PROJECT DOCUMENTATIN

MEDICAL CLINIC MANAGEMENT SYSTEM

OUR TEAM

1-Hadeer Nady Farghal	2100746
2-Sondos Abdelmohsen Abdelnafea	2101647
3-Mariam Ashraf Awny	2100993
4-Jessica Talaat	2100821
5-Mira Mamdoh Yousef	2101119
6-Hager Ayman Nabeh	2100751

INTRODUCTION

The Medical Clinic Management System is a desktop application designed to streamline the management of patient records, doctor schedules, appointments, and medical histories in a medical clinic. The system aims to improve the efficiency of administrative tasks by automating the handling of data and offering a simple interface for users (patients, doctors, and clinic staff).

The system incorporates design patterns to address key software design challenges such as data consistency, creation of objects, and separation of concerns. The two main design patterns used in this system are the Singleton Pattern and the Factory Pattern.

DESIGN PATTERNS USED:

1. Singleton Pattern

Overview:

The Singleton Pattern ensures that a class has only one instance, providing a global point of access to that instance. It is useful when a class should be instantiated only once and accessed from multiple places. In this project, the Singleton Pattern is applied to manage shared resources (like the patient database and appointment system), ensuring consistency across multiple instances and preventing redundant object creation.

1.1 PATIENT DATABASE MANAGER (SINGLETON)

Class Description:

- The Patient Database Manager class is a Singleton that manages the entire patient database. It ensures that there is only one instance of the database manager, which guarantees consistent and synchronized access to the patient records.
- **Responsibilities**:
    - Handle operations like adding, updating, and retrieving patient records.
    - Ensure that all patient data modifications are consistent across all parts of the system.

JUSTIFICATION FOR SINGLETON PATTERN:

- The Patient Database Manager needs to maintain consistency in data across the entire system, ensuring no conflicting updates. By using the Singleton pattern, we ensure that only one instance of the manager is used, thereby centralizing control over patient data.
- Having a single instance minimizes overhead and prevents issues like multiple conflicting instances accessing or modifying the database concurrently.

1.2 APPOINTMENT SCHEDULING SYSTEM (SINGLETON)

Class Description:

- The Appointment Scheduling System is another Singleton that manages the allocation of appointment slots for patients. It ensures that all users (doctors, receptionists, etc.) are working with the same data, preventing double-booking of appointment slots.
-

Responsibilities:

- Schedule appointments based on available time slots.
- Prevent conflicting appointments by ensuring only one instance can modify the scheduling data at any given time.

JUSTIFICATION FOR SINGLETON PATTERN:

- By applying the Singleton pattern to the Appointment Scheduling System, we ensure that the scheduling data is consistent and controlled. This is critical in avoiding issues such as double-booking, where multiple users try to assign appointments to the same time slot.
- It also simplifies management by ensuring only one instance is responsible for appointment allocation, avoiding potential synchronization issues.

2-FACTORY PATTERN

Overview:

The Factory Pattern provides a way to create objects without specifying the exact class of object that will be created. This is useful when the object creation process is complex or varies depending on the context. In this project, the Factory Pattern is used to handle the creation of different types of medical records and doctor specializations.

2.1 FACTORY FOR MEDICAL RECORDS (FACTORY)

Class Description:

- The Medical Records Factory is responsible for creating different types of medical records, such as Patient History, Prescriptions, and Lab Results.
- **Responsibilities:**
  - Instantiate and return the appropriate medical record object based on the type of record requested.
  - The factory abstracts the creation logic, so users of the system do not need to know how to instantiate these different types of records.

JUSTIFICATION FOR FACTORY PATTERN:

- The Medical Records Factory abstracts the creation of different types of medical records. Depending on the context (e.g., prescription vs. lab results), the factory can instantiate the correct type of object without the caller needing to manage object creation directly.
- This pattern is particularly useful for maintaining the scalability of the system, allowing the addition of new types of medical records in the future without altering existing code that calls the factory.

2.2 FACTORY FOR DOCTOR SPECIALIZATIONS (FACTORY)

Class Description:

- The Doctor Specialization Factory creates doctor objects based on the specialization, such as Cardiologist, Neurologist, or General Practitioner.
- **Responsibilities:**
  - Dynamically instantiate doctors with their specific specialization and properties (e.g., consultation hours, patient type).
  - Encapsulate the logic of creating specialized doctor objects to keep the rest of the system clean and maintainable.

JUSTIFICATION FOR FACTORY PATTERN:

- By applying the Factory pattern to the creation of doctor objects, we ensure that the system can easily create doctors of different specializations without needing to modify the core logic of the system.
- This decouples the logic of doctor instantiation from the rest of the system, making the code more modular and maintainable. If new specializations need to be added in the future, they can be incorporated into the factory without disrupting other parts of the application.

3. ABSTRACT FACTORY PATTERN

Pattern Overview

The Abstract Factory pattern provides an interface for creating families of related or dependent objects without specifying their concrete classes. It involves multiple factories that produce related objects.

Justification:

- The Abstract Factory pattern allows the creation of related UI elements that conform to the look and feel of the operating system. This helps in maintaining a consistent user interface across different environments.

4. BUILDER PATTERN

Pattern Overview

The Builder pattern is used to construct complex objects step by step. It allows for the creation of different representations of a product using the same construction process.

Justification:

- The Builder pattern separates the construction logic of complex objects (like windows) from the code that uses them, allowing for easier customization and future extensions.

5. PROTOTYPE PATTERN

Pattern Overview

The Prototype pattern allows for creating new objects by copying an existing object, known as the prototype. This is useful when object creation is costly, and cloning an existing object is more efficient.

Justification:

- The Prototype pattern is useful for duplicating window configurations quickly without having to reinitialize every property. This improves performance, especially when creating multiple windows with similar properties.

SUMMARY OF PATTERN USAGE AND JUSTIFICATIONS

Pattern	Class	Justification
Singleton	Patient Database Manager	Ensures consistent access to the patient database and prevents issues with concurrent data access, ensuring data integrity across the system.
Singleton	Appointment Scheduling System	Prevents double-booking by ensuring that only one instance handles appointment scheduling, ensuring consistency and centralized control.
Factory	Medical Records Factory	Abstracts the creation of different medical record types, allowing new types to be added without altering the code that uses them.
Factory	Doctor Specialization Factory	Encapsulates the creation logic for different doctor specializations, making it easy to extend the system with new specializations without modifying existing code.

CONCLUSION

The use of the Singleton and Factory design patterns in the Medical Clinic Management System ensures efficient resource management, ease of extension, and clean separation of concerns. By centralizing the management of patient data and appointment scheduling (via Singleton) and abstracting the creation of medical records and doctors (via Factory), the system can easily scale, remain flexible, and maintain high levels of consistency and maintainability.

Thank You