Detallado**

**1. Arquitectura del Sistema**

**Diseño por Capas**

**El sistema implementa una arquitectura multicapa claramente definida:**

**Capa de Entidades (Domain Layer):**

* Contiene 13 clases que representan el modelo de negocio hospitalario
* Implementa validaciones de dominio en los constructores
* Mantiene invariantes de negocio mediante métodos privados de validación.
* Utiliza immutabilidad estratégica (campos final para identificadores). Por ejemplo, el nombre, el tipo de sangre no cambian. Al ser inmutables no puede existir un setter.

**Recordando:**

**Pasaje por valor**: paso el objeto

**Pasaje por referencia:** paso la dirección donde esta guardado ese objeto. Ej: arrays

**Capa de Servicios (Service Layer):**

* CitaService define el contrato de operaciones de negocio
* CitaManager implementa la lógica de coordinación entre entidades
* Maneja transacciones de negocio complejas (programación de citas)
* Implementa patrones de índices para optimización de consultas

**Capa de Persistencia:**

* Sistema de serialización personalizado a CSV
* Manejo de referencias circulares mediante mapas de lookup
* Estrategia de reconstrucción de grafos de objetos

**Patrones de Diseño Implementados**

**Template Method en Persona:**

**java**

**public abstract class Persona {**

***// Algoritmo común de validación***

**protected void validarString(String valor, String mensajeError)**

**protected void validarDni(String dni)**

***// Las subclases extienden con validaciones específicas***

**}**

**Strategy Pattern en Especialidades**: Las enumeraciones encapsulan comportamientos específicos por dominio, permitiendo extensibilidad sin modificación de código existente.

**Observer Pattern Implícito:** Las relaciones bidireccionales mantienen sincronización automática:

**java**

**public void setHospital(Hospital hospital) {**

**if (this.hospital != hospital) {**

***// Desregistro del hospital anterior***

***// Registro en el nuevo hospital***

***// Actualización bidireccional***

**}**

**}**

**2. Modelo de Datos y Relaciones**

**Herencia Estratégica**

La clase Persona abstrae características comunes pero permite especialización:

* **Atributos compartidos:** identificación, datos biométricos, validaciones base
* **Comportamientos específicos:** Paciente maneja historia clínica, Medico gestiona especialización

**Composición vs Agregación**

**Composición (ciclo de vida dependiente):**

* **Paciente → HistoriaClinica:** Se crea automáticamente, no puede existir independientemente
* **Medico → Matricula:** Validación específica, lifecycle acoplado

**Agregación (ciclo de vida independiente):**

* **Hospital → Departamento:** Los departamentos pueden reasignarse
* **Departamento → Sala:** Las salas pueden reconfigurarse

**Clase Asociativa Compleja**

Cita resuelve múltiples relaciones many-to-many simultáneamente:

* **Paciente ↔ Medico (N:M)**
* **Medico ↔ Sala (N:M)**
* **Paciente ↔ Sala (N:M)**

Además incorpora:

* Atributos propios (fecha, costo, observaciones)
* Estados de negocio (EstadoCita)
* Validaciones de consistencia temporal y espacial. (ej: si ya esta dado un turno no puedo “pisarlo”, tampoco puedo otorgar turnos si ya paso la fecha, etc)

**3. Validaciones de Integridad Multi-Nivel**

**Nivel Sintáctico (Formato):**

java

private String validarDni(String dni) {

if (!dni.matches("\\d{7,8}")) {

throw new IllegalArgumentException("El DNI debe tener 7 u 8 dígitos");

}

}

**Nivel Semántico (Reglas de Negocio):**

java

private void validarCita(LocalDateTime fechaHora, BigDecimal costo) {

if (fechaHora.isBefore(LocalDateTime.now())) {

throw new CitaException("No se puede programar una cita en el pasado");

}

*// Múltiples validaciones de coherencia*

}

Nivel de Consistencia (Integridad Referencial):

java

if (!medico.getEspecialidad().equals(sala.getDepartamento().getEspecialidad())) {

throw new CitaException("Especialidad incompatible"); cuando se trata de agendar una cita en un lugar que no corresponde

}

**4. Gestión de Estado y Concurrencia**

**Índices de Rendimiento**

El CitaManager mantiene estructuras de datos optimizadas:

java

private final Map<Paciente, List<Cita>> citasPorPaciente = new ConcurrentHashMap<>();

private final Map<Medico, List<Cita>> citasPorMedico = new ConcurrentHashMap<>();

private final Map<Sala, List<Cita>> citasPorSala = new ConcurrentHashMap<>();

**Ventajas:**

* Búsquedas O(1) en lugar de O(n)
* Thread-safety mediante ConcurrentHashMap
* Consistencia automática mediante métodos de actualización privados

**Gestión de Disponibilidad**

Algoritmo de detección de conflictos temporales:

java

private boolean esMedicoDisponible(Medico medico, LocalDateTime fechaHora) {

*// Verifica superposición temporal con margen de 2 horas*

*// Implementa lógica de ventana de disponibilidad*

}

**5. Persistencia y Serialización**

**Estrategia de Serialización Personalizada**

El sistema implementa un protocolo CSV que maneja:

**Serialización de Referencias:**

java

public String toCsvString() {

return String.format("%s,%s,%s,%s,%s,%s,%s",

paciente.getDni(), *// Referencia por clave natural*

medico.getDni(), *// Evita dependencias circulares*

sala.getNumero(), *// Identificador único*

fechaHora.toString(), *// Formato ISO estándar*

costo.toString(), *// Precisión decimal preservada*

estado.name(), *// Enum serialization*

observaciones.replaceAll(",", ";")); *// Escape de caracteres*

}

**Deserialización con Reconstrucción de Grafos:**

java

public static Cita fromCsvString(String csvString,

Map<String, Paciente> pacientes,

Map<String, Medico> medicos,

Map<String, Sala> salas) {

*// Parsing robusto con validación de integridad*

*// Reconstrucción de referencias mediante lookup tables*

*// Manejo de errores con contexto específico*

}

**6. Manejo de Errores y Recuperación**

**Jerarquía de Excepciones Específicas**

java

public class CitaException extends Exception {

*// Proporciona contexto específico del dominio médico*

*// Permite manejo granular de diferentes tipos de error*

*// Facilita logging y debugging*

}

**Estrategias de Validación Fail-Fast**

Las validaciones ocurren en el momento de construcción del objeto, no en uso posterior:

* Constructor validation para invariantes
* Método validation para operaciones complejas
* Estado validation para transiciones de estado

**7. Extensibilidad y Mantenibilidad**

**Puntos de Extensión Diseñados**

**Nuevas Especialidades:**

java

public enum EspecialidadMedica {

*// Agregar nuevas sin modificar código existente*

NUEVA\_ESPECIALIDAD("Descripción")

}

**Nuevos Tipos de Personal:**

java

public class Enfermero extends Persona {

*// Herencia natural de comportamientos base*

*// Especialización específica del rol*

}

**Nuevos Estados de Cita:**

java

public enum EstadoCita {

*// Estados adicionales sin impacto en lógica existente*

REAGENDADA("Reagendada")

}

**Principios de Diseño Aplicados**

**Single Responsibility Principle:**

* **HistoriaClinica:** Solo gestiona información médica del paciente
* **CitaManager:** Solo coordina operaciones de citas
* **Matricula:** Solo valida y almacena credenciales médicas

**Open/Closed Principle:**

* Extensible mediante herencia (Persona)
* Extensible mediante enumeraciones
* Cerrado para modificación de comportamientos base

**Dependency Inversion:**

* CitaManager depende de abstracción (CitaService)
* Permite testing mediante mocks
* Facilita implementaciones alternativas

**8. Características Avanzadas del Sistema**

**Generación Automática de Identificadores**

java

private String generarNumeroHistoria() {

return "HC-" + paciente.getDni() + "-" + fechaCreacion.getYear();

*// Formato: HC-12345678-2025*

*// Garantiza unicidad temporal por paciente*

}

**Cálculos de Negocio Integrados**

java

public int getEdad() {

return LocalDate.now().getYear() - fechaNacimiento.getYear();

*// Cálculo dinámico, siempre actualizado*

}

public long getDiasEstadia() {

return java.time.temporal.ChronoUnit.DAYS.between(checkIn, checkOut);

*// Métrica de negocio directamente disponible*

}

**Sincronización Bidireccional Automática**

El sistema mantiene consistencia automática entre relaciones:

* Agregar paciente al hospital actualiza ambas referencias
* Asignar médico a departamento sincroniza automáticamente
* Crear cita actualiza índices en paciente, médico y sala

**9. Análisis de Complejidad y Rendimiento**

**Operaciones de Búsqueda:**

* Por índice: O(1) - getCitasPorPaciente()
* Por filtro: O(n) - búsquedas con criterios complejos
* Carga inicial: O(n) - construcción de índices

**Operaciones de Escritura:**

* Creación simple: O(1) - objetos individuales
* Creación con relaciones: O(k) donde k = número de relaciones
* Validaciones: O(1) - verificaciones algorítmicas constantes

**Consumo de Memoria:**

* Índices duplicados por eficiencia de consulta
* Referencias bidireccionales requieren gestión cuidadosa
* Immutabilidad selectiva reduce bugs pero aumenta footprint

Este sistema demuestra un diseño maduro que equilibra complejidad de implementación con facilidad de uso, proporcionando una base sólida para operaciones hospitalarias reales.

**Elementos Representados:**

**Enumeraciones (3):**

* TipoSangre - Con 8 tipos sanguíneos
* EspecialidadMedica - Con 12 especialidades médicas
* EstadoCita - Con 5 estados de citas

**Clases Principales (10):**

* Hospital - Entidad central
* Departamento - Organización por especialidades
* Persona - Clase abstracta base
* Paciente y Medico - Heredan de Persona
* Matricula y HistoriaClinica - Clases asociadas
* Sala - Espacios médicos
* Cita - Clase asociativa principal
* CitaException - Manejo de errores

**Servicios:**

* CitaService - Interface
* CitaManager - Implementación

**Relaciones Modeladas:**

**Herencia:**

* Persona ← Paciente
* Persona ← Medico

**Composición/Agregación:**

* Hospital contiene Departamentos y Pacientes
* Departamento tiene Médicos y Salas
* Paciente posee HistoriaClinica
* Medico tiene Matricula

**Clase Asociativa:**

* Cita representa la relación muchos-a-muchos entre Paciente, Medico y Sala

**Asociaciones con Enums:**

* Conexiones con TipoSangre, EspecialidadMedica y EstadoCita