selsoft
build better, deliver faster

Tasarım Kalıpları (Design Patterns)

1. Bölüm - Tasarım Kalıplarına Giriş

1

Eğitmen:

Akın Kaldıroğlu

Çevik Yazılım Geliştirme ve Java Uzmanı

(Sıkıntı,) çözümü görememeleri değildir.
(Sıkıntı,) problemi görememeleridir.

It isn't that they can't see the solution.
It is that they can't see the problem.

G. K. Chesterton,

The Point of a Pin in The Scandal of Father Brown

Tanımlar

selsoft
build better, deliver faster

Pattern Nedir?

- *pat·tern noun \ˈpa-tərn*
  - *: a repeated form or design especially that is used to decorate something*
  - *: the regular and repeated way in which something happens or is done*
  - *: something that happens in a regular and repeated way*
  - *…*
  - *: something designed or used as a model for making things <a dressmaker's pattern>*

Biraz Kelime Bilgisi

- Design Patterns => Tasarım Kalıpları
  - Design => Tasarım
  - Pattern => Kalıp, Şablon, Desen, (hatta Örüntü!)
- Tasarım, ifade edilen probleme kavramsal bir çözümün oluşturulduğu safhadır.
  - Tasarım kalıpları, geliştirmenin ya da kodlamanın değil tasarım çalışmasının bir parçasıdır.

Tasarım Kalıbı Nedir? I

- Christopher Alexander says, "**Each pattern describes a problem which 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“.**

- **“Her bir kalıp önce ortamımızda tekrar tekrar oluşan bir problemi, sonra da o probleme çekirdek bir çözümü tanımlar öyle ki siz bu çözümü, hiç bir iki kullanımınız birbirinin aynısı olmadan, bir milyon defa kullanırsınız.“**

Tasarım Kalıbı Nedir? II

- Tasarım Kalıplarının amacı, nesne-merkezli prensipleri kullanarak
  - doğru sorumlulukları bulmak (finding correct responsibilities),
  - değişimi göz önüne alarak bu sorumlulukları nesnelere dağıtmak (highly-cohesive objects)
  - nesneleri, aralarında az bağımlılık oluşturacak şekilde kurgulamaktır (lowly-coupled objects)
- Yüksek birliktelikli ve düşük bağımlılıklı yapıları nasıl kurgulayacağımızı, sıklıkla karşılaşılan problemler bağlamında, genel bir yapıda ve tekrar kullanılabilecek şekilde modeller.

Tekrarlanan Problemler

Tekrarlanan Problemler - I

- Nesneleri nasıl yaratırız?
  - Karmaşık nesneleri nasıl yaratırız?
  - Nesne ailelerini nasıl yaratırız?
- Bir sınıftan sadece bir ya da belirli sayıda nesne nasıl yaratırız?
- Nesnelere erişimi nasıl kontrol ederiz?
- Nesneler arasındaki bütün-parça ilişkisini nasıl tasarlarız?

Tekrarlanan Problemler - II

- Bir işi yapmanın pek çok yolu varsa bunları nasıl ifade ederiz?
- Emir-komuta ya da olay-bilgilendirme zincirini nasıl oluştururuz?
- Bir sürü nesne arasındaki haberleşmeyi nasıl yönetiriz?

Tekrarlanan Problemler - III

- Bir nesneye çalışma zamanında yetkinlik nasıl kazandırırız?
- Karmaşık duruma sahip olan nesneleri nasıl yönetiriz?
- Nesnenin durumunu nasıl saklar ve sonra yine nasıl ulaşırız?
- Birden fazla nesneyi nasıl yönetiriz?
- Aynı işi birden fazla nesneye nasıl uygularız?

Nesne Rolleri

- Bahsedilen bu problemlerin çözümlerinde, sık karşılaşılan sorumluluklar ile bunları yerine getiren nesne rolleri ortaya konur.
- Tasarım kalıpları ile elde edilen nesne rolleri, zihnimizde bir rol soyutlaması ve kategorizasyonu oluşturur.
- Bu roller, tasarım kalıpları kullanılmasa bile genel yazılım tasarımının bir parçası olurlar.

Kalıbın Yapısı

selsoft
build better, deliver faster

Tasarım Kalıplarının Yapısı

- Bir tasarım kalıbının temelde dört bileşeni vardır:
  - **İsmi** ki o kalıbı, problemi ve çözümüyle birlikte ayırt etmemizi sağlar,
  - **Problem** ki hangi bağlamda nasıl ortaya çıktığını ifade eder,
  - **Çözüm** ki parçaları ve aralarındaki ilişkileri ifade eder,
  - **Sonuçlar** ki kazanılan ve kaybedilenleri ifade eder.

Kalıbın Bileşenleri

Item	Description
Name	All patterns have a unique name that identifies them.
Intent	The purpose of the pattern.
Problem/Motivation	The problem that the pattern is trying to solve.
Solution/Structure	How the pattern provides a solution to the problem in the context in which it shows up.
Participants and collaborators	The entities involved in the pattern.
Consequences	The consequences of using the pattern. Investigates the forces at play in the pattern.
Implementation	How the pattern can be implemented. Note: Implementations are just concrete manifestations of the pattern and should not be construed as the pattern itself.
Generic Structure	A standard diagram that shows a typical structure for the pattern.
Applicability	What are the situations in which the design pattern can be applied?
Sample code	Code fragments that illustrate the pattern in a object-oriented language
Known uses	Examples of the pattern found in real systems
Related patterns	What design patterns are closely related to this one? What are the important differences?

Neden Tasarım Kalıpları?

selsoft

build better, deliver faster

Tasarım Kalıpları Bizi Korur

- Tasarım kalıpları çoğu defa bizi, normalde nesne-merkezli programlama dili kullanmamıza rağmen prosedürel anlayışla yazılım geliştirmeye düşmekten kaçınmamızı sağlar.
- Bu yüzden normalde tek bir metot altında pek çok “`if else if`” ile yaptığımız **functional decomposition** tarzındaki yapıları, nesne seviyesinde nasıl ifade edeceğimizi bize öğretir.
- Dıolayısıyla tasarım kalıpları gerçek nesne-merkezli programlama yapmamıza yardıumcı olur.

Farklı Tipte Kalıplar - I

- Tasarım kalıpları nesne seviyesindeki problemler içindir, iş alanından bağımsızdır.
- Mimari kalıplar (Architectural patterns)
  - Güvenlik kalıplar (Security patterns)
  - Performans kalıplar (Performance patterns)
  - vs.
- İş Alanı Kalıplar (Business domain patterns)

Farklı Tipte Kalıplar - II

- Genelde tasarım kalıbı denince, yazılım geliştirirken sıklıkla karşılaşılan problemlere, dilden ve iş alanından bağımsız, karmaşıklığı önleyecek ve değişimi kolaylaştıracak şekilde, tekrar kullanılabilir ve soyut çözümlerdir.
  - Dil seviyesindeki kalıplara ise “idiom” denir,
  - Mimari kalıplar ise tasarım kalıplarından beslenir.

Neden Tasarım Kalıpları?

- Tekerleği yeniden keşfetmemek, var olan ispatlanmış çözümleri kullanmak,
- Formal ve yaygın bir dil oluşturmak,
- Tasarıma uygun, yüksek soyutlama gücü kazandırmak, detaylardan sıyrılıp, daha yüksek hedefler cinsinden düşünmek.

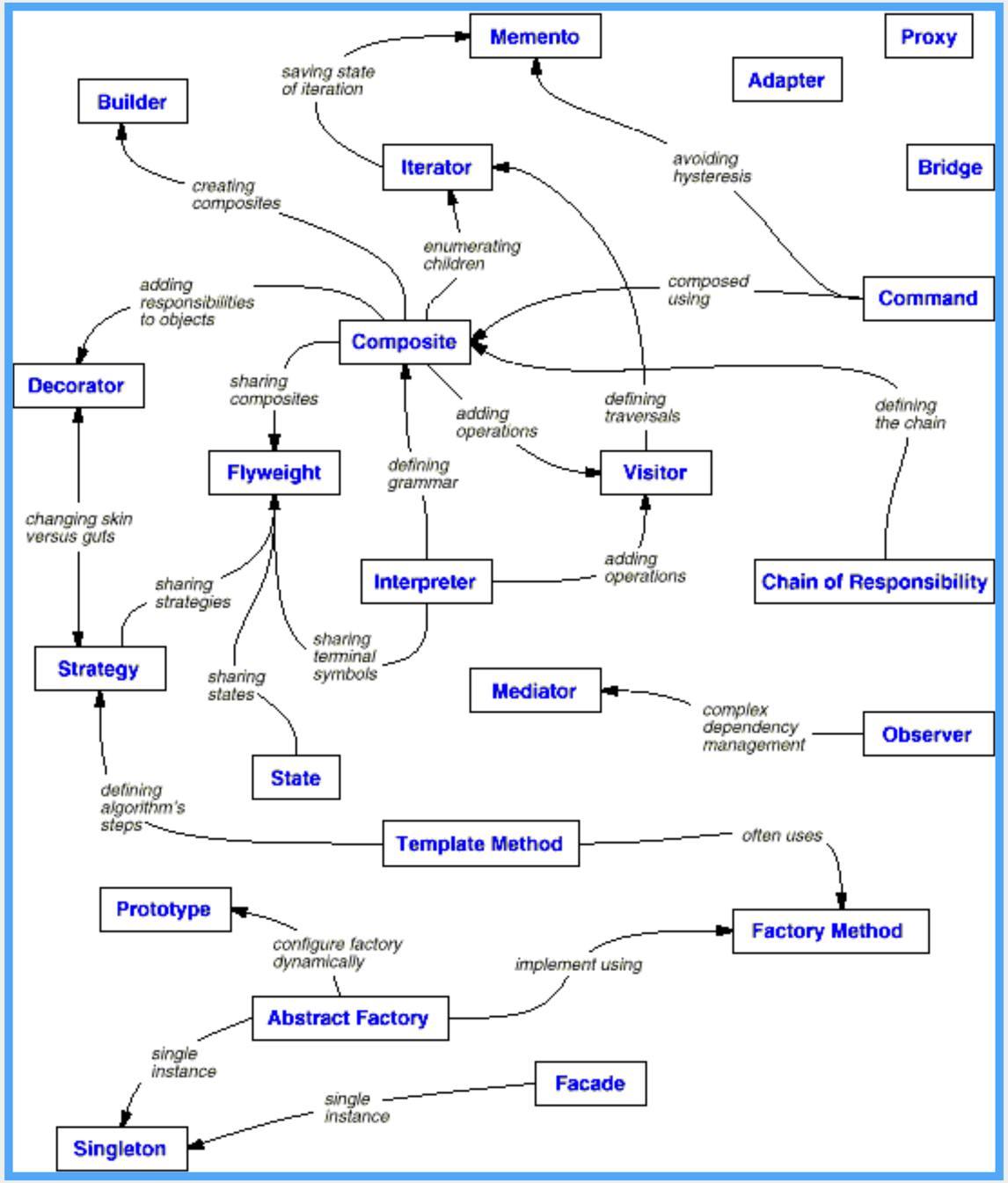
GoF

selsoft

build better, deliver faster

Dörtlü Çete

- Dörtlü çete (Gang of Four) kitapta 3 kategoride toplam 23 kalıba yer vermişlerdir:
  - Creational (yaratımsal)
  - Structural (yapısal)
  - Behavioral (davranışsal)
- Okuması tecrübeli olmayanlar için zordur,
- Örnekleri C++ ile verilmiştir,
- İlk iki bölümü nefis bir OO özetidir.


Memento
Proxy
saving state of iteration
Builder
Adapter
Iterator
avoiding hysteresis
Bridge
creating composites
enumerating children
adding responsibilities to objects
composed using
Command
Composite
Decorator
sharing composites
adding operations
defining traversals
defining the chain
Flyweight
defining grammar
Visitor
changing skin versus guts
sharing strategies
Interpreter
adding operations
Chain of Responsibility
sharing terminal symbols
Strategy
sharing states
Mediator
complex dependency management
Observer
State
defining algorithm's steps
Template Method
often uses
Prototype
configure factory dynamically
Factory Method
implement using
Abstract Factory
single instance
Facade
single instance
Singleton

Introduction

Designing object-oriented software is hard, and designing reusable object-oriented software is even harder. You must find pertinent objects, factor them into classes at the right granularity, define class interfaces and inheritance hierarchies, and establish key relationships among them. Your design should be specific to the problem at hand but also general enough to address future problems and requirements. You also want to avoid redesign, or at least minimize it. Experienced object-oriented designers will tell you that a reusable and flexible design is difficult if not impossible to get "right" the first time. Before a design is finished, they usually try to reuse it several times, modifying it each time.

How Design Patterns Solve Problems?

- How Design Patterns Solve Design Problems?
  - Finding appropriate objects,
  - Determining object granularity,
  - Specifying object interfaces,
  - Specifying object implementations
  - Determining relationships between objects

Üç Prensip

- GoF'un kitabının başında vurguladığı ve kalıplarına temel yaptığı üç prensip şunlardır:

Program to an interface, not an implementation.

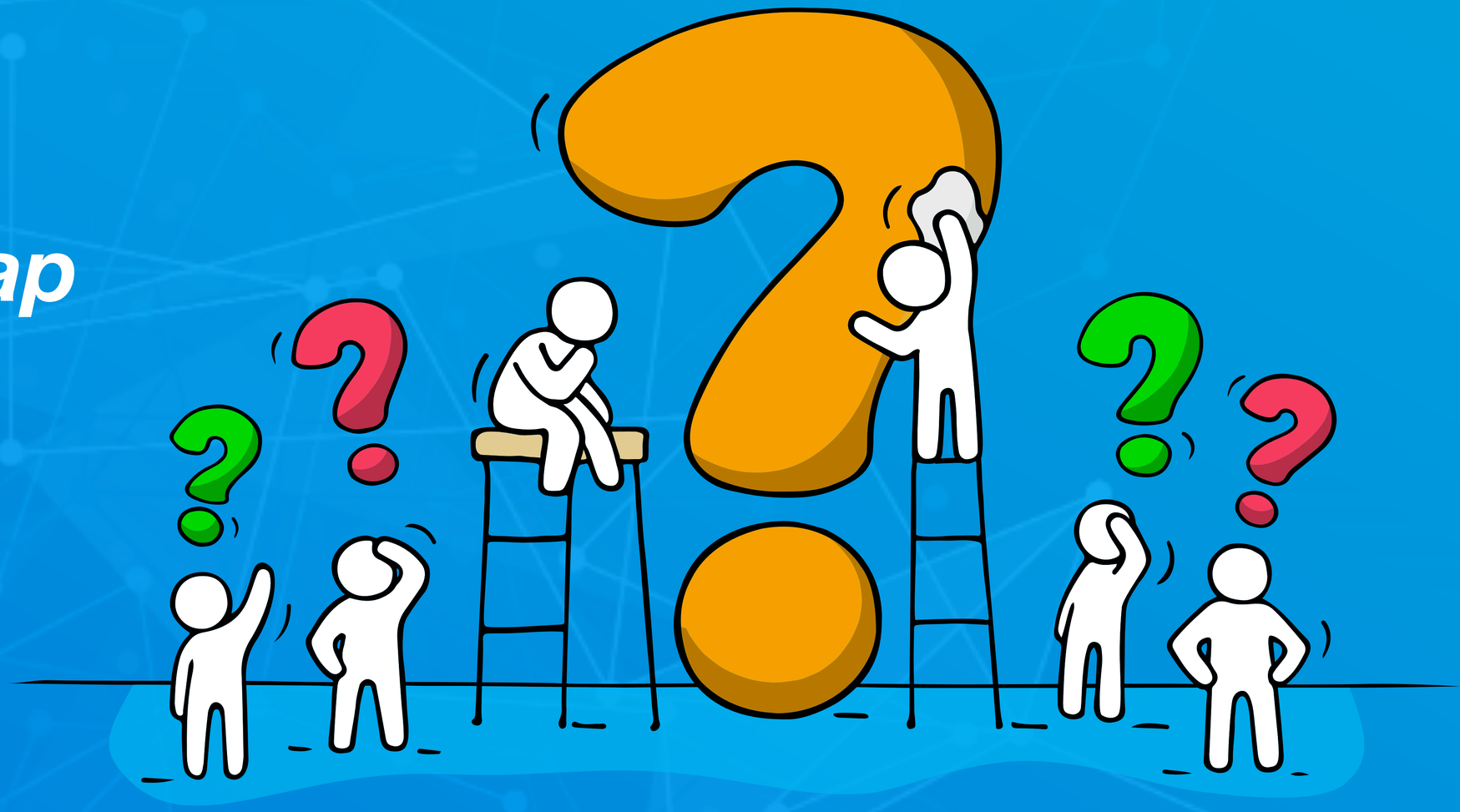
Favor object composition over class inheritance.

Design for change.

Bazı Noktalar

- Unutulmaması gereken şey, bu dörtlü çetenin kitaptaki kalıpları yoktan var etmediği gerçeğidir.
  - Discovered not invented!
- Bağlamınızı bu 23 kalıptan bir ya da birkaçıyla süslemek zorunda değilsiniz.
  - Sizler de size özel bağlamınızda kendinize özel kalıplar bulabilirsiniz, bulmalısınız.

Soru ve Cevap Zamanı!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

selsoft

build better, deliver faster

Uygulama

Örnek: Proxy Pattern

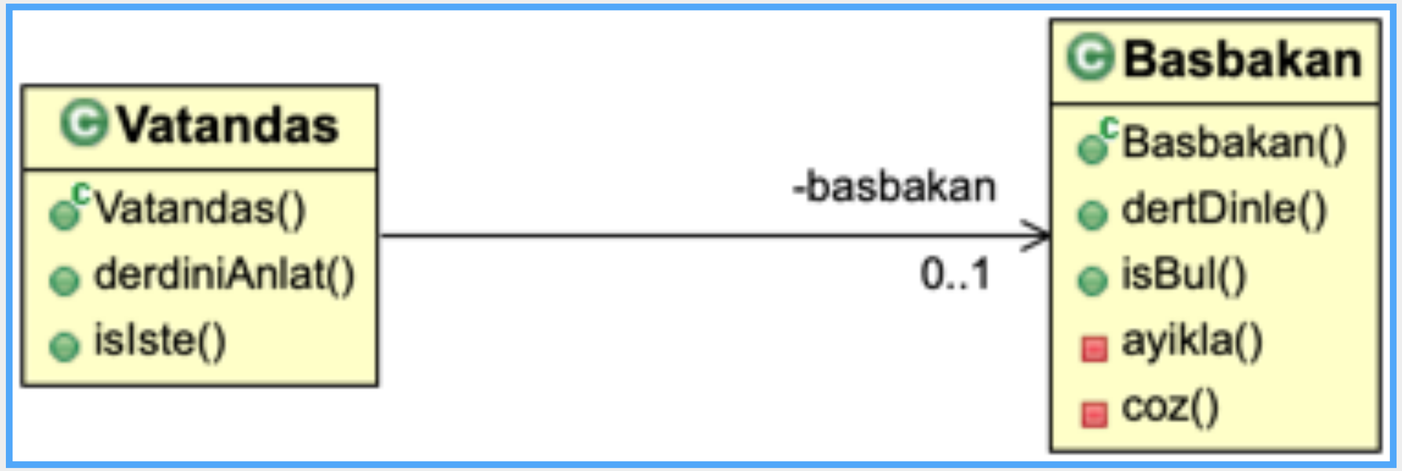
- Aşağıdaki problemi GoF'un Proxy (Vekil) tasarım kalıbı ile nasıl çözeceğimizi görelim:
  - Demokrasilerde vatandaşların kendilerini yönetenlere ulaşma hakları vardır,
  - Başbakan'ın da en tepe yönetici olarak vatandaşları dinleme zorunluluğu vardır,
  - Ama Başbakan'ın 75 milyon kişiyle görüşmesi hem imkansızdır hem de uygun değildir,
  - Bu durumda vatandaşların Başbakan'la görüşme haklarını nasıl yönetiriz?

Sorumlulukları Bulmak - I

- Sistemde bulunacak sorumluluklar:
  - **`Basbakan`**
    - dert dinlemek
    - iş bulmak
  - **`Vatandas`**
    - Başbakan'a derdini anlatmak
    - Başbakan'dan iş istemek

Sorumlulukları Bulmak - II

- Ayrıca **`Basbakan`**'ın şu sorumluluklarının olduğu da düşünülebilir:
  - istekleri ayıklamak
  - istekleri çözmek



Birliktelik-Bağımlılık-Değişim - I

- **`Vatandas`** nesnenin birlikteliği yüksektir çünkü fonksiyonel olarak odaklı iş yapmaktadır.
- Ama **`Başbakan`**'ın birlikteliği daha düşüktür çünkü fonksiyonel olarak farklı işleri bir araya getirmiştir.
  - Çözmek belki ama ayıklamak kesin olarak onun görevi değildir.

Birliktelik-Bağımlılık-Değişim - II

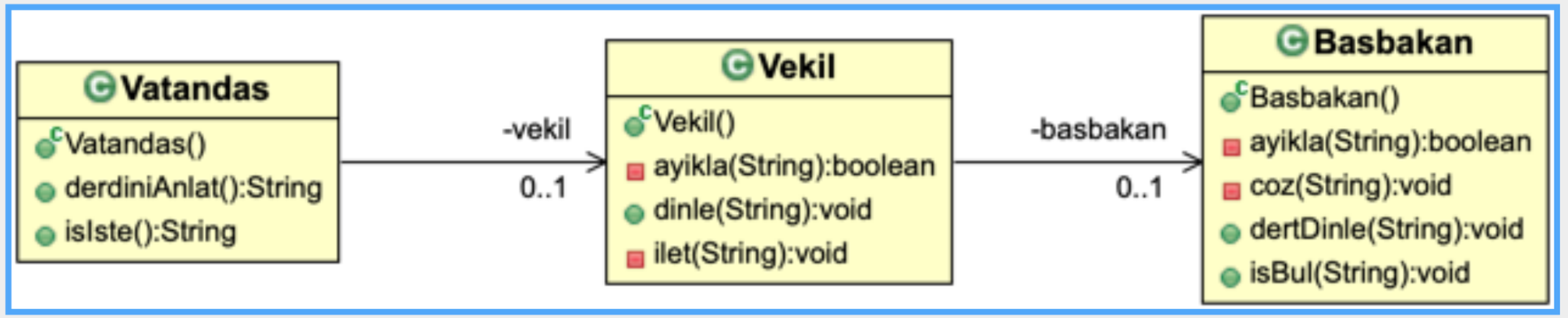
- **`Vatandas`** ve **`Basbakan`** arasındaki aralarındaki bağımlılık yüksektir.
- **`Vatandas`**'ın **`Basbakan`**'ı doğrudan bilmesi, çok sayıda nesnesinin olmasından dolayı hem pratik değil hem de güvenlik açısından problemlidir.

Birliktelik-Bağımlılık-Değişim - III

- En fazla değişecek yerler nereleridir?
  - **`Başbakan`** nesnesinin sık değiştiği durumlarda ciddi güncelleme problemleri oluşur.
  - **`Başbakan`**'ın dert dinleme, iş bulma, ayıklama vb. davranışları da değişme eğilimindedir.
    - Bir **`Başbakan`** nesnesi iş bulma isteklerini yok sayabilir.
- Bu değişimler **`Vatandas'`**ı nasıl etkiler?

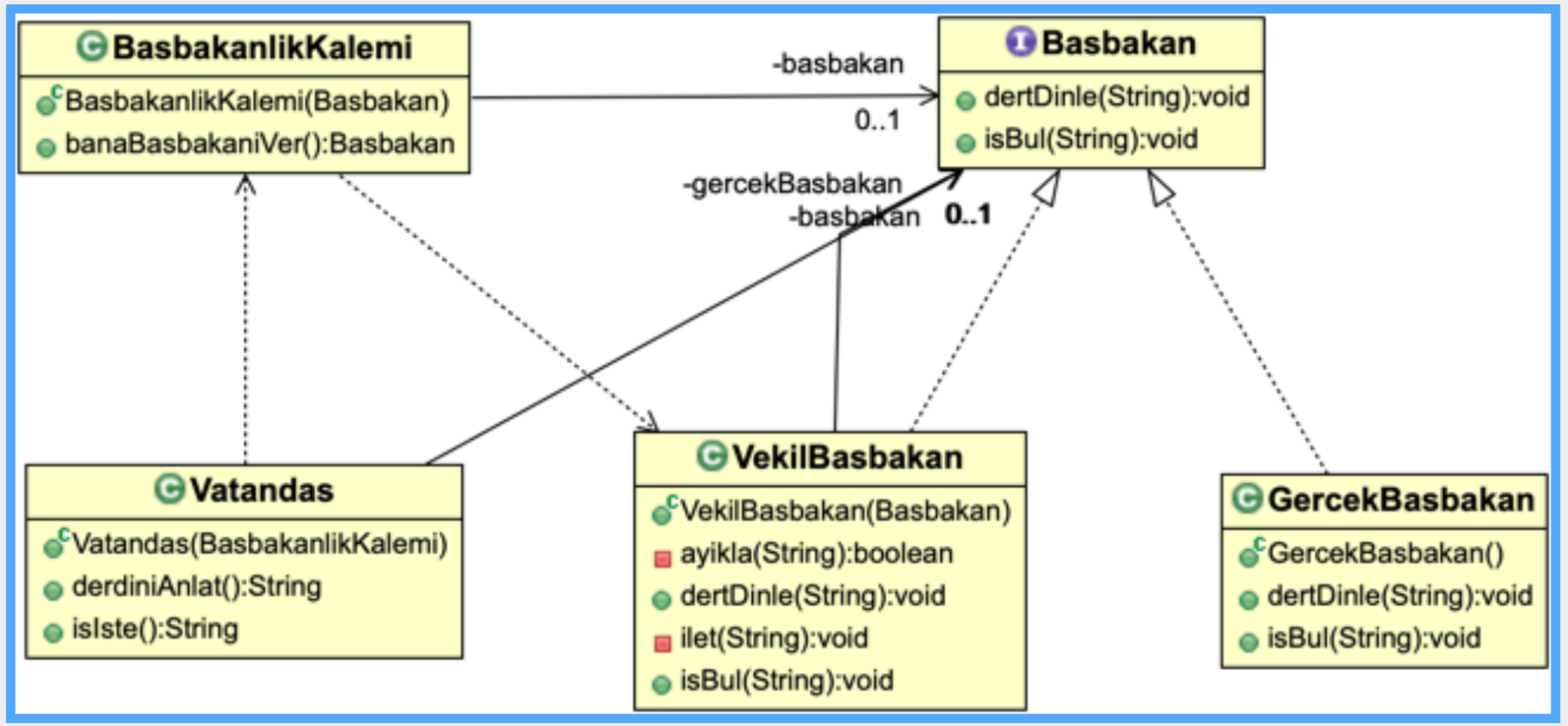
Çözüm - I

- Araya bir **`Vekil`** nesnesi koyup bu sıkıntılardan kurtulabiliriz.
- Fakat bu durumda da demokrasinin en temel prensibini ihlal etmiş oluruz:
  - **`Vatandas`**, **`Basbakan`**'a doğru ulaşamamaktadır.

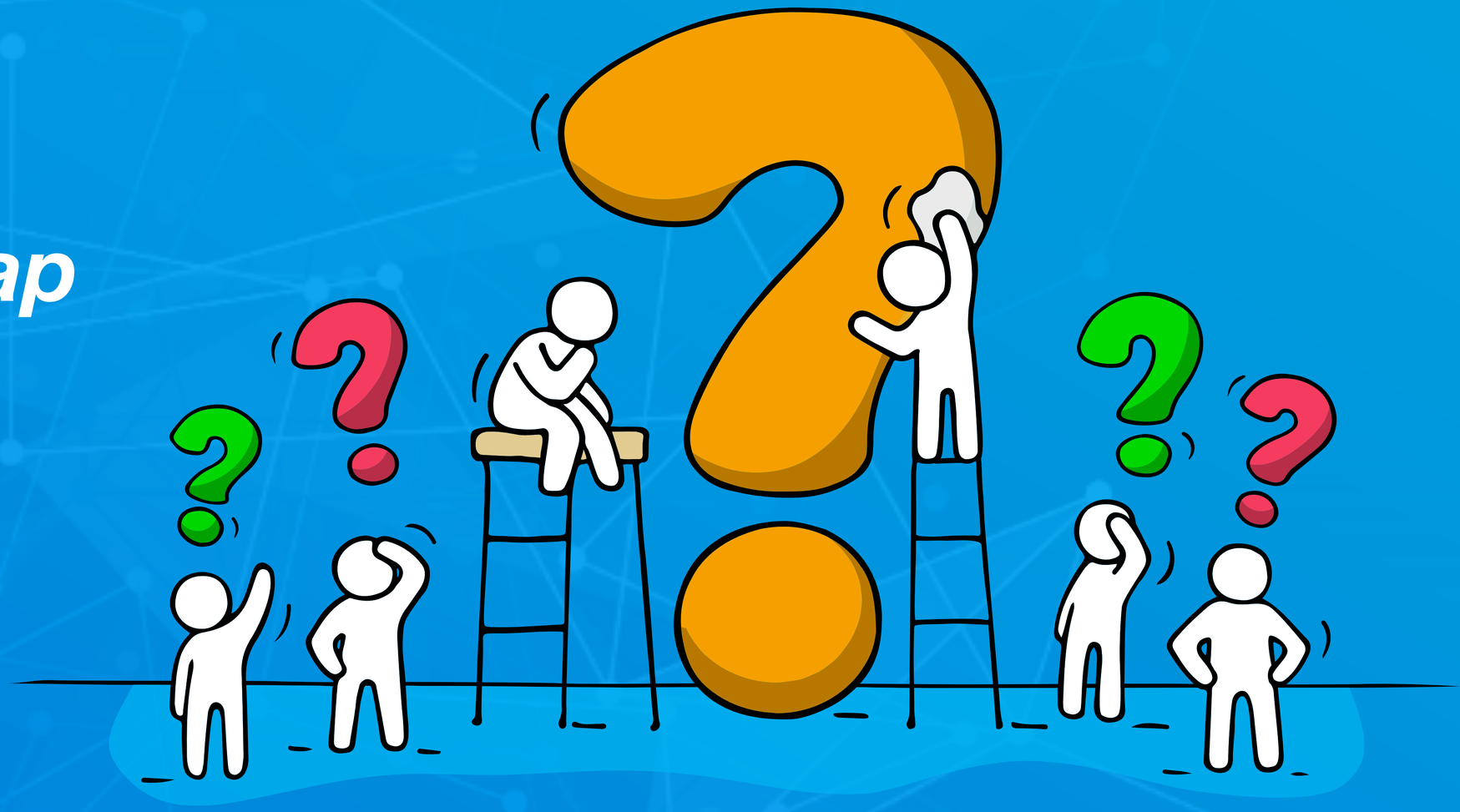


Çözüm - II

- En sık değişen kısımları soyutlayıp, **`Vatandas`**'ın gerçek **`Basbakan`**'la bağımlılığını soyut hale getirdiğimizde ortaya şu model çıkar:



Soru ve Cevap Zamanı!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

selsoft

build better, deliver faster

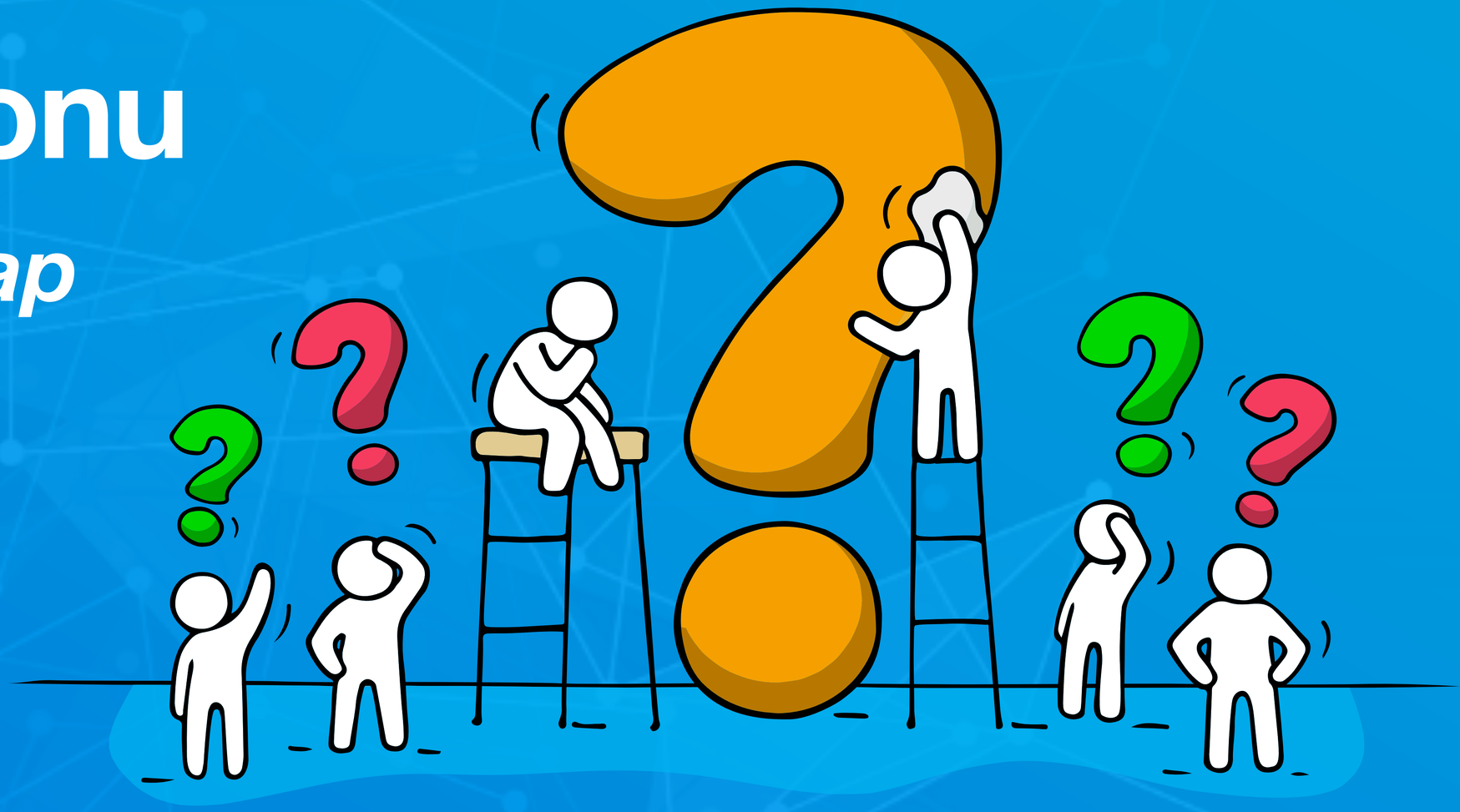
Ödev

Ödev - I

1. İlk dersin ödevi, sadece bu bölümü tekrar etmek ve **Proxy** kalıbını anlamaktır.

Bölüm Sonu

Soru ve Cevap Zamanı!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr