UNIVERSIDAD NACIONAL de MAR DEL PLATA

Trabajo Práctico Grupal 2025

Primera Etapa

- **Asignatura:** Programación C
- **Integrantes del grupo:**
  - Aldana Juan
  - Distefano Gian Luca
  - Giordano Valentino
  - Proia Agustin
  - Yanosky Santiago
- **Fecha de Entrega:** 12/10/2025

Descripción General del Sistema

El **Sistema** que formamos a partir del intercambio de ideas general que realizamos previo al inicio del proyecto nos quedó conformado de la siguiente manera:
En primer lugar, sabíamos que tanto la sala de espera, como el patio iban a formar parte esencial del sistema, ya que nos indican donde "esperaba" el paciente a ser atendido, a su vez, cada uno de los pacientes ubicados aquí estaban ubicados en un determinado espacio de la lista de espera por ende también íbamos a necesitar un atributo que simule a la lista de espera.
Las habitaciones y los médicos fueron causa de dudas y preguntas, ya que nos preguntamos el tipo de relación que debían tener dentro de la clínica, llegamos a la conclusión de que si no hay clínica, ni las habitaciones ni los médicos deben existir, por eso, estos se cargan a una lista de habitaciones y una lista de médicos al iniciar el sistema, tal y como se ve en el constructor de este, primero se crean ambas listas y luego se cargan con valores *hard-codeados*, en un futuro cuando se implemente una pantalla interactiva esto muy probablemente termine cambiándose.
Las dudas, cabe aclarar, surgieron del siguiente fragmento de código de "prueba" otorgado por la cátedra en el enunciado del TPG:

```
sistema.registraMedico(medico1);
sistema.registraMedico(medico2);
……
sistema.registraPaciente(paciente1);
sistema.registraPaciente(paciente2);
….
sistema.ingresaPaciente(paciente1);
sistema.ingresaPaciente(paciente2);
sistema.atiendePaciente(medico1, paciente1);
sistema.atiendePaciente(medico2, paciente1);
factura1 = sistema.egresaPaciente(paciente1);
sistema.atiendePaciente(medico3, paciente2);
sistema.internaPaciente(paciente2,habitacion3);
factura2 = sistema.egresaPaciente(paciente2, cantidadDias);
```

Más exactamente, surgieron con el método ***registraMedico*** y el método ***internaPaciente***. Por ejemplo, en el primer método mencionado, se da a entender que los médicos son creados fuera de la clínica y luego se registran en este, cosa que a nuestro parecer no podía ser ya que el comportamiento de un médico sólo funciona si este es parte de una clínica.

Más adelante explicaremos en detalle cómo esta decisión tomó relevancia en nuestra implementación.

Otro atributo importante de la clase **Sistema** es la ***lista_atendidos*** que es un *ArrayList* de **Pacientes** al cual se le van añadiendo aquellos pacientes que fueron atendidos por primera vez (A los que se les asignó un médico). Esta lista nos permite retirar al paciente ya atendido de la lista de espera (del patio o la sala) y asignarle más médicos si fuese necesario. Cada paciente se mantendrá dentro de la *lista_atendidos* hasta que **egrese** del sistema.

El resto de los atributos del Sistema son los diferentes módulos que explicaremos de manera individual a continuación.

Módulos del Sistema

Hay que dejar en claro que cuando uno llama a los métodos del sistema como **ingresaPaciente**, **internaPaciente**, **registraPaciente**, y todos aquellos métodos relacionados con los módulos, el sistema *delega* estas tareas a los propios módulos que implementa el **Sistema** y estos se encargan de manera propia de realizar cada una de las tareas (se separan las responsabilidades).

- **Registro de pacientes:**

  Este es el primer módulo del Sistema, y al cual se le realiza el primer llamado en caso de querer registrar un nuevo paciente a la Clínica.
  Este módulo simplemente se encarga de añadir al paciente a la *lista de espera*.

- **Ingreso de pacientes:**

  Este es el segundo módulo del Sistema, y es quien se encarga de ubicar al paciente recién ingresado en la Sala de Espera o en el Patio, según corresponda. Esta asignación se va a hacer según la importancia que se le dé al paciente ingresado. Si la sala está ***vacía***, el paciente ingresado será asignado a esta. Si esta **no está vacía**, se compararán pacientes mediante la implementación del patrón ***Double Dispatch*** que nos permite tomar una decisión a partir de 2 objetos distintos, en este caso los pacientes.

Una vez concluida la comparación (si es que la hubo), el paciente ingresado debería estar asignado en el patio o en el caso que corresponda en la sala.

- **Atención a pacientes:**

El tercer módulo implementado del Sistema, este módulo se encarga de que al paciente se le adjunte una consulta médica, de parte del médico pasado como parámetro, y este módulo trabaja de la siguiente manera:

1. En primer lugar, verifica que el paciente que se vaya a atender haya sido debidamente ingresado y registrado, es decir que esté ubicado en la *lista de espera* o en la *lista de atendidos*, de no ser así, lanza una excepción del tipo **PacienteNoRegistradoException**.

2. En segundo lugar, verifica que el medico que vaya a atender esté licenciado para atender pacientes, es decir, que esté registrado en el sistema de la clínica. En caso de no estarlo lanza una excepción del tipo **MedicoNoRegistradoException**.

3. En el caso de que ambos pasos anteriores hayan sido satisfactorios. Añade el médico al historial de consultas médicas del paciente, este "historial" es un *ArrayList* de **consultas médicas** que forma parte de los atributos del paciente y será de vital importancia para la generación de la factura.

4. Verifica que el paciente esté en la *lista de espera*, si no está en esta necesariamente está en la *lista de atendidos*. De cumplirse que esté en la *lista de espera*, al llamar al método de este módulo, se elimina al paciente de la *lista de espera* y del patio/sala según corresponda y se lo añade a la *lista de atendidos*.

- **Internación a Pacientes:**

  Este módulo es el encargado de asignarle una habitación a aquellos pacientes que requieren ser internados. El problema que nos encontrábamos acá, indicado en un uno de los párrafos de arriba era el hecho de que en el ejemplo del enunciado se pasaba la habitación como parámetro, pero en nuestra forma de ver la relación entre las habitaciones y la clínica (una de **composición**) pasar la habitación como parámetro no era factible, ya que debías previamente pedirle las habitaciones al sistema, guardar la habitación solicitada en una variable del tipo habitación y pasarla luego como parámetro. La solución que encontramos fue hacer que, por parámetro, se pasen al paciente a internar y el tipo de habitación en la cual se quiere internar al paciente, luego el módulo del sistema se encarga de buscar alguna habitación del tipo solicitado, y en caso de encontrar una habitación del tipo solicitado que esté libre, internar al paciente, asignando al atributo habitación del paciente la referencia a la habitación a ocupar, y a su vez ocupar la habitación.
  El método declarado en el sistema, se encarga de buscar la habitación solicitada, en caso de no encontrar una habitación del tipo solicitado libre, el método lanzará una excepción de tipo **LugarNoDisponibleException** la cual debe ser manejada por quien intentó internar al paciente y resolver el inconveniente, otra excepción posible es que el tipo solicitado de habitación no exista, en tal caso se lanza la excepción **NoExisteHabitacionException**, como parámetro se le va a pasar el *tipo de habitación* que se le quería asignar al paciente.
  Otra excepción que puede ser lanzada por el método es la de **PacienteNoRegistradoException**, en el caso de querer internar a un paciente que no se registró en el sistema.

- **Egreso de Paciente:**

  Este es el último módulo que compone el Sistema, es quien se encarga de principalmente generar la factura correspondiente a la atención médica que recibió el paciente. Este además se encarga de cargar las consultas médicas realizadas por cada médico en su “historial” personal de consultas, por cada médico que atendió al paciente se genera una consulta, que detalla *nombre del paciente*, *fecha* y *honorario* y que posteriormente se guarda en el atributo consultasMedicas del médico en cuestión.
  El método **egresaPaciente** tiene una sobrecarga, uno de los 2 métodos recibe un parámetro *dias*, y otro no, en el caso del primero es cuando el paciente llegó a

internarse, por ende se debe aclarar la cantidad de días que este pasó en la clínica, también es necesario diferenciar en 2 métodos los pacientes que se internaron y los que no, ya que los que se internaron deben liberar la habitación la cual estaban ocupando, ya que si esto no se hace podría traer problemas en módulos como el de Internación.

Una curiosidad del método es que, a la hora de pasarle las consultas médicas realizadas por el paciente, se fija cuales corresponden al periodo actual, ya que un paciente *conserva su historia clínica* y esta se va alargando a medida que el paciente se atienda. Por esta razón el método se encarga de buscar y enviar solo aquellas consultas realizadas en el periodo de estadía del paciente, para así no cobrar cosas que no correspondan.

Este módulo, en caso de intentar egresar un paciente el cual no fue atendido lanzará una excepción del tipo **PacienteNoAtendidoException**, que recibe como parámetro al paciente al cual se quería egresar. Lo mismo en el caso de intentar egresar un paciente que no está registrado, lanzará una excepción del tipo **PacienteNoRegistradoException**.

- **Módulo de Registros:**

  Este módulo es el encargado de manejar los distintos registros que lleva el sistema, para las "Historias Clínicas" de los pacientes y el Historial de Consultas de los médicos. Este es llamado en ciertos métodos del Sistema:

  - **IngresaPaciente**: A la hora de que un paciente ingrese o reingrese a la clinica, se llama a **addPaciente** del módulo registros para que cree la nueva historia clínica del paciente en caso de ser la primera vez que llega a la clínica, si fuera el caso de que no es su primera vez no hace nada.
  - **AddMedico**: Cuando un nuevo médico se suma a la clínica, el sistema además de añadirlo a su *lista de médicos*, llama al método **addMedico** del módulo registros y crea su nuevo historial de consultas.
  - **AtiendePaciente**: Cuando se atiende un paciente, el módulo de registros se encarga de 2 cosas, actualizar la historia clínica del paciente y actualizar las consultas realizadas por el médico.

Clases Principales

A continuación, vamos a describir la composición y el comportamiento de las clases que nosotros consideramos fundamentales para el funcionamiento del programa, las cuales son la clase **Médico** y **Paciente**:

- **Médico:**

  La clase Médico es una subclase de la clase **Persona**, es decir hereda todos los métodos, atributos e implementa las mismas interfaces que esta clase. La clase Médico a su vez implementa una *interface médico* que se realizó para modelar el comportamiento de los médicos y también ya que iba a ser de utilidad a la hora de modelar el decorator, ya que para el cálculo de sueldos de cada médico precisamos el uso del patrón **Decorator**, los atributos implementados en la clase médico son *numMatricula*, *especialidad*, *sueldo* (Médico base, sin modificar por especialidad, estudios o tipo de contratación).

  Lo más llamativo de la clase Medico es la forma en la que se calcula su sueldo, ya que es a través del patrón **Decorator**, debido a que el sueldo depende exclusivamente de 3 factores:

  - Especialidad: Cirugia - Pediatria - Clinica
  - Título de Posgrado: Magister - Doctor
  - Tipo de Contratación: Permanente - Residente

  Según el valor que tomarán estos 3 parámetros, el sueldo del Médico iba a ser uno u otro. Es así como gracias al patrón **Decorator** pudimos calcular de forma dinámica el sueldo de cada Médico. En nuestro caso cada decorador va a recibir un honorario previamente calculado. Así fue como lo razonamos nosotros:

  1. Creamos las clases concretas donde cada especialidad era el nombre de esa clase, y donde lo único modificado de la superclase **Médico** era el **getSueldo**, que el valor que devuelve variaba según el tipo de Médico que era. A su vez estas clases implementan una *interface especialidad*, que modela el comportamiento “extra” de estas clases extendidas de Médico, por otro lado, esta interface es implementada por nuestro próximo Decorador, el de tipo de Posgrado.

2. Ahora llegamos al primer Decorador que funciona realmente como decorador, el **DecoratorPosgrado**, que implementa la interface *Interfaz_Medico* y la interface *Interfaz_Especialidad*, esta clase abstracta va a implementar todos los comportamientos de las interfaces y luego de esta misma clase se van a extender los **DecoratorPosgradoDoctorado** y **DecoratorPosgradoMagister** que son quienes van a modificar dinámicamente el sueldo de nuestro **Médico Especializado,** el cual es el "encapsulado" de tipo *Interfaz_Especialidad* que implementa la clase Decorator en cada caso según corresponda.
3. Por último, tenemos el último Decorator, el **DecoratorContratacion**, que implementa solo la *Interfaz_Medico*, y que se extiende luego en 2 subclases, **DecoratorContratacionPermanente** y **DecoratorContratacionResidente**, que como en el caso anterior, van a modificar de manera dinámica el sueldo de nuestro **Médico Doctor** o **Magister** (Especialista), quien como en el Decorator anterior es el "encapsulado" en este caso de tipo **DecoratorPosgrado** que implementa el Decorator en cada caso según corresponda.

- **Paciente:**

  Al igual que la clase Médico, la clase **Paciente** hereda de **Persona**. Como atributos propios se tiene el *número de historia clínica*; un *rango etario*, que puede tener los valores de Nino (Niño), Joven y Mayor.

  El *rango etario* cumple un rol importante a la hora de definir la prioridad en la *Sala de espera*, ya que según las distintas combinaciones de rangos etarios se le va a asignar la sala al que tenga prioridad. Para resolver este conflicto se necesita la utilización del patrón **Double Dispatch**, y es por esto que el paciente implementa la interfaz *IPacientesComparables*, la cual permite utilizar el patrón.

  Como el paciente es abstracto tiene que ser extendida, en este caso, se extiende en la clase **Nino**, **Joven** y **Mayor**, la funcionalidad principal de estas clases es implementar los métodos de la interfaz.

Descripción Clases

A continuación, en orden alfabético, vamos a hacer una resumida descripción de las clases que no se mencionaron previamente:

- **Consulta**: Clase que representa una consulta médica. Contiene atributos como el *nombre del paciente*, la *fecha de la consulta* y el *honorario del médico*. Se utilizan para armar los registros de los médicos.

- **ConsultasHistoricas**: Es el tipo de consulta de las cuales se componen las historias clínicas de los pacientes. Poseen el *nombre del médico* que realizó la consulta, la *fecha de la consulta*, la *especialidad del médico* y el *honorario* que este cobró.

- **Decoradores.** Todos los decoradores utilizan la misma forma que vimos durante las clases prácticas, usando encapsulados para "decorar" a este mismo. Utilizados para decorar el *tipo de Posgrado* y el *tipo de Contratación*.

- **Domicilio:** Clase que representa un domicilio con atributos como *calle* y *número*.

- **factoryHabitaciones:** Clase **Factory** para crear instancias de diferentes tipos de habitaciones. Posee un método estático para que se pueda acceder sin necesidad de crear una instancia de la clase, se le pasa un *tipo de habitación* y un *costo fijo*, el método puede lanzar una excepción de tipo **NoExisteHabitacionException**.

- **FactoryMedicos**: Clase **Factory** para la creación de **Medicos**. Posee un método estático que crea una *interfaz_medico* que es un médico ya decorado, se le deben pasar todos los parámetros correspondientes para la creación de un Médico.

- **Facturacion**: Clase que genera y guarda facturas de los clientes. El método que muestra la factura va a distinguir entre si el paciente fue internado o no.

- **Habitacion**: Clase abstracta que representa una habitación. A partir de esta se crearon las subclases de **HabitacionCompartida**, **HabitacionPrivada** y **Sala de Internacion**.

- **HabitacionCompartida**: Clase que representa a la habitación compartida, es una subclase de habitación. Se le añaden los atributos para verificar las camas por habitación.

- **HabitacionPrivada**: Clase que representa una habitación privada. es una subclase de habitación.

- **Patio**: Clase que representa el lugar donde "esperan" los pacientes que son ingresados, siempre y cuando no les corresponda ir a la sala de espera.

- **Sala**: Clase que representa el lugar donde "espera" un solo paciente. A medida que van llegando pacientes el paciente que se encuentre en la sala puede ir cambiando.

Descripción Excepciones

Las excepciones utilizadas a lo largo del programa en esta primera parte fueron las siguientes:

- **LugarNoDisponibleException**: Esta excepción puede o no estar según como implementes las habitaciones en el sistema. Según nuestro punto de vista no vimos adecuado crear habitaciones a medida que se iban necesitando ya que eso no es algo que pase en las clínicas, sino que se cuenta con un número limitado de habitaciones, por este motivo creamos esta excepción que se lanza cuando se quiere internar a un paciente en una habitación de un tipo específico y no hay habitaciones libres del tipo solicitado.
- **MedicoNoRegistradoException**: Esta excepción ocurre en el caso de que se intente asignar a un médico que no está en el sistema a un paciente, cosa que no está permitido.
- **NoExisteHabitacionException**: Esta excepción es lanzada cuando se quiere crear o se quiere internar a un paciente en una habitación de un tipo que no es "Habitación Compartida", "Habitación Privada" o "Sala de Internacion".
- **PacienteNoAtendidoException**: Esta excepción se lanza en el caso de querer egresar a un paciente que no ha sido atendido, que no tendría sentido alguno por eso elegimos que lance una excepción.
- **PacienteNoRegistradoException**: La excepción es generada en el caso de querer atender a un paciente que no esté en la lista de espera, ni en la lista de atendidos (En caso de que ya se haya atendido) y también en el caso de querer internar a un paciente que tampoco este en ninguna de las listas.

Interfaces

A continuación, se describe lo relacionado al uso de *interfaces del programa*, las mismas subsanan una necesidad que surge en la POO, que una clase puede heredar múltiples clases. Si bien no hacen exactamente esto, las *interfaces* permiten la herencia de un contrato abstracto. Las *interfaces* son simplemente una lista de cabeceras de métodos, los mismos deberán ser implementados por todas aquellas clases que implementen la interfaz. Es decir, la *interfaz* obliga a seguir un comportamiento predefinido, sin especificar cómo se sigue tal comportamiento en cada caso. Esto estandariza el código y permite trabajar sabiendo que cosas similares se comportan de manera similar, incluso si la implementación está encapsulada y difiere entre los casos. Las interfaces utilizadas en el programa son:

- **IHabitacion:**

  IHabitacion será implementada por la clase abstracta **Habitacion**, que será posteriormente extendida por cada tipo particular de habitación.

  Contiene los métodos de las habitaciones, que se utilizan para internar. Las funciones especificadas son necesarias para la posterior generación de la factura, mediante el cálculo del costo de internación.

- **Interfaz_Persona:**

  Interfaz_Persona será implementada directa o indirectamente, por herencia, por todas aquellas clases cuyos objetos sean **Personas**, es decir, **Médicos** y **Pacientes**. Todas estas clases y sus subclases extienden la clase persona, que implementa la *interfaz*.

  Contiene los métodos referentes a la información más básica de las personas, la que es común a todas ellas y necesaria para su identificación.

- **Interfaz_Medico e Interfaz_Especialidad:**

  Interfaz_Medico, que extiende de *Interfaz_Persona*, será implementada por la clase abstracta **Medico**, la cual será posteriormente heredada por cada una de las especialidades, que implementan la *Interfaz_Especialidad*.

  Contienen los métodos necesarios para identificar un médico como tal, y no como una persona, es decir por su matrícula. Contienen también métodos necesarios para implementar posteriormente el patrón **Decorator**, para los sueldos específicos de cada médico según sus características.

- **IPacientesComparables:**

  IPacientesComparables será implementada por la clase abstracta **Paciente**, de la cual extienden los distintos tipos de paciente.

  Contiene los métodos necesarios para implementar posteriormente el **Double Dispatch** según el tipo de paciente, que organizara las salas de espera según los pacientes que vayan ingresando.

Diagrama UML