# Spring Framework'ünün Kullandığı Design Patternler

Tasarım kalıpları projelerimizde karşılaştığımız yazılım sorunlarına standart çözümlerle yaklaşıp, yazdığımız kodları kod tekrarından kurtarıp daha esnek, performansı yüksek ve yeniden kullanılabilir hale getirmemize yardımcı olur.

## Proxy Design Pattern:

Proxy tasarım deseni, Structural grubuna ait olup, web servisi ve remoting uygulamalarda oluşturulması zaman alan ve sistem kaynaklarını zorlayan nesnelere vekalet edilmesi için kullanılan bir yazılım mimarisidir.

Proxy Pattern kullanmak için bir nesneyi temsil eden başka bir proxy(vekil) nesne kullanılır. Bu vekil nesne diğer nesnenin tüm metotlarına erişebilir ve kullanıcı sınıf ile vekil olunan nesne arası aracılık yapar. Bu sayede Proxy sınıf, esas sınıfa müdahale etmeden istediği tüm işlemleri gerçekleştirebilir. Proxy tasarım kalıbı ile kod tekrarı yapmayız ve sistem performansımızda düşüklük yaşamayız.

Somut bir örnek vermek gerekirse; projemizde bir sınıfımız(class) var ve nesne olarak emlak satış sitesindeki tüm satıcıların bilgilerini tutsun. Satıcı bilgilerimiz arasında satıcı tipleri(kurumsal,bireysel), ad soyad bilgisi, ilanları ve mesajları olsun. Her işlem yaptığımızda bu emlak satış sitesindeki satıcıları bir defa dolaşıp, gerekli bilgilere sahip olsun. Burada client olarak tüm bu bilgiler yerine, sınıflardaki bize gerekli olan bilgilere ihtiyacımız olabilir. Böyle durumlarda Proxy Class oluşturup sadece bize lazım olan metotlara ve kullanıcılara erişim izni verebiliriz. Client oluşturulan, bu proxy nesnesi kopyaladığında orijinal nesnenin kopyalanmasına göre daha kısa sürer. Böylece oluşturulması zaman gerektiren ve karmaşık işlemlerin kontrolünü daha rahat bir şekilde gerçekleştirebiliriz.

## Singleton Design Pattern:

Singleton tasarım deseni, Creational Design Pattern grubuna ait olup projemizde kullanacağımız nesneden sadece bir tane yaratmak istediğimizde kullanırız. Neden Singleton kullanmamız gerekir diye düşünürsek; bir nesnenin yapacağı işlem parametrelere göre değişirse o nesneden birkaç tane yaratmanın anlamı yoktur. Her nesne bellekte yer kaplar ve aynı işleve sahip nesne üretirsek fazla kaynak tüketimine sebep oluruz. Bu yüzden singleton yapma ihtiyacı duyarız. Singleton tasarım kalıbını database işlemlerinde, uygulama ayarları gerektiren işlemlerde ve bağlantı işlemlerinde kullanırız.

Singleton yapmanın bazı kuralları vardır:
Bir sınıftan sadece bir instance alabiliriz. Nesne yaratmak için sınıfın Constructor üyesi üzerinden oluşturur. Sınıfa ait instance ulaşmak için global erişim yapmalıyız. Constructor üyesini private olarak yapıp, new anahtar sözcüğü kullanarak nesne üretilmesini engelleriz. İnstance alabilmemiz için public static bir metot kullanmamız gerekir.

## Factory Design Pattern

Factory Design Pattern, uygulamamızda bir superclass'ın nesneler oluşturması için bir interface oluşturmasını sağlar ve subclass nesnelerin tür değiştirmesine izin veren creational design pattern grubuna ait tasarım desenidir. Kalıtım özelliği olan nesnelerin üretilmesi amacıyla kullanılır. Factory method sayesinde var olan nesneleri yeniden oluşturmak yerine yeniden kullanarak sistem kaynaklarını daha verimli bir şekilde kullanabiliriz. Veritabanı bağlantıları, dosya sistemleri ve ağ kaynakları gibi büyük kaynak gerektiren nesneler için kullanabiliriz. Fabrika yöntemleri genellikle Constructor methodlarında private ya da protected erişim belirleyicisi kullanılır.

## Template Design Pattern

Behavioral Design Patterns grubuna ait olup, suberclass'da algoritmanın bir şablonunu oluşturup, subclass'da yapıyı değiştirmeden algoritmanın belirli adımlarının değiştirilmesi için kullanılan bir yazılım mimarisidir. Template method, algoritma yapısını bozmadan kod tekrarından kurtulmamızı sağlar.

## MVC Design Pattern

Model View Controller projemizin modüler bir yapıda olmasını sağlayıp yeniden kullanılabilirliği ve proje kontrolünü arttıran, uygulamaların business logic ile kullanıcı arayüzüne birbirinde ayıran yazılım mimarisidir.

Projemizde değişiklik yapmak istediğimizde MVC'nin gerekli katmanında geliştirme ve güncelleme yapabiliriz.
MVC avantajları:
Model View Controller katmanlarının ayrı olması sebebiyle bir projede birden fazla yazılımcı eş zamanlı olarak aynı proje üzerinde farklı katmanlarda sorunsuz bir şekilde çalışıp daha sonra bu katmanları birleştirebilirler. Böylece proje yönetimini daha verimli yapmış oluruz. Model, View ve Controller katmanında tam olarak hangi işlemlerin yapıldığına kısaca bakarım.

Model: Yazılım projemizdeki verilerin tutulduğu ve validasyonu yapıldığı yerdir. Örneğin emlak satış sitesi yapmak istiyorsak ilan modeline ihtiyaç duyarız. İlan modelimizde satacağımız ev tipi, konumu, oda sayısı ve fiyatı gibi nesnelerimiz yer alır. Projemizin büyüklüğüne göre Model tek katmandan oluşabileceği gibi birden fazla katmandan da oluşabilir.

View: Yazılım projemizin arayüz katmanıdır. View kullanıcılardan alınan istekleri Controller katmanına iletir.

Controller: MVC yapısının temel amacı Model ve View yapısının birbirinden ayrılmasıdır. Model ve View arasındaki iletişimi sağlar. Projemizdeki tüm işlemlerin (Veri tabanı işlemleri hesaplamalar vb.) yapıldığı katmandır. Model nesnelerinde veri akışını kontrol edip, veri değiştiğinde View katmanı güncellenir. Somut bir örnek vermek gerekirse; Emlak satış sitesindeki kullanıcı ayarlar sayfası Controller katmanı olarak gösterilebilir. Kullanıcı girişlerinde yapılan değişiklik Model katmanına iletilir ve eğer değişiklik arayüz ile ilgiliyse View katmanında güncelleme yapılır.

**Front Controller Design Pattern**

Front Controller Design Pattern, web uygulamalarında gelen bütün istekleri karşılayan bir controller sınıfı üzerinden işlemleri gerçekleştirir. İstekleri merkezi bir yerde karşılaşması sebebiyle kimlik doğrulama(authentication), yetkilendirme(authorization) ve loglama işlemlerini gerçekleştirip, kod tekrarını azaltmaya yardımcı olur.

**Prototype Design Pattern**

Prototype tasarım deseni, Creational Design Pattern grubuna ait olup projemizde kullanacağımız nesneleri new operatörüyle oluşturmanın maliyetini azaltmak için kullanılır. Bir nesneyi birden fazla kullanmak zorunda olduğumuzda nesnenin clone(kopyasını) alırız. Diğer oluşturulacak nesnelerde bu nesnenin prototipi üzerinden üretilir. Üretilen nesnenin çok fazla kaynak tüketmesi engelleyebilmek ve projemizin sınıflardan bağımsız olması gereken yerlerde Prototype kullanabiliriz.

**Dependency Injection Design Pattern (DI)**

Bir sınıfının bağımlı olduğu nesnelerden bağımsız hareket edebilmesini sağlamak için bağımlılık oluşturacak nesnelerin dışardan verilmesiyle sınıflar içerisindeki bağımlılığı azaltmak için kullanılır. DI sayesinde bağımsızlığı sağlanan sınıflar tek başına test edilebilir, kod yapımız daha esnek ve yeniden kullanılabilir olur.

**Referanslar**


https://www.turkayurkmez.com/singleton-design-pattern/
http://www.kurumsaljava.com/2009/10/08/proxy-vekil-tasarim-sablonu/
https://www.evrenbal.com/proxy-tasarim-deseni-nedir/
https://blog.koddit.com/yazilim/mvc-nedir-gercek-orneklerle-mvc-nedir-anlayalim/
https://www.javaguides.net/2020/02/design-patterns-used-in-spring-framework.html
https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm
https://www.gencayyildiz.com/blog/front-controller-design-pattern-nedir-nasil-uygulanir/
https://muhendisyagmursari.wordpress.com/fabrika-deseni/
http://ucemucar.com/design-patterns/factory-pattern/