Archivo de Documentación de Proyecto_Hospital_Simulador.pdf

Portada

- **Título**:" Simulador de Gestión de Hospital"
- **Descripción**: Proyecto desarrollado en Python utilizando Programación Orientada a Objetos, Programación Funcional, Decoradoresy Concurrencia.
- **Autor**: Joaquin Manuel Lamelas
- **Fecha**: 15/11/2024

Índice

---

1. Introducción

Este proyecto tiene como objetivo simular la gestión de un hospital, permitiendo la admisión de pacientes, asignación de doctores, asignando y seguimiento de tratamientos. Está desarrollado en Python, utilizando conceptos avanzados de programación como P. O. O., Programación Funcional, Decoradores y Concurrencia.

---

2. Funcionamiento del Programa

Este programa incluye varias clases, métodos, y un decorador para automatizar el registro de las acciones clave. Aquí se describe el flujo principal de ejecución y la estructura de las clases.

- **Flujo Principal**: El flujo de ejecución del programa comienza en main.py, donde se crea un hospital, un doctor, y pacientes. Luego, los pacientes son admitidos en el hospital y se asigna un doctor a uno de ellos.
- **Decorador @registrar_historial**: Este decorador, ubicado en decoradores.py, registra automáticamente en el archivo historial.txt las acciones importantes, como la admisión de un paciente o la asignación de un doctor. Esto proporciona trazabilidad sin necesidad de escribir código de registro adicional en cada método.

  A continuación se explica los componentes principales y su papel en el sistema:

- **Clase Persona**: Clase base que representa a cualquier persona en el sistema, con nombre y edad.

- **Clase Paciente**: Hereda de Persona y añade atributos específicos, como condición médica y doctor asignado.
- **Clase Doctor**: Hereda de Persona y gestiona una lista de pacientes, además de una especialidad médica.
- **Clase Hospital**: Contiene la lógica para administrar pacientes y doctores. Permite admitir pacientes, asignar doctores, y ejecutar tratamientos de forma concurrente.
- **Decoradores**: "**registrar_historial"**: al ser aplicado a un método, registra el nombre del método y los argumentos usados en un archivo de texto (historial.txt).

---

3. Retos y Soluciones

Retos y Soluciones:

- **Reto 1**: Manejo de relaciones entre "Doctor" y "Paciente".
    - **Descripción**: Fue necesario crear una relación bidireccional entre Doctor y Paciente, de forma que cada doctor pudiera tener una lista de pacientes y cada paciente pudiera referirse a su doctor.
    - **Solución**: he utilizado un método *"asignar_paciente"* en la clase Doctor que actualiza tanto la lista de pacientes del doctor como el atributo *"doctor_asignado"* en el paciente.
- **Reto 2**: Documentación clara y completa.
    - **Descripción**: La documentación del proyecto debía cumplir con los estándares de PEP8 y proporcionar docstrings en cada clase y método.
    - **Solución**: he intentado hacerlo bien usando docstrings detallados para cada módulo, clase y método, explicando su propósito, parámetros y valores de retorno, aunque para muchos es la parte más fácil, para mi no tanto.

---

4. Conclusión

He querido plasmar los conceptos mas importantes que hemos visto durante estos meses en el bootcamp, con tipo de herramienta, que pueda tener una aplicación práctica concreta, y que además entregue un archivo ".txt", hay cosas por mejorar, y algunas que quisiera tener más pulidas pero he intentado poner lo mejor que podía.-