logo, metin, simge, sembol, kırpıntı çizim içeren bir resim

Açıklama otomatik olarak oluşturuldu

T.C.

ONDOKUZ MAYIS ÜNİVERSİTESİ LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ BİLGİSAYAR MÜHENDİSLİĞİ

TASARIM DESENLERİ-BM634

Observer Pattern Çalışma Raporu

FIRAT KAAN BİTMEZ - 23281855

SAMSUN, 2024-2025 Eğitim Öğretim Yılı Güz Yarıyılı

**1. Giriş**

Yazılım mimarileri, sürekli olarak gelişen ve karmaşıklaşan uygulamalara cevap vermek üzere tasarlanır. Bu süreçte, iyi tanımlanmış prensipler, standartlar ve desenler, geliştiricilere kalıcı, bakımı kolay ve genişletilebilir çözümler sunar. Tasarım desenleri (Design Patterns), bu bağlamda sık tekrarlanan problemleri yeniden kullanılabilir şablonlarla çözmeyi hedefleyen, yazılım mimarisinin temel yapı taşlarındandır.

Bu raporun odak noktası olan Observer Tasarım Deseni, nesneler arası etkileşim ve bilgi akışını gevşek bağlamlı bir şekilde yöneterek, dinamik ve esnek yazılım çözümleri sunar. Observer Pattern, bir "yayın-abone" (publish-subscribe) yaklaşımını temsil eder ve gerçek hayatta sıkça karşılaşılan "bir şey değiştiğinde ilgililerin haberdar edilmesi" gereksinimine doğrudan cevap verir.

**2. Observer Pattern’in Tanımı ve Temel Prensipleri**

Observer Pattern, bir nesnenin (Subject ya da Observable) durumundaki değişiklikleri, bu değişikliklerle ilgilenen diğer nesnelere (Observer ya da Gözlemci) otomatik olarak bildirmek için kullanılır. Temel prensip, Subject’in durumunu izleyen bir veya birden çok Observer’ın, Subject’te bir güncelleme olduğunda bildirim alması üzerine kuruludur.

Bu desen, ilgilenen tarafların (Observer’ların) tek tek izlenecek nesneden veri "çekmesine" gerek kalmadan, nesnenin kendisini takipçilerine bilgilendirmesi mantığını benimser. Böylece:

* Subject, Observer’ların sayısını veya türünü bilmez. Bu sayede gevşek bağlılık (loose coupling) elde edilir.
* Observer’lar istedikleri zaman abonelikten çıkabilir ya da yenileri eklenebilir.
* Sistemdeki değişikliklerin yayılması otomatik olarak halledilir ve kodun tekrar kullanılabilirliği artar.

**3. Tarihsel Arka Plan ve Motivasyon**

Observer Pattern, yazılım literatüründe ilk defa resmi olarak "Gang of Four" (GoF) olarak bilinen yazarlar tarafından popülerleştirilen 1994 tarihli "Design Patterns: Elements of Reusable Object-Oriented Software" adlı eserde tanımlanmıştır. Bu dönemde geliştirilen grafik arayüz kütüphaneleri, MVC (Model-View-Controller) mimarisi, finansal uygulamalarda gerçek zamanlı fiyat güncellemeleri, dağıtık sistemlerdeki anlık bildirimler ve benzeri ihtiyaçlar, Observer Pattern’in önemini artırmıştır. Uygulamalarda modeldeki (Subject) bir değişikliğin arayüzdeki (Observer) görünüme otomatik yansıtılması, kullanıcı deneyiminin iyileştirilmesi ve kodun esnekliğinin arttırılması gibi motivasyonlar bu deseni yaygınlaştırmıştır.

**4. Yapısal Elemanlar**

Observer Pattern, iki temel rolü merkezine alır:

1. **Subject (Observable)**: Takip edilen nesnedir. Sahip olduğu durumu yönetir, Observer’ların abonelik işlemlerinden (ekleme/çıkarma) sorumludur. Durumu değiştiğinde, kayıtlı tüm Observer’lara bildirim gönderir.
2. **Observer (Gözlemci)**: Subject’in durumundaki değişikliklerden haberdar olmak isteyen nesnedir. Observer, Subject’ten gelen bildirimler doğrultusunda kendi iç mantığını günceller. Observer’ın görevi genellikle gelen veriyi işlemek, görüntülemek, kaydetmek veya farklı bir eylemi tetiklemektir.

Bu iki rol arasındaki etkileşim "publish-subscribe" modelini andırır: Subject veriyi yayınlar, Observer’lar ise bu yayına abonedir.

A diagram of a product

Description automatically generated

**5. Teorik Temeller ve Matematiksel Modelleme**

Observer Pattern’i soyut bir çerçevede ele alırsak, Subject’in durumu S(t) adı verilen bir değişken veya vektörle temsil edilebilir. Zaman veya durum güncellemelerini t anına göre modellediğimizde, her Observer O\_i(t), S(t)’nin bir fonksiyonu olarak düşünülebilir:

Oi(t)=fi(S(t))O\_i(t) = f\_i(S(t))Oi​(t)=fi​(S(t))

Bu ifade, Subject durum uzayından Observer çıktı uzayına bir fonksiyonel bağımlılık kurar. Subject’in durumu güncellendiğinde, tüm Observer fonksiyonları bu yeni değeri kullanarak kendilerini günceller. Bu mekanizma, bilgi teorisi bakış açısından, Subject’in ürettiği bir sinyali (veri akışını) birden fazla tüketiciye (Observer’a) yönlendiren bir yayınlama/dağıtım yapısıdır.

**6. Kullanım Alanları**

Observer Pattern, pek çok pratik senaryoda kullanılır:

* **Grafik Kullanıcı Arayüzleri (GUI)**: Modelde yapılan değişikliklerin arayüze otomatik yansıtılması.
* **Haber Bültenleri / E-Posta Listeleri**: Yeni içerik veya kampanya eklendiğinde abonelere bildirim gönderme.
* **Finansal Uygulamalar**: Hisse senedi fiyatları değiştiğinde ilgili izleyicilere gerçek zamanlı veri sağlama.
* **Oyun Motorları**: Oyun nesnesindeki duruma bağlı olarak, farklı bileşenlerin (health bar, skor tablosu vb.) güncellenmesi.
* **Sensör Verileri İzleme**: Sensörlerden gelen anlık verilerin ilgili izleyicilere aktarılması.
* **Dağıtık Sistemler ve Mikroservisler**: Bir serviste oluşan değişikliğin diğer servis ve bileşenlere asenkron biçimde iletilmesi.

**7. C# .NET Ortamında Uygulama**

C# ve .NET platformu, Observer Pattern’i uygulamak için birden fazla yaklaşım sunar. Bunlar arasında basit delegate/event tabanlı çözümlemelerden, IObserver<T> ve IObservable<T> arayüzlerine, hatta Reactive Extensions (Rx) kütüphanesine kadar geniş bir yelpaze bulunur.

**7.1 Event/Delegate Tabanlı Uygulama**

En temel yaklaşım, C#’ın sunduğu event ve delegate mekanizması kullanılarak gerçekleşir. Subject, bir event tanımlar; Observer ise bu event’e abone olarak güncellemeleri alır. Bu yaklaşım, Observer Pattern’in doğal bir dil özelliği olan event’lerle elde edilmesini sağlar.

**7.2 IObserver<T> ve IObservable<T> Arayüzleri**

.NET 4.0 sonrası, dilin standart kütüphanesi IObservable<T> ve IObserver<T> arayüzlerini sunar. Bu arayüzler, tasarım desenine standart bir çerçeve kazandırır. Subject IObservable<T>’yi uygular, Subscribe metodu ile Observer’ları kaydeder. Observer ise IObserver<T> arayüzü ile OnNext, OnError ve OnCompleted metodlarını barındırır. Bu yöntem, Observer Pattern’i daha resmi ve reusable bir forma sokar.

**7.3 Reactive Extensions (Rx)**

Reactive Extensions, Observer Pattern’in fonksiyonel ve deklaratif bir yorumunu getirir. Rx, veri akışlarını gözlemleyerek, LINQ-benzeri sorgularla dönüştürüp filtreleyerek asenkron ve zaman tabanlı işleme olanak tanır. Bu sayede, Observer Pattern çok daha zengin ve güçlü bir ekosisteme sahip olur. Zaman tabanlı operasyonlar, bir veri akışını geciktirmek, birleştirmek ya da hata yönetimi stratejileri eklemek Rx ile oldukça kolay hale gelir.

**8. Avantajlar ve Dezavantajlar**

**8.1 Avantajlar**

* **Gevşek Bağlılık**: Subject Observer’ların kim olduğunu bilmez, sadece değişikliği yayınlar. Bu sayede bileşenler arasında sıkı bir bağ yoktur.
* **Kolay Genişletilebilirlik**: Yeni Observer eklemek veya mevcut bir Observer’ı çıkarmak, Subject tarafında değişiklik gerektirmez.
* **Yeniden Kullanılabilirlik**: Aynı Subject birden fazla farklı Observer tarafından izlenebilir, bu da kodun tekil bileşenlerinin birden çok senaryoda tekrar kullanımını kolaylaştırır.
* **Bakım Kolaylığı**: Değişiklik yaparken yaygın etkiyi azaltır, sistemdeki güncellemeler kolayca izlenip yönetilebilir.

**8.2 Dezavantajlar**

* **Çoklu Observer Yönetimi**: Çok sayıda Observer olduğunda performans ve yönetim açısından karmaşıklık artabilir.
* **Bildirim Sıkıntıları**: Hatalı veya gereksiz bildirimler sistemde gereksiz yüke yol açabilir.
* **Senkronizasyon Sorunları**: Çok iş parçacıklı (multi-threaded) ortamlarda güncellemelerin senkronizasyonu zor olabilir. Eşzamanlılık konularında dikkatli tasarım ve bazen ek senkronizasyon mekanizmaları şarttır.

**9. Test, Bakım ve Performans Değerlendirmesi**

Observer Pattern tabanlı sistemlerin test edilmesi görece kolaydır. Subject için test senaryolarında, sahte (mock) Observer’lar kullanılarak yayınlanan bildirimlerin doğru zamanda ve doğru sayıda yapıldığı doğrulanabilir. Observer için ise sahte Subject aracılığıyla tetiklenen bildirimlerin doğru tepkiyi verdiği test edilebilir.

Performans tarafında, çok sayıda Observer söz konusu olduğunda, Subject her değişimde tüm Observer’lara bildirim gönderir. Bu, bazı durumlarda bant genişliği ve işlemci tüketimini artırabilir. Bu tür sorunları aşmak için filtreleme, durum değişimlerinin toplu bildirimleri, önbellekleme veya Rx üzerinde operatörlerle veri akışını optimize etme yöntemleri kullanılabilir.

Bakım aşamasında, Observer Pattern sayesinde sistemin bileşenlerini bağımsız olarak değiştirmek daha kolaydır. Bir Observer’ın kaldırılması ya da eklenmesi, Subject’te bir değişiklik gerektirmez. Bu modüler yapı uzun vadede yazılımın evrimini ve güncellenmesini destekler.

**10. Diğer Tasarım Desenleriyle İlişki**

Observer Pattern çoğu zaman MVC, MVP, MVVM gibi yaygın mimari kalıplar içerisinde kullanılır. Model verilerini izleyen View veya ViewModel bileşenleri, Observer Pattern mantığıyla güncellenir. Ayrıca Decorator, Mediator, Publisher-Subscriber, Strategy, Command gibi desenlerle birlikte kullanılarak daha zengin mimariler oluşturulabilir.

A diagram of a program

Description automatically generated

**11. Örnek Uygulama Senaryosu**

**12. Sonuç**

**Kaynaklar**

* <https://github.com/firatkaanbitmez/ShoppingMasterApp>
* <https://github.com/firatkaanbitmez/DesignPatterns>
* <https://medium.com/kodcular/observer-design-pattern-nedir-671f61969c91>
* <https://www.gencayyildiz.com/blog/c-observer-design-patternobserver-tasarim-deseni/>
* <https://www.gokhan-gokalp.com/tr/c-observer-pattern-kullanimi/>
* <https://ahmetkucukoglu.com/observer-design-pattern-dotnet>
* <https://www.yasinatilkan.com/observer-design-pattern-tasarim-deseni-nedir/>
* <https://alkanfatih.com/observer-design-pattern-tasarim-deseni-c/>