**Observer (Abone) Kalıbı (Pattern)**

• **Intent**: Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

• **Amaç**: Nesneler arasında bire-çoklu bir bağımlılık tanımla öyle ki bir nesnenin durumunda değişiklik olduğunda, tüm bağımlı nesneler haberdar edilsin ve otomatik olarak güncellensin.

**PROBLEMS**

• Zaman zaman bir nesnenin durumundaki değişiklikleri takip etmek ve onlardan haberdar olmak isteriz.

• Bunu yapmanın bir yolu, belli aralıklarla nesnenin durumunu sorgulamaktır.

• Bu yaklaşım hem etkin değildir çünkü gereksiz yere devamlı sorgulama gerektirir, hem de değişikliklerden anında haberdar olunamaz, bir gecikme yaşanır.

**SOLUTION**

• Bir nesnenin durumundaki değişikliklerden haberdar olmanın bir başka ve daha etkin yöntemi ise **Observer** kalıbını kullanmaktır.

• Observer (abone) nesne, durumundaki değişikliklerden haberdar olmak istediği konu (subject, observable) nesneye abone olur.

• Konu nesne, durumunda bir değişiklik olduğunda, kendisine abone olan nesneleri bilgilendirir.

**• Bu yaklaşım Hollywood Prensibi olarak da adlandırılır:**

**• Don’t call us, we’ll call you!**

• Bu mekanizmadaki bazı tanımlar şunlardır:

• **Observable**: Durumundaki değişikliklerin takip edildiği nesne, konu ya da subject, publisher.

• **Observer**: Konu nesneye abone olup, ondaki değişikliklerden haberdar olmak isteyen nesne, abone ya da subscriber.

• Bu türden yapılara **listener** da denir.

• **Notification**, Broadcast: Konu nesnenin, durumunda bir değişiklik olduğunda abonelerini uyarması, güncelleme.

• **Event**: Konu nesnelerin abonelerini uyarırken kendilerine gönderdiği,neyin olduğuyla ilgili detay içeren bilgilendirme.

• Dolayısıyla konu nesne üzerinde, nesnelerin abone olmasına izin veren **arayüz** bulunur.

• Abone nesneler üzerinde de konudan gelen güncelleme mesajını (event notification) alacak bir **arayüz** bulunur.

• **Bundan dolayı bu kalıba “Publisher-Subscriber”, “Producer- Consumer” ya da “Event-Notification” da denir**

**SONUCLAR**

• Bir nesnedeki durumun değişiminden haberdar olmak isteyen pek çok nesne varsa **Observer** kalıbı kullanılır.

• **Observer** kalıbında durumu değişen konu nesne (subject), kendisine abone olanlarla ilgili bilgilendirme (notification) arayüzü dışında herhangi bir kabule ya da bağımlılığa sahip değildir.

• **Observer** kalıbı ile konu ve aboneleri arasında soyut bir bağımlılık kurulur.

• Konu, kendisine abone olan pek çok nesneye yayın yapar.

• Konuya abone olan nesneler, istemeseler de her türlü güncellemeden haberdar olurlar.

• Bunun önüne geçmek için farklı detayda konular kurgulanabileceği gibi filtreleme mekanizmaları da düşünülebilir.

**• Konuya abone olan nesnelerin, olayın oluşumundan sonra uyarılmaları ciddi zaman ve kaynak tüketen bir iş olduğunda, uyarım mekanizmasının senkron (synchronous) yerine asenkron (asynchronous) yapılması çok sık karşılaşılan bir durumdur.**

• Mesajların çok olması, email, rapor, imaj vb. boyut problemlerinin olması vb. sebepler.

• Böylece, sayısı ve performansı kontrol edilemeyen sistemlere olay bilgisi göndermenin maliyeti düşer.

• **Bu amaçla dil seviyesinde kanal (thread) yapıları, sistemler arası entegrasyonda ise Message Oriented Middleware (MOM) sistemleri kullanılır.**

**• Message Oriented Middleware (MOM), sistemler arası olay (event) mekanizması kuran bir yapıdır ve IBM MQ ya da Microsoft MSMQ gibi ürünler ve Java’da Java Messager Service (JMS), ve .NET Azure Bus Services gibi arayüzlerle gerçekleştirilir.**

• Bir sistemde pek çok konu ve abone varsa, bunlar arasındaki bildirimler **ChangeManager** gibi bir isme sahip merkezi bir yapı tarafından yönetilebilir.

• **ChangeManager** senkron-asenkron, garantili gönderme (guarranteed delivery), filtreleme vb. pek çok işi merkezi olarak yönetebilir

**KULLANIMLAR**

Sistemlerdeki olay (**event**) mekanizmalarında **Observer** kalıbı sık kullanılır.

• **Observer** kalıbının ilk defa **Smalltalk’ta** **MVC** kalıbında kullanıldığı bilinmektedir.

**• MVC’de Model konudur ve View da observerdır.**

• .NET ve Java’da GUI çerçevelerinde nesnelerin durumlarındaki değişimler için **Observer** kalıbı kullanır..

• **java.uti**l paketinde **Observer** kalıbı için gerekli **Observer** **arayüzü** ve **Observable** sınıfı vardır.

• Aboneler **Observer’ı** gerçekleştirip üzerindeki **update**()’i ezerler.

• Konu nesneler ise **Observable’ı** genişletir ve nesnenin durumundaki değişiklikleri **setChange**() metodu ile **JVM’e** haber verir.

• Abonelerin bu durumdan haberdar olmasını sağlamak için yine konu nesne üzerinde **notifyObserver**() metodunu çağırır.

• Bu yapı Java 9 ile deprecate edilmiştir.

• **Observer** kalıbı sıklıkla Reactive sistemlerle kıyaslanır.

**DİĞER KALIPLARLA İLİŞKİ**

**• Observer, ChangeManager için Mediator’u kullanabilir.**

**• ChangeManager Singleton olabilir.**

**UYGULAMA**

• **Command** kalıbında verilen **Account** örneğinde **Account** nesnesine yapılan işlemleri (**Transaction**) takip eden yapıyı **Observer** kalıbı ile kurgulayın.

• **Account** nesnesine karşı yapılan işlemleri için oluşturulan **Transaction** nesnelerini, **Account** nesnesini takip eden **TransactionListener** tarafından yaratılmasını sağlayın.