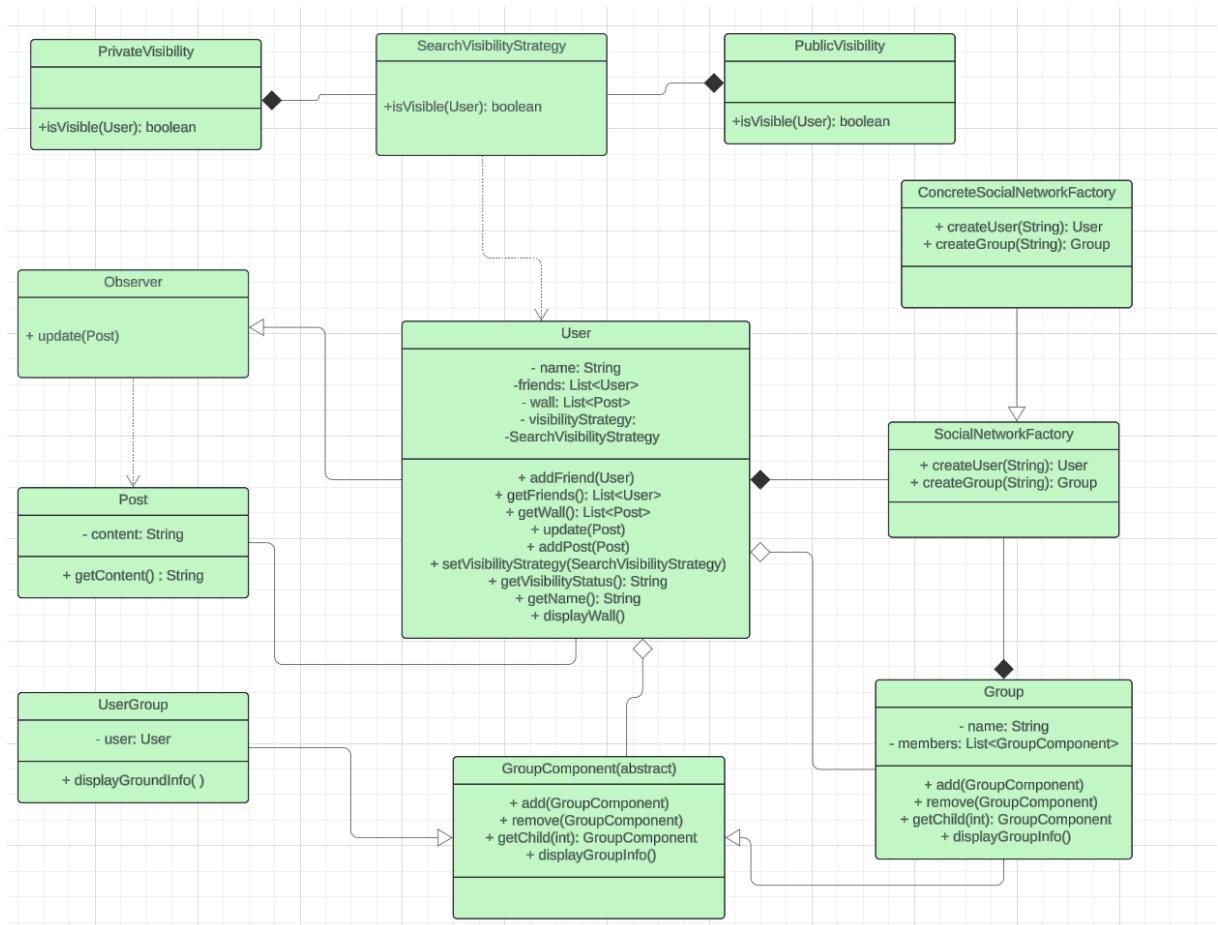


UML CLASS DIAGRAM



DESIGN PATTERNS

1. Observer Pattern (Gözlemci Deseni):

Amaç: Kullanıcıların duvarlarına yeni bir post eklendiğinde bu değişikliklerden haberdar olmalarını sağlamak.

Sağladıkları: Observer deseni, kullanıcıların postlarını gözlemleyen ve bu postların değişikliklerinden haberdar olan bir mekanizma sağlar. Bu sayede, bir kullanıcının duvarına yeni bir post eklendiğinde, diğer kullanıcılar bu değişikliklerden otomatik olarak haberdar olabilirler.

2.Factory Method Pattern (Fabrika Metodu Deseni):

Amaç: Somut nesnelerin oluşturulmasını alt sınıflara bırakarak, nesne oluşturma sürecini genişletilebilir hale getirmek.

Sağladıkları: Factory Method deseni, farklı tipteki nesnelerin oluşturulmasını genişletilebilir hale getirir. Bu projede ConcreteSocialNetworkFactory sınıfı, SocialNetworkFactory soyut sınıfını genişleterek somut bir sosyal ağ fabrikası sağlar. Bu, kullanıcı ve grup nesnelerinin oluşturulması için somut bir yöntem sağlar ve gelecekte yeni nesnelerin eklenmesini kolaylaştırır.

3.Strategy Pattern (Strateji Deseni):

Amaç: Belirli bir algoritmanın davranışını bir dizi farklı algoritma ile değiştirilebilir hale getirmek ve böylece algoritmayı kullanıcılardan bağımsız hale getirmek.

Sağladıkları: Strategy deseni, arama sorgularında kullanılacak görünürlük stratejisini tanımlar. Bu sayede, farklı görünürlük kurallarını temsil eden farklı

stratejiler tanımlanabilir ve bu stratejileri uygulayan nesneler (örneğin PublicVisibility) kullanılabilir. Bu desen, görünürlük kurallarının değişebilirliğini ve esnekliğini sağlar.

4.Composite Pattern (Bileşik Desen):

Amaç: Nesnelerin hiyerarşik bir yapıda oluşturulmasını ve bu nesneler arasında parçaların ve bütünü tek bir arayüz üzerinden kullanılmasını sağlamak.

Sağladıkları: Composite deseni, ağaç benzeri bir yapıda nesneleri organize eder. Bu desen, parçaları ve bütünü temsil eden nesneler arasında aynı arayüzü kullanarak işlem yapılmasını sağlar. Bu projede, GroupComponent soyut sınıfı, gruplar içindeki kullanıcıları ve grupları temsil eder. Group sınıfı, bu soyut sınıfından türetilir ve hem kullanıcıları hem de alt grupları içerebilir. Böylece, herhangi bir düğümün, hem parça hem de bütün olarak işlev görebilmesi sağlanır. Bu desen, gruplar içindeki kullanıcılar ve gruplar arasında hiyerarşik bir ilişki kurulmasını ve bu yapının tek bir arayüz üzerinden kullanılmasını sağlar. Bu, kodun daha esnek olmasını sağlar, çünkü istemci kodu, parçaları ve bütünü işlemek için aynı arayüzü kullanabilir ve böylece grupların yapısı değiştiğinde kodda minimal değişiklik yapılmasını sağlar.

SOURCE CODES AND OUTPUT EXAMPLE

```
public abstract class GroupComponent {  
    public void add(GroupComponent groupComponent) {  
        throw new UnsupportedOperationException();  
    }  
    public void remove(GroupComponent groupComponent) {  
        throw new UnsupportedOperationException();  
    }  
    public GroupComponent getChild(int i) {  
        throw new UnsupportedOperationException();  
    }  
    public abstract void displayGroupInfo();  
}
```

```
public class UserGroup extends GroupComponent {  
    private final User user;  
  
    public UserGroup(User user) {  
        this.user = user;  
    }  
  
    @Override  
    public void displayGroupInfo() {  
        System.out.println("User: " + user.getName());  
    }  
}
```

```

public class Main {
    public static void main(String[] args) {
        SocialNetworkFactory factory = new ConcreteSocialNetworkFactory();

        User berat = factory.createUser(name: "Berat");
        User can = factory.createUser(name: "Can");

        can.setVisibilityStrategy(new PrivateVisibility());
        berat.setVisibilityStrategy(new PublicVisibility());

        berat.addFriend(user: can);

        Post post = new Post(content: "Hello, this is Berat's first post!");
        berat.addPost(post);

        Group group = factory.createGroup(name: "Family");
        group.add(new UserGroup(user: berat));
        group.add(new UserGroup(user: can));

        group.displayGroupInfo();

        // Displaying walls
        berat.displayWall();
        can.displayWall();

        // Checking visibility
        System.out.println(x: "Visibility Check:");
        System.out.println("Alice is " + berat.getVisibilityStatus());
        System.out.println("Bob is " + can.getVisibilityStatus());
    }
}

```

```

import java.util.ArrayList;
import java.util.List;

public class User implements Observer {
    private String name;
    private List<User> friends;
    private List<Post> wall;
    private SearchVisibilityStrategy visibilityStrategy;

    public User(String name) {
        this.name = name;
        this.friends = new ArrayList<>();
        this.wall = new ArrayList<>();
        this.visibilityStrategy = new PublicVisibility(); // Varsayılan görünrlük
    }

    public void addFriend(User user) {
        this.friends.add(: user);
    }

    public List<User> getFriends() {
        return friends;
    }

    public List<Post> getWall() {
        return wall;
    }

    @Override
    public void update(Post post) {
    }

    public void addPost(Post post) {
        wall.add(: post);
    }
}

```

```
public void setVisibilityStrategy(SearchVisibilityStrategy visibilityStrategy) {
    this.visibilityStrategy = visibilityStrategy;
}

public String getVisibilityStatus() {
    return visibilityStrategy.isVisible(user: this) ? "visible" : "invisible";
}

public String getName() {
    return name;
}

public void displayWall() {
    System.out.println(name + "'s Wall:");
    for (Post post : wall) {
        System.out.println(" " + post.getContent());
    }
}
}
```

```
public abstract class SocialNetworkFactory {
    public abstract User createUser(String name);
    public abstract Group createGroup(String name);
}
```

```
public interface SearchVisibilityStrategy {
    boolean isVisible(User user);
}
```

```
public class PublicVisibility implements SearchVisibilityStrategy {
    @Override
    public boolean isVisible(User user) {
        return true;
    }
}
```

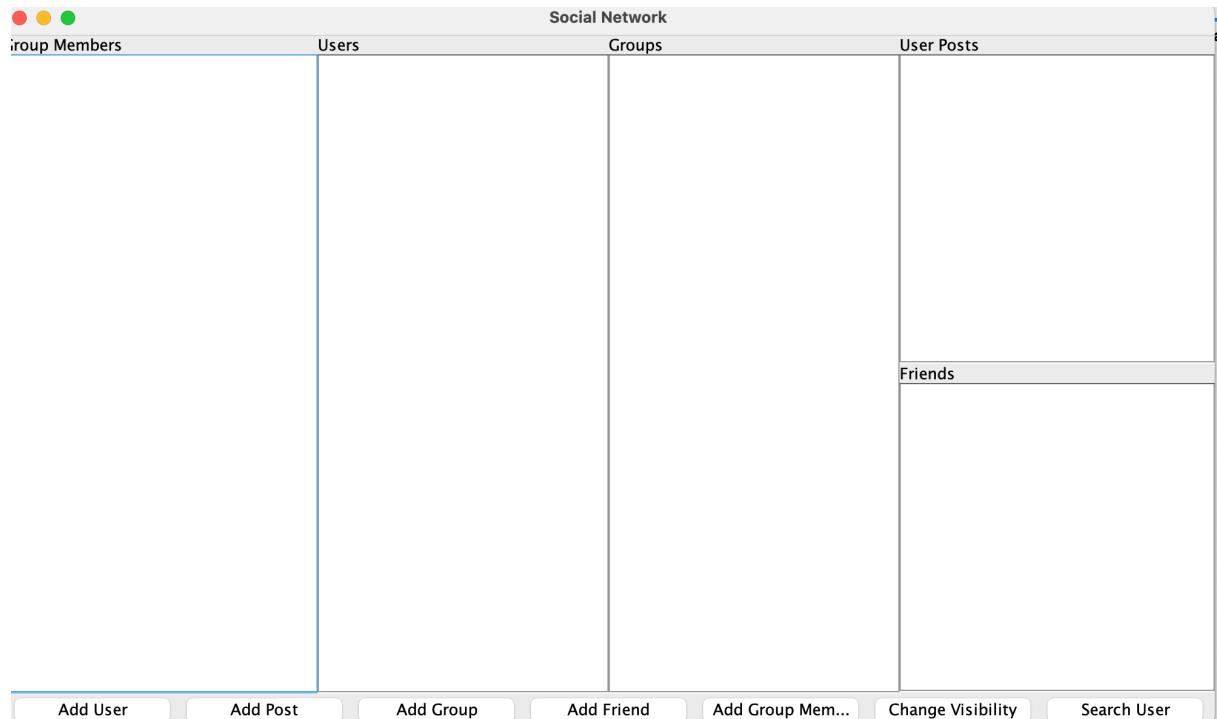
```
public class PrivateVisibility implements SearchVisibilityStrategy {  
    @Override  
    public boolean isVisible(User user) {  
        return false;  
    }  
}
```

```
public class Post {  
    private final String content;  
  
    public Post(String content) {  
        this.content = content;  
    }  
  
    public String getContent() {  
        return content;  
    }  
}
```

```
public interface Observer {  
    void update(Post post);  
}
```

```
public class ConcreteSocialNetworkFactory extends SocialNetworkFactory {  
    @Override  
    public User createUser(String name) {  
        return new User(name);  
    }  
  
    @Override  
    public Group createGroup(String name) {  
        return new Group(name);  
    }  
}
```

KULLANICI KİLAVUZU



- 1.Add User:** Kullanıcı ekler.
- 2.Add Post:** Kullanıcının duvarına gönderi ekler.
- 3.Add Group:** Yeni grup oluşturur.
- 4.Add Friend:** Userları arkadaş yapar.
- 5.Add Group Member:** Gruba üye ekler.
- 6.Change Visibility:** Kullanıcının gizliliğini belirler.
- 7.Search User:** Girilen isimde kullanıcı olup olmadığını sorgular.

ÖRNEK

Group Members	Users	Groups	User Posts
berat can	berat can	ege bilmüh	berat: merhaba
			Friends can

Add User Add Post Add Group Add Friend Add Group Mem... Change Visibility Search User

