İZMİR EKONOMİ ÜNİVERSİTESİ

SE 311 PROJECT REPORT

PATIENT MONITORING SYSTEM

ARDA OĞULCAN ÜZER

FULYA KORU

SİNAN YAŞBEK

YAĞMUR ZEYNEP TOPRAK

Contents

1. Introduction

In this project, we created a simulation of a patient monitoring system using design patterns. The system represents a hospital where doctors provide medical care to patients. Every morning, the doctors have consultations with their patients and prescribe a series of procedures, which are performed by the nurses in the afternoon. Patients are divided into two groups: Patients with heart problems (cardiology patients) and patients with stomach problems (gastroenterology patients).

For cardiology patients, doctors order a virtual angiography and hema blood test. For patients with gastrointestinal problems, doctors order an MRI and gastro blood test. These procedures are performed by different departments at the hospital. The hospital has only one radiology department and one laboratory department to meet the diagnostic needs of all patients. The radiology department performs the virtual cardiac and gastroscopy procedures, while the laboratory department performs the blood tests. It's important to note that the radiology department doesn't perform blood tests and the laboratory department doesn't perform scans. Some patients may require multiple tests depending on their condition.

If the patient's monitoring device detects a dangerously high blood pressure reading, the nurse is notified immediately and then the physician. The nurse then manually measures the patient's blood pressure to ensure immediate medical attention.

2. Thought Process and Pattern Selection

In developing the patient monitoring system, we carefully considered the system requirements and the potential benefits of using specific design patterns. We selected the following patterns based on their suitability for the system's architecture and functionality:

- Singleton Pattern
- Observer Pattern
- Adapter Pattern
- Iterator Patter
- Factory Pattern

2.1. Singleton Pattern

When we were thinking about how to represent the radiology department and the laboratory department in the hospital system, we realized that we needed a way to ensure that there was only one copy of each department. We wanted to avoid having multiple versions that could cause confusion. After considering several options, we decided to use the Singleton pattern.

The singleton pattern ensures that there is only one instance of a class. So we applied this pattern to the radiology department and the lab department. This means that no

matter how many times we need to use these departments, there will only ever be one instance of either.

To make this work, we made the radiology department and lab department constructors private, meaning that they can only be accessed within the class itself. We have also created a private static variable that contains the single instance of each department. This variable is accessed through a special method called getInstance(). This method checks if an instance of the department already exists and returns it. If not, a new instance is created and returned.

By using the singleton pattern, we ensure that the radiology department and the laboratory department are represented consistently throughout the hospital system. It prevents us from accidentally creating multiple copies and provides a centralized way for other parts of the system to access these departments. This helps us maintain clarity and avoid confusion that could result from multiple versions of the same department.

2.2. Observer Pattern

In studying the task of monitoring patient blood pressure readings, we realized the importance of immediately notifying the nurse and physician when a patient's blood pressure exceeds a certain threshold. To efficiently meet this requirement, we decided to use the Observer pattern.

The Observer pattern establishes a relationship between objects, where changes in one object automatically inform and update other objects. It seemed to be a suitable solution for our situation, as it allowed us to establish a connection between the patient (subject) and the nurse/physician (observer) without the need for constant verification.

To implement the Observer pattern, we first created the necessary classes. The Patient class served as the subject or observable that keeps track of observers and manages the notification process. It contained methods to register and remove observers, as well as to notify when the patient's blood pressure changed. We also added a setChanged() method to indicate that the patient's condition had changed before the observers were notified.

To handle different types of patients, such as those with cardiological or gastroenterological conditions, we created special subclasses such as CardiologicalPatient and GastroenterologicalPatient. These subclasses inherited from the Patient class and allowed us to customize behavior and override methods such as performConsultation() and getProcedureIterator() to meet the specific needs of each patient type.

On the observer side, we introduced the Observer interface, which defined the update() method. The Nurse and Physician classes implemented this method, enabling them to receive notifications from the patient. Upon receiving an update, the nurse and doctor could take appropriate actions based on the new information. This mechanism ensured that they could respond promptly to any changes in the patient's blood pressure.

In summary, our thought process behind applying the Observer pattern revolved around creating an efficient solution to monitor blood pressure levels and notify the nurse and doctor in a timely manner. By designing the Patient, CardiologicalPatient, GastroenterologicalPatient, Nurse, and Physician classes with simplicity in mind, we established a reliable communication system that kept the medical staff informed about critical changes in the patient's condition. This application of the Observer pattern enhanced patient care by enabling quick intervention when necessary.

2.3. Adapter Pattern

As we explored the project requirements, we realized that we needed to incorporate additional functionalities like virtual angiography, MRI scans, and blood tests into our existing system. However, we faced a challenge because the interfaces of our Radiology and Lab units didn't match the specific requirements of the doctors.

To overcome this challenge, we decided to use the Adapter pattern. This pattern allows us to make different components work together smoothly, even if they have different interfaces. By implementing the Adapter pattern, we could create special adapters that act as intermediaries between our system and the radiology and lab departments.

To begin, we designed the adapter classes: RadiologyAdapter and LabAdapter. These classes would enable communication between our system and the respective departments. Although we didn't see the complete code for these adapters, we understood that they would handle translating requests, coordinating with the departments, and returning results in a format our system could understand.

While we couldn't see the specific classes representing the radiology and lab departments, we recognized their importance in our thought process. These classes would provide the functionalities we needed to integrate into our system.

In our planning, we also considered the need for a target interface. This interface would act as a common link between our adapters and the desired functionalities, ensuring smooth integration within our system.

Moving forward, we identified the Nurse and Physician classes as the main users of the adapters. They would utilize the adapters to perform tasks related to tests and consultations. Through the adapters, these users could access functionalities like virtual angiography, MRI scans, and blood tests seamlessly within our system.

While our thought process provided an understanding of the Adapter pattern and its potential benefits, we acknowledged that we needed more information and implementation details to fully evaluate its effectiveness in our project. However, we remained confident that incorporating the Adapter pattern would enable us to integrate the desired functionalities efficiently, improving the workflow for doctors and medical staff.

2.4. Iterator Pattern

As we explored ways to effectively handle the ordered tests for patients with cardiology and gastroenterology conditions, we brainstormed different approaches.

We wanted a solution that would allow us to easily go through these tests without revealing their internal complexity. That's when we realized that the Iterator pattern could be a great fit for our needs.

First, we needed to define an interface that would act as our Iterator. We called it ProcedureIterator, which would help us navigate through the ordered tests. This interface provided methods to move through the test elements smoothly.

Next, we created the Concrete Iterator, which implemented the ProcedureIterator interface. We named it ProcedureIteratorImpl to clearly indicate its purpose. This class contained the logic to iterate over the ordered tests, enabling us to access them one by one in a clear and organized manner.

To establish the connection between the Iterator and the test objects, we designed the Aggregate interface. We decided to use the Patient abstract class as our Aggregate since we were dealing with patients and their tests. This interface-like class defined the contract for creating the Iterator object, allowing us to obtain an instance of ProcedureIterator.

Finally, we focused on the Concrete Aggregate, which required specific implementations for cardiology and gastroenterology patients. We created the CardiologicalPatient and GastroenterologicalPatient classes as Concrete Aggregates. These classes not only stored patient-specific information but also fulfilled the requirements of the Aggregate interface by creating the appropriate Concrete Iterator object.

By adopting the Iterator pattern and utilizing the ProcedureIterator, ProcedureIteratorImpl, Patient, CardiologicalPatient, and GastroenterologicalPatient classes, we successfully developed a solution to navigate through the ordered tests for patients with different conditions. This approach enabled us to iterate over the tests in a systematic and controlled manner, ensuring efficient access to the relevant information.

2.5. Factory Pattern

When designing our patient consultation system, we needed a way to efficiently create consultations for patients with cardiological and gastroenterological problems. We recognized that the Factory Method pattern could be a suitable solution. This pattern allows us to define an interface for creating objects while delegating the responsibility of class instantiation to subclasses. This flexibility appealed to us, as it would enable us to tailor consultations based on specific patient types.

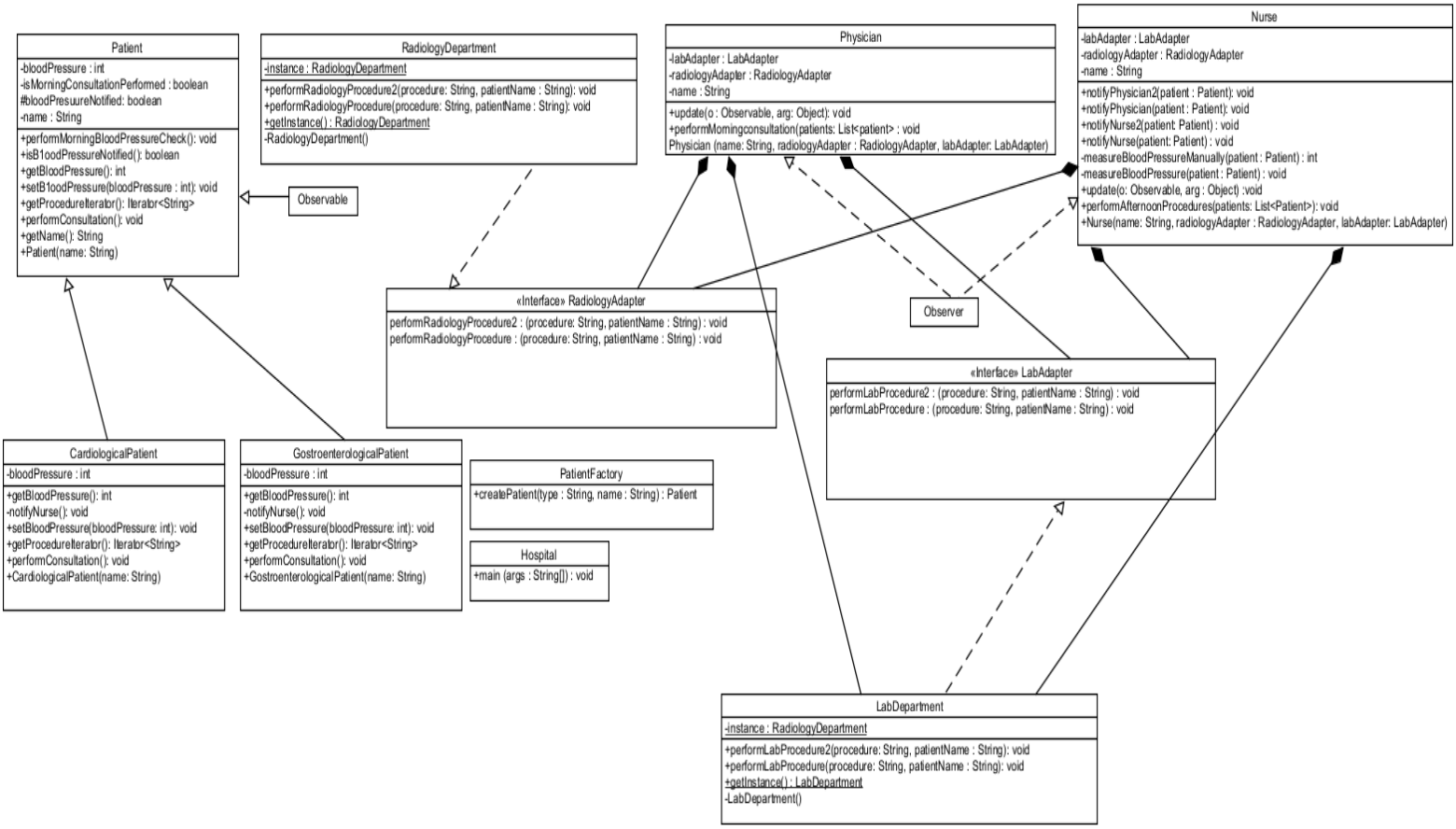
To begin, we identified the Patient class as our product within the factory pattern. This class would serve as the base, containing shared attributes and behaviors for all patients. We then proceeded to create ConcreteProduct classes, such as CardiologicalPatient and GastroenterologicalPatient, which would represent the distinct implementations of the Patient class. These concrete products would possess

specialized characteristics and behaviors tailored to cardiological and gastroenterological patients respectively.

Next, we shifted our focus to the creator role in the pattern. We introduced the PatientFactory class to encapsulate the creation logic for different patient types. This class became the central entity responsible for generating instances of the appropriate patient subclass based on input parameters. To achieve this, we implemented a static method called createPatient within the PatientFactory class. This method would receive the patient type and name as arguments and return an instance of the corresponding patient subclass.

By implementing the Factory Method pattern in this manner, we achieved an efficient and flexible system for creating patient consultations. The PatientFactory class acted as the creator, while the Patient class served as the product, and CardiologicalPatient and GastroenterologicalPatient became the concrete products. This pattern allowed us to streamline the consultation creation process, ensuring that patients with different medical conditions received appropriate and specialized care.

UML DIAGRAM



4. EXPLANATION OF THE CLASSES AND THE METHODS IN THE CLASSES

The given code represents a simulation of a Hospital system with various classes. Here's a breakdown of each class and their key methods:

1) Hospital class:
    - main(String[] args): This is the entry point of the program. It creates instances of RadiologyAdapter, LabAdapter, Nurse, and Physician objects. It also creates a list of patients using the PatientFactory. It simulates blood pressure changes for patients and then calls methods to perform morning consultations and afternoon procedures.

2) Patient (abstract class):
    - getName(): Returns the name of the patient.
    - performConsultation(): Abstract method to perform a consultation for the patient.

- getProcedureIterator(): Abstract method to get an iterator for the patient's procedures.
- setBloodPressure(int bloodPressure): Sets the patient's blood pressure and notifies observers if it is above the dangerous level.
- getBloodPressure(): Returns the patient's blood pressure.
- isBloodPressureNotified(): Checks if the patient's blood pressure has been notified.
- performMorningBloodPressureCheck(): Method to perform a morning blood pressure check (empty implementation).

3) Cardiological Patient (subclass of Patient):
    - perform Consultation(): Overrides the abstract method from Patient to perform a cardiological consultation for the patient.
    - get ProcedureIterator(): Overrides the abstract method from Patient to return an iterator for cardiological procedures.
    - set Blood Pressure(and blood Pressure): Overrides the method from Patient to set the patient's blood pressure and notify observers, including nurses.
    - notify Nurse(): Private method to notify the nurse (observer).
    - get Blood Pressure(): Returns the patient's blood pressure.

4) Gastroenterological Patient (subclass of Patient):
    - perform Consultation(): Overrides the abstract method from Patient to perform a gastroenterological consultation for the patient.
    - get ProcedureIterator(): Overrides the abstract method from Patient to return an iterator for gastroenterological procedures.
    - set Blood Pressure(and blood Pressure): Overrides the method from Patient to set the patient's blood pressure and notify observers, including nurses.
    - notify Nurse(): Private method to notify the nurse (observer).
    - get Blood Pressure(): Returns the patient's blood pressure.
5) Radiology Department (singleton class implementing Radiology Adapter):
    - getInstance(): Returns the singleton instance of the Radiology Department.
    - perform Radiology Procedure(String procedure, String patientName): Implements the method from Radiology Adapter to perform a radiology procedure.
    - perform Radiology Procedure 2(String procedure, String patientName): Additional method specific to the RadiologyDepartment class.
6) Lab Department (singleton class implementing Lab Adapter):
    - getInstance(): Returns the singleton instance of the Lab Department.

- perform Lab Procedure(String procedure, String patientName): Implements the method from Lab Adapter to perform a lab procedure.
- perform Lab Procedure 2(String procedure, String patientName): Additional method specific to the Lab Department class.

7) Radiology Adapter (interface):
    - perform Radiology Procedure(String procedure, String patientName): Abstract method to perform a radiology procedure.
    - perform Radiology Procedure 2(String procedure, String patientName): Additional abstract method.
8) Lab Adapter (interface):
    - perform Lab Procedure(String procedure, String

4.Screen Dump

```
Physician: Dr. Arda Oğulcan Üzer will perform a consultation for patient: Sinan Yaşbek
Physician: Dr. Arda Oğulcan Üzer will perform a consultation for patient: Belinay Koru
Nurse Yağmur Zeynep Toprak will take care of patient: Belinay Koru and prescribe certain procedure in the afternoon.
Physician will be notified about high blood pressure for patient: Belinay Koru if require.
```

```
  --------PERFORMING MORNING PROCEDURES FOR PATIENTS BY PHYSICIAN---------

Cardiological consultation for patient: Fulya Koru
Nurse will take Virtual angiography the test for patient: Fulya Koru (Radiology Department)


Nurse will take Hema blood test the test for patient: Fulya Koru (Lab Department)


Cardiological consultation for patient: Sinan Yaşbek
Nurse will take Virtual angiography the test for patient: Sinan Yaşbek (Radiology Department)


Nurse will take Hema blood test the test for patient: Sinan Yaşbek (Lab Department)


Gastroenterological consultation for patient: Belinay Koru
Nurse will take MRI the test for patient: Belinay Koru (Radiology Department)


Nurse will take Gastro blood test the test for patient: Belinay Koru (Lab Department)



ALL MORNING CONSULTATION FOR PATIENTS PERFORMED BY PHYSICIANS!
```

```
----------PERFORMING AFTERNOON PROCEDURES FOR PATIENTS BY NURSE---------

Cardiological consultation for patient: Fulya Koru

Nurse will take Virtual angiography the test for patient: Fulya Koru (Radiology Department)

Nurse performed Virtual angiography the test for patient: Fulya Koru (Radiology Department)

Nurse will take Hema blood test the test for patient: Fulya Koru (Lab Department)

Blood pressure for patient Fulya Koru is: 120
Nurse performed Hema blood test the test for patient: Fulya Koru (Lab Department)

Cardiological consultation for patient: Sinan Yaşbek

Nurse will take Virtual angiography the test for patient: Sinan Yaşbek (Radiology Department)

Nurse performed Virtual angiography the test for patient: Sinan Yaşbek (Radiology Department)

Nurse will take Hema blood test the test for patient: Sinan Yaşbek (Lab Department)
```

```
Blood pressure for patient Sinan Yaşbek is: 181
Nurse Yağmur Zeynep Toprak has been notified for the patient Sinan Yaşbek
Nurse will measure the blood pressure manually because it's critical.
Physician will be notified about high blood pressure for patient: Sinan Yaşbek
Enter the blood pressure for patient Sinan Yaşbek: 190
Blood pressure is still high.
Dear patient: Sinan Yaşbek. Your blood pressure is a bit high. Do not worry. Our nurse will take care of you.
Manually measured blood pressure for patient Sinan Yaşbek is: 190
Nurse performed Hema blood test the test for patient: Sinan Yaşbek (Lab Department)

Gastroenterological consultation for patient: Belinay Koru
Nurse will take MRI the test for patient: Belinay Koru (Radiology Department)

Nurse performed MRI the test for patient: Belinay Koru (Radiology Department)

Nurse will take Gastro blood test the test for patient: Belinay Koru (Lab Department)

Blood pressure for patient Belinay Koru is: 200
Nurse Yağmur Zeynep Toprak has been notified for the patient Belinay Koru
Nurse will measure the blood pressure manually because it's critical.
Physician will be notified about high blood pressure for patient: Belinay Koru
Enter the blood pressure for patient Belinay Koru: 150
Manually measured blood pressure is within the safe range.
Dear patient: Belinay Koru. Your blood pressure is normal. Have a good day.
Manually measured blood pressure for patient Belinay Koru is: 150
Nurse performed Gastro blood test the test for patient: Belinay Koru (Lab Department)


ALL AFTERNOON PROCEDURES FOR PATIENTS PERFORMED BY NURSES!
**********GET WELL SOON**********
```