객체지향개발론및실습

Laboratory 2. Observer 패턴: 그룹채팅 프로그램

1. 목적

- 1대다 관계를 형성해주며, 한 객체의 상태에 관심을 가지고 있는 모든 객체에게 객체의 상태가 변하면 항상 자동으로 통보를 해주는 관찰자 패턴을 실습해 본다.

2. 개요

- 카카오톡과 같은 메신저 프로그램은 보통 그룹채팅 기능을 제공한다. 그룹채팅은 직관적으로 생각했을 때 관찰자 패턴으로 구현하기 매우 적합한 프로그램이다. 그룹에 새로운 메시지가 도착하면 참여하고 있는 모든 사용자에게 해당 메시지를 전달해야 한다.
- 이 실습에서는 카카오톡과 같은 메신저 프로그램을 실제 구현하지는 않는다. 특히, 통신 프로그래밍을 하지 않는다. 하지만 메신저 프로그램의 그룹채팅 부분만 관찰자 패턴을 이용하여 간단하게 시뮬레이션 형태로 구현해봄으로써 그룹채팅이 실제 관찰자 패턴에 적합한지 살펴본다.
- 시각적 효과를 위해 JavaFX, PyQt6, Qt6를 사용한다.

3. KoreatechTalk

3.1 설계

- **`ChatRoom`**, **`User`**
  - 그룹채팅 프로그램을 생각해보면 가장 쉽게 떠오르는 객체는 채팅방과 사용자이다.
  - 각 사용자와 채팅방은 id와 방이름으로 식별한다고 가정한다.
  - 채팅방에는 여러 사용자가 참여하고 있을 수 있고, 한 사용자는 여러 채팅방에 참여하고 있을 수 있다.
  - 채팅방에 새 글이 게시되면 해당 방의 모든 사용자에게 알려야 한다. 이 부분을 관찰자 패턴을 이용해 구현해 본다.
  - 사용자는 항상 온라인 상태가 아닐 수 있다. 이 부분도 고려하여 구현한다.
  - 실제 메신저 프로그램의 경우 채팅방과 사용자 객체는 서버와 클라이언트에 각각 별도 정의하여 사용하겠지만 이 실습에서는 각 객체가 실제 존재해야 하는 위치는 고려하지 않고 각 하나씩 만들어 구현한다.
  - 이론 수업에서 살펴본 관찰자 패턴 예제에서 관찰 대상 객체는 관찰자 객체를 유지한다. 이 실습에서는 **`ChatRoom`** 객체를 관찰 대상, **`User`** 객체를 관찰자로 모델링한다, 하지만 **`ChatRoom`** 객체는 **`User`** 객체의 목록을 유지하지 않고 사용자의 ID만 유지하고, 절대 **`User`** 객체와 직접 상호작용하지 않는다.
- **`ChatMessage`**: 채팅방을 통해 교환하는 메시지의 전송자와 내용을 유지한다. 이 실습에서는 텍스트 메시지만 교환할 수 있다고 가정한다.
  - **`ChatRoom`**은 이 방에서 교환된 모든 메시지를 유지해야 하며, 각 사용자도 이 방을 통해 받은 모든 메시지를 유지해야 한다. 채팅방은 교환된 모든 메시지를 유지해야 하지만 사용자는 자신이 가입된 시점부터 자신이 최근에 받은 메시지까지만 유지한다. 따라서 채팅방과 사용자가 유지하는 정보가 다를 수 있다.
  - 채팅방과 각 사용자가 유지하는 메시지 목록은 **`ChatMessage`** 타입의 리스트 자료구조에 유지한다.

  - 실제 메신저 프로그램에서는 교환되는 메시지를 유지하는 기간이 정해져 있을 수 있지만 이 실습에서는 이 부분은 고려하지 않는다.

- **ChatServer**
  - 통신 서버와 데이터베이스 서버를 대신하는 객체이다.
  - 통신 서버: 사용자는 이 서버를 이용하여 메시지를 전송하고, 채팅방은 이 서버를 이용하여 메시지가 도착하였을 때 이 방의 소속된 사용자에게 메시지를 전달한다.
    - 일반적으로 채팅 서버는 인라인(inline) 방식으로 통신한다. 모든 메시지는 일차적으로 채팅 서버에 전달되며, 채팅 서버는 이 메시지를 목적 대상 사용자에게 전달한다. 즉, 사용자 간의 직접 통신을 하지 않는다.
  - 데이터베이스 서버: 개설된 모든 채팅방과 모든 사용자를 유지한다. 실제 모든 채팅방 객체와 사용자 객체를 유지한다. 데이터베이스 서버라 보기 어려운 측면이 있지만...

- 통신 흐름: 사용자가 사용자 뷰(**UserChatWindow**)를 이용하여 특정 채팅방을 선택하고 메시지를 전송할 수 있다. 메시지의 전송은 **ChatServer**의 **sendMessage** 메소드를 활용한다. 이 메소드는 메시지를 **ChatRoom**에 등록한다. **ChatRoom**은 관찰자 패턴의 관찰 대상이 되어, 이 방에 가입된 모든 사용자에게 **updateUsers**를 이용하여 새 메시지를 전달한다. 이 메소드는 내부적으로 **ChatServer**의 **forwardMessage**를 이용한다.

- 전략 패턴을 포함하여 특정 패턴을 사용할 경우 추상화와 DIP 원칙에 충실히 구현하기 위해 상위 타입이나 **interface**를 많이 활용한다. 관찰자 패턴은 관찰 대상(subject) **interface**와 관찰자(observer) **interface**를 정의하고, 관찰자 대상 역할을 할 구체적 클래스나 관찰자 역할을 할 구체적 클래스는 이들을 구현하도록 한다. 이 실습에서는 이와 같은 상위 타입의 **interface**를 사용하지 않고 구현한다. 만약 관찰자 대상의 종류가 다양하고 관찰자도 다양하면 이 추상화는 반드시 필요하다.

- GUI와 관련되어 있는 **KoreaTechChatTalk**와 **ChatWindow**는 이 문서의 후반부에 중요 사항만 설명한다.

3.2 User 클래스

- 사용자 관련 정보를 유지하는 클래스
- 멤버변수
  - **String userID**: 사용자 ID
  - **boolean online**: 사용자의 온라인 여부
  - **Map<String, List<ChatMessage>> roomLogs**: 사용자가 가입한 각 채팅방의 대화목록 유지
  - **ChatWindow chatWindow**: 이 사용자의 GUI 윈도우 (메신저 대화방과 유사한 윈도우)
- 메소드
  - **public boolean setOnline(boolean)**: **online** 값을 주어진 인자로 바꿈
  - **public void joinRoom(String roomName)**: 주어진 이름의 채팅방을 위한 **ChatRoomLog**을 생성하고 **roomLogs**에 추가한다.
  - **public void leaveRoom(String roomName)**: 주어진 이름의 채팅방을 **roomLogs**에서 제거한다.
  - **public void clearRoom(String roomName)**: 주어진 이름의 채팅방의 모든 대화를 삭제한다. 이것은 사용자가 유지하는 **roomLog**에만 반영한다.
  - **public List<ChatMessage> getRoomLog(String roomName)**: 주어진 이름의 채팅방의 대화 목록을 반환한다.
  - **public void update(String roomName, ChatMessage message)**: **ChatRoom**에 새 메시지가 생기면 모든 참여 사용자에게 그 사실을 알리기 위해 사용한다. 관찰자 패턴에서 관찰자가 구현해야 하는 **update** 메소드에 해당한다.

3.3 ChatRoom 클래스

- 채팅방 관련 정보를 유지하는 클래스
- 멤버변수
  - **`String roomName`**: 채팅방 이름
  - **`List<ChatMessage> roomLog`**: 이 채팅방의 대화 목록
  - **`Map<String, Integer> userList`**: 가입된 사용자 정보를 유지한다. 사용자 **`id`**가 키가 되고, 최종적으로 받은 메시지 색인을 값으로 유지한다.
- 메소드
  - **`public void addUser(String userID)`**: 주어진 **`userID`**에 해당하는 사용자의 가입을 처리한다. **`userList`**에 **`userID`**가 키가 되고, 이 채팅방의 마지막 메시지의 색인이 값이 되도록 추가한다. (관찰자 패턴의 addObserver 해당)
  - **`public void deleteUser(String userID)`**: 주어진 **`userID`**에 해당하는 사용자의 탈퇴를 처리한다. (관찰자 패턴의 deleteObserver 해당)
  - **`public void newMessage(ChatMessage message)`**: 채팅 서버가 새 메시지를 수신하면 이 메소드를 이용하여 채팅방에 새 메시지를 전달한다. 이 메소드는 **`roomLog`**에 새 메시지를 추가하고, **`updateUsers`** 메소드를 이용하여 모든 사용자에게 메시지를 전달해야 한다. **이 실습에서는 메시지를 작성한 사용자에게도 메시지를 보냄**
  - **`public void updateUsers()`**: 채팅방의 새 메시지를 채팅방의 모든 사용자에게 전달한다. 이 메소드는 내부적으로 **`updateUser`**를 활용한다. (관찰자 패턴의 notifyObservers 해당)
  - **`public void updateUser(String userID)`**: **`userID`**에 해당하는 사용자에게 아직 받지 못한 새 메시지를 전달한다. 이때 **`ChatServer`**의 **`forwardMessage`** 활용한다. 현재 온라인 상태인 사용자에게만 전달해야 하며, 이 부분은 **`ChatServer`**가 처리한다.

3.4 ChatMessage 클래스

- 채팅방에서 교환되는 메시지 관련 정보를 유지하는 클래스
- 멤버변수
  - **`String userID`**: 메시지의 전송자 ID
  - **`String content`**: 메시지의 내용에 해당하는 문자열
- 메소드: 단순 getter만 존재한다. (불변 클래스임. 자바는 이 때문에 자바16부터 확정되어 제공하는 **`record`**를 이용하여 정의함)

3.5 ChatServer 클래스

- 이 응용에서 통신 서버, 데이터베이스 서버 역할을 하는 클래스이다.
- 싱글톤 클래스로 구현한다. **`ChatServer`** 클래스는 오직 하나의 객체만 생성해서 사용한다.
- 이 객체는 **`ChatServer`** 내에서 생성하며,이 객체는 getServer **`static`** 메소드를 이용하여 어디서나 접근할 수 있다.
- 멤버변수
  - **`Map<String, ChatRoom> chatRooms`**: 채팅방 데이터베이스 (ChatRoom 객체 유지)
  - **`Map<String, User> users`**: 사용자 데이터베이스 (User 객체 유지)
- 메소드
  - **`void addUser(User), Collection<User> getUsers()`**: 사용자를 데이터베이스에 추가하는 메소드와 데이터베이스에 있는 모든 사용자를 얻는 메소드이다.

- **`void addRoom(String roomName)`**: 새 방을 데이터베이스에 추가하는 메소드이다.
- **`public void addUserToRoom(String userID, String roomName)`**: 주어진 사용자를 주어진 채팅방에 가입시킨다.
- **`public void deleteUserFromRoom(String userID, String roomName)`**: 주어진 사용자를 주어진 채팅방에 탈퇴시킨다.
- **`public void sendMessage(String roomName, ChatMessage message)`**: 사용자가 새 메시지를 특정 채팅방에 보낼 때 사용하는 메소드이다. (통신 서버 모방 메소드)
- **`public void forwardMessage(`**
  `String destID, String roomName, ChatMessage message)`:
  특정 메시지를 특정 채팅방에 소속되어 있는 사용자에게 전달한다. 사용자가 온라인일 경우에만 실제 전송이 이루어진다. (통신 서버 모방 메소드)
- **`public void requestMessages(String receiverID, String roomName)`**: 사용자가 오프라인에서 온라인이 되었을 때 오프라인 동안 받지 못한 메시지를 요구할 때 사용하는 메소드이다.

4. GUI

- **`ChatWindow`**에서 수신된 메시지를 보여주기 위해 **`ListView`**를 사용한다. 또 자신이 보낸 메시지와 수신한 메시지를 구분하기 위해 자신이 보낸 메시지는 노란색 바탕으로 보여준다. C++와 파이썬은 **`QListView`**를 사용한다.
- **`ListView`**를 이용한 이유는 과제로 진행할 개별 메시지 삭제 기능을 위해 개별 메시지를 선택할 수 있도록 하기 위함이다. 개별 메시지를 선택하면 **`messageSelected`** 메소드가 호출되며, 이 메소드에서는 선택한 행 정보를 기록하여 필요할 때 활용한다.

5. 실습

- el에서 관련 프로젝트를 다운받아 주어진 프로그램을 분석한 후, 이 문서에 있는 자료를 토대로 구현되어 있지 않는 부분을 완성하시오.
- **`ChatRoom`**의 **`addUser`**와 **`updateUser`** 메소드를 완성해야 함
- **`ChatRoom`**의 **`newMessage`**, **`ChatServer`**의 **`addUserToRoom`**와 **`deleteUserFromRoom`**에 필요한 예외처리를 추가하시오.

6. 숙제

- 다음 질문들에 대한 답변을 작성하여 **`ChatRoom`** 클래스 프로그램 주석에 포함하시오.
  1. 이 실습에서는 관찰자 패턴을 사용하여 구현하고 있다. 특히, 채팅방을 관찰 대상, 채팅방의 참여자를 관찰자로 모델링하였다. 관찰자 패턴은 크게 push 또는 pull 방법으로 구현할 수 있다. 현재 제시된 소스는 어느 방식으로 구현되어 있는지 설명하시오.
  2. 관찰자인 참여자는 하나만 관찰하는 형태가 아니다. 여러 개의 채팅방을 관찰하고 있을 수 있다. 종류는 같지만 관찰 대상이 복수일 수 있는 구조이다. 관찰자는 관찰 대상을 어떻게 구분하는지 간단히 설명하시오.
  3. 실제 코드를 보면 이 두 객체 외에 관찰 대상과 관찰자로 모델링하는 것이 있다. 그것이 무엇인지 제시하시오.
- 개별 메시지를 선택하여 삭제하는 기능을 추가하시오. 이미 채팅방에 있는 모든 메시지를 삭제하는 기능은 구현되어 있다. 하지만 이 기능은 사용자가 유지하는 정보에서만 삭제하는 것이고 서버가 유지하는 정보에는 영향을 주지 않는다. 여기서 추가하는 개별 메시지의 삭제 기능은 서버가 유지하는 정보에 영향을 주어야 한다. 물론 이미 방에 소속된 참여자가 해당 메시지를 수신하였다면 해당 사용자에게는 영향이 없지만 오프라인 상태인 참여자의 경우에는 나중에 온라인 상태가 되었을 때 삭제된 메시지는 받을 수 없다. 자신이 전송한 메시지만 서버가 유지하는 정보에 영향을 주며, 수신한 메시지의 삭제는 사용자가 유지하는 정보에만 영향을

준다. 메시지의 삭제는 메시지를 선택한 후에 메뉴에서 메시지 삭제를 선택하면 된다. 보통 메신저는 파업 메뉴를 이용하지만 파업 메뉴까지 올바르게 구현하기 어렵기 때문에 이 과제에서는 이 방식을 사용한다.

- 현재 채팅방은 각 사용자마다 그 사용자가 최종 수신한 메시지의 색인 정보를 유지하고 있다.
- 이 때문에 채팅방이 유지하는 메시지 목록에서 실제 메시지를 삭제하면 각 사용자가 유지하는 색인 정보가 맞지 않게 된다. 어떻게 처리해야 할까?
- 또 하나 메시지 삭제에서 고려해야 하는 것은 사용자가 유지하는 목록에서 메시지의 색인과 채팅방이 유지하는 목록에서 메시지의 색인이 다르다는 것이다.
- 사용자가 삭제를 선택하면 **ChatWindow**의 **deleteMessage**를 호출하게 되며, 이 메시지에서는 삭제할 메시지가 본인이 전송한 메시지인지 수신한 메시지인지 구분해야 한다. 본인이 작성한 메시지이면 서버의 **deleteMessage**를 통해 실제 채팅방이 유지하는 메시지 목록에서도 삭제를 위한 작업을 해야 한다. 이 작업은 채팅방의 **deleteMessage**를 통해 이루어진다. 모든 경우 **User** 클래스의 **deleteMessage**를 이용해 사용자가 유지하는 메시지 목록에서 해당 메시지를 삭제해야 한다.