Creational Patterns

1. 2.1. Iterator Pattern

- Iterator pattern, farklı nesne koleksiyonlarına erişilmesi hususunda tek tip bir yol sağlamaktadır.
- Nesnelerden oluşan Array, ArrayList, Hashtable yapılarına sahip olduğumuzu düşünelim. Iterator pattern ile bu yapıların her birine aynı şekilde erişip yapı içerisinde dolaşılabilir. Bu da farklı koleksiyonlar arasında geçiş yapmak için tek tip bir yol sağlamaktadır.
- Iterator ayrıca client'in, ilgili veri setlerinin yapısını bilmesine gerek kalmadan dizi içerisinde kolayca dolaşabilmesine imkan sağlar.
    - Aynı zamanda veri setinin yapısı, client'tan gizlenmiş olur.

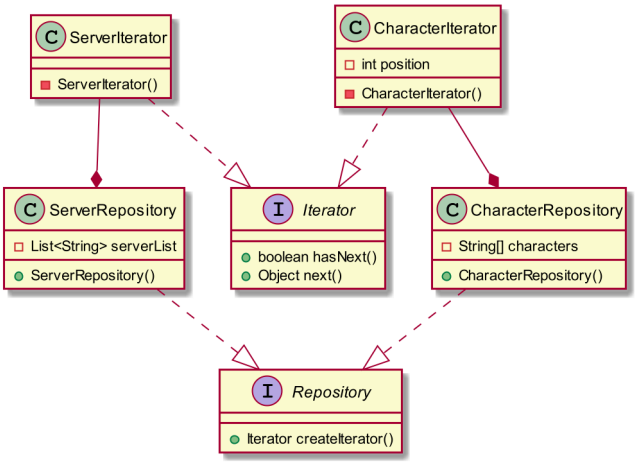

Figure 1. Iterator Pattern UML Diyagramı

- **Iterator:** Iterator yapısını tanımlayan arayüz. **hasNext()** ve **next()** metotlarını barındırır.
- **ConcreateIterators:** Iterator yapısını uygulayan, sınıfa (veri yapısına) özel Iteratorlerdir. Bu örnekte **ServerIterator** ve **CharacterIterator**, ConcreateIterator sınıflardır.
    - ServerRepository içerisinde List<String> yapısı mevcuttur. Bu yapıya özel Iterator implement edilmiştir.

- **Repository:** Repolarda iterator yapısının nasıl implement edileceğini belirler. createIterator() metodu ile Iterator oluşturulacaktır.
- **ConcreateRepositories:** Farklı veri tiplerini içeren repolardır. Bu örnekte **ServerRepository** ve **CharacterRepository**, Concreate repo sınıflardır.
    - İçerilerindeki **sub-class** iterator yapısı sebebiyle, eğer bir concreate repository mevcut değilse, bu repository için bir iterator olmayacaktır. Bu sebeple **composition** yapısı kullanılmıştır.