

# **Java & Spring Bootcamp**

**Mustafa Güneş**

# Hakkımda

- Yıldız Teknik Üniversitesi, Bilgisayar Mühendisliği (2017)
- Yapı Kredi Bankası (Teknik Mimari)
- Scotty
- Craftbase (Trendyol Go)
- Midas
- <https://twitter.com/mgunes17>
- <https://www.linkedin.com/in/mgunes17/>

# Takvim

- Başlangıç 18-19 Aralık 2021
  - 1-2 Ocak ders yok
  - Bitiş 5-6 Şubat 2022
  - 10.00 - 11.00
  - 11.00 - 11.15 mola
  - 11.15 - 12.15
  - 12.15 - 13.00 mola
  - 13.00 - 14.00
  - 14.00 - 14.10 mola
  - 14.10 - 15.00
- \* (Küçük sapmalar olasılık dahilindedir.)

# Kullanılacak Araçlar

- IntelliJ Idea 2021.3
  - Öğrenci maili / 30 günlük deneme sürümü / Community Edition
  - Eclipse, Netbeans, ...
  - Lombok ve Sonarlint eklentileri kurulması faydalı olur.
- openjdk 17.0.1 2021-10-19
  - 8+ büyük oranda yeterli olacaktır.
- Apache Maven 3.8.3
- Docker version 20.10.8

# Eğitim Hakkında - 1

- Saçma soru yoktur, soru sormamak saçmadır.
- Anlatımlar ağırlık olarak kod üzerinden olacak.
- Kod yazmak ifadesi her zaman test yazmayı da kapsar.
- Güncel teknolojiler, pragmatik kod örnekleri ile anlatılacak. Amacımız problem çözmek.
- Bir konuya başlayınca baştan sona bitirilmeyecek. Gerektiği yerde gerektiği kadar öğreneceğiz. (Öğrenmek = neyi nerede nasıl neden kullanıyoruz)
- Kitaplardan özetler incelenecak (Ağırlıklı olarak Effective Java ve Clean Code). Çalışan kod yazmak ilk amacımız olsa da fırsatlarımız dahilinde kodu temiz ve verimli bir hale getireceğiz.
- Teorik kavamlar araştırma ödevi olarak verilecek, derslerde tartışılacak.
- Geride kaldığınızı, koptığınızı hissettiğiniz zamanlarda bana ulaşın. Buradaki herkes birebir aynı noktada değil, bitirirken de birebir aynı noktada olmayacak. Kendiniz için maksimum verimi almaya odaklın.

# Eğitim Hakkında - 2

- Eğitim sonunda ayrıca proje isteniyor olsa da aralarda da kodlama ödevleri olacak.
- Sunumlar derse yön vermek için var. Ayrıca kodlama detayına giremeyeceğimiz ama bilinmesi faydalı konular sunumlardan anlatılacak.
- Soru sormak için beklemeyin, araya girin sorun veya yazın. Sorular teknik bir konu hakkında olabileceği gibi sektör hakkında da olabilir.
- Bu bootcamp formal/akademik/sistematik bir eğitim değil. Bir grup Yazılım Mühendisinin potansiyel çalışma arkadaşlarıyla birlikte geliştiği bir süreç olacak.
- Interaktif eğitimin avantajlarından faydalanalım. Konuşalım, tartışalım, fikir yürütelim. Video eğitimlerin, kitapların, google bilgilerinin ötesine geçmek için çaba harcayalım.
- Not almanız tavsiye olunur. Not alma alışkanlığınız yoksa başlamak için güzel bir fırsat.
- “No silver bullet” “Sihirli degnek yok”. Bu bootcamp veya diğer kaynaklar size yol gösterebilir, yeni yollar açabilir. Ama o yolda sizi yürütemez. Bireysel olarak da çaba harcamalı; araştırmalı, okumalı, kodlamalısınız.

# İçerik (Güncellenecek)

- **1. Hafta, Temel Konular;** Java, Object-Oriented Concepts, Veri Yapıları, Maven, Lombok, Java Streams, Lambda Functions
- **2. Hafta, Spring Framework;** Dependency Injection, Spring Core, Spring Boot, Restful Servisler
- **3.Hafta, Rest Mimarisi;** Spring Data, JPA, Hibernate
- **4. Hafta, Spring Boot Data;** NoSql Solutions, İlişkisel veri tabanları, Transaction Yönetimi
- **5. Hafta, Security;** Spring Security, JWT
- **6. Hafta, Test Yazım Teknikleri;** Xunit patterns, mock-stub-fake, unit test, integration test, contract test
- **7. Hafta, Uygulama;** Spring Boot, Docker, Postgresql, Github actions

# 1. Bölüm

# Java

## Tarihine Hızlı Bir Bakış

- Sun Microsystems, 1982, Silikon Vadisi
- Oak ismi (Meşe ağacı), elektronik cihazlar için (başlıca interaktif televizyonlar); proje istenen başarıya ulaşamıyor.
- Web'in gelişimi, Java ismine evrilmesi (kahve), 1995, James Gosling ve ekibi
- İlk kararlı sürüm 1999, son kararlı sürüm 2021 (Java 17)
- Esinlendikleri; Smalltalk, C/C++, ...
- Java Community Process (JCP), 1998; standartların oluşmasına, dilin/teknolojinin gelişimine yön veren ekip
- Oracle'ın Sun Microsystems'ı satın alması

<https://webrazzi.com/2009/04/21/oracle-suni-satin-aldi/>

- Mottosu: "Write once, run everywhere"
- <https://docs.oracle.com/javase/tutorial/tutorialLearningPaths.html>

# Java

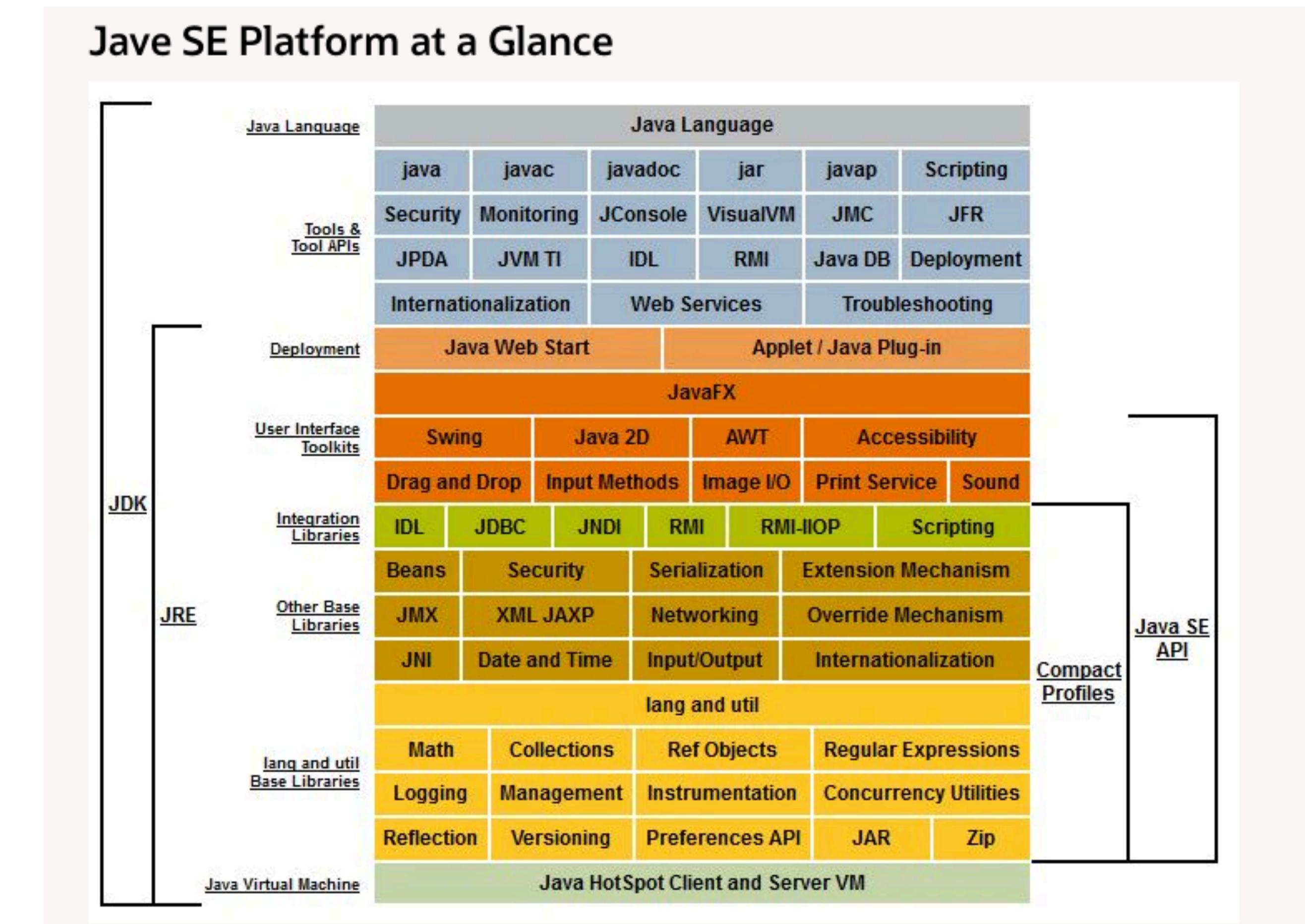
## Genel Özellikleri

- “Java: an Overview”, James Gosling, 1995
  - [https://www.researchgate.net/publication/345758345\\_Java\\_an\\_Overview\\_the\\_original\\_Java\\_whitepaper](https://www.researchgate.net/publication/345758345_Java_an_Overview_the_original_Java_whitepaper)
  - Simple, Object-Oriented, Distributed, Robust, Secure, Architecture Neutral, Portable, Interpreted, High Performance, Multithreaded, Dynamic
- **Compiled languages:** Yazılan kod “execute” edilebilen bir hale (makine dili) çevrilir. C/C++, GO, ...
  - **Interpreted Languages:** Yazılan kod çalışma anında “gerektikçe” makine koduyla eşleştirilir. PHP, Ruby, Python, ...
  - Just in time compilers
  - <https://www.freecodecamp.org/news/compiled-versus-interpreted-languages/>
- Java Virtual Machine (JVM), byte codes
  - Java kodu “compile” olur, compile çıktısı olan byte kodlar JVM tarafından “interpret” edilir. (Aynı kod JVM olan herhangi bir ortamda çalışabilir. Spesifik ihtiyaçlar için mimari detaylara özel kod yazılması gibi durumlar dışında)
- Object Oriented, Java 8+ için fonksiyonel programlama desteği
  - Structred, edger Dijkstra
- **Statically typed languages:** değişkenlerin veri tipleri derleme anında bilinir, hatalar daha erken tespit edilebilir; Java, C/C++, Go, Scala, ...
  - **Dynamically typed languages:** değişkenlerin veri tipleri çalışma anında (runtime) bilinir; Ruby, Python, Javascript, ...
  - <https://stackoverflow.com/questions/1517582/what-is-the-difference-between-statically-typed-and-dynamically-typed-languages>

# Java

## Java Standard Edition

- Java SE (Standard Edition)
- JVM (Java Virtual Machine)
- JDK (Java Development Kit)
- JRE (Java Runtime Environment)
- $\text{JDK} > \text{JRE} > \text{JVM}$
- <https://stackoverflow.com/questions/1906445/what-is-the-difference-between-jdk-and-jre>
- <https://www.oracle.com/java/technologies/platform-glance.html> (image)



# Fundamentals

## Object Oriented Concepts

- Abstraction
- Class
  - Data
  - Behaviour
- Object
- Encapsulation
- Access Modifiers
  - private, public, protected , (default)
- Inheritance
- Abstract Class
- Interface
- Polymorphism
- Overloading
- Overriding
- Method signature
  - Name, parameters (type, sort, count)

# Fundamentals

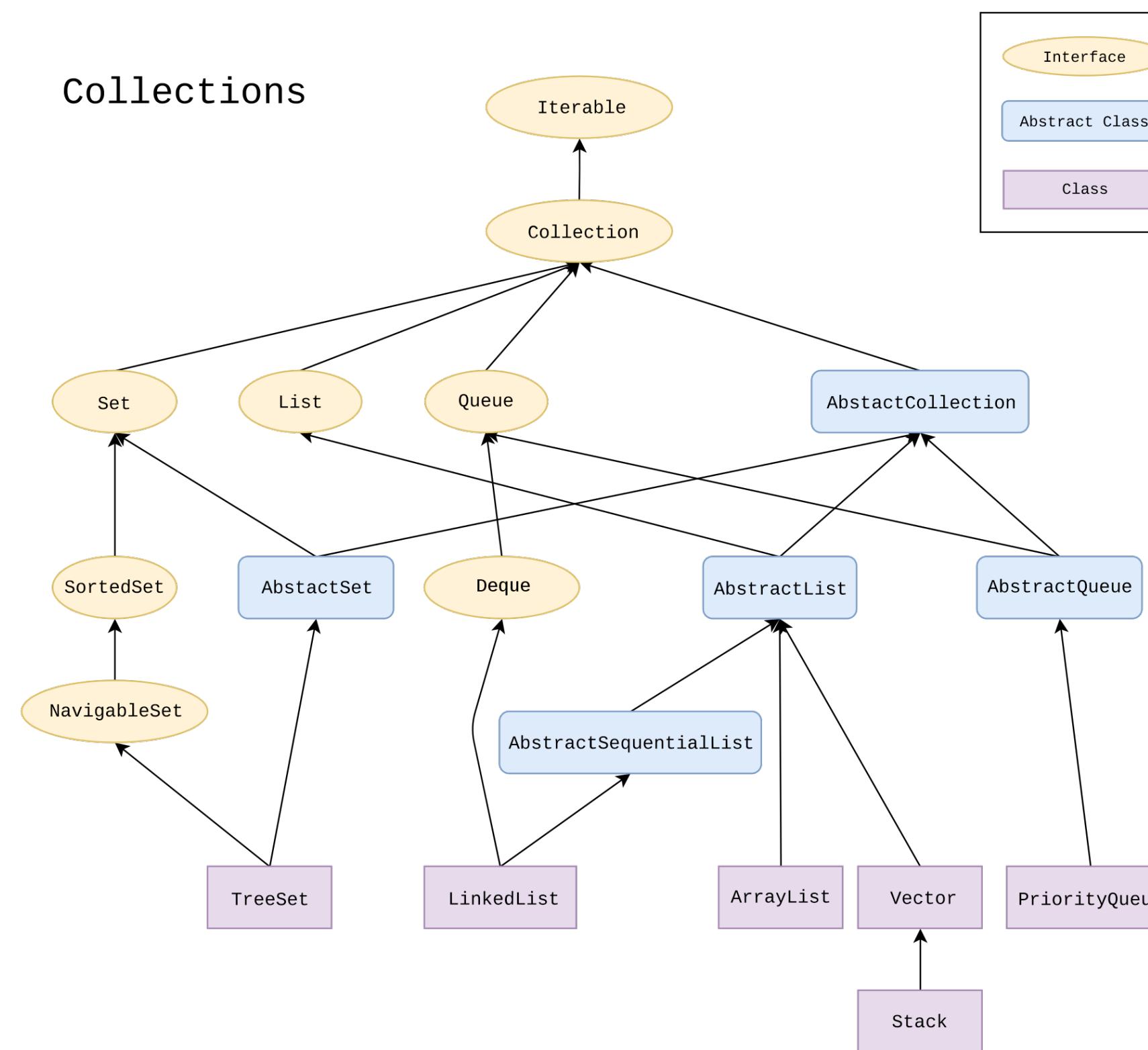
## Data Structure

- Array
  - Bellekte ardışık yerleşim
  - Index değeri üzerinden erişin -  $O(1)$
  - Oluştururken kullanacağı alan belirlidir.
- List
  - ArrayList in Java; Array'lerin dinamik ekleme / çıkarma yapabildiğimiz bir hali olarak düşünebiliriz (bellekte ne kadar yer tutacağı değişebilir.) Index ile erişim mevcuttur -  $O(1)$
  - LinkedList in Java; Sadece bir sonraki (ve /veya implementasyona göre bir önceki) elemanı biliyoruz. Index üzerinden erişim yok
- Map
  - Key - Value çiftlerinden oluşur. Bir map içindeki Key unique degere sahiptir.
  - Set - Map yapısının sadece key' lerden oluşan kısmı olarak düşünebiliriz
  - In Java; HashMap, HashSet
- Array ve ArrayList yapılarında elemanlar eklediğimiz sırada yer alır.
- Map ve Set için sıradan söz edemeyiz.
  - Sıranın korunduğu map için bkzn. SortedMap (interface) TreeMap (implementasyon)
  - <https://docs.oracle.com/javase/tutorial/collections/interfaces/sorted-map.html>

# Fundamentals

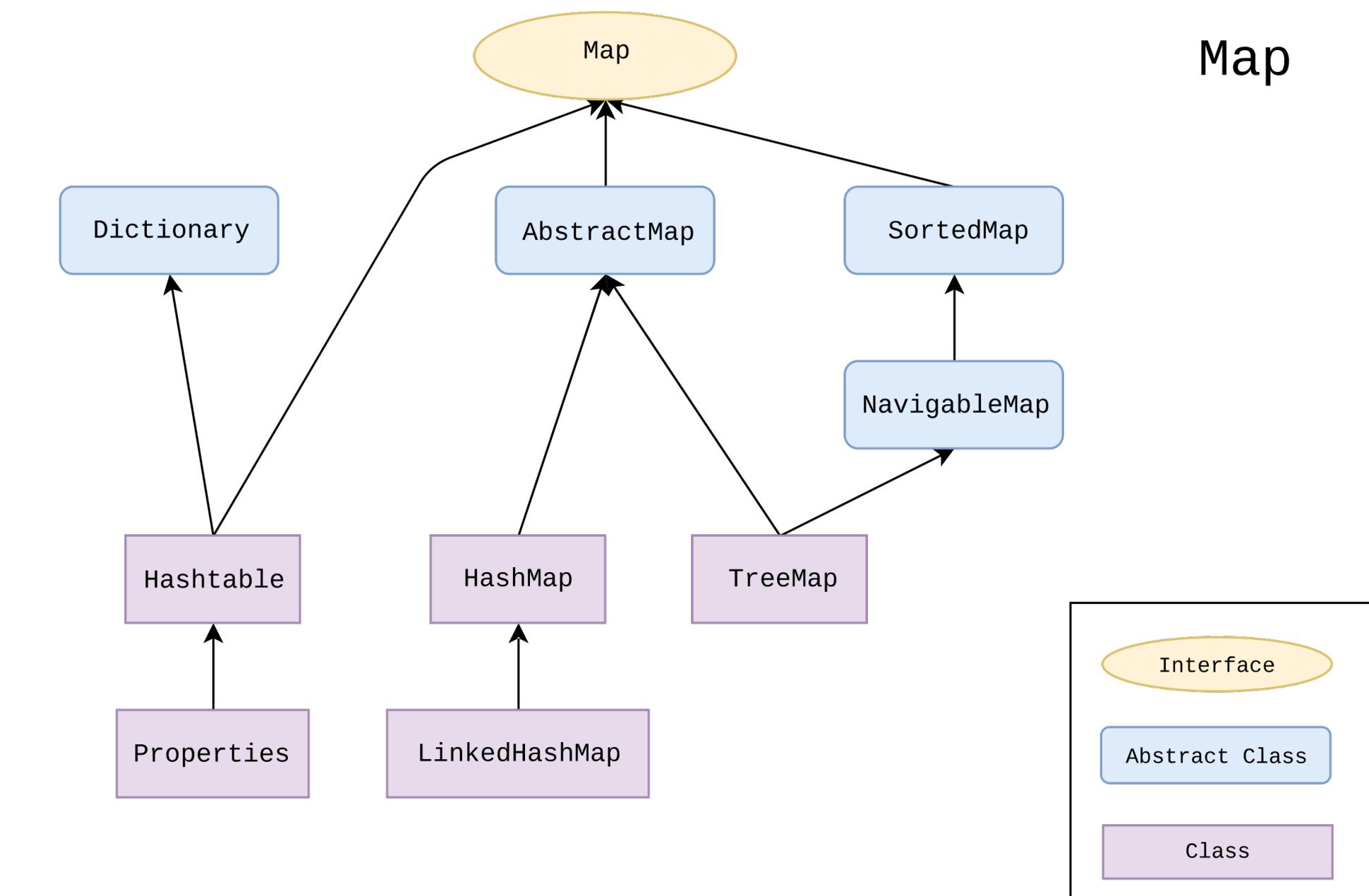
## Java Data Structures

Collections



[https://en.wikipedia.org/wiki/Java\\_collections\\_framework#/media/File:Java.util.Collection\\_hierarchy.svg](https://en.wikipedia.org/wiki/Java_collections_framework#/media/File:Java.util.Collection_hierarchy.svg)

Map



[https://en.wikipedia.org/wiki/Java\\_collections\\_framework#/media/File:Java.util.Map\\_hierarchy.svg](https://en.wikipedia.org/wiki/Java_collections_framework#/media/File:Java.util.Map_hierarchy.svg)

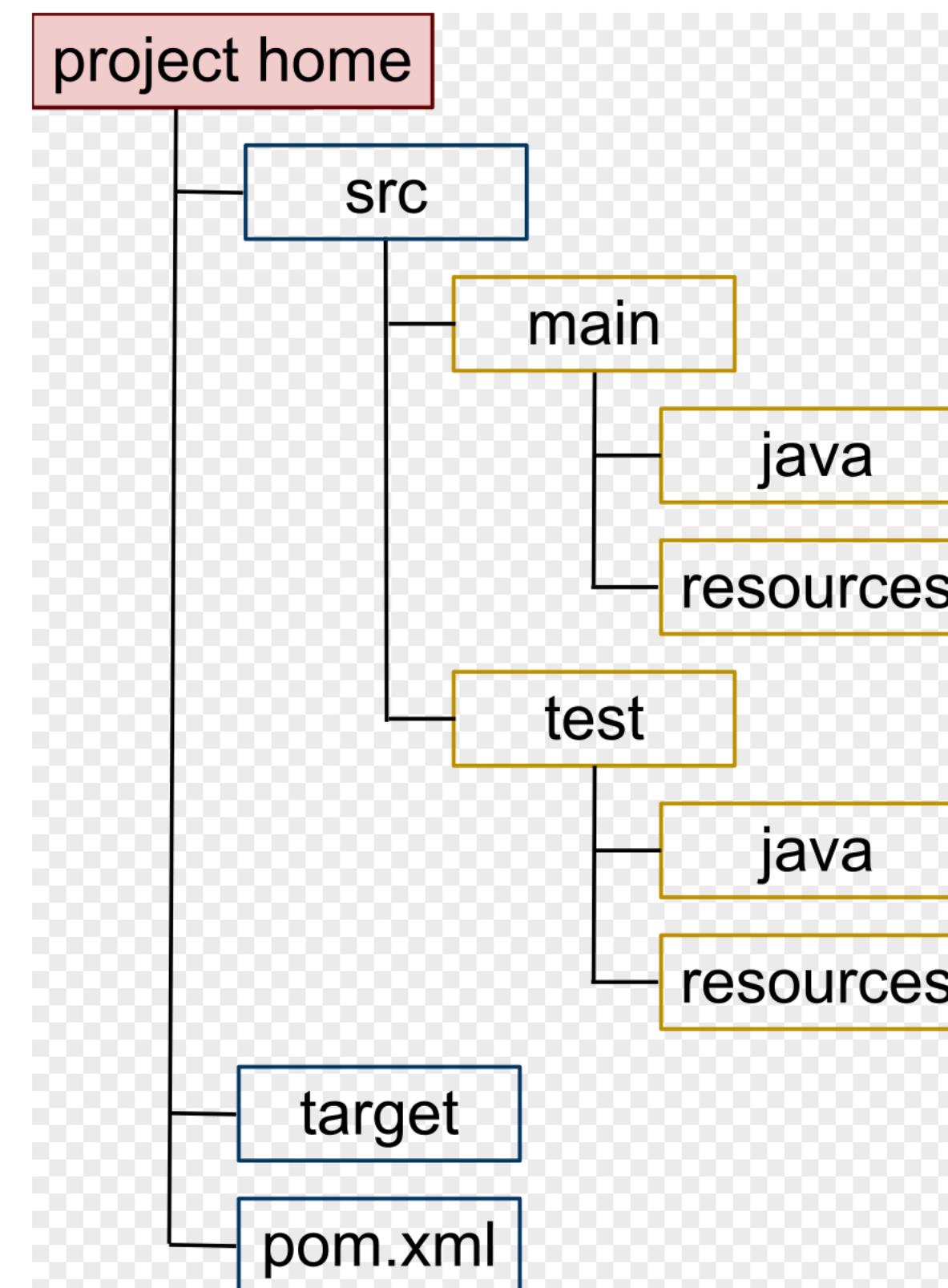
# Maven

## İle Proje Yönetimi

- Maven ile bağılıkların yönetimi
- pom.xml' in incelenmesi
- Popüler Komutlar
  - mvn clean
  - mvn install
  - mvn clean install
  - mvn install -DskipTests=true
- Semantic versioning
  - major.minor.bugfix
  - <https://semver.org/>
- Spring initialize
  - <https://start.spring.io/>
- .gitignore içerisinde neler var

# Maven

## Project Structure



[https://commons.wikimedia.org/wiki/File:Maven\\_CoC.svg](https://commons.wikimedia.org/wiki/File:Maven_CoC.svg)

# Lombok

## Sık Kullanılanlar

- Getter
- Setter
- ToString
- NoArgsConstructor
- AllArgsConstructor
- RequiredArgsConstructor
- Data
- EqualsAndHashCode
- Builder
- With
- SneakyThrows
- Value

# Effective Java'dan Alıntılar

## Effective Java'nın Yaklaşımı

- Effective Java, Third Edition, Joshua Bloch
  - <https://www.amazon.com.tr/Effective-Java-Joshua-Bloch/dp/0134685997>
- Bir programlama dilini öğrenirken 3 ayrı şekilde uzmanlaşın.
  - **Grammer:** Dilin yapısı nasıl, syntax, OOP ve Fonksiyonel paradigmaları nasıl sağlıyor ?
  - **Vocabulary:** Core libraries, Data Structures, problem çözme şekli nasıl ?
  - **Usage:** Dilin yukarıdaki iki madde kapsamında sağladıkları günlük hayatında nasıl işine yarayacak, pragmatik olan nedir, community neyi nasıl sahiplenmiş ?
- Verilen kuralların/tavsiyelerin gözardı edileceği durumlar ile karşılaşabilirsiniz. Benimseyin ama çok sıkı sahiplenmeyin.
- Amaç performanslı kod yazmaya yönlendirmek yerine “clear, correct, usable, robust, flexible, maintainable” kod yazmaya yönlendirmek.
- İçerik: birbirilerine referans veren ama ayrı ayrı da okunabilecek 90 item.

# Effective Java'dan Alıntılar

## Item 1 - Consider static factory methods instead of constructors

- Static factory methods != factory method (in design patterns)
- Constructor'lara isim veremiyoruz. Fakat farklı constructor'ların farklı amaçları var.
- Aynı parametrelerle farklı işler yaptmak constructor'lar ile mümkün değil. (Sebep aynı; isim veremiyoruz).
- Belki de gerçekten yeni obje yaratma ihtiyacımız yok. Static factory metodumuz gereklirse constructor kullanabilir.)
- Amacımıza uygun farklı bir değer return ettirmek isteyebiliriz. (Constructor her zaman ait olduğu sınıf tipinde bir obje oluşturur.)
- Dezavantajı; constructor içerisinde önce parent class'a ait constructor çağrılıyor. Bu işleyişten mahrum kalıyoruz.
- Tavsiye isimlendirmeler; from (type conversion with 1 parameter), of (type conversion with multiple parameters), valueOf, getInstance, newInstance, type, getType, newType

# Effective Java'dan Alıntılar

## Item 2 - Consider a builder when faced with many constructor parameters

- Kastedilen Design Pattern'larda yer alan Builder Pattern değil.
- Constructor ve Static Factory Method ortak bir dezavantaja sahip; parametre sayısının artınca scale olmazlar. (Kullanım sağa doğru uzar).
- Telescoping constructor (**anti-pattern**); gerekli tüm parametre kombinasyonlarıyla constructor'lar oluşturmak.
  - <https://medium.com/@modestofiguereo/design-patterns-2-the-builder-pattern-and-the-telescoping-constructor-anti-pattern-60a33de7522e>
- Builder kullanımı, kodu aşağı doğru uzatır.
  - Benzer analogiyi veri tabanı için de kullanabiliriz. Tablonun büyümесini column sayısını artırarak (sağa) değil, row sayısını artırarak(ağı) görmek isteriz.
- **Avoid** construction with JavaBean style
  - Boş bir constructor çağrııp gerekli parametreleri tek tek atamak
  - <https://stackoverflow.com/questions/22472018/why-java-builder-pattern-over-javabean-pattern>
  - Obje inconsistent bir halde kalabilir; immutable veya thread-safe yapmak zor olur.
- Parametre sayısı 4'ten az ise builder kullanımını tercih edilmeyebilir.

# Effective Java'dan Alıntılar

## Item 4 - Enforce non-instantiability with a private constructor

- Bazı class'lardan obje oluşturmak anlamsızdır.
  - Utility class'lar, amacı static field ve/veya metotları grüplamak olan class'lar.
  - Default constructor'lar bu class'lardan yeni obje oluşturmanın yolunu açar.
  - Abstract yapmak doğrudan obje oluşturmayı engeller ancak dolaylı yoldan o sınıfı extend etmeyi engellemeyecektir.
    - (Bir seçenek class final yapılabilir.)
  - Basitçe private bir constructor ekleyip obje oluşturma ihtimalini ortadan kaldırabiliriz.
    - Kodu okuyan yazılımcıya bilgi verme amacıyla olarak private constructor içerisinde Runtime exception fırlatılabilir.
    - (Bunu yapmadığımızda sonarlint uyarı veriyor.)

# Effective Java'dan Alıntılar

## Item 40 - Consistently use the Override annotation

- *Koddaki Bigram class'ı örneği*
- Override ettiğini sandığın halde override edemiyor olabilirsin.
- @Override kullan compiler seni uyarın.
  - Abstract metodlar için anotasyon koymasan bile compiler uyarır. (Abstract metodlar override edilmelidir.)

# Effective Java'dan Alıntılar

## Item 11 - Always override **HashCode** when you override **equals**

- 2 objenin eşit olması için **HashCode**'larının eşit olması gereklidir.
  - Fakat **HashCode**'ları eşit olan 2 obje her zaman aynı olmayı bilir. (Collision durumları), fakat farklısa her zaman farklıdır.
- **equals** metodu override edilirken hangi field'lar kullanıldığa, **hashCode** metodu override edilirken de o field'lar kullanılmalıdır.
- Equals ve **HashCode**'un birlikte ve doğru override edilmesiyle Map, Set gibi Collection'lar doğru çalışır.
- Lombok `@EqualsAndHashCode`
  - <https://projectlombok.org/features/EqualsAndHashCode>
- **HashCode** hesaplanan bir değer olduğu için değişimdir. Immutable class'larda değişmez.
  - Değişmediği durumda hesaplama maliyeti projenizin performans beklentilerini sekteye ugratıyorsa cache'lenebilir.

# Java 8 ve sonrası

## Göze çarpan değişiklikler

- Stream API, `java.util.Stream`
  - Stream = sequence of data items
  - Stream'in farklı bölgeleri farklı cpu core'lar üzerinde çalışabilir. (Kodu paralelleştirmek kolaylaştı)
- lambda function, method reference
  - Behaviour parameterization = davranış parametrik hale getirme (anonymous class'lar yerine)
  - Nasıl ki user input değişecek diye metodlara sabit değer koymayarak parametre alır halde yazıyoruz, value'ler gibi "behavior" lar da değişimdir. Kodumuz daha esnek olur.
  - First class citizen (functional programming style)
- Predicate<P> predicate = Return değeri Boolean olan ve parametre olan P tipinde değer alan bir lambda function veya metot assign edilebilir.
- Optional<T> = to avoid null pointer exceptions
  - Anekdot; Tony Hoare, 1965, invention of null reference, "my billion dollar mistakes"
- default methods in interfaces

# Java 8 ve sonrası

## Lambda Functions

- Lambda calculus, Alonzo Church, 1930'lar
- Lambda expression özellikleri
  - İsmi yok (anonim)
  - Function (metot değil, side effect'i yok)
  - Bir değişkene atayabiliriz veya bir metoda parametre olarak geçebiliriz.
  - Kısa ve sadedir.
- (Lambda parameters) -> { lambda body};
- Method reference
  - Daha okunabilir kod
  - Bazı lambda ifadeleri, metot reference olarak yazılabilir.
- <https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html>

# Effective Java ’dan Alıntılar

## Chapter 7 - Lambdas and Streams

- **Item 42:** Prefer Lambdas to anonymous classes
  - Boilerplate kodu azaltır.
- **Item 43:** Prefer method references to lambdas
  - Shorter + clearer code
- **Item 44:** Favor the use of standard functional interfaces
  - Java'da işini görecek functional interface'ler varsa onları kullan, kendin yazma. Yazarsan da @FunctionalInterface'i ekle
- **Item 45:** Use streams judiciously
  - Bazen basit bir for stream'dan daha anlaşılır olabilir. (Bazen)
- **Item 48:** Use caution when making streams parallel
  - Gerçekten ihtiyacın olana kadar stream'ler için .parallel() kullanma. Kullandığın takdirde çok iyi test et.

# 1. Bölüm Sonu

## Neler Yaptık ?

- Java Programlama Dili hakkında genel bilgiler
- Object Oriented Kavramlar
- Array, Map, Set, List veri yapıları
- Java Collections
- Spring initializr
- Maven ile proje yönetimi, pom.xml
- Sık kullanılan Lombok anotasyonları
- Java streams, lamda function, method reference
- Effective Java Item 1, 2, 4, 40, 11 ; 42, 43, 44, 45, 48

# 1. Bölüm Sonu

## Ödev 1 /Araştırma Soruları

1. Pass by value, pass by reference kavramları nedir ? Java ile ilişkili olarak açıklayınız.
2. Immutability nedir, neden önemlidir ? Bir Java sınıfı nasıl immutable yapılır ?
3. Framework ve library arasındaki fark nedir ?
4. Java'da Garbage Collector' un görevi nedir ?
5. Memory leak nedir ? Java'da memory leak oluşması mümkün müdür ?
6. Yeni Java sürümleri ne sıklıkla çıkmaktadır ?
7. Stack ve Heap nedir ? Java'da hangi yapılar stack ile, hangi yapılar heap ile ilişkilidir ?
8. OpenJDK ve OracleJDK arasındaki farklar nelerdir ?
9. @FunctionalInterface anotasyonu nerelerde kullanılabilir, neleri sağlar ?
10. Java'da hangi functional interface'ler yer almaktadır ? Yaptığınız araştırmada en popüler/göze çarpan functional interface'ler hangileridir ?

# 1. Bölüm Sonu

## Ödev 2 /Kodlama

- <https://martinfowler.com/articles/collection-pipeline/>
- Beklenenler (yukarıda yer alan yazıdan)
  - Ana başlıkların birkaç cümleyle özetinin çıkarılması
  - Spring initializr ile bir proje oluşturulması; Lombok ve spring-boot-starter-test dependency'lerinin eklenmesi
  - Yazıda yer alan operatörlerin Java'daki karşılıklarının bulunup her biri için birer örnek metot yazılması (bazı operatörlerin birebir karşılığı olmayabilir, küçük dokunuşlar yapmak durumunda kalabilirsiniz)
  - (Bonus) Metotlara Junit ile test yazılması

# **2. Bölüm**

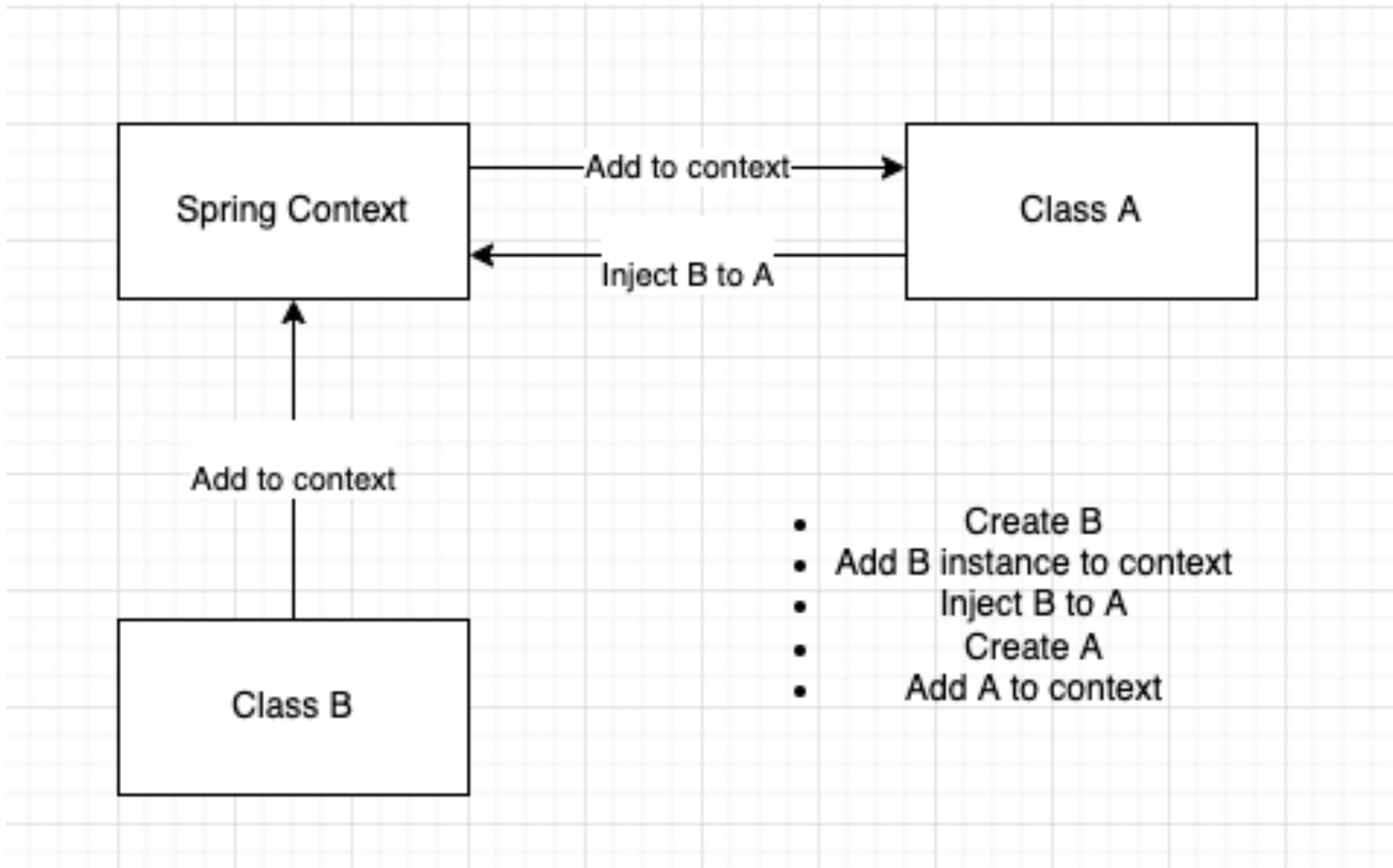
# Inversion of Control

- Farklı kaynaklarda pattern, design principle, yazılım geliştirme tekniği gibi farklı ifadeler yer alıyor.
- Kodumuzdaki bileşenler arasındaki ilişkileri yönetme akışını framework'e bırakıyoruz.
  - OOP için sınıf / obje
  - "Ne" yapacağımızı halen biz yönetiyoruz. "Ne zaman" yapacağımıza ilişkin kararların ilgili kısımları framework' e hallediyor.
  - Obje ne zaman oluşturulacak, ne zaman bağımlılık kurulacak, ne zaman yok edilecek
  - Böylece daha az boilerplate kod oluşuyor ve işimize odaklanabiliyoruz. Normalde kod yazmamız gereken kısımları framework çözüyor.
- Framework'ü library' den ayıran özelliklerden biridir.
- Dependency Injection (En popüler kullanımı)
- Diğer kullanımlar; Service Locator, Events, Delegates, ... ?
- <https://stackoverflow.com/questions/3058/what-is-inversion-of-control>

# Dependency Injection

- IoC daha genel bir kavram. DI, IoC uygulamanın yollarından biri.
- Hollywood' Law, “Don't call us, we'll call you”,
  - <https://www.digibarn.com/friends/curbow/star/XDEPaper.pdf>
- Hayatımızdan “new” keyword' ünü çıkarmaya başlıyoruz.
- Objelerin oluşturulması, bağımlılıklar kapsamında birbirine atanması işlemleri ile doğrudan ilgilenmemiş oluyoruz.
- Injection types; Constructor, field, method (setter).
- <https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring>
- <https://www.martinfowler.com/articles/injection.html>

# Dependency Injection Örnek Akış



Bu akışı programcı değil, framework yönetiyor. Akışın bütünü Inversion of Control kapsamında değerlendirilebiliriz.

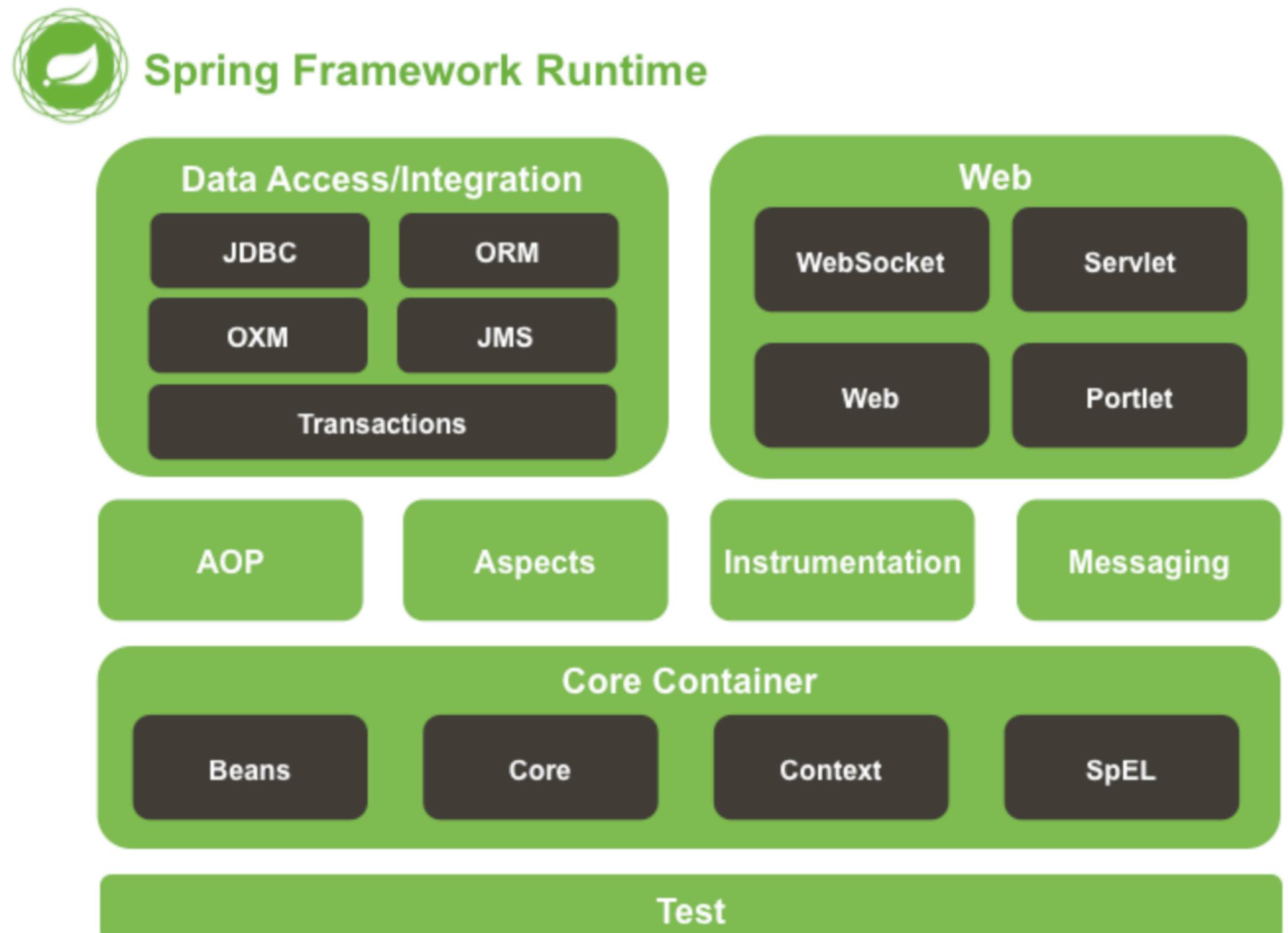
Dependency Injection ise B'ye ait bir objenin A sınıfına verilmesidir.

A, B'ye ihtiyaç duyuyor ama B'yi create etme ve A'ya geçirme işini framework yapıyor.

# Spring Framework

## Headlines

- <https://spring.io/>
- Java ile kullanabileceğimiz açık kaynak kodlu bir framework.
- **Spring core**
  - Inversion of Control / Dependency Injection
- **Spring Boot**
  - Spring Framework'ün extend edilmiş hali diyebiliriz.
  - Spring projesi oluşturmayı ve yönetmeyi kolaylaştırır.
  - Embedded Tomcat (veya Jetty) sever barındırır.
  - Spring modüllerinin versiyonlarını ayrıca yönetmemize gerek kalmaz.
  - <https://spring.io/projects/spring-boot>
- **Spring Cloud**
  - Cloud üzerinde çalışacak uygulamaların ihtiyaçlarına yönelik çözümleri barındırır.
  - Load balancer, circuit breaker, distributed messaging, service communications, ...
  - <https://spring.io/projects/spring-cloud>



<https://docs.spring.io/spring-framework/docs/4.0.x/spring-framework-reference/html/overview.html>

# Spring Framework

## Fundamentals

- XML based configuration vs Annotations
- @Component
  - Class seviyesinde kullanılır. Spring' e bunu sınıfından obje oluşturma ve/veya ihtiyaç duyduğu objeleri atama işini sen yapmış oluyoruz.
  - Spring, component'leri bulur, obje oluşturur ve ApplicationContext'e ekler.
  - @RestController, @Controller, @Service, @Repository, @Configuration
  - Scope; singleton (default), prototype
- @Bean
  - Metot seviyesinde kullanılır.
  - ApplicationContext'e kendi ihtiyaçlarımızı gören bir objeyi doğrudan eklemek üzere Spring' e bırakmış oluyoruz.
  - Objeyi oluşturma işini biz yönetmiş oluyoruz. Spring sadece dependency' i yönetir
- @Autowired
  - Doğrudan kullanmasak bile Spring Boot gerekli bağımlıkları tespit edip yönetebiliyor.
- Spring proxy mechanism
  - <https://spring.io/blog/2012/05/23/transactions-caching-and-aop-understanding-proxy-usage-in-spring>

# JSON

## JavaScript Object Notation

- Veri gösterme (represent) formatlarında biri, veriyi saklama (store) veya taşıma (transport) işlemlerinde kullanabiliriz.
- Çoğunlukla network üzerinde veri transferinde kullanılır. İstenen durumlarda objelerin JSON karşılığı cache'lenebilir veya veri tabanına yazılabilir.
- Herhangi bir programlama dilinden bağımsızdır. Genelde her programlama dilinde JSON parse etmeyi / objeleri JSON'a çevirmeyi sağlayan kütüphaneler bulunur.
- Popüler diğer alternatifler YML ve XML olarak söylenebilir. Bu 3 veri gösterme yöntemi de “human readable” dır.
- Bir JSON ifadesini nested key-value çiftleri olarak düşünebiliriz.
- Desteklediği veri tipleri; number, string, boolean, array, object, null
- Jackson, popüler bir JSON parser’ı (maven’da dependency ekleyerek kullanabiliyoruz.)
- <https://www.json.org/json-en.html>

# JSON

## JSON vs Java Classes

```
1  {  
2      "name": "Baeldung",  
3      "area": "tech blogs",  
4      "author": "Eugen",  
5      "id": 32134,  
6      "topics": [  
7          "java",  
8          "kotlin",  
9          "cs",  
10         "linux"  
11     ]  
12 }  
13
```

```
8  @Getter  
9  @Setter  
10 public class Input {  
11  
12     private String name;  
13     private String area;  
14     private String author;  
15     private Integer id;  
16     private List<String> topics;  
17 }  
18
```

# REST Architecture

## Genel Bilgiler

- Roy Fielding, Architectural Styles and the Design of Network-based Software Architectures, Chapter 5, 2000, Doktora Tezi
  - Representational State Transfer, Architectural Style
  - [https://www.ics.uci.edu/~fielding/pubs/dissertation/rest\\_arch\\_style.htm](https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm)
  - Daha sonra sektörde yaygınlaşmaya başladı. Ancak pratikte her soruna çözüm değil.
- Temel olarak REST = HTTP (Hyper-Text Transfer Protocol) üzerine eklenmiş kurallar
  - HTTP kullanmadan REST servisler geliştirme teorik olarak mümkün değildir.
  - REST pratiklerini gözardı ederek HTTP servisler geliştirmek mümkündür. (Bazen de bunu istemeden yaparız.)
  - HTTP, TCP ile çalışır, dolayısıyla REST servisler de TCP üzerinde çalışır.
  - REST pratiklerini uygulamak yeni bir network protocol kullandığımızı göstermez.

# REST Architecture

## HTTP' den REST' e Geçiş

- Architectural Constraints
  1. **Client Server;** Client ve Server birbirine doğrudan bağlı değildir. Bağımsız olarak üzerlerinde çalışabilir, değişiklik yapabiliriz.
  2. **Stateless;** server aldığı request'leri, öncekilerden bağımsız olarak tek başına değerlendirir. Session / history yok.
  3. **Cache;** Response' un cacheable olup olmadığını belirtiliyorsa client cache' eleyebilir.
  4. **Uniform interface;** Sunucunun hizmete sunduğu kaynaklarlar ilişkilendirdiği tek bir URI (identifier) olmalıdır ki client istediği kaynağa bunu kullanarak erişebilsin.
  5. **Layered System;** server' ların görevlerini ayırip katmanlı bir şekilde kullanabiliriz. Authentication, Business, Data Store
  6. **Code-On Demand (optional);** “You are free to return executable code”. Böylece client, feature'ları extend edebilir.
- <https://restfulapi.net/rest-architectural-constraints/>
- REST denince aklımıza gelmesi gerekenler; resources / groups , identifiers , HTTP metodlarının doğru kullanımı, endpoint' lerin state bağımsız olması
- Naming convention; <https://restfulapi.net/resource-naming/>
- REST pratikleri ile geliştirdiğimiz servislere RESTful servisler deriz.

# Restful Services With Spring

## Popular Annotations

@RestController

@ResponseStatus

Http Methods

@RequestMapping

@PostMapping

@GetMapping

@PutMapping

@DeleteMapping

@PostMapping

@RequestHeader

@PathVariable

@RequestParam

@RequestBody

Validations

@Valid

@NotNull

@NotBlank

@NotEmpty

@Max, @Min, @Size

@ConfigurationProperties

# Clean Code

## A handbook of Agile Software Craftsmanship

- Robert C. Martin (Uncle Bob), 2008
- Principles, patterns, practices of writing clean code
- Complexity ve çözümleri üzerine senaryolar
- “Code smells” üzerine derlem (tek chapter)
- “İyi” kod nedir ? “Kötü” kod nedir ? Kötü kodu iyi koda nasıl çeviririz ?
- İyi isimlendirme, OOP prensiplerini temiz kullanma, açık/anlaşılır fonksiyonlar yazma
- Unit test ve TDD (Test Driven Development) üzerine güzel pratikler
- Boy Scout Rule (İzci Kuralı)
- Temiz kod yazma üzerine tek kaynak değil, -bazlarının bu kitaptakilerle ters düştüğü prensipler içeren-farklı kaynaklar bulunabilir.

# Clean Code

## Chapter 2 - Meaningful Names (1)

- Use intention-revealing names
- Eğer yanına comment eklemek zorunda kaldıysan daha iyi bir isim bulmalısın.
  - int d; // elapsed time in days
  - int elapsedTimeInDays;
- Avoid disinformation
  - accountList diyorsan gerçekten bir list olsun
  - 0 vs o . 1 vs l . (Sıfır ve küçük 0, bir ve küçük l. IDE ayrimı tam gösteremiyor olabilir.)
- Make meaningful distinctions
  - ProductInfo ve ProductData. Görevler tam net değil
- Use pronounceable names
  - Date modymdhms -> modification year month day hour minute second
  - Date modificationTimestamp -> zaten tipi Date

# Clean Code

## Chapter 2 - Meaningful Names (2)

- Use searchable names
  - Tek harflik isimler veya kodda yer alan sayılar hatırlanabilir isimlere sahip olursa arama kolaylaşır. (7 yerine MAX\_CLASSES\_PER\_STUDENT)
- Avoid Encodings
  - IMoviveService -> Interface olduğunu belirtmek için I koyma. (Java'da List -> ArrayList, Map -> HashMap)
- “Smart programmer” ve “Professional Programmer” arasındaki fark = “Professional Programmer” şunu bilir -> “Clarity is king”
- Class names -> noun olmalıdır. Customer, Account, AddressParser, ...
- Method names -> verb olmalıdır. save(), deletePage()
- Pick one word per concept
  - Fetch, retrieve, get ? Benzer anlamlar, hangini ne için kullanıyorsun?
- Okunabilirliği artırmak adına rename etmekten kaçınmayın. IDE' ler refactoring konusunda zaten işinizi kolaylaştıracak.

# Clean Code

## Chapter 2 - Meaningful Names (3)

```
public List<int[]> getThem() {  
    List<int[]> list1 = new ArrayList<int[]>();  
  
    for (int[] x : theList)  
        if (x[0] == 4)  
            list1.add(x);  
  
    return list1;  
}
```

```
public List<int[]> getFlaggedCells() {  
    List<int[]> flaggedCells = new ArrayList<int[]>();  
  
    for (int[] cell : gameBoard)  
        if (cell[STATUS_VALUE] == FLAGGED)  
            flaggedCells.add(cell);  
  
    return flaggedCells;  
}
```

# Clean Code

## Chapter 3 - Functions (1)

- İlk kural, fonksiyonlar “small” olmadı. İkinci kural fonksiyonlar “smaller” olmalı.
  - KISS - Keep it simple and short
  - Sizce maksimum kaç satır olmalı ?
- if, else if, while içerisindeki ifadeler 1 satırı geçmemeli
- Bir fonksiyon yalnız 1 şey yapmalı, onu iyi yapmalı, sadece onu yapmalı.
- Switch-case kullanıyorsanız ve kod uzunsa, switch statement' larını fonksiyonlara bölebilirsiniz. (Veya daha iyisi polymorphism ile kodu refactor edebilirsiniz.)
- Fonksiyona isim seçmek zaman harcamaya değer, isim açıklayıcı olacaksa uzun olmasında bir sakınca yok. Aynı tip fonksiyonlarda benzer isimleri kullan (insert-save-create)
- Bir fonksiyonun alacağı ideal parametre sayısı 0' dır. 3'ten fazla olursa bir sınıf oluşturmayı düşünebiliriz.
- Flag argument tercih etme (fonksiyona boolean değer geçip true veya false olup olmadığını göre logic oluşturmak. Bu durumda aslında 2 fonksiyona ihtiyacın var demektir.

# Clean Code

## Chapter 3 - Functions (2)

- Argument Objects
  - Circle makeCircle (double x, double y, double radius)
  - Circle makeCircle (Point center, double radius)
- Side effect yaratma. Kodda beklenmedik bug' lara sebep olabilir.
- Command Query Separation
  - Bir fonksiyon ve bir iş yapmalı ya da bir soruya cevap vermelii. Ancak ikisini aynı anda yapmamalı.
- Prefer exceptions to Returning error codes
  - Try-catch bloklarını mümkün olduğunca küçük tut
  - "Error handling is one thing" (Bknz. "Bir fonksiyon yalnız ı şey yapmalı")
- Don't repeat yourself (DRY principle)
  - Duplication'ları oluşturmak kolay, maintain etmek zordur.
  - Her şeyi hemen reusable yapmaya kalkmayın. (Rule of three)
  - YAGNI (you are not gonna need it)

# Clean Code

## Chapter 4 - Comments

- “Don’t comment bad code - rewrite it” (Brian W. Kernighan and P. J. Plaugher)
- Kod değişir, test ederiz. Yorumlar için aynı şey geçerli değil. Eski kalırlarsa gözden kaçma, kod ile güncelliği koruyamama ihtimali yüksek.
- Yorumları kötü kodu “süslemek” için kullanma.
- Mümkün olduğunda kod kendini açıklasın.
- Aşikar olan bir durum için yorum yazmak sadece kodu kalabalıklaştırır.
- Class’ı kimin ne zaman oluşturduğuna dair class’ın üzerine eklenen yorumlar anlamsız - git zaten bunu tutuyor.
- Kodu yorumla alıp commit etmek iyi bir alışkanlık değil. Kod neden yorumla, ne zaman açılacak vs kafa karışıklığına sebep olur.
- Peki hiç mi yorum yazmayalım ?
  - Legal Comments - copyright, license bilgileri vs
  - Bilgilendirici yorumlar - bir formatter veya regex ifadesi için gerekebilir
  - Bazen kodda yapılan bir işlem için neden o şekilde bir karar alındığı sorgulanabilir. Bu durumlarda diğer geliştiriciler için not bırakmak elzem olabilir. Örneğin bir performans iyileştirmesi yapmak için kodu uzatmak, farklı veri yapısına çevirmek gibi bir durumda kaldınız. İlk bakış bu kod daha basit yazılabilemiş algısı oluşabilir.
  - `assertTrue(a.compareTo(b) == -1); // a < b`
  - TODO comments (çok uzun orada kalmalarına izine vermeyin, yapın veya iş listesine iş açın. Muhtemelen oradaki kod değişecek.)
  - JavaDocs in Public APIs

# Clean Code

## Ek kaynaklar

- <https://medium.com/@busrauzun/clean-code-kitabindan-notlar-1-temiz-kod-derken-44e6f7a27ebo>
- <https://martinfowler.com/bliki/FlagArgument.html>
- <https://martinfowler.com/bliki/CommandQuerySeparation.html>
- <https://blog.cleancoder.com/uncle-bob/2021/03/06/ifElseSwitch.html>
- <https://www.youtube.com/watch?v=7m88T8-96X4> (Temiz Kod Tasarımı - Lemi Orhan Ergin)
- <https://www.amazon.com.tr/Refactoring-Improving-Existing-Addison-Wesley-Signature/dp/0134757599/>
- <https://www.amazon.com.tr/Philosophy-Software-Design-2nd/dp/173210221X/>

# 2. Bölüm Sonu

## Neler Yaptık

- Inversion of Control, Dependency Inversion Kavramlar
- Spring Framework'e giriş
- JSON
- REST Architecture
- Spring Boot ile RESTful servisler geliştirme
- Clean Code; Meaningful Names, Functions, Comments

# 2. Bölüm Sonu

## Ödev 2 / Araştırma (Son tarih: 7 Ocak Cuma 23:59)

1. Spring dışında dependency injection için kullanabileceğimiz framework'ler / kütüphaneler nelerdir ? (Herhangi bir programlama dili için olabilir.)
2. @SpringBootApplication anotasyonu hangi anotasyonları kapsamaktadır ? Görevlerini kısaca açıklayınız.
3. @Primary, @Qualifier anotasyonlarının kullanım amaçlarını açıklayınız.
4. Convention over configuration kavramını seçtiğiniz bir örnek üzerinden açıklayınız.
5. Aspect Oriented Programlama nedir ? Avantajları ve dezavantajları nelerdir ?
6. SOLID prensiplerini kısaca açıklayınız. Sizce her koşulda bu prensipler çerçevesinde mi kod yazılmalıdır ? Neden ? (High cohesion, low coupling)
7. Swagger nedir, ne amaçla kullanılmaktadır ?
8. Richardson Maturity Model'i seviyeleriyle birlikte açıklayınız.
9. URL, URI, URN kavramlarını bir örnek üzerinden açıklayınız.
10. Idempotency nedir ? Hangi HTTP metotları idempotent' tir ?
11. RFC (Request For Comment) neyi ifade etmektedir ? HTTP hangi RFC dokümanında açıklanmıştır ? Bu dokümanda HTTP hakkında ne tür bilgiler yer almaktadır ?

# 2. Bölüm Sonu

## Ödev 2 / Kodlama (Son tarih: 7 Ocak Cuma 23:59)

- Film ve Kullanıcı CRUD endpoint'lerinin REST pratiklerine uygun olarak yazılması (Aşağıdaki 6 maddeye karşılık gelen birer metot yazılacak. Metotlarınızı postman üzerinden test etmeniz tavsiye olunur.)
  1. Create -> HTTP 201 Dönecek
  2. GET isteği için HTTP 200 dönecek
  3. DELETE için HTTP 204 dönecek
- MemberController (gerekli field'lar tarafınızdan belirlenecek.)
  4. Kullanıcı filmlere puan verebilecek - member id, movie id, point
  5. Kullanıcılarına kendilerine izleme listesi oluşturabilecek (watchlist) - member id, watch list name
  6. Kullanıcılar izleme listelerine film ekleyebilecek - watch list id, movie id
- Controller sınıfındaki metodlar builder ile oluşturulmuş dummy veri dönecek.
- (Bonus) Projeye Swagger entegre edilmesi

# 3. Bölüm

# İlişkisel Veritabanları -1

**By Edgar F. Codd (Turing Ödüllü Bilgisayar Bilimci, 1981)**

- 1970 yılında Edgar Codd tarafından yayınlanan bir makalede, “Relational Algebra” ile temellendirilen karamsar ile “Relational Database” kavramı hayat bulmuş oldu.
- “Relation” kavramı matematik ile ilişkili olup küme teorisine (set theory) dayalıdır. İlişkisel dememizin sebebi, küme ilişkileri kurarak işlem yapabilmemizdir (kesişim-intersection, birleşim-union gibi)
  - Foreign key tanımlayabiliyor olmak tek başına ilişkisel kavramını ifade etmek için yeterli değildir.
- **Temel yapılar:** Row, Column, Table, Result Set (View)
- **Kısıtlar (Constraints):** Primary key, foreign key, not null, unique
- **Küme Teorisi:** Union, difference, intersection, cartesian product, projection
- **Operasyonel İşlem Grupları:** Data Definition Language (DDL), Data Manipulation Language (DML)
- Bilişim sistemlerine 2 temel katkı sağlamıştır: Verimlilik ve Güvenirlik
  - Sizce ilişkisel veri tabanlarından önce nasıl bir hayat vardı ?

# İlişkisel Veritabanları -2

## Sql ve Postgresql

- SQL - Structured Query Language
  - 1973 IBM, 1986 ANSI, 1987 ISO
  - Square > Sequel > SQL
  - Programlama dili değil, İlişkisel veri tabanları için “Declarative” ve “Domain Specific Language”
  - Farklı sorgulama dilleri; GraphQL, CQL (Cassandra QL), HQL (Hibernate QL) vs.
  - İlişkisel veri tabanları desteklemektedir; PostgreSQL, MySQL; OracleSQL, MsSQL vs.
- PostgreSQL = Postgres + SQL
  - <https://www.postgresql.org/>
  - Ingres veritabanının devamı olarak Berkeley Üniversitesi tarafından Postgres adıyla geliştirilmeye başlandı. (1982)
  - SQL desteği 1996 yılında verilmeye başlandı ve PostgreSQL adını aldı.

# Object Relational Mappings

## ORM Tools

- Object oriented bir yapı üzerinden ilişkisel veritabanına ait işlemleri gerçekleştirmemizi kolaylaştırıyor.
- Kolaylaştırıyor (?)
  - <https://blog.codinghorror.com/object-relational-mapping-is-the-vietnam-of-computer-science/>
- Hibernate, Java, 2001
  - 1-1, 1-n, n-n ilişkileri yönetebiliriz. (Genel ifadeyle “Class” yapısı, “Table” yapısı ile; “field” yapısı “column” yapısı ile eşleşmektedir.
  - Hibernate Query Language (HQL) ile SQL’e benzer bir syntax ile çalışabiliriz.
  - Search, validator, history <versiyonlama> (Envers) gibi ek özellikleri destekler
  - 2010 yılından itibaren JPA ile uyumlu olarak kullanılabilmektedir.
- JPA ( Java Persistence API, Jakarta Persistence?)
  - Implementasyon değil specification’dır. Hibernate, EclipseLink gibi kütüphanelerin desteği vardır.
  - Java dünyasındaki ORM araçları için ortak bir arayüz oluşturmaktadır.
  - JPQL (Java Persistence Query Language)

# Spring Data

- <https://spring.io/projects/spring-data>
- İlişkisel veritabanları için ayrıca ihtiyaç yok.
  - JPA ve JDBC desteği
- NoSQL çözümlerinin her biri için ise ayrı modüller mevcuttur.
  - MongoDB, Redis, Cassandra, Neo4j, ...
- Spring data JPA
  - @Id, @GeneratedValue(GenerationType.Identity)
  - @Table, @Entity, @Column,
  - @OneToOne(fetch = FetchType.Eager) , @JoinColumn(name = "entityId")
  - @ManyToOne(fetch = FetchType.Eager), @JoinColumn(name = "entityId")
  - @OneToMany(mappedByBy = "..")
  - .... extends JpaRepository <EntityName, idDataType>

# Spring ile Transaction Yönetimi

## @Transactional

- **Transaction:** Veritabanı üzerinde yapılan bir işlemi ya da işlemler bütünü ifade eder. İlişkisel veri tabanlarında bir transaction ACID kriterlerini sağlar.
  - Begin - Commit - Rollback
- ACID kriterleri
  - **Atomictiy:** Bir transaction kapsamında yapılan N işlemin ya hepsi gerçekleşir ya da hiçbiri gerçekleşmez.)
  - **Consistency:** Bir transaction veritabanını daima consistent durumda bırakır. (Hatalı durumlarda rollback mekanizması durumu transaction'ın başladığı ana getirir.)
  - **Isolation:** Bir transaction açısından diğer bir transaction ya hiç başlamamıştır, ya da sona ermiştir. Transaction, diğer transaction'ların yaptığı işlemleri tamamlanmadan göremez. (Farklı isolotion seviyeleri için bu durum farklılık gösterebilir.)
  - **Duration:** Transaction commit edildikten sonra sistemde meydana gelen sorunlar transaction'ın yaptığı işlemleri etkilemez.
- **@Transactional**
  - readOnly, propagation, isolation

# Ayrıca Bakınız

- Lazy initialization problem
- Hibernate n + 1 problem
- Lost update problem
- Optimistic Lock
- Pessimistic Lock
- <https://stackoverflow.com/questions/129329/optimistic-vs-pessimistic-locking>

# Jpa Kullanmanın Püf Noktaları

## By Kenan Sevindik

- <http://www.kenansevindik.com/jpahibernate-pratikleri-ve-puf-noktalari/>
- <https://twitter.com/kenansevindik/status/1332238882230361025>
- İlişkilerin lazy tanımlanması (default olmayanların da)
- 1-N ilişkilerde, genel olarak sorgulamaların 1-N tarafında, ekleme - çıkarma - güncellemelerin N-1 tarafında yapılması
- N-N ilişkilerin ara tablo üzerinden kullanılması
- CascadeType.ALL yerine özellikle ihtiyacınız kadarının kullanılması (persist, merge, vs)
- Primary Key oluşturma yöntemlerinden SEQUENCE 'in tercih edilmesi. (Eğer Hibernate' in batch insert yöntemini kullanırsanız özellikle sequence tercih etmeniz gereklidir.)
- Hiyerarşilerin mümkün olduğunca tek tablo üzerinden kurulmaya çalışılması (Tek tablo olsa da farklı entity'ler ile eşleştirilebiliriz.)
- ....

# NoSQL Çözümler

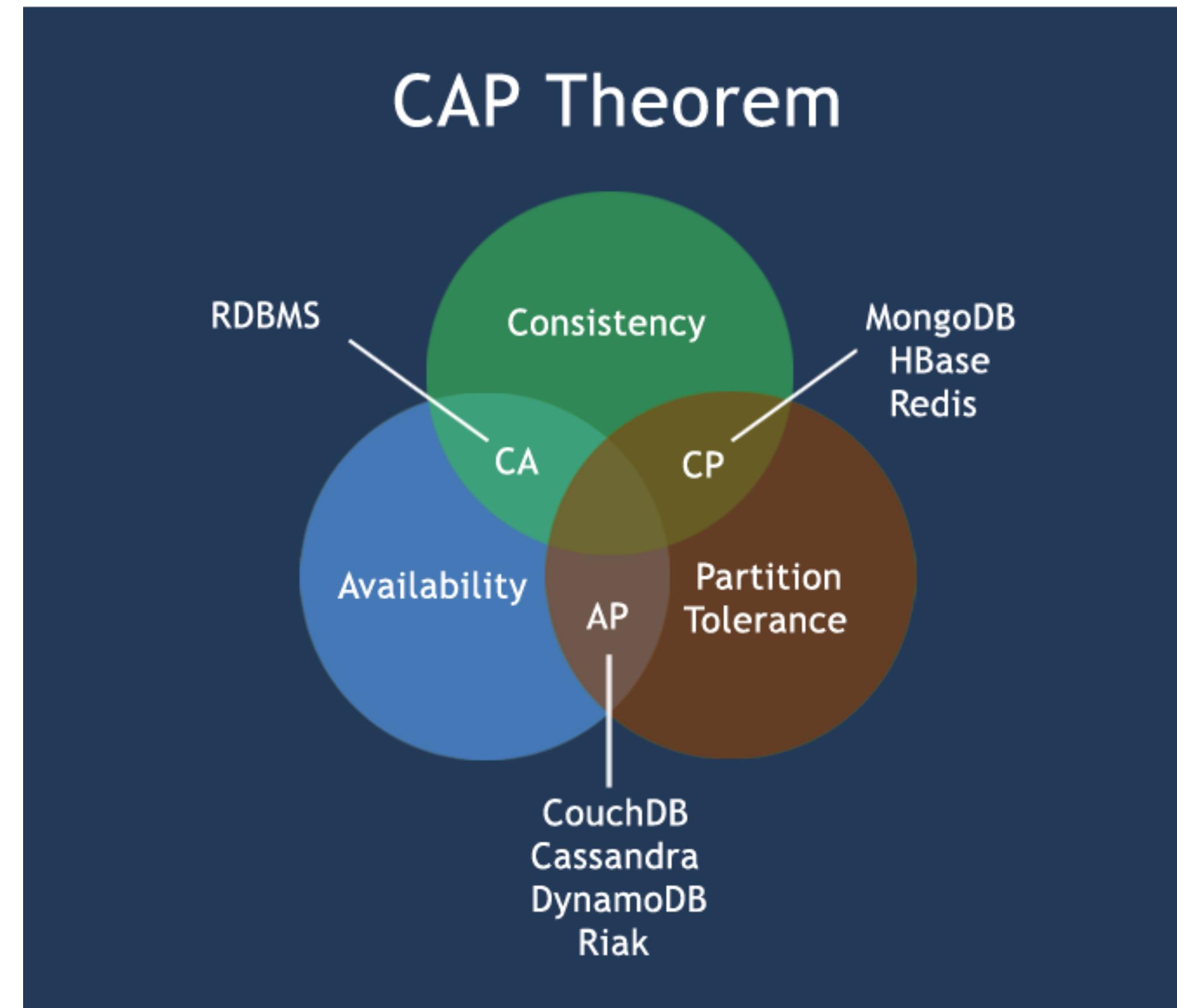
## No Sql No Cry

- NoSQL -> not only SQL (sql ve ötesi)
- 1960'lardan itibaren farklı adlarla kullanıldı. (NoRel = No relation)
- 2 temel katma değer: Yazılım geliştirme üretkenliği (esneklik) ve geniş ölçekli verilerle çalışma kolaylığı (dağıtık çalışma)
- “Strict” bir şekilde şemaya uyma zorunluluğu bulunmaz. (İstisnalar olabilir)
- ACID kriterleri sağlamak gibi bir kaygı bulunmaz.
- Çoğu NoSQL çözümü “eventually consistent” çalışır.
- **Key-Values:** Redis, Dynamo DB
- **Column Family:** Cassandra, HBase
- **Document:** Mongo, Couchbase
- **Graph:** Neo4J, Infinite Graph

# CAP Theorem -1

- **Consistency:** Tüm istemciler aynı datayı aynı şekilde görür. (Bir tutarsızlık olusma ihtimali yoktur.)
- **Availability:** Sistem başarılı veya başarısız mutlaka cevap verir.
- **Partition Tolerance:** Shutdown olan partition' lar olsa bile kalanlar sistemi başarıyla ayakta tutmaya devam eder.
- CAP teoremi bize üçünün aynı anda olamayacağını söyler.
- Dağıtık bir sistem ise zaten “partition tolerance” sağlamak zorunda olduğundan konu o açıdan A veya C'den birini sağlamak üzerindedir.
- Eric Brewer, published in 1998
  - <https://ieeexplore.ieee.org/document/798396>

# CAP Theorem - 2



<https://stackoverflow.com/questions/47453975/as-per-cap-theorem-which-category-does-memcached-fall-into>

# Redis

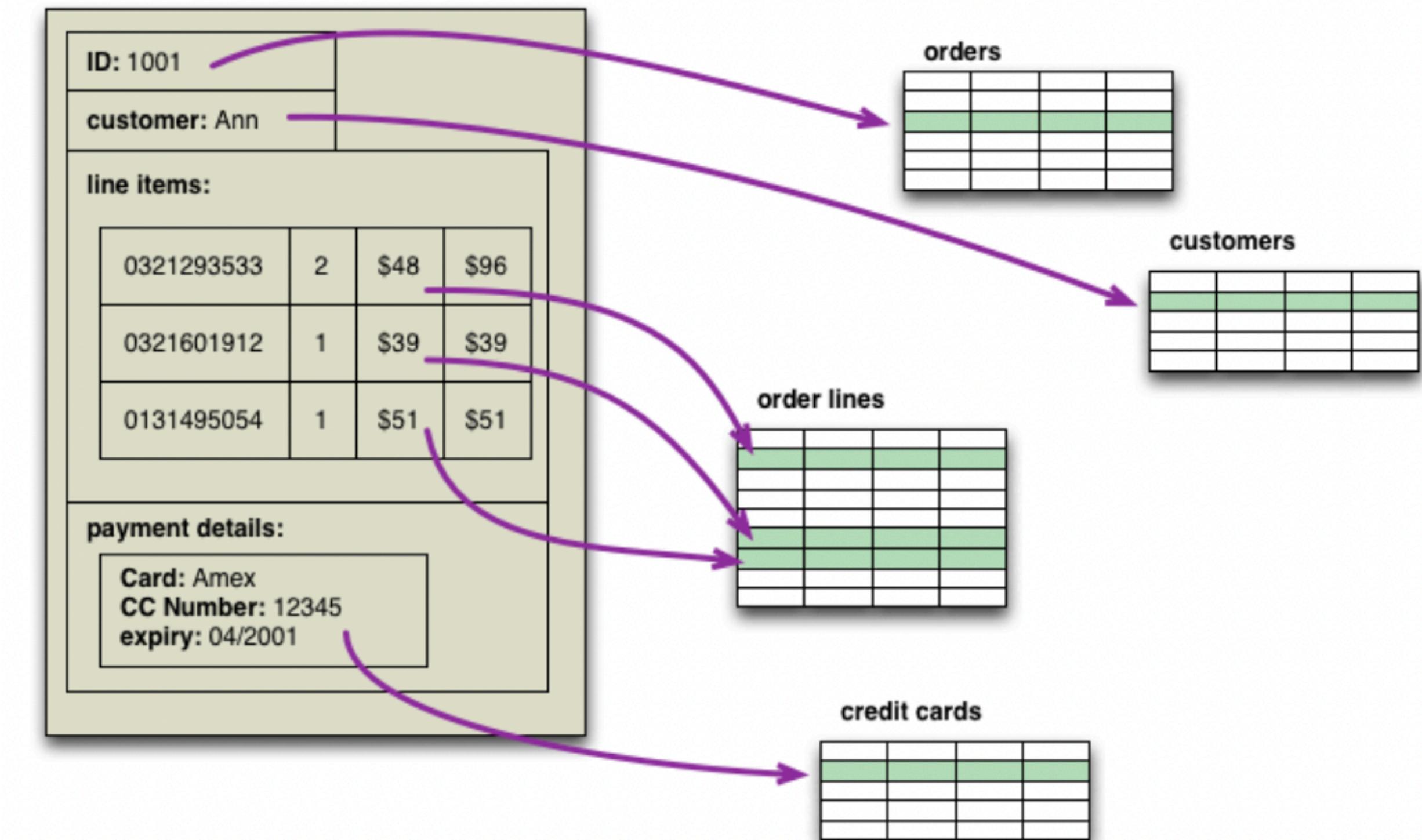
## Key-Value Pairs

- <https://redis.io/>
- Redis - Remote Dictionary Server. Basit ifadeyle bir “hash table” dır. (Key-Value store)
- Value olarak obje, list, set gibi yapılar saklayabiliriz.
- Redis yapısı gereği in-memory çalışır. Persist edilmek istenirse konfigürasyon ile mümkündür.
- Bazı kullanım şekilleri: Caching, Pub/Sub, Streaming, Distributed Locking
- TTL - Time to live. Redis'e eklediğimiz verilen ne kadar süre sonra “expire” olacağını belirleyebiliyoruz. Veya cache çözümü olarak kullanıyorsak veriyi kendimiz “evict” edebiliriz.
- Arayüz aracı: redis-commander
- RedisTemplate ile veri ekleme-çıkarma yapabilir ve / veya @Cachable anotasyonu kullanabiliriz.
  - <https://spring.io/projects/spring-data-redis>
  - <https://www.baeldung.com/spring-data-redis-tutorial>
  - <https://www.baeldung.com/spring-cache-tutorial>

# Aggregate Oriented Database

<https://martinfowler.com/bliki/AggregateOrientedDatabase.html>

- Domain Driven Design'da yer alan Aggregate kavramından esinlenilmiştir.
- İlişkili tüm bilgiler bir aradadır (value) ve bizi o bilgilere götüren unique değerler vardır (key).
- Geliştirici açısından “kod - veri” uyumunu kolay sağladığı için avantajlıdır.
- Key-Value ve Document Oriented tipindeki yapılar bu şekilde çalışır.



# MongoDB

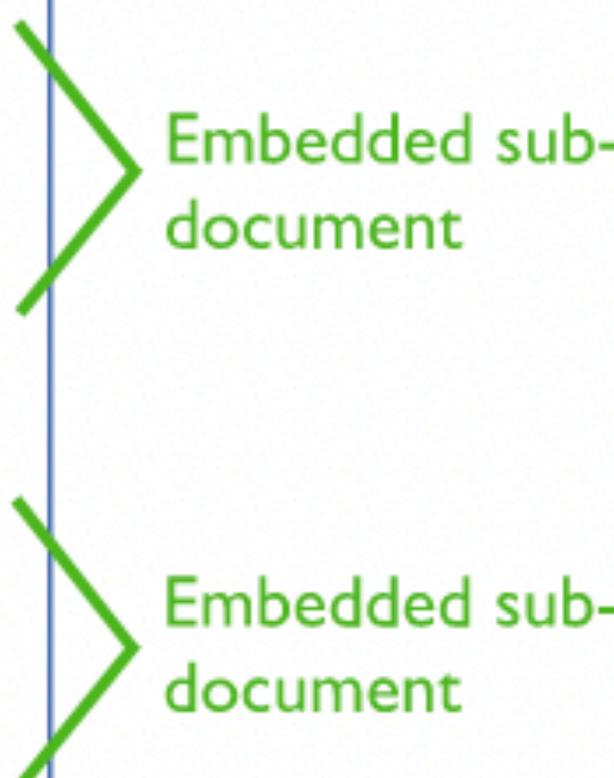
## Document Based

- İsmi “humongous” kelimesinden gelir. (Devasa, çok büyük)
- JSON formatındaki dokümanlarla çalışır. Dokümanlar Binary JSON olarak kaydedilir. (BSON)
- Collection = table ; Document = Row olarak düşünebilir.
  - Verilerin dokümanlardaki saklanma şekli “tree” veri yapısına yakındır.
- Şema değişikliği on-the-fly yapılabilir. (İlişkisel veritabanlarında yeni bir tablo veya sütun istenirse önce create edilmeli)
  - Bu esneklik avantajlı olduğu gibi hatalara karşı dikkatli olmayı da gerektirir.
- Birden fazla doküman üzerinde transactional işlem yapmayı destekler. (Version 4.0+)
  - <https://docs.mongodb.com/manual/core/transactions/>
  - 4.2 ve üzeri distributed transaction yönetimini de destekler.
  - İşlemlerin mümkün olduğunda tek doküman üzerinden halledileceği bir tasarım yapılması öneriliyor.

# Mongo DB

## Embedded Document

```
{  
  _id: <ObjectId1>,  
  username: "123xyz",  
  contact: {  
    phone: "123-456-7890",  
    email: "xyz@example.com"  
  },  
  access: {  
    level: 5,  
    group: "dev"  
  }  
}
```



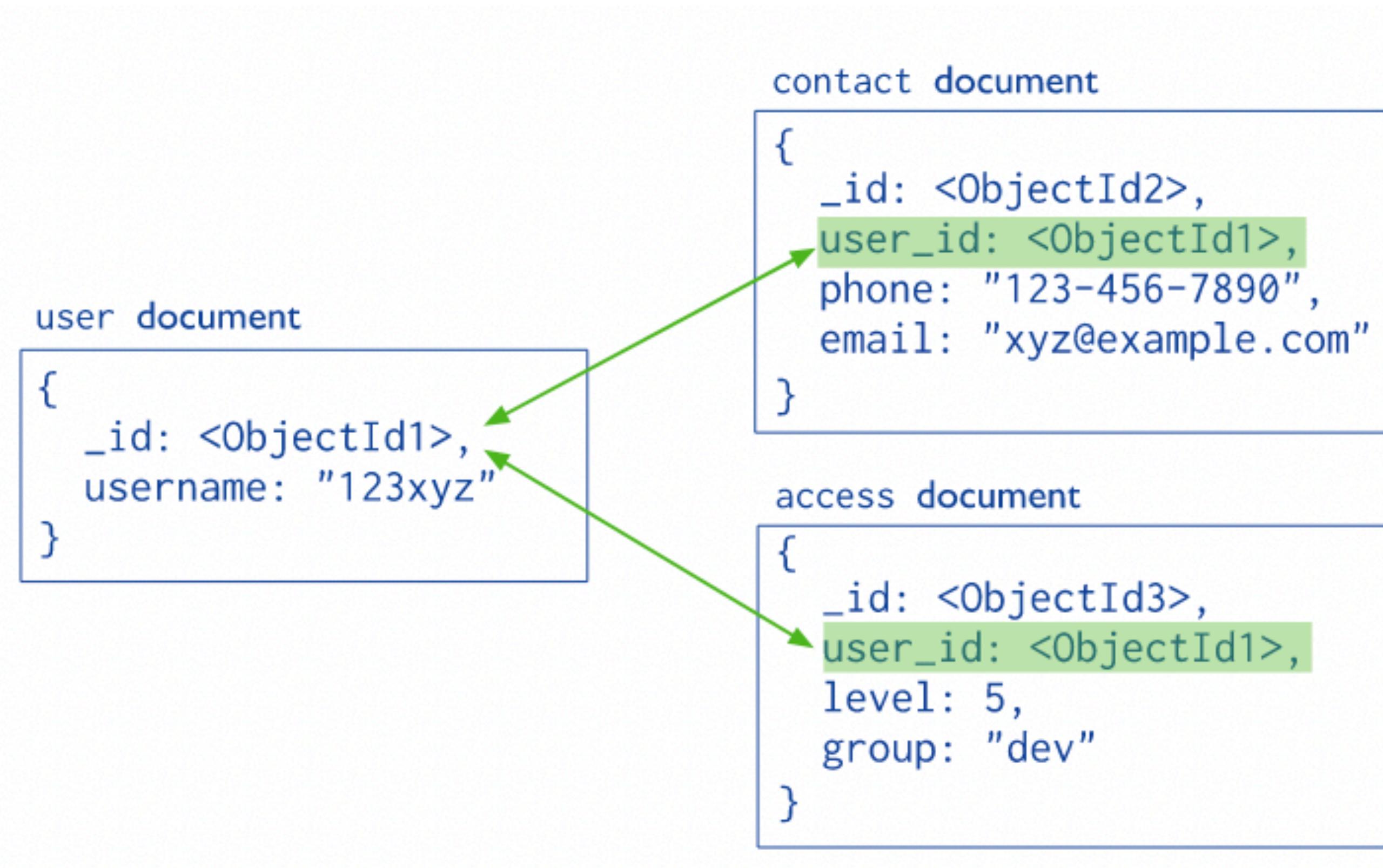
Embedded sub-document

Embedded sub-document

<https://docs.mongodb.com/manual/core/data-model-design/#std-label-data-modeling-embedding>

# Mongo DB

## Normalized Data



<https://docs.mongodb.com/manual/core/data-model-design/#std-label-data-modeling-embedding>

# Polygot Persistence

<https://martinfowler.com/bliki/PolyglotPersistence.html>

- İsmi Neal Ford'un kullandığı “polygot programming” ifadesinden gelir. (2006)
- Tek bir veri tabanı yaklaşımıyla tüm problemleri çözmeye çalışma yerine projenin farklı noktalarında probleme uygun çözümler tercih edebiliriz.
  - Özellikle Mikroservislerin yaygınlaşmasıyla birlikte bu yaklaşımı benimsemek kolaylaşmıştır.
  - *Ödeme işlemi: PostgreSQL*
  - *Faturaların saklanması: Mongo DB*
  - *Sık sorgulanan (nispeten az güncellenen) verilerin sorgulanması: Redis*
  - *Complex search işlemlerinin gerçekleştirilmesi: Elasticsearch*
- Eklenen her yeni çözümün beraberinde bir complexity getireceğini unutmamak gereklidir.
- Eklenen çözümlerin projenin kalanına bir soyutlama ile sunulması faydalı olacaktır. (Bu soyutlama aynı codebase içerisinde olabileceği gibi farklı bir mikroservis üzerinden sunulması şeklinde de olabilir.)

# Veritabanı Gelişmeleri

## Kronolojik Sıradan

- 1970, Edgar F. Codd ilişkisel veri tabanları hakkındaki makalesini yayınladı.
- 1973 IBM laboratuvarında SQL kullanılmaya başlandı.
- 1981, Jim Gray ACID kriterlerinin de temelini oluşturan transaction yönetimi ile ilgili makalesini yayınladı.
- 1982, Postgres, Ingres projesi olarak Berkeley üniversitesinde geliştirilmeye başlandı.
- 1986, SQL ANSI standardı olarak kabul edildi.
- 1987, SQL ISO standardı olarak kabul edildi.
- 1983, A. Rueter ve T. Harder ACID akronimi oluşturuldu.
- 1996, Postgres SQL standartlarına uyumlanarak PostgreSQL' e dönüştü.
- 2000, CAP teoremi Eric Brewer tarafından duyuruldu.
- 2007, 10gen firması tarafından MongoDB yayınlandı.
- 2008, Cassandra Facebook tarafından yayınlandı.
- 2009, Redis open-source bir proje olarak yayınlandı.

# Ek Kaynaklar

## SQL ve NoSQL çözümler

- <https://martinfowler.com/data/>
- <https://www.youtube.com/watch?v=qLgo7CQ5I> (Martin Fowler)
- <https://www.youtube.com/watch?v=AbAt7wngN2s> (Kenan Sevindik)
- <https://www.youtube.com/watch?v=pHKZw5EyTi4&list=PLh9ECzBB8tJOS7WQKdeUaAa5fmPLYAouD> (Sadi Evren Şeker - daha akademik formatta)
- <https://stackoverflow.com/questions/1108/how-does-database-indexing-work>
- <https://vladmihalcea.com/optimistic-vs-pessimistic-locking/>

# 3. Bölüm Sonu

## 3. Ödev / Araştırma (Son Tarih: 16 Ocak Pazar 09:59)

1. Imperative Programming ve Declarative Programming kavramlarını kısaca açıklayıp farklarını belirtiniz.
2. Veri tabanlarının sorgu optimizasyonlarında index oluşturmanın avantajı nedir ? Sık index kullanmak bir probleme yol açar mı?
3. İlişkisel veritabanları için normalizasyon kavramı neyi ifade etmektedir ? İlk 3 normal formu örnek üzerinden açıklayınız.
4. ORM kütüphaneleri kullanmak her zaman avantajlı mıdır ? ORM kütüphanelerinin ne gibi dezavantajları olabilir ?
5. Domain Specific Language (DSL) kavramını açıklayınız.
6. Long lived transaction kavramı hangi tip transactionları ifade etmektedir ? Dezavantajları var mıdır ? Varsa nelerdir ?
7. Thread Pool nedir ? Nerelerde kullanılır ?
8. Scalability nedir ? Horizontal ve Vertical Scalability kavramlarını açıklayınız.
9. Data replication ve sharding nedir ? Aralarında nasıl bir fark bulunmaktadır ?

# 3. Bölüm Sonu

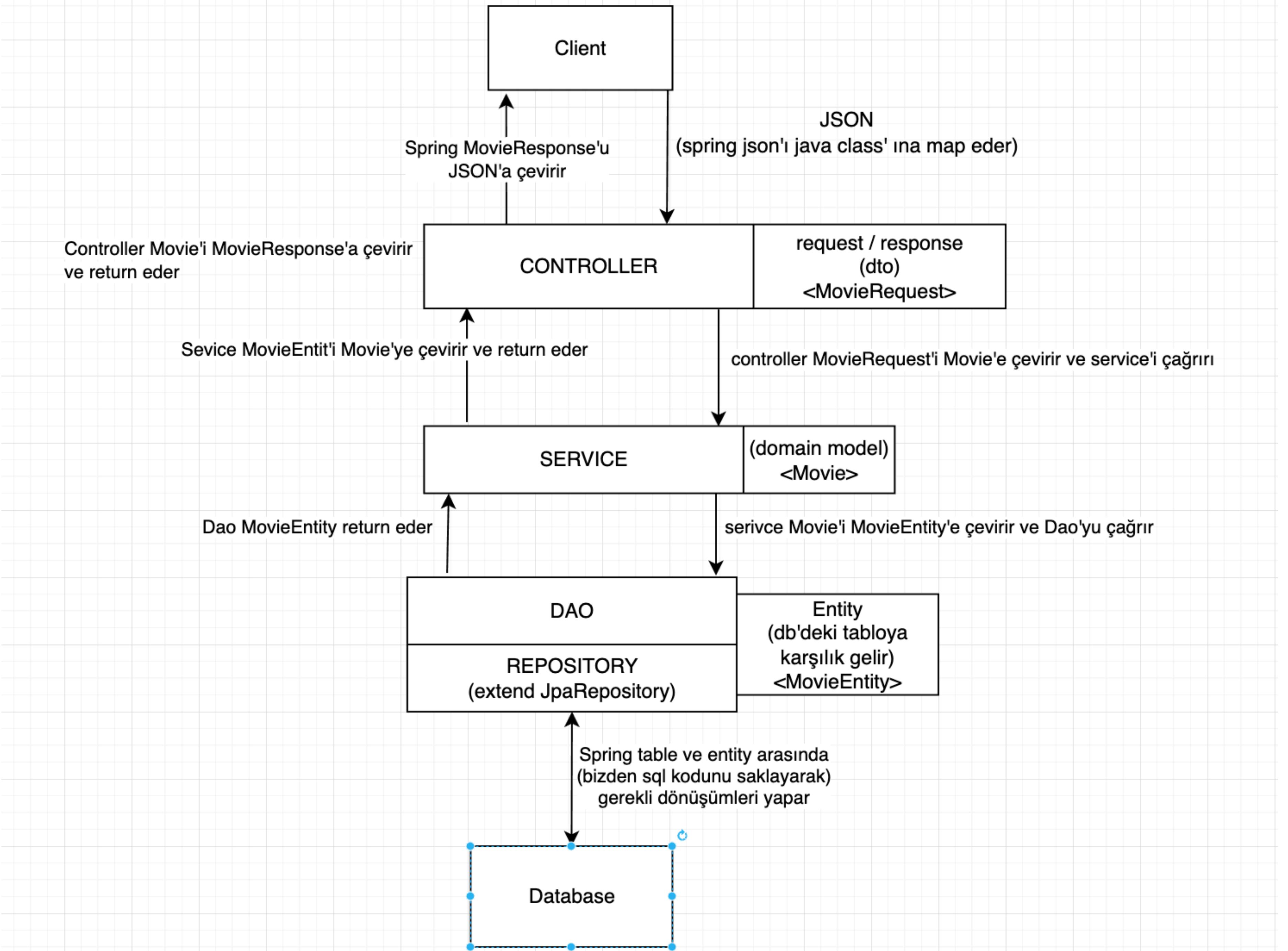
## 3. Ödev / Kodlama (Son Tarih: 16 Ocak Pazar 09:59)

- 2. Ödevde yer alan endpoint'lerin Spring Data ile veritabanı entegrasyonu yapılacaktır. Katmanlı mimari yaklaşımına uygun olarak (controller-service-dao) geliştirilmesi beklenmektedir.
  - Postgresql veya MySql tercih edebilirsiniz.
  - (Optional) En az bir noktaya Redis entegrasyonu ekleyerek datanın cache' den getirilmesini sağlanacaktır.

# **4. Bölüm**

# **Layered Architecture**

## **Katmanlı Mimari**



# **Hexagonal Architecture**

## **Ports & Adapters**

# **Test Pyramid**

# Kitap Tavsiyeleri

