Universidad Nacional de Mar del Plata.

Facultad de Ingeniería.

Alumnos:

Bellone, Martín;

García Di Martino, Laureano

Sorrenti, Enzo;

Paniagua, Lucas;

Zuviría, Nicolás

Programación C

Trabajo Práctico Grupal

Docentes a cargo:

Gellone, Ivonne;

Lazzurri, Guillermo Carlos

Guccione, Leonel Domingo

De Lellis, Lucas

Fecha de entrega:

Lunes 13 de Octubre de 2025.

Introducción

Para este proyecto se requirió el desarrollo de módulos de software para incorporar al sistema de gestión de una pequeña clínica privada. Los módulos solicitados fueron: Facturación al Paciente, Reportes de Actividad de los Médicos (con suma de honorarios), Resolución de Conflictos de Sala de Espera. Con el objetivo de mantener los módulos escalables y abiertos para su extensión se debieron respetar los principios SOLID de la programación orientada a objectos.

En esta primera entrega del informe se realizará un resumen de las clases más significativas del modelo planteado, de las decisiones respecto a los módulos mencionados y de las implementaciones de los patrones de diseño requeridos: El patrón *Facade,* el patrón *Decorator* y el patrón *Double Dispatch*. Cabe recalcar que el presente informe no es reemplazo de la lectura de la documentación presente en carpeta JavaDoc.

Clases más significativas

Clínica:

Esta clase es donde usamos el patrón *Singleton* y *Facade*. Es la clase que contiene atributos como el nombre, la dirección, el teléfono, la ciudad, médicos registrados, pacientes registrados, y sistemas de ingreso, egreso y reportes.

Se usó el patrón *Singleton* para asegurar la existencia de una única instancia con el método *getInstancia*, esto significa que no va a haber más de una clínica.

También se implementó el patrón *Facade* para interactuar de manera más sencilla con los subsistemas que tiene el proyecto. Por medio de los métodos *registrarPaciente*, *registrarMedico*, *ingresarPaciente*, *egresarPaciente*, *atenderPaciente*, *reportesMedicos*, *internarPaciente*. Dichos métodos delegaban la responsabilidad de resolver la implementación a cada módulo correspondiente.

Otros métodos destacables de dicha clase son los métodos *atenderPaciente* y el método *internarPaciente.* En el primero se retira un paciente de la sala de espera y se lo pasa a la lista de pacientes en atención y se registra la consulta para el paciente y para el médico asignado. En el segundo simplemente se registra a un paciente en una habitación y se la marca como ocupada.

Paciente:
Para crear los objetos se usó una clase *PacienteFactory*, en ella se crean pacientes según el rango etario (Joven, Mayor o Niño). Estas clases están implementadas usando el patrón *Double Dispatch* para manejar la prioridad en la sala de espera.

Ejemplo en clase Niño:

```
public PacienteNino(/*…atributos de clase persona… */, String
nroHistoriaMedica, LocalDate fechaIngreso)

{
        super(nombre, apellido, dni, domicilio,
telefono,fechaIngreso,   nroHistoriaMedica);
}


@Override
public boolean prioridad(Paciente paciente){
    return paciente.prioridadConNino();
}

@Override
public boolean prioridadConJoven(){
   return true; // la sala queda para niño
}


@Override
public boolean prioridadConMayor(){
   return false; // el niño va para el patio
}


@Override
public boolean prioridadConNino(){
   return false; // el niño anterior queda en la sala
}
```

Médico:

Para crear este tipo de persona se utiliza la clase *MedicoFactory*, que instancia a los médicos según su especialidad (Cirujano, Clínico o Pediatra), contratación (Permanente o Residente) y posgrado (Magister, Doctorado o de Grado). En esta clase se implementa el patrón *Decorator* para especificar los tipos de contrataciones y posgrados.

El siguiente código ejemplifica nuestra implementación del *MedicoFactory*, el cual instancia al médico en 3 pasos:

- Primero se aplica la Especialidad.
- Luego se decora con el Posgrado.
- Por último, se le aplica la decoración por Contratación.

```
/*
    …
    variable medico ya instanciada con la especialidad
   */
   DecoratorPosgrado decoratorPosgrado = null;
   if(titulo.equalsIgnoreCase("magister")){
       decoratorPosgrado = new DecoratorPosgradoMagister(médico);
   }
   else
     if(titulo.equalsIgnoreCase("doctor"))
     {
        decoratorPosgrado = new DecoratorPosgradoDoctorado(médico);
     }
     else
        throw new TituloNoExistenteException("El tipo de postgrado
ingresado no existe");

   DecoratorContratacion decorator = null;
   if(contratacion.equalsIgnoreCase("permanente")){
      decorator = new
DecoratorContratacionPermanente(decoratorPosgrado);
   }else
      if(contratacion.equalsIgnoreCase("residente"))
      {
        decorator = new
DecoratorContratacionResidente(decoratorPosgrado);
      }
      else
        throw new ContratacionNoExistenteException("El tipo de
contratación ingresada no existe");

   return decorator;
```

Habitaciones:

Para las habitaciones se aplicó el patrón *Factory* nuevamente, implementando la clase *HabitacionesFactory* que instancia la habitación solicitada, devolviendo un objeto de tipo *Habitacion*, ya sea privada, compartida o de terapia intensiva. Dichos objetos se encuentran agrupados en un *ArrayList* dentro de la clase clínica.

Descripción de Módulos

Módulo de Ingreso:
Este módulo se encarga de manejar la incorporación de un paciente a la clínica llevándolo a la sala de espera, en esa clase se realiza el análisis para saber si el paciente irá a la sala de espera privada o al patio. En este proceso se utiliza el patrón *Double Dispatch* para la prioridad de los pacientes, de acuerdo con las especificaciones dadas.
Además, se encarga de sacar los pacientes de la sala de espera y devolver un valor booleano según la respuesta dependiendo si se logró retirar al paciente de la sala de espera o no. Esta respuesta la usa el método *atenderPaciente* para manejar la atención.

Módulo de Egreso:
Este módulo gestiona la generación y emisión de la factura, permitiendo la salida del paciente tras el egreso.
La factura se genera a través del método *egresarPaciente* de la clase *SistemaDeEgreso*. Este método utiliza los datos del paciente y su historial de atenciones/consultas para crear la factura.
La factura tiene un paciente asignado, con relación de composición, un número de factura y un campo detalle que tiene la información sobre las consultas realizadas y el costo final de la misma. Asimismo, previo a retornar una factura, el método de *egresarPaciente* se encarga de eliminar las consultas realizadas por el paciente, para que en caso de un reingreso a la clínica no se cuenten las consultas ya realizadas a la nueva factura. Cabe aclarar que estas consultas se eliminan únicamente del lado del paciente, el médico que lo atendió mantiene registro de dichas consultas para luego poder retornar su reporte.

Módulo de Reportes:
Este módulo se encarga de llevar un registro de las consultas médicas realizadas por cada paciente y, simultáneamente, otro registro de pacientes atendidos por cada médico. Este sistema actualiza dichos registros de cada vez que un médico atiende a un paciente, y elimina las consultas realizadas por un paciente cada vez que este egresa, usando un *HashMap* de *ArrayList* para cada uno.