Facultad de Ingeniería-UNMDP

Programación “C”

INFORME TRABAJO PRÁCTICO GRUPAL -PRIMERA PARTE-

Fecha de entrega: 12/10/2025
Docentes a cargo: Leonel Guccione, Guillermo Lazzurri, Ivonne Gellon

Integrantes:
Salvador Munduteguy, Federico Calla Aliende, Francisco Marino, Iván Etcheverry y Santiago Soldani

Introducción

Este informe tiene como objetivo describir la manera en que se pensó y llevó a cabo el sistema de funcionamiento de una clínica. Se explicará la organización de clases, paquetes, métodos relevantes que colaboran con el objetivo del proyecto, excepciones, problemas que se presentaron en el proceso, así como sus respectivas soluciones implementadas.

Objetivos del Proyecto

El objetivo principal para esta primera parte del proyecto es generar un sistema para una clínica, con el cual puedan realizar las siguientes tareas:

- Calcular honorarios médicos e informar sobre su actividad
- Crear y mostrar las facturas de los pacientes acorde a sus patologías y tratamientos
- Tratar la espera de los pacientes
- Poder asignar los distintos pacientes a sus correspondientes habitaciones
- Aplicar patrones abordados durante la cursada

Herramientas Utilizadas

Las siguientes herramientas han sido seleccionadas para el desarrollo:

Herramienta	Descripción
Lenguaje de Programación	Java
IDE (Entorno de Desarrollo Integrado)	Eclipse
Herramientas Adicionales	Gemini, Chat gpt, Thinking in Java 4th edition

Análisis del Problema:

Para encarar este trabajo, en primer lugar se comenzó analizando el contexto del problema, los objetivos concretos del sistema, las distintas entidades que participaron del mismo y la forma en la que organizaremos los datos.
De esta manera se modelaron y encapsularon las tareas que el sistema debería realizar en una clase sistema y a partir de allí, extenderse a las demás clases que interactúan con el mismo (desarrolladas posteriormente).

Uno de los primeros problemas que se detectaron fue la forma de estructurar y vincular las entidades del sistema, tales como pacientes, médicos, habitaciones, facturas y por supuesto, la clínica. Para solucionar ello, se decidió que en su mayoría, las clases tendrían que estar en una relación de composición con la clínica, y, de esta manera tener una colección de objetos perteneciente a la clínica, para cada entidad del sistema. Para los pacientes ya atendidos se usa un arrayList de pacientes, los cuales están ya siendo atendidos en la clínica. Así mismo con los médicos, que trabajan en la clínica, sin importar su especialidad, se encuentran en un arrayList de médicos. Para los pacientes que ya fueron registrados, pero que aún están esperando a ser atendidos, se concluyó que la colección con una naturaleza más afín al problema sería una cola, específicamente una LinkedList(cumple el principio FIFO) ya que actúa como una cola real de personas.
Finalmente con las habitaciones se presentaron varias alternativas, sin embargo, a prueba y error se seleccionó una colección de tipo Pila, ya que ahorraba el hecho de tener que hacer búsquedas y recorridos para encontrar habitaciones disponibles. De esta manera, si alguien necesita una habitación se toma de la pila, se la asigna, y en caso de que siga habiendo espacio se la reinserta. Así se logró la estructuración de las clases fundamentales del sistema, a continuación, se profundizará más en cada una.

Desarrollo del sistema

Clases Fundamentales:

Clínica

La clase clínica es “el centro” del sistema. Es quien lleva a cabo el almacenamiento de las entidades, facturas, atenciones médicas, lista de espera, etc. Dado que solo debe existir una única instancia de esta clase, se eligió aplicar el patrón “Singleton” en ella.

SistemaClinica

Esta clase es muy útil dado que desde ella se puede invocar los métodos fundamentales del sistema, como ingresar un paciente, mostrar la actividad de un médico, internar un paciente, etc.
Desde esta clase se logra acceder a todo el sistema. En este caso, su único atributo es de tipo clínica, pero sí en otro momento se decidiera crear otra clase que no tuviera nada que ver con ella, podría agregarse un atributo de ese tipo a SistemaClinica para también tener acceso a todos sus métodos públicos. Esta es la clase donde se aplicó el patrón “Facade”

Médico

Clase abstracta, sólo implementa algunos getters y su respectivo toString.
El cálculo de honorarios no se implementa en esta clase dado que existen distintos importes dependiendo de las características del médico. Se llegó a la conclusión de que la mejor manera de implementar esta diferencia de características es utilizando el patrón “Decorator”. Además, se utilizó el patrón “Factory” para crear las instancias de Médico.

Habitaciones

Para la parte de internación, se dispone de diferentes tipos de habitaciones, acordes al tratamiento que deba llevar el paciente. Se declaró una clase abstracta habitación con las características comunes de las mismas, dejando para las clases que la extienden (tipos de habitación) el cálculo de su costo. Además, se consideró un límite de personas por habitación: 1 persona para privadas y terapia intensiva, 5 para las compartidas. A la hora de asignar un paciente a una habitación, de acuerdo a la que deba ir, se tiene una pila de habitaciones disponibles. Se toma la primera que pueda alojarlo (siempre y cuando sea del tipo correcto, sino se sigue buscando) y en caso que se complete su capacidad, se la quita de la pila. Caso contrario, cuando se da de alta un paciente, se lo desvincula de la habitación y esta se la vuelve a colocar en la pila en caso de que hubiese sido quitada anteriormente.

Factura

En cuanto a la clase Factura, acompaña el ciclo de vida del paciente dentro de la clínica. Se crea al momento en que se da de alta el Paciente, guarda registro de su actividad: las distintas consultas con cierta cantidad de médicos, caso de internación y días adentro. Una vez otorgado el alta médico, comienza el proceso de egreso, aquí la factura será la encargada de informar el monto total a abonar, mediante un toString, muestra cada consulta médica realizada (con su correspondiente detalle), en caso de internación, en qué tipo de habitación se hospedó y acorde a los días la suma a pagar.

Paciente

La clase Paciente, es una clase abstracta extendida de persona. Esta clase simplemente modela las características que tiene que tener cualquier tipo de paciente, junto con sus comportamientos básicos. La disputa por un lugar en la sala de espera está modelado en las subclases Joven, mayor y ninio, extendidas de Paciente. Esta clase es la que dará lugar a la mayoria de las interacciones del sistema, ser atendido, hacer una cola en la sala de espera o el patio, ser internado, obtener una factura etc.

Patrones destacables:

Patrón Decorator

Para el modelado de los diversos tipos de médicos que puede presentar la clínica (especialidad, contrato y posgrado) se utiliza el patrón Decorator. Gracias a este mecanismo, se salva la llamada “explosión de clases” dado que cada combinación otorga un resultado diferente (por ejemplo, en el cálculo de los honorarios). Se considera una interfaz IMedico, la cual modela los métodos capaces de obtener un honorario, reporte, matrícula. Se implementa en dos clases abstractas: Medico y DecoratorMedico.

Medico, a su vez, extiende de Persona (ya que también tiene nombre, dni, domicilio, etc.) y plantea el método abstracto getHonorario. Este último método será definido acorde a las especialidades que puede tener un médico, las cuales extienden a esta clase.

Por su parte DecoratorMedico es extendida por DecoratorPosgrado (clase abstracta), la cual, a su vez es extendida por los tipos de posgrados, y por otro decorador: DecoratorContrato. Finalmente, de este último, se extienden las clases de los tipos de contratos existentes.

Este planteo del Decorator brinda mayor flexibilidad a la creación de una instancia, ya que todas las clases reciben un IMedico. Por ende, se aceptan médicos que no completen todos los campos, podrían no tener posgrado o contrato.

Gracias a la presencia del MedicoFactory, se comienzan a instanciar los médicos, con su respectiva especialidad (obligatorio) y se le agregan capas “decorando” esa clase base. Ante una clasificación desconocida se lanza su excepción acorde.

Double Dispatch

Para determinar quién ocupa la sala de espera, se aplicó el patrón Double Dispatch. Se define una clase abstracta Paciente que declara los métodos abstractos getGanador, enfrentaNinio, enfrentaJoven y enfrentaMayor. A partir de ella derivan las subclases Ninio, Joven y Mayor, que implementan estas operaciones codificando las reglas del enunciado.

La resolución se realiza en dos despachos dinámicos:

1. getGanador(Paciente otro) constituye el primer despacho, ya que se selecciona el método según el tipo dinámico del receptor (this).

2. Dentro de getGanador, se invoca en el objeto contrincante una sobrecarga específica (enfrentaNinio/…/enfrentaMayor), produciendo el segundo despacho según el tipo dinámico de otro.

Las principales ventajas de este enfoque son que brinda polimorfismo real, porque el resultado depende simultáneamente de los dos tipos dinámicos involucrados (el receptor y el argumento), y favorece la escalabilidad y el mantenimiento: para incorporar un nuevo tipo de Paciente no es necesario reescribir lógica existente, sino simplemente agregar sus métodos enfrentaXXX y definir cómo interactúa con los demás, respetando el principio de abierto/cerrado. Además, mejora la claridad del dominio, ya que cada clase concreta expresa explícitamente su comportamiento frente a las otras categorías, eliminando condicionales globales y concentrando la lógica donde corresponde.

Paquetes

Sin dudas la organización de los directorios, paquetes, y cualquier estrategia de almacenamiento, es lo que permite tener un trabajo más limpio, rápido, entendible y escalable, sin mencionar la cantidad de problemas que se pueden llegar a evitar. Es por

eso, que se dividió al sistema en distintos paquetes donde almacenar las clases y así tener un desarrollo más fluido y menos agotador. Todos los paquetes están dentro del src (paquete que indica donde estará el código de la app).

Clínica

En el paquete clínica, se trabajó todo sobre la estructuración de los datos y comportamientos intrínsecos a la clínica. Específicamente la clase Clínica, y la clase SalaDeEspera.

Facturación

En el paquete facturación únicamente se trabajó con la clase factura, no obstante, crear este paquete solamente para guardar una clase se consideró pertinente debido a que, si el dia de mañana se necesitan más tipo de facturas, dependiendo si se paga con obra social, con efectivo o en cualquier otro caso, no habría que mover la clase factura a otro paquete sino que la escalabilidad de la clase se extiende en el mismo, sin afectar la visualización de los demás componentes.

Hospedaje

En el paquete hospedaje, se guardó todo lo relacionado a las habitaciones, desde la clase abstracta Habitación, hasta sus hijas, el argumento de la creación de esta clase es la misma que la del paquete facturación.

Individuos

El paquete individuos guarda la clase Persona, una clase que solamente está para moldear el comportamiento de cualquier entidad que sea un humano dentro de la clínica (médicos, pacientes). Esta clase tiene su propio paquete, sin embargo, al principio de la creación del sistema se almacenaban los pacientes y médicos juntos, en el paquete individual. Pero finalmente se concluyó en que esto era costoso visualmente, así que se optó en dividir a cada uno en su propio paquete.

Medicos

En el paquete médicos se guardó cualquier tipo de interacción perteneciente a un médico, desde los patrones usados, como factory y decorator, hasta las clases de consulta médica que hablan de una relación paciente-médico, pero que solamente le es relevante al médico para saber su actividad reciente.

Pacientes

Misma lógica que con el paquete médicos, este paquete tenía como objetivo abstraer todo lo pertinente a un paciente, de esta manera, el patrón factory, double dispatch y las subclases están aquí dentro.

Sistema

Contiene la clase sistema y una interfaz ISistema que dicta lo que el sistema tiene permitido hacer con la clínica. De esta manera quedan implementadas todas las tareas del sistema como una serie de pasos de métodos de la clínica.

Prueba

Entry point de la app y donde se ejecuta la clase Prueba con su método estático main para comenzar a ejecutar el sistema.

Excepciones

En este sistema se utilizan una variedad de excepciones en distintos módulos con el objetivo de aislar problemas durante la ejecución y que sean resueltos en un ámbito distinto al de producción.

Ante violaciones de reglas (por ejemplo, intentar atender con la cola de espera vacía, cargar un rango etario inválido o referenciar un médico inexistente), las cuales no están especificadas como precondiciones, por la complejidad de la validación para el programador, se lanzan excepciones, con los mensajes y datos de contexto cuando corresponda.

Diagrama UML:

Diagrama UML adjunto aparte.

Conclusiones

La primera parte de este proyecto ayudó a comprender la utilidad de los distintos patrones de diseño. Facilitan el planteo de una correcta estructura de un sistema el cual estará abierto a su extensión y facilita su escalabilidad.

Hacer hincapié en que una correcta modularización, en un principio puede parecer tediosa, pero al evolucionar el proyecto se concluye en que es una práctica que hace la no repetición del código (reutilización) y bajo acoplamiento. Por ende a la hora de solucionar un problema, se lo puede abordar directamente y sin la necesidad de editar código que lo implemente.