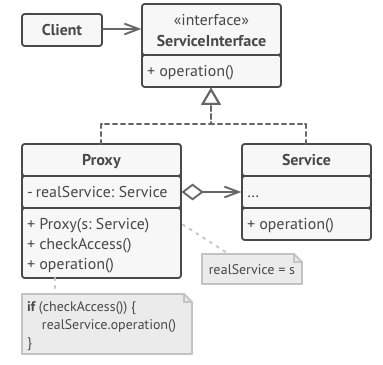
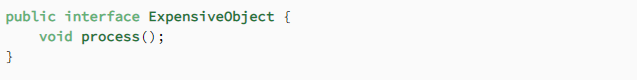
**Spring Framework’ de Kullanılan Design Patternler**

**Proxy Design Pattern**

**Proxy Factory Bean**

Yüksek oranda sistem kaynağı tüketen ve arada sırada ihtiyaç duyduğumuz nesneleri her zaman bellekte tutmak istemeyiz, istek dahilinde bu nesneyi oluşturup işlem tamamlandığında da nesneyi bellek dışı bırakmak isteriz. Bu problemi Proxy Design Pattern ile çözebiliriz. Aşağıdaki diyagramda da görüldüğü gibi bir interface oluşturup, bu interface’ i implemente edecek Proxy ve Service sınıflarımızı oluştururuz. Bu sınıflar aynı interface’ i implemente ettiğinden ortak methodlara sahip olurlar. Proxy ve Service, türleri aynı olduğundan Proxy’ deki bir method içinden Service’ deki method çağrılır. Proxy’ deki methoda Client erişir ancak Service’ e direkt erişemez. Proxy aracılığı ile erişmiş olur. Proxy’ e bu özelliğinden dolayı erişimi kontrol eden sınıf denir. Service’ den nesne oluşturma işlemi Proxy sınıfında yapılır.





metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

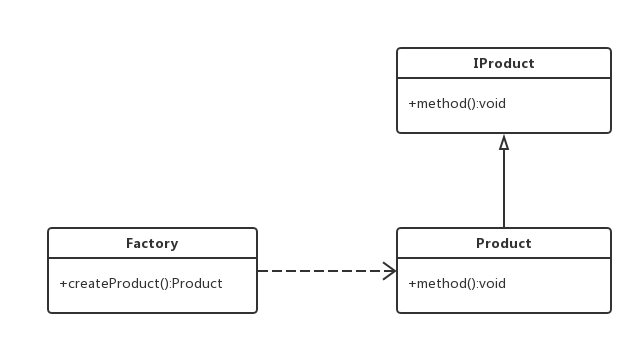
metin içeren bir resim

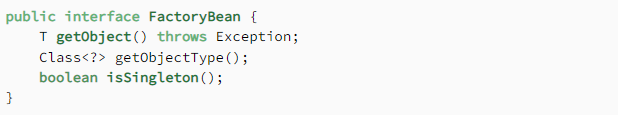
Açıklama otomatik olarak oluşturuldu

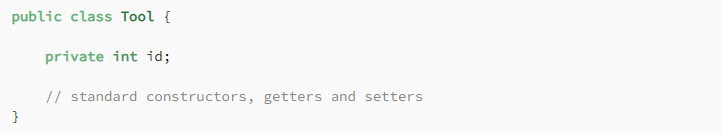
**Factory Design Pattern**

**Spring BeanFactory Container**

Factory Design Pattern, Spring içine entegre edilmiş, FactorBean anotasyonu ile kullanılabilen design patterndir. Aşağıdaki diyagramda da görüldüğü gibi, bir nesneyi, Factory aracılığı ile oluştururuz yani bir nesneyi oluşturma görevi artık Factory’ e aittir. Factory Pattern, kalıtımsal ilişkileri olan nesnelerin üretilmesi amacı ile kullanılır.





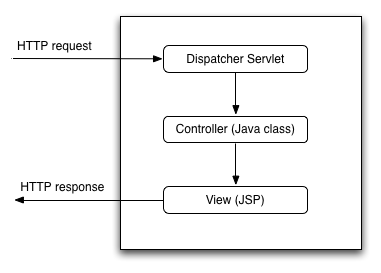


metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Model View Contoller Pattern**

Spring Model View Controller, Spring framework’ ün desteklediği bir modül ve bir design pattern’ dir. Bu patternde, aşağıdaki diyagramda görüldüğü gibi üç katman bulunur. Spring, bu katmanların birinde, DispatcherServlet kullanarak aslında front controller pattern’ i de kullanmış olur. Gereken nesne tanımlamalarının yapıldığı katman Model katmanı, bu nesnelerin arayüz olarak kullanıcıya sunulduğu yer View katmanı ve isteklerin yapılıp modelde güncellemelerin yapıldığı katman da Controller katmanıdır.



**Controller Class**

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Model Class**

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

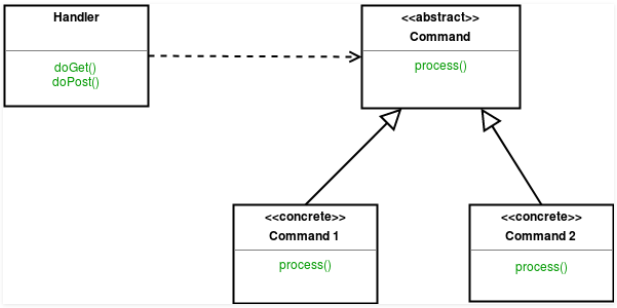
**View Pages**

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Front Controller Pattern**

Spring framework’ de Dispatcher Servlet olarak adlandırılır. MVC’ de View katmanında kullanılır. Front Controller’ da amaç, aşağıdaki diyagramda görüldüğü gibi istekleri merkezi bir yerde karşılayıp, ilgili yere yönlendirmektir. Kullanıcıdan gelen her istek farklı yetkiler gerektirebilir. Her isteği kontrol etmektense, gelen istekleri merkez noktada toplayıp sonrasında ilgili yere yönlendirmek kodumuzda tekrarlamanın önüne geçer. Gelen bütün istekleri karşılayan sınıf FrontController’ dır. Sonrasında aşağıdaki Front Controller Pattern’ i basitçe anlatan programdaki gibi, istekler kategorilerine göre FrontController’ da dağılır.



metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin içeren bir resim

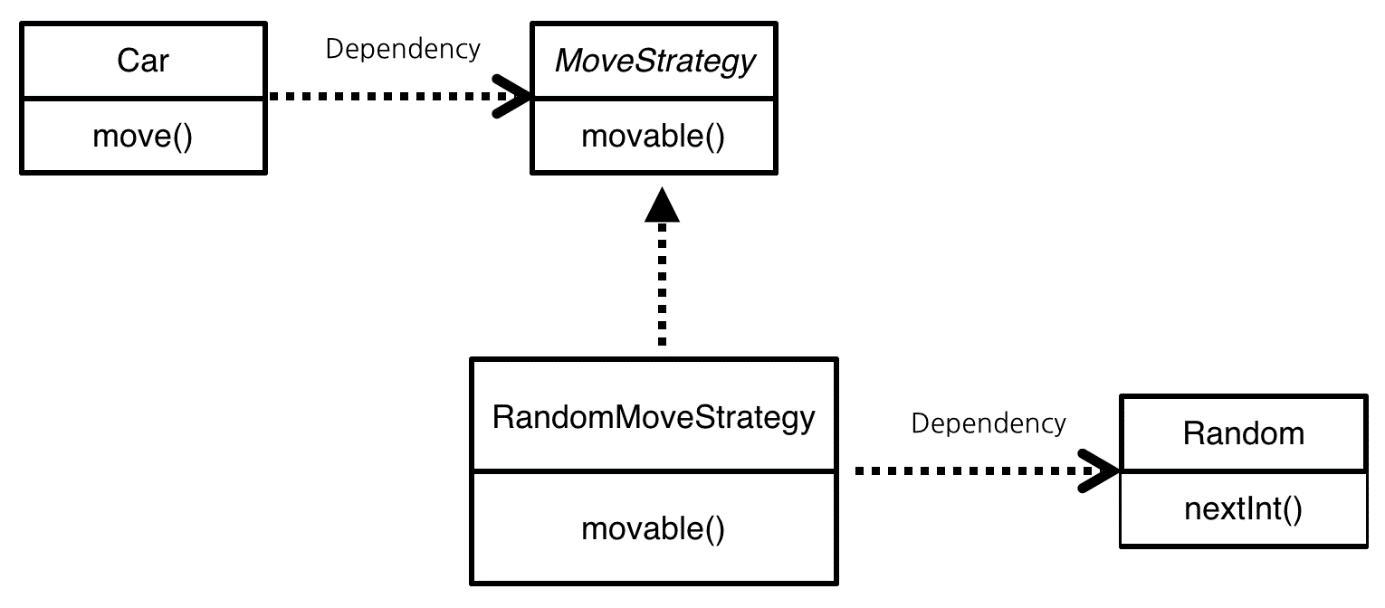
Açıklama otomatik olarak oluşturuldu

metin, iç mekan, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Dependency Injection**

Dependency Injection, bağımlılıkların kontrolü için kullanılır. Inversion of Control’ ün bir uygulama methodudur. Bir sistemi tasarlarken ve bu sistemi yazılıma döktüğümüzde, kullandığımız bir class da başka bir class a mutlaka ihtiyaç duyarız. Buna bağımlılık denir. Aşağıdaki diyagramda bağımlılıklar basitçe açıklanmış.



metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Bağımlı olunan classı, içinde kullanmak istediğimiz classa enjekte ederiz.

Dependency Injection’ da başka bir classda kullanmak istediğimiz başka bir classa ait nesneyi, new ile oluşturmak yerine constructor yardımı ile ya da getter setter methodları ile parametre olarak almamız gerektiği anlatılır. Böylece iki sınıfı birbirinden izole edeceğimiz ve düşük seviyede bir bağımlılık elde edeceğimiz savunulur.

Yukarıda görülen kod parçasındaki gibi, Controller classında Data Access Object nesnesine ihtiyaç duyulmuş ve bu class dan bir bir obje oluşturup, UsersController’ a verilmiş.

**KAYNAKLAR**

1. <https://www.baeldung.com/>
2. <https://refactoring.guru/>
3. <https://www.javaguides.net/>
4. <https://www.tutorialspoint.com/index.htm>
5. <http://www.javaturk.org/>
6. <https://www.geeksforgeeks.org/>
7. <https://www.journaldev.com/>
8. <https://www.injavawetrust.com/>

**Not**: Çok fazla siteden okudum bu yüzden tüm sitelere yer veremedim. Ancak en çok baktığım, kodlarından yararlandığım siteler bunlar oldu.

Şeyma Tezcan