Nesneye Yönelik Yazılım Mühendisliği (376)

Yrd. Doç. Dr. Ahmet Arif AYDIN

Software Design Patterns

- Each pattern **describes a problem** that occurs over and over again in our environment, and then describes **the core of the solution** to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice (**Christopher Alexander 1970)**





Software Design Patterns

- Tasarım Kalıplarını (Desen) neden kullanmalıyız?
  - *Bir yazılım probleminin kabul görmüş ve yazılım geliştirmede yaygın olarak kullanılan çözüm yoludur.*
  - ***general repeatable solution*** to a commonly occurring problem in software design
  - solutions to general problems that software developers faced during software development
  - well described solution to a common software problem

Why use Software Design Patterns ?

- Nesne tabanlı programlama ve Tasarım kalıpları

  - Genel geçerliliği olan olan en iyi çözümleri kendi yazılım problemlerimize uygulamamızı imkan sağlar (*capture best practices*)
  - Değişim istekleri daha kolay bir biçimde cevaplanır (flexibility)
  - Yazılımın bakımı uzun süreli ve sürdürülebilir (maintainable) olur
  - tekrar kullanılabilen (re-usable) kod oluşturmaya imkan saglar
  - Geliştiriciler arasında yazılım anlaşılmasını kolaylaştırır (Shared vocabulary)

Categories of Design Patterns

- **Creational patterns** (Oluşturucu)
  - Nesnelerin ve örneklerinin(instance) oluşturulması ile alakalı kalıplar
  - instantiation process of objects
  - create objects while **hiding the creation logic**, rather than instantiating objects directly using new operator.
- **Structural patterns** (Yapısal)
  - Sınıfların ve neslerin oluşturulması ile alakalıdır
  - **inheritance** is used to compose interfaces or implementations
  - How classes and objects are composed to form larger structures
- **Behavior patterns** (Davranışsal)
  - Sınıfların nesneleri arasındaki etkileşim ve sorumluluklar ile alakalı kalıplar
  - patterns that are most specifically concerned with communication between objects

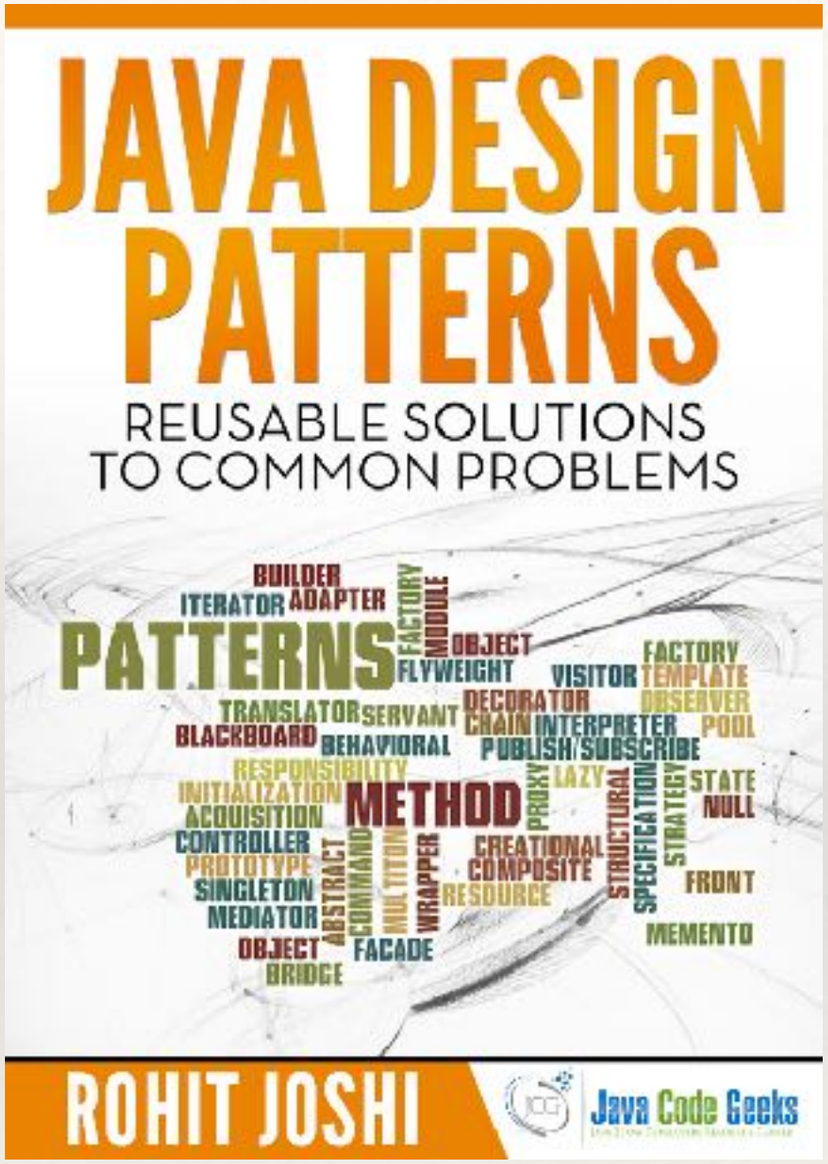
Software Design Patterns

- Creational patterns
- Structural patterns
- Behavior patterns

https://sourcemaking.com/design_patterns

http://www.oodesign.com/

http://www.blackwasp.co.uk/gofpatterns.aspx



Creational Design Patterns

Abstract Factory	Bir birleriyle ilişkili sınıfların oluşturulmasını düzenler
Builder	Bir nesnenin örneğinin (instance) alt sınıfa bulunan farklı özelliklerini kullanılarak oluşturulmasında kullanılır (*Separates object construction from its representation*)
Factory Method	Aynı arayüzü (interface) kullanan neslerin oluşturulması ve yönetimini sağlar
Prototype	Var olan nesnelerin kopyalarının oluşturulmasında kullanılır
Singleton	Uygulamanın yaşam süresince bir sınıfın global olan bir nesnenin bir kez oluşturulmasını sağlar

http://harunozer.com/makale/tasarim_desenleri__design_patterns.htm

Structural Design Patterns

Adapter	Sınıflar arasındaki uyumu sağlamak için kullanılır.
Bridge	İnterface ile implementation birbirinden bağımsız olarak işlem görmesini sağlar
Composite	Nesnelerin ağaç yapısı biçiminde kullanılarak parça bütün ilişkisini tanımlamayı sağlar
Decorator	Bir nesneye dinamik olarak yeni sınıf ve davranış eklemeyerek extend işlemini gerçekleştirir.
Facade	Bir sistemde yüksek seviyeli bir arayüz oluşturup alt arayüzlerin tek bir arayüz altında gösterimini sağlar
Flyweight	Sık kullanılan nesnelerin yönetiminde kullanılır
Proxy	Bir nesne yerine başka bir nesnenin kullanılabilmesi (vekil)

Behavioral Design Patterns

Chain of Responsibility	Bir isteğin birden fazla nesne tarafından sırayla kullanılmasını sağlar
Command	istegi bir nesne gibi kapsüllemeyi sağlar *(Encapsulate a request as an object, thereby letting you parametrize clients with different requests, queue or log requests)*
Interpreter	*define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language*
Iterator	*Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation*
Mediator	*The Mediator defines an object that controls how a set of objects interact*
Memento	The Memento captures and externalizes an object's internal state so that the object can later be restored to that state
Observer	a one-to-many relationship so that when one object changes state, the others are notified and updated automatically
State	Allow an object to alter its behavior when its internal state changes.
Strategy	A Strategy defines a set of algorithms that can be used interchangeably. (*Modes of transportation to an airport)*
Template Method	Tanımlanan aşamaları alt sınıflar için farklı tanımlamayı sağlar
Visitor	İstenilen yeni bir işlemi sınıfları değiştirmeden gerçekleştirmekte kullanılır