Documentación del Sistema de Gestión de Citas Médicas

Introducción

Este documento describe la implementación del sistema de gestión de citas médicas desarrollado en Python utilizando el patrón de diseño Singleton para la gestión del sistema, Observer para notificaciones y Strategy para la generación de reportes. El sistema permite registrar doctores, pacientes, agendar citas, cancelar citas, enviar notificaciones, y generar reportes.

Clases Principales

1. Clase `SistemaGestionCitas`

- **Descripción:** Gestiona las principales funcionalidades del sistema, como el registro de doctores y pacientes, la creación de citas, y la generación de reportes. Implementa el patrón Singleton para asegurar que solo exista una instancia del sistema.
- **Métodos:**
  - `__init__(self)`: Constructor de la clase, inicializa los atributos principales.
  - `getInstance(cls)`: Método de clase que implementa el patrón Singleton.
  - `registrar_doctor(self, doctor)`: Registra un nuevo doctor en el sistema.
  - `registrar_paciente(self, paciente)`: Registra un nuevo paciente en el sistema.
  - `agendar_cita(self, paciente_id, doctor_id, horario)`: Agenda una cita para un paciente con un doctor específico.
  - `cancelar_cita(self, cita_id)`: Cancela una cita previamente agendada.
  - `generar_reporte(self, tipo_reporte)`: Genera un reporte basado en el tipo especificado.

2. Clase `Medico`

- **Descripción:** Representa a un doctor en el sistema. Cada doctor tiene un identificador, nombre, especialidad y una lista de horarios disponibles.
- **Atributos:**
  - `id`: Identificador único del doctor.
  - `nombre`: Nombre del doctor.

    - `especialidad`: Especialidad médica del doctor.
    - `horarios`: Lista de horarios disponibles para citas.
- **Métodos:**
    - `__init__(self, id, nombre, especialidad)`: Constructor de la clase, inicializa los atributos del doctor.
    - `agregar_horario(self, horario)`: Agrega un horario disponible para el doctor.
    - `esta_disponible(self, horario)`: Verifica si el doctor está disponible en un horario específico.

3. Clase `Paciente`

- **Descripción:** Representa a un paciente en el sistema. Cada paciente tiene un identificador, nombre y una lista de citas.
- **Atributos:**
    - `id`: Identificador único del paciente.
    - `nombre`: Nombre del paciente.
    - `citas`: Lista de citas agendadas por el paciente.
- **Métodos:**
    - `__init__(self, id, nombre)`: Constructor de la clase, inicializa los atributos del paciente.
    - `agendar_cita(self, cita)`: Agenda una cita para el paciente.
    - `cancelar_cita(self, cita_id)`: Cancela una cita del paciente.

4. Clase `Cita`

- **Descripción:** Representa una cita médica entre un paciente y un doctor en un horario específico.
- **Atributos:**
    - `id`: Identificador único de la cita.
    - `paciente`: Referencia al paciente que agendó la cita.
    - `doctor`: Referencia al doctor con quien se agendó la cita.
    - `horario`: Horario en el que se llevará a cabo la cita.
- **Métodos:**
    - `__init__(self, id, paciente, doctor, horario)`: Constructor de la clase, inicializa los atributos de la cita.

5. Clase `Horario`

- **Descripción:** Representa un horario específico disponible para agendar citas.
- **Atributos:**
    - `fecha_hora`: Fecha y hora del horario disponible.
- **Métodos:**
    - `__init__(self, fecha_hora)`: Constructor de la clase, inicializa la fecha y hora del horario.

    - `get_fecha_hora(self)`: Devuelve la fecha y hora del horario.

6. Clase `Notificacion`

- **Descripción:** Gestiona el envío de notificaciones a pacientes y doctores. Implementa el patrón Observer para notificar cambios en las citas.
- **Métodos:**
    - `__init__(self, mensaje, destinatario)`: Constructor de la clase, inicializa el mensaje y destinatario de la notificación.
    - `enviar(self)`: Envía la notificación al destinatario.

7. Clase `Reporte`

- **Descripción:** Genera reportes sobre las citas, la demanda de doctores, y otros aspectos del sistema. Implementa el patrón Strategy para permitir diferentes tipos de reportes.
- **Métodos:**
    - `generar(self)`: Genera el reporte según el tipo de estrategia implementada.

Patrones de Diseño Utilizados

- **Singleton:** Utilizado en la clase `SistemaGestionCitas` para garantizar que solo exista una instancia del sistema en toda la aplicación.
- **Observer:** Implementado en la clase `Notificacion` para notificar automáticamente a los pacientes y doctores cuando se realiza una acción relevante como la cancelación de una cita.
- **Strategy:** Utilizado en la clase `Reporte` para permitir la generación de diferentes tipos de reportes según la estrategia seleccionada.

Conclusión

Este sistema está diseñado para ser escalable y fácil de mantener, con un enfoque en la modularidad y el uso de patrones de diseño para gestionar la complejidad. Cada clase y método está diseñado para cumplir una función específica dentro del sistema, facilitando la gestión de citas médicas de manera eficiente.