TRABAJO PRÁCTICO GRUPAL - PROGRAMACIÓN C

Grupo 8 - integrantes:

- Mariano Andrés Horianski
- Fernando Daniel Rosal
- Jazmín Milagros Piriz
- Jennifer Beltrán Flores

Este informe buscará describir brevemente la lógica de las clases y los métodos más destacables del código. El diagrama de clases estará adjuntado aparte por temas de legibilidad.

package clinica;

- SingletonClinica.java:
  Esta es la clase que se comporta como Facade, concentra toda la lógica del programa y, además, utiliza el patrón Singleton, pues la instancia de la clínica debe ser única (sería un error hablar de más de una clínica en este caso).

  Algunos métodos que vale la pena mencionar:
  - *public static SingletonClinica getInstance()*
    Corresponde al patrón Singleton, si la instancia es nula la crea, si ya existe devuelve la existente.
  - *public void ingresaPaciente(Paciente p) throws PacienteNotFoundException*
    Usa el HashMap de "pacientes" (pacientes registrados en el sistema) para verificar si el paciente existe y está registrado, en caso de no estarlo lanza una excepción y en caso de estarlo se lo añade a la lista de espera.
    Si la sala de espera privada está vacía, se manda al paciente allí, en caso de no estarlo se utiliza el Double Dispatch (paquete *double.d.dispatch*) para resolver la lógica de prioridades y decidir si el paciente se va al patio o se queda en la sala de espera (en cuyo caso la persona que antes ocupaba la sala de espera se va al patio).
  - *public void atiendePaciente(IMedico m, Paciente p) throws PacienteNotFoundException*
    Si el paciente no está registrado lanza una excepción.
    Se quita al paciente de la lista de espera. Si estaba en el patio de lo quita del patio y si estaba en la sala de espera se lo quita de ella, en caso de haber más pacientes la sala de espera se ocupa con el siguiente y se lo quita del patio.
    Se lo añade a la lista de pacientes en atención y se le asigna su fecha de ingreso. Finalmente se crea una Consulta (carpeta *clinica.model*).
    En esta última parte se utilizan los atributos:

private HashMap<Paciente, ArrayList<Consulta>> consultasPorPaciente;
private HashMap<IMedico, ArrayList<Consulta>> consultasPorMedico;
Si consultasPorPaciente no posee al paciente de la consulta, se agrega un nuevo par clave-valor con el paciente y un nuevo ArrayList que representa sus consultas.
Luego, independientemente de si el paciente existía anteriormente o se acaba de crear, se añade la consulta a su ArrayList.
Ídem para el médico.

- *public Factura egresaPaciente(Paciente p)*
  Usa un atributo del tipo IHabitacion para obtener la habitación del paciente en caso de haber sido internado, en caso de no haberlo sido regresa null (que es importante para la impresión de la factura en pantalla).
  Se crea un ArrayList de consultas con "consultasPorPaciente.get(p)", que sería el ArrayList con las consultas del paciente dado.
  Al final se crea la factura del paciente, se quita su ArrayList de consultas del HashMap, se lo quita de la lista de internados, se lo quita de la ListaEnAtencion y se cambia su fecha de ingreso a null para que se renueve en caso de volver a ser atendido.
- *public IMedico crearMedico(...)*
  Delega la responsabilidad de crear un médico a su respectivo Factory y atrapa sus excepciones. Presenta polimorfismo dependiendo de si tiene posgrado y contratación, uno de ellos o ninguno (base);
- *public IHabitacion crearHabitacion(String tipo)*
  Delega la creación de la habitación al HabitacionFactory, crearPaciente usa la misma lógica con su respectivo Factory.
- *public Reporte generarReporte(IMedico medico, LocalDate fechaInicio, LocalDate fechaFin) throws MedicoNotRegisteredException*
  Genera el reporte del médico solicitado dentro de un rango de fecha especificado.
  Lanza una excepción en caso de que el médico no esté registrado en la clínica, en caso de que lo esté asigna al ArrayList consultasMedico el ArrayList dentro del HashMap consultasPorMedico del mismo y lo utiliza para hacer el reporte.

package clinica.d.dispatch;

- *IPrioridad.java:*
  Esta es la interface del Double Dispatch que indica el comportamiento de cada subclase de paciente según los criterios de prioridad de la cátedra.

Se utiliza de esta manera ya que el criterio de comparación depende de dos clases.

- *public boolean prioridadSala(IPrioridad paciente);*
  Será el método, definido en las clases concretas que implementan esta interface, que haga el doble envío, utilizando su parámetro para llamar al método secundario.
  Tanto este método como los restantes de la interface estarán redefinidos en las clases Nino, Joven y Mayor para devolver resultados coherentes según el criterio de prioridad señalado.

package clinica.exceptions;

Paquete donde se definen las excepciones, todas con nombres representativos.

package clinica.factories;

Paquete donde se definen los Factory para Medico, Habitación y Paciente.

package clinica.habitaciones;

- IHabitacion.java
  Es la interface que define el comportamiento de las habitaciones Compartida, Privada y TerapiaIntensiva.
  Cada una de estas habitaciones tiene su propio costoDia como constante, que es utilizado en el método *public double calcularCosto(long dias)* para calcular el monto a pagar por el paciente según el criterio especificado por la cátedra de cada una.
- SalaEspera.java
  Es la clase que corresponde a la sala de espera privada, tiene métodos para ingresar un paciente a la sala, para desocuparla, consultar el paciente en ella y consultar si está ocupada o no (en ese orden).

package clinica.model;

- Consulta.java
  - *public Consulta(...)*
    Constructor de la clase que presenta polimorfismo, dependiendo de si se lo invoca con una fecha o no. Lo más importante a destacar es que es en esta parte donde se agrega el 20% sobre el costo del médico.

    - *public int compareTo(Consulta otra)*
      Reescribe el metodo compareTo de la interface Comparable para comparar dos consultas por fecha.
- Domicilio.java
  Clase utilizada como atributo en la clase abstracta Persona.
- Especialidad.java
  Clase abstracta extendida por MCirujano, MClinico y MPediatra. Cada una de estas subclases son responsables de calcular su honorario.
- Factura.java
    - *public Factura(Paciente paciente, IHabitacion habitacion, ArrayList<Consulta> consultasPaciente)*
      Constructor de la clase donde se inicializan los atributos, se incrementa el contador Static y se invoca el método setFechaConsultas().
    - *private double calcularTotal()*
      Calcula el importe total a pagar, primero calcula el importe de cada consulta médica y si la habitacion no es null (el paciente tuvo que ser internado) se le suma el importe de la habitación más el costo por internación
    - *private void setFechaConsultas()*
      En este método se asigna a todas las consultas del paciente su fecha de egreso, pues la fecha de consulta debe ser la fecha de facturación.
    - *public String toString()*
      Este es el Override del método toString que se encarga de mostrar todo el formato de la factura. Si el paciente fue internado (habitacion distinto de null) se agrega la habitación y su costo a la factura, si no lo fue, se omite.
- IMedico.java
  Interface que funciona como el encapsulado y componente del Decorator. Es implementada por la clase Medico.
- Medico.java
  Esta clase es el componente concreto del Decorator, tiene una constante para el honorario base, implementa IMedico y extiende Persona.
- Paciente.java
  Clase abstracta que extiende a Persona e implementa IPrioridad (interface del Double Dispatch).
- Persona.java
  Clase abstracta que define el comportamiento en común de Medico y Paciente.
- Reporte.java
  Es la clase encargada del reporte de actividad de un médico.
    - *public Reporte(IMedico medico, LocalDate fechaInicio, LocalDate fechaFin, ArrayList<Consulta> consultasMedico)*

Es el constructor de la clase. Recibe el ArrayList consultasMedico del HashMap consultasPorMedico del Singleton, si este ArrayList del HashMap es null para este médico se crea una nueva lista y si no se asigna la existente.

- public String toString()
  Es el Override del metodo toString que muestra la actividad del médico con su formato correspondiente.
  Si el médico no tuvo consultas, se informa; si no, el *Collections.sort(consultasMedico)* las ordena según el criterio redefinido en el *compareTo* de la clase Consulta, que en este caso es la fecha.
  Se recorren las consultas con un *for* y se ignoran tanto las que no tienen fecha como las que están fuera del rango de las fechas solicitadas, mostrando la fecha de consulta como encabezado sólo cuando esta cambia.

package clinica.model.decorators;

- DecoratorMedico.java
  Este es el Decorator. Es una clase abstracta que tiene una referencia a IMedico (encapsulado) además de implementarla. Es extendida por otras clases abstractas (DecoratorPosgrado y DecoratorContratacion) que a su vez son finalmente extendidas por sus respectivos Decorator concretos.
  Este patrón se encarga de añadir la responsabilidad a los objetos IMedico de calcular el honorario de cada médico de manera dinámica, dependiendo de su contratación y posgrado (la especialidad es un atributo propio del médico, por lo que no se implementa acá).