Trabajo grupal
CLÍNICA
Segunda Parte

Facultad de Ingeneria
de la Universidad Nacional
de Mar del Plata

Profesores:

- Gellon, Ivonne
- Guccione, Leonel Domingo
- Lazzurri, Carlos Guillermo

Integrantes:

- Fernandez Ordoqui, Julian Agustin
- Stufano, Nazareno
- Ganduglia, Ciro Exequiel
- Teruel López, Leonel
- Pezzi, Esteban

1- Introducción

Este informe amplía el sistema de gestión de la clínica desarrollado previamente, incorporando: gestión de asociados, simulación de uso de ambulancia con concurrencia, persistencia de datos mediante JDBC y una interfaz gráfica Swing organizada en MVC. Se aplican nuevos patrones (State, Observer, Singleton) manteniendo los previos (Facade, Factory, Decorator, Double Dispatch). El alcance excluye la persistencia de la simulación y la facturación para asociados; los asociados no son pacientes ni intervienen en honorarios o facturas.

2- Arquitectura del sistema

Estructura del proyecto

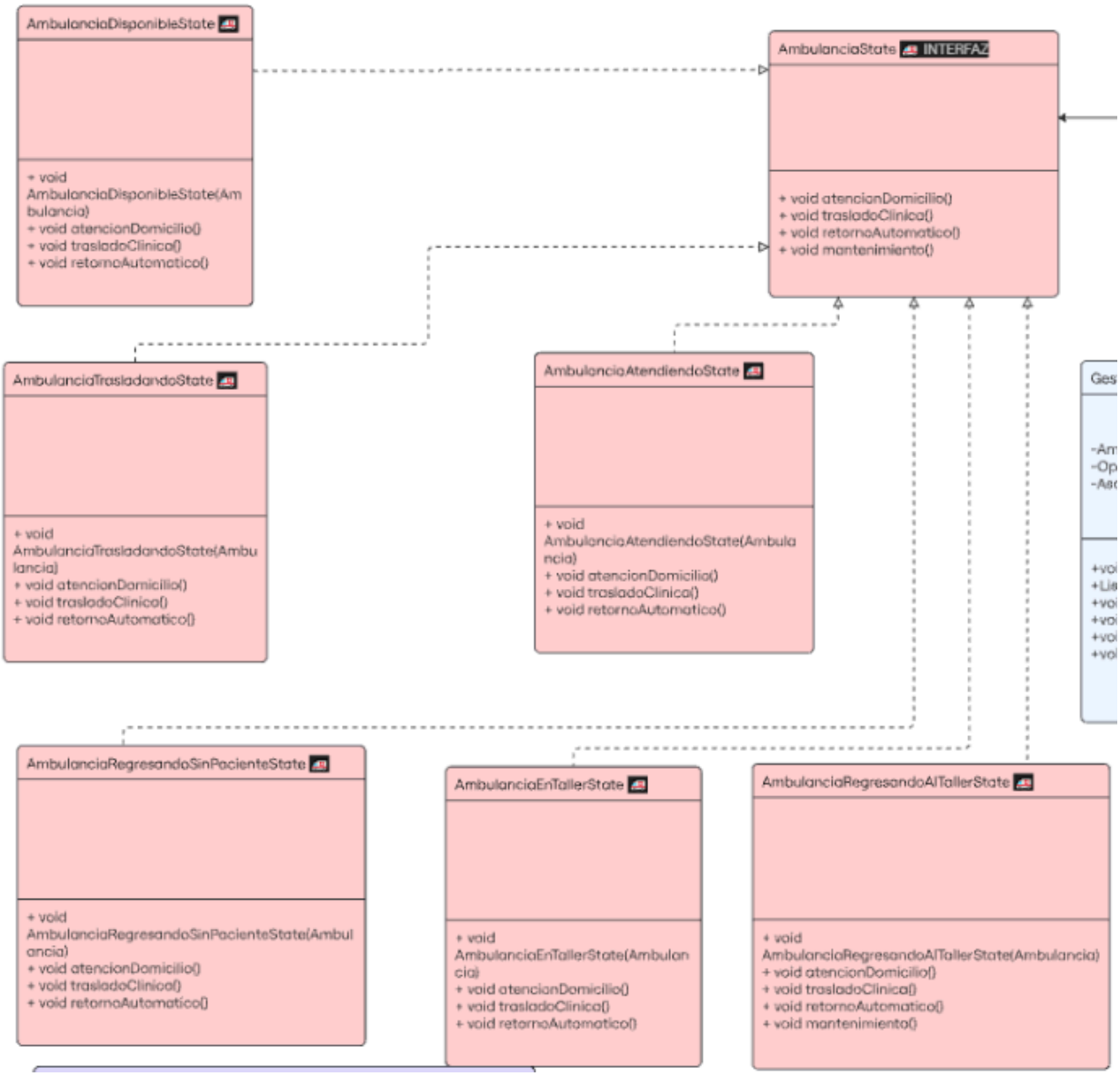
El proyecto sigue una arquitectura organizada por los siguientes paquetes:

- **modelo/Ambulancia**: Contexto del patrón State. Gestiona el estado actual y las transiciones según solicitudes concurrentes.
- **modelo/Asociado**: Persona registrada para solicitar el uso de la ambulancia. No genera facturación médica. Igualdad por DNI.
- **modelo/GestionLlamados:** Coordinador de solicitudes concurrentes. Encola pedidos de asociados y operario, y los despacha según disponibilidad.
- **modelo/Operario**: Empleado que solicita mantenimiento de la ambulancia. Puede ser disparado desde la interfaz gráfica.
- controlador**/ControladorAsociados**: Coordina la gestión de asociados: alta, baja, listado y persistencia. Traduce excepciones en mensajes visuales para la interfaz.
- controlador**/ControladorSimulacion**: Gestiona la simulación concurrente: arranque de hilos, solicitud de mantenimiento, finalización ordenada. Comunica cambios al modelo y actualiza la vista.
- vista**/IVista**: Interfaz que define los métodos comunes para las ventanas Swing. Facilita el uso de Observer y desacopla la lógica del controlador.
- vista/VentanaPestanas: Ventana principal con pestañas para navegar entre módulos (gestión de asociados, simulación, configuración). Implementa IVista.
- persistencia/AsociadoDAO: Encapsula el acceso a la base de datos para operaciones CRUD sobre asociados. Usa JDBC.
- persistencia/AsociadoDTO: Objeto de transferencia que representa los datos de un asociado para persistencia.
- persistencia/PersistenciaAsociado: Clase de servicio que coordina la carga y guardado de asociados desde/hacia la base.
- persistencia/AsociadoExistenteException: Excepción lanzada al intentar insertar un asociado duplicado en la base

3- Patrones de diseño

3.1 Patrón State

Objetivo: Modelar el comportamiento dinámico de la ambulancia según su estado actual, permitiendo que las transiciones entre estados se realicen de forma clara, extensible y sin condicionales anidados.

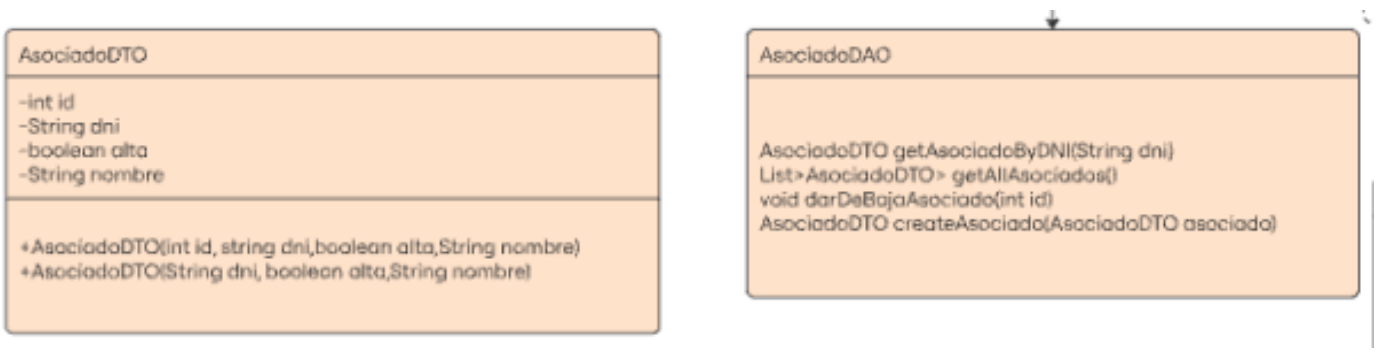


Ventajas:

- Modela claramente los distintos estados de la ambulancia.
- Evita condicionales complejos, delegando la lógica en cada estado.
- Facilita la simulación concurrente y las transiciones controladas.
- Mejora la mantenibilidad y extensibilidad del sistema.
- Se integra con Observer para reflejar cambios en tiempo real en la interfaz.

3.2 Patrón DAO y DTO

Objetivo: Separar la lógica de acceso a datos del modelo de negocio, permitiendo persistencia desacoplada y organizada. El patrón DAO encapsula las operaciones sobre la base de datos, mientras que el patrón DTO representa los datos transferidos entre capas.

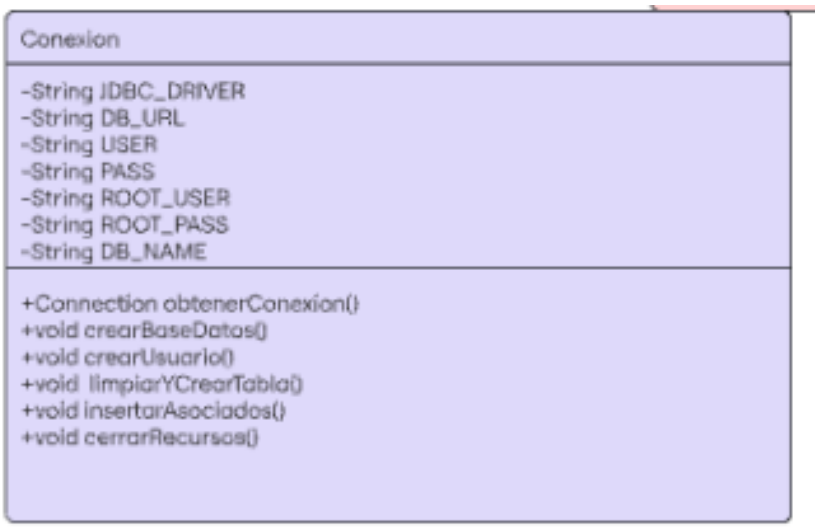


Ventajas:

- Permite persistir los datos sin acoplar la lógica de negocio a JDBC. · Facilita el mantenimiento y la evolución del sistema.
- El DTO actúa como contenedor de datos plano, sin lógica, ideal para transporte entre capas.
- El DAO centraliza las operaciones de base de datos, evitando duplicación de código.
- La clase de servicio (PersistenciaAsociado) coordina la persistencia sin exponer detalles técnicos al resto del sistema.

3.3 Patrón Singleton

Objetivo: Garantizar que exista una única instancia de conexión a la base de datos durante toda la ejecución del sistema, evitando duplicación de recursos y facilitando el acceso centralizado.



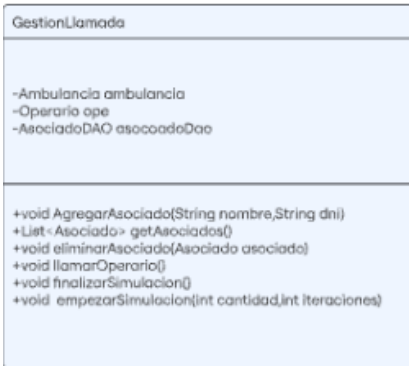
Ventajas:

- **Asegura una única instancia de conexión a la base de datos.**
- **Evita la creación repetida de conexiones JDBC.**
- **Facilita el acceso centralizado desde cualquier módulo del sistema.**
- **Mejora el rendimiento y la organización del código.**
- **Compatible con el patrón DAO para persistencia desacoplada.**

3.4 Patrón Facade

Objetivo: Simplifica la interacción con el sistema mediante una interfaz unificada.

Implementación:



Ventajas:

- **Reúne en una sola clase las operaciones de simulación y gestión de asociados.**
- **Oculta la complejidad de la ambulancia, el operario y la persistencia.**
- **Facilita el uso desde la interfaz gráfica con métodos simples y claros.**
- **Reduce el acoplamiento entre controladores y lógica interna.**
- **Permite extender funcionalidades sin modificar otras capas..**

3.4 Patrón Observer-Observable

Objetivo: Permitir que las vistas Swing se actualicen automáticamente cuando cambia el estado del modelo, sin necesidad de sondear ni acoplar directamente la interfaz a la lógica interna. Se utiliza en paralelo con el patrón State para reflejar en tiempo real los cambios de estado de la ambulancia durante la simulación.

Ventajas:

- **Permite que las vistas se actualicen automáticamente ante cambios en el modelo.**
- **Desacopla la lógica de simulación de la interfaz gráfica.**
- **Facilita la visualización en tiempo real del estado de la ambulancia y los eventos del sistema.**
- **Compatible con múltiples vistas observadoras.**
- **Mejora la organización y escalabilidad del sistema.**

4. Modelo de negocio

4.1 Jerarquía de clases principales

Ambulancia

↑ EstadoAmbulancia (abstract)

├── EstadoDisponible

├── EstadoTrasladandoPaciente

├── EstadoAtendiendoDomicilio

├── EstadoRegresandoSinPaciente

├── EstadoEnTaller

└── EstadoRegresandoDelTaller

Ambulancia: clase principal que gestiona el estado actual y delega el comportamiento según el patrón State.

EstadoAmbulancia: clase abstracta que define las operaciones posibles en cada estado. Cada subclase implementa la lógica específica para atender, trasladar, regresar o entrar en mantenimiento.

5. Funcionalidades principales

5.1 Gestión de Asociados

Flujo de operaciones:

1. **Alta: altaAsociado() – Verifica duplicados por DNI y registra el asociado en el sistema.**
2. **Baja: bajaAsociado() – Elimina el asociado por DNI, con manejo de excepción si no existe.**
3. **Listado: listarAsociados() – Muestra todos los asociados registrados.**
4. **Persistencia: guardarAsociados() / cargarAsociados() – Guarda y recupera los datos desde la base de datos.**
5. **Visualización: muestra mensajes descriptivos en la interfaz ante errores o acciones exitosas.**

Durante la simulación, las operaciones de alta y baja quedan deshabilitadas.

5.2 Simulación de Ambulancia

La simulación permite:

- **Configurar cantidad de asociados y número de solicitudes por asociado.**
- **Ejecutar hilos concurrentes que representan solicitudes simultáneas. · Visualizar en tiempo real el estado de la ambulancia, la evolución de cada asociado y las acciones del operario.**
- **Finalizar manualmente mediante botón "Finalizar" o automáticamente al completar todas las solicitudes. · Usar wait/notifyAll para coordinar el acceso a la ambulancia.**

Tipos de solicitud:

- **Atención a domicilio.**
- **Traslado a la clínica. · Solicitud de mantenimiento (operario).**
- **Retorno automático a la clínica (evento temporal).**

5.3 Comportamiento de Ambulancia

La clase Ambulancia gestiona:

- **Estado actual mediante el patrón State.**
- **Transiciones según tipo de solicitud y estado vigente.**
- **Sincronización de acceso con métodos synchronized y wait/notifyAll.**
- **Notificación de cambios a la interfaz mediante Observer.**

Estados posibles:

1. **Disponible**
2. **Trasladando paciente**
3. **Atendiendo paciente en domicilio**
4. **Regresando sin paciente**
5. **En taller**
6. **Regresando del taller**

5.4 Interfaz gráfica

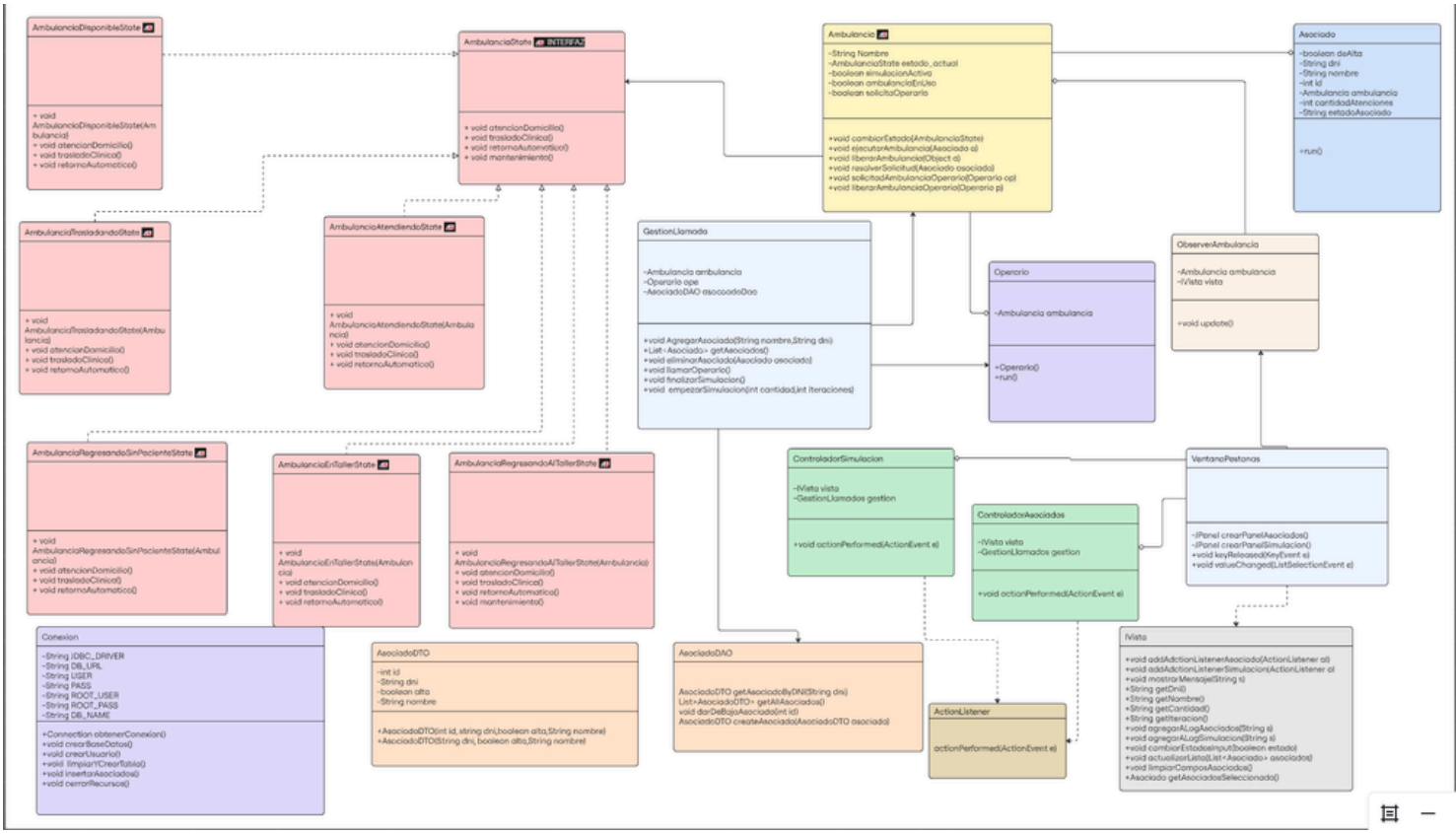
La interfaz Swing permite:

- **Navegar entre módulos mediante pestañas. · Gestionar asociados con formularios y listados.**
- **Configurar y ejecutar la simulación. · Visualizar el estado de la ambulancia y el log de eventos.**
- **Mostrar errores y excepciones mediante mensajes descriptivos.**

Organizada bajo el patrón MVC:

- **Modelo: lógica de negocio (asociados, ambulancia, estados).**
- **Vista: ventanas Swing (JframePrincipal, JframeSimulacion, etc.).**
- **Controlador: coordina eventos y comunica la interfaz con el modelo.**

7. Diagrama de clases UML



Puede verlo también en este link

8. Conclusiones

El sistema desarrollado en esta segunda entrega amplía las capacidades creadas en la fase anterior, integrando nuevos módulos y patrones que enriquecen la arquitectura general. Se han incorporado:

- Gestión de asociados: alta, baja y listado, con verificación por DNI y persistencia en la base de datos.
- Simulación concurrente del uso de la ambulancia: atención a domicilio, traslado, mantenimiento y retorno automático, con visualización en tiempo real.
- Persistencia de datos: almacenamiento y recuperación de asociados mediante JDBC, utilizando conexión Singleton.
- Interfaz gráfica: ventanas Swing organizadas bajo el patrón MVC, con actualización automática mediante Observer.

Los patrones de diseño aplicados permiten:
Modelar el comportamiento dinámico de la ambulancia a través del patrón State.

- Actualizar las vistas en tiempo real sin acoplamiento directo (Observer).
- Coordinar eventos de interfaz y lógica de negocio de manera clara (MVC).
- Mantener una única instancia de conexión a la base de datos (Singleton) patrones previos: Facade, Factory, Decorator y Double Dispatch.