Observer Pattern

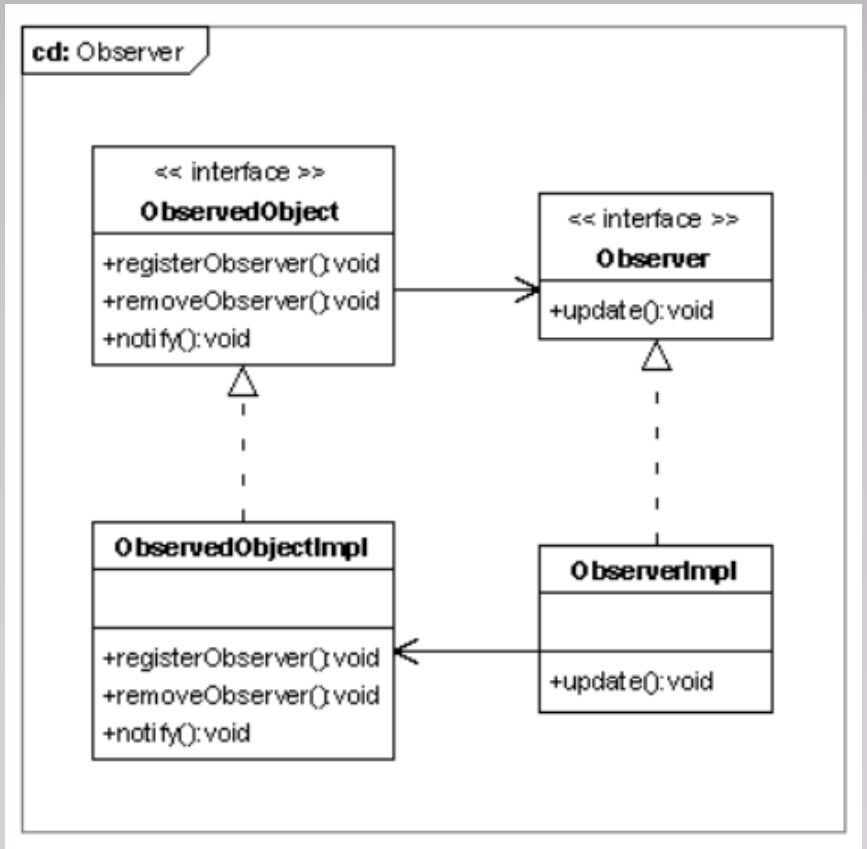
Dr. Öğr. Üyesi Fatih ÖZYURT

Fırat Üniversitesi Yazılım Mühendisliği Bölümü

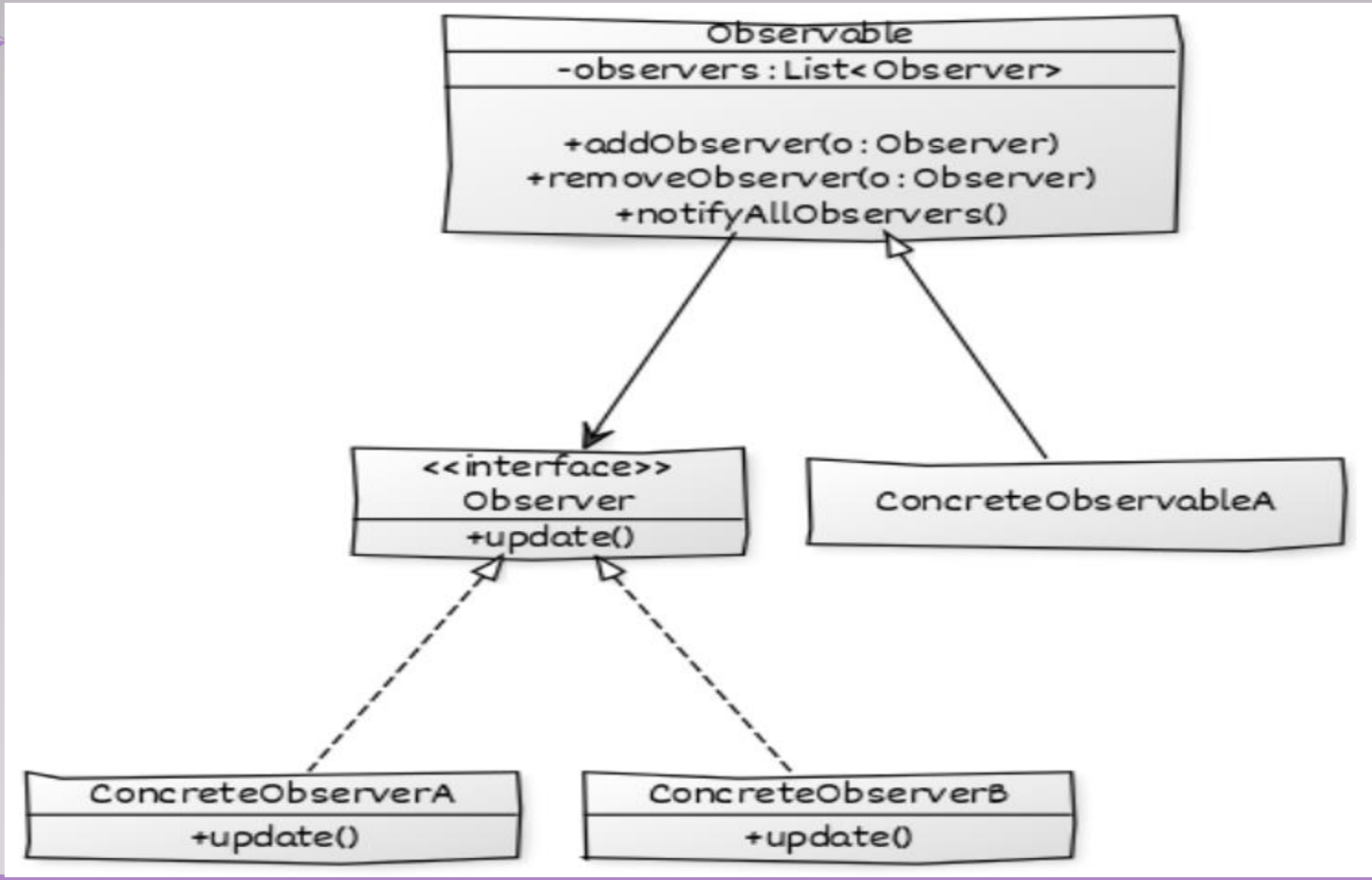
Gözetleyici (Observer Pattern)

- Observer tasarım deseni, bir nesnede meydana gelen değişikliği içinde bulunduğu listedeki tüm elemanlara bildiren **davranışsal** bir tasarım modelidir.
- Bir gazeteye günlük abone olmak istediğimizi düşünelim. Abone olduğumuz günden itibaren gazete her gün evimize posta aracılığıyla gönderilir. Burada gazete ile aboneleri arasında doğrudan bir ilişki oluşmaktadır.
- Hergün yeni bir baskı yapıldığı zaman aboneler otomatik olarak gazete tarafından bilgilendirilir. Bu bilgilendirme işlemi, gazetenin aboneye posta ile gönderilmesi anlamına gelmektedir.
- Abone olduğumuz sürece her gün gazete bize gönderilir. İstediğimiz bir zaman gazeteye haber vererek, aboneliğimizi iptal ettirebiliriz. Abonelik sona erdikten sonra bize posta ile gazete gönderilmez. Gözetleyici tasarım şablonu bir abonelik sistemi gibi çalışmaktadır.

Gözetleyici (Observer Pattern)



Gözetleyici (Observer Pattern)



Gözetleyici (Observer Pattern)

- **Observer:** Değişimden etkilenecek, izleyecek olan sınıfların uyguladığı arayüzdür. Duruma göre soyut sınıf (abstract class) da olabilir.
- **ConcreteObserver:** Observer arayüzünü uygulayan, değişimi takip eden sınıflardır. Değişim update metotu ile bildirilir.
- **Observable:** Takip edilecek olan yapımızdır. İçerisinde Observer (değişimden etkilenecek) nesneleri tutar.
- **ConcreteObservable:** Observer sınıfından türerler.

Gözetleyici (Observer Pattern)

- Senaryo olarak kullanıcılarımızın olduğunu ve bu kullanıcıların fiyatını takip ettiği bir ürün olduğunu farz edelim. Ürün fiyatı değiştikten sonra takip eden kullanıcılara bir mesaj gittiğini düşünelim.

```
// Değişimden etkilenecek, izleyecek olan sınıfların uyguladığı arayüzdür.
// UML diyagramındaki Observer yapısına denk gelmektedir.
interface IObserverUser
{
    void SendNotification(ProductUpdateMessage message);
}

// Observer arayüzünü uygulayan, değişimi takip eden sınıflardır.
// Değişim update metotu ile bildirilir.
// UML diyagramındaki ConcreteObserver yapısına denk gelmektedir.
class ObserverUser : IObserverUser
{
    public void SendNotification(ProductUpdateMessage message)
    {
        Console.WriteLine(message.ToString());
    }
}
```

Gözetleyici (Observer Pattern)

```
// Bildirilecek olan nesnelere paremetre olarak giden mesaj.
// UML diyagramı ile ilgisi yoktur.
class ProductUpdateMessage
{
    public string productName;
    public string message;

    public override string ToString()
    {
        return $"{productName} updated. Message: {message}";
    }
}
```

Gözetleyici (Observer Pattern)

```
// Takip edilecek olan yapımızdır.
// İçerisinde Observer (değişimden etkilenecek) nesneleri tutar.
// UML diyagramındaki Observable yapısına denk gelmektedir.
abstract class ObservableProduct
{
    private List<IObserverUser> _users;
    protected ProductUpdateMessage message;

    public ObservableProduct()
    {
        _users = new List<IObserverUser>();
        message = new ProductUpdateMessage();
    }

    public void AddObserver(IObserverUser observerUser)
    {
        _users.Add(observerUser);
    }

    public void RemoveObserver(IObserverUser observerUser)
    {
        _users.Remove(observerUser);
    }

    public void NotifyObserver()
    {
        foreach (IObserverUser user in _users)
        {
            user.SendNotification(message);
        }
    }
}
```

Gözetleyici (Observer Pattern)

```
// Observer sınıfından türer.
// UML diyagramındaki ConcreteObservable yapısına denk gelmektedir.
class Samsung : ObservableProduct
{
    // Ürün fiyatı değiştikten sonra Observer örneklerine bildirim gönderilir.
    public void ChangePrice()
    {
        // Ürün fiyatlarının güncellenmesi vs.

        this.message.productName = "Samsung";
        this.message.message = "Samsung's price updated.";
        this.NotifyObserver();
    }
}
```

Gözetleyici (Observer Pattern)

```
// Güncellemeden etkilenecek olan sınıf örnekleri.
IObserverUser mehmet = new ObserverUser();
IObserverUser derya = new ObserverUser();
IObserverUser sema = new ObserverUser();
IObserverUser aleyna = new ObserverUser();


// İzlenecek olan sınıf örneği.
Samsung samsung = new Samsung();

// İzlenecek olan sınıfa etkilenecek olan nesnelerin atanması.
samsung.AddObserver(mehmet);
samsung.AddObserver(derya);
samsung.AddObserver(sema);
samsung.AddObserver(aleyna);

samsung.ChangePrice();

// output:
//  Samsung updated. Message: Samsung's price updated.
//  Samsung updated. Message: Samsung's price updated.
//  Samsung updated. Message: Samsung's price updated.
//  Samsung updated. Message: Samsung's price updated.
```

Observer tasarım şablonu ne zaman kullanılır?

- Bir nesne üzerinde yapılan değişiklik başka nesnelerin de değiştirilmesini zorunlu kılıyorsa ve kaç nesnenin değiştirilmesi gerektiğini bilmiyorsak, gözetleyici tasarım şablonun kullanarak nesneler üzerinde gerekli değişiklikleri uygulayabiliriz.

Referanslar

1. Özcan acar Design pattern kitabı

2. https://javabeginnerstutorial.com/

3. Yusuf Yılmaz Ders notları

Sorularınız