Sistema de Gestión Hospitalaria con JPA/Hibernate

**📋 Descripción del Proyecto**

**JpaHospital** es un sistema integral de gestión hospitalaria desarrollado en Java que demuestra la implementación de conceptos avanzados de persistencia con JPA (Jakarta Persistence API) y Hibernate ORM. El sistema modela digitalmente la operación completa de un hospital, incluyendo la gestión de pacientes, médicos, departamentos especializados, salas médicas, historias clínicas y el sistema de programación de citas.

**Contexto del Dominio**

El sistema aborda el complejo dominio de la gestión hospitalaria, donde múltiples entidades interactúan de manera coordinada para proporcionar atención médica de calidad. La arquitectura está diseñada siguiendo principios de **Domain-Driven Design (DDD)**, con especial énfasis en:

* **Integridad de datos médicos**: Cada paciente tiene una única historia clínica inmutable que registra diagnósticos, tratamientos y alergias
* **Gestión de disponibilidad**: Sistema de validación que asegura la disponibilidad de médicos y salas con buffers de tiempo entre citas
* **Especialización médica**: Correspondencia estricta entre las especialidades de médicos, departamentos y salas
* **Trazabilidad**: Relaciones bidireccionales que mantienen la consistencia del modelo de dominio
* **Validaciones de negocio**: Reglas médicas y administrativas aplicadas a nivel de entidad y servicio

**🏗️ Arquitectura del Sistema**

**Modelo de Dominio**

El sistema implementa un modelo de dominio rico con las siguientes entidades principales:

**Jerarquía de Personas**

Persona (abstract @MappedSuperclass)

├── Medico

│ ├── Matrícula profesional (Value Object)

│ ├── Especialidad médica

│ └── Colección de citas asignadas

└── Paciente

├── Historia clínica (OneToOne)

├── Datos de contacto

└── Colección de citas programadas

**Estructura Organizacional**

Hospital (Aggregate Root)

├── Departamentos médicos

│ ├── Médicos especializados

│ └── Salas por especialidad

└── Pacientes registrados

**Entidades Clave**

1. **Persona** (Clase Abstracta)
   * Superclase para Médico y Paciente
   * Atributos comunes: nombre, apellido, DNI, fecha de nacimiento, tipo de sangre
   * Validación de DNI argentino (7-8 dígitos)
   * Cálculo automático de edad
2. **Medico** (extends Persona)
   * Matrícula profesional embebida (formato: MP-XXXXX)
   * Especialidad médica obligatoria
   * Relación ManyToOne con Departamento
   * Relación OneToMany con Cita (gestión de agenda)
3. **Paciente** (extends Persona)
   * Historia clínica única auto-generada (OneToOne)
   * Datos de contacto (teléfono, dirección)
   * Relación ManyToOne con Hospital
   * Relación OneToMany con Cita
4. **Hospital** (Aggregate Root)
   * Administra departamentos médicos
   * Registra pacientes
   * Mantiene consistencia del agregado completo
   * Operaciones en cascada (CascadeType.ALL)
5. **Departamento**
   * Organización por especialidad médica
   * Agrupa médicos de la misma especialidad
   * Administra salas especializadas
   * Factory method para creación de salas
6. **Sala**
   * Identificación única por número
   * Clasificación por tipo (consultorio, quirófano, emergencias)
   * Pertenece a un departamento específico
   * Gestiona citas programadas
7. **Cita**
   * Vincula paciente, médico y sala
   * Fecha/hora con validación de futuro
   * Estado (PROGRAMADA, COMPLETADA, CANCELADA)
   * Costo de consulta (BigDecimal para precisión)
   * Observaciones médicas
   * Serialización CSV para persistencia alternativa
8. **HistoriaClinica**
   * Relación OneToOne única con Paciente
   * Número de historia auto-generado (HC-DNI-timestamp)
   * Colecciones @ElementCollection:
     + Diagnósticos médicos
     + Tratamientos prescritos
     + Alergias conocidas
   * Fecha de creación para auditoría

**Patrones de Diseño Implementados**

**1. Aggregate Root (DDD)**

**Implementación**: Hospital

* Actúa como punto de entrada para operaciones sobre departamentos y pacientes
* Mantiene invariantes del agregado completo
* Controla ciclo de vida de entidades dependientes

**2. Value Object (DDD)**

**Implementación**: Matricula

* Objeto embebido (@Embedded) sin identidad propia
* Validación de formato en constructor (MP-XXXXX)
* Inmutable y reutilizable

**3. SuperBuilder Pattern**

**Implementación**: Jerarquía Persona

* Permite herencia de builders con Lombok @SuperBuilder
* Construcción fluida para clases con herencia
* Validaciones en constructor personalizado

**4. Service Layer Pattern**

**Implementación**: CitaService / CitaManager

* Separa lógica de negocio de entidades
* Encapsula reglas de validación complejas
* Gestiona índices para consultas eficientes

**5. Factory Method**

**Implementación**: Departamento.crearSala()

* Encapsula creación de salas con relación bidireccional automática
* Garantiza consistencia al crear entidades relacionadas

**6. Builder Manual**

**Implementación**: Hospital, Departamento, Sala, HistoriaClinica

* Validación personalizada en constructor
* Inicialización explícita de colecciones
* Mayor control sobre construcción de objetos

**Gestión de Relaciones Bidireccionales**

El sistema implementa un patrón consistente para todas las relaciones bidireccionales:

// Métodos públicos mantienen consistencia

public void agregarDepartamento(Departamento dept) {

departamentos.add(dept);

dept.setHospital(this);

}

// Getters públicos retornan colecciones inmutables

public List<Departamento> getDepartamentos() {

return Collections.unmodifiableList(departamentos);

}

// Getters package-private permiten sincronización interna

List<Departamento> getInternalDepartamentos() {

return departamentos;

}

**🔧 Tecnologías Utilizadas**

**Core Framework**

* **Java**: Lenguaje de programación principal
* **Jakarta Persistence API 3.1.0**: Estándar de persistencia Java EE
* **Hibernate ORM 6.4.4**: Implementación JPA de referencia
* **H2 Database 2.2.224**: Base de datos embebida file-based

**Utilidades**

* **Lombok 1.18.38**: Reducción de boilerplate code
  + @Getter, @Setter, @ToString
  + @SuperBuilder para herencia de builders
  + @NoArgsConstructor para constructores JPA
  + @RequiredArgsConstructor para enums

**Build Tool**

* **Gradle 8.x**: Sistema de construcción y gestión de dependencias

**📊 Modelo de Base de Datos**

**Tablas Principales**

**Personas**

* **medicos**: Médicos con especialidad y matrícula
* **pacientes**: Pacientes con datos de contacto

**Organización**

* **hospitales**: Hospitales del sistema
* **departamentos**: Departamentos por especialidad
* **salas**: Salas médicas por departamento

**Atención Médica**

* **citas**: Citas médicas programadas
* **historias\_clinicas**: Historias clínicas de pacientes

**Colecciones (@ElementCollection)**

* **diagnosticos**: Diagnósticos por historia clínica
* **tratamientos**: Tratamientos por historia clínica
* **alergias**: Alergias por historia clínica

**Estrategias de Persistencia**

1. **Generación de IDs**: IDENTITY para todas las entidades
2. **Cascading**: CascadeType.ALL en relaciones OneToMany padre-hijo
3. **Orphan Removal**: true para eliminar entidades huérfanas
4. **Fetch Type**: LAZY para optimizar consultas
5. **Schema Management**: hibernate.hbm2ddl.auto=update

**🚀 Comenzando**

**Prerrequisitos**

* Java JDK 17 o superior
* Gradle 8.x (incluido wrapper)

**Instalación**

1. **Clonar el repositorio**

[](http://localhost:63342/markdownPreview/1249679436/markdown-preview-index-sgtj5af44ml6old2qmgs6om7km.html)git clone <repository-url>

cd JpaHospital

1. **Compilar el proyecto**

[](http://localhost:63342/markdownPreview/1249679436/markdown-preview-index-sgtj5af44ml6old2qmgs6om7km.html)./gradlew build

1. **Ejecutar la aplicación**

[](http://localhost:63342/markdownPreview/1249679436/markdown-preview-index-sgtj5af44ml6old2qmgs6om7km.html)./gradlew run

**Comandos Disponibles**

[](http://localhost:63342/markdownPreview/1249679436/markdown-preview-index-sgtj5af44ml6old2qmgs6om7km.html)# Compilar el proyecto

./gradlew build

# Ejecutar aplicación principal

./gradlew run

# Ejecutar tests

./gradlew test

[](http://localhost:63342/markdownPreview/1249679436/markdown-preview-index-sgtj5af44ml6old2qmgs6om7km.html)

# Limpiar build artifacts

./gradlew clean

# Compilación completa limpia

./gradlew clean build

**💼 Reglas de Negocio**

**Validaciones de Citas Médicas**

El sistema CitaManager aplica las siguientes reglas al programar citas:

1. **Validación Temporal**
   * ❌ No se permiten citas en fechas pasadas
   * ✅ Solo citas futuras son válidas
2. **Validación Económica**
   * ❌ Costo debe ser mayor a cero
   * ✅ Se utiliza BigDecimal para precisión monetaria
3. **Validación de Especialidad**
   * ❌ La especialidad del médico debe coincidir con el departamento de la sala
   * ✅ Garantiza coherencia en atención especializada
4. **Disponibilidad de Médico**
   * ❌ Buffer de 2 horas entre citas del mismo médico
   * ✅ Evita sobrecarga y garantiza tiempo de atención
5. **Disponibilidad de Sala**
   * ❌ Buffer de 2 horas entre citas en la misma sala
   * ✅ Permite limpieza y preparación entre pacientes

**Validaciones de Datos**

**DNI**

* Formato: 7-8 dígitos numéricos
* Validación: \d{7,8}
* Ejemplos válidos: "12345678", "1234567"

**Matrícula Profesional**

* Formato: "MP-" seguido de 4-6 dígitos
* Validación en constructor de Matricula
* Ejemplo: "MP-12345"

**Historia Clínica**

* Número auto-generado: HC-{DNI}-{timestamp}
* Constraint UNIQUE en paciente\_id
* Creación automática en constructor de Paciente

**📝 Uso del Sistema**

**Ejemplo: Crear y Persistir Datos del Hospital**

// Crear EntityManager

EntityManagerFactory emf = Persistence.createEntityManagerFactory("hospital-persistence-unit");

EntityManager em = emf.createEntityManager();

em.getTransaction().begin();

// 1. Crear hospital

Hospital hospital = Hospital.builder()

.nombre("Hospital Central")

.direccion("Av. Libertador 1234")

.telefono("011-4567-8901")

.build();

// 2. Crear departamento

Departamento cardiologia = Departamento.builder()

.nombre("Cardiología")

.especialidad(EspecialidadMedica.CARDIOLOGIA)

.build();

hospital.agregarDepartamento(cardiologia);

// 3. Crear sala

Sala consultorio = Sala.builder()

.numero("CARD-101")

.tipo("Consultorio")

.departamento(cardiologia)

.build();

// 4. Crear médico

Medico cardiologo = Medico.builder()

.nombre("Carlos")

.apellido("González")

.dni("12345678")

.fechaNacimiento(LocalDate.of(1975, 5, 15))

.tipoSangre(TipoSangre.A\_POSITIVO)

.numeroMatricula("MP-12345")

.especialidad(EspecialidadMedica.CARDIOLOGIA)

.build();

cardiologia.agregarMedico(cardiologo);

// 5. Crear paciente (historia clínica se crea automáticamente)

Paciente paciente = Paciente.builder()

.nombre("María")

.apellido("López")

.dni("11111111")

.fechaNacimiento(LocalDate.of(1985, 12, 5))

.tipoSangre(TipoSangre.A\_POSITIVO)

.telefono("011-1111-1111")

.direccion("Calle Falsa 123")

.build();

hospital.agregarPaciente(paciente);

// 6. Agregar información a historia clínica

HistoriaClinica historia = paciente.getHistoriaClinica();

historia.agregarDiagnostico("Hipertensión arterial");

historia.agregarTratamiento("Enalapril 10mg");

historia.agregarAlergia("Penicilina");

// 7. Programar cita usando CitaManager

CitaManager citaManager = new CitaManager();

try {

Cita cita = citaManager.programarCita(

paciente,

cardiologo,

consultorio,

LocalDateTime.now().plusDays(1).withHour(10).withMinute(0),

new BigDecimal("150000.00")

);

System.out.println("Cita programada: " + cita.getId());

} catch (CitaException e) {

System.err.println("Error: " + e.getMessage());

}

// 8. Persistir todo (cascade hace el resto)

em.persist(hospital);

em.persist(consultorio);

em.persist(cardiologo);

em.persist(historia);

em.getTransaction().commit();

**Ejemplo: Consultas JPQL**

// Consultar médicos por especialidad

TypedQuery<Medico> query = em.createQuery(

"SELECT m FROM Medico m WHERE m.especialidad = :esp",

Medico.class

);

query.setParameter("esp", EspecialidadMedica.CARDIOLOGIA);

List<Medico> cardiologos = query.getResultList();

// Contar citas por estado

Long citasCompletadas = em.createQuery(

"SELECT COUNT(c) FROM Cita c WHERE c.estado = :estado",

Long.class

)

.setParameter("estado", EstadoCita.COMPLETADA)

.getSingleResult();

// Obtener pacientes con alergias

TypedQuery<Paciente> queryAlergicos = em.createQuery(

"SELECT DISTINCT p FROM Paciente p " +

"JOIN p.historiaClinica h " +

"WHERE SIZE(h.alergias) > 0",

Paciente.class

);

**⚠️ Consideraciones Importantes**

**Lombok + JPA + SuperBuilder**

**Problema conocido**: @Builder.Default no funciona correctamente con @SuperBuilder

// ❌ INCORRECTO: La lista será null

@SuperBuilder

public class Medico extends Persona {

@OneToMany(...)

@Builder.Default

private List<Cita> citas = new ArrayList<>();

}

// ✅ CORRECTO: Inicializar explícitamente en constructor

protected Medico(MedicoBuilder<?, ?> builder) {

super(builder);

this.citas = new ArrayList<>(); // Obligatorio

}

**Historia Clínica Única**

**Regla crítica**: Cada paciente tiene exactamente UNA historia clínica

// ✅ CORRECTO: Usar la auto-generada

Paciente paciente = Paciente.builder()...build();

HistoriaClinica historia = paciente.getHistoriaClinica();

historia.agregarDiagnostico("Diabetes tipo 2");

// ❌ INCORRECTO: Violación UNIQUE constraint

HistoriaClinica nueva = HistoriaClinica.builder()

.paciente(paciente) // ERROR: paciente ya tiene historia

.build();

**Relaciones Bidireccionales**

**Siempre usar métodos helper para mantener consistencia**:

// ✅ CORRECTO

hospital.agregarDepartamento(departamento);

departamento.agregarMedico(medico);

// ❌ INCORRECTO

departamento.setHospital(hospital); // Solo actualiza un lado

**📂 Estructura del Proyecto**

JpaHospital/

├── src/

│ ├── main/

│ │ ├── java/org/example/

│ │ │ ├── entidades/ # Entidades JPA

│ │ │ │ ├── Persona.java

│ │ │ │ ├── Medico.java

│ │ │ │ ├── Paciente.java

│ │ │ │ ├── Hospital.java

│ │ │ │ ├── Departamento.java

│ │ │ │ ├── Sala.java

│ │ │ │ ├── Cita.java

│ │ │ │ ├── HistoriaClinica.java

│ │ │ │ ├── Matricula.java

│ │ │ │ ├── EspecialidadMedica.java

│ │ │ │ ├── EstadoCita.java

│ │ │ │ └── TipoSangre.java

│ │ │ ├── servicio/ # Capa de servicio

│ │ │ │ ├── CitaService.java

│ │ │ │ ├── CitaManager.java

│ │ │ │ └── CitaException.java

│ │ │ └── Main.java # Aplicación principal

│ │ └── resources/

│ │ └── META-INF/

│ │ └── persistence.xml # Configuración JPA

│ └── test/

│ └── java/ # Tests unitarios

├── data/ # Base de datos H2

├── build.gradle # Configuración Gradle

├── CLAUDE.md # Guía para Claude Code

└── README.md # Este archivo

**🎯 Funcionalidades Principales**

**✅ Gestión de Pacientes**

* Registro con datos personales y contacto
* Historia clínica automática e inmutable
* Registro de diagnósticos, tratamientos y alergias
* Consulta de citas programadas

**✅ Gestión de Médicos**

* Registro con matrícula profesional validada
* Asignación a departamentos especializados
* Gestión de agenda con validación de disponibilidad
* Consulta de citas asignadas

**✅ Organización Hospitalaria**

* Gestión de hospitales y departamentos
* Creación de salas por especialidad
* Agrupación de médicos por especialidad
* Mantenimiento de estructura organizacional

**✅ Sistema de Citas**

* Programación con validaciones exhaustivas
* Verificación de disponibilidad (médico y sala)
* Control de especialidades compatibles
* Estados de cita (programada, completada, cancelada)
* Persistencia CSV alternativa

**✅ Consultas y Reportes**

* Estadísticas por especialidad
* Citas por paciente/médico/sala
* Reportes de ocupación
* Conteo de citas por estado

**🔍 Enumeraciones del Sistema**

**EspecialidadMedica**

CARDIOLOGIA, NEUROLOGIA, PEDIATRIA, TRAUMATOLOGIA, GINECOLOGIA, UROLOGIA, OFTALMOLOGIA, DERMATOLOGIA, PSIQUIATRIA, MEDICINA\_GENERAL, CIRUGIA\_GENERAL, ANESTESIOLOGIA

**EstadoCita**

PROGRAMADA, COMPLETADA, CANCELADA

**TipoSangre**

A\_POSITIVO, A\_NEGATIVO, B\_POSITIVO, B\_NEGATIVO, AB\_POSITIVO, AB\_NEGATIVO, O\_POSITIVO, O\_NEGATIVO

**📚 Configuración de Persistencia**

**Archivo**: src/main/resources/META-INF/persistence.xml

<persistence-unit name="hospital-persistence-unit">

<properties>

<!-- Base de datos H2 file-based -->

<property name="jakarta.persistence.jdbc.url"

value="jdbc:h2:file:./data/hospidb"/>

<!-- Auto-actualización de schema -->

<property name="hibernate.hbm2ddl.auto" value="update"/>

<!-- Logging SQL -->

<property name="hibernate.show\_sql" value="true"/>

<property name="hibernate.format\_sql" value="true"/>

</properties>

</persistence-unit>

**👥 Equipo de Desarrollo**

**Los Cortez Development Team**

**📄 Licencia**

Este proyecto es de código abierto y está disponible para fines educativos y de demostración.

**🔗 Referencias**

* [Jakarta Persistence (JPA) 3.1](https://jakarta.ee/specifications/persistence/3.1/)
* [Hibernate ORM Documentation](https://hibernate.org/orm/documentation/)
* [H2 Database](https://www.h2database.com/)
* [Project Lombok](https://projectlombok.org/)
* [Domain-Driven Design](https://martinfowler.com/bliki/DomainDrivenDesign.html)

**Versión**: 1.0 **Última actualización**: Octubre 2025