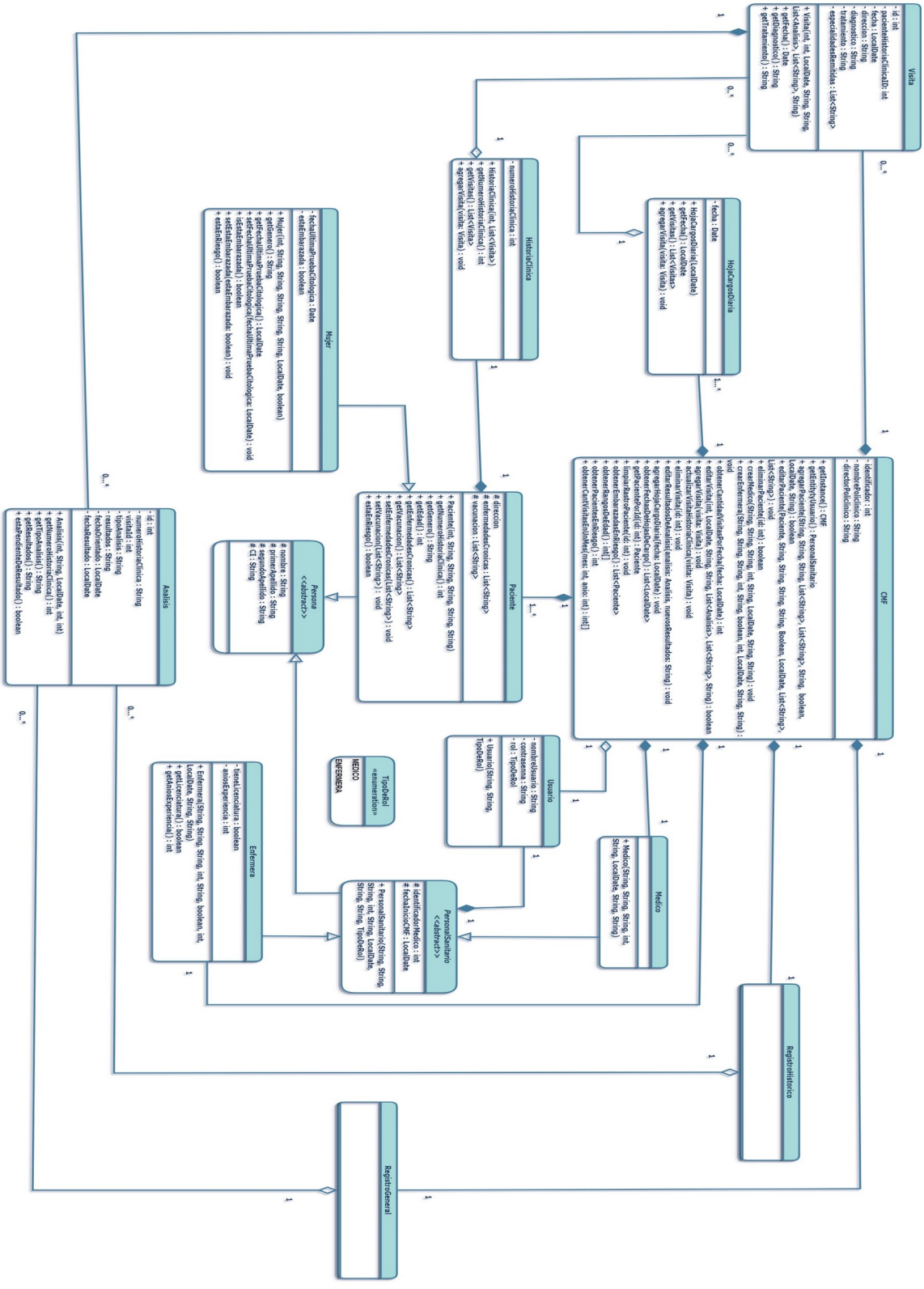
Tarea Final DPOO

Grupo **13** - Tarea **6**

David Alfonso Hernandez
Reynaldo Daniel Quesada García



(En la carpeta del proyecto se adjunta un archivo llamado UML.jpg con el diagrama de clases mucho más visible)

Expliación de Clases, Atributos y Relaciones del Diagrama UML

Explicación de uso y propósito de cada relación en el diagrama

En los siguiente párrafos se explicará el diseño elegido por los integrantes del equipo, argumentando la mayor optimización del mismo. Para ello se seleccionará clase por clase, y se explicarán sus multiples relaciones con el resto de clases del programa.

CMF

Representa la clase controladora del programa.

Relaciones:

*** HojaCargosDiaria**

- Composición
- Cardinalidad: 1 - 1 a muchos

Esta relación expresa que la clase CMF es la encargada de crear a HojaDeCargosDiaria, además de realizar todas las operaciones con ella. La cardinalidad dice que para un CMF pueden existir de 1 a muchos HojaCargoDiaria, imprescindible por el uso que se le dará en el problema.

*** Paciente**

- Composición
- Cardinalidad: 1 - 1 a muchos

CMF es la encargada de crear al paciente dentro del sistema, el paciente no puede existir en el sistema si no existiera CMF. La caridanlidad refleja que para un CMF pueden existir de 1 a muchos Pacientes, lo cual es evidente en la problemática.

*** Médico**

- Composición
- Cardinalidad: 1 – 1

CMF es la encargada de crear al médico dentro del sistema, el médico no puede existir en el sistema si no existiera la clase CMF. La cardinalidad refleja que para un CMF existe un médico, ni mas ni menos, ajustándonos a lo planteado en la problemática.

*** Enfermera**

- Composición
- Cardinalidad: 1 – 1

CMF es la encargada de crear a la enfermera dentro del sistema, la enfermera no puede existir en el sistema si no existiera la clase CMF. La cardinalidad refleja que para un CMF existe una enfermera, ni mas ni menos, ajustándonos a lo planteado en la problemática.

*** RegistroHistorico y RegistroGeneral**

- Composición

- Cardinalidad: 1 – 1

Estas dos clases solo se mencionan en el problema pero no se brinda más información al respecto y se menciona que son propios del CMF

*** Usuario**

- Agregación

- Cardinalidad: 1 – 1

CMF contiene una referencia del Usuario actual que esta usando el sistema

Persona

Es una clase abstracta que tiene 2 hijos (Paciente, y la clase abstracta PersonalSanitario) en nuestra problemática. Esta relación la argumentamos bajo el principio de la Herencia y siguiendo las buenas prácticas de Programación Orientada a Objetos.

Personal Sanitario

Es una clase abstracta que tiene 3 relaciones (médico, usuario y enfermera) en nuestra problemática. Esta relación la argumentamos bajo el principio de la Herencia y siguiendo las buenas prácticas de Programación Orientada a Objetos.

HojaCargosDiaria

*** Visita**

- Agregación

- Cardinalidad: 0 a muchos

La hoja de cargos contiene una agregación de de Visitas, necesaria para controlar las visitas del dia

RegistroHistorico y RegistroGeneral

*** Analisis**

- Agregación

- Cardinalidad: 1 – 1 a muchos

Se menciona que a estas clases se le agregan los resultados de los análisis.

Paciente

*** Mujer**

- Herencia

Se plantea que una entidad Mujer, es un Paciente, con algunos atributos extra. Por lo que es idóneo usar una relación de Herencia, en la que la Mujer herede de la clase Paciente.

*** HistoriaClinica**

- Composicion

- Cardinalidad: 1 - 1

Cada paciente tiene su Historia Clinica

Visita

*** Analisis**

- Composición

- Cardinalidad: 1 - 1 a muchos

Cada visita requiere una historia clínica donde se utilicen los datos de la visita

Explicacion de Metodos

Entidad: CMF

- Métodos:
  - getInstanciaCMF(): Obtiene la instancia única del consultorio médico.
  - crearPaciente(): Crea un nuevo paciente con datos específicos.
  - actualizarVisita(): Actualiza los detalles de una visita médica.
  - eliminarVisita(): Elimina una visita médica existente.
  - obtenerCantidadVisitasPorFecha(): Devuelve la cantidad de visitas realizadas en una fecha específica.
  - obtenerPacientesEnRiesgo(): Identifica pacientes en riesgo según criterios definidos.
  - obtenerRangosEdad(): Calcula la distribución de pacientes por rangos de edad.
  - obtenerCantVisitasEnRango(): Devuelve la cantidad de visitas realizadas en un rango de fechas.

Entidad: HistoriaClinica

- Métodos:
  - getNumeroHistoriaClinica(): Obtiene el número único de la historia clínica.
  - getVisitas(): Devuelve la lista de visitas asociadas a la historia clínica.
  - agregarVisita(): Añade una nueva visita a la historia clínica.

Entidad: Paciente

- Métodos:
  - getGenero(): Devuelve el género del paciente basado en su CI.
  - getEnfermedadesCronicas(): Obtiene la lista de enfermedades crónicas del paciente.
  - getVacunacion(): Devuelve la lista de vacunas aplicadas al paciente.
  - estaEnRiesgo(): Determina si el paciente está en riesgo según sus condiciones.

Entidad: Mujer (Extiende Paciente)

- Métodos:
  - getFechaUltimaPruebaCitologica(): Devuelve la fecha de la última prueba citológica.
  - setFechaUltimaPruebaCitologica(): Establece la fecha de la última prueba citológica.
  - setEmbarazada(): Define si la paciente está embarazada.
  - estaEnRiesgo(): Evalúa si la paciente está en riesgo considerando embarazo y pruebas citológicas.

Entidad: Enfermera (Extiende PersonalSanitario)

- Métodos:
  - getLicenciatura(): Indica si la enfermera tiene licenciatura.
  - getAniosExperiencia(): Devuelve los años de experiencia de la enfermera.

Entidad: Visita

- Métodos:
  - getFecha(): Devuelve la fecha de la visita.
  - getDiagnostico(): Obtiene el diagnóstico realizado en la visita.
  - getTratamiento(): Devuelve el tratamiento aplicado.

Entidad: HojaCargosDiaria

- Métodos:
  - getFecha(): Devuelve la fecha de la hoja de cargos.
  - setFecha(): Valida y establece la fecha de la hoja de cargos.
  - getVisitas(): Devuelve la lista de visitas registradas en la hoja.
  - agregarVisita(): Añade una visita a la hoja de cargos.

Entidad: Usuario

- Métodos:
  - getUserName(): Devuelve el nombre de usuario.
  - getRole(): Obtiene el rol asignado al usuario.

Entidad: Análisis

- Métodos:
  - getNombreHistoriaClinica(): Obtiene el nombre asociado a la historia clínica del análisis.
  - getVitalidad(): Devuelve la vitalidad registrada en el análisis.
  - getResultados(): Obtiene los resultados del análisis.
  - estaPendienteDeResultado(): Determina si el análisis aún está pendiente de resultados.

Explicación de la División de Capas y Paquetes para la Solución

El proyecto está organizado en una estructura de paquetes que sigue un enfoque de arquitectura en capas, lo que facilita la separación de responsabilidades y el mantenimiento del código. A continuación, se describe la división de capas y paquetes:

1. Capa de Entidades (entidades)
Este paquete contiene las clases que representan los datos y la lógica de negocio del dominio. Está subdividido en varios subpaquetes:

- entidades.personal: Contiene las clases relacionadas con las personas que interactúan con el sistema, como:
  - Persona: Clase base abstracta para representar a cualquier persona.
  - Paciente: Extiende Persona y representa a los pacientes.
  - Mujer: Subclase de Paciente que incluye atributos específicos como embarazo.
  - Medico y Enfermera: Representan roles específicos del personal sanitario.
- entidades.registros: Maneja las clases relacionadas con los registros médicos y administrativos, como:
  - HistoriaClinica: Representa el historial médico de un paciente.
  - Visita: Detalla las visitas médicas realizadas.
  - HojaCargosDiaria: Registra las visitas diarias.
  - RegistroGeneral y RegistroHistorico: Clases para registros generales e históricos.
- entidades: Incluye la clase principal CMF, que actúa como el núcleo del sistema, gestionando pacientes, visitas y otros datos.

2. Capa de Servicios (service)
Este paquete contiene clases utilitarias y de validación que encapsulan lógica común, como:

- Validations: Proporciona métodos para validar datos como nombres, CI (carnet de identidad), etc.

3. Capa de Utilidades (util)
Este paquete incluye constantes y generadores de datos que son utilizados en todo el sistema:

- ConstantesAnalisis y ConstantesEspecialidades: Contienen listas de valores predefinidos para análisis y especialidades.
- MockDataGenerator: Genera datos simulados para pruebas.

4. Capa de Frontend (frontend)
Este paquete contiene la interfaz gráfica de usuario (GUI) y está subdividido en:

- frontend.formularios: Maneja los formularios principales de interacción, como:
  - FormularioVisitas: Permite gestionar visitas médicas (crear, editar, visualizar).
  - FormularioPaciente: Gestiona la información de los pacientes.
- frontend.panelesPrincipales: Contiene paneles principales de la aplicación, como:
  - VentanaPacientes: Muestra la lista de pacientes y permite gestionarlos.
  - VentanaVisitas: Muestra las visitas médicas y permite gestionarlas.
- frontend.ui: Incluye componentes personalizados para la interfaz, como:
  - ScrollPaneModerno: Un JScrollPane con diseño moderno.
  - BotonBlanco: Botones estilizados.
  - BuscadorTabla: Componente para buscar en tablas.

5. Capa de Excepciones (excepciones)
Este paquete define excepciones personalizadas para manejar errores específicos, como:

- IllegalAddressException: Para direcciones inválidas.
- IllegalVaccinationException: Para errores relacionados con vacunación.

6. Capa de Ejecución (runner)
Este paquete contiene clases relacionadas con la autenticación y el usuario actual:

- Usuario: Representa a los usuarios del sistema y sus roles.
- Auth: Maneja la autenticación de usuarios.

División de Responsabilidades Del Equipo

David Alfonso Hernández

1. Diseño de Interfaz Gráfica (Swing/JavaFX)

- Creación de maquetas y prototipos de la interfaz usando herramientas como **Figma** o diseños en papel.
- Implementación de la UI utilizando **Swing** o **JavaFX** (según requerimientos).
- Asegurar que la interfaz sea intuitiva y funcional para el usuario final.

2. Desarrollo del Frontend (Lógica de Presentación)

- Conexión entre la interfaz gráfica y la lógica del backend.
- Validación de datos ingresados por el usuario.
- Manejo de eventos (botones, formularios, ventanas).

3. Generación de Reportes (Gráficos y Estadísticas)

- Implementación de gráficos para visualización de datos:
- **Gráficos de línea** (visitas mensuales).
- **Gráficos de barras** (distribución de edades de pacientes).

4. Gestión de Proyecto (Organización y Documentación)

- Seguimiento de tareas en **Notion**
- Documentación del código y manual de usuario básico.

Reynaldo Daniel Quesada García

1. Desarrollo del Backend (Lógica de Negocio)

- Programación de la lógica principal de la aplicación
- Manejo de estructuras de datos, algoritmos y procesamiento de información.

2. Persistencia de Datos (Guardado y Carga de Información)

- Garantizar que los datos se guarden y recuperen correctamente.

3. Seguridad y Validación de Datos

•Evitar inyección de código o corrupción de archivos.
•Validación de entradas críticas

4. Optimización y Pruebas

•Asegurar que la aplicación funcione sin errores en diferentes sistemas operativos.
•Pruebas manuales y corrección de bugs.

Metodología de Trabajo

•Control de versiones con Git
•Comunicación constante
•Pruebas manuales