SPRING FRAMEWORK DE KULLANILAN DESIGN PATTERNLER

Spring framework birçok Java severin JavaEE uygulamaları oluşturmak için kullandığı, geliştiriciler için yorucu, onlardan vakit çalan konfigurasyon işlemlerini kolaylaştıran açık kaynak kodlu bir frameworktur.

Spring literatürde 'framework of frameworks' olarak anılır. Böyle anılmasının sebebi Struts, Hibernate, Tapestry, EJB, JSF gibi frameworklerin kullanılmasını desteklemesidir.

Şimdi hep birlikte önce design pattern kavramı nedir açıklayalım sonrasında da Spring frameworkte sıkça başvurulan patternlere göz atalım.

DESIGN PATTERN NEDİR ?

Yazılım geliştirirken zaman zaman hepimiz içinden çıkamadığımız hatalarla karşılaşmışızdır. Bazen bu hataların çözümünü bulamazken bazen oldukça vakit harcamışızdır. Daha önce bizim geçtiğimiz yollardan geçen yazılım geliştiriciler bu sorunların çözümünü ele almış ve Design Patternleri bize sunmuştur.

Gang Of Four(Dörtlü Çete) 'nin 1994 te çıkardığı Design Pattern kitabında 23 kalıba yer verilirken, günümüzde bu sayı çok daha fazladır. Bizler kendimiz de yazılım geliştirirken kendi kalıplarımızı belirleyip buna göre kodlama yapabiliriz.

En çok kullanılan Design Patternler;

1-Singleton

Singleton Design Pattern ‘ i Creational Patternler arasında yer alır. Çalışma zamanı boyunca nesnenin tek Instance ‘ i olacağını garanti eden bu pattern, Instance yoksa yaratır, varsa aynı Instance’i bizlere döndürür.

Şimdi bir Singleton Sınıfı oluşturarak kod örneğimizi verelim;

```
1   public class Singleton{
2
3   // Sınıfımızın Construcutor Methodu
4   private Singleton(){
5   }
6
7   // static değişkenimiz singleton class'ımızın instance'ı
8   private static Singleton singleton;
9
10  private static Singleton getInstance(){
11
12      If(singleton == null )
13          singleton = new Singleton();
14      return singleton;
15
16      }
17  }
18
```

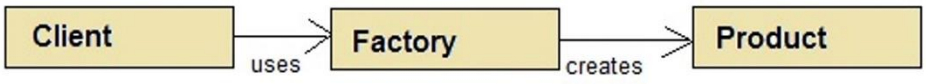
Basit şekilde oluşturduğumuz bu Singleton sınıfından nesne yaratmak istediğimizde getInstance() isimli methodu çağıracağız. Eğer nesnemiz mevcutsa bize ramde bulunan Instance’i gönderecektir.

Kısaca Singleton patterni ve örneklemesi bu kadardı.

2- Factory

Object Oriented Programming yaklaşımı ile günlük hayattan örnek vermek gerekirse, bazı basit nesnelerin üretim süreci kolay olmakta bunları evde kendimiz bile üretebiliriz. Fakat bazı nesnelerin üretim süreçleri daha karmaşık bir yapıya sahiptir. Bu nesnelerin üretimini bizim yerimize özel fabrikalar gerçekleştirir ve üretim aşamalarını biz kullanıcılardan gizleyerek bu süreci yönetirler.

Factory Pattern de tam olarak böyle bir yaklaşıma sahiptir. Oluşturulan nesneleri kullanacak sınıflar bu süreçten haberdar olmak detayları bilmek istemezler.



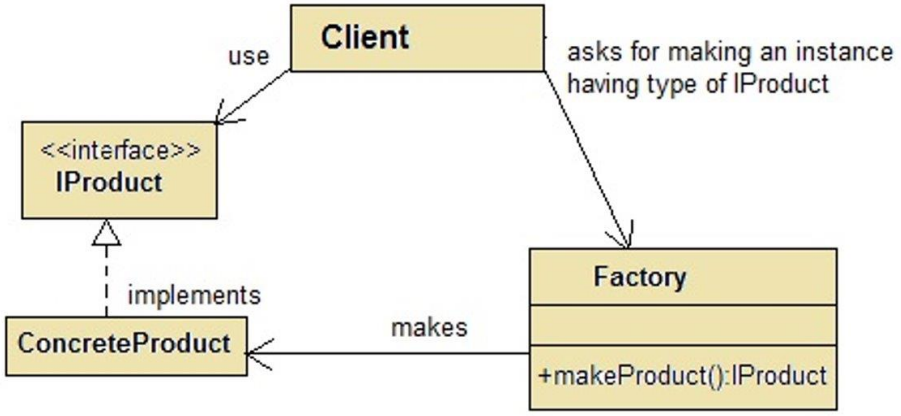
Client olan nesne Factory nesnesini kullanarak ihtiyacı olan Product nesnesini elde eder.

Nesne oluşturma sürecinin Factory sınıfına aktarılması ile birlikte Client sınıfı bu işlemden soyutlanmış olur. Bu sayede Client sadece kendi rolüne odaklanır.

Factory Pattern genel olarak:

İstenen tipte nesne oluşturma sürecini Client'ın bu konuda detay bilgi sahibi olmadan gerçekleştirilmesini sağlar.

Yeni oluşturulan nesneye bir interface ile referans edilerek ulaşılmasını sağlar.



Factory design patern i bir de kod üzerinde görelim;

```
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        SekilFactory sf = new SekilFactory();

        ISekil Sekil_1 = sf.GetSekil(0); //Kare sınıfı ISekil arayüzünü implemente ettiği için(kalıtım gibi) böyle bir atama yapılabilir.
        Sekil_1.Ciz();

        ISekil Sekil_2 = sf.GetSekil(1);
        Sekil_2.Ciz();

        Console.ReadKey();
    }

    5 references
    public interface ISekil //Aynı işlem abstract sınıf ile de yapılabilirdi
    {
        4 references
        void Ciz();
    }

    1 reference
    public class Kare : ISekil
    {
        4 references
        public void Ciz()
        {
            Console.WriteLine("Kare çizildi.");
        }
    }

    1 reference
    public class Cember : ISekil
    {
        4 references
        public void Ciz()
        {
            Console.WriteLine("Çember çizildi.");
        }
    }

    2 references
    public class SekilFactory
    {
        2 references
        public ISekil GetSekil(int secim)
        {
            switch (secim)
            {
                case 0:
                    return new Kare();

                case 1:
                    return new Cember();

                default:
                    return null;
            }
        }
    }
}
```

3 – MVC

MVC design patterni birçok yazılım ve frameworkte kullanılmakta olan ve Java Spring Frameworkte de yaygın olarak kullanılan patternlerden biridir.

Model-View-Controller yapılarını içinde barındıran ve 3 katmandan oluşan bu pattern, yazılımcılara proje geliştirme sürecinde oldukça yardımcı olur. Çünkü her katmandan birbirinden bağımsızdır ve takım içerisindeki kişiler ayrı bir şekilde geliştirme süreçlerine devam edebilirler.

Çoğunlukla büyük çaplı projelerde tercih edilen bu pattern proje yönetimini ve kontrolünü kolaylaştırır.

Şimdi hep birlikte Model- View- Controller terimlerinin ne anlama geldiklerine bakalım.

Model:

Tek katmandan oluşabileceği gibi birçok katmandan da oluşabilen Model yapısı projenin büyüklüğüne ve Yazılım geliştiricisinin planlamasına bağlıdır.

Bu katmanda veriler veri sağlayacılardan alınıp Controller ve Model katmanlarına aktarılır.

Örnek model yapısı aşağıda yer almaktadır.

```
1   public class MailViewModel
2       {
3           public int Id { get; set; }
4           public int? DepartmentId { get; set; }
5           [Required]
6           [Display(Name = "Message.To")]
7           public string ToList { get; set; }
8           public string Subject { get; set; }
9           [AllowHtml]
10          [Required]
11          public string Content { get; set; }
12          public bool isSMS { get; set; }
13          public string Attachments { get; set; }
14      }
15
```

View:

View, MVC pattern yapısına sahip projelerde, projenin Frontend(arayüz) bölümünü oluşturur. İstenilen yazılım dilinde geliştirilebilen bu katmanda dikkat edilmesi gereken konulardan biri de;

Süreç içerisinde çok fazla dosya oluşacağı için burada karmaşanın önüne geçmektir. Klasörleme yapısının doğru kurulması ve proje geliştirme süresinde bu yapının korunması yapılacak olan işlemleri daha da kolaylaştıracaktır.

Bu katmanın bir diğer görevi de ekran üzerinden alınan istekleri Controller katmanına aktarmaktır.

Controller:

Controller katmanı, Model ve View arasındaki iletişimi kurmak üzere görevlendirilmiştir. Projedeki iç süreçleri takip eder.

Kullanıcıların View üzerinden gerçekleştirdiği işlemlerle alınan veriyi Model'e taşır, Model'den aldığı veriyi View üzerinden kullanıcıya gösterir.

MVC Pattern in şeması aşağıdaki gibidir.