Spring Framework Kullandığı Design Patternler

1)Singleton pattern :Çalışma zamanında yalnızca 1 obje yaratılmasını garanti eden tasarım desenidir. Birden çok sınıf aynı nesneyi kullanması gerekiyorsa ya da sadece bir nesne olması gerekiyorsa singleton tasarım deseni kullanılarak durum sağlanıyor. Örneğin veri tabanı için bir bağlantı nesnesi sadece bir defa oluşturulmalı ve kullanılmalıdır. Sınıf yapıcıları(constructor) private olması gerekiyor. Yapıcıları private olan bir sınıftan,başka bir sınıf new operatörü ile nesne oluşturamaz. Singleton sınıfından sadece bir tane nesne oluşturulması gerektiği için oluşturulması gereken nesneyi sınıfın static değiskeni olarak tanımlamamız gerekiyor.

private static Singleton instance = null;

Singleton sınıfında instance() isminde static bir metodun olması ve bu metodun static olarak tanımlanmış nesneyi geri vermesi gerekiyor. instance() metodu içinde sınıfın tek nesnesi olacak değişken oluşturulur.

```
  public static Singleton getInstance() {

        if(instance == null){
            instance = new Singleton();
        }

        return instance;
    }
}
```

Programda getInstance metodu yardımıyla aynı instance kullanmış oluyoruz.

2)Factory Method: Aynı interface'i veya abstract sınıfı implement etmiş factory nesnelerinin üretiminden sorumlu pattern dir. Yani oluşturulan soyut sınıftan ya da arayüzden yeni bir sınıf oluşturma işlemi için kullanılan tasarım desenidir. Kullanıcı kalıp seçecek ancak seçilen kalıbın şekli kare mi yoksa yuvarlak mı olacağını bilmiyoruz ve bu değişkenlik gösterebilir. Kullanıcının seçimine bırakılan bu işlemde kullanıcının her seçiminde kod değişikliği yapmamak için Factory Pattern'i kullanabiliriz

```
public interface  Mold {
    public abstract String getType();
}
```

Kalıbımızın tipi kare ya da yuvarlak. Bu tipleri oluşturabilecek sınıflarımızı kalıp arayüzünü implement edecek şekilde tasarlıyoruz.

```
public class SquareMold implements Mold{

    public String getType() {
        return "SquareMold.";
    }
}

public class CircularMold implements Mold {

    public String getType() {
        return "CircularMold.";
    }
}
```

Seçilen kalıp tipine göre oluşturulacak olan nesneyi factory pattern yardımıyla oluşturuyoruz.

```
public class MoldFactory {

    public static Mold getMold(String type){

        if("Squaremold".equalsIgnoreCase(type)) return new SquareMold();
        else if("CircularMold".equalsIgnoreCase(type)) return new CircularMold();

        return null;
    }
}
```

```
public class MoldFactory {

    public static Mold getMold(String type){

        if("Squaremold".equalsIgnoreCase(type)) return new SquareMold();
        else if("CircularMold".equalsIgnoreCase(type)) return new CircularMold();

        return null;
    }
}
```

Son olarak da main sınıfımızda istenilen kalıp türüne göre MoldFactory sınıfına göndererek nesnenin oluşması sağlandı.

3)Proxy pattern :Spring Framework'ün en çok faydalandığı tasarım kalıplarından biridir. Nesneyle istemci arasına yeni bir katman koyarak nesnenin kontrollü bir şekilde paylaşılması sağlanır. Böylece istemci, işlem yapan sınıfla doğrudan temasa geçmemiş olur. Bu durum sayesinde işlemin yapılma performansında bir düşüklük olmaması sağlanır. Bu yüzden vekil kalıp fazla yük getiren işlemlerde kullanılır.Örnek olarak kredi kartı,banka hesabımızdakiler için bir vekildir. Nakit yerine kullanılabilir ve gerektiğinde bu nakde erişim imkanı sağlar. Bu tam olarak Proxy modelinin yaptığı şeydir.Korudukları nesneye erişimi kontrol eder ve yönetir. Başka bir örnekte internete erişim sağlarken bulunulan ağa göre belli sitelere erişimin kısıtlanması olabilir.

4)Template pattern: Soyut bir üst sınıf tanımlayarak alt sınıflarına belli metodları belirleyerek genel bir davranış şekli çizen tasarım desenidir. Davranışsal tasarım kalıbıdır. Mesela bir sistemde farklı türde veritabanları kullanıldığını temelde hepsinin yaptığı işlemlerin benzer olduğunu düşünelim. Bu durumda her bir veritabanı sınıfı işlemin yürütmesini kendi içinde tanımlayarak tasarım yapılabilir. Böylece sistemde genişletme yapmak istediğimizde tanımlanan ana soyut sınıfı kullanan yeni veritabanı sınıfları eklenebilir ve ayrı ayrı diğer sınıflarda değişiklik yapılmasına gerek kalmaz.Kod tekrarlarından da kurtulmamızı sağlar.