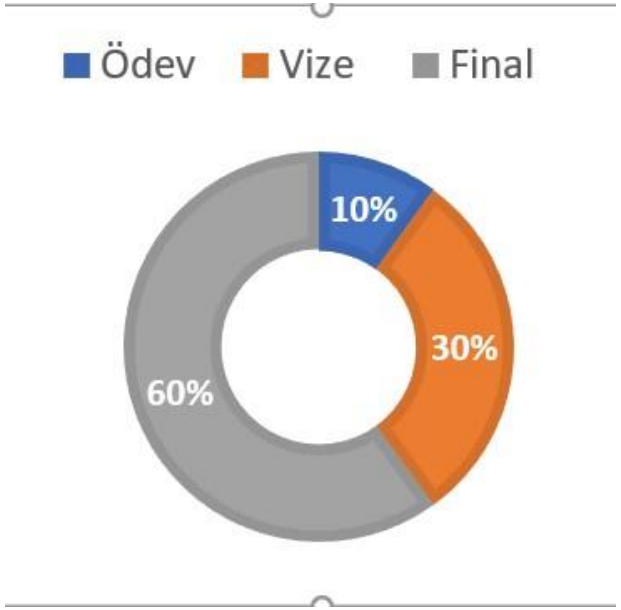
Yazılım Mühendisliği
1906003082015

Dr. Öğr. Üy. Önder EYECİOĞLU
Bilgisayar Mühendisliği

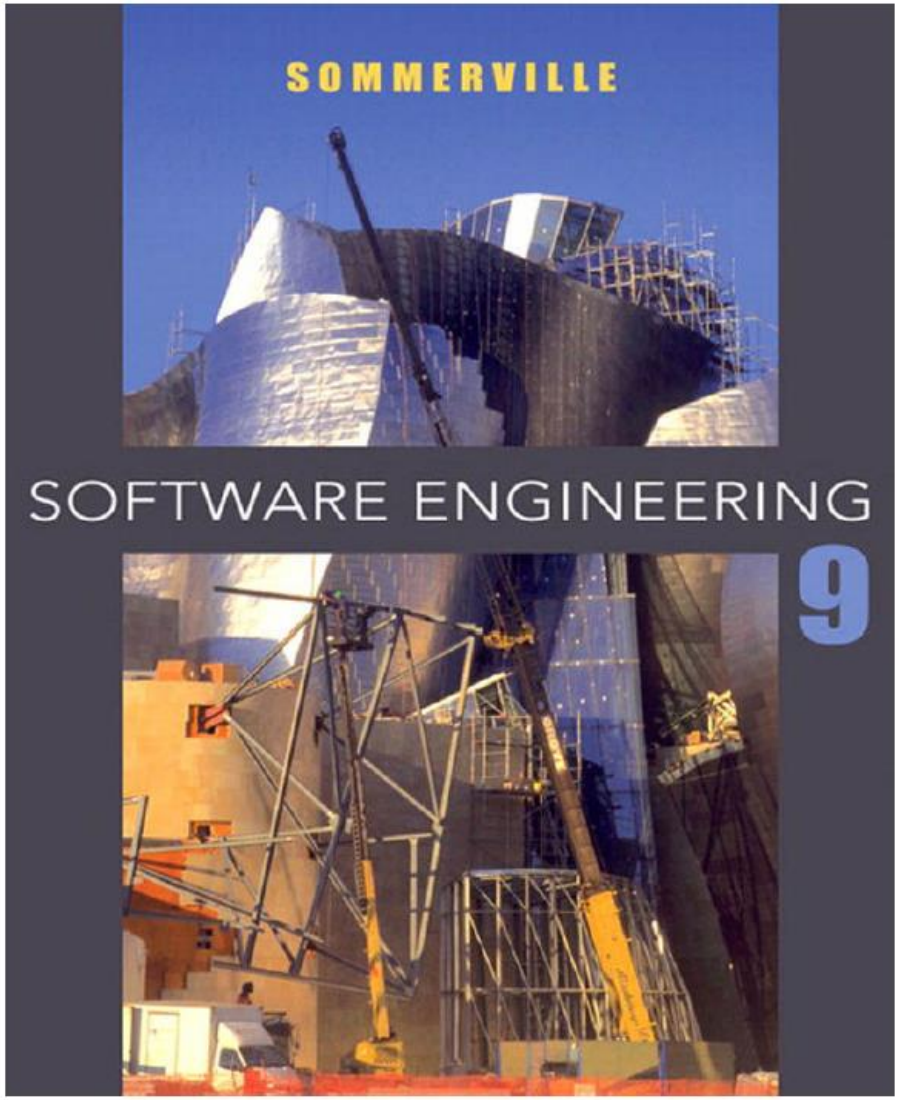
Giriş

Ders Günü ve Saati:
Salı: 09:15-13:00
Devam zorunluluğu %70



HAFTA	KONULAR
Hafta 1	Yazılım Mühendisliğine Giriş
Hafta 2	Yazılım Geliştirme Süreç Modelleri
Hafta 3	Yazılım Gereksinim Mühendisliği
Hafta 4	Yazılım Mimarisi
Hafta 5	Nesneye Yönelik Analiz ve Tasarım
Hafta 6	Laboratuar Çalışması: UML Modelleme Araçları
Hafta 7	Yazılım Test Teknikleri
Hafta 8	Ara Sınav
Hafta 9	Yazılım Kalite Yönetimi
Hafta 10	Yazılım Bakımı - Yeniden Kullanımı ve Konfigürasyon Yönetimi
Hafta 11	Yazılım Proje Yönetimi (Yazılım Ölçümü ve Yazılım Proje Maliyet Tahmin Yöntemleri)
Hafta 12	Yazılım Proje Yönetimi (Yazılım Risk Yönetimi)
Hafta 13	Çevik Yazılım Geliştirme Süreç Modelleri
Hafta 14	Yazılım Süreci İyileştirme, Yeterlilik Modeli (CMM)

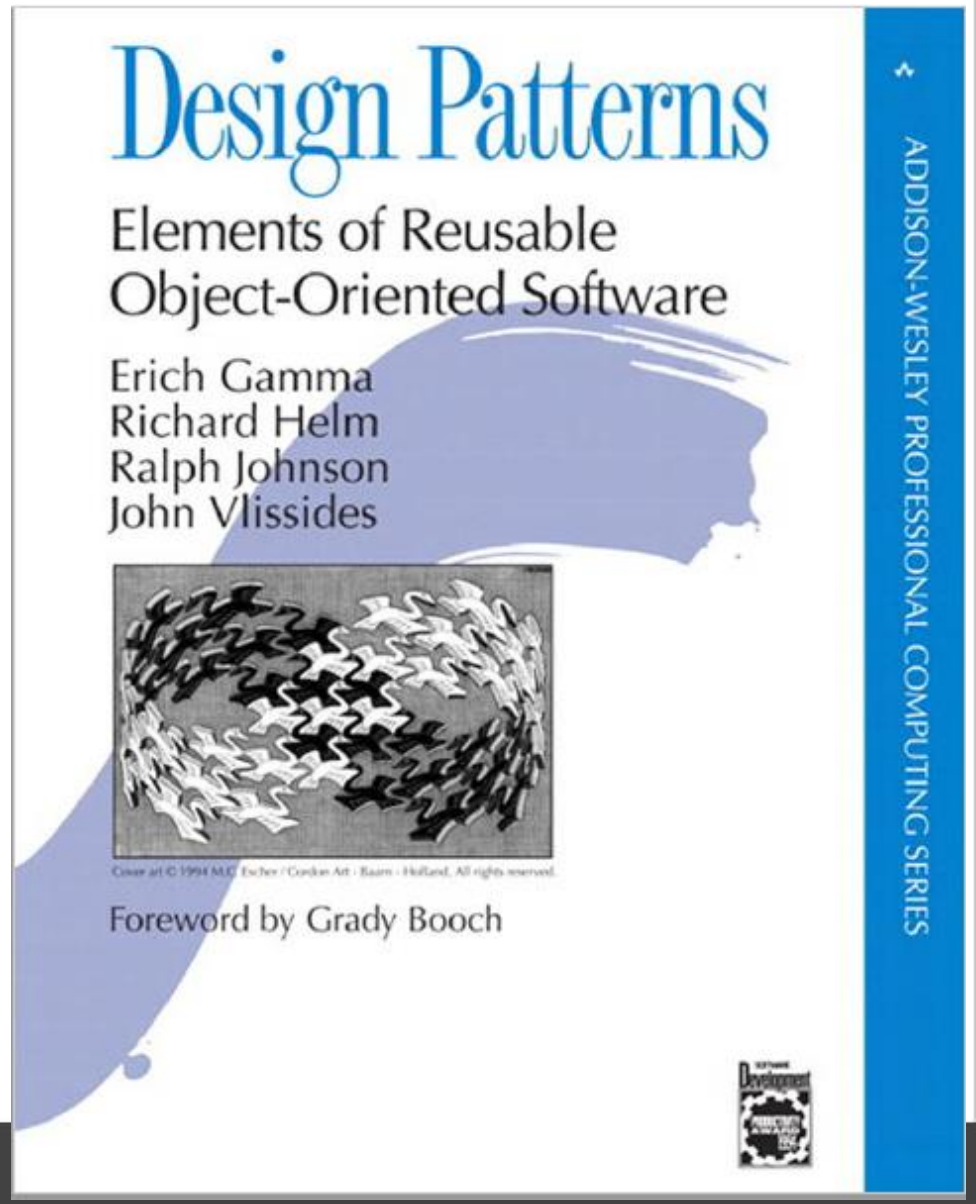
Kaynaklar



12.
Tasarım Desenleri

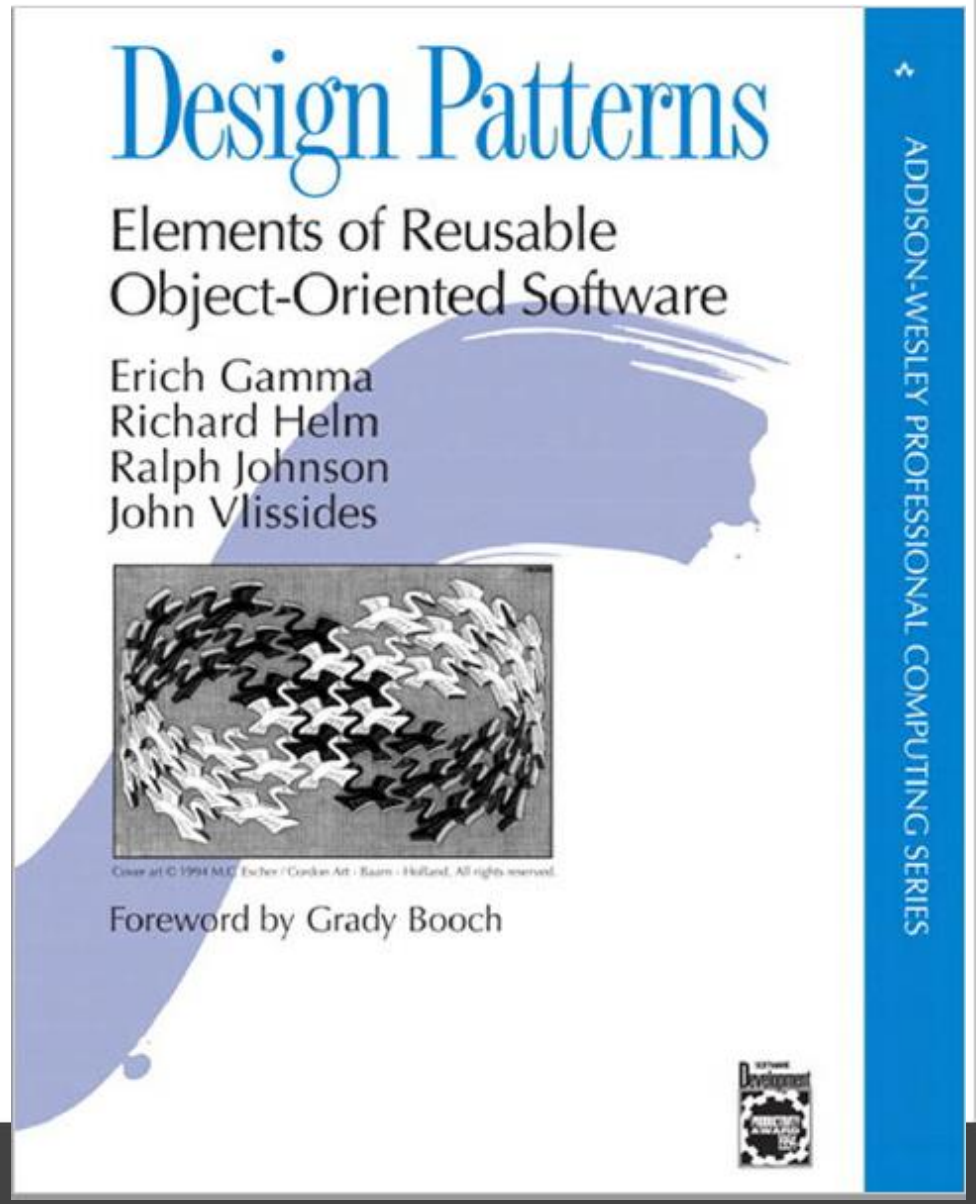
Tasarım Desenleri

- Yazılım geliştirmede en önemli adımlardan birisi "tasarım"dır.
- Nesne yönelimli yazılım tasarlamak zordur ve yeniden kullanılabilir nesne yönelimli yazılım tasarlamak daha da zordur.
- İlgili nesneleri bulmalı, bunları doğru ayrıntı düzeyinde sınıflara ayırmalı, sınıf arabirimlerini ve miras hiyerarşilerini tanımlamalı ve bunlar arasında anahtar ilişkiler kurmalısınız.



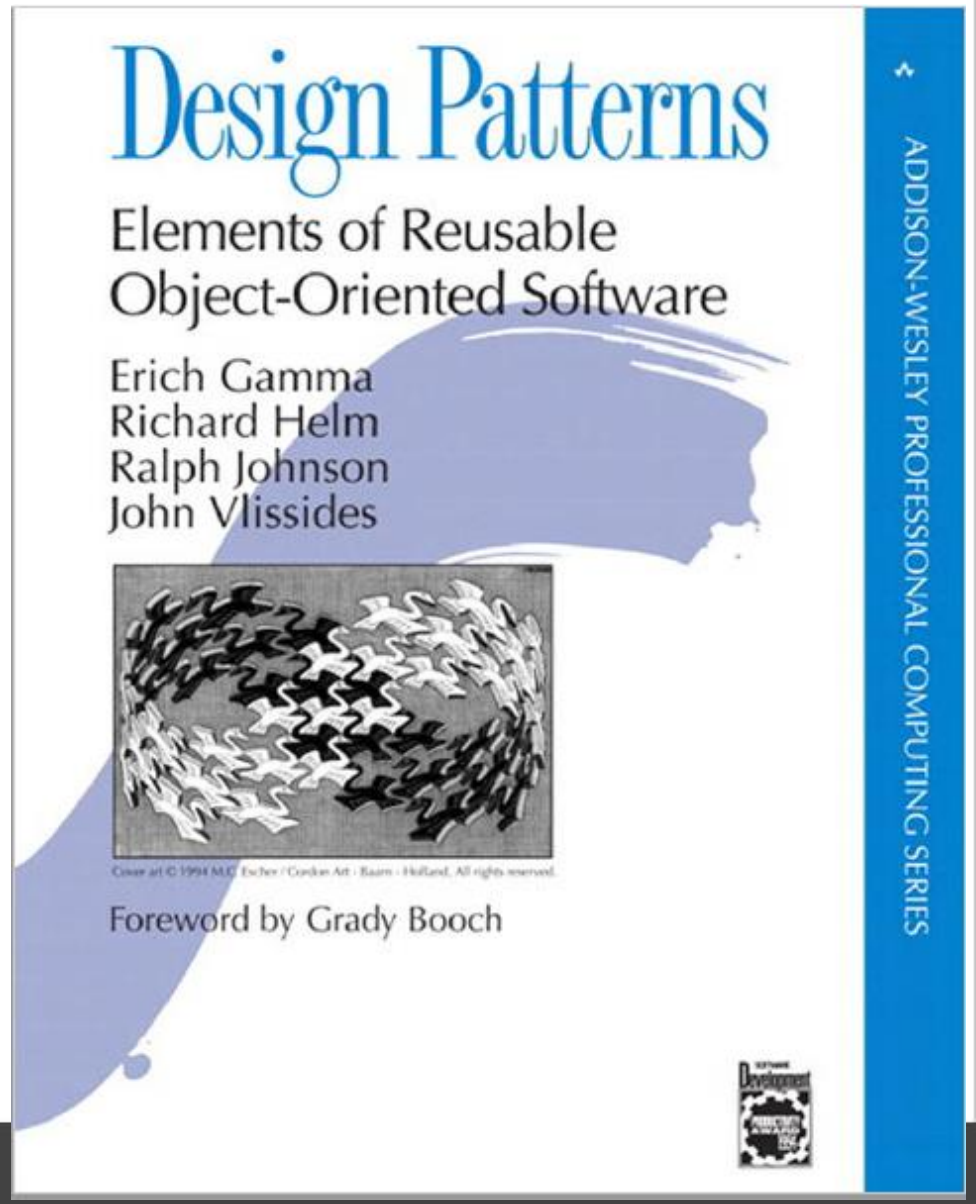
Tasarım Desenleri

- Tasarımınız, eldeki soruna özel olmalı, aynı zamanda gelecekteki sorunları ve gereksinimleri ele alacak kadar genel olmalıdır.
- Ayrıca yeniden tasarlamaktan kaçınmak veya en azından en aza indirmek istersiniz.
- Birçok nesne yönelimli sistemde yinelenen sınıf kalıpları ve iletişim nesneleri bulunur.
- Bu modeller belirli tasarım sorunlarını çözer ve nesne yönelimli tasarımları daha esnek, zarif ve nihayetinde yeniden kullanılabilir hale getirir.



Tasarım Desenleri

- Yazılım tasarımında bazı genel problemler tekrar tekrar önümüze gelir ve bunlar için benzer tasarım çözümleri tekrar tekrar üretilmek zorunda kalınır.
- Tasarım kalıpları, başarılı tasarımların ve mimarilerin yeniden kullanılmasını kolaylaştırır. Kanıtlanmış teknikleri tasarım kalıpları olarak ifade etmek, onları yeni sistem geliştiricileri için daha erişilebilir hale getirir.
- Tasarım örüntüleri, bu genel problemlere iyi veya kabul edilmiş genel çözümler önerir.

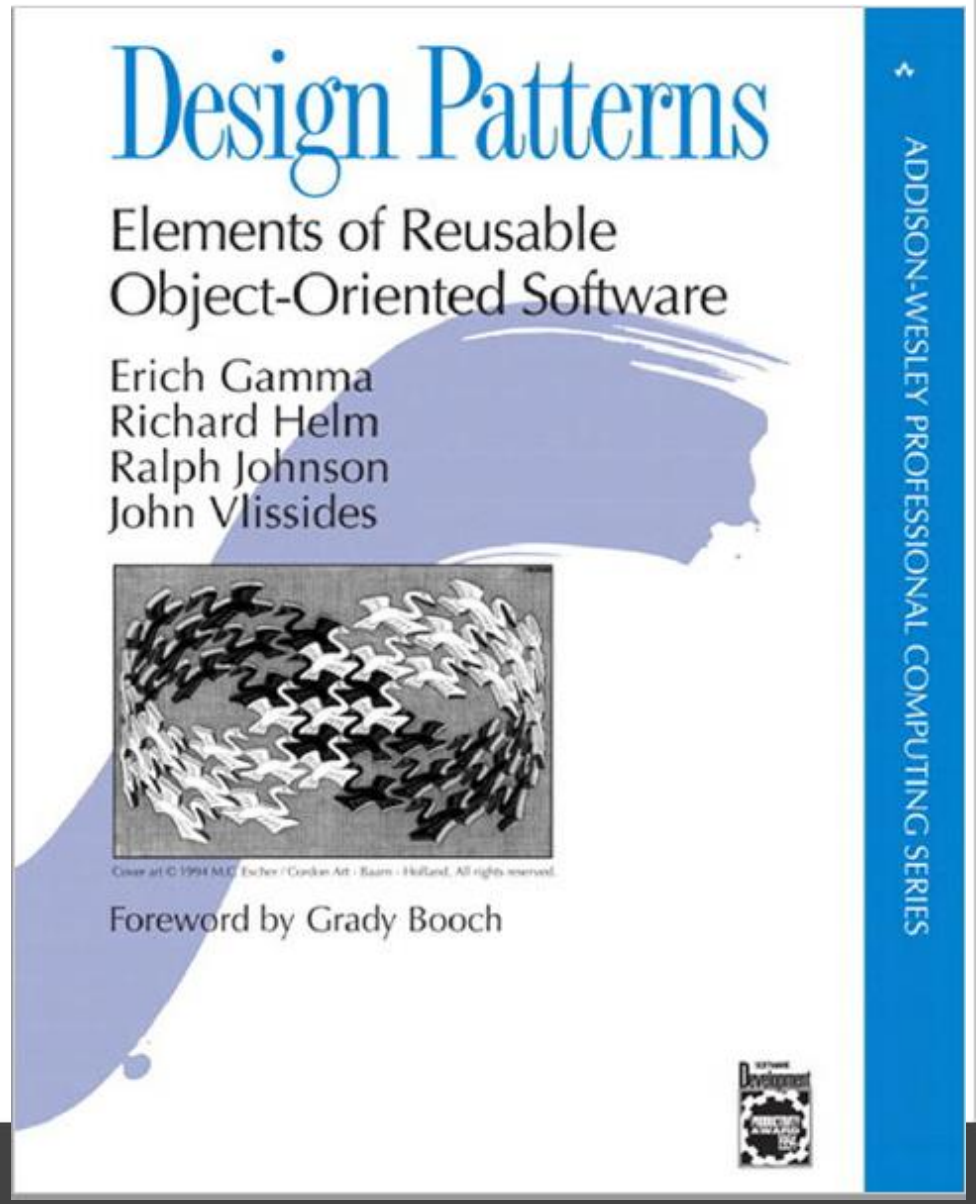


Tasarım Desenleri

- Tasarım kalıpları, sınıf ve nesne etkileşimlerinin ve bunların altında yatan amaçların açık bir spesifikasyonunu sağlayarak mevcut sistemlerin dokümantasyonunu ve bakımını iyileştirebilir.

Her desen, çevremizde tekrar tekrar ortaya çıkan bir sorunu açıklar ve daha sonra bu soruna çözümün uygulanmasını, bu çözümü iki kez aynı şekilde yapmadan milyonlarca kez kullanabileceğiniz şekilde tanımlar.

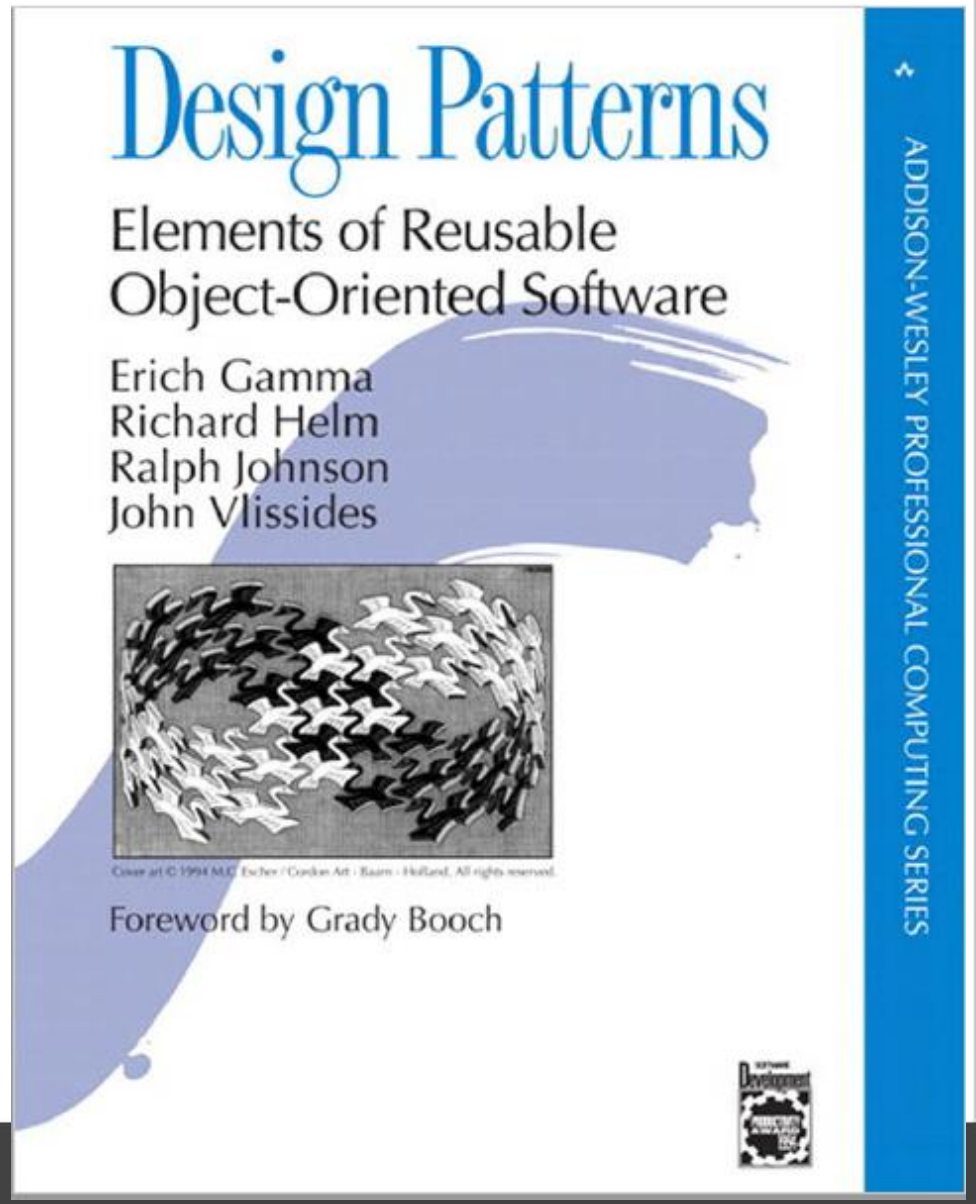
Christopher Alexander



Tasarım Desenleri Nedir?

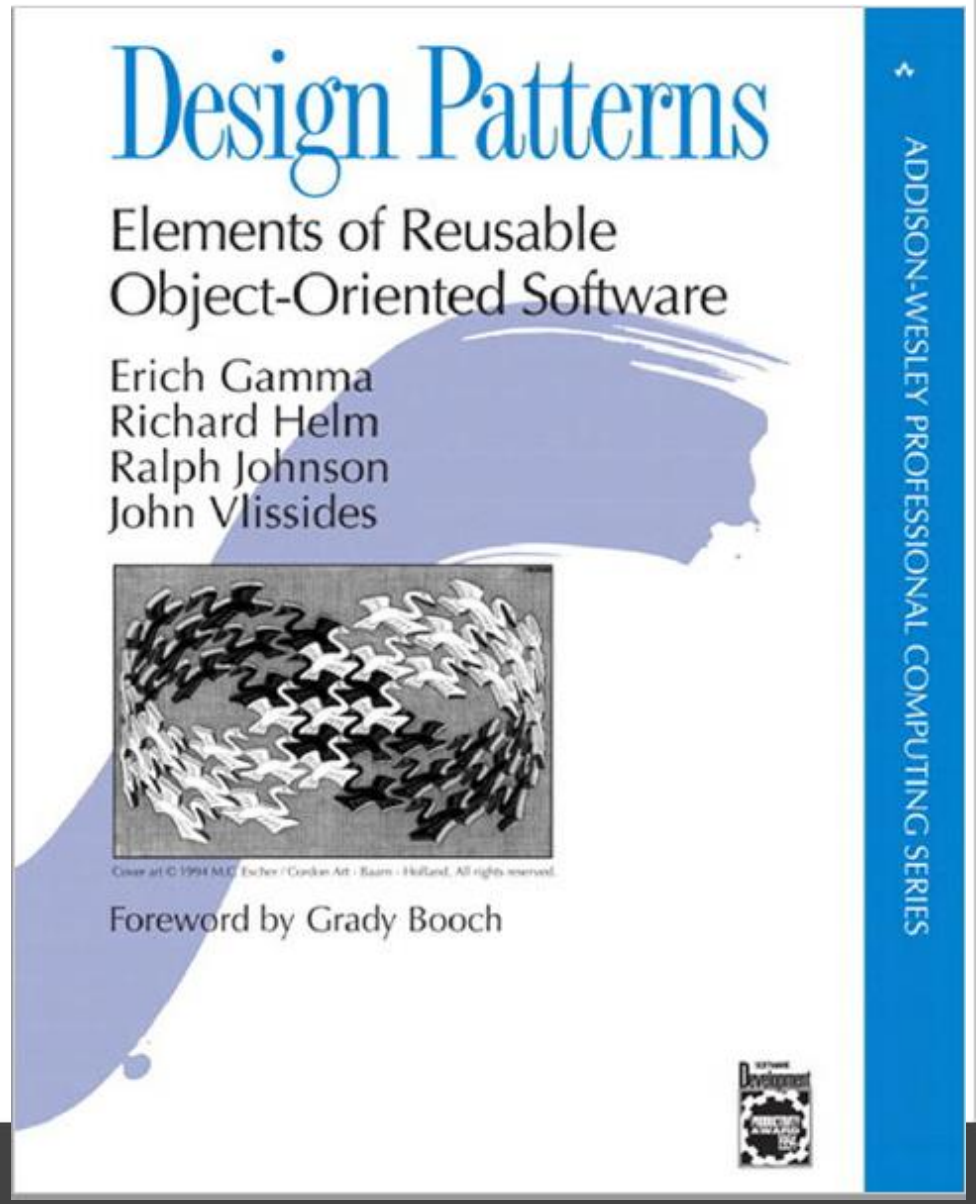
Tasarım desenleri; yazılım tasarımında, problemlerde karşımıza sıkça çıkan ortak sorunları çözmek için oluşturulmuş desenlerdir. Tasarım desenleri, yazılım sürecinde uygulanan çözümlerin esnekliği ve tekrar kullanılabilirliği ile de ilgilenmektedir.

Tasarım desenleri , yazılım tasarımında yaygın olarak ortaya çıkan sorunlara tipik çözümlerdir. Kodunuzda yinelenen bir tasarım sorununu çözmek için özelleştirebileceğiniz önceden hazırlanmış planlar gibidirler.



Tasarım Desenleri Nedir?

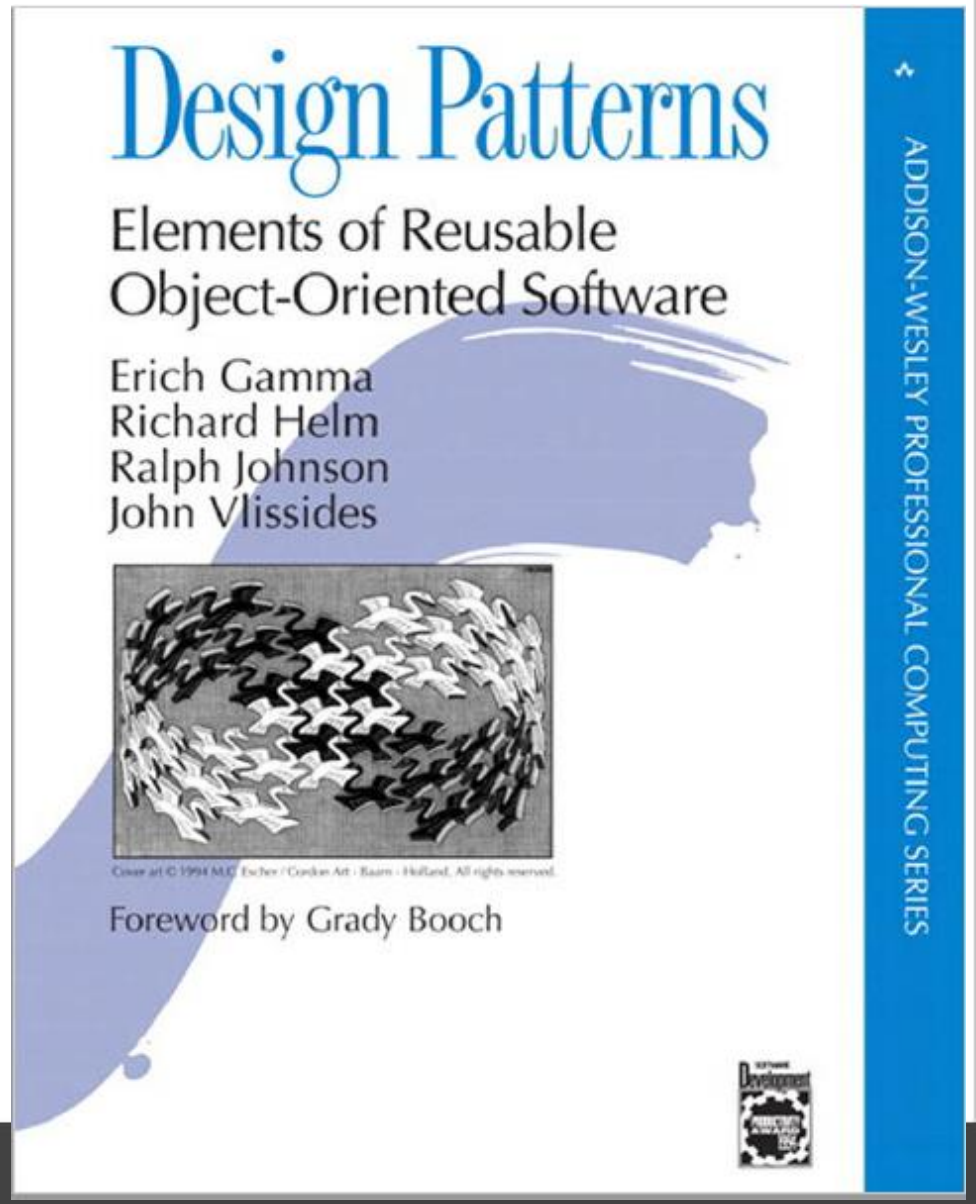
1. Sıklıkla ortaya çıkan sorunlara çözüm bulmak için tasarım kalıplarını bilmemiz gerekir. Ve gelecekte benzer bir durumla karşılaştığımızda bu çözümler yeniden kullanılabilir.
2. Bunlar, birçok farklı durumdaki çözümlere yönelik bir tür şablondur.
3. Başka bir deyişle, bunlar farklı nesnelerin ve ilgili sınıflarının belirli bir bağlamda bir tasarım problemini nasıl çözdüğünün açıklamalarıdır.



Tasarım Desenleri Nedir?

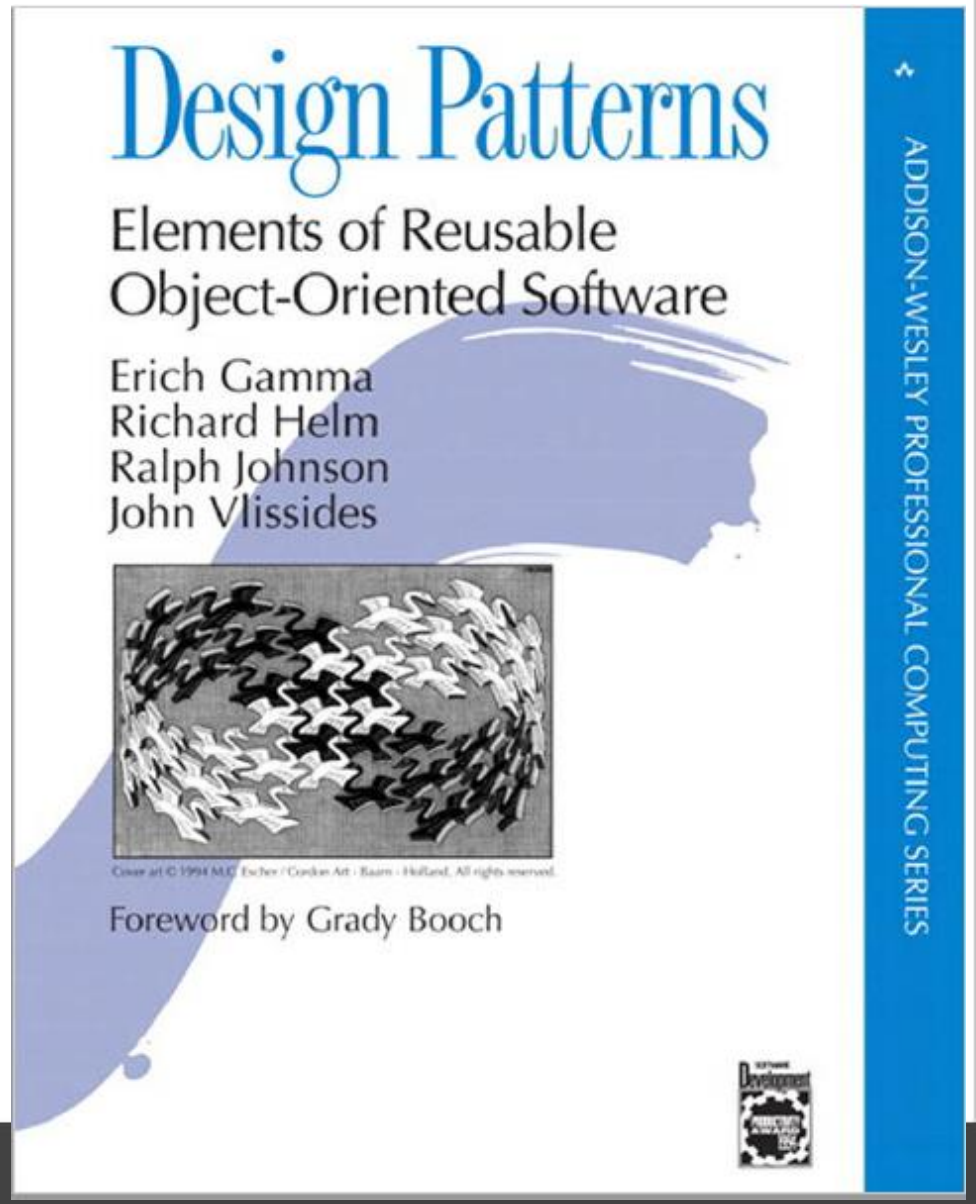
Genel olarak, bir Desenin dört temel unsuru vardır:

1. Desen adı, bir tasarım problemini, çözümlerini ve sonuçlarını bir veya iki kelimeyle tanımlamak için kullanabileceğimiz bir tutamaçtır. Bir kalıba isim vermek, tasarım kelime dağarcığımızı hemen arttırır. Daha yüksek bir soyutlama düzeyinde tasarlamamızı sağlar. Desenler için bir kelime dağarcığına sahip olmak, onlar hakkında meslektaşlarımızla, belgelerimizde ve hatta kendimizle konuşmamızı sağlar. Tasarımlar hakkında düşünmeyi ve bunları ve bunların takaslarını başkalarına iletmeyi kolaylaştırır.



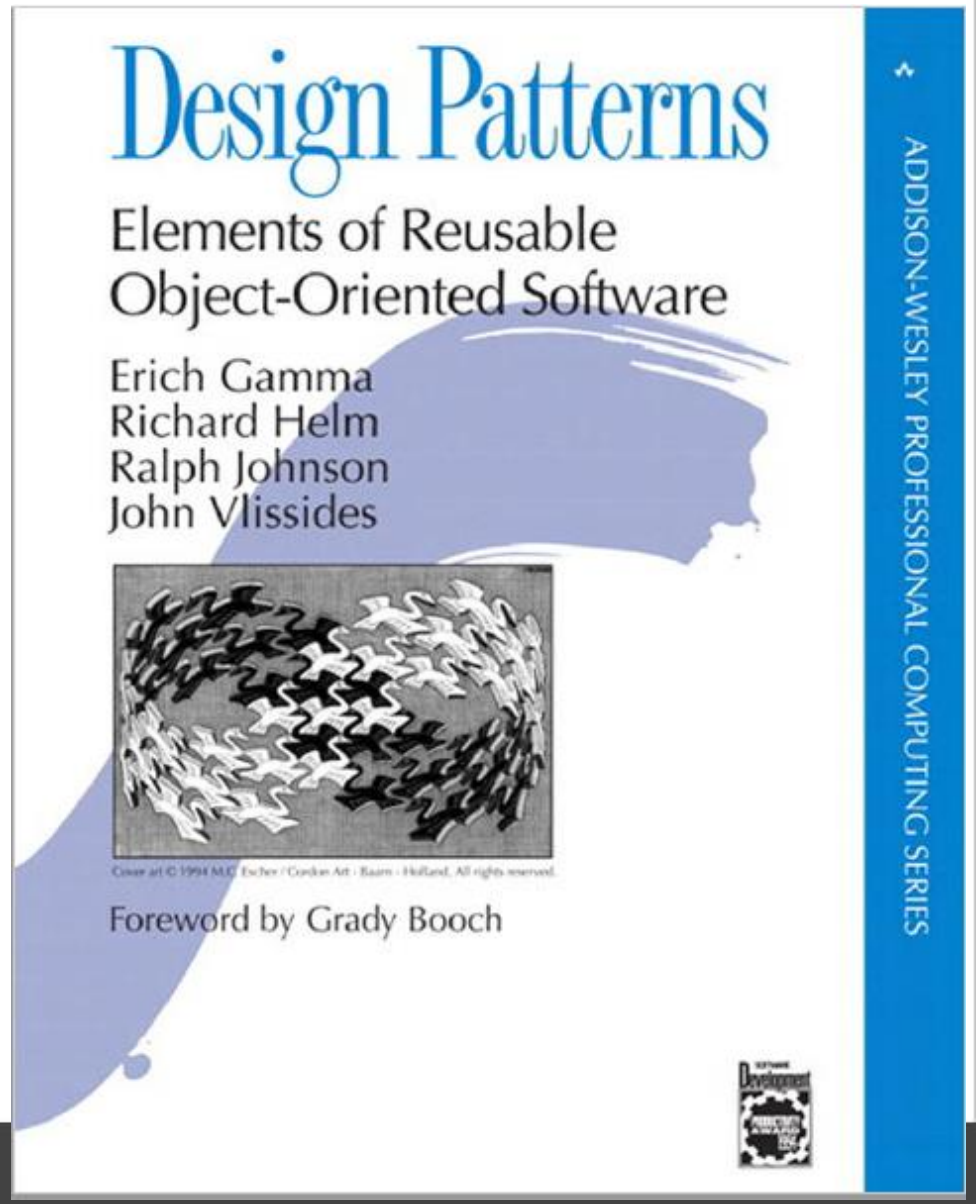
Tasarım Desenleri Nedir?

2. Sorun, desenin ne zaman uygulanacağını açıklar. Sorunu ve bağlamını açıklar. Algoritmaların nesneler olarak nasıl temsil edileceği gibi belirli tasarım problemlerini tanımlayabilir. Esnek olmayan bir tasarımın belirtisi olan sınıf veya nesne yapılarını tanımlayabilir. Bazen problem, kalıbı uygulamak için mantıklı hale gelmeden önce karşılanması gereken koşulların bir listesini içerecektir.



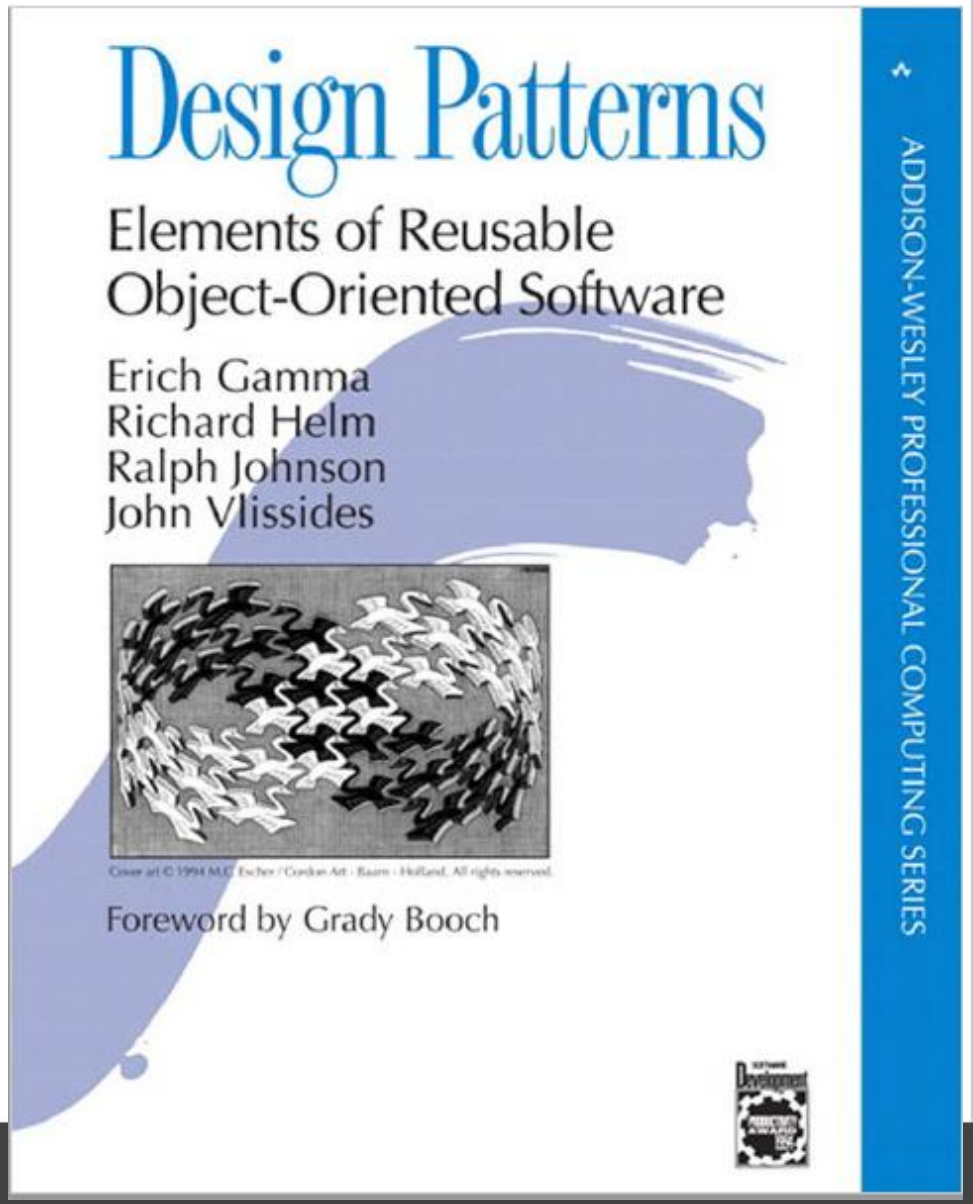
Tasarım Desenleri Nedir?

3. Çözüm, tasarımı oluşturan öğeleri, bunların ilişkilerini, sorumluluklarını ve işbirliklerini tanımlar. Çözüm, belirli bir somut tasarımı veya uygulamayı tanımlamaz, çünkü bir Desen, birçok farklı durumda uygulanabilen bir şablon gibidir. Bunun yerine, Desen, bir tasarım probleminin soyut bir tanımını ve genel bir element düzenlemesinin (bizim durumumuzda sınıflar ve nesneler) onu nasıl çözdüğünü sağlar.



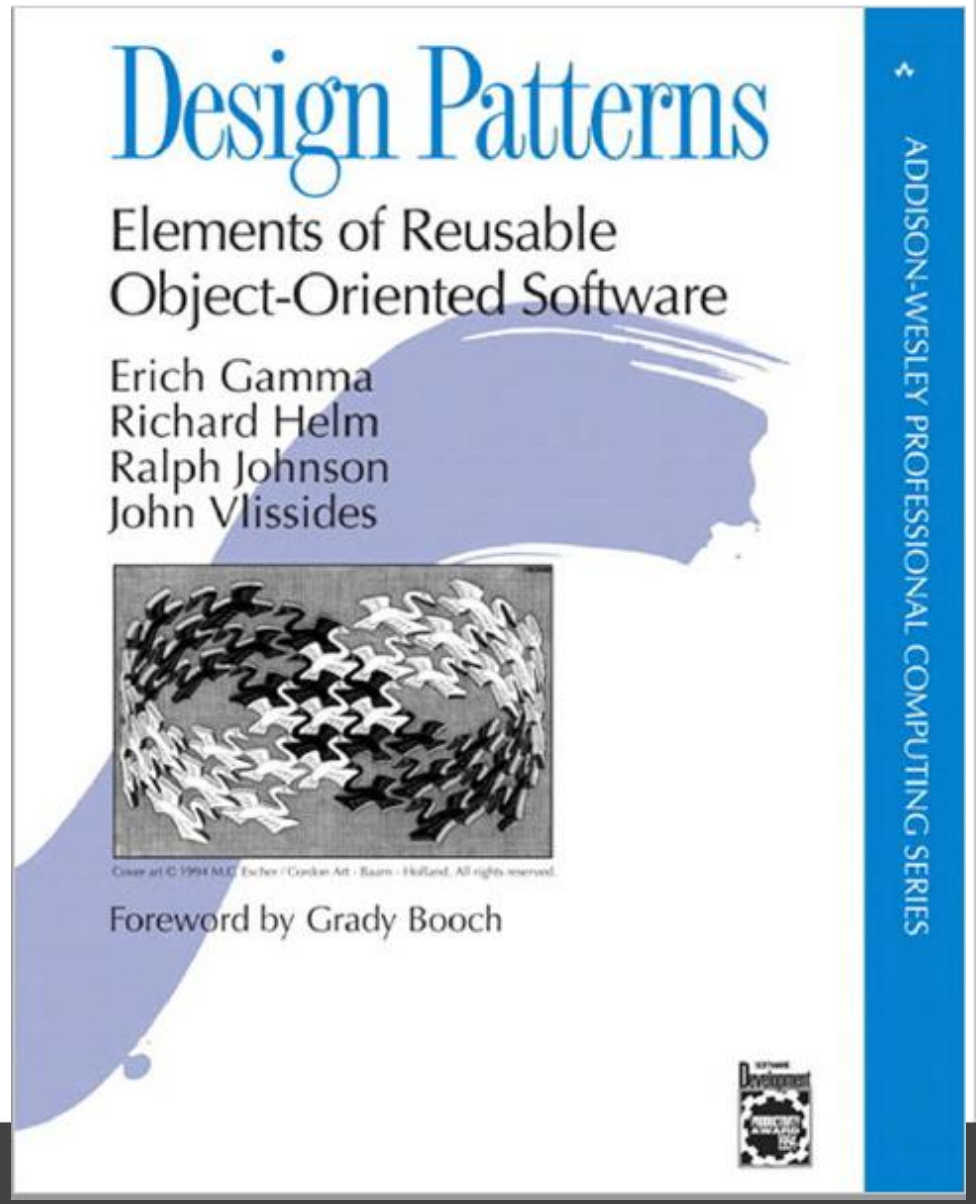
Tasarım Desenleri Nedir?

4. Sonuçlar, kalıbı uygulamanın sonuçları ve takaslarıdır. Tasarım kararlarını tanımladığımızda sonuçlar genellikle dile getirilmese de, tasarım alternatiflerini değerlendirmek ve Deseni uygulamanın maliyet ve faydalarını anlamak için kritik öneme sahiptirler. Yazılımın sonuçları genellikle yer ve zaman değiş tokuşu ile ilgilidir. Dil ve uygulama konularını da ele alabilirler. Yeniden kullanım genellikle nesne yönelimli tasarımda bir faktör olduğundan, bir kalıbın sonuçları sistemin esnekliği, genişletilebilirliği veya taşınabilirliği üzerindeki etkisini içerir. Bu sonuçları açıkça listelemek, onları anlamanıza ve değerlendirmenize yardımcı olur.



Tasarım Desenleri Nedir?

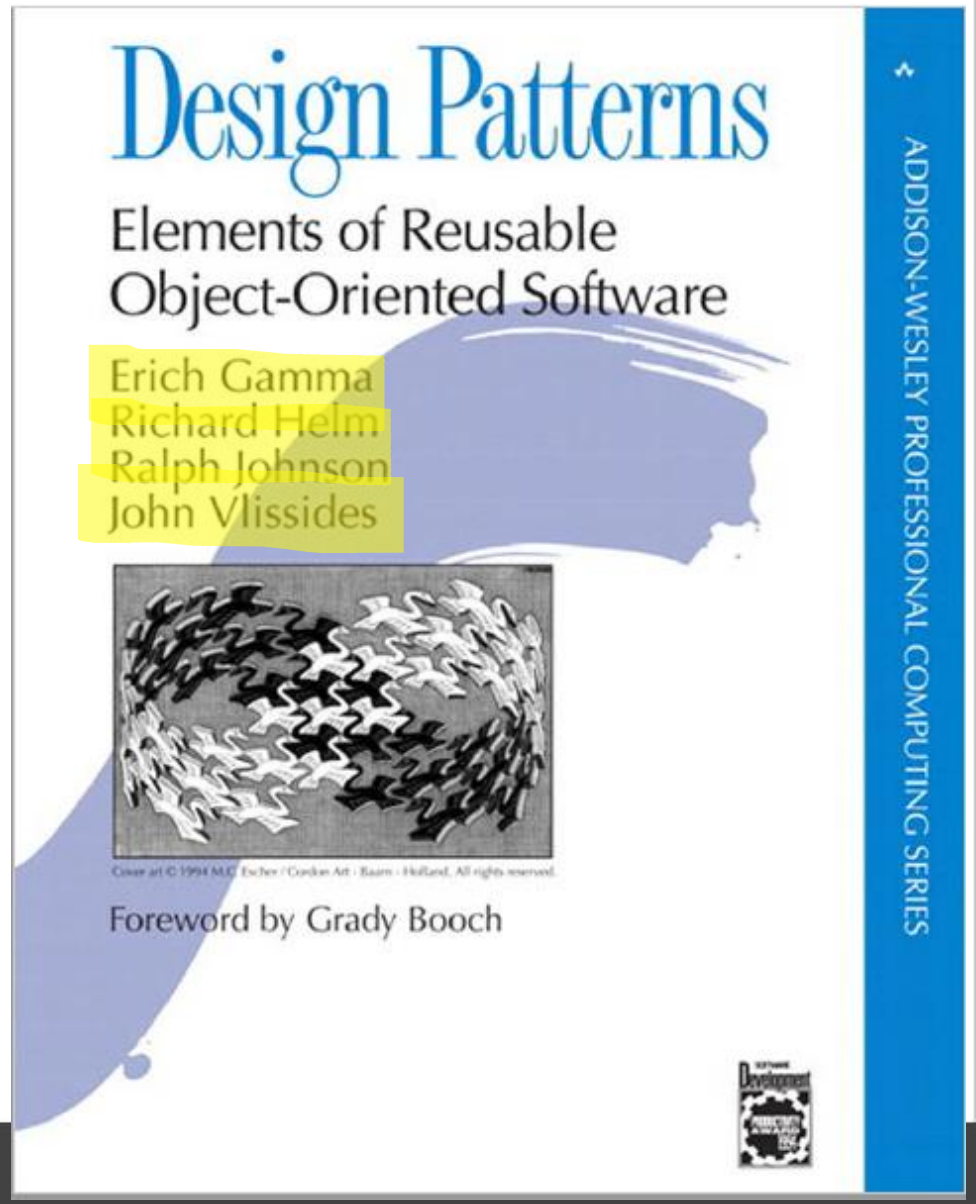
- Bir tasarım deseni, yeniden kullanılabilir bir nesne yönelimli tasarım oluşturmak için onu faydalı kılan ortak bir tasarım yapısının anahtar yönlerini adlandırır, özetler ve tanımlar.
- Tasarım deseni, katılan sınıfları ve örnekleri, rollerini ve işbirliklerini ve sorumlulukların dağıtımını tanımlar.
- Her tasarım deseni, belirli bir nesne yönelimli tasarım sorununa veya konusuna odaklanır.
- Ne zaman uygulanacağını, diğer tasarım kısıtlamaları açısından uygulanıp uygulanamayacağını ve kullanımının sonuçlarını ve değiş tokuşlarını açıklar.



Tasarım Desenleri Nedir?

Burada Gang of Four tarafından özetlenen 23 tasarım modeli anlatılacak.

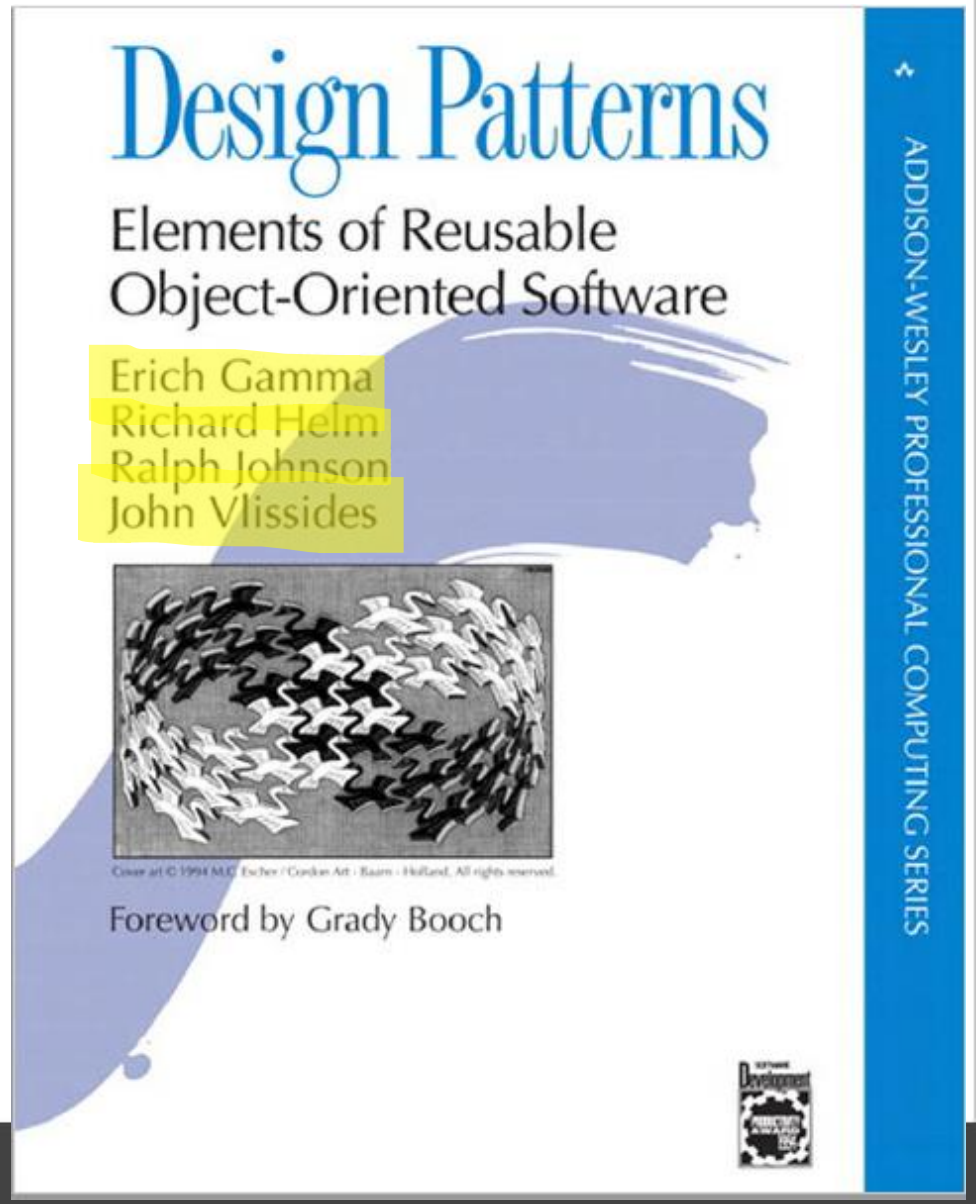
GoF yazarları, C++ kullanarak büyük ölçekli kurumsal sistemleri kodlama konusundaki deneyimleriyle ortak kalıpların ortaya çıktığını gördüler. Bu tasarım desenleri C++'a özgü değildir. Tasarım desenleri herhangi bir nesne yönelimli dilde uygulanabilir.



Tasarım Desenleri Tanımlanması

Tasarım kalıplarını, Tasarım sürecinin son ürününü sınıflar ve nesneler arasındaki ilişkiler olarak yakalarlar.

Tutarlı bir format kullanarak tasarım deseni açıklıyoruz. Her desen aşağıdaki şablona göre bölümlere ayrılmıştır.



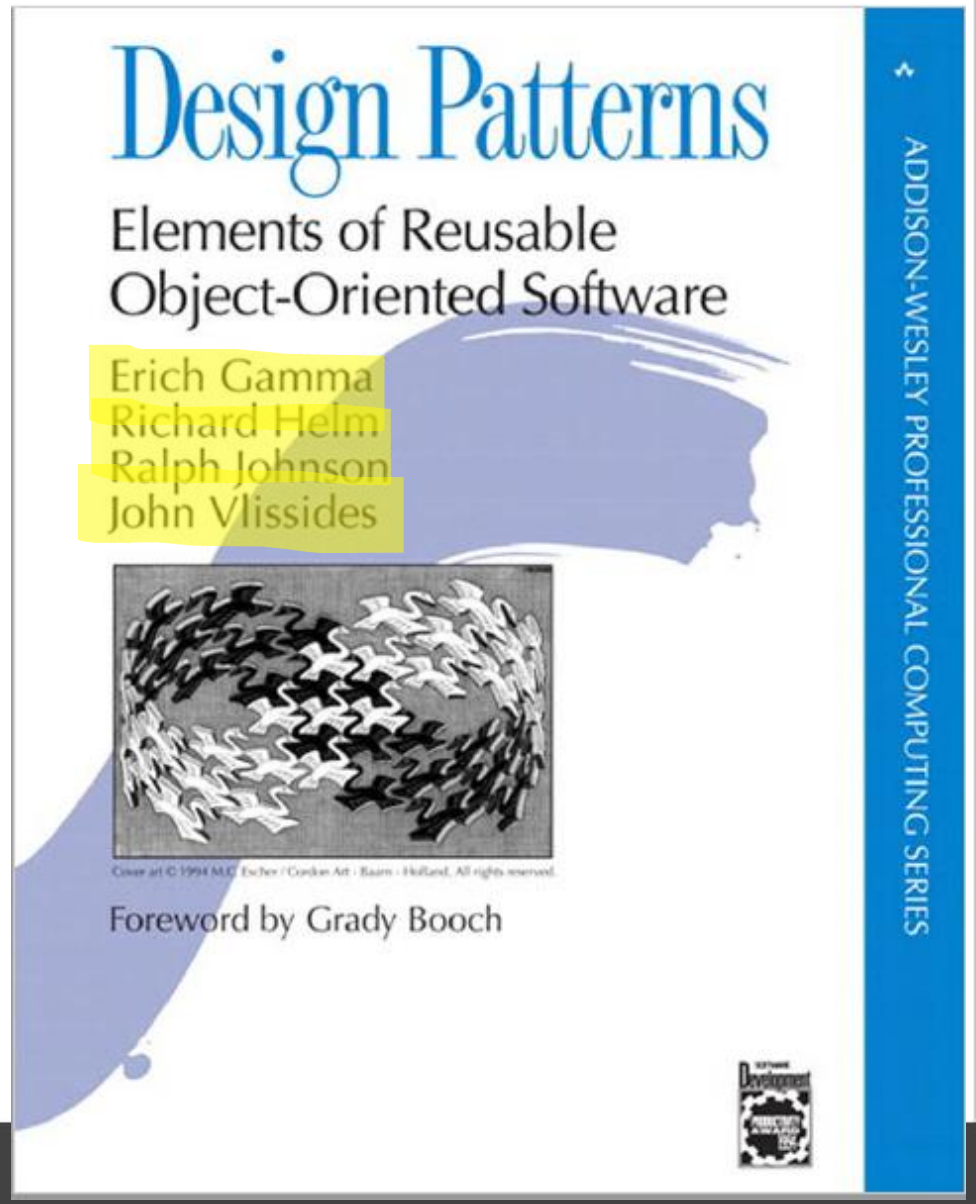
Tasarım Desenleri Tanımlanması

Model Adı ve Sınıflandırması

Desenin adı, desenin özünü kısa ve öz bir şekilde aktarır. İyi bir isim çok önemlidir, çünkü tasarım kelime dağarcığınızın bir parçası olacaktır. Modelin sınıflandırması, verilecektir.

Niyet (intent)

Aşağıdaki soruları yanıtlayan kısa bir açıklama: Tasarım deseni ne işe yarar? Gerekçesi ve amacı nedir? Hangi özel tasarım sorununu veya sorununu ele alıyor?



Tasarım Desenleri Tanımlanması

Ayrıca şöyle bilinir (also known as)

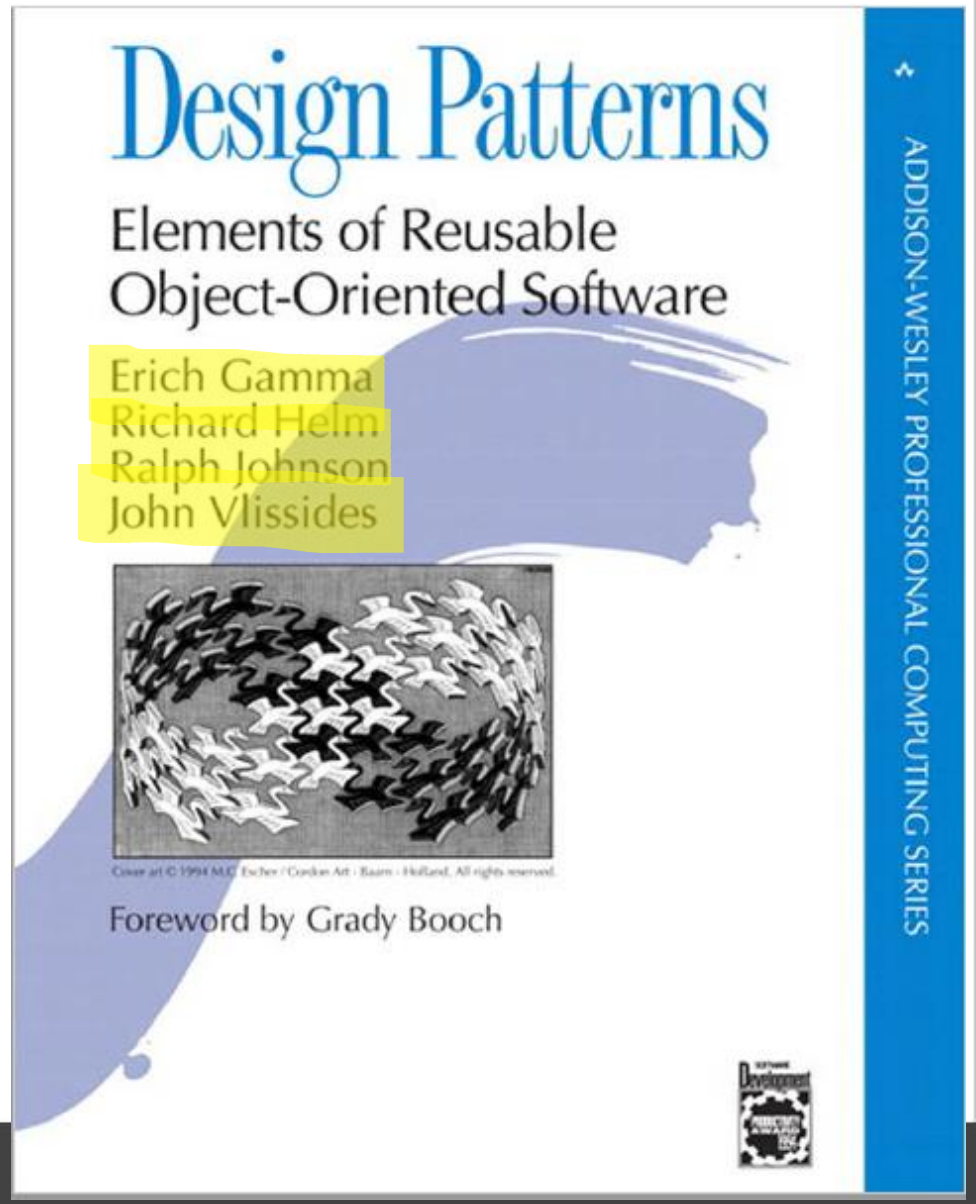
Varsa, desen için diğer iyi bilinen isimler.

Motivasyon

Bir tasarım problemini ve modeldeki sınıf ve nesne yapılarının problemi nasıl çözdüğünü gösteren bir senaryo. Senaryo, aşağıdaki kalıbın daha soyut açıklamasını anlamanıza yardımcı olacaktır.

Uygulanabilirlik

Tasarım deseninin uygulanabileceği durumlar nelerdir?
Desenin ele alabileceği zayıf tasarım örnekleri nelerdir?
Bu durumları nasıl tanıyabilirsiniz?



Tasarım Desenleri Tanımlanması

Yapı

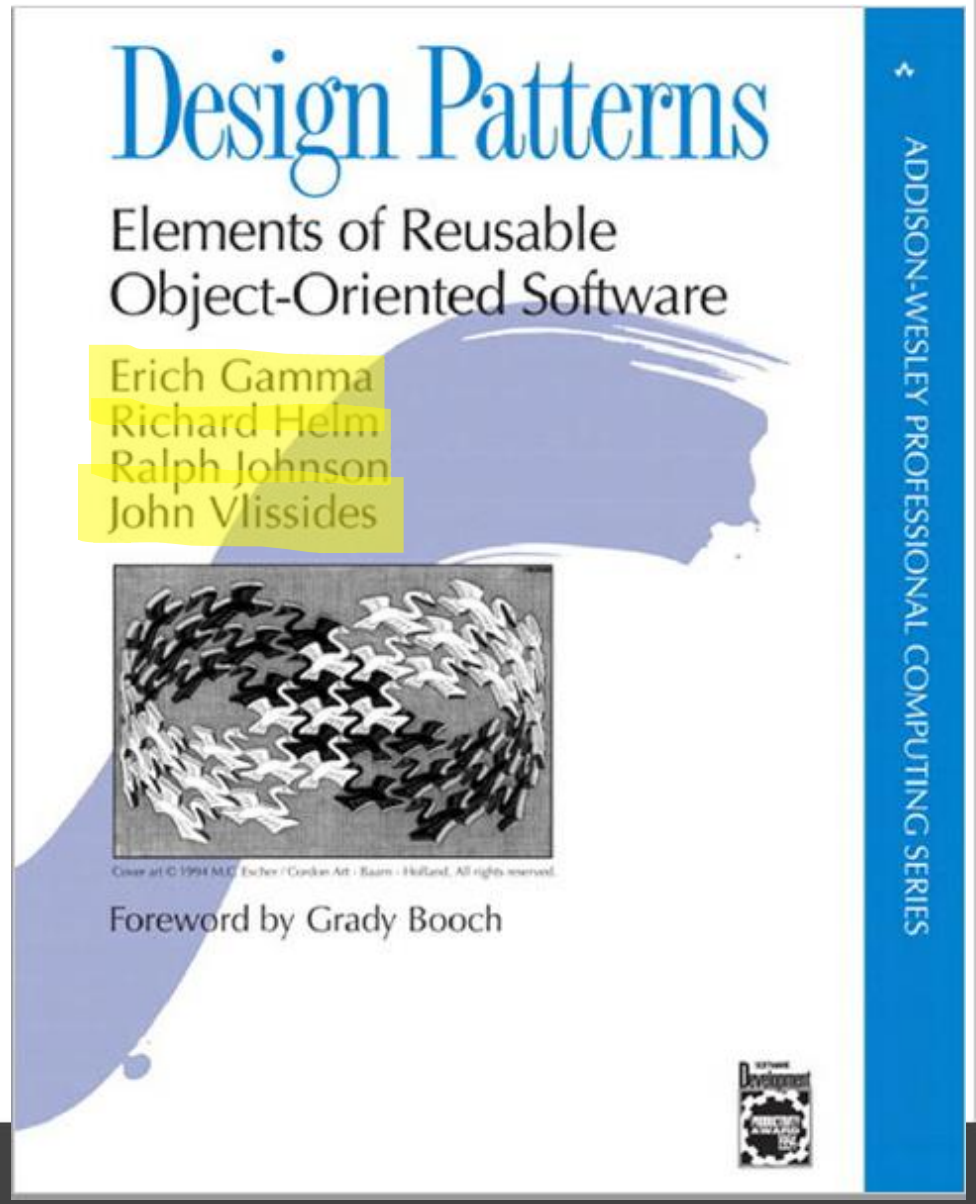
Nesne Modelleme Tekniğine (OMT) dayalı bir notasyon kullanılarak desendeki sınıfların grafiksel temsili. Ayrıca nesneler arasındaki istek ve işbirliği dizilerini göstermek için etkileşim diyagramlarını kullanırız.

Katılımcılar

Tasarım desenine katılan sınıflar ve/veya nesneler ve sorumlulukları.

İşbirlikleri

Katılımcılar sorumluluklarını yerine getirmek için nasıl işbirliği yaparlar.



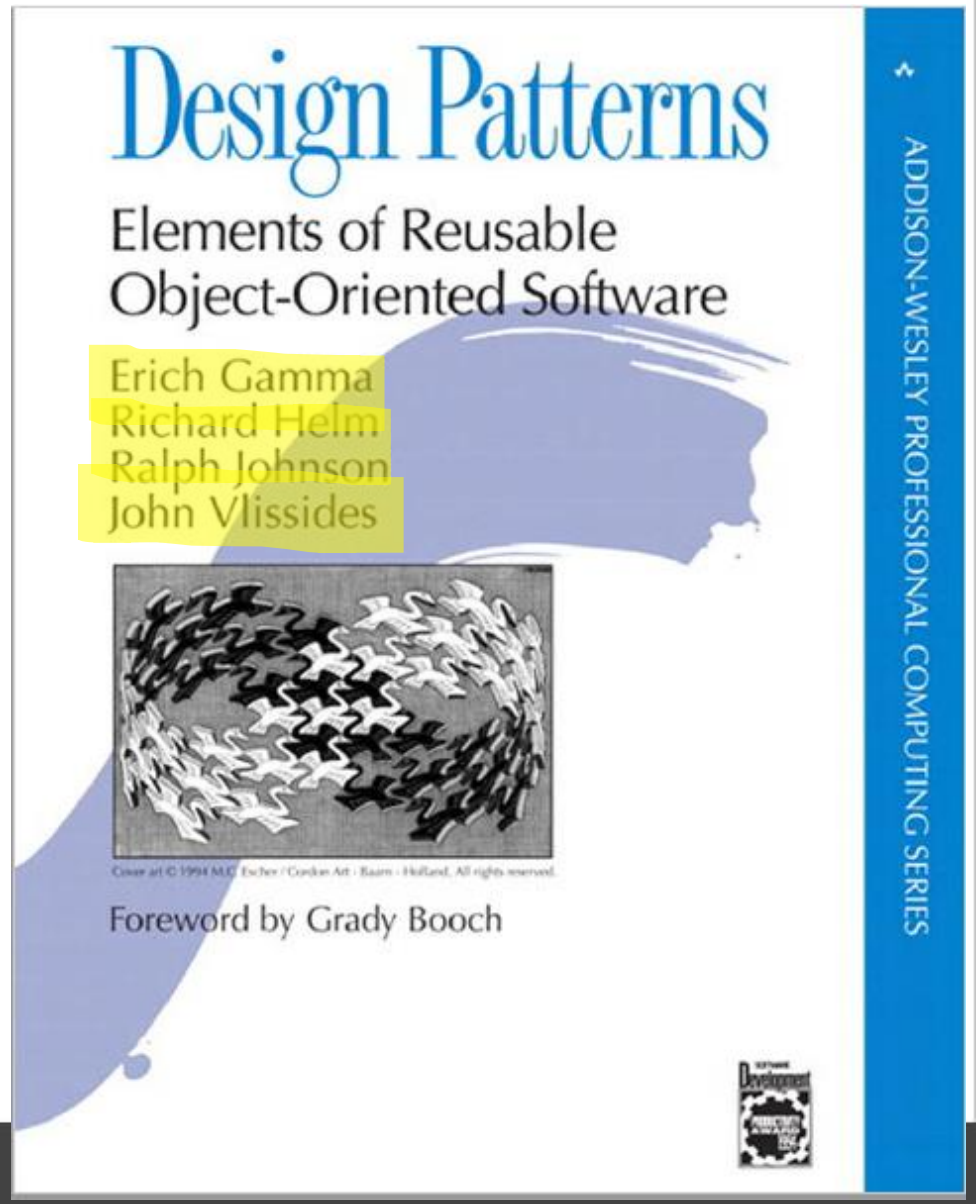
Tasarım Desenleri Tanımlanması

Sonuçlar

Model hedeflerini nasıl destekliyor? Modeli kullanmanın ödünleşimleri ve sonuçları nelerdir? Sistem yapısının hangi yönü bağımsız olarak değişiklik yapmanıza izin veriyor?

Uygulama (implementation)

Modeli uygularken hangi tuzakların, ipuçlarının veya tekniklerin farkında olmalısınız? Dile özgü sorunlar var mı?



Tasarım Desenleri Tanımlanması

Basit kod

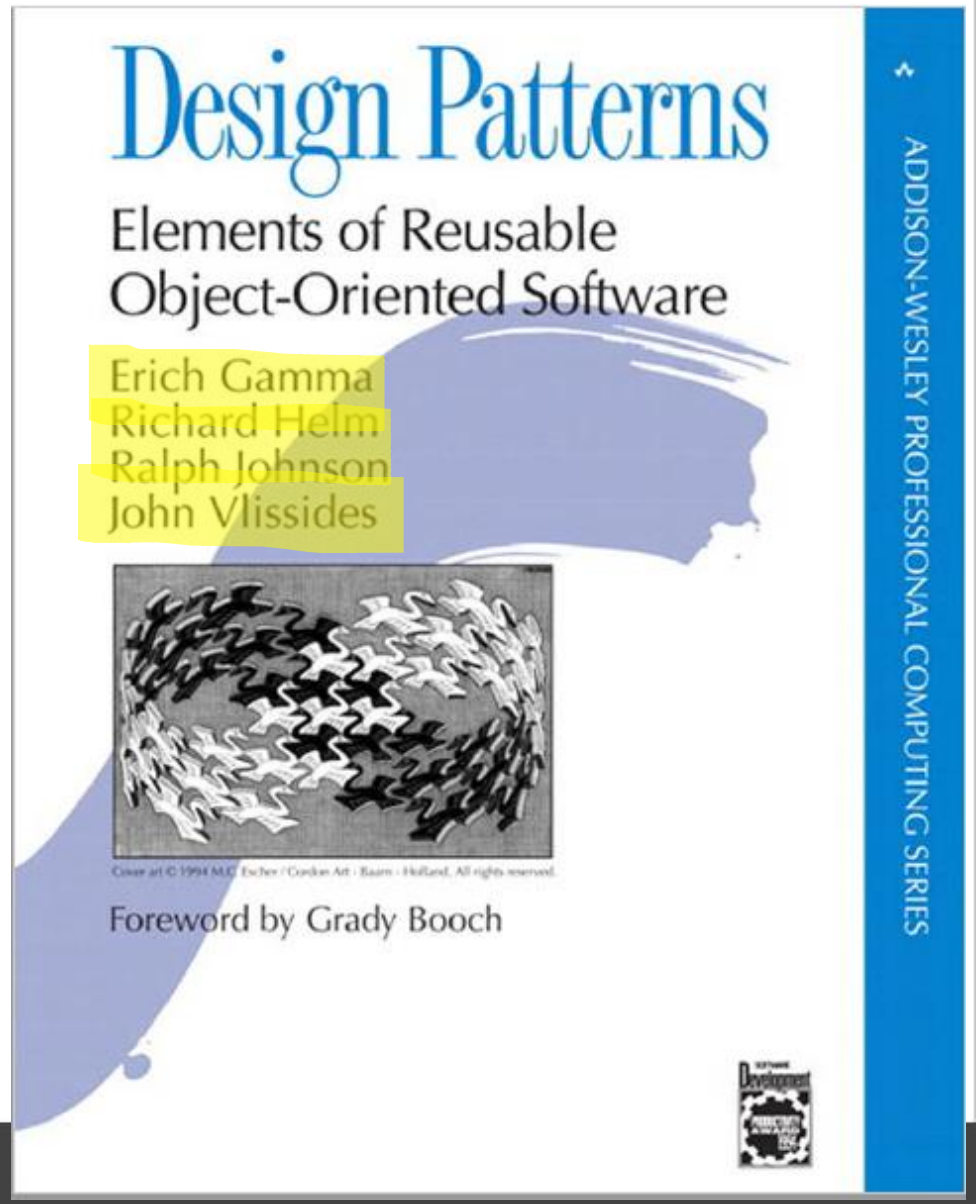
Kalıbı C++ (Java) nasıl uygulayabileceğinizi gösteren kod parçaları.

Bilinen Kullanımlar

Gerçek sistemlerde bulunan model örnekleri. Farklı alanlardan en az iki örnek ekledik.

İlgili Desenler

Hangi tasarım desenleri bununla yakından ilişkilidir? Önemli farklar nelerdir? Bu başka hangi desenlerle kullanılmalıdır?

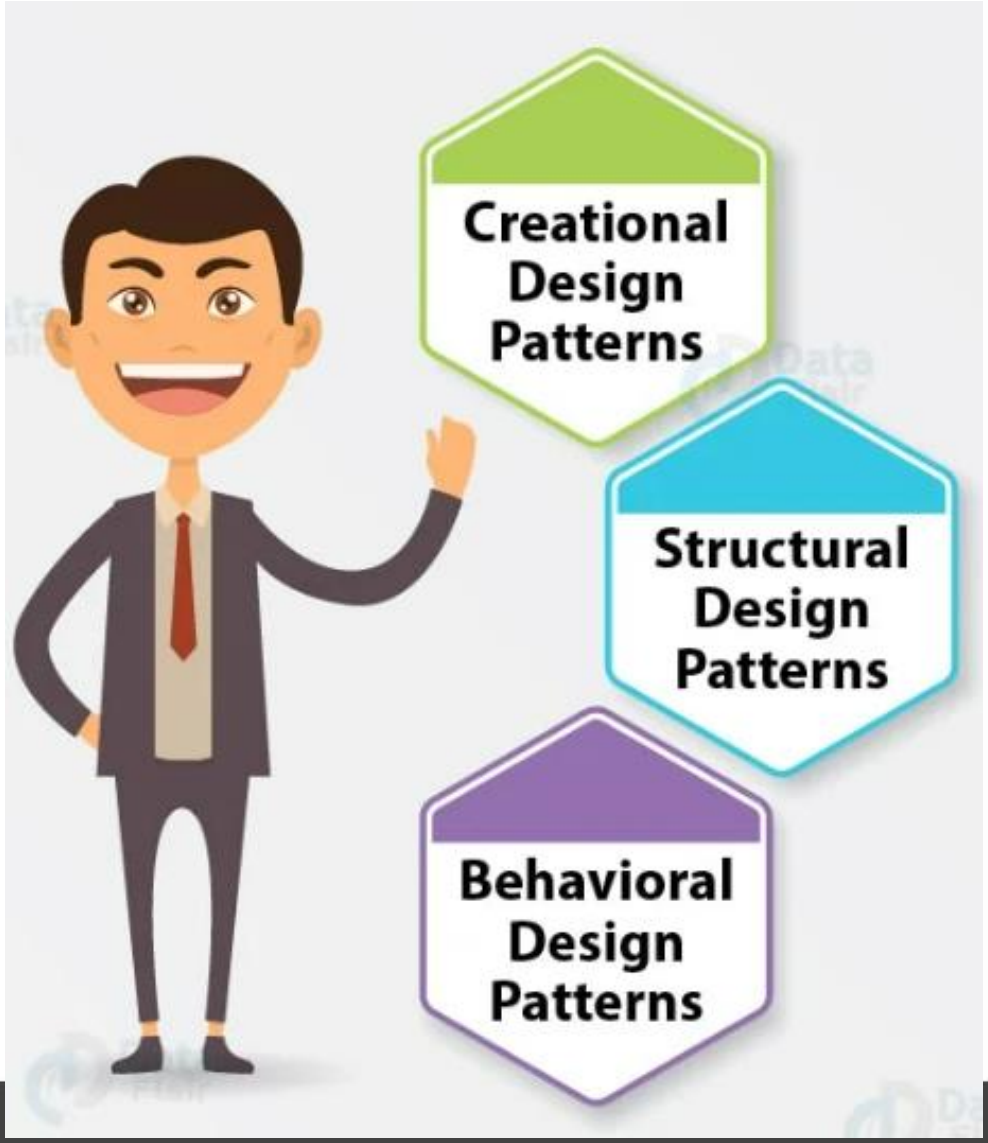


Tasarım Desenleri Kataloğu

GoF, Tasarım Örüntüleri üç kategoriye ayrılır:

1. Oluşsal (Creational) Örüntüler

Nesne oluşturma (instantiation) ile ilgili örüntülerdir. Bu kategorideki örüntüler, ilgili istemci nesnelerini oluşturdukları nesnelerden (instances) ayrıştırma ve bağımsızlaştırma, böylece istemci nesneleri daha kolay yönetilebilir yapma veya görev ayrıştırmasını yerine getirmekle ilgilidir. Tekli örüntü, soyut fabrika, fabrika örüntüleri bu sınıfa ait örüntülerdir.

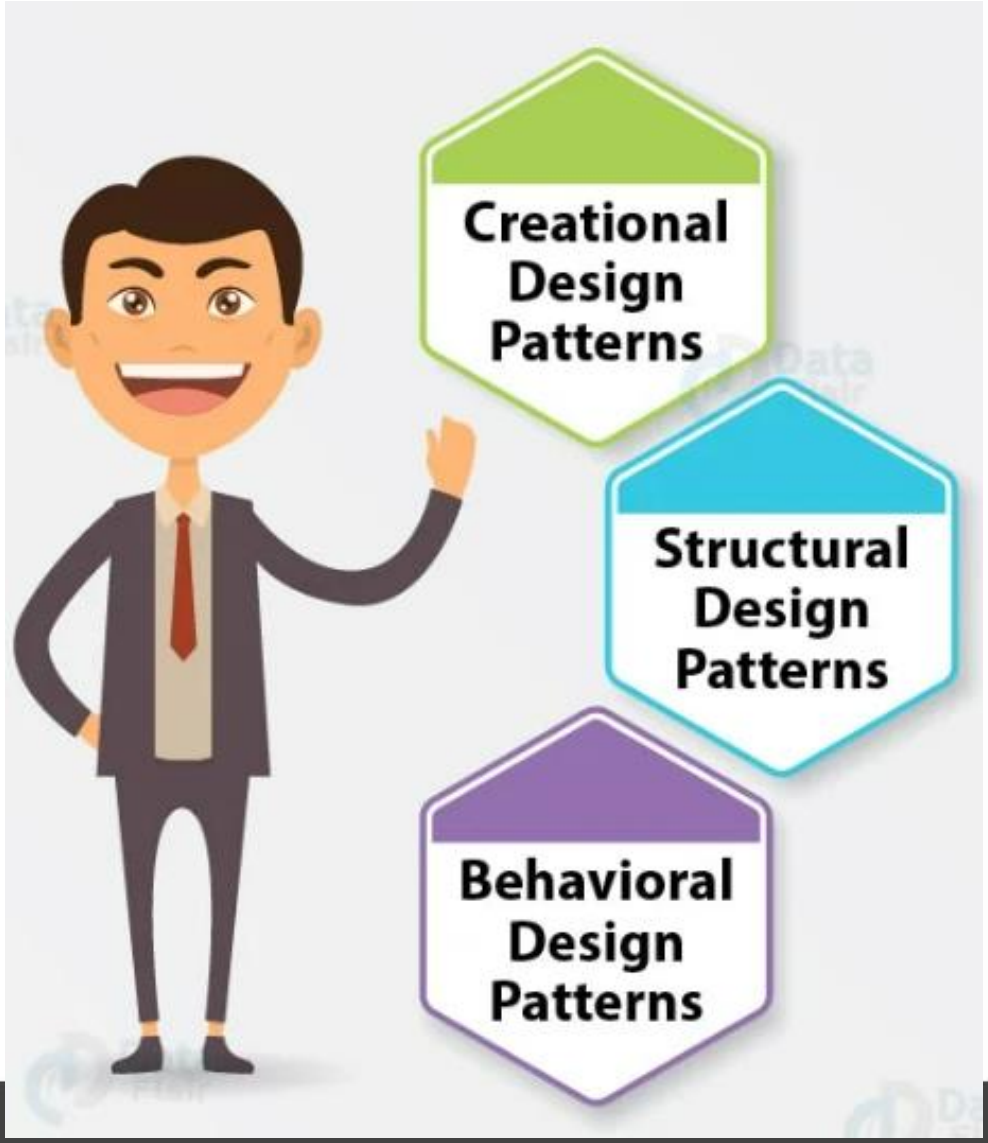


Tasarım Desenleri Kataloğu

GoF Tasarım Kalıpları üç kategoriye ayrılır:

2. Davranışsal (Behavioral) Örüntüler

Davranışsal örüntüler, sınıfların ve nesnelerin nasıl birbirleriyle ilişkilendirildikleri, mesajlaştıkları (nesneler arası iletişim) ve görevlerinin nasıl dağıtıldığı ile ilgilidir. Kalıp metodu, komut, iterasyon, gözlemci, durum, strateji örüntüleri bu kategori türüne ait örüntülerdir.

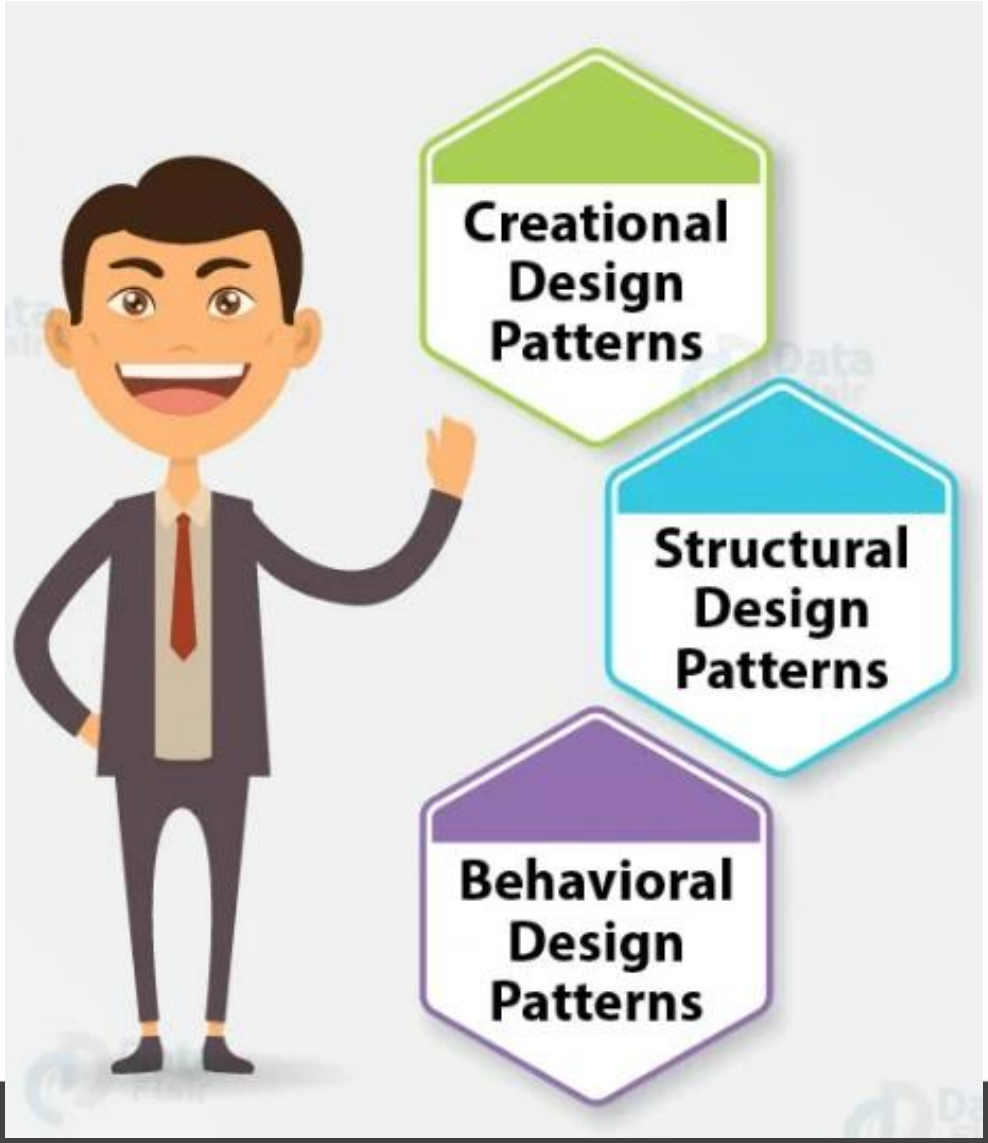


Tasarım Desenleri Kataloğu

GoF Tasarım Kalıpları üç kategoriye ayrılır:

3. Yapısal (Structural) Örüntüler:

Nesne ve sınıfların nasıl bir araya getirilip birleştirildikleri (kompozisyon) ve böylece daha kapsamlı görevleri olan yapıları oluşturuldukları ile ilgilidir. Komposit, dekoratör, adaptör, fasat, proksi bu türden örüntülerdir.



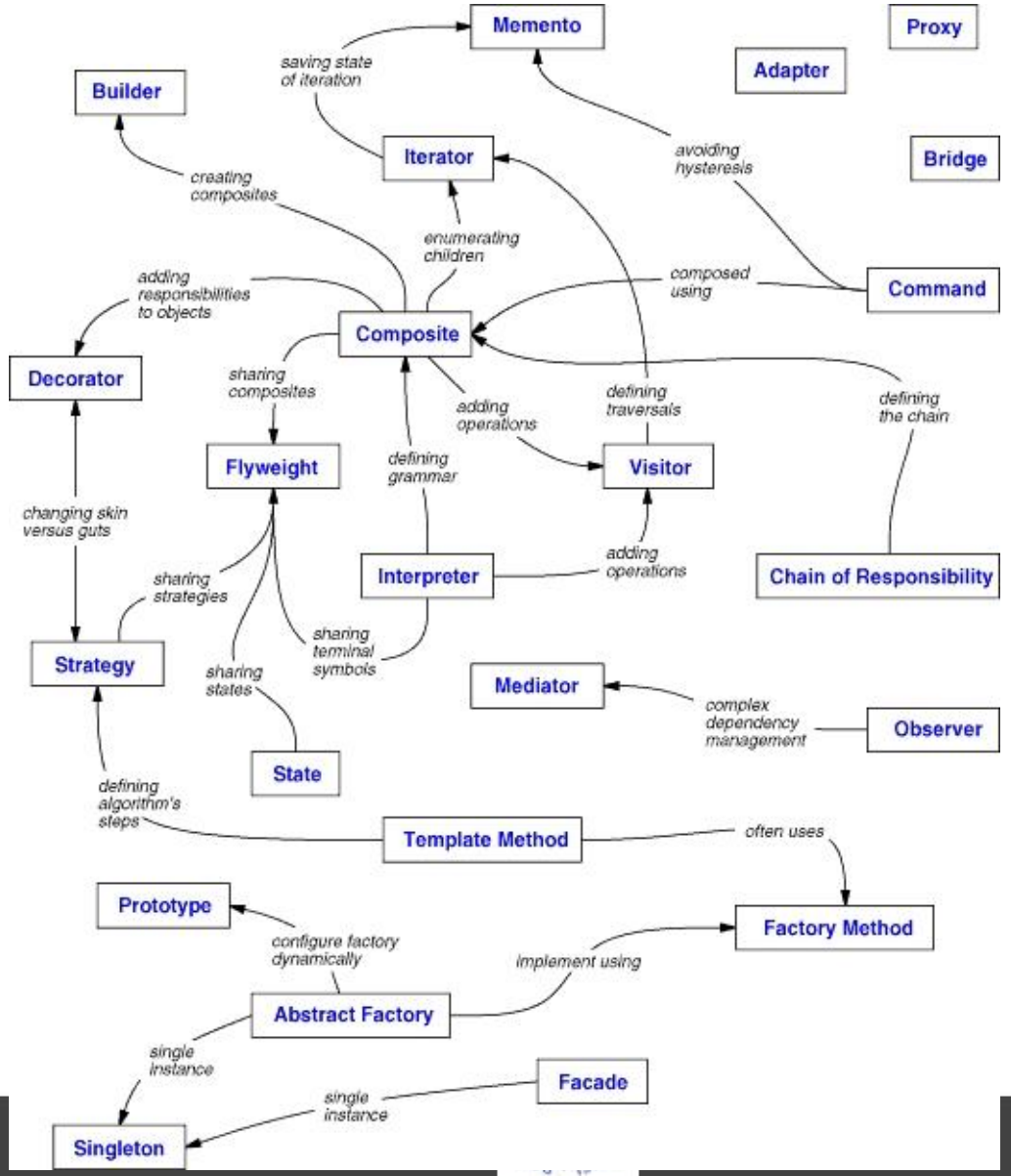
Tasarım Desenleri Kataloğu

		Purpose		
		Creational	**Structural**	**Behavioral**
Scope	**Class**	Factory Method (107)		Interpreter (243) Template Method (325)
	Object	Abstract Factory (87) Builder (97) Prototype (117) Singleton (127)	Adapter (139) Bridge (151) Composite (163) Decorator (175) Facade (185) Proxy (207)	Chain of Responsibility (223) Command (233) Iterator (257) Mediator (273) Memento (283) Flyweight (195) Observer (293) State (305) Strategy (315) Visitor (331)

Tasarım Desenleri Kataloğu

- Kapsam (Scope) olarak adlandırılan ikinci ölçüt, kalıbın öncelikle sınıflara mı yoksa nesnelere mi uygulanacağını belirtir.
- Sınıf kalıpları, sınıflar ve onların alt sınıfları arasındaki ilişkilerle ilgilenir. Bu ilişkiler kalıtım yoluyla kurulur, dolayısıyla statiktirler–derleme zamanında sabitlenirler.
- Nesne kalıpları, çalışma zamanında değiştirilebilen ve daha dinamik olan nesne ilişkileriyle ilgilenir.
- Hemen hemen tüm desenler bir dereceye kadar kalıtımı kullanır. Dolayısıyla "sınıf kalıpları" olarak adlandırılan kalıplar sadece sınıf ilişkilerine odaklanan kalıplardır. Çoğu kalıbın Nesne kapsamındadır.

Tasarım Desenleri Kataloğu



Tasarım Modelleri Tasarım Sorunlarını Nasıl Çözer?

Burada, tasarım kalıplarının uygun nesneleri bulmanıza, nesne ayrıntı düzeyini belirlemenize, nesne arabirimlerini belirlemenize ve tasarım kalıplarının tasarım problemlerini çözdüğü diğer birkaç yola nasıl yardımcı olduğunu tartışılacak

Tasarım Modelleri Tasarım Sorunlarını Nasıl Çözer?

Uygun Nesneleri Bulma

Nesne yönelimli programlar nesnelerden oluşur. Bir nesne hem verileri hem de bu veriler üzerinde çalışan prosedürleri paketler. Prosedürlere tipik olarak yöntemler veya işlemler denir. Bir nesne, bir istemciden bir istek (veya mesaj) aldığında bir işlem gerçekleştirir.

Tasarım desenleri, daha az belirgin olan soyutlamaları ve bunları yakalayabilen nesneleri belirlemenize yardımcı olur.

Örneğin, Strateji modeli, değiştirilebilir algoritma ailelerinin nasıl uygulanacağını açıklar.

Durum modeli, bir varlığın her durumunu bir nesne olarak temsil eder. Bu nesneler, analiz sırasında ve hatta tasarımın ilk aşamalarında nadiren bulunur; daha sonra bir tasarımı daha esnek ve yeniden kullanılabilir hale getirme sürecinde keşfedilirler.

Tasarım Modelleri Tasarım Sorunlarını Nasıl Çözer?

Nesne Tanecikliliğini Belirleme

Nesneler boyut ve sayı olarak çok farklı olabilir. Donanıma veya tüm uygulamalara kadar her şeyi temsil edebilirler. Neyin nesne olması gerektiğine nasıl karar veririz?

Tasarım kalıpları da bu konuyu ele alır.

Cephe (Facade) modeli, tüm alt sistemlerin nesneler olarak nasıl temsil edileceğini açıklar ve Flyweight modeli, çok sayıda nesnenin en ince ayrıntı düzeylerinde nasıl destekleneceğini açıklar. Diğer tasarım desenleri, bir nesneyi daha küçük nesnelere ayırmanın belirli yollarını tanımlar.

Tasarım Modelleri Tasarım Sorunlarını Nasıl Çözer?

Nesne Arayüzlerini Belirtme

Bir nesne tarafından bildirilen her işlem, işlemin adını, parametre olarak aldığı nesneleri ve işlemin dönüş değerini belirtir. Bu, işlemin imzası olarak bilinir. Bir nesnenin işlemleri tarafından tanımlanan tüm imzaların kümesine, nesnenin arabirimi denir. Bir nesnenin arayüzü, nesneye gönderilebilecek isteklerin tamamını karakterize eder.

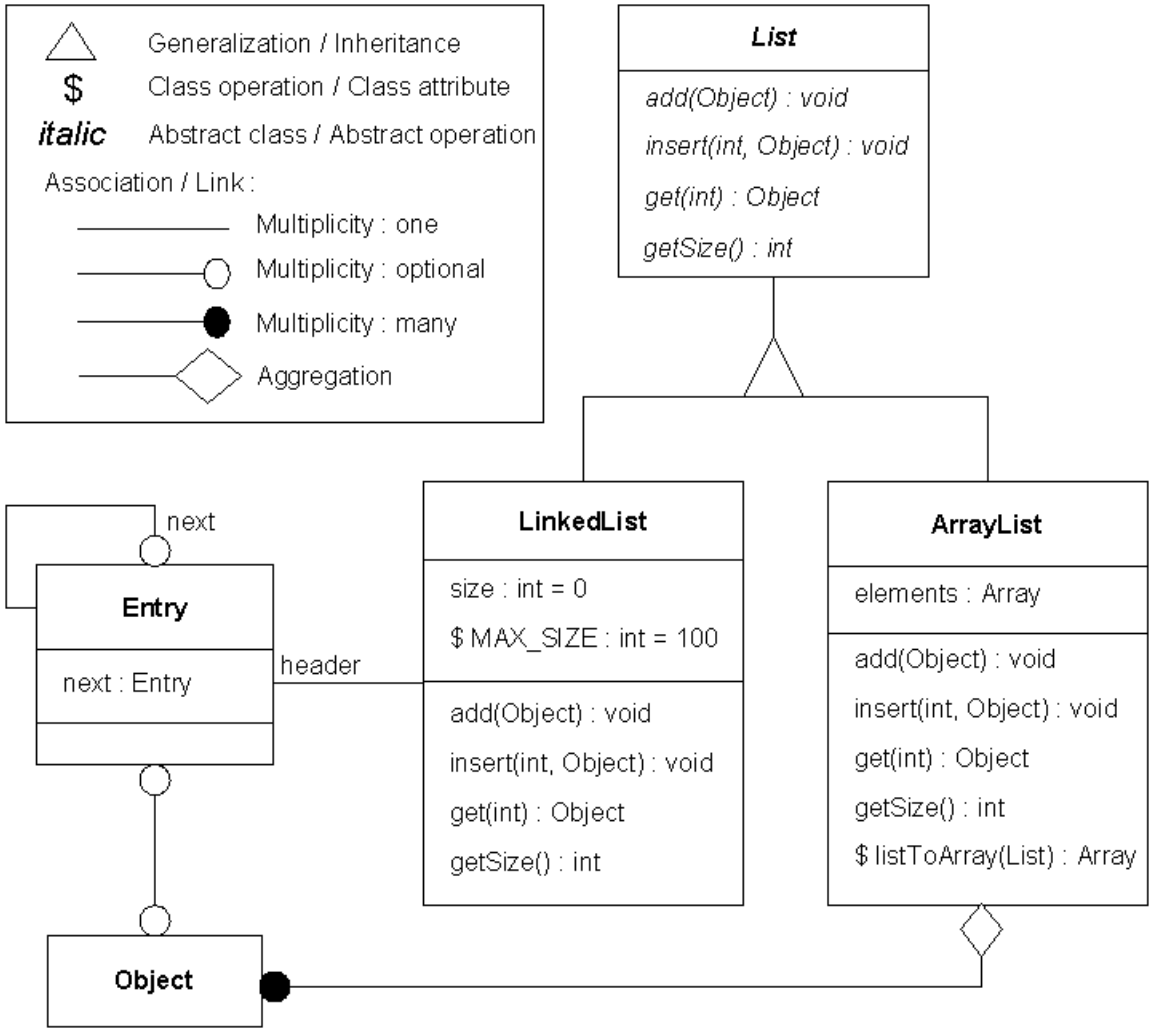
Tasarım kalıpları, temel öğelerini ve bir arabirim üzerinden gönderilen veri türlerini tanımlayarak arabirimleri tanımlamanıza yardımcı olur. Bir tasarım deseni, arayüze ne koymayacağınızı da söyleyebilir. Memento (283) modeli iyi bir örnektir. Nesnenin daha sonra bu duruma geri yüklenebilmesi için bir nesnenin dahili durumunun nasıl kapsülleneceğini ve kaydedileceğini açıklar.

Tasarım kalıpları, arayüzler arasındaki ilişkileri de belirler.

Tasarım Modelleri Tasarım Sorunlarını Nasıl Çözer?

Nesne Uygulamalarını (Implementation) Belirtme

Bir nesnenin uygulaması, sınıfı tarafından tanımlanır. Sınıf, nesnenin dahili verilerini ve temsilini belirtir ve nesnenin gerçekleştirebileceği işlemleri tanımlar.



Tasarım Modelleri Tasarım Sorunlarını Nasıl Çözer?

Sınıfa karşı Arayüz Kalıtımı

Bir nesnenin sınıfı ile türü arasındaki farkı anlamak önemlidir.Bir nesnenin sınıfı, nesnenin nasıl uygulanacağını tanımlar. Sınıf mirası ile arayüz mirası (veya alt tipleme) arasındaki farkı anlamak da önemlidir. Sınıf mirası, bir nesnenin uygulamasını başka bir nesnenin uygulaması açısından tanımlar. Kısacası, kod ve temsil paylaşımı için bir mekanizmadır. Buna karşılık, arayüz kalıtımı (veya alt tipleme), bir nesnenin ne zaman başka bir nesnenin yerine kullanılabileceğini tanımlar. Pek çok dil bu ayrımı açıkça ortaya koymaz.

Tasarım desenlerinin çoğu bu ayrıma bağlıdır. Örneğin, bir Sorumluluk Zincirindeki (Chain of Responsibility) nesnelerin ortak bir türü olmalıdır, ancak genellikle ortak bir uygulamayı paylaşmazlar.

Tasarım Modelleri Tasarım Sorunlarını Nasıl Çözer?

Bir Uygulamaya Değil, Bir Arayüze Programlama

Sınıf kalıtımı, temel olarak, üst sınıflardaki işlevselliği yeniden kullanarak bir uygulamanın işlevselliğini genişletmeye yönelik bir mekanizmadır. Eski bir nesneye göre yeni bir tür nesneyi hızlı bir şekilde tanımlamanıza olanak tanır.

Ancak, uygulamanın yeniden kullanımı hikayenin yalnızca yarısıdır. Kalıtımın aynı arabirimlere sahip nesne ailelerini tanımlama yeteneği (genellikle soyut bir sınıftan miras alarak) da önemlidir (polimorfizm).

Tasarım Modelleri Tasarım Sorunlarını Nasıl Çözer?

Bir Uygulamaya Değil, Bir Arayüze Programlama

Nesneleri yalnızca soyut sınıflar tarafından tanımlanan arabirim açısından değiştirmenin iki faydası vardır:

1. Nesneler, istemcilerin beklediği arabirime uyduğu sürece, istemciler kullandıkları belirli nesne türlerinden habersiz kalırlar.
2. İstemciler, bu nesneleri uygulayan sınıflardan habersiz kalır.

Bir uygulamaya değil, bir arayüze programlayın.

- *Değişkenleri belirli somut sınıfların örnekleri olarak bildirmeyin.*
- *Bunun yerine, yalnızca soyut bir sınıf tarafından tanımlanan bir arabirime bağlı kalın.*

Tasarım Modelleri Tasarım Sorunlarını Nasıl Çözer?

Yeniden Kullanım Mekanizmalarını Çalıştırmak

Çoğu insan nesneler, arayüzler, sınıflar ve kalıtım gibi kavramları anlayabilir. Zorluk, esnek, yeniden kullanılabilir yazılım oluşturmak için bunları uygulamakta yatmaktadır ve tasarım kalıpları size nasıl olduğunu gösterebilir.

Tasarım Modelleri Tasarım Sorunlarını Nasıl Çözer?

Kalıtıma karşın Nesne Kompozisyonları

Nesne yönelimli sistemlerde işlevselliği yeniden kullanmak için en yaygın iki teknik, sınıf kalıtımı ve nesne kompozisyonlarıdır.

Sınıf kalıtımı, bir sınıfın uygulamasını diğerinin terimleriyle tanımlamanıza izin verir. Alt sınıflama yoluyla yeniden kullanım, genellikle beyaz kutu yeniden kullanımı olarak adlandırılır.

Nesne kompozisyonları, sınıf mirasına bir alternatiftir. Burada, daha karmaşık işlevsellik elde etmek için nesneleri bir araya getirerek veya bir araya getirerek yeni işlevsellik elde edilir. Nesneler yalnızca "kara kutular" olarak görünür.

Kalıtım ve kompozisyonun her birinin avantajları ve dezavantajları vardır.

Tasarım Modelleri Tasarım Sorunlarını Nasıl Çözer?

Yetkilendirme

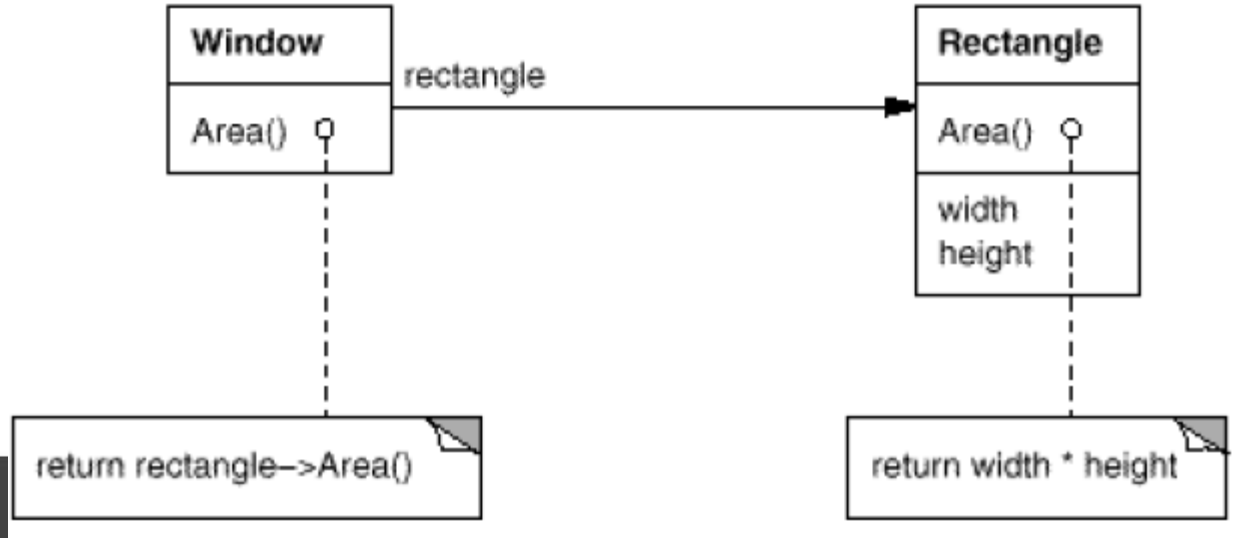
Yetkilendirme, kompozisyonu yeniden kullanım için kalıtım kadar güçlü hale getirmenin bir yoludur .

Yetkilendirmede, bir isteğin ele alınmasında iki nesne yer alır: bir alıcı nesne, işlemleri temsilcisine devreder. Bu, istekleri üst sınıflara erteleyen alt sınıflara benzer. Ancak kalıtımla, devralınan bir işlem her zaman C++'da this üye değişkeni aracılığıyla alıcı nesneye başvurabilir. Yetkilendirme ile aynı etkiyi elde etmek için, alıcı, devredilen işlemin alıcıya başvurmasına izin vermek için kendisini temsilciye iletir.

Tasarım Modelleri Tasarım Sorunlarını Nasıl Çözer?

Yetkilendirme

Örneğin, Window sınıfını Rectangle'ın bir alt sınıfı yapmak yerine (çünkü pencereler dikdörtgen olur), Window sınıfı bir Rectangle örnek değişkeni tutarak ve ona Rectangle'a özgü davranış atayarak Rectangle davranışını yeniden kullanabilir. Başka bir deyişle, Pencerenin Dikdörtgen olması yerine Dikdörtgen olması gerekir. Pencerenin şimdi istekleri açıkça Rectangle örneğine iletmesi gerekir, oysa daha önce bu işlemleri devralırdı.



Tasarım Modelleri Tasarım Sorunlarını Nasıl Çözer?

Yetkilendirme

Delegasyonun ana avantajı, çalışma zamanında davranışları oluşturmayı ve oluşturulma şeklini değiştirmeyi kolaylaştırmasıdır. Penceremiz, Rectangle ve Circle'ın aynı türe sahip olduğunu varsayarak, Rectangle örneğini bir Circle örneğiyle değiştirerek çalışma zamanında dairesel hale gelebilir.

Delegasyonun, nesne kompozisyonu aracılığıyla yazılımı daha esnek hale getiren diğer tekniklerle paylaştığı bir dezavantajı vardır: Dinamik, yüksek düzeyde parametreli yazılımları anlamak, daha statik yazılımlardan daha zordur. Çalışma zamanı verimsizlikleri de vardır.

Tasarım Modelleri Tasarım Sorunlarını Nasıl Çözer?

Kalıtım veya Parametreli Türler

İşlevselliği yeniden kullanmak için başka bir (kesinlikle nesne yönelimli olmayan) teknik, Generics (Java) ve şablonlar (C++) olarak da bilinen parametreli türlerdir. Bu teknik, kullandığı diğer tüm türleri belirtmeden bir tür tanımlamanıza olanak tanır.

Tasarım Modelleri Tasarım Sorunlarını Nasıl Çözer?

Çalışma Zamanı ve Derleme Zamanı Yapılarını İlişkilendirme

Nesne yönelimli bir programın çalışma zamanı yapısı genellikle kod yapısına çok az benzerlik gösterir. Kod yapısı derleme zamanında dondurulur; sabit kalıtım ilişkilerinde sınıflardan oluşur. Bir programın çalışma zamanı yapısı, hızla değişen iletişim kuran nesne ağlarından oluşur. Aslında, iki yapı büyük ölçüde bağımsızdır.

Tasarım Modelleri Tasarım Sorunlarını Nasıl Çözer?

Uygulama programları

Belge düzenleyici veya elektronik tablo gibi bir uygulama programı oluşturuyorsanız, dahili yeniden kullanım, sürdürülebilirlik ve genişletme yüksek önceliklerdir. Dahili yeniden kullanım, yapmanız gerekenden fazlasını tasarlamamanızı ve uygulamamanızı sağlar. Bağımlılıkları azaltan tasarım kalıpları, dahili yeniden kullanımı artırabilir.

Tasarım kalıpları ayrıca, platform bağımlılıklarını sınırlamak ve bir sistemi katmanlamak için kullanıldıklarında bir uygulamayı daha sürdürülebilir hale getirir.

Tasarım Modelleri Tasarım Sorunlarını Nasıl Çözer?

Araç takımları (toolkits)

Genellikle bir uygulama, araç takımları adı verilen bir veya daha fazla önceden tanımlanmış sınıf kitaplığından sınıfları içerecektir. Araç takımı, kullanışlı, genel amaçlı işlevsellik sağlamak üzere tasarlanmış, ilgili ve yeniden kullanılabilir sınıflar kümesidir.

Araç takımı tasarımı, uygulama tasarımından tartışmalı bir şekilde daha zordur, çünkü araç takımlarının faydalı olması için birçok uygulamada çalışması gerekir.

Tasarım Modelleri Tasarım Sorunlarını Nasıl Çözer?

Çerçeveler(Frameworks)

Çerçeve, belirli bir yazılım sınıfı için yeniden kullanılabilir bir tasarım oluşturan bir dizi işbirliği yapan sınıftır.

Başka bir çerçeve, farklı programlama dilleri ve hedef makineler için derleyiciler oluşturmanıza yardımcı olabilir.

Çerçeveden soyut sınıfların uygulamaya özel alt sınıflarını oluşturarak bir çerçeveyi belirli bir uygulamaya göre özelleştirebilirsiniz.

Tasarım Modelleri Tasarım Sorunlarını Nasıl Çözer?

Çerçeveler(Frameworks)

Kalıplar ve çerçeveler bazı benzerliklere sahiptirler

Üç ana yönden farklıdırlar:

1. Tasarım kalıpları çerçevelerden daha soyuttur.
2. Tasarım desenleri, çerçevelerden daha küçük mimari öğelerdir. Tipik bir çerçeve birkaç tasarım deseni içerir, ancak bunun tersi asla doğru değildir.
3. Tasarım kalıpları çerçevelerden daha az uzmanlaşmıştır. Çerçevelerin her zaman belirli bir uygulama alanı vardır.

Tasarım Deseni Nasıl Seçilir?

1. Tasarım kalıplarının tasarım problemlerini nasıl çözdüğünü düşünün.
2. Niyet bölümlerini araştırın.
3. Modellerin birbiriyle nasıl ilişkili olduğunu inceleyin.
4. Benzer amaca yönelik kalıpları inceleyin.
5. Yeniden tasarımın bir nedenini inceleyin.
6. Tasarımınızda neyin değişken olması gerektiğini düşünün.

Observer Patterns

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

Konsept

Bu modelde, belirli bir özneyi (nesneyi) gözlemleyen birçok gözlemci (nesne) vardır. Gözlemciler temelde ilgilenirler ve o konuda bir değişiklik yapıldığında haberdar olmak isterler. Böylece kendilerini o konuya kaydettirirler. Konuya olan ilgilerini kaybettiklerinde, konunun kaydını silerler. Bazen bu modele Yayıncı-Abone modeli de denir.

Observer Patterns

Bilgisayar Dünyası Örneği

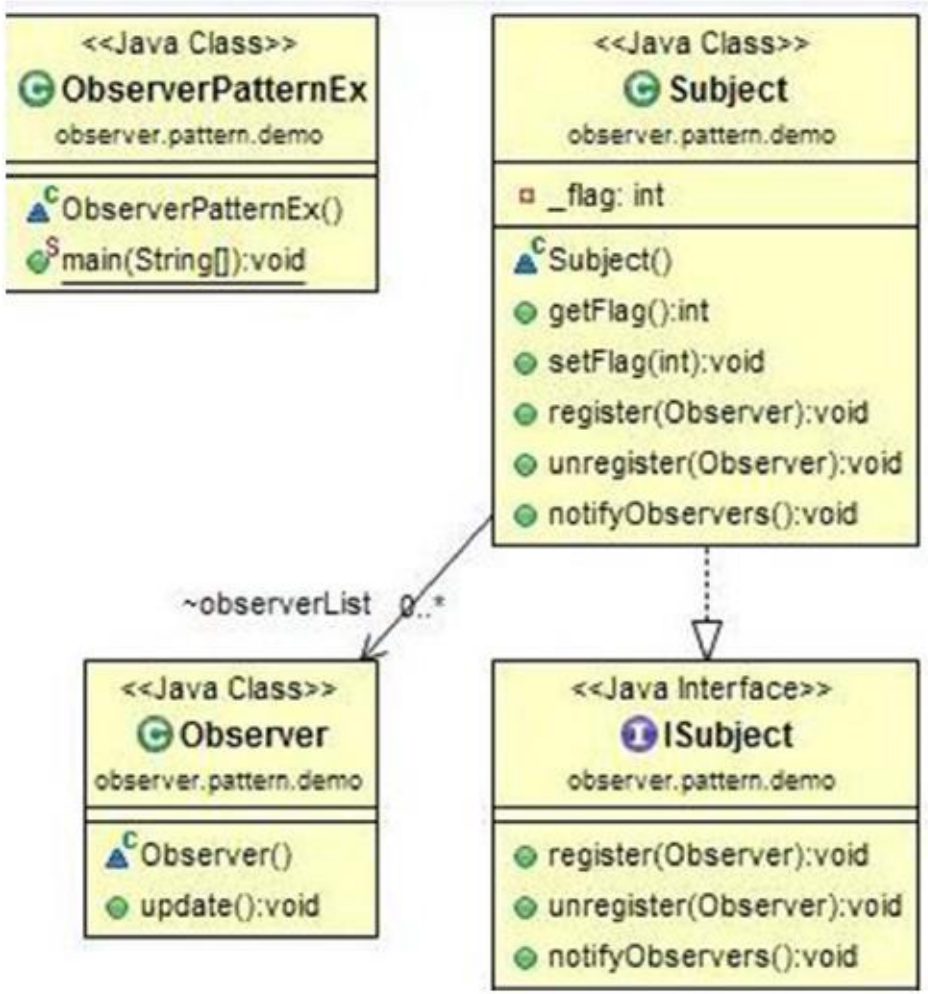
Bilgisayar bilimi dünyasında, bu kullanıcı arabiriminin bir veritabanıyla (veya iş mantığıyla) bağlantılı olduğu, kullanıcı arabirimi tabanlı basit bir örnek düşünün. Bir kullanıcı, bu UI aracılığıyla bazı sorgular yürütebilir ve veritabanını aradıktan sonra, sonuç UI'ye geri yansıtılır. Çoğu durumda,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

Observer Patterns

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

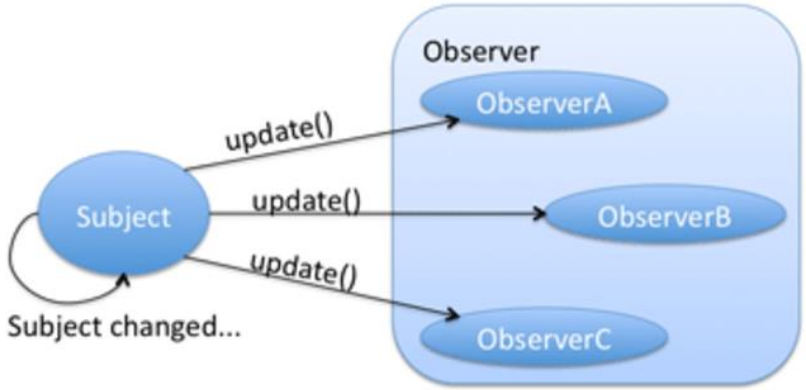


Observer Patterns -Behavioral (Davranışsal)

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

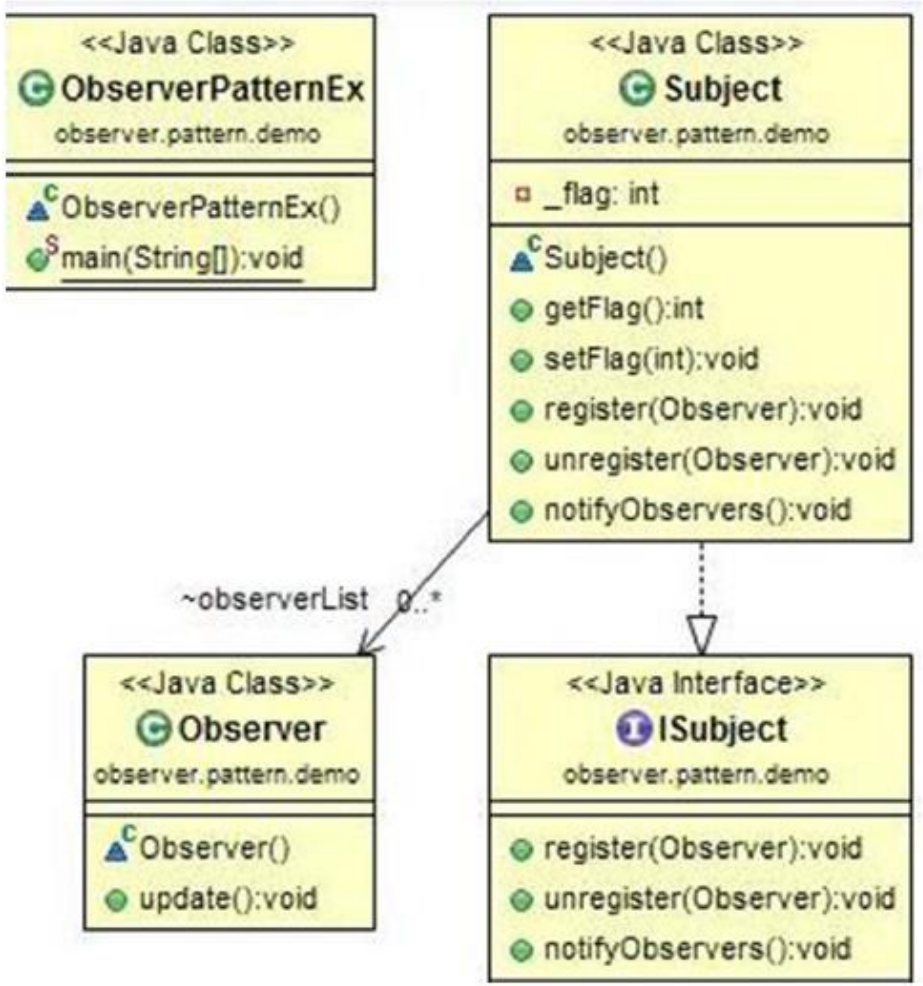
Bilgisayar bilimi dünyasında, bu kullanıcı arabiriminin bir veritabanıyla (veya iş mantığıyla) bağlantılı olduğu, kullanıcı arabirimi tabanlı basit bir örnek düşünün. Bir kullanıcı, bu UI aracılığıyla bazı sorgular yürütebilir ve veritabanını aradıktan sonra, sonuç UI'ye geri yansıtılır. Çoğu durumda,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

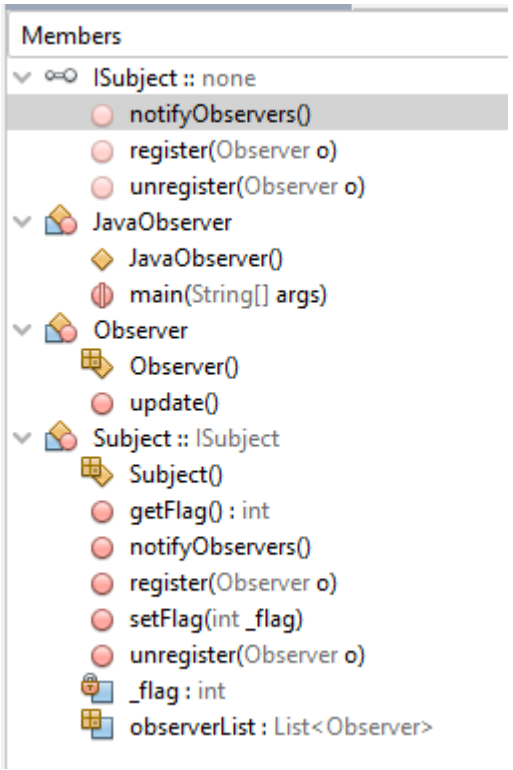
Observer Patterns-Behavioral (Davranışsal)

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)

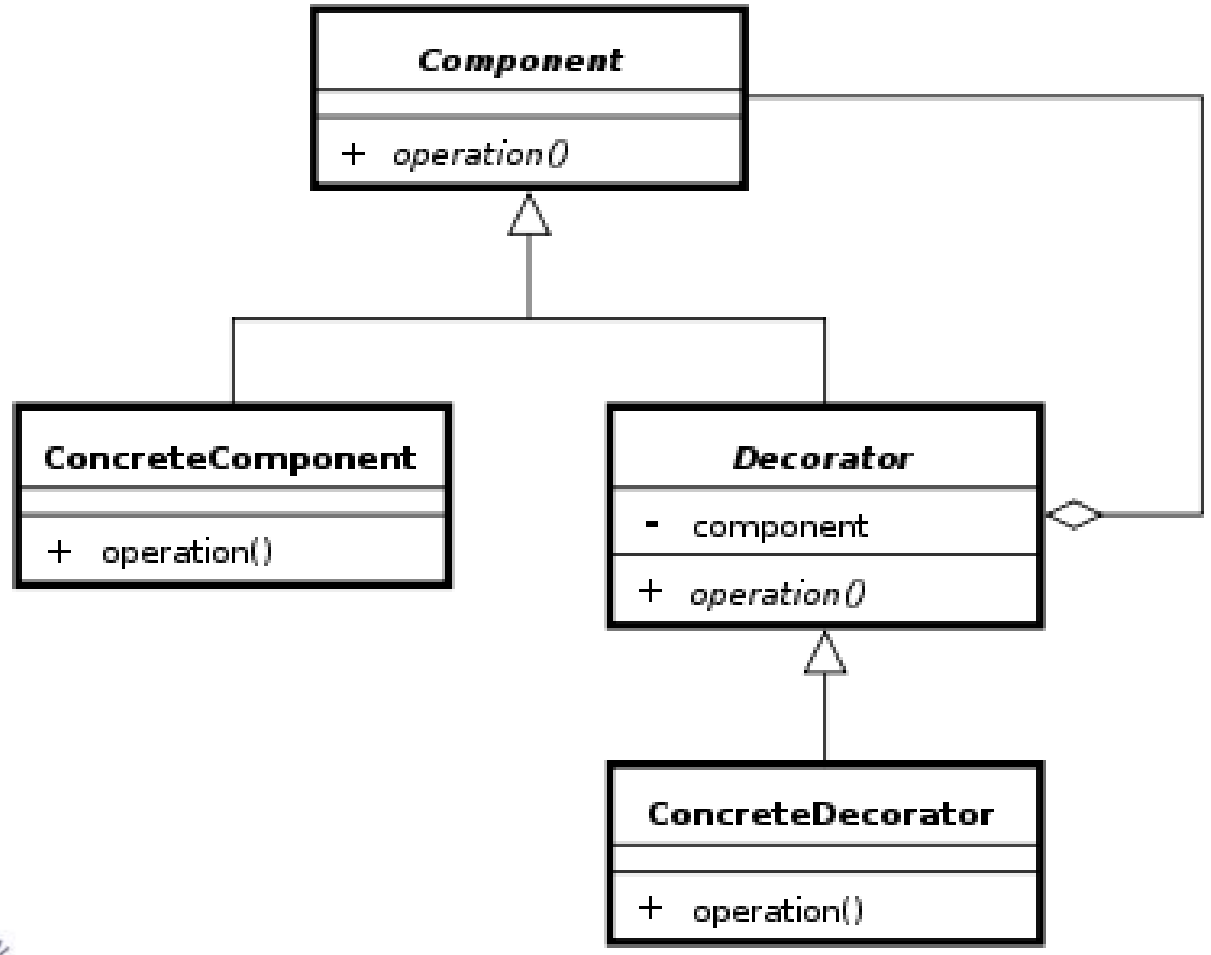
Dekoratör Tasarım modelinin katılımcıları şunlardır:

•**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

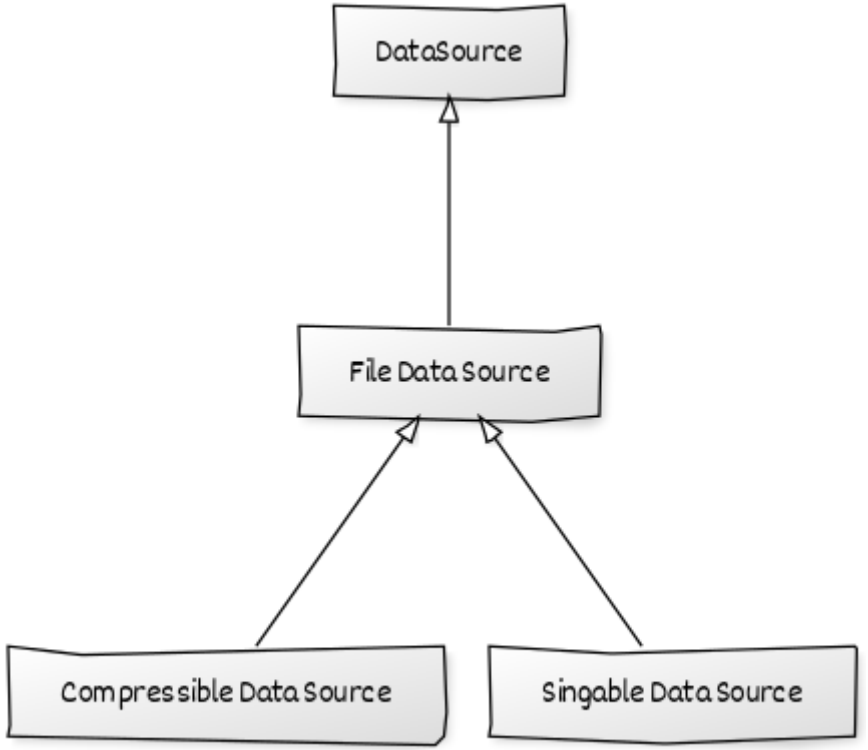
Bir dosyanın okunması ve dosyaya yazılma işlemi.

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.

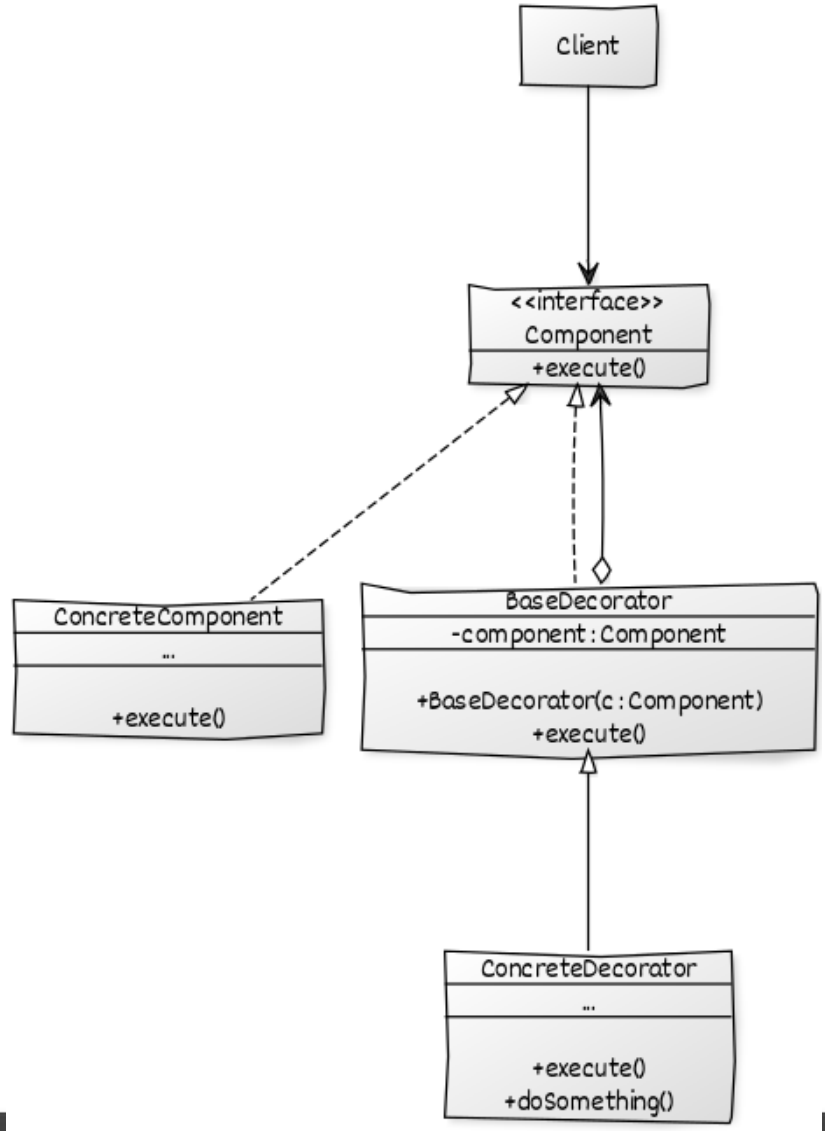


Decorator Patterns-Structural (Yapısal)

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)

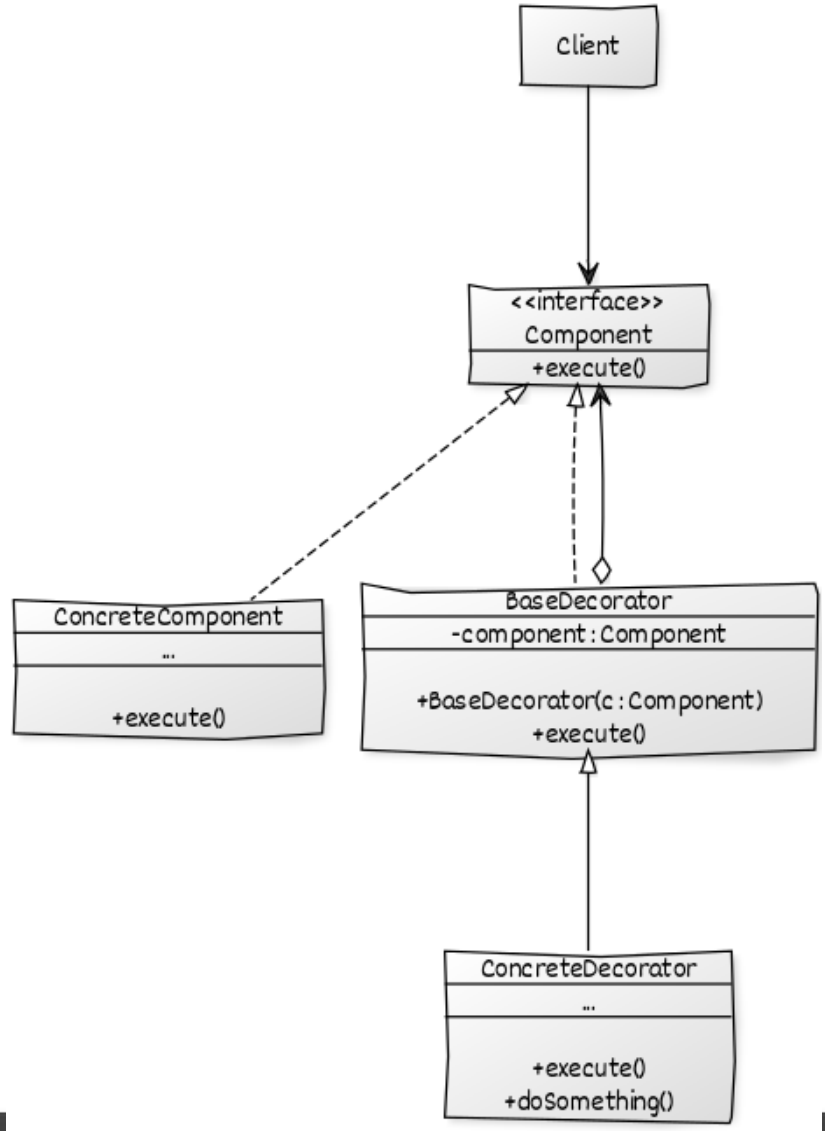
İllüstrasyon

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)

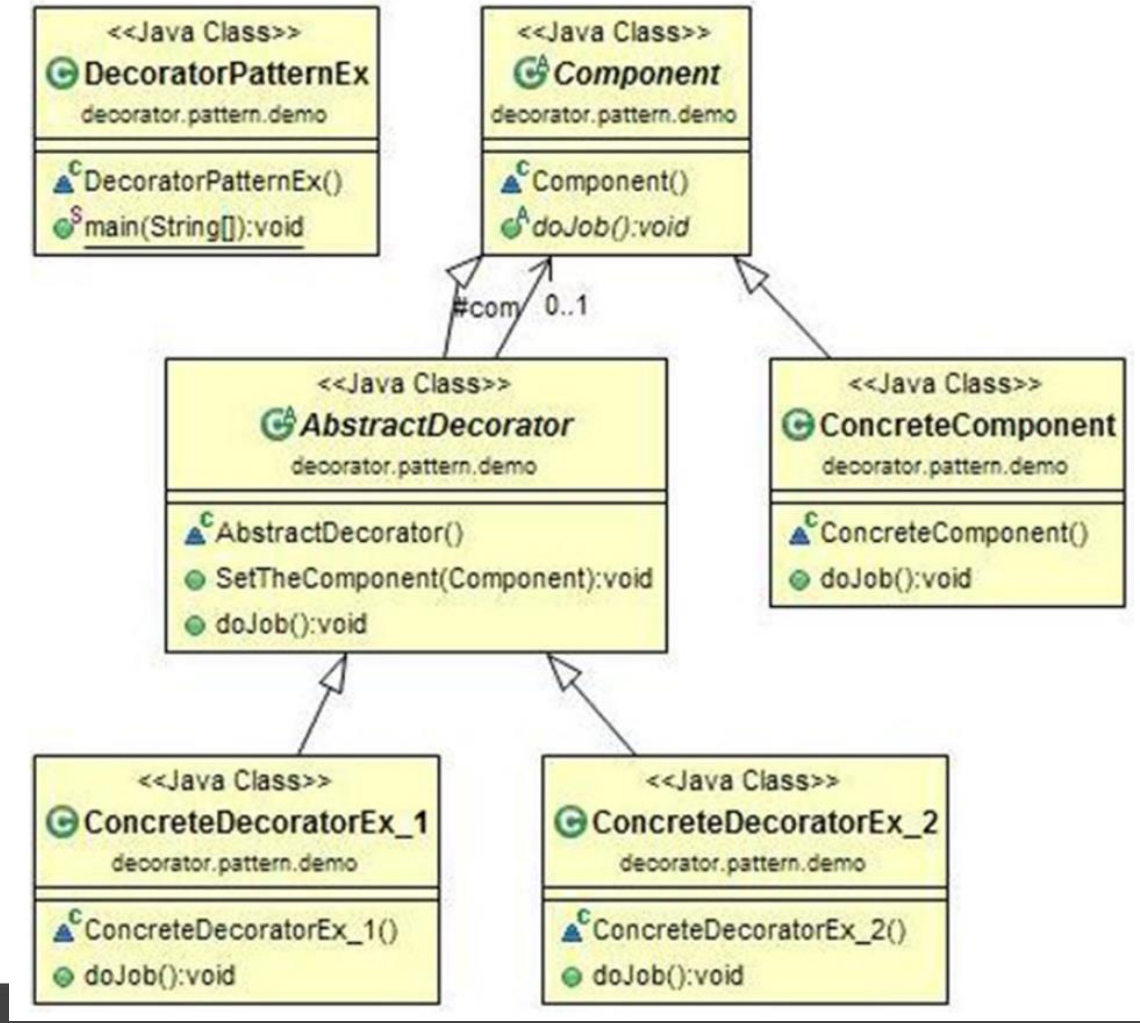
Bilgisayar Dünyası Örneği

•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.

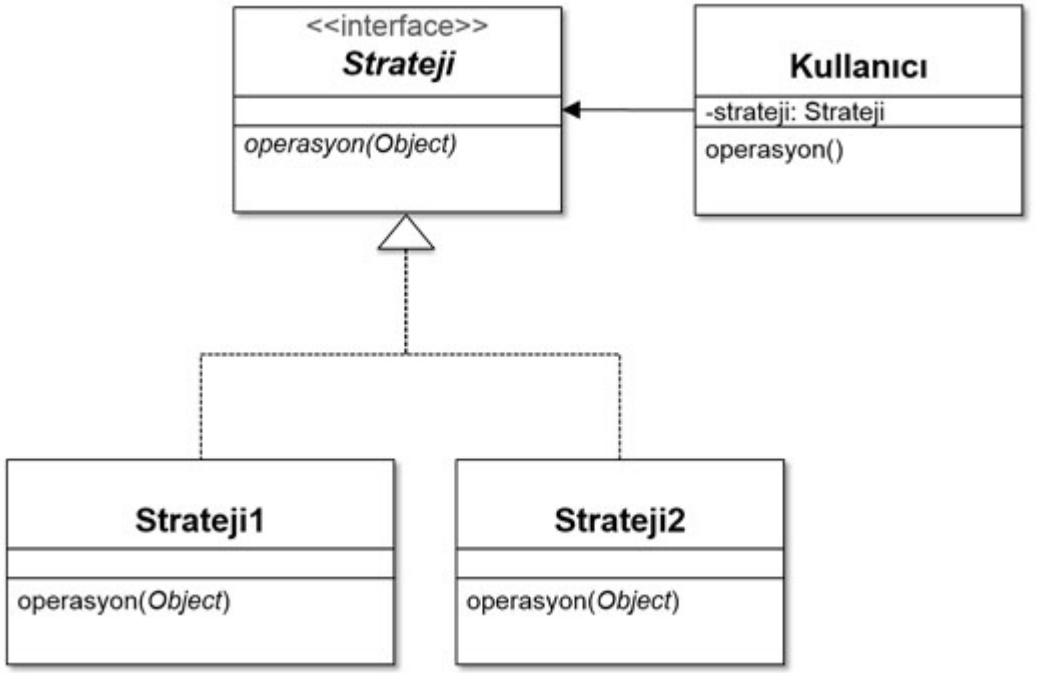
Açık/kapalı Prensibi ile Hareket Eder

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

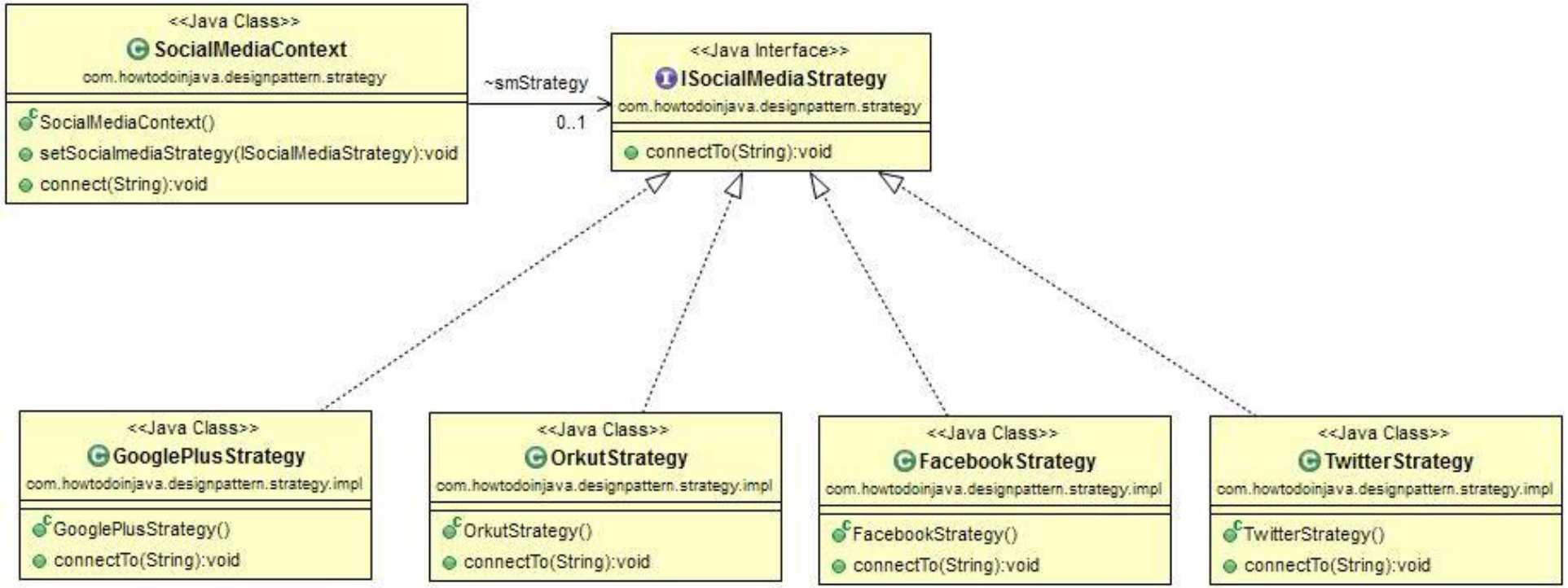
Yukarıdaki problemde, birden çok yolla (arkadaşla bağlantı kur) yapılabilecek bir işlemimiz var ve kullanıcı çalışma zamanında istediği yolu seçebiliyor. Bu yüzden strateji tasarım modeli için iyi bir adaydır.

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

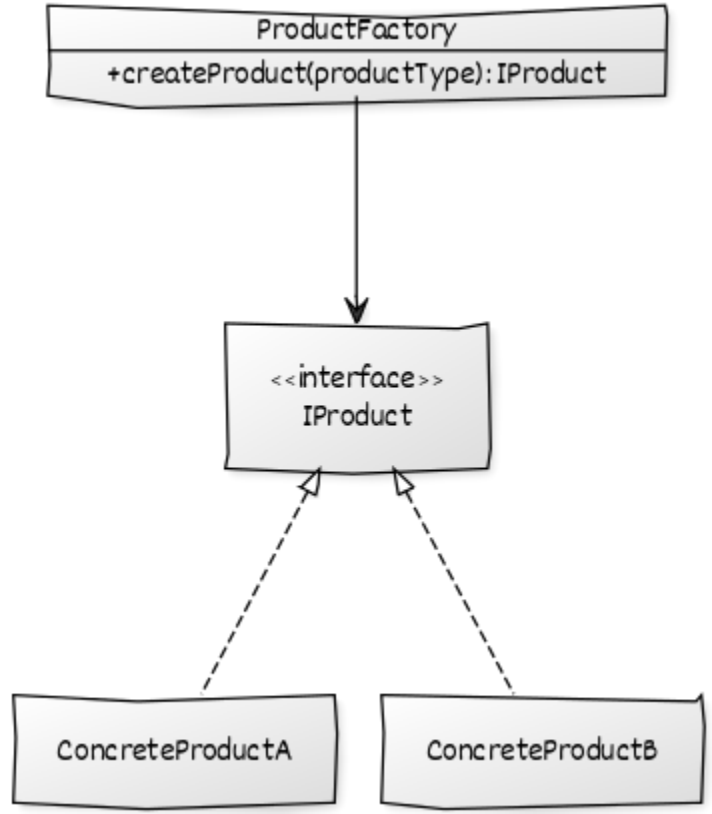


Factory Patterns-Creational (Oluşturucu)

GoF Tanımı: Factory tasarım deseni birbirleri ile ilişkili nesneleri oluşturmak için bir arayüz sağlar ve alt sınıfların hangi sınıfın örneğini oluşturacağına olanak sağlar..

Konsept

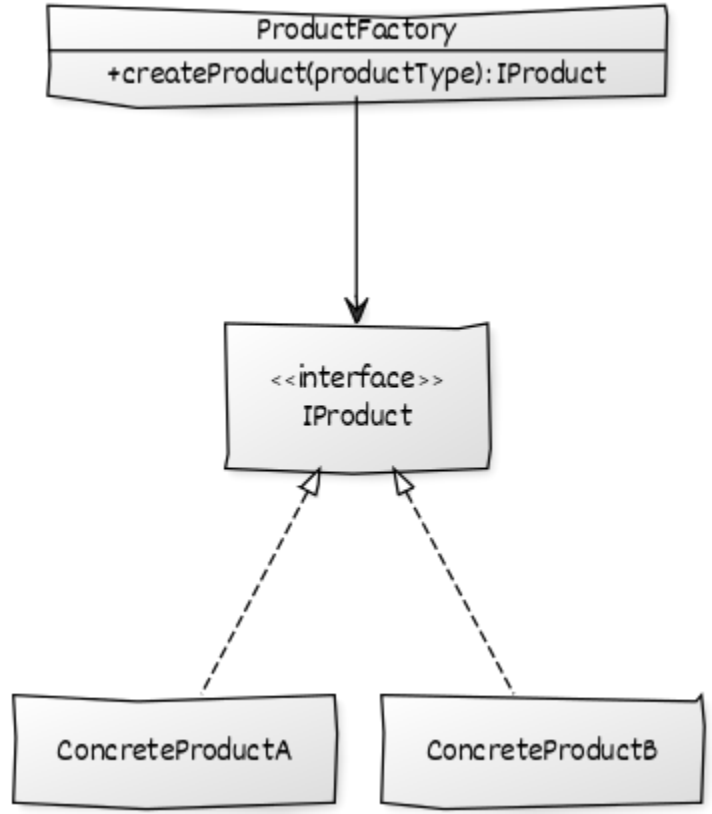
Buradaki amaç istemci tarafından birbirleri ile ilişkili nesnelerin oluşturulma anını soyutlamak, istemci hangi sınıf örneğini alabileceğini bilebilir ama oluşturulma detayları bilmez. Detaylar yani nesnenin nasıl oluşturulacağı soyutlanır.



Factory Patterns-Creational (Oluşturucu)

Konsept

Fabrika modeli , uygulama mimarisi tasarlanırken göz önünde bulundurulması ve uygulanması gereken en önemli ilke olan **sınıflar arasında gevşek bağlantı sağlar.** Gevşek bağlantı, somut uygulamalar yerine soyut varlıklara karşı programlama yaparak uygulama mimarisinde tanıtılabilir. Bu, mimarimizi yalnızca daha esnek kılmakla kalmaz, aynı zamanda daha az kırılgan hale getirir.



Factory Patterns-Creational (Oluşturucu)

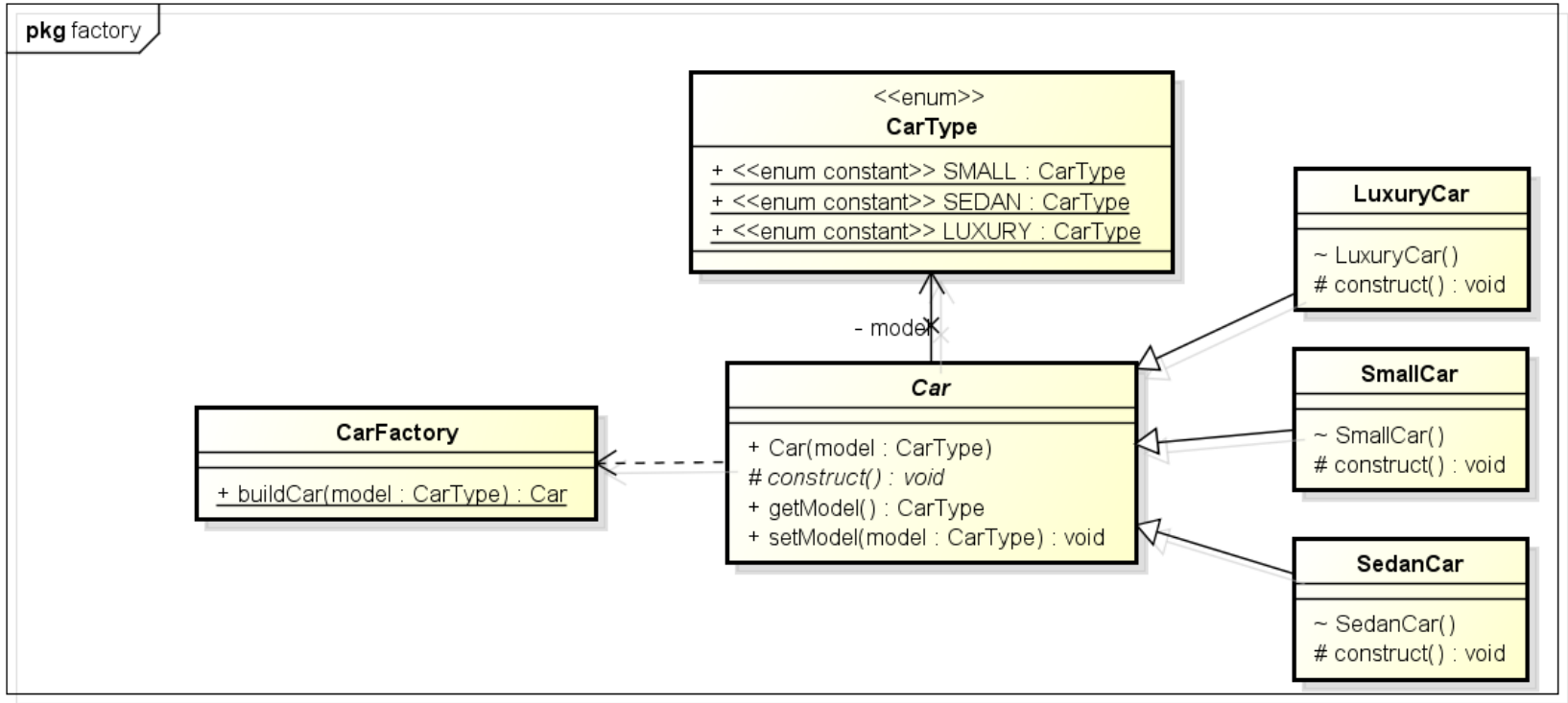
Bilgisayar Dünyası Örneği

•küçük, sedan ve lüks olmak üzere 3 tip araba üretebilen bir araba fabrikası örneğini kullanan ortak bir senaryoyu göstermektedir. Bir araba inşa etmek, aksesuarların tahsis edilmesinden son makyaja kadar birçok adım gerektirir. Bu adımlar, programlamada yöntemler olarak yazılabilir ve belirli bir araba tipinin bir örneğini oluştururken çağrılmalıdır.

•Eğer şanssızsak, SmallCaruygulama sınıflarımızda araba tiplerinin (örn.) örneklerini oluşturacağız ve böylece araba inşa mantığını dış dünyaya açacağız ve bu kesinlikle iyi değil. Ayrıca kod merkezi olmadığı için araba yapım sürecinde değişiklik yapmamızı da engelliyor ve tüm beste sınıflarında değişiklik yapmak mümkün görünmüyor.

Factory Patterns-Creational (Oluşturucu)

İllüstrasyon



Abstract Factory Patterns-Creational (Oluşturucu)

GoF Tanımı: Abstract Factory tasarım deseni birbirleri ile ilişkili ürün ailesini oluşturmak için bir arayüz sağlar.

Konsept

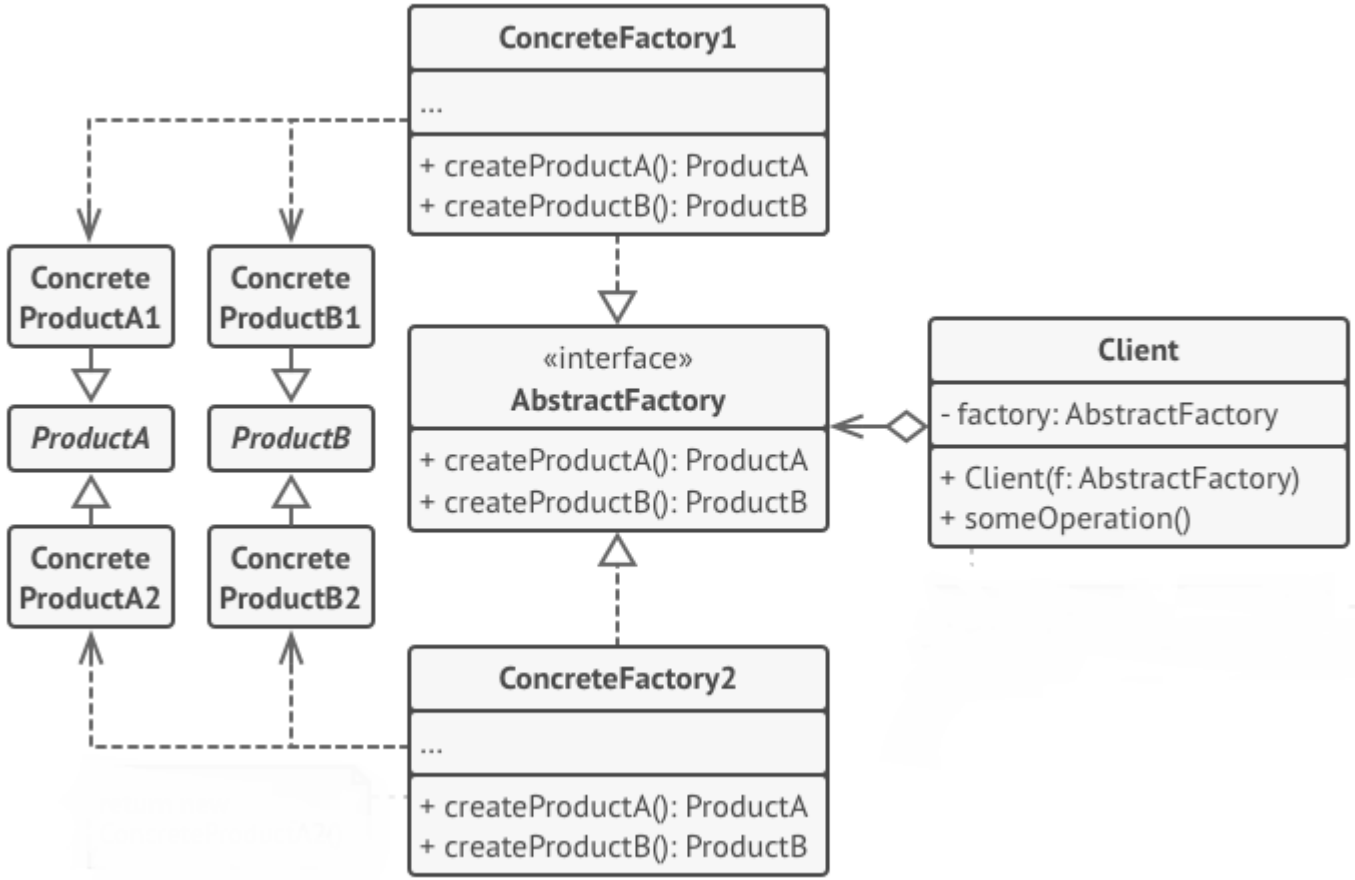
Soyut fabrika deseni , başka bir yaratıcı tasarım desenidir ve **fabrika deseni** üzerinde başka bir soyutlama katmanı olarak kabul edilir . Soyut fabrika örüntüsü, bir istemcinin, somut sınıfları belirtmeden nesneleri oluşturmasını sağlar. Bunun için bir fabrika nesnesi kullanır. Fabrika nesneleri istenen türde nesne oluşturmakla görevlidir. Aşağıda soyut fabrika örüntüsünün sınıf diyagramı görülmektedir. Burada istemci SoyutFabrika arayüzünü gerçekleştiren SomutFabrika nesnelerinden birini üretir. Bu somut fabrika da SoyutÜrün tiplerinden birinden istenen türde nesneleri üretmekle görevlidir. Sonrasında istemci hangi somut fabrika ve hangi somut ürün olduğu ile ilgilenmeden soyut fabrika arayüzünü kullanarak istediği tipte ürünü üretmek için metodu çağırarak nesne ürettirir. Böylece istemci fabrika ve ürün detaylarından tamamen soyutlanmış olur.

Abstract Factory Patterns-Creational (Oluşturucu)

Konsept

Factory tasarım deseninde bir ürünün oluşturulması soyutlanmış iken Abstract Factory deseninde birbirleri ile ilişkili ürün ailelerininin oluşturulması soyutlanmıştır. **Factory üreten Factory deseni olarak da düşünülebilir.**

Anlayacağımız; birden fazla ürün ailesi ile çalışmak zorunda kaldığımız durumlarda, istemciyi bu yapılardan soyutlamak için Abstract Factory doğru bir yaklaşım olacaktır.



Abstract Factory Patterns-Creational (Oluşturucu)

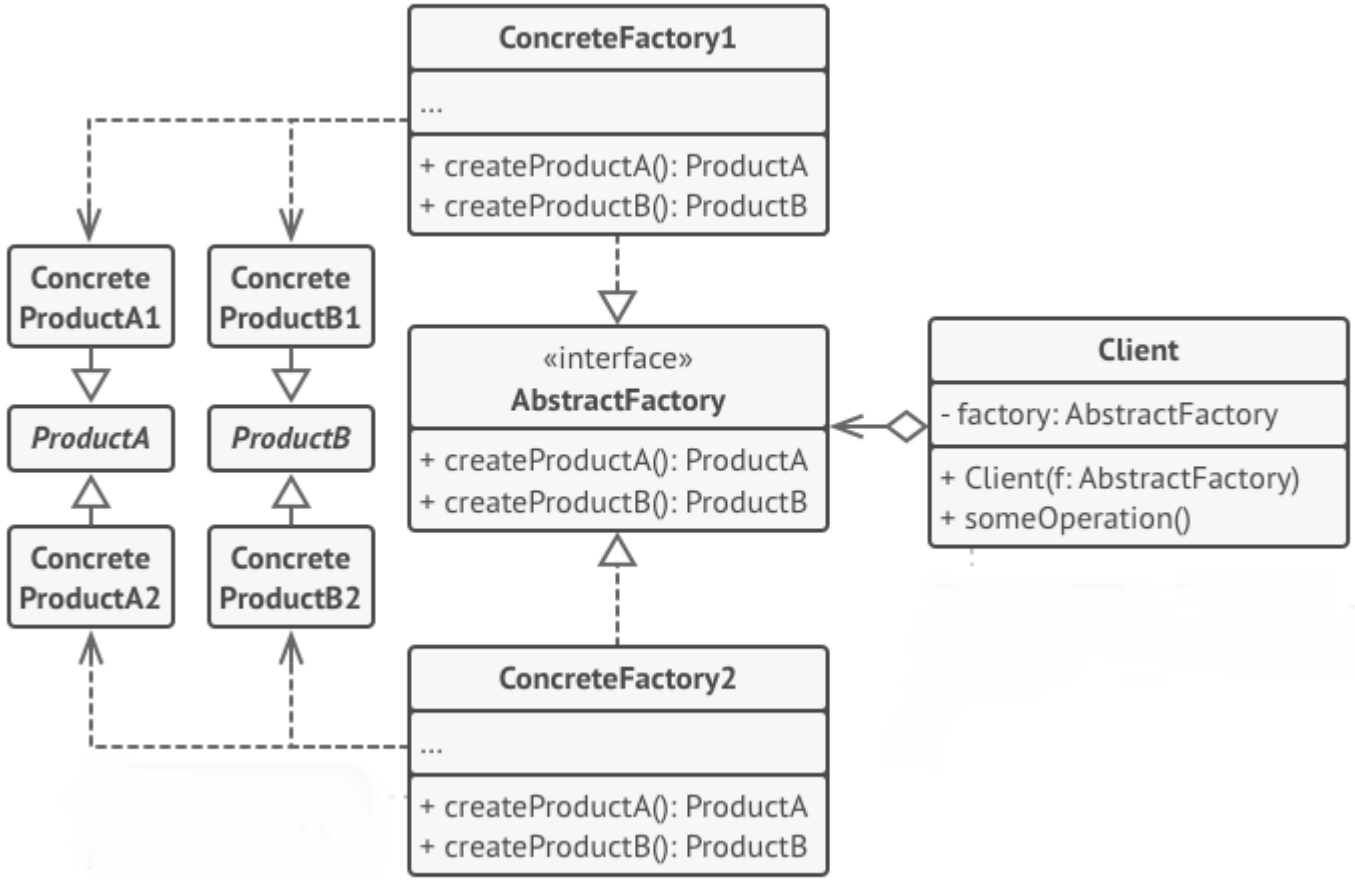
Konsept

ProductA, ProductB: Temel sınıflarımız, soyuttur ve oluşturulmasını istediğimiz sınıflar bunlardan türer.

ConcreteProduct: Üretmek istediğimiz sınıflardır.

AbstractFactory: Herbir sınıfın oluşturulması için metotların tanımlandığı arayüzdür.

ConcreteFactory: AbstractFactory arayüzünü uygulayarak gerekli sınıfların oluşturulmasını sağlar.



Abstract Factory Patterns-Creational (Oluşturucu)

Bilgisayar Dünyası Örneği

•Soyut fabrika modelini kullanarak küresel otomobil fabrikasını tasarlayalım.

•Küresel operasyonları desteklemek için , farklı ülkeler için farklı araba yapım tarzlarını desteklemek için sistemi geliştirmemiz gerekecek. Örneğin bazı ülkelerde direksiyonu sol tarafta, bazı ülkelerde ise sağ tarafta görüyoruz. Arabaların farklı kısımlarında ve yapım süreçlerinde bunun gibi daha birçok farklılık olabilir.

•Soyut fabrika modelini tarif etmek için 3 tür markayı ele alacağız – ABD , Asya ve diğer tüm ülkeler için varsayılan . Birden fazla konumun desteklenmesi, kritik tasarım değişikliklerine ihtiyaç duyacaktır.

Abstract Factory Patterns-Creational (Oluşturucu)

Bilgisayar Dünyası Örneği

•Her şeyden önce, sorun bildiriminde belirtilen her lokasyonda araba fabrikalarına ihtiyacımız var. yani USACarFactory , AsiaCarFactory ve DefaultCarFactory . Şimdi, uygulamamız kullanıldığı yeri tespit edecek kadar akıllı olmalı, bu yüzden dahili olarak hangi araba fabrikası uygulamasının kullanılacağını bile bilmeden uygun araba fabrikasını kullanabilmeliyiz. Bu aynı zamanda bizi belirli bir konum için yanlış fabrikayı arayan birinden de kurtarır.

•Temel olarak, konumu belirleyecek ve kullanıcıya tek bir ipucu bile vermeden dahili olarak doğru araba fabrikası uygulamasını kullanacak başka bir soyutlama katmanına ihtiyacımız var. Soyut fabrika deseninin çözmek için kullanıldığı sorun tam olarak budur.

Abstract Factory Patterns-Creational (Oluşturucu)

İllüstrasyon