CSE 351 DESIGN PATTERNS TERM PROJECT REPORT

Berkay Acar - 20190808036
Ali Çolak - 20190808064

STATEMENT OF WORK

We have an application called community. The purpose of this application is to create certain communities and organize events. For example, there may be a community named Software community and this community can organize an event called Object Oriented principle. All communities can organize events. Our users can become members of communities and participate in events. In addition, our users can create their own communities. They can then create events under this community. But there are some requirements that we want our app to meet. First, we want to generate an object at runtime from the app class where the main processes of our application are executed, and then we want everything that wants to access that class to be accessed through this object. In short, we want to produce at most one object from this class and provide a global access point to the created object.

STATEMENT OF WORK -continued

Another is to send notifications to users' notifications box in some cases. For example, if the user is a member of a community, if that community adds any new event, the user will be aware of it. In this way, we plan to increase the rate of participation in events. Another is to ensure that users are aware of the cancellation of events that users have already participated in. In this way, no one will be victimized in the event of a possible cancellation. Finally, our users must log in to the application in order to see their notifications. Users who forget to log in to the application or users who do not look at the message box even though they are logged in may not see these notifications. In order to minimize such communication problems, we want to offer notification options on platforms such as e-mail, whatsapp, telegram, which users use more in their daily lives. If the option to receive a notification is added or removed later, users will be notified.

WHICH DESIGN PATTERNS DID WE CHOOSE ?

- Singleton Design Pattern - with eager instantiation
- Observer Design Pattern
- Decorator Design Pattern

WHY DO WE USE SINGLETON DESIGN PATTERN ?

We used a singleton because we want to generate at most one object from the App class and provide a global access point to this object. In addition, we want an object to be created from the App class the first time the application is run. So we created an object as soon as the application was run with eager instantiation. With eager instantiation, the JVM guarantees that the instance will be created before any thread can access the static App class variable.

WHY DO WE USE OBSERVER DESIGN PATTERN ?

As it is known, in our application, users could join the communities. It was necessary to send a notification to all community members to ensure more participation every time there was an event. Likewise, a message had to be sent to all participants when the event was cancelled. For a more user-friendly application, the community owner should not take any action in these cases. We solved this problem by using the observer design pattern, since in these cases notifications should be sent to the relevant people automatically.

WHY DO WE USE DECORATOR DESIGN PATTERN ?

Our application's notification system was sometimes missing. In addition to the notification system of the application in Runtime, we needed to add additional notification features such as whatsapp and email. There could be some platforms that could be popular in the coming years. Considering these, we may need to add these applications to the notification options when necessary. We used the decorator design pattern to add additional notification features to our application's notification system at runtime and to easily implement new notification features that may come to our application later on.

SINGLETON DESIGN PATTERN

At runtime, an object is created directly from the App and Admin classes. Those who want to access these classes can access it through that object.

```
public class Client {

    46 usages
    public static App app = App.getInstance();
```

```
public class Admin implements Subject {

    1 usage
    private static final Admin admin = new Admin(); // Eager instantiation
```

```
public class App {


    15 usages
    private static final Admin admin = Admin.getInstance();

    1 usage
    private final static App app = new App(); // Eager instantiation
```

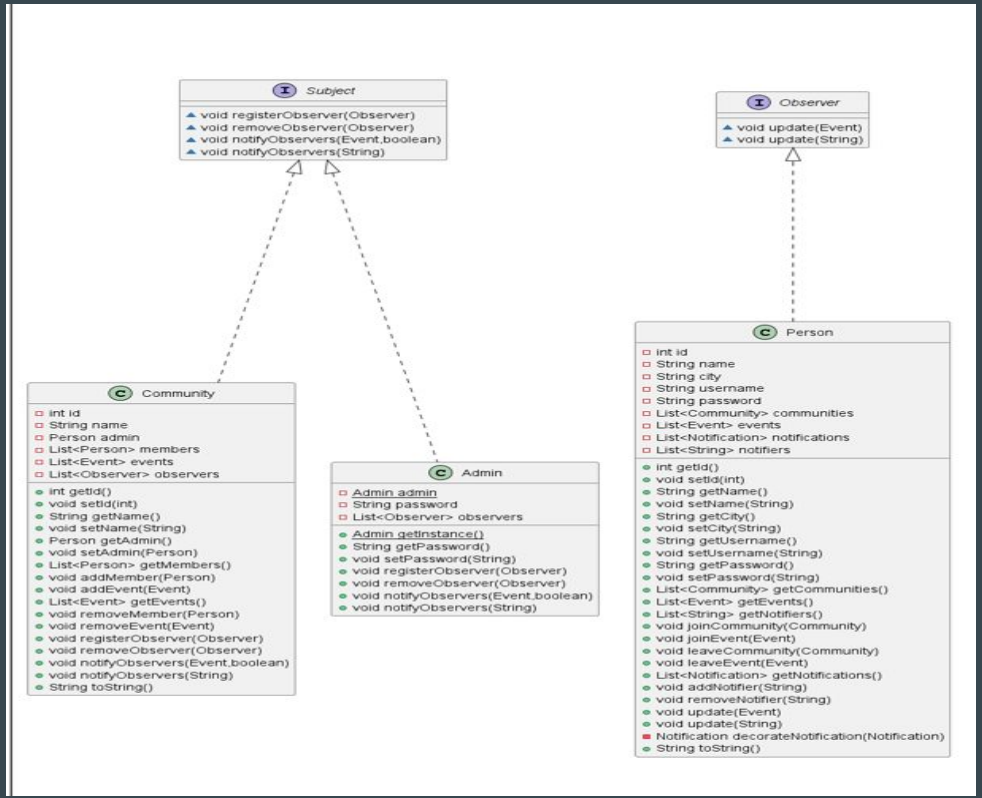
OBSERVER DESIGN PATTERN

We have 2 concrete subject class.

(Community and Admin)

We have one concrete observer class

(Person).



OBSERVER DESIGN PATTERN

Person becomes a member of the communities. Communities inform the person of any developments.

Person becomes an observer for the admin class as soon as he registers to the system. admin Notifies the person of any notifier additions or deletions.

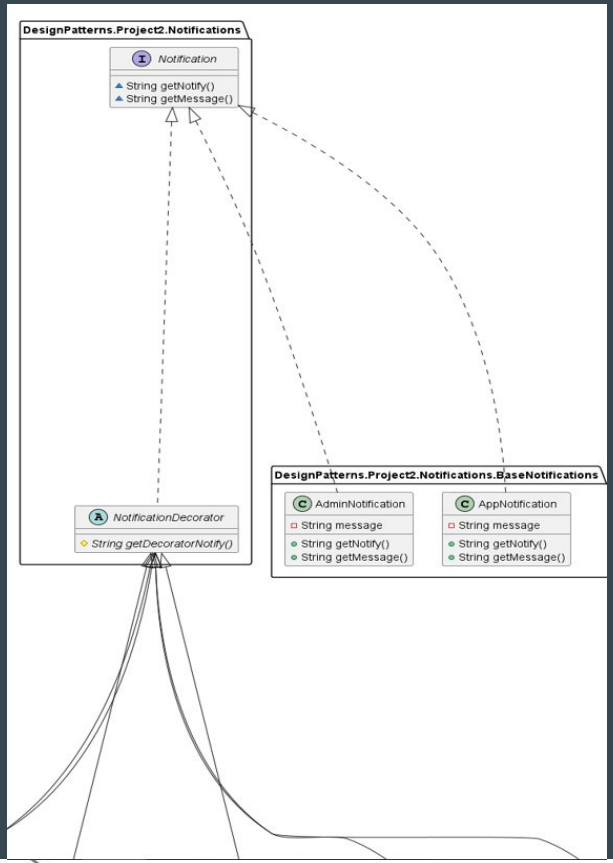
```
public class Admin implements Subject {

    1 usage
    private static final Admin admin = new Admin(); // Eager instantiation
    4 usages
    private final List<Observer> observers;
```

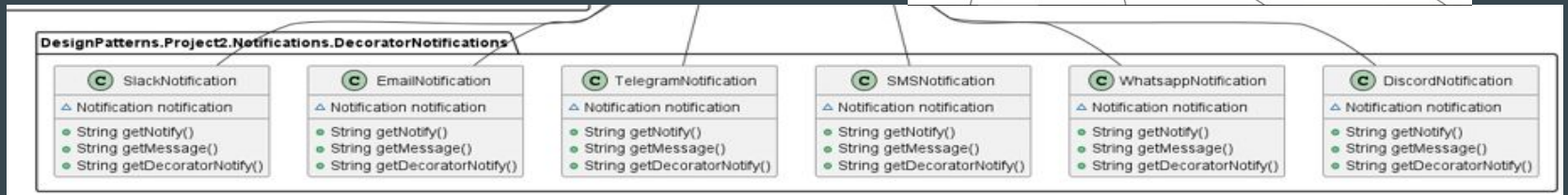
```
public class Community implements Subject {

    4 usages
    private final List<Observer> observers;
```

```
public class Person implements Observer {

    4 usages
    private final List<Community> communities;
```

DECORATOR DESIGN PATTERN

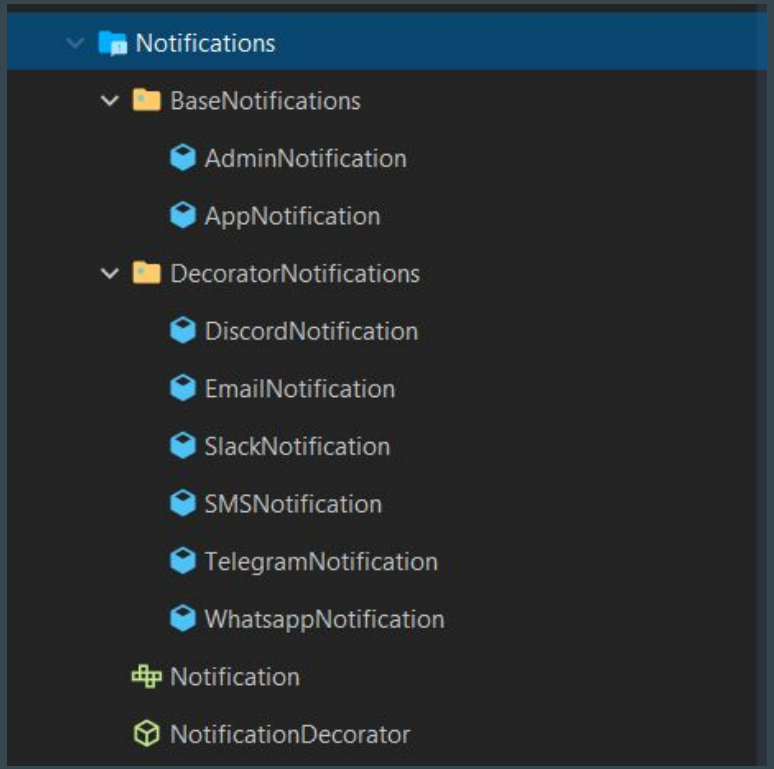
- Notification → Component
- Notification Decorator → Decorator
- AdminNotification → Concrete component
- App Notification → Concrete component
- Other notification class→ Concrete decorators

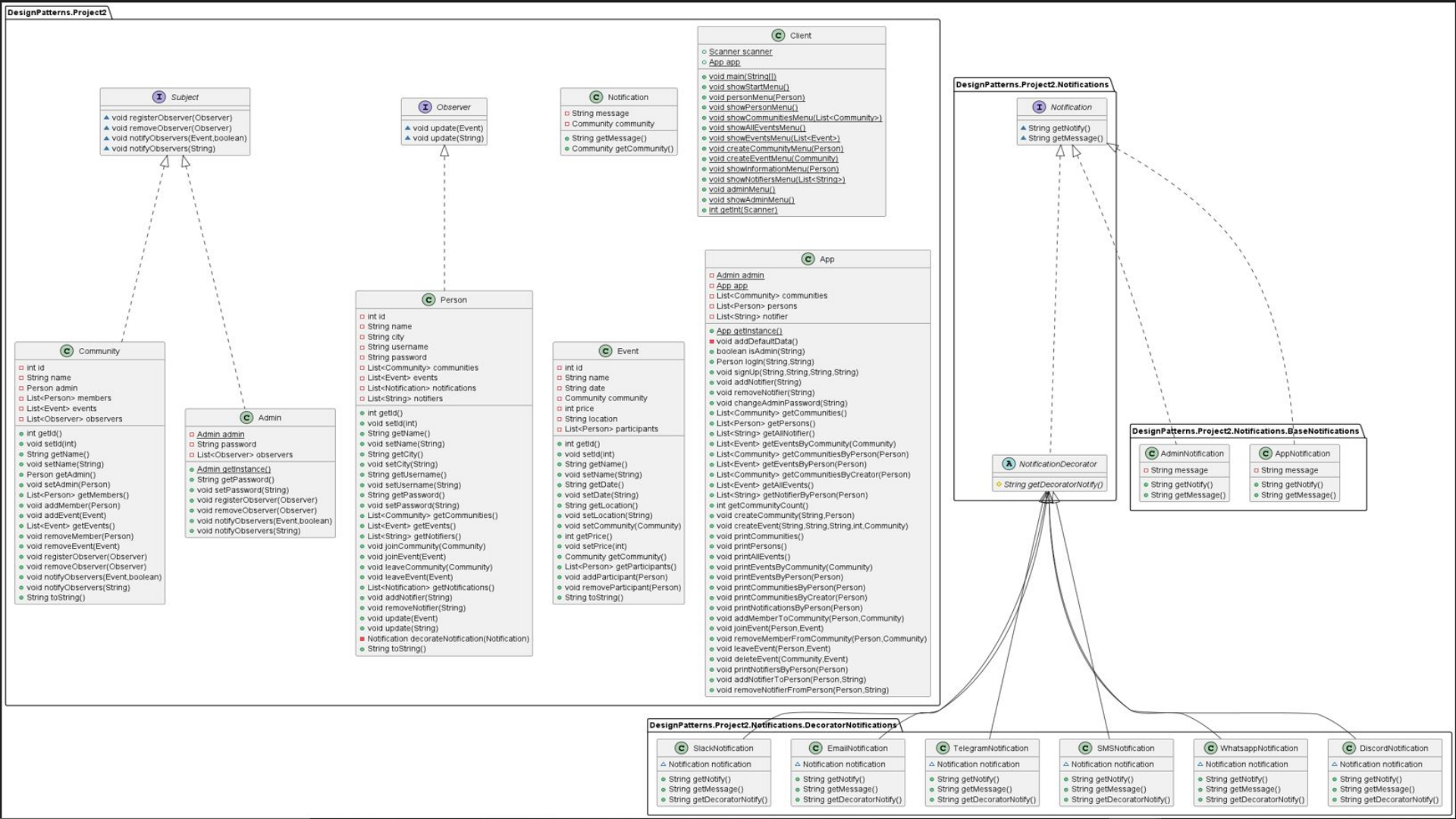




DECORATOR DESIGN PATTERN

We have 2 base notifications. Admin and app. Some additions such as whatsapp, telegram can be made to these notifications at runtime. In this way, while receiving both admin and app notifications, notifications can be received from another source at the same time.




DesignPatterns.Project2
Subject
void registerObserver(Observer)
void removeObserver(Observer)
void notifyObservers(Event,boolean)
void notifyObservers(String)
Observer
void update(Event)
void update(String)
Notification
String message
Community community
String getMessage()
Community getCommunity()
Client
Scanner scanner
App app
void main(String[])
void showStartMenu()
void personMenu(Person)
void showPersonMenu()
void showCommunitiesMenu(List<Community>)
void showAllEventsMenu()
void showEventsMenu(List<Event>)
void createCommunityMenu(Person)
void createEventMenu(Community)
void showInformationMenu(Person)
void showNotifiersMenu(List<String>)
void adminMenu()
void showAdminMenu()
int getInt(Scanner)
Community
int id
String name
Person admin
List<Person> members
List<Event> events
List<Observer> observers
int getId()
void setId(int)
String getName()
void setName(String)
Person getAdmin()
void setAdmin(Person)
List<Person> getMembers()
void addMember(Person)
void addEvent(Event)
List<Event> getEvents()
void removeMember(Person)
void removeEvent(Event)
void registerObserver(Observer)
void removeObserver(Observer)
void notifyObservers(Event,boolean)
void notifyObservers(String)
String toString()
Admin
Admin admin
String password
List<Observer> observers
Admin getInstance()
String getPassword()
void setPassword(String)
void registerObserver(Observer)
void removeObserver(Observer)
void notifyObservers(Event,boolean)
void notifyObservers(String)
Person
int id
String name
String city
String username
String password
List<Community> communities
List<Event> events
List<Notification> notifications
List<String> notifiers
int getId()
void setId(int)
String getName()
void setName(String)
String getCity()
void setCity(String)
String getUsername()
void setUsername(String)
String getPassword()
void setPassword(String)
List<Community> getCommunities()
List<Event> getEvents()
List<String> getNotifiers()
void joinCommunity(Community)
void joinEvent(Event)
void leaveCommunity(Community)
void leaveEvent(Event)
List<Notification> getNotifications()
void addNotifier(String)
void removeNotifier(String)
void update(Event)
void update(String)
Notification decorateNotification(Notification)
String toString()
Event
int id
String name
String date
Community community
int price
String location
List<Person> participants
int getId()
void setId(int)
String getName()
void setName(String)
String getDate()
void setDate(String)
String getLocation()
void setLocation(String)
void setCommunity(Community)
int getPrice()
void setPrice(int)
Community getCommunity()
List<Person> getParticipants()
void addParticipant(Person)
void removeParticipant(Person)
String toString()
App
Admin admin
App app
List<Community> communities
List<Person> persons
List<String> notifier
App getInstance()
void addDefaultData()
boolean isAdmin(String)
Person login(String,String)
void signUp(String,String,String,String)
void addNotifier(String)
void removeNotifier(String)
void changeAdminPassword(String)
List<Community> getCommunities()
List<Person> getPersons()
List<String> getAllNotifier()
List<Event> getEventsByCommunity(Community)
List<Community> getCommunitiesByPerson(Person)
List<Event> getEventsByPerson(Person)
List<Community> getCommunitiesByCreator(Person)
List<Event> getAllEvents()
List<String> getNotifierByPerson(Person)
int getCommunityCount()
void createCommunity(String,Person)
void createEvent(String,String,String,int,Community)
void printCommunities()
void printPersons()
void printAllEvents()
void printEventsByCommunity(Community)
void printEventsByPerson(Person)
void printCommunitiesByPerson(Person)
void printCommunitiesByCreator(Person)
void printNotificationsByPerson(Person)
void addMemberToCommunity(Person,Community)
void joinEvent(Person,Event)
void removeMemberFromCommunity(Person,Community)
void leaveEvent(Person,Event)
void deleteEvent(Community,Event)
void printNotifiersByPerson(Person)
void addNotifierToPerson(Person,String)
void removeNotifierFromPerson(Person,String)
DesignPatterns.Project2.Notifications
Notification
String getNotify()
String getMessage()
NotificationDecorator
String getDecoratorNotify()
DesignPatterns.Project2.Notifications.BaseNotifications
AdminNotification
String message
String getNotify()
String getMessage()
AppNotification
String message
String getNotify()
String getMessage()
DesignPatterns.Project2.Notifications.DecoratorNotifications
SlackNotification
Notification notification
String getNotify()
String getMessage()
String getDecoratorNotify()
EmailNotification
Notification notification
String getNotify()
String getMessage()
String getDecoratorNotify()
TelegramNotification
Notification notification
String getNotify()
String getMessage()
String getDecoratorNotify()
SMSNotification
Notification notification
String getNotify()
String getMessage()
String getDecoratorNotify()
WhatsappNotification
Notification notification
String getNotify()
String getMessage()
String getDecoratorNotify()
DiscordNotification
Notification notification
String getNotify()
String getMessage()
String getDecoratorNotify()