

**EGE UNIVERSITY FACULTY OF ENGINEERING**

**COMPUTER ENGINEERING DEPARTMENT**

## 2023–2024 SPRING SEMESTER

**ADVANCED OBJECT ORIENTED PROGRAMMING**

**PROJECT REPORT TEMPLATE DELIVERY DATE**

23/05/2024

## PREPARED BY

05210000261, Bahrihan Torpil

05210000292, Beyza Çelik

05210000260, Kutlu Çağan Akın

05210000238, Mehmet Ali Avcı

CONTENTS

Giriş 3

Paneller 4

Kodun İşleyişi 5

User Class 7

Post Class 9

Group Class………………………………………………………………………………10

FriendManagerClass……………………………………………………………………11

Patterns…………………………………………………………………………………12

UML Diyagram………………………………………………………………………..14

# GİRİŞ

BBCM Sosyal Ağ projesi, kullanıcıların arkadaşlarıyla sosyal ağ arayüzü üzerinden etkileşimde bulunabilecekleri bir platform oluşturmayı amaçlamaktadır. Bu platform, kullanıcıların duvarlarında gönderi paylaşmalarına, arkadaş ekleyip kaldırmalarına ve arkadaşlarıyla gruplar içinde etkileşimde bulunmaları için gruplar oluşturmalarına olanak tanır. Projenin hedefi, kullanıcıların sosyal bağlantılarını ve aktivitelerini sorunsuz bir şekilde yönetebilecekleri etkileşimli ve kullanıcı dostu bir ortam sağlamaktır.

Bu proje, grafik kullanıcı arayüzü için Java Swing kullanarak sağlam ve duyarlı bir tasarım sunar. Kullanıcı kimlik doğrulama, gönderi paylaşma ve grup yönetimi gibi işlevsellikleri uygulayarak, BBCM Sosyal Ağ, popüler sosyal ağ sitelerinin temel özelliklerini taklit etmeyi amaçlamaktadır. Uygulama, kullanıcı verilerini dosyalara okuyup yazarak Java'da temel dosya işlemlerini göstermektedir.

Genel olarak, BBCM Sosyal Ağ projesi, bir Java tabanlı uygulamada kullanıcı arayüzü tasarımı, dosya yönetimi ve etkileşimli özelliklerin entegrasyonuna kapsamlı bir örnek olarak hizmet vermektedir.

# PANELLER

**GİRİŞ VE KAYIR PANELİ**

Kullanıcılar uygulamayı açtıklarında karşılarına gelen ilk ekran giriş ve kayıt panelidir. Bu panelde kullanıcılar sisteme giriş yapabilir veya yeni bir kullanıcı olarak kayıt olabilirler.

1)Giriş Yapma:

-Kullanıcı adı ve şifre girilerek giriş yapılır.

-Kullanıcı adı veya şifre yanlış ya da boş ise hata mesajı verir.

2)Kayıt Olma:

-Kullanıcı adı ve şifre girilerek yeni kullanıcı olarak kayıt olunabilir.

-Aynı kullanıcı adıyla tekrar kayıt olunamaz. Bu durumda hata mesajı verir.

-Yeni kullanıcı bilgileri dosyaya kaydedilir.

**ANA PANEL**

Başarılı bir girişin ardından kullanıcı ana panele yönlendirilir. Ana panelde kullanıcı, arkadaşlarını, gruplarını ve duvarındaki gönderileri görebilir.

1)Arkadaşlar ve Gruplar:

-Sol tarafta arkadaşlar listelenir. Kullanıcı arkadaş ekleyip çıkarabilir.

-Sol alt tarafta gruplar listelenir. Kullanıcı grup oluşturabilir, gruplara arkadaş ekleyip çıkarabilir ve grupları görüntüleyebilir.

2)Duvar:

-Merkezde kullanıcı ve arkadaşlarının gönderileri görüntülenir. Kullanıcı yeni gönderi ekleyebilir.

3)Kontrol Paneli:

-Arkadaş ekleme ve çıkarma butonları bulunur.

-Grup oluşturma butonu bulunur.

-Kullanıcının çıkış yapmasını sağlayan buton bulunur.

# 3-KODUN İŞLEYİŞİ

**Giriş ve Kayıt İşlemleri:**

**-**HandleLogin();

Kullanıcı adı ve şifreyi doğrular. Hatalı ya da boş ise hata mesajı verir. Kullanıcı adı ve şifre doğru girildiğinde ana panele yönlendirir.

-HandleRegister();

Yeni kullanıcı kaydı oluşturur. Hatalı veya boş girişte hata mesajı verir.

**Kullanıcı Bilgilerini Dosyadan Okuma ve Kaydetme**

-AddToMapRegisteredUsers();

Kayıtlı kullanıcıyı dosyadan okur belleğe yükler. Dosya boşşsa kullanıcıyı bilgilendirir.

-saveUserToFile();

Yeni kullanıcı kayıtlarını dosyaya yazar. Kullanıcı adı ve şifre boşsa hata mesajı verir.

**Gönderi Paylaşımı**

-handlePost();

Kullanıcının yeni gönderi paylaşmasını sağlar. Gönderi paylaştıktan sonar duvarı günceller.

-updateWall();

Kullanıcının ve arkadaşlarının gönderilerini sıralı şekilde duvarda gösterir.

**Arkadaş İşlemleri**

**-**handleAddFriend();

Kullanıcıya yeni arkadaş ekler ve dosyaya kaydeder. Kullanıcı kendini ekleyemez. Arkadaş zaten ekliyse hata mesajı verir.

-handleRemoveFriend();

Kullanıcını arkadaşını siler. Arkadaşlık dosyadan silinir. Çıkarıldığına dair mesaj verir.

-updateFriendsList();

Arkadaş listesini günceller ve kullanıcıya gösterir. Kullanıcını arkadaş listesindeki her bir arkadaş ismini ekrana yansıtır.

**Grup İşlemleri**

-handleCreateGroup();

Yeni grup oluşturur. Grup adı boş ise grup oluşturmaz. Grup oluşturulduktan sonar kullanıcıyı bilgilendirir.

-handleAddFriendtoGroup();

Belirtilen gruba belirtilen arkadaşı ekler. Kullanıcı ya da grup yoks ahata mesajı verir.

-handleRemoveFriendFromGroup();

Belirtilen gruptan arkadaş çıkarır. Grup veya kullanıcı bulunmazsa hata mesajı verir.

-handleViewGroup();

Belirtilen grubun üyelerini grup içi mesajlaşmaları gösterir.

# 4-USER CLASS

1. **Constructor (User(String username, String password))**
   * Kullanıcı adı ve şifre alarak bir kullanıcı nesnesi oluşturur.
   * Kullanıcı adı ve şifre ile ilgili alanları başlatır.
   * Duvar, arkadaşlar, gruplar, arkadaş gönderileri ve gözlemciler listelerini başlatır.
2. **post(String content)**
   * Kullanıcının duvarına yeni bir gönderi ekler.
   * Gönderiyi duvara ekler.
   * Arkadaşlarına bildirim gönderir.
3. **isFriend(User user)**
   * Verilen kullanıcının bu kullanıcının arkadaşı olup olmadığını kontrol eder.
4. **addFriend(User user)**
   * Yeni bir arkadaşı ekler.
   * Gözlemci olarak ekler.
   * İlgili kullanıcıya bildirim gönderir.
5. **removeFriend(User user)**
   * Bir arkadaşı listeden çıkarır.
   * Gözlemciyi kaldırır.
6. **addGroup(Group group)**
   * Kullanıcıyı bir gruba ekler.
7. **getGroupByName(String name)**
   * İsim verilen bir gruba erişmek için kullanılır.
8. **displayGroupMembers(String groupName)**
   * Belirtilen grubun üyelerini görüntüler.
9. **update(Post post)**
   * Yeni bir gönderi eklendiğinde gözlemciyi günceller.
   * Kullanıcıya bildirim gönderir.
10. **addObserver(Observer observer)**
    * Gözlemciyi listeye ekler.
11. **removeObserver(Observer observer)**
    * Gözlemciyi listeden çıkarır.
12. **notifyObservers(Post post)**
    * Gözlemcilere bildirim gönderir.
13. **wallIterator()**
    * Kullanıcının duvarındaki gönderileri (post'ları) üzerinde döngü oluşturmak için bir iterator döndürür.
14. **friendsPostsIterator()**
    * Kullanıcının arkadaşlarının gönderileri üzerinde döngü oluşturmak için bir iterator döndürür.

# 5-POST CLASS

1. **Constructor (Post(String username, String content))**
   * Kullanıcı adı ve içerik alarak bir gönderi nesnesi oluşturur.
   * Gönderinin kullanıcı adı, içeriği ve zaman damgasını başlatır.
2. **getUsername()**
   * Gönderiyi oluşturan kullanıcının kullanıcı adını döndürür.
3. **getContent()**
   * Gönderinin içeriğini döndürür.
4. **getTimestamp()**
   * Gönderinin zaman damgasını (oluşturulma zamanını) döndürür.

6-GROUP CLASS

1. **Constructor (Group(String name, User creator))**
   * Bir grup oluştururken grup adını ve oluşturucuyu alır.
   * Grubun adını ve üyeler listesini başlatır ve oluşturucuyu üyelere otomatik olarak ekler.
2. **getName()**
   * Grubun adını döndürür.
3. **getMembers()**
   * Grubun üyelerini içeren bir liste döndürür.
4. **addMember(User user)**
   * Belirtilen kullanıcıyı gruba ekler.
   * Eğer kullanıcı zaten grupta değilse, kullanıcıyı gruba ekler.
5. **removeMember(User user)**
   * Belirtilen kullanıcıyı gruptan kaldırır.
6. **displayMembers()**
   * Grubun üyelerini içeren bir metin dizisi oluşturur.
   * Her bir üyenin kullanıcı adını içeren bir metin dizesi oluşturur.
   * Bu metin dizisini grup adı ile birleştirerek geri döndürür.

FRİENDMANAGER CLASS

1. **Constructor (FriendManager(String filename))**
   * Dosya adını alarak bir **FriendManager** örneği oluşturur.
   * **friendsMap** adında bir **Map** oluşturur ve dosyadan arkadaş ilişkilerini yükler.
2. **getInstance(String filename)**
   * Tekil bir **FriendManager** örneği döndürür.
   * Eğer örnek daha önce oluşturulmamışsa, yeni bir örnek oluşturur ve dosya adını kullanır.
3. **loadFriendsFromFile()**
   * Dosyadan arkadaş ilişkilerini yükler.
   * Her satırı **:** ile bölerek kullanıcı adını ve arkadaşlarını ayırır.
4. **addFriend(String user, String friend)**
   * Belirtilen kullanıcıya ve arkadaşına birbirlerini ekler.
   * Arkadaşlık ilişkilerini dosyaya kaydeder.
5. **removeFriend(String user, String friend)**
   * Belirtilen kullanıcı ve arkadaş arasındaki arkadaşlık ilişkisini kaldırır.
   * Değişiklikleri dosyaya kaydeder.
6. **getFriends(String user)**
   * Belirtilen kullanıcının arkadaşlarını döndürür.
   * Eğer kullanıcı listede yoksa, boş bir küme döndürür.
7. **saveFriendsToFile()**
   * Mevcut arkadaşlık ilişkilerini dosyaya kaydeder.
   * Her bir kullanıcı ve onun arkadaşlarını **:** ile ayırarak dosyaya yazar.
8. **isFriend(String username, String friendUsername)**
   * Belirtilen kullanıcının belirtilen kişinin arkadaşı olup olmadığını kontrol eder.
   * Kullanıcı ve arkadaşı **friendsMap** içinde bulunuyorsa ve arkadaş listesi bu kişiyi içeriyorsa **true** döndürür, aksi takdirde **false** döndürür

PATTERNS

FACTORY PATTERN

public class UserFactory {

public static User createUser(String username, String password) {

return new User(username, password);

}

}

UserFactory sınıfı, yeni kullanıcı nesneleri oluşturmak için bir fabrika yöntemi sağlar. Kodun daha okunabilir, daha bakımı kolay ve daha esnek olmasını sağlar.

SINGLETON PATTERN

public static FriendManager getInstance(String filename) {

if (instance == null) {

synchronized (lock) {

if (instance == null) {

instance = new FriendManager(filename);

}

}

}

return instance;

}

Singleton deseni, bir sınıfın yalnızca bir örneğini oluşturmayı ve bu örneğe genel bir erişim noktası sağlamayı amaçlar.

Bu metodun amacı, FriendManager sınıfının yalnızca bir örneğini oluşturmak ve bu örneği geri döndürmek için kullanılır. Eğer instance henüz oluşturulmamışsa, instance'ı oluşturur. Eğer instance zaten varsa, mevcut olanı döndürür.

ADAPTER PATTERN

Adapter tasarım desenini uygular. Adapter deseni, bir sınıfın arayüzünü, başka bir arayüze dönüştürmeyi sağlar. Bu sayede, uyumsuz iki sınıfın birlikte çalışmasını sağlar.

import java.util.ArrayList;

import java.util.List;

class GroupAdapter implements Chat {

private Group group;

private List<String> messages;

public GroupAdapter(Group group) {

this.group = group;

this.messages = new ArrayList<>();

}

@Override

public void sendMessage(String message) {

messages.add(message);

// İsteğe bağlı: Mesajları tüm üyelere gönderme işlemi

for (User member : group.getMembers()) {

// Mesaj gönderme mantığı

}

}

@Override

public void displayMessages() {

for (String message : messages) {

System.out.println(message);

}

}

}

Group sınıfı, Chat arayüzüyle uyumlu hale gelir ve Chat arayüzünü kullanan diğer bileşenlerle entegre edilebilir hale gelir. Bu da kodun daha esnek ve yeniden kullanılabilir olmasını sağlar.

OBSERVER PATTERN

bir nesnenin durumu değiştiğinde ve diğer nesnelerin bu değişiklikten etkilendiği durumlarda kullanılır.

public interface Subject {

void addObserver(Observer observer);

void removeObserver(Observer observer);

void notifyObservers(Post post);

}

public interface Observer {

void update(Post post);

}

Subject (Konu): Gözlemlenecek olan nesnedir. Bu nesne, bir veya daha fazla gözlemciyi kaydedebilir, kaldırabilir ve onlara durum değişikliklerini bildirebilir.

Observer (Gözlemci): Konuyu izleyen ve durum değişikliklerine yanıt olarak güncellenen nesnelerdir. Her gözlemci, konudan gelen güncellemeleri işlemek için bir update() yöntemine sahiptir.

UML Diyagram

