1. Tasarım Kalıbı Ne Demektir?

a. Tanımı

Tasarım Kalıpları, Design Patterns'in dilimizdeki bir kaç karşılığından birisi. "Design"in "tasarım" olarak çevrilmesinde problem yok. Peki ya "pattern"i nasıl çevirmeli? Asıl derdimiz tabi olarak bu kelimenin nasıl çevrileceğinden çok anlamıdır. Şöyle anlatabilirim İngilizce'deki pattern kelimesinin anlamını: Ben çocukken devamlı annemin bana ördüğü kazakları ve süveterleri giyerdim. Annem hamarat bir kadındır ve dikiş, örgü vs. konularında da eğitimlidir. Çok sıklıkla, sokakta yürürken, kadınların beni durdurup, "örneğini" almak için kazağımı incelediklerini hatırlarım. Kadınların benim kazağımdan almaya çalıştıkları şey, bizim dilimizde "örnek", İngilizce'de ise "pattern"dır. Zaten Mariam-Webster sözlüğündeki açıklamalardan birisi de "something designed or used as a model for making things " şeklindedir. Bu yüzden model kelimesi belki daha iyi karşılardı "pattern"ı. Bu anlamda bence kalıp, desen ya da şablon anlamlı ve uygun çevirilerdir. Ama örüntü gibi kelimelere alışamadığımı belirteyim. Dolayısıyla tasarım kalıpları yanında tasarım şablonları ve tasarım desenleri ile de hep aynı şeyi kastediyoruz. Bu kitapta ise "tasarım kalıpları" tercih edilecektir.

Gariptir ama yazılım dünyasına bu kelimenin girişi yazılımcılarla değil Christopher Alexander isimli bir mimar ile oldu. Kendisi yüzyıllar boyunca insanlığın bina ettiği yapılardaki mimari özellikler üzerine çalışırken "design pattern" kavramını geliştirdi. Alexander tasarım kalıpları ile ilgili olarak şöyle demektedir: "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice."

Bu güzel tanımı dilimizde ifade edecek olursak: "Her bir kalıp (önce) ortamımızda tekrar tekrar olan bir problemi tanımlar ve sonra da bu problem için çekirdek bir çözümü tarif eder öyle ki siz bu çözümü bir milyon defa kullansanız bile, hiç birisini aynıyla iki defa kullanmazsınız."

Bu anlamda tasarım kalıbı, bir bağlamda sıklıkla karşılaşılan tasarım problemine genel (generic), soyut (abstract) ve tekrar kullanılabilen (reusable) bir çözümdür. Tasarım kalıpları soyutturlar, diller seviyesinde değil tasarım seviyesinde ifade edilirler. Bundan dolayı tasarım kalıpları anlatılırken sıklıkla class ve sequence diyagramlarını kullanırız. Tasarım kalıpları soyut olduklarından, her uygulanmalarında, bağlama ve ihtiyaca göre şekillendirilirler. Bu yüzden aynı tasarım kalıbını defalarca uygulasanız bile hiç bir iki uygulamanız birbirisinin aynısı olmayabilir.

b. Faydaları

Tasarım kalıpları, temel nesne-merkezli prensipleri kullanarak doğru sorumlulukları bulmamıza (finding responsibilities), değişimi göz önüne alarak bu sorumlulukları nesnelere dağıtmamıza (highly-cohesive objects) ve nesneleri, aralarında az bağımlılık olacak şekilde (lowly-coupled objects) kurgulamamıza yardımcı olur. Bir başka deyişle tasarım kalıpları, yüksek birliktelikli ve düşük bağımlılıklı yapıları nasıl kurgulayacağımızı, sıklıkla karşılaşılan problemler bağlamında, model olarak ortaya koyar. Bu anlamda tasarım kalıplarını kullanmadan doğru düzgün bir tasarım yapamayız desek sanırım abartmış olmayız. Çünkü tasarım kalıplarının çözdüğü ve bir kısmı hemen aşağıda sıralanan problemler, her ortamda sıklıkla karşımıza çıkan problemlerdir.

Tasarım kalıplarının çözdükleri problemlerden bazıları şunlardır:

- Nesneleri nasıl yaratırız?
    - Karmaşık nesneleri nasıl yaratırız?

- Nesne ailelerini nasıl yaratırız?

- Bir sınıftan sadece bir ya da belirli sayıda nesne nasıl yaratırız?
- Nesnelere erişimi nasıl kontrol ederiz?
- Nesneler arasındaki bütün-parça ilişkisini nasıl tasarlarız?
- Bir işi yapmanın pek çok yolu varsa bunları nasıl ifade ederiz?
- Emir-komuta ya da olay-bilgilendirme zincirini nasıl oluştururuz?
- Bir sürü nesne arasındaki haberleşmeyi nasıl yönetiriz?
- Bir nesneye çalışma zamanında yetkinlik nasıl kazandırırız?
- Karmaşık duruma sahip olan nesneleri nasıl yönetiriz?
- Nesnenin durumunu nasıl saklar ve sonra yine ulaşırız?
- Birden fazla nesneyi nasıl yönetiriz?
- Aynı işi birden fazla nesneye nasıl uygularız?

c. Çeşitleri

Tasarım kalıpları kavramı yayıldıktan sonra sadece tasarımın değil farklı konuların da kalıpları ortaya çıkmaya başladı. Bu sebeple "security patterns", "performance patterns" gibi kavramlar oluştu. Bu anlamda ele aldığımız tasarım kalıpları, karmaşıklığı ve değişimi yönetmek amacıyla oluşturulmuş, iş alanından ve mimari katmanlardan bağımsız modellerdir. Programlama dillerine has tipik kullanımlar ya da kalıplar da mevcuttur. Örneğin Java'da bir Map nesnesinin üzerindeki değerlerin alınması için yaptığımız tipik işlemler vardır. Bu gibi dile özel kalıplara "idiom" denir.

Tasarım kalıpları tipik olarak sınıf seviyesinde ifade edilirler. Mimari kalıplar ise daha çok modüller ve katmanlar (layer/tier) bazında ifade edilirler. Örneğin Model-View-Controller (MVC) mimari bir kalıptır. Ayrıca mimari kalıpların, tasarım kalıplarından beslendiği de söylenebilir.

d. Dörtlü Çete (Gang of Four)

Tasarım kalıpları üzerine ilk çalışmalar, yukarıda da bahsedildiği gibi Christopher Alexander tarafından yapılmıştır. Sonrasinda Trygve Reenskaug, Smalltalk'la MVC (Model-View-Controller) üzerine çalışmışlardır. Benzer şekilde 80'li yıllarda Kent Beck ve Ward Cunningham, Smalltalk ile tasarım kalıpları uzerine calışmalar yapmışlardır. 1994 yılında ise Dörtlü Çete (Gang of Four (GoF)) olarak adlandırılan Eric Gamma, Richard Helm, Ralph Johnson ve John Vlissides, ünlü "Design Patterns" kitabını yazmışlardır.

Dörtlü çete kitaplarında 3 farklı kategoride toplam 23 tane kalıba yer vermişlerdir:

- Yaratımsal (Creational) – nesneleri yaratmakla ilgili olan 5 tane,
- Yapısal (Structural) – nesneler arasindaki yapısal iliskileri ifade eden 7 tane,
- Davranışsal (Behavioral) – nesnelerin çalışma zamanı davranışlarını değiştirmek için 11 tanedir.

Sonrasında çıkan kitaplar tasarım kalıpları ,kataloğunu farklı kalıplara da yer vererek zenginleştirmişlerdir. Fakat genelde tasarım kalıpları denince akla GoF'daki 23 kalıp gelmektedir.

2. Tasarım Kalıplarını Neden Öğrenilmelidir?

Tasarım kalıplarını kullanmadan program yazmaya başladığınızda ilk yapacağınız hata, program olarak ifade etmeye çalıştığınız iş süreci ya da problemindeki sorumlulukları, roller olarak ayırt edememenizdir. Yani "bu

problemin çözümünde hangi sorumluluklar var?" ve "bu sorumluluklar farklı nesnelere nasıl dağıtılmalı?" sorularının sorulması ve cevaplanması aslen bir tasarım faaliyetidir. Hatta tasarım daha doğrusu fonksiyonel tasarım faaliyeti temelde bu soruları açıklığa kavuşturmak, alternatif cevaplar verip bunları kıyaslayarak sonuca ulaşmadan ibarettir bile denebilir. Bu noktada bu iki soruyu, bir zanaatkarın alet-edevat çantası oalrak görebiliriz. Yazılım tasarımcısının alet edavatı ise sorumluluklar ve bunların nesnelere dağıtımıdır.

Tasarım yapma niyetimiz yoksa ki bu "zamanımız yok" diye ifade ederiz, bu durumda zaten bu kitabın geri kalanı anlamsız olur. Yok, doğru düzgün bir yapı ortaya çıkarmak için tasarıma niyetimiz varsa tasarıma vakit ayırırız ki yukarıdaki soruları sorarız ve tasarım kalıplarının yardımıyla vereceğimiz cevaplarla nesne modellerini oluştururuz.

Tasarım yapılmazsa dolayısıyla da tasarım kalıpları kullanılmazsa, istenildiği kadar en yeni teknolojiler kullanılsın ya da nesne-merkezli bir dille program yazılsın, sağlıklı bir nesne yapısı kurgulama ihtimali sıfıra yakındır. Çünkü, yukarıda bahsedilen, sorumlulukları bulma ve nesnelere dağıtma işinin sağlıklı yapılmaması söz konusudur. Dikkat edilirse burada bu işlerin hiç yapılmamasından değil, sağlıklı yapılmamasından bahsediliyor. Çünkü, sorumlulukları bulma ve nesnelere dağıtma her halükarda yapılan bir faaliyettir. Fakat önemli olan bu faaliyetin keyfiyetidir. Bu işler programcı tarafından, kodu yazarken, hızlıca, birkaç saniyelik zihni bir aktivite olarak mı yapılıp geçiliyor yoksa birden fazla zihnin işin içine katıldığı ve sonucunda daha nesnel ve tekrar kullanılacak şekilde doğrulanmış ve belgelenmiş, apayrı bir yazılım geliştirme süreci olarak mı ele alınıyor? Fark buradadır. İlk halde gidilecek yön, nesne-merkezli programlama değil, olsa olsa prosedürel programlama hatta daha da kötüsü, karman-çorman kod yığınıdır.

Örneğin, normalde tasarım yaparak ve tasarım kalıplarını kullanarak nesnelere ve üzerinde muhtemelen override edilmiş metotlara bölünecek bir davranışın, tek bir metot altında pek çok "if" kullanarak kodlandığı sıklıkla görülür. Bu durum ise olsa olsa "functional decomposition"dır ve başınızı gittikçe belaya sokan bir yaklaşımdır. Dolayısıyla tasarım kalıpları çoğu defa bizi, normalde nesne-merkezli programlama dili kullanmamıza rağmen prosedürel anlayışla yazılım geliştirmeye düşmekten korur. Tam da bu yüzden pek çok refactoring yani kod iyileştirmesi tekniği, var olan karmaşık ve devasa hale gelmiş nesne ve metotları, farklı kalıpları kullanarak, ufak-tefek nesne ve metotlara dönüştürür. Tasarım kalıplarının bunu yaparken elinde tuttuğu temel teknik, yukarıda bahsedilen iki sorudur: "bu problemin çözümünde hangi sorumluluklar var?" ve "bu sorumluluklar farklı nesnelere nasıl dağıtılmalı?" Zaten bu sorular sorulmadığından bakımı ve değiştirilmesi zor hatta imkansız sınıflara ve metotlara ulaşılmıştır.

Ben danışmanlıklarımda, özellikle kod kalitesini arttırıcı işlerimde bu durumlarla çok sık karşılaşıyorum. Tek yapacağım şey konuyu anladıktan sonra bu iki soruyu sormak. Tasarım kalıpları, bu iki sorunun cevabını ciddi oranda veriyorlar. Çünkü tasarım kalıpları hem bir yöntemdir hem de bir sorumluluk kataloğudur. Tasarım kalıpları yöntemdir çünkü bu iki soruyu sormanızı ve sorduğunuzda da nasıl cevap vereceğinizi size öğretir.

Tasarım kalıpları aynı zamanda bir sorumluluk kataloğudur, çünkü en basitinden en karmaşığına kadar, farklı sektördeki yazılımlarda bulunan en yaygın sorumlulukları ortaya koyar. Dahası tasarım kalıpları, bu sorumlulukları nasıl bulacağınızı ve bunları iç tutarlılığı yüksek (highly-cohesive) ve dış bağımlılığı düşük (lowly-coupled) nesne modellerine nasıl çevireceğimizi de bize gösterir.

Eğer tasarım faaliyeti, yazılım geliştirme sürecinizin bir parçası ise zaten yolunuz bir şekilde kalıplarla kesişiyor demektir. Yani sadece Gof'un 23

kalıbını kullanmakla kalmıyor, kendi iş alanınızda sıklıkla karşılaştığınız sorumluluk problemlerine özel kalıplar da geliştiriyorsunuz demektir.

Dolayısıyla tasarım kalıplarını şu amaçlarla öğrenilmelidir:

- Tekerleği yeniden keşfetmemek, var olan ispatlanmış çözümleri kullanmak: Tasarım kalıpları sıklıkla karşılaştığımız, yaygın sorumluluk bulma ve dağıtma problemlerini çözerek, tekrar kullanılabilen, faydası ispatlanmış modeller ortaya koyar.
- Formal ve yaygın bir dil oluşturmak: Tasarım kalıpları, formal bir kalıp dili (pattern language) oluşturmaktadır. Bu kalıp dilinin en temel kavramı sorumluluktur. Dolayısıyla tasarım kalıpları zihnimize, "şu şey yapar, şunu alır şuna verir" gibisinden sokak ağzı bir konuşma yerine "kontrol nesnesi (controller), request ve response nesnelerini, ilgili üreticilerden (factory) alıp sunum (view) nesnesine geçer" şeklinde konuşmayı öğretir. Bu dilin yazılım geliştiriciler arasında yayılması, anlaşmayı kolaylaştıracak, ifade gücümüzü arttıracaktır.
- Tasarıma uygun, yüksek soyutlama gücü kazanmak, detaylardan sıyrılıp, daha yüksek hedefler cinsinden düşünmek: Bu anlamda tasarım kalıpları daha iyi bir yazılım geliştirici hatta mühendisi olmanın en temel gerekliliklerindendir.

Zaman zaman karşılaşıyorum, "biz aslında bir miktar tasarım yapmıştık" şeklinde cümlelerle. Bırakın kalıbı sistemde ne bir arayüz var, ne bir nesne hiyerarşisi var, sınıflar devasa, metotlar yüzlerce satır, nesneler arası ilişkiler karmakarışık, kim neyi biliyor belli değil. Peki tasarım nerede? Yukarıda anlatıldığı gibi tasarım yapılacaksa o iki sorudur üzerinde düşünülmesi ve çözülmesi gereken şey. Ve yolunuz bir şekilde tasarım kalıplarıyla kesişir. Dolayısıyla tasarım kalıplarını bilmiyorsanız tasarım da yapamazsınız.

3. Tasarım Kalıplarını Nasıl Öğrenilmelidir?

Bir kere peşinen şunu söylemekte fayda var: tasarım kalıpları bir teknoloji değildir. Bu yüzden de tonla farklı konuda kitap okuyan, karıştıran ya da bilgi konusunda abur-cubur beslenip bilgi obezitesi ya da bilgi sindirimi bozukluğu gibi sıkıntılara duçar olmuş pek çok genç arkadaşın, benzer düşünceyle tasarım kalıplarına yaklaşması çok ciddi zihinsel sorunlar oluşturacaktır. Demek istediğim şu: Bir kitabı okumak, bir yaklaşımı öğrenmek ya da bir fikri hazmetmek için belli bir zihni olgunluğa gelmiş olmanız gereklidir. Benzer şey tasarım kalıpları için de geçerlidir. Tasarım kalıplarını öğrenmek, kişiyi en çok soyut düşünme konusunda zorlar ve aynı zamanda da geliştirir. Hatta bu gibi soyut yapıları öğrenmek için odaklanmanız da gerekir çünkü kavranmaları zordur. Odaklanmak her başarının ardındaki önemli noktalardan ama tasarım kalıplarını öğrenmek için teknolojinin ötesine odaklanmanız gerekecektir.

a. UML

Tasarım kalıplarını öğrenmek, kişiyi en çok soyut düşünme konusunda zorlar ve aynı zamanda da geliştirir. Soyut yapılar üzerinden konuşmak ve anlaşmak ise zordur. Bu yüzden özellikle mühendislikte modelleri betimlemek ve göstermek amaçla oluşturulmuş notasyonlar ve standartlar vardır. Unified Modeling Language (UML) ise, yazılımdaki modelleri betimlemek amacıyla oluşturulmuş görsel notasyon ve ilgili standardın ismidir. Biz de bu kitapta kalıplarımızın modellerini ifade etmek için UML diyagramlarını kullanacağız. Unun için bir miktar UML öğrenmeniz, özellikle de sınıf (class) ve akış (interaction) diyagramlarına bakmanız yerinde olur.

b. Programlama Dili

Tasarım kalıpları, nesne merkezli tasarım yapmanın kalıplarıdır. Dolayısıyla tasarım kalıplarını öğrenmek için nesne-merkezli temel soyutlamaları ve mekanizmaları iyi bilmek gereklidir.

Bu temel soyutlamalar ve mekanizmalar da şunlardır:

- Sarmalama (encapsualtion), durumu (state) ve arayüzü (interface) ve bunları bir sınıfta ifade etme,
- Bilgi saklama (information hiding) ve erişim yönetimi,
- Nesneler arası ilişki (association) ve has-a ilişkisi,
- Kalıtım ya da miras (inheritance) ve is-a ilişkisi ile sınıf ve arayüz kalıtımı,
- Çok şekillilik (polymorphism).

Yukarıdaki mekanizmaları iyi bir şekilde bilmenin yolu, çoğunlukla nesne merkezli bir programlama dilini iyi bilmekten geçer. Çünkü, nesne merkezli soyutlamalar ve mekanizmaları, ezici çoğunlukla bir nesne merkezli dil ile programlama yaparken öğrenilir. Dolayısıyla yukarıdaki yapılara sahip, C++, Java, C#, Python, vb. herhangi bir nesne merkezli dil, tasarım kalıplarını öğrenmek amacıyla kullanılabilir. Zaten kalıpları örnek gerçekleştirmeleri de bu dillerde yapılır. Biz bu kitapta tüm örnekleri Java ile yapacağız.

Bu kitaptan faydalanmak için Java bilmeye gerek yoktur. Yukarıdaki kavramları herhangi bir nesne-merkezli programlama diliyle gerçekleştirebilecek kişi, bu kitaptaki anlatım ve Java ile gerçekleştirilmiş kodlardan yola çıkarak kendi kullandığı dilde kendi kodunu yazabilir, yazmalıdır.

c. Kitaplar

Tasarım kalıplarını öğrenmek için ne gibi malzemelerimiz var? Önce kitaplardan başlayalım. Tasarım kalıplarını öğrenmekten bahsedildiğinde tabi olarak akla ilk gelen kitap GoF'un Design Patterns isimli kitabıdır. İşin açıkçası bu kitap 1994'de yazılmış olmasına rağmen henüz devrini tamamlamadığı gibi, pek aşılmış da değildir. Konu ile ilgili yazılmış pek çok kitap, bu kitabın tefsiri ya da farklı programlama dillerine uygulaması mahiyetindedir. Gerçi bu cins açıklayıcı kitaplar arasında çok başarılar da var elbette ama temelde bu kitapların pek çoğu, Design Patterns kitabına orijinal bir şey katmazlar.

Tecrübeli olmayanlar için Design Patterns zor bir kitaptır. Ciddi bir nesne altyapınız yok ise, nesne-merkezli dillerle uzunca süredir uğraşmıyorsanız, sırf ismi çok sık geçiyor diye bu kitabı okumaya çalışmak çok da faydalı bir hareket değildir, sadece öğrenilmiş çaresizlik üretir. Bu kitabı anlamak için yazılım üzerine çok ciddi soyut düşünce yeteneğine sahip olmak gereklidir. Kitabın örnekleri C++ ile verilmiştir ama Java ve C# gibi dillerde örnekleri olan pek çok açıklama ve uygulama mahiyetinde kitaplar da mevcuttur. Ben egitimlerimde bu kitabı kullanıyorum, burada da sıklıkla bu kitaba atıflar yapacağız.

Design Patterns kitabının ilk iki bölümü nefis bir OO özetidir, hiç okuyamazsanız, bu iki bölümü, altını çizerek ve anlayarak okumanızı tavsiye ederim. Özellikle ilk bölümdeki nesnelerin bulunması, sorumluluklarının atanması, interface ve implementation ayrımı ve ilgili miras yapıları son derece keyifle okunacak konulardır. Yine bu bölümdeki "program to an interface, not an implementation", "favor object composition over class inheritance" ve "design for change" prensipleri, her daim akılda tutulması gereken en temel yazılım tasarımı prensiplerdendir. Bu kitabı okumaya kesinlikle ilk iki bölümünü atlayarak başlamayın. Bu iki bölümü hazmederek okuyun sonrasında kalıplara geçin. Ben tasarım kalıpları eğitimlerinde,

sınıfın durumuna göre bazen 1-1,5 günü bu üçünün yanında "tek sorumluluk" gibi diğer başka prensipleri de örneklerle açıklayarak geçiriyorum, aksi taktirde katalog gibi kalıpların üzerinden gitmenin bir faydasını görmem mümkün olmuyor.

Peki bu kitabı okuması zorsa ne ile başlayalım? Uzunca bir müddet burun kıvırarak baktığım Head First serisi kitaplarından olan "Head First Design Patterns" kitabı ile son zamanlarda biraz haşır neşir oldum ve bu seri ile ilgili fikrim değişti. Bu kitap GoF'un kalıplarının ciddi bir kısmını son derece eğlenceli tarzda anlatıyor. Ben kitabı "Lise mezunları için Tasarım Kalıpları" olarak niteliyorum zaman zaman. Temel nesne-merkezli prensipleri bilen, hatta bu konularda kafası karışık olan kişilerin bile sabrederek ilerlediklerinde çok faydalı olacak bir kitaptır. Bu kitapla, gayret ederseniz, muhakkak konuyu anlarsınız. Bu kitap hem anlaşılır, detaylı anlatımı, hem konuyu özetleyen güzel UML çizimleri hem de son derece kolay örnekleri ile tasarım kalıplarının dünyanıza girmesini sağlayacaktır. Kod örneklerini indirip, konularla atbaşı işleyebilirsiniz.

Head First Design Patterns kitabı, kalıplardan önce temel prensipleri açıklayan bir ön bölüme sahip. Aslında pek çok prensibi kalıpları anlatırken yapıyor. Fakat ilk bölümde, yazılımın en temel özelliklerinden olan "değişim"i ele alarak, GoF'un yukarıda bahsettigim üç prensibini açıklıyor. Bu amaçla ördekler üzerine kurgulanan bir problemi önce kalıtım sonra da arayüz ile tasarlıyor. Kitap sonrasında GoF'un kalıplarının yaklaşık yarısını, 11 bölümde ele alıyor. Geriye kalan kalıplara da son bölümde kısaca değiniyor. Bu kitabın genişçe ele aldığı kalıplar, aslında öğrenilmesi en kolay olanlar. Dolayısıyla kitap okuyarak ve uygulamalarını yaparak kalıpları öğrenmek istiyorsanız bu kitap tam bu iş için.

Benim tasarım kalıpları konusunda faydalandığım bir başka kaynak da A. Shalloway'in Design Patterns Explained isimli kitabıdır. Bu kitap GoF'un açıklaması mahiyetindedir ama bir kalıp kataloğu değildir.

Dolayısıyla kitap daha çok kalıp merkezli düşümeyi sağlamak üzere, temel nesne-merkezli yaklaşım problemlerini ele almakta ve kalıplarla bu problemlerin nasıl giderildiğini anlatmaktadır. Kitabın örnekleri Java ile yapılmıştır ve bu örneklere online olarak ulasabilirsiniz. Kitapta, GoF'un Facade ya da Strategy gibi sadece bazı kalıpları yer almaktadır. Head First kitabi size çok basit geliyorsa, bu kitapla başlayabilirsiniz.

Craig Larman'ın geniş ve güzel "Applying UML and Patterns" kitabından bahsetmeden geçmek olmaz. Bu kitap aslen bir kalıp kataloğu değildir. Bu kitap hem temel prensipleri hem de bazı kalıpları, UML kullanarak, baştan sona bir tasarım süreci içerisinde ele alır ve detaylıca anlatır. Bu kitap kendi kendinize çalışmak için de nefis bir kaynaktır. Bu kitabın pek çok üniversitede UML ile nesne-merkezli analiz ve tasarım amaçlı derslerde kullanıldığını da biliyorum.

d. Türkçe Kitaplar

Kaynaklardan bahsedince tabi olarak "Türkçe kitap yok mu?" sorusu akla geliyor. Malesef dilimizde bu konularla ilgili özgün kitap pek yok. Zaten okuyan bir millet olmadığımız için, yazmak hepten bize uzak. Az da olsa bizim sektörümüzde fedakarlık yapıp, zaman ayırıp, güzel kitaplar yazanlar var. Bunlardan birisi de Özcan Acar. Özcan, "Java Tasarım Şablonları ve Yazılım Mimarileri" isimli 290 sayfalık kitabında GoF'un 22 kalıbı yanında Java EE'nin de en bilinen mimari kalıplarını ele almıştır. Kitabın giriş mahiyetindeki ilk bölümünde soyut sınıflar ve arayüzler ele alınmakta ve altı temel prensipten bahsedilmektedir. Kitabın ikinci bölümü ise tasarım kalıpları kavramını açıklamaktadır. Kitabın 3., 4. ve 5. bölümlerinde GoF'un kalıpları, 6. bölümde ise Java EE kalıpları açıklanmaktadır. 7. bölümü ise tasarım kalıplarının, 3 katmanlı bir yazılım mimarisi içinde nasıl uygulandığı bir örnek uygulama ile ele alınmıştır. Son bölüm ise Spring mimarisi ve bu mimaride Data Access Object (DAO) kalıbının kullanımını ele almaktadır.

Kitapta konuların sıklıkla UML sınıf diyagramlarıyla zenginleştirilmiş olması anlaşılmayı kolaylaştırmaktadır.

Özcan, muhtemelen okumama alışkanlığımızı aşmak için kitabını olabildiğince kısa tutmaya çalıştı diye düşünüyorum. Zaten önsözünde de belirttiği gibi anlaşılması için Java örneklerini olabildiğince basit halde vermiştir. Kitap ile birlikte gelen CD'den kitabın örneklerini edinebilirsiniz.

e. Online Kaynaklar

Ayrıca DZone'un konu ile ilgili referans kartlarını (https://dzone.com/refcardz/design-patterns) da elinizin altında tutmak isteyebilirsiniz.

Zaman zaman konuyla ilgili baktığım bazı web kaynakları da şunlardır:

- Design Patterns Library: http://c2.com/cgi bin/wiki?DesignPatterns ve http://c2.com/cgi-bin/wiki?SoftwareDesignPatternsIndex
- http://www.oodesign.com/
- http://www.vincehuston.org/dp/ Burada problemlerin kalıp uygulanmadan önceki ve sonraki halleri kod olarak verilmektedir.
- http://sourcemaking.com/design_patterns
- http://www.javacamp.org/designPattern/

Konu ile ilgili bazı güzel makaleleri de şöyle verebilirim:

- Robert C. Martin'in "Principles and Patterns" makalesi
- Martin Fowler'in "Writing Software Patterns" makalesi

4. Tasarım Kalıbının Tanımlanması

Bir tasarım kalıbının temelde dört bileşeni vardır:

İsim: Kalıbın ismi, o kalıbı, problemi ve çözümüyle birlikte ayırt etmemizi sağlar. Dolayısıyla kalıpların isimleri önemlidir ve genelde ilk bulanın koyduğu isimle anılır.

Problem: Tasarım kalıbının hangi bağlamda ve nasıl ortaya çıktığını ifade eder. Genelde sözel anlatım olarak verilir.

Çözüm: Tasarım kalıbının problemi nasıl çözdüğünü, bu amaçla kullandığı parçaları ve aralarındaki ilişkileri ifade eder. Çözüm, genelde UML'in class ve sequence diyagramları ile soyut olarak betimlenir, örnek bir gerçekleştirme ise C++ ya da Java gibi uygun bir nesne-merkezli dilde verilir.

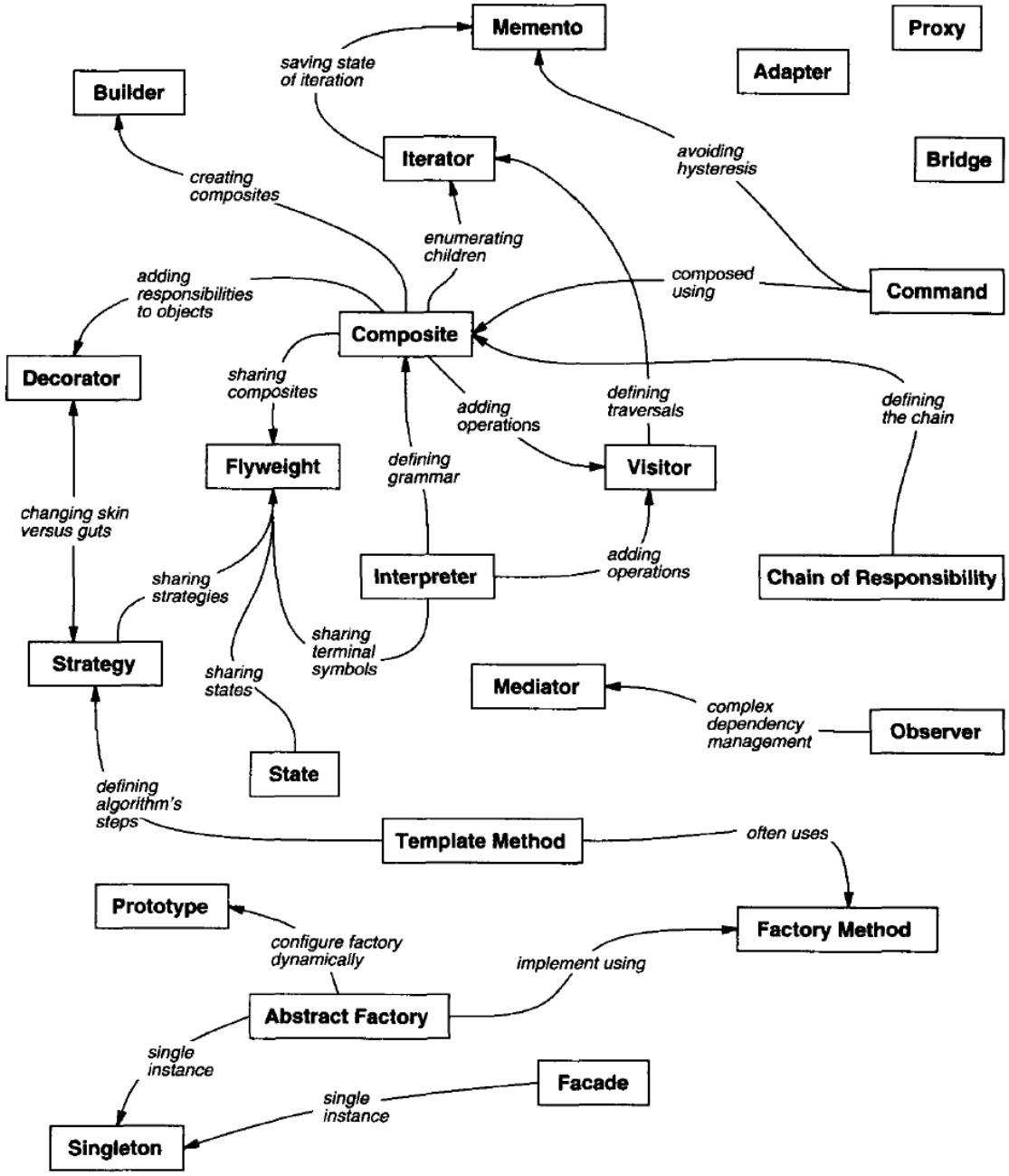
Sonuçlar: Kalıbın sağladıklarını ifade eder. Kalıp hem bazı kazanımlar sağlar hem de bunların karşılığında bazı kayıplara sebep olur. Sonuçlar, ayrıntılı olarak bunları ifade eder.

GoF kitabında yukarıda sayılan dört bileşeni, daha detaylı bir şekilde ele almışlar ve aşağıdaki gibi bir kalıp tanımlama yapısı kullanmışlardır. Aşağıdaki listede var olanların bazıları doğrudan yukarıdaki dört maddeden birine denk gelmektedir.

- **İsim (Name)**: Kalıbın ismidir.
- **Amaç (Intent)**: Kalıbın amacıdır.
- **Problem/Motivasyon (Problem/Motivation)**: Kalıbın çözmeye çalıştığı, geniş bir alandaki pek çok özel problemi ifade eden ve soyut olarak tanımlanan tipik problemdir.
- **Çözüm/Yapı (Solution/Structure)**: Ortaya çıktığı bağlamda kalıbın verdiği çözümdür.

- **Katılımcılar ve Paydaşları (Participants and Collaborators)** : Çözümde var olan ana nesneler ve onlarla beraber bulunan arayüz, yardımcı nesne vb. diğer yapılardır.
- **Sonuçlar (Consequences)**: Kalıbın sağladıklarını ifade eder.
- **Gerçekleştirme (Implementation)**: Kalıbın önerdiği çözümün gerçekleştirilmesidir.
- **Genel Yapı (Generic Structure)**: Genelde çözümün, UML'in class ve sequence diyagramları ile soyut olarak çizimidir.
- **Uygulanabilirlik (Applicability)**: Kalıbın uygulanabileceği bağlamları ifade eder.
- Örnek Kod (Sample Code): Gerçekleştirmeden bahsedilen ve nesne-merkezli bir dilde yazılan örnek kod parçalarıdır.
- **Bilinen Kullanımlar (Known Uses)**: Gerçek sistemlerde bilinen kullanımları.
- **İlgili kalıplar (Related Patterns)**: İlgili diğer kalıplar, aralarındaki benzerlik ve farklar ifade edilir.

GoF kitabının sonunda, kalıplar arasındaki ilişkileri şöyle bir çizimle vermişlerdir:


Memento
Proxy
saving state of iteration
Adapter
Builder
Iterator
avoiding hysteresis
Bridge
creating composites
enumerating children
adding responsibilities to objects
composed using
Command
Composite
Decorator
sharing composites
adding operations
defining traversals
defining the chain
Flyweight
defining grammar
Visitor
changing skin versus guts
sharing strategies
Interpreter
adding operations
Chain of Responsibility
Strategy
sharing terminal symbols
sharing states
Mediator
complex dependency management
Observer
State
defining algorithm's steps
Template Method
often uses
Prototype
Factory Method
configure factory dynamically
implement using
Abstract Factory
single instance
Facade
single instance
Singleton