UGANDA CHRISTIAN UNIVERSITY
A Centre of Excellence in the Heart of Africa

FACULTY OF EN GINEERING, DESIGN AND TECHNOLOGY
DEPARTMENT OF COMPUTING AND TECHNOLOGY

ADVENT 2024 SEMESTER OOP COURSEWORK PROJECT REPORT

PROGRAM: BSCS, BSDS 2:1
COURSE: Object-Oriented Programming

COURSE LECTURER: Ian Raymond Osolo

PROJECT TITLE: *HOSPITAL MANAGEMENT SYSTEM*

Submitted by

S/N	Reg Number	Name	Signature
1.	M23B23/016	YAWE ARTHUR SHALOM	
2.	M23B38/007	AMOIT LYNN	
3.	S23B38/014	NAMBOOZE HELLEN NOELINE	
4.	S23B23/047	OBBA MARK CALVIN	
5.	M23B23/023	LAKICA LETICIA	

Date Submitted: 12/11/2024

1.0 Abstract

Our project a hospital management system developed in Python to address the administrative needs of healthcare facilities. The system is designed to streamline and automate tasks such as patient and doctor record management, appointment scheduling, room allocation and billing. By organizing the elements in a centralized digital system, this solution aims to reduce administrative burdens, enhance data accuracy and facilitate efficient access to critical information.

The system is structured using object-oriented principles, with individual classes for managing patients, doctors, appointments, rooms and bills. cutomtkinter's is employed for GUI.

2.0 Introduction, problem statement, and project objectives

Introduction

The hospital management system presented here is designed to automate the administration and record-keeping of a healthcare facility. With the increase in patient data, doctor records, appointment scheduling, billing and room management requirements, the system engineered aims to streamline operations, minimize human error and make data retrieval efficient. By leveraging Python and customtkinter library for UI elements this system enables healthcare providers to focus more on patient care by providing tools for managing records effortlessly.

Problem Statement

Hospitals require efficient systems for managing diverse data related to patients, doctors, appointments, room allocations and billing. The traditional paper-based systems make it challenging to retrieve, update and analyse data effectively. This often leads to duplicate records, appointment clashes, patient care delays and billing inconsistencies. The lack of integration between patient, doctor, and room records can further complicate hospital operations and lead to inefficiencies.

Project Objectives

Data Management: Develop a unified system that can store and retrieve information on patients, doctors, appointments, rooms and billing.

Record Keeping: Enable easy addition and viewing of patient and doctor records to ensure up-to-date information is always available.

Appointment Scheduling: Allow scheduling of appointments with unique IDs to prevent duplicate or conflicting entries thus preventing overlaps.

Room Allocation: Implement room management for assigning patients to rooms ensuring no double bookings.

Billing System: Integrate billing functions to keep track of charges for each patient, ensuring billing transparency and reducing billing errors.

Enhanced User Interface with Alerts: Using customtkinter's messagebox alerts to inform users of duplicate entries or completion of tasks, providing a user-friendly experience.

3.0 Methods, tools, and designs used for the project

Methods

Object-Oriented Programming (OOP): The system is designed using OOP principles in Python. Key entities such as *Patient*, *Doctor*, *Appointment*, *Room* and *Bill* are implemented as classes. Each class encapsulates relevant attributes and methods providing a modular structure that simplifies maintenance and expansion.

Inheritance and Polymorphism: Inheritance is used in classes like *Patient* and *Doctor* which inherit from a base *Hospital* class, allowing shared methods across classes.

Data Storage (In Dictionary): Each type of record is stored in a dictionary within its respective class. This approach enables rapid lookup and retrieval of data.

Tools

Python Programming Language: The entire system is coded in Python due to its readability, simplicity, and extensive libraries that make it ideal for rapid prototyping.

customtkinter Library: customtkinter is used to create basic user interface elements, especially messagebox, which provides alerts and feedback to the user.

Designs

Modular Class-Based Design: Each primary functionality (e.g., record keeping, appointments) is encapsulated in its own class. This design choice ensures modularity, making it easy to maintain, extend, or update each module individually.

4.0 Results

(Description of the project developed, justifying how it solves the problem and achieves the set objectives.)
The hospital management system developed in this project effectively addresses the need for organized and efficient management of patient records, doctor information, appointments, room allocations, and billing in a healthcare setting. The modular, class-based structure encapsulates each aspect of hospital operations, allowing for easy addition, viewing, and management of records. By implementing object-oriented principles, the system is flexible, scalable, and adaptable to future enhancements.

The project meets the objectives outlined as follows;

The system stores information for patients, doctors, appointments, room allocations and billing within their respective classes, each using Python dictionaries as in-memory data structures. This allows for quick retrieval and organized record-keeping. This approach addresses the problem of fragmented data by keeping all information within one organized framework.

By allowing the easy addition of patient and doctor records via the add_to_system method in each class, the system ensures data consistency. The system also prevents duplicate entries through ID checks, alerting

users if a patient or doctor ID is already present. These features help achieve the goal of reducing errors and maintaining accurate records.

The Appointment class provides an appointment_scheduling method, allowing new appointments to be scheduled while checking for conflicts. If an appointment ID, doctor ID, or date is already scheduled, the system alerts the user, preventing appointment overlaps. This scheduling check ensures efficient resource use and minimizes potential issues from double-booking or scheduling conflicts.

The Room class, through its assign_room method, facilitates the assignment of patients to specific rooms. The system prevents double-occupancy by verifying room availability before assignment, notifying users if a room is already occupied. This functionality meets the objective of efficiently managing room assignments and ensures that each patient has a designated space within the facility.

The Bill class handles billing by associating each patient ID with the services provided and the total charges, using the add_to_records method. It ensures no duplicate billing by checking for existing patient IDs and alerts the user if a bill has already been generated for a particular patient. This feature provides accurate billing and meets the objective of reducing billing discrepancies.

The Bill class handles billing by associating each patient ID with the services provided and the total charges, using the add_to_records method. It ensures no duplicate billing by checking for existing patient IDs and alerts the user if a bill has already been generated for a particular patient. This feature provides accurate billing and meets the objective of reducing billing discrepancies.

5.0 Conclusion & Recommendation

Conclusion

The hospital management system developed in this project provides a structured, efficient and reliable solution for managing critical hospital operations including patient records, doctor information, appointment scheduling, room assignments and billing. By leveraging object-oriented programming principles. This system offers modularity, scalability and ease of maintenance while customtkinter provides the GUI.

Recommendations

Implement Persistent Data Storage;

Currently, the system uses Python dictionaries for in-memory data storage, which is suitable for small-scale applications or as a prototype. For a fully functional hospital management system it is recommended to integrate a database such as MySQL or MongoDB to store data persistently enabling data retrieval across a network.

Add User Authentication and Access Control;

Implementing a login system with access levels (e.g., admin, doctor, receptionist) would ensure that only authorized users can access sensitive data or perform certain actions. This would enhance security and help maintain patient confidentiality.

Automate Notifications and Alerts;

Adding email or SMS notifications to remind patients of appointments or inform doctors of scheduled sessions would improve communication and reduce missed appointments. Alerts could also be implemented for billing and check-out processes.

7.0 Appendices

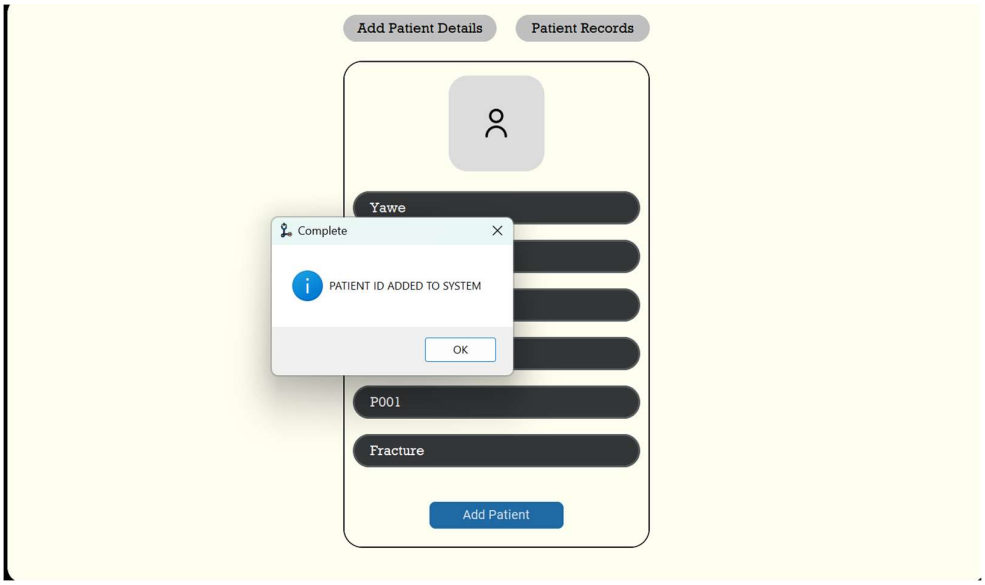
Sample of the source code

```
# Class Hospital----------------------------------------
class Hospital:
    def __init__(self):
        pass
    def add_to_system(self):
        pass
    def displayRecords(self):
        pass
```

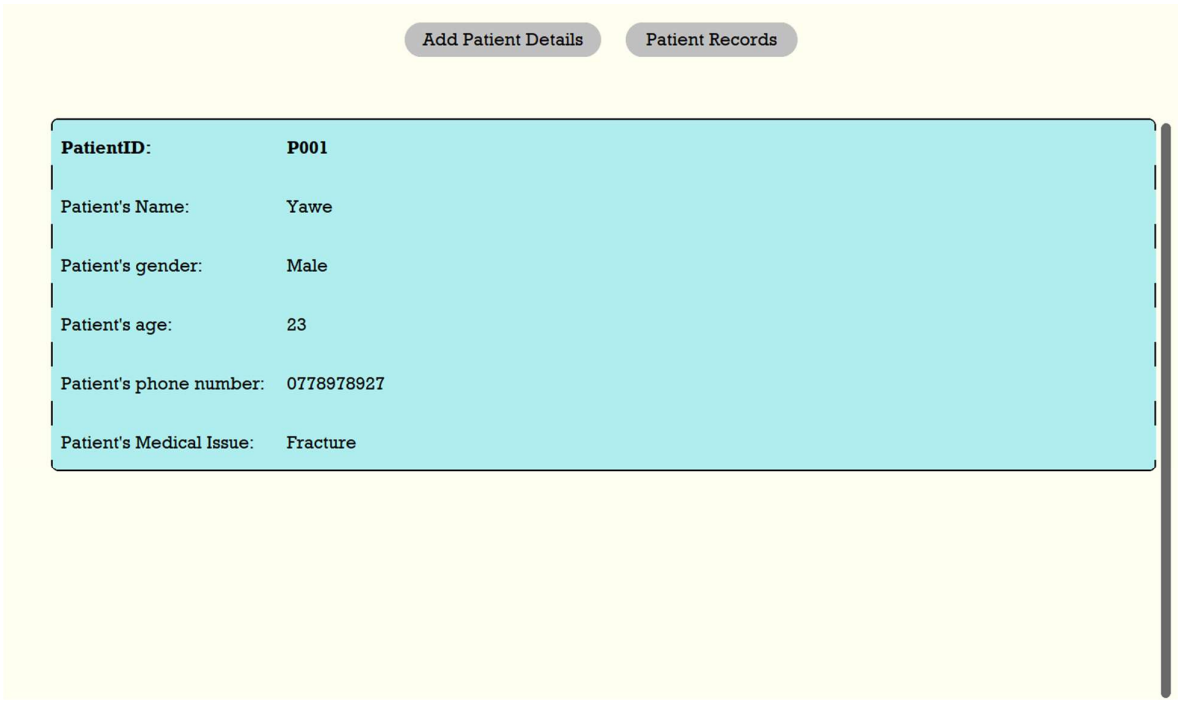
The Hospital class with the add_to_system and displayRecords

```
class Patient(Hospital):
    def __init__(self,name,age,gender,phone_number,PatientID,MedicalIssues):
        self.__PatientID = PatientID
        self.__PatientMedical = MedicalIssues

    @property
    def add_to_system(self):
        if self.__PatientID in self.Patient_records:
            messagebox.showinfo("Alert","PATIENT ID ALREADY IN SYSTEM")
        else:
            self.Patient_records[self.__PatientID] = [self.__name,self.__gender,self.__age,self.__phone_number,self.__PatientMedical]
            messagebox.showinfo("Complete","PATIENT ID ADDED TO SYSTEM")

    @property
    def displayRecords(self):
        for key,value in records.items():
            if len(self.Patient_records) == 0:
                print("\nNo patient records available\n")
                print('----------------------------------------')
            else:
                for key, patient in self.Patient_records.items():
                    print(f'\nPatient ID: {key}')
                    print(f'Patient name: {patient[0]}')
                    print(f'Patient gender: {patient[1]}')
                    print(f'Patient age: {patient[2]}')
                    print(f"Patient Medical Issues: {patient[4]}")
                    print(f'Patient Phone number: {patient[3]}')
                    print('----------------------------------------')
```

The Patient class code snippet showing the inherited methods



The patient page sending a notification to the user after adding a new patient to the system.



The patient details being displayed in the patient records after adding a new patient.