1906003172019

Tasarım Desenleri

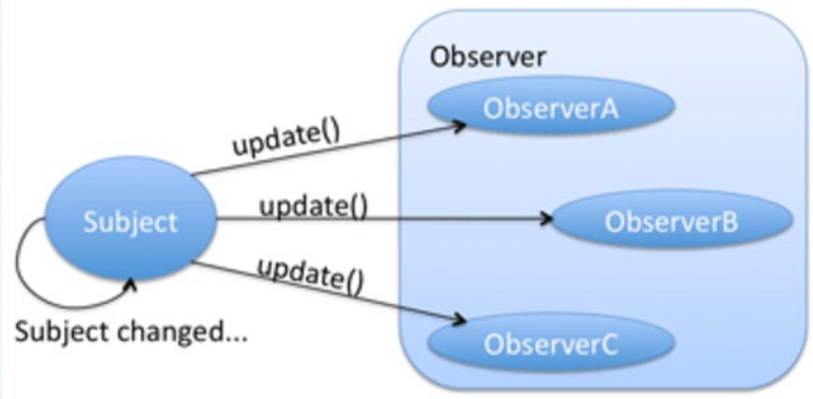
Dr. Öğr. Üy. Önder EYECİOĞLU
Bilgisayar Mühendisliği

Observer Patterns -Behavioral (Davranışsal)

GoF Tanımı: Nesneler arasında birden çoğa bağımlılık tanımlanır, böylece bir nesne durum değiştirdiğinde, tüm bağımlıları otomatik olarak bildirilir ve güncellenir.

Konsept

Bu modelde, belirli bir özneyi (subject) gözlemleyen birçok gözlemci (observer) vardır. Gözlemciler birkonuda bir değişiklik yapıldığında haberdar olmak isterler. Böylece kendilerini o konuya kaydettirirler. Konuya olan ilgilerini kaybettiklerinde, konunun kaydını silerler. Bazen bu modele Yayıncı-Abone (publish-subscribe pattern) modeli de denir.



Observer Patterns-Behavioral (Davranışsal)

Bilgisayar Dünyası Örneği

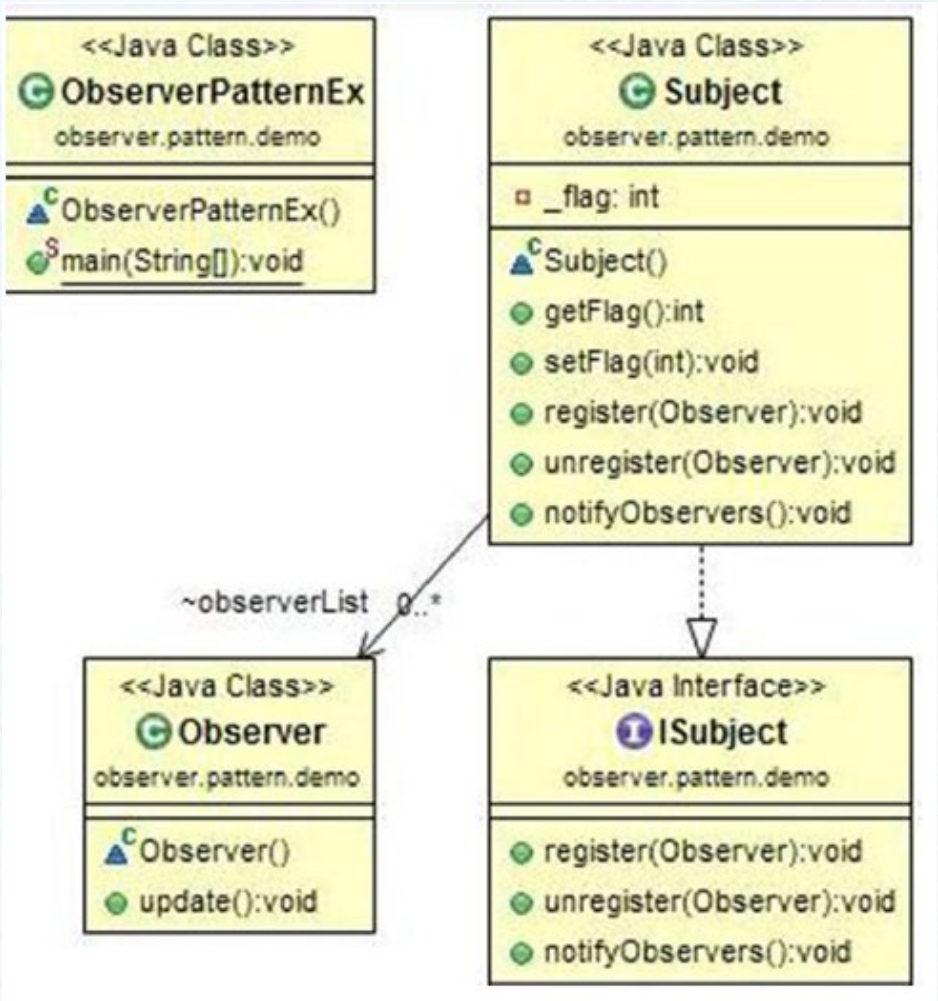
Bilgisayar bilimi dünyasında, bu kullanıcı arabiriminin bir veritabanıyla (veya iş mantığıyla) bağlantılı olduğu, kullanıcı arabirimi tabanlı basit bir örnek düşünün. Bir kullanıcı, bu UI aracılığıyla bazı sorgular yürütebilir ve veritabanını aradıktan sonra, sonuç UI'ye geri yansıtılır. Çoğu durumda, kullanıcı arayüzünü veritabanıyla ayırırız. Veritabanında bir değişiklik olursa, değişikliğe göre görüntüsünü güncelleyebilmesi için UI'ye bildirilmelidir.

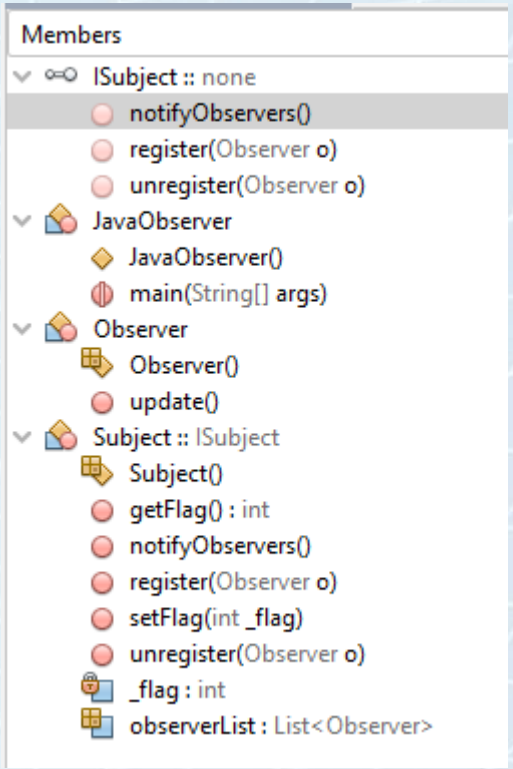
Observer Patterns-Behavioral (Davranışsal)

İllüstrasyon

Şimdi doğrudan basit örneğimize girelim. Burada bir «**observer**» (daha fazlasını yaratabilirsiniz) ve bir **«subject»** oluşturalım. **«subject»** , tüm «**observer**» için bir liste tutar (ancak burada basitlik için sadece bir tane var). Buradaki «**observer»**, **«subject» ile** ilgili bayrak değeri değiştiğinde bilgilendirilmek istiyor. Çıktı ile, bayrak değeri 5 veya 25 olarak değiştirildiğinde gözlemcinin bildirimleri aldığını keşfedeceksiniz. Ancak bayrak değeri 50'ye değiştiğinde herhangi bir bildirim yok çünkü bu zamana kadar gözlemci kendisini konudan kaydını silmiş durumda olacak.

Observer Patterns-Behavioral (Davranışsal)





Decorator Patterns-Structural (Yapısal)

GoF Tanımı: Bir nesneye dinamik olarak ek sorumluluklar ekler. Dekoratörler, işlevselliği genişletmek için alt sınıflamaya esnek bir alternatif sunar.

Bir sınıfın davranışını değiştirmemiz gerektiğinde akla gelen ilk şeylerden birisi temel operasyonların tanımlandığı bir sınıf tanımlamak ve daha sonra o sınıfı genişletmek bu sayede aynı davranışı farklı şekillerde sergileyen birçok sınıf olacaktır.

Konsept

Bu modelin bu ana ilkesi, mevcut işlevleri değiştiremeyeceğimizi, ancak genişletebileceğimizi söyler. Başka bir deyişle, bu örüntü genişlemeye açık, ancak modifikasyona kapalıdır. Çekirdek kavram, tüm sınıf yerine belirli bir nesneye bazı belirli işlevler eklemek istediğimizde geçerlidir.

Yazılım mühendisliğinde, **dekoratör tasarım deseni** , aynı sınıfın diğer örneklerini değiştirmeden, bir sınıfın belirli bir örneğine ek özellikler veya davranışlar eklemek için kullanılır. Dekoratörler, işlevselliği genişletmek için alt sınıflandırmaya esnek bir alternatif sunar.

Decorator Patterns-Structural (Yapısal)

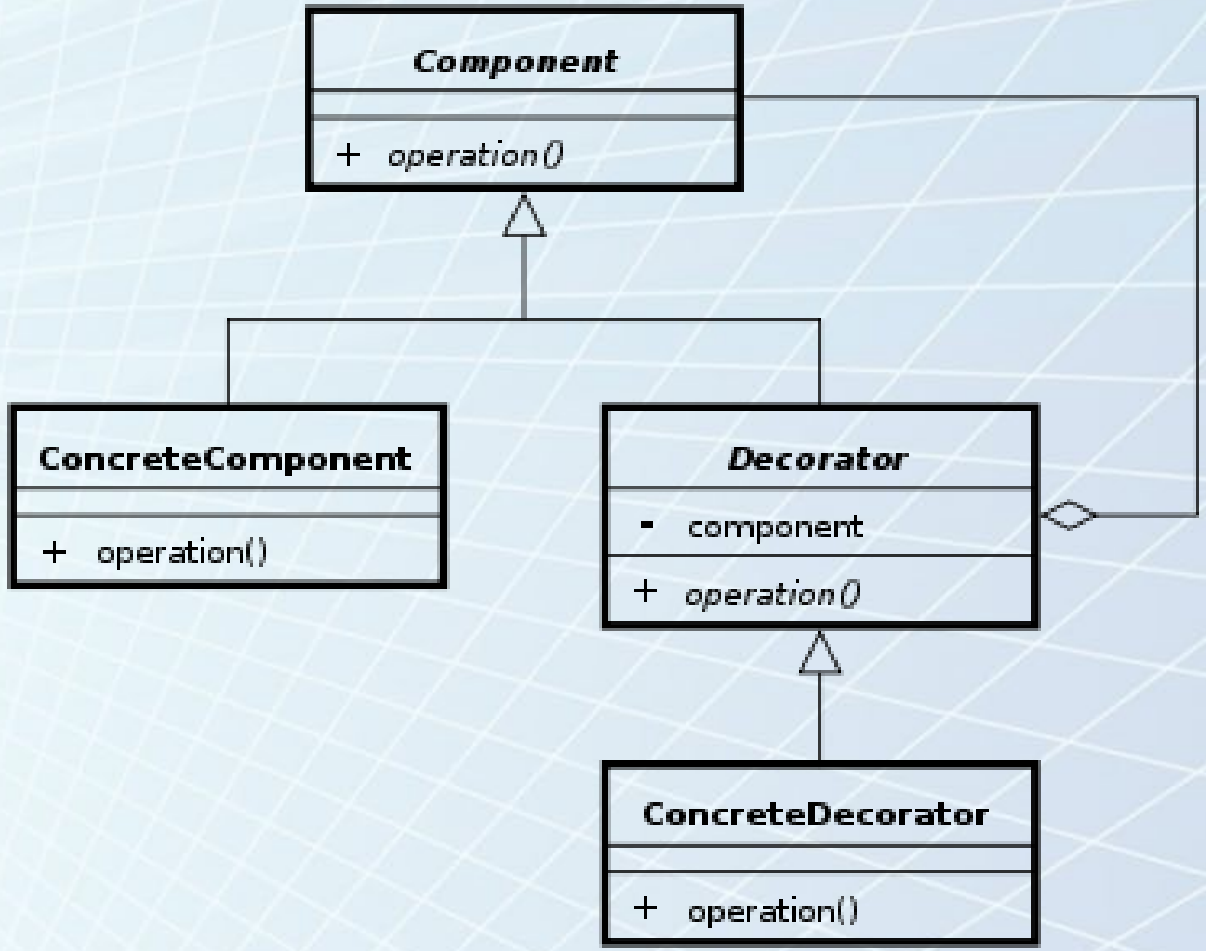
Dekoratör Tasarım modelinin katılımcıları şunlardır:

•**Bileşen** – bu, çalışma zamanında kendisiyle ilişkili ek sorumluluklara sahip olabilecek sarıcıdır.

•**Somut bileşen** – programda ek sorumlulukların eklendiği orijinal nesnedir.

•**Dekoratör** -bu, bileşen nesnesine bir başvuru içeren ve ayrıca bileşen arabirimini uygulayan soyut bir sınıftır.

•**Somut dekoratör** -dekoratörü genişletir ve Bileşen sınıfının üzerine ek işlevsellik oluşturur.



Decorator Patterns-Structural (Yapısal)

Bilgisayar Dünyası Örneği

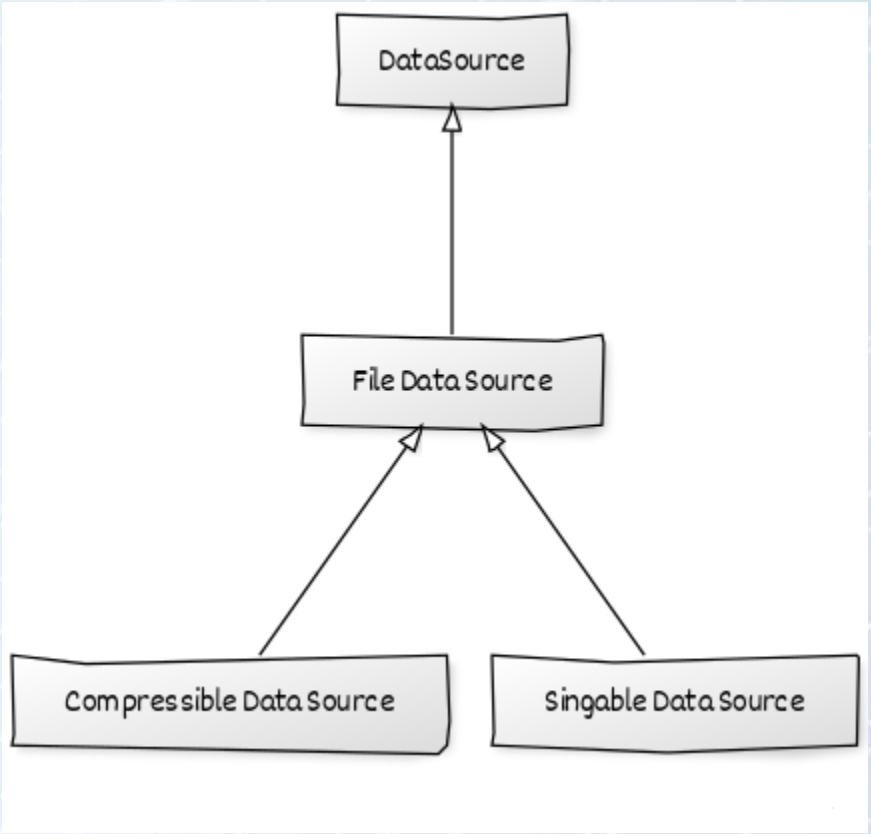
Bir dosyanın okunması ve dosyaya yazılma işlemi.

Dosya işlemlerinde; yazılan dosyanın sıkıştırılması ya da bir çeşit imzalama işlemlerine tabi tutulması, okunan dosyaların değiştirilmiş olup olmadığı kontrol gibi senaryolar da gelebilir. Bu tür senaryolar karşısında mevcut sınıftan yeni sınıflar türetme yolunda ilerleriz.

Gittikçe hiyerarşi artmaktadır. Bunun yanı sıra;

•Üst sınıflarda yapılacak herhangi bir değişiklik hiyerarşinin alt kısımlarını da etkilemektedir.

•Sıkıştırma işlemlerini yaparken imzalama işleminin de yapılması istendiğinde ortaya bu farklı durumların kombinasyonu kadar sonuç çıkmaktadır.



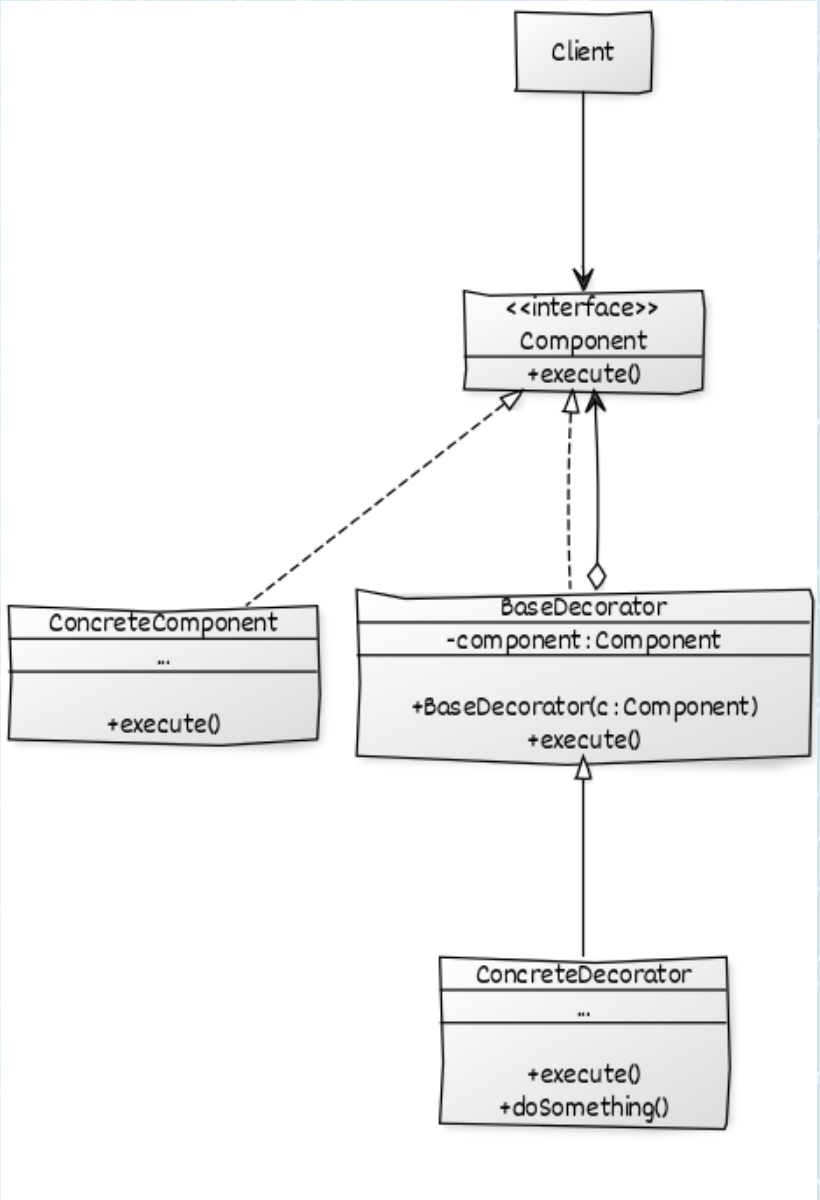
Decorator Patterns-Structural (Yapısal)

Bilgisayar Dünyası Örneği

•Çoğu programlama dilinde bir sınıfın sadece bir adet üst sınıfı olmaktadır, yani bir sınıftan türeyebiliyor. Bu da hiyerarşi için oldukça zor duruma sokacaktır bizleri.

•Kalıtım statik bir yapıdadır bu yüzden run-time sırasında nesnenin davranışı değiştirilemeyecektir. (Alt sınıf davranışları hariç) gibi bir çok dezavantaj ortaya çıkmaktadır bu tür olaylarda. Bu tür durumlarda Composite tasarım deseni kullanılmaktadır. Kalıtımdan ziyade composition (bileşim) işlemi yapılmalıdır. Yanda bu desene ait UML diyagramı bulunmaktadır.

.



Decorator Patterns-Structural (Yapısal)

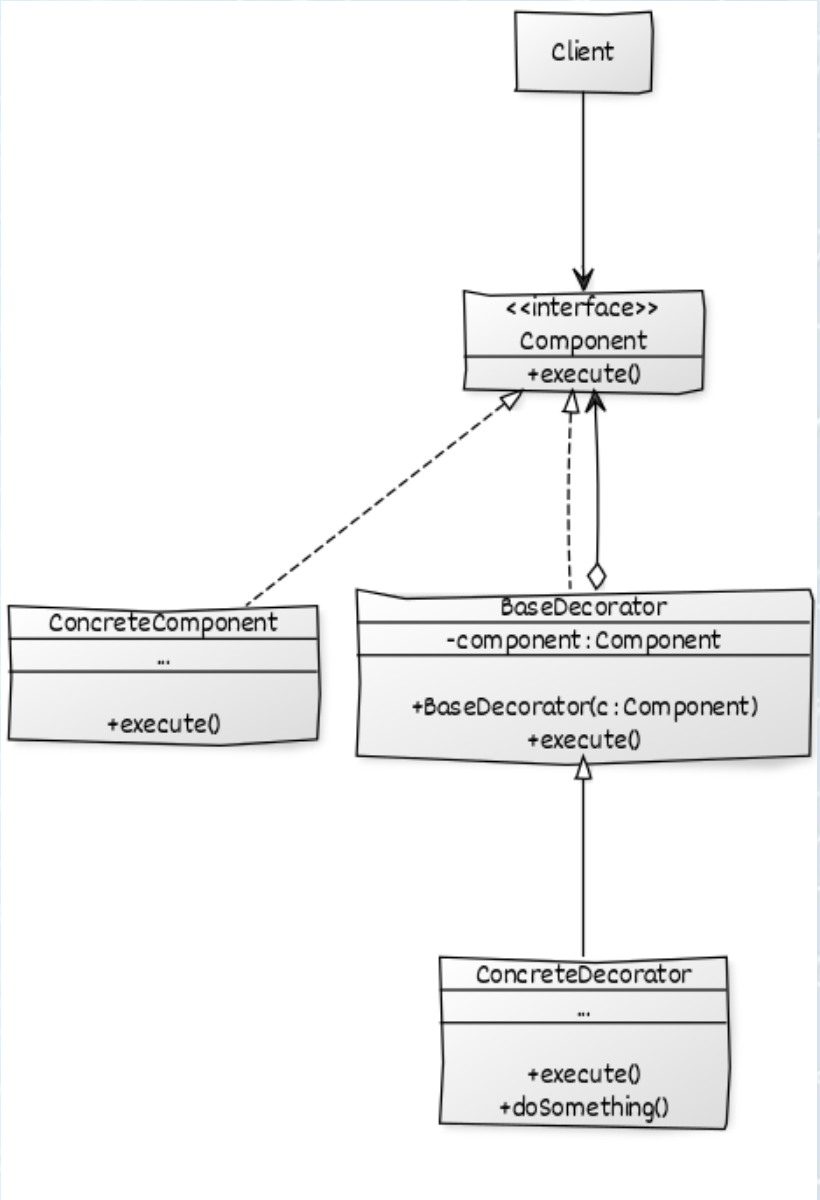
İllüstrasyon

Component: Üst sınıfların uygulaması için ortak bir arayüz. Bu arayüzde tanımlanan işlemler daha sonra ConcreteDecorator sınıfları tarafından değiştirilen tanımlamalardır.

ConcreteComponent: Temel davranışın uygulandığı sınıftır. ConcreteDecorator sayesinde değiştirilecektir.

BaseDecorator: Component arayüzünü uygular ve bu arayüzü uygulayan yapının referansını da barındırır.

ConcreteDecorator: Yeni davranışların tanımlandığı sınıftır, BaseDecorator sınıfından türer.



Decorator Patterns-Structural (Yapısal)

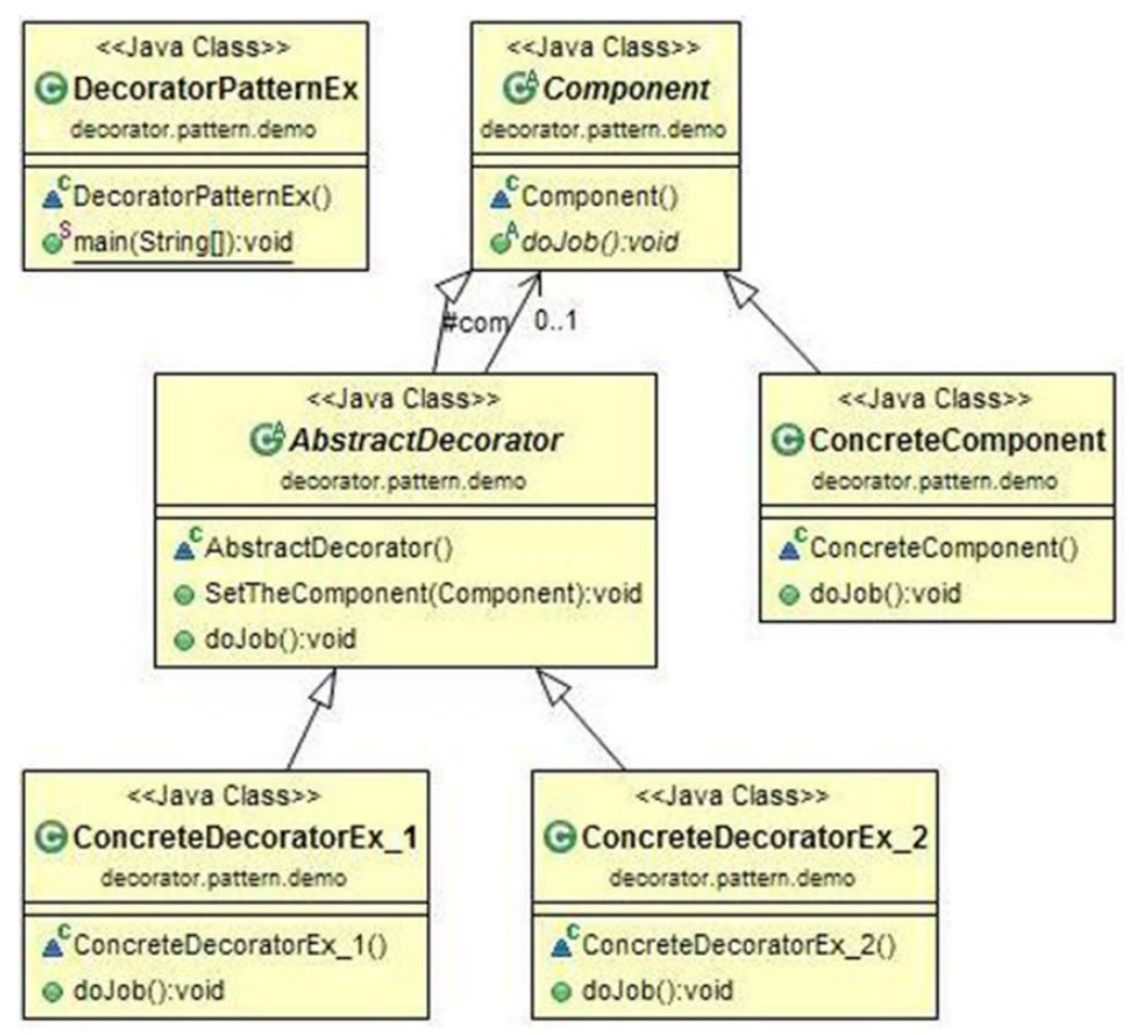
Bilgisayar Dünyası Örneği

•GUI tabanlı bir araç setinde bazı sınır özellikleri eklemek istediğimizi varsayalım. Bunu miras yoluyla yapabiliriz. Ancak en iyi çözüm olarak değerlendirilemez çünkü kullanıcımız veya müşterimiz oluşturma üzerinde mutlak kontrole sahip olamaz. Bu seçimin özü orada statiktir. Dekoratörler bize daha esnek bir yaklaşım sunabilir: **burada bileşeni (component) başka bir nesneyle çevreleyebiliriz**. Çevreleyen nesne "dekoratör" olarak adlandırılır. Bu, Dekore edilen bileşenin arayüzüne uygundur. İstekleri bileşene iletir. Bu yönlendirme isteklerinden önce veya sonra ek işlemler gerçekleştirebilir. Bu konsepte sınırsız sayıda sorumluluk eklenebilir.

Decorator Patterns-Structural (Yapısal)

İllüstrasyon

Burada orijinal doJob() yönteminin işlevselliğini değiştirmeye çalışmadık. İki dekoratör, ConcreteDecoratorEx_1 ve ConcreteDecoratorEx_2, işlevselliği geliştirmek için buraya eklenir, ancak orijinal doJob()'un çalışması bu ekleme nedeniyle bozulmaz.



Strategy (Policy) Patterns-Behavioral (Davranışsal)

GoF Tanımı: Strategy tasarım deseni, bir algoritma ailesi tanımlamamızı, her birini ayrı bir sınıfa koymamızı, her birinin kapsüllenmesini ve nesnelerinin birbiriyle değiştirilebilir hale getirmenizi sağlayan davranışsal bir tasarım modelidir.

Bir algoritmanın davranışını çalışma zamanında dinamik olarak seçebiliriz.

Birbirinin yerine geçen işlevleri biraraya getirir ve delagasyon kullanarak hangisinin kullanılacağına çalışma zamanında karar verilmesini sağlar. Bu örüntü kullanılarak bir işlevin farklı gerçekleştirimleri veya farklı algoritmaları isteğe bağlı olarak uygulanabilir.

Önemli olan nokta, bu uygulamaların birbirinin yerine geçebilmesidir - göreve bağlı olarak, uygulama iş akışını bozmadan bir uygulama seçilebilir..

Konsept

Strateji modeli, bir algoritmayı ana bilgisayar sınıfından kaldırmayı ve aynı programlama bağlamında, çalışma zamanında seçilebilecek farklı algoritmalar (yani stratejiler) olabilmesi için onu ayrı bir sınıfa koymayı içerir.

Strategy (Policy) Patterns-Behavioral (Davranışsal)

Konsept

Strateji modeli , bir istemci kodunun ilgili ancak farklı algoritmalar ailesinden seçim yapmasını sağlar ve istemci bağlamına bağlı olarak çalışma zamanında herhangi bir algoritmayı seçmesi için basit bir yol sağlar.

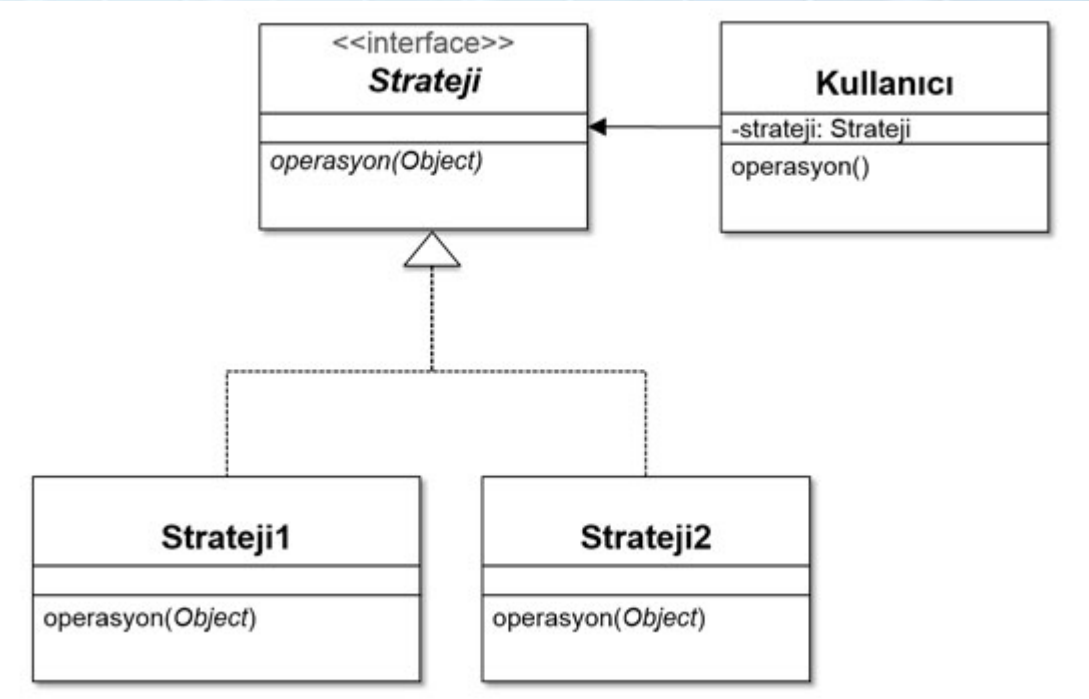
Açık/kapalı Prensibi ile Hareket Eder

Bu model Açık/kapalı ilkesine dayanmaktadır . [Değişiklik için kapalı] bağlamı değiştirmemize gerek yok, ancak herhangi bir uygulamayı [uzantıya açık] seçip ekleyebiliriz.

Örneğin, Collections.sort()– farklı sıralama sonuçları elde etmek için sıralama yöntemini değiştirmemize gerek yoktur. Çalışma zamanında sadece farklı karşılaştırıcılar sağlayabiliriz.

Strategy Patterns-Behavioral (Davranışsal)

Strateji örüntüsünü gösteren sınıf diyagramı : Burada Kullanıcı sınıfı, aynı işlevi (operasyon) farklı stratejileri uygulayarak yerine getirir. Farklı stratejileri gerçekleştirmek için Strateji türünde bir alan tutar (strateji) ve bu sınıftan oluşturulan nesnelere, atanan farklı stratejileri (aynı anda yalnızca bir strateji) uygulayabilir.



Strategy Patterns-Behavioral (Davranışsal)

Bilgisayar Dünyası Örneği

•Facebook, Google Plus, Twitter ve Orkut gibi dört sosyal platformda (örneğin sake) arkadaşlarımla bağlantı kurmamı sağlayan bir sosyal medya uygulaması tasarlamak istiyorum. Şimdi, müşterinin arkadaşının adını ve istenen platformu söyleyebilmesini istiyorum - o zaman uygulamam ona şeffaf bir şekilde bağlanmalıdır.

•Daha da önemlisi, uygulamaya daha fazla sosyal platform eklemek istersem, uygulama kodu tasarımı bozmadan buna uyum sağlamalıdır..

Strategy Patterns-Behavioral (Davranışsal)

İllüstrasyon

Yukarıdaki problemde, birden çok yolla (arkadaşla bağlantı kur) yapılabilecek bir işlemimiz var ve kullanıcı çalışma zamanında istediği yolu seçebiliyor. Bu yüzden strateji tasarım modeli için iyi bir adaydır.

Çözümü uygulamak için her seferinde bir katılımcı tasarlayalım.

- **ISocialMediaStrategy** – İşlemi özetleyen arayüz.
- **SocialMediaContext** – Uygulamayı belirleyen bağlam.
- **Uygulamalar** – Çeşitli uygulamalar: ISocialMediaStrategy. Örneğin FacebookStrategy, GooglePlusStrategy TwitterStrategy, OrkutStrategy.

Strategy Patterns-Behavioral (Davranışsal)

İllüstrasyon