Creational Patterns

1. 1.4. Builder Pattern

Builder, karmaşık nesnelerin adım adım oluşturulmasını sağlayan bir tasarım modelidir. Tasarım deseni, aynı builder kodunu kullanarak bir nesnenin farklı türlerinin ve temsillerinin üretilmesini sağlar.

1.1. Problem

İç içe nesneli, birçok alana sahip bir nesnenin oluşturulması senaryosunu düşünelim. Bu tür nesnelerin oluşturulması zahmetli ve adım adım tamamlanan işlemler dizisidir. Bu da **çok fazla parametreye sahip kurucu metotlar** ve de **ana metotta birçok satırda tanımlanmaya çalışılan nesneler**e sahip bir düzeni ortaya çıkaracaktır.

Örnek olarak bir MMORPG oyununda karakterin oluşturulmasını etkileyen unsurları ele alalım. Karakteri tanımlayan birçok özelliğin yanında her bir karakter türüne ait özel birtakım özellikler olacaktır; ırka özel silahlar, zırhlar, alt-türler ve özel yetenekler gibi.

Tüm bu özellikleri kapsamak adına hepsini kurucu metotta belirtip nesneyi oluşturacağımızı düşünürsek, bir ırka ait özellik diğer ırkta mevcut olmayacağından, **kurucu metot içerisinde oldukça fazla null ataması** ile karşılaşılır. Bu durumda aşağıdaki örnekte de görüldüğü üzere parametrelerin çoğu kullanılmamakta ve kod okunurluğu oldukça düşmektedir.

```
class Pizza {
    Pizza(int size) { ... }
    Pizza(int size, boolean cheese) { ... }
    Pizza(int size, boolean cheese, boolean pepperoni) { ... }
    // ...
```

1.2. Çözüm

Builder deseni, nesne oluşturma kodunu kendi sınıfından ayıklamanızı ve builder adı verilen nesnelere taşımanızı önerir. Pattern, nesne oluşumunu bir dizi adımda düzenler (silahıOluştur, AltTürüOluştur vb..).

Bir nesne oluşturmak için, bu ilgili adımlardan oluşan kodları bir builder nesnesinde yürütürsünüz. Önemli olan, tüm adımları çağırmanıza gerek olmamasıdır. Yalnızca bir nesnenin belirli bir yapılandırmasını oluşturmak için gerekli olan adımları çağırabilirsiniz.

Bazı oluşum süreçlerinde **implementasyon farklılıkları oluşabilir** ya da gerekebilir. **Bu durumda** ilgili adımları takip ederek nesne oluşturan **birden fazla builder sınıfı oluşturulur**. Bu builder sınıfları ile farklı türdeki ilgili nesneler oluşturulur.

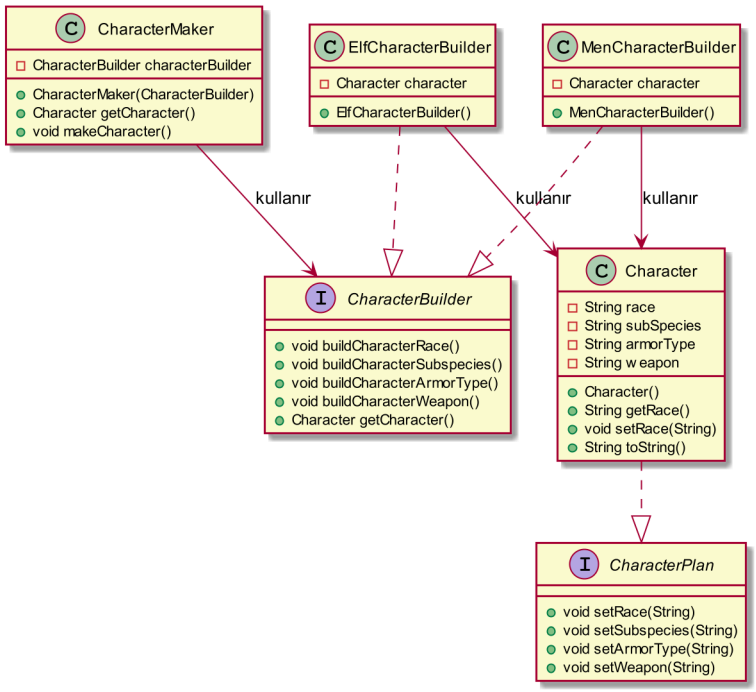

Figure 1. Builder Pattern UML Diyagramı

1.3. Örnek Üzerindeki Tasarım Deseni Yapısı

- **Character (Product):** Ürün sınıfı, builder pattern tarafından oluşturulacak karmaşık nesnenin türünü tanımlar. Verilen örnekte MMORPG karakter ırkı, product olarak düşünülebilir.
- **Builder:** Bu interface sınıfında, doğru karakterin oluşturulması için izlenmesi gereken tüm adımlar tanımlanır. Oyundaki temel karakterin oluşturulması için ilgili metotların şart olduğu gerekçesi ile sınıf interface olacak şekilde tasarlanır.

Oluşan karakteri döndürmek için getCharacter metodu kullanılır. Builder sınıfı bazı örneklerde abstract olarak da kullanılabilir.

- **ConcreteBuilder (ElfCharacterBuilder, MenCharacterBuilder):** Builder'dan miras kalan herhangi bir sayıda beton ConcreteBuilder sınıfı olabilir. Bu sınıflar belirli bir karmaşık ürünü oluşturma işlevini içerir. Örnekte belirtildiği gibi oyunda birçok farklı ırk bulunmakta ve her ırka özel birçok özellik mevcut durumdadır.
- **Engineer (Director, Maker):** Maker sınıfı, nihai ürün nesnesini oluşturan algoritmayı kontrol eder. Bir maker nesnesi oluşturulduğunda yapıcı metotta, ürünü oluşturmak için kullanılacak belirli ConcreteBuilder nesnesini yakalamak amacıyla bir parametre içerir. Daha sonra maker, ürün nesnesini oluşturmak için ConcreteBuilder metotlarını ilgili sırayla çağırır. İşlem tamamlandığında, ürünü döndürmek için maker nesnesinin getCharacter metodu kullanılır.

TestCharacterBuilder.java.

```
        CharacterBuilder elvenCharacter = new ElfCharacterBuilder();

  CharacterMaker characterMaker = new CharacterMaker(elvenCharacter);
  characterMaker.makeCharacter();

  Character firstElf = characterMaker.getCharacter();

  System.out.println("Character Created!");
  System.out.println(firstElf);
```

```
Character Created!
Character [race=Elves, subSpecies=Aen Elle, armorType=Light Armor,
 weapon=Staff]
```

Engineer (Maker) sınıfının çalışma mantığında belirtildiği üzere, ilgili concreateBuilder nesneleri oluşturulup, üretilen CharacterMaker sınıfına parametre olarak verilir. Bu parametre ile de hangi CharacterBuilder sınıfından ilgili nesnenin üretileceği belirlenmiş olur.