**1 – IOC and DI means ?**

**IOC:**

Nesneler arasındaki bağımlılıkların kontrolünü uygulama tarafından yapmak yerine, kontrolün uygulama çatısına yaptırmaktır. Kontrolü tersine çevirerek kontrolü uygulama yerine uygulama çatısına yaptırılır.

Uygulamanın yaşam d**ön**güsü boyunca birbirine bağımlılığı az nesneler oluşturmayı amaçlayan bir yazılım geliştirme prensibidir. Nesnelerin yaşam döngüsünden sorumludur.

IoC kullanmanın avantajlarını şöyle sıralayabiliriz:

* Loosely coupled (bağımlılığı az) sınıflar oluşturma
* Kolay unit test yazma
* Yönetilebilirlik
* Modüler programlar
* Farklı implementasyonlar arası kolay geçiş

**Dependency Injection (DI)**

IoC’nin implement edilmesini sağlayan bir pattern’dır. İhtiyaç duyulan bağımlılıkların giderilmesini sağlar. Koddaki injeck ettiğimiz bağımlılıkları sınıf içerisine inject etmemizi sağlar. Bunun 3 yolu var.

1. Constructor ile Dependency Injection: Constructor üzerinden inject etme işlemi
2. Setter ile Dependency Injection: : Setter üzerinden inject etme işlemi
3. Field ile Dependency Injection: : Field üzerinden inject etme işlemi

**2- Spring Bean Scopes ?**

Kullanılan Bean’in yaşam döngüsünü ve oluşumunu tanımlar. Spring framework çatısında 6 adet bean scope bulunmaktadır. Spring Beanlerimizdeki scopeleri Spring IoC container tarafından yönetilir ve beanlerimizdeki nesnelerin ne zaman ve nasıl oluşturulacağını belirler.

* **Singleton**: Tekil nesneler oluşturulması için kullanılmaktadır. Spring’ de varsayılan kapsam(scope) tanımlamasıdır.
* **Prototype**: Her nesne oluşturma talebi için yeni bir nesne oluşturulmasını sağlayan tanımlamadır.
* **Request**: Web uygulamalarında istek(http request) geçerli olduğu sürece geçerli olacak bir nesne oluşturulabilmesi için kullanılır. Web ile uyumlu bir applicationContext.xml dosyası oluşturulduğunda kullanılabilir.
* **Session**: Web uygulamalarında oturum(http session) geçerli olduğu sürece geçerli olacak bir nesne oluşturulabilmesi için kullanılır. Web ile uyumlu bir applicationContext.xml dosyası oluşturulduğunda kullanılabilir.
* **Application**: Bir application scope, ServletContext’in yaşam döngüsü için bean örneğini oluşturur.
* **WebSocket:** WebSocket scope’a sahip bean’ler tipik olarak singleton scope yapısındadır ve herhangi bir WebSocket oturumundan daha uzun yaşar. Bu nedenle, WebSocket scope’una sahip bean’ler için bir proxyMode tanımı gerekir.

Kullanımı;

Anotasyon olarak @Scope” …”) olarak Bean’in en başına konulur.

@Scope("request")

@Scope("singleton ")

**3- What does @SpringBootApplication do ?**

Uygulamanın giriş metodunu belirler. Bir nevi main metodudur. Bu anotasyon ile birlikte uygulama bu metod ile başlar.

Bu anotasyon @Configuration,  @EnbaleAutoConfiguration, @ComponentScan  anotasyonlarının üçünü de içeren temel bir anotasyondur.

@Configuration : Java tabanlı konfigürasyon işlemi yapan bir anotasyondur

@ComponentScan : Projeye dahil edilen komponenetlerin otomatik taranmasını sağlar.

@EnableAutoConfiguration : Varsayılan konfigürasyonların otomatik gerçekleşmesini sağlar.

**4- Why Spring Boot over Spring?**

Prensip olarak aynı şeyler olsa da farklılıkları mevcuttur.

* Spring boot, spring çatısının daha kolaylaştırılmış halidir.
* Spring ile çalışılacaksa bir web sunucu üzerinde çalışılması gerekmektedir, spring boot’ta ise hazırda içine eklenmiş sunucu ile gelmektedir.
* Spring boot starter paketleriyle beraber proje tipine göre hazır kütüphaneler eklenmiş olarak gelirken, spring ile çalışırken core, security, web, data taraflarını kendimiz eklememiz gerekmektedir.
* Bağımsız (stand-alone) uygulamalar oluşturur.
* Gömülü olarak Tomcat, Jetty veya Undertow birlikte gelir.
* XML konfigürasyonuna ihtiyaç duymaz.
* LOC (Lines of Code) ‘u azaltmayı hedefler.
* Başlatması kolaydır.
* Özelleştirme ve yönetim basittir.
* Spring Boot bir kütüphane olmayıp, Spring tabanlı hazır bir proje başlatıcıdır. Otomatik yapılandırma gibi özelliklerle sizi uzun kod yazmaktan kurtarır ve gereksiz yapılandırmalardan kurtulmanızı sağlar.

**5- What is Singleton and where to use it ?**

Genel olarak bir sınıftan kaç tane obje üretilidğini kontrol etmek istendiğinde singleton yapısı kullanılmış olur.Tek bir nesneye ihtiyaç duyulan durumlarda (veri tabanı bağlantıları, port bağlantıları, dosya işlemleri, loglama işlemleri,bildirimlerde, iş katmanı servislerimizde) kullanılır.

Singleton Pattern İlkeleri

* Singleton pattern bir sınıftan birden fazla instance oluşmasını engeller ve sanal makinada oluşturulan tek instance’ın kullanılmasını garanti eder.
* Singleton sınıfı, sınıfa ait instance’a ulaşmak için global erişim sağlamalıdır.
* Singleton sınıfının constructor’ı private olmalıdır. Böylece o sınınftan yeni instance oluşturmak engellenmiş olur.
* Singleton sınıfın instance’ı içeride **private static** olarak tutulmalıdır.
* Tutulan instance’ı döndüren bir **public static** metoda sahip olmalıdır.

**6- Explain @RestController annotation in Sprint boot?**

Bu anotasyon ilgili sınıftaki bütün metodların birer REST servis noktası olmasını sağlar. Ayrıca bu anotasyon @Controller ve @ResponseBody anotasyonlarının bileşimi niteliğindedir.

**7- What is the primary difference between Spring and Spring Boot ?**

Spring boot, spring çatısının daha kolaylaştırılmış halidir.

**8 – Why to use VCS ?**

* Versiyon kontrol sistemi, değişiklikleri takip etmek ve her ekip üyesinin en son sürümle çalışmasını sağlamak için önemlidir.
* Birden çok ekip üyesinin üzerinde işbirliği yapacağı tüm kodlar, dosyalar ve varlıklar için sürüm kontrol yazılımı kullanmak gerekir.
* Her dosyanın tam bir uzun vadeli değişiklik geçmişi tutulur.  Buda, dosya üzerinde yıllar içinde birçok kişi tarafından yapılan her değişikliğin tutulması anlamına gelir.
* Versiyon kontrol sistemi ürünleri daha hızlı geliştirmenize yardımcı olur.
* Görünürlüğü arttırır.
* Ekiplerin dünya çapında işbirliği yapmasına yardımcı olur.
* Ürün geliştirme sürecini hızlandırır.

**9 – What are SOLID Principles ? Give sample usages in Java ?**

Geliştirilen yazılımın esnek, yeniden kullanılabilir, sürdürülebilir ve anlaşılır olmasını sağlayan, kod tekrarını önleyen ve Robert C. Martin tarafından öne sürülen prensipler bütünüdür.

Amaç;

* Geliştirdiğimiz yazılımın gelecekte gereksinimlere kolayca adapte olması,
* Yeni özellikleri kodda bir değişikliğe gerek kalmadan kolayca ekleyebileceğimiz
* Yeni gereksinimlere karşın kodun üzerinde en az değişimi sağlaması,
* Kod üzerinde sürekli düzeltme hatta yeniden yazma gibi sorunların yol açtığı zaman kaybını da minimuma indirmektir.

[Single-responsibility principle](https://gokhana.medium.com/single-responsibility-prensibi-nedir-kod-%C3%B6rne%C4%9Fiyle-soli%CC%87d-c8b1602be602)

Bir sınıf (nesne) yalnızca bir amaç uğruna değiştirilebilir, o da o sınıfa yüklenen sorumluluktur, yani bir sınıfın(fonksiyona da indirgenebilir) yapması gereken yalnızca bir işi olması gerekir.

[Open-closed principle](https://gokhana.medium.com/open-closed-prensibi-nedir-kod-%C3%B6rne%C4%9Fiyle-soli%CC%87d-679619d5376a)

Bir sınıf ya da fonksiyon hali hazırda var olan özellikleri korumalı ve değişikliğe izin vermemelidir. Yani davranışını değiştirmiyor olmalı ve yeni özellikler kazanabiliyor olmalıdır.

[Liskov substitution principle](https://gokhana.medium.com/liskov-substitution-prensibi-nedir-kod-%C3%B6rne%C4%9Fiyle-soli%CC%87d-3cfc1cd63c1a)

Kodlarımızda herhangi bir değişiklik yapmaya gerek duymadan alt sınıfları, türedikleri(üst) sınıfların yerine kullanabilmeliyiz.

[Interface segregation principle](https://gokhana.medium.com/interface-segregation-prensibi-nedir-kod-%C3%B6rne%C4%9Fiyle-soli%CC%87d-ac0fd6812ecf)

Sorumlulukların hepsini tek bir arayüze toplamak yerine daha özelleştirilmiş birden fazla arayüz oluşturmalıyız.

[Dependency Inversion Principle](https://gokhana.medium.com/dependency-inversion-prensibi-nedir-kod-%C3%B6rne%C4%9Fiyle-soli%CC%87d-b61296523565)

Sınıflar arası bağımlılıklar olabildiğince az olmalıdır özellikle üst seviye sınıflar alt seviye sınıflara bağımlı olmamalıdır.

**10 - What is RAD model ?**

RAD Modeli veya Hızlı Uygulama Geliştirme modeli, herhangi bir özel planlama olmaksızın prototiplemeye dayalı bir yazılım geliştirme sürecidir. RAD modelinde, planlamaya daha az dikkat edilir ve geliştirme görevlerine daha fazla öncelik verilir. Kısa sürede yazılım geliştirmeyi hedefler.

Aşamaları ;

* İş modeli
* Veri Modelleme
* Süreç Modelleme
* Uygulama Üretimi
* Test ve Ciro

İş modeli: Ürün, çeşitli iş kanalları arasındaki bilgi akışı ve dağıtım temelinde tasarlanır.

Veri Modelleme: İş modellemesinden toplanan bilgiler, işletme için önemli olan bir dizi veri nesnesi olarak rafine edilir.

Süreç Modelleme: Veri modelleme aşamasında bildirilen veri nesnesi, bir iş fonksiyonunu uygulamak için gerekli bilgi akışını sağlamak için dönüştürülür.

Uygulama Üretimi: Yazılımın oluşturulması, süreç ve veri modellerinin prototiplere dönüştürülmesi için otomatik araçlar kullanılır.

Test ve Ciro: Prototipler her yineleme sırasında ayrı ayrı test edildiğinden, genel test süresi RAD'de kısalır.

**11 - What is Spring Boot starter ? How is it useful ?**

Uygulamanıza ekleyebileceğiniz bir dizi bağımlılık tanımlayıcısıdır. Sizi kullanmak istediğiniz teknolojilerin her biri için arama yapıp teker teker bağımlılık olarak ekleme zahmetinden kurtarır. Starterlar sayesinde ihtiyacınız olan Spring ve ilgili teknolojileri kolayca uygulamanıza ekleyebilirsiniz. Örnek olarak Spring ve JPA kullanmak istiyorsanız spring-boot-starter-data-jpa bağımlılığını projenize eklemeniz yeterli olacaktır.

**12– What are the Spring Boot Annotations?**

Developlement anında IDE veya compiler tarafından ya da run-time anında framework tarafından yorumlanan ifadelerdir. Bir öğenin tanımını yapar, ne yapması gerektiğini açıklar ve yazılım geliştirme sürecini hem hızlandırır hem de kolaylaştırır.

Her durumda bir öğeyi tanımlamak ve anlamını netleştirmek için kullanılan notasyon türleri vardır. Burada öğe olarak bahsedilen şey bir değişken olabilir, bir fonksiyon veya bir sınıf olabilir.

**Temel Spring Boot Anotasyonları**

@Bean - Bir metodun Spring tarafından yönetilen bir Bean ürettiğini belirtir

@Service - Belirtilen sınıfın bir servis sınıfı olduğunu belirtir.

@Repository - Veritabanı işlemlerini gerçekleştirme yeteneği olan yapıldığı repository sınıfını belirtir.

@Controller - Requestleri yakalayabilme yeteneği olan bir web controller sınıfını belirtir.

@RequestMapping - controller sınıfının handle ettiği HTTP Requestlerin path eşleştirmesini yapar

@Autowired - Constructor, Değişken ya da setter metodlar için dependency injection işlemi gerçekleştirir

@SpringBootApplication - Spring Boot autoconfiguration ve component taramasını aktif eder.

**13 – What is Spring Boot dependency management?**

Uygulama üzerindeki bağımlılıkları pom.xml dosyası üzernden takip etme ve buradan yönetme üzerine kurulu bir yapıdır. Tüm bağımlılıklar buraya eklenerek maven üzerinden projeye yüklenir.

**14 -  What is Spring Boot Actuator?**

Spring boot actuator'ı projemize bir dependency olarak ekleyerek, projemizin çalışma durumu(ayakta/ayakta değil), trafik durumu, son yüz http isteği, veritabanımızın durumu vs gibi bilgilere çok kolay bir şekilde endpointler üzerinden erişebiliriz.

Spring Boot uygulama linkimizin sonuna actuator/beans , actuator/health gibi parametreler yazarak endpointlere ulaşıp bilgi alabiliriz

Uygulamanın ayakta olup olmadığını kontrol etmek istenirse;

Request: ...uygulama linki).../actuator/health

Response:  "status" : "UP" olarak alınırsa uygulama ayakta demektir.

Actuator için maven projemize eklememiz gereken dependency şudur:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>