1 – IOC and DI means?

IoC (Inversion Of Control), uygulamanın yaşam döngüsü boyunca birbirine bağımlılığı az (loose coupling) olan nesneler oluşturmayı amaçlayan bir yazılım geliştirme prensibidir. Nesnelerin yaşam döngüsünden sorumludur, yönetimini sağlar. IoC kullanan sınıfa bir interface implement edildiğinde, ilgili interface metotları kullanılabilir olur. Böylece IoC kullanan sınıf sadece kullanacağı metotları bilir, sınıf içerisinde daha fazla metot olsa bile interface'de belirtilen metotlara erişebilecektir.

Sınıf içerisinde yapılacak herhangi bir değişiklikte IoC kullanan sınıf etkilenmeyeceği için yeniden yazılabilir ve test edilebilir yazılım geliştirmemizi sağlar. IoC nesne bağlamalar genellikle uygulama başlangıcında yapılandırılmaktadır. Bu anlamda tek bir yerden yapılan IoC yapılandırmalarının değiştirilmesi ve yönetimi de oldukça kolaydır.

IoC kullanmanın avantajlarını şöyle sıralayabiliriz:

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

DI (Dependency Injection) bağımlılık oluşturacak parçaların ayrılıp, bunların dışardan verilmesiyle sistem içerisindeki bağımlılığı minimize etme işlemidir. Dependency Injection, bağımlılıkları soyutlamak demektir. Yani, temel olarak oluşturulacak bir sınıf içerisinde başka bir sınıfın nesnesi kullanılacaksa new anahtar sözcüğüyle oluşturulmaması gerektiğini söyleyen bir yaklaşımdır. Gereken nesnenin ya Constructor'dan ya da Setter metoduyla parametre olarak alınması gerektiğini vurgulamaktadır. Böylece iki sınıfın birbirinden izole etmiş olduğunu savunmaktadır.

Constructor base ile bağımlılıkları kaldırmak için, birbiriyle benzer metotlara sahip ama bu metotları farklı kullanan sınıflar için ortak bir interface oluşturulur. Bu interface içerisinde bu benzer metotları barındıracaktır. Bu sınıflardan biri oluşturulmaya kalkıldığında interface üzerinden referans verilerek, kodda daha sonra yeni bir sınıf geldiğinde değişiklik yapmadan, dilenen sınıf nesnesi constructor parametresi ile oluşturularak kullanılabilir.

2 – Spring Bean Scopes?

Bir bean'in context'i, kullandığımız bağlamlarda o bean'in yaşam döngüsünü ve oluşumunu tanımlar. Spring framework çatısında 6 adet bean scope bulunmaktadır:

- Singleton
- Prototype

Yalnızca web uyumlu uygulamalar ile kullanılabilen scope'lar ise:

- Request
- Session
- Application
- WebSocket

Bu scope'ları biraz inceleyelim…

a. **Singleton Scope**
   Bir bean varsayılan olarak Singleton Scope'a sahiptir. Bean Singleton Scope ile tanımlandığı zaman mevcut Application Context içerisinde o bean'den yalnızca ve yalnızca tek bir adet initialize edileceği garanti edilir. Bu bean ile yapılacak olan tüm request'ler cache'lenmiş olan aynı nesne üzerinden yapılır. Bean nesnesi üzerinde yapılan bir değişiklik bean'i kullanan tüm yerlerde etkilenecektir.
b. **Prototype Scope**
   Prototype ile belirlenmiş bir bean, container içerisinde çağırıldığı her request'te yeniden oluşturulacaktır. Her nesne oluşturma talebi için yeni bir nesne oluşturulmasını sağlayan tanımlamadır.
c. **Request Scope**
   Web uygulamalarında istek(http request) geçerli olduğu sürece geçerli olacak bir nesne oluşturulabilmesi için kullanılır. Web ile uyumlu bir applicationContext.xml dosyası oluşturulduğunda kullanılabilir.
d. **Session Scope**
   Web uygulamalarında oturum(http session) geçerli olduğu sürece geçerli olacak bir nesne oluşturulabilmesi için kullanılır. Web ile uyumlu bir applicationContext.xml dosyası oluşturulduğunda kullanılabilir.
e. **Application Scope**
   Bu Singleton Scope'a benzer ancak aralarında farklılıklar mevcuttur. Bir bean Application Scope değerine sahipken bu bean çoklu servlet tabanlı uygulamalar ile de paylaşılabilirken, Singleton Scope değerine sahip bir bean yalnızca mevcut Aapplication Context'i içerisinde tanımlıdır.
f. **WebSocket Scope**
   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. Singleton davranış sergilediğini, ancak yalnızca bir WebSocket sesion'ı ile sınırlı olduğunu da söyleyebiliriz.

3 – What does @SpringBootApplication do?

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

- @Configuration: Java tabanlı konfigürasyon işlemi yapan bir anotasyondur.
- @EnbaleAutoConfiguration: Varsayılan konfigürasyonların otomatik gerçekleşmesini sağlar.
- @ComponentScan: Projeye dahil edilen bileşenlerin otomatik taranmasını sağlar.

4 – Why Spring Boot over Spring?

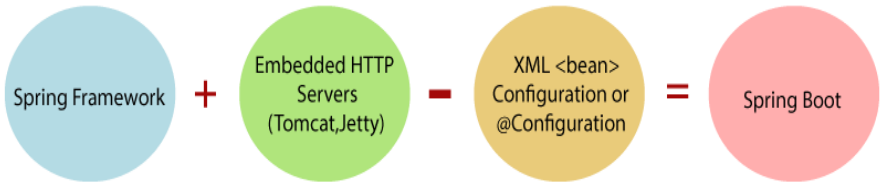
Spring kütüphanesi bize esneklik uygulamaya odaklanırken, Spring Boot kod uzunluğunu kısaltmayı ve bir web uygulaması geliştirmenin en kolay yolunu bize sunmayı amaçlamaktadır. Spring Boot, uygulama geliştirme için gerekli olan süreyi bir hayli kısaltır. Neredeyse hiçbir konfigürasyon yapmadan tek başına bir uygulama oluşturulmasına yardımcı olur.

Oto konfigürasyon Spring Boot için özel bir özelliktir.

a. Spring kütüphanesinin yararları
   - Spring kütüphanesi bir web uygulamasının tüm katmanlarına uygulanabilir.
   - Gevşek bağlılık (Loose Coupling) ve kolay test edilebilirlik sağlar.
   - XML ve Annotation konfigürasyonlarını destekler.
   - Singleton ve Factory sınıflarının ortadan kaldırılması için gerekli yeteneğe sahiptir.
   - Bildirimsel (Declarative) programlamayı destekler.

b. Spring Boot'un yararları
   - 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.

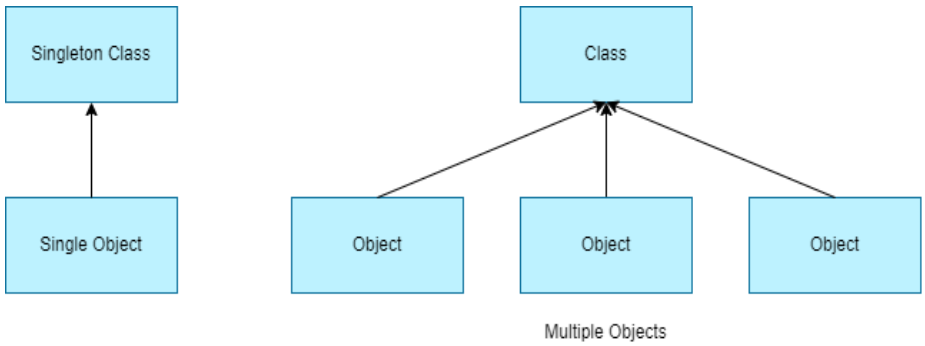
Bu nedenle Spring Boot bir kütüphane olmayıp, Spring tabanlı hazır bir proje başlatıcıdır. Otomatik yapılandırma gibi özelliklerle geliştiricileri uzun kod yazmaktan kurtarır ve gereksiz yapılandırmalardan kurtulmayı sağlar.



5 – What is Singleton and where to use it?

Singleton tasarım kalıbı, bir sınıfın tek bir örneğini almak için kullanılır. Amaç oluşturulan nesneye global erişim noktası sağlamaktır. Sistem çalıştığı sürece ikinci bir örnek oluşturulmaz, böylelikle istenen nesnenin tek bir defa oluşturulması garanti altına alınacaktır. Singleton nesneler ilk çağırıldıklarında bir kere oluşturulurlar ve sonraki istekler bu nesne üzerinden karşılanır.

Singleton tasarım kalıbı, "Creational" tasarım kalıbı kategorisine girer. Bu kategori bir ya da daha fazla nesnenin oluşturulması ve yönetilmesi ile ilgilidir.



6 – Explain @RestController annotation in Sprint boot?

Bu anotasyonun anlanması için öncelikle Controller anotasyonu bilinmelidir.

a. **@Controller**
   - Sınıfın, dışardan gelen request'leri yakalaması gereken bir sınıf olduğu belirtilir.
   - Anotasyon taramaları esnasında @Controller ve onun altındaki @RequestMapping tanımları taranır. Bu sebeple @RequestMapping ifadesini yalnızca @Controller tanımlı sınıflarda kullanılabilir.
   - @Controller sınıf seviyesinde bir anotasyonken, @RequestMapping fonksiyon seviyesinde bir anotasyondur.

b. **@RestController**
   - @RestController tanımı @Controller tanımının özelleşmiş halidir.
   - @ResponseBody varsayılan olarak tanımlanır.
   - @RestController tanımlaması olan sınıflar için ekstradan @ResponseBody tanımlaması yapmaya gerek yoktur.
   - Rest işlemlerinde @Controller tanımlaması yapılan sınıflarda @ResponseBody eklemek gerekir.
   - Controller Spring MVC için bir view döndürürken, @RestController bir view döndürmez.

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

Spring Boot, Spring Framework'ün üzerine inşa edilmiş bir projedir. Hem basit hem de web tabanlı uygulamaları kurmak, yapılandırmak ve çalıştırmak için daha kolay ve hızlı bir yol sağlar.

Spring Framework'e RAD (Hızlı Uygulama Geliştirme) özelliğini sağlayan bir Spring modülüdür. Minimum Spring konfigürasyonuna ihtiyaç duyduğu için bağımsız bir Spring tabanlı uygulama oluşturmak için kullanılır.

8 – Why to use VCS?

Versiyon Kontrol Sistemi (VCS), revizyon kontrol (revision control) veya kaynak kontrol (source control) diye de geçip, değişiklik yönetim sistemi anlamına gelmektedir. Bir ya da daha fazla dosya üzerinde yapılan değişiklikleri kaydeden ve daha sonra belirli bir sürüme geri dönebilmenizi sağlayan bir sistemdir.

Bir dosyanın değişik sürümlerini korumak istiyorsanız, Sürüm Kontrol Sistemi (VCS) kullanmanız çok akıllıca olacaktır. VCS, dosyaların ya da bütün projenin geçmişteki belirli bir sürümüne erişmenizi, zaman içinde yapılan değişiklikleri karşılaştırmanızı, soruna neden olan şeyde en son kimin değişiklik yaptığını, belirli bir hatayı kimin, ne zaman sisteme dahil ettiğini ve daha başka pek çok şeyi görebilmenizi sağlar. Öte yandan, VCS kullanmak, bir hata yaptığınızda ya da bazı dosyaları yanlışlıkla sildiğinizde durumu kolayca telâfi etmenize yardımcı olur. Üstelik, bütün bunlar size kayda değer bir ek yük de getirmez.

Amaçları:

- Geliştiricilerin, kod değişikliklerini takip etmelerini sağlamak.
- Geliştiricilerin, kod değişiklik geçmişini görmelerini sağlamak.
- Geliştiricilerin, aynı kod dosyalarında aynı anda çalışmasına izin vermek.
- Geliştiricilerin, kodlarını dallanma yoluyla ayırmalarına izin vermek.
- Farklı dallardan (branch) kodu birleştirmek.
- Geliştiricilerin, çakışmalarını gösterir ve bunları çözmelerine izin vermek.
- Geliştiricilerin, değişikliklerini önceki bir duruma döndürmelerine izin vermek.

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

SOLID yazılım prensipleri; geliştirilen yazılımın esnek, yeniden kullanılabilir, sürdürülebilir ve anlaşılır olmasını sağlayan, kod tekrarını önleyen ve Robert C. Martin tarafından öne sürülen prensipler bütünüdür.

Amaçları:

- Geliştirdiğimiz yazılımın gelecekte gereksinimlere kolayca adapte olmasını sağlamak.
- Yeni özellikleri kodda bir değişikliğe gerek kalmadan kolayca ekleyebilmek.
- Yeni gereksinimlere karşın kodun üzerinde en az değişimi sağlamak.
- Kod üzerinde sürekli düzeltme hatta yeniden yazma gibi sorunların yol açtığı zaman kaybını da minimuma indirmek.

S — Single-responsibility principle

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.

O — Open-closed principle

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

L — Liskov substitution principle

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

I — Interface segregation principle

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

D — Dependency Inversion Principle

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

10 - What is RAD model?

RAD (Rapid Application Development) yani Hızlı Uygulama Geliştirme bir yazılım geliştirme yöntemidir. Çok fazla detaya girilmeden, hızlı şekilde çalışan bir uygulama oluşturma amacıyla benimsenen bu yöntem için kullanılan birçok araç ve kütüphane bulunmaktadır. Tabii ki olay sadece kullanılacak araç veya kütüphanede bitmiyor, ekibin bu yöntemi bilmesi, kullanılacak araçları yakından tanıması, hızlı çalışırken az hata yapması, hızlı geliştirmenin getirebileceği olumsuz durumlar (performans, güvenlik gibi) için dikkatli davranılması… gibi konulara hususlarında yine çok detaya girilmeden önemle ele alınması gerekir.

RAD sürecinde geliştirilen yazılımın kalitesini arttırmak için alınabilecek bazı önlemler:

- Etkili Hata İzleme
- Alternatif Stabil Versiyonlar Belirlemek
- Riskli, Küçük Hatalar ve Stabil İşleri Birbirinden Ayırmak
- Her Önemli Modül için Farklı Dal (Branch) Oluşturmak
- Çıkacak Versiyon için Bir Yol Haritası Belirlemek
- Birkaç Müşteri için Özel Versiyon Çıkarmak
- Regresyon Testi
- Duraklat & Gözden Geçir
- Yapılan İşlerin Farklı Geliştiriciler Arasında Kaydırılması
- Tasarımın Baştan Olabildiğince İyi Yapılması

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

Starter'lar kısaca uygulamanıza ekleyebileceğiniz bir dizi bağımlılık tanımlayıcısıdır. Sizi kullanmak istediğiniz teknolojilerin her biri için arama yapıp teker teker bağımlılık olarak ekleme zahmetinden kurtarır. Starter'lar 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.

Bazı Spring Boot starter'ları aşağıda görülmektedir:

a. **spring-boot-starter:**
   Core starter, auto configuration desteğini içerir, ayrıca logging ve YAML içindir.
b. **spring-boot-starter-data-jpa:**
   Hibernate ile Spring Data JPA'nın kullanılması için gerekli starter.
c. **spring-boot-starter-jdbc:**
   Tomcat JDBC connection pool ile JDBC kullanmak için gerekli starter.
d. **spring-boot-starter-mail:**
   Java Mail ve Spring Framework'ün email gönderme desteği için kullanılan starter.
e. **spring-boot-starter-web:**
   Spring MVC kullanarak RESTfuşl dahil olmak üzere web uygulamaları inşaa etmek için kullanılan starter. Tomcat'i default gömülü web container olarak kullanır.
f. **spring-boot-starter-web-services:**
   Spring web servislerini kullanmak için gerekli starter.
g. **spring-boot-starter-websocket:**
   Spring Framework'in Websocket desteğini kullanrak Websocket uygulamaları geliştirmeyi sağlayan starter.

12 – What are the Spring Boot Annotations?

Başlıca kullanılan Spring Boot anotasyonları aşağıdaki gibidir:

- @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.
- @Configuration - Bean tanımlamaları gibi tanımlamalar için bir Bean sınıfı olduğunu belirtir.
- @Controller - Requestleri yakalayabilme yeteneği olan bir web controller sınıfını belirtir.
- @RequestMapping - Controller sınıfının handle ettiği HTTP request'lerin path eşleştirmesini yapar.
- @Autowired - Constructor, değişken yada setter metotlar için Dependency Injection işlemi gerçekleştirir.
- @Component - Bir sınıfı "Bean" olarak işaretler. Bu sayede Spring'in component tarayıcısı bu sınıfı alıp "App Context" içine ekler. Class seviyesinde bir anotasyondur.
- @SpringBootApplication - Spring Boot autoconfiguration ve component taramasını aktif eder.
- @EnableAutoConfiguration - Bu anotasyon @SpringBootApplication tarafından otomatik olarak getirilir. @EnableAutoConfiguration ve @SpringBootApplication birlikte kullanılmamalıdır. Ya birisi ya da öbürü kullanılır.
- @ComponentScan - Projeye dahil edilen komponenetlerin otomatik taranmasını sağlar.
- @Required – Bean özelliği ayarlayıcı yöntemleri için geçerlidir ve etkilenen Bean özelliğinin yapılandırma zamanında XML yapılandırma dosyasında doldurulması gerektiğini belirtir.
- @Qualifier – Bir interface implementasyonu birden fazla ise veya bir sınıfın örnekleri hafızada birden fazla ise Spring IoC yapısında hangi interface ile hangi sınıfı bağlayacağını bilemeyecektir. Hangi Java Bean'ını hangi anda kullanmak istediğini daha net belirlemek için kullanılır.

13 – What is Spring Boot dependency management?

Bağımlılık Yönetimi, gerekli tüm bağımlılıkları tek bir yerde yönetmenin ve bunlardan verimli bir şekilde yararlanmanın bir yoludur.

Spring Boot, bağımlılıkları ve yapılandırmayı otomatik olarak yönetir. Spring Boot'un her sürümü, desteklediği bağımlılıkların bir listesini sağlar. Bağımlılıklar listesi, Maven ile kullanılabilecek Bills of Materials'ın bir parçası olarak mevcuttur. Bu yüzden konfigürasyonumuzdaki bağımlılıkların versiyonunu belirtmemize gerek yok. Spring Boot kendini yönetir. Spring Boot sürümünü güncellediğimizde Spring Boot, tüm bağımlılıkları tutarlı bir şekilde otomatik olarak yükseltir.

Bağımlılık Yönetiminin Avantajları:

- Spring Boot sürümünü tek bir yerde belirterek bağımlılık bilgilerinin merkezileştirilmesini sağlar. Bir sürümden diğerine geçtiğimizde yardımcı olur.
- Spring Boot kütüphanelerinin farklı sürümlerinin uyumsuzluğunu önler.
- Sadece sürüm belirterek bir kütüphane adı yazmamız gerekir. Bu kullanım çok modüllü projelerde yardımcı olur.
- Spring Boot, gerekirse bağımlılık sürümünün geçersiz kılınmasına da izin verir.

Spring Boot'ta Bağımlılık Yönetiminin Çalışması:

- Bağımlılık, uygulamamızda kullanabileceğimiz belirli işlevleri sağlayan bir 'Kütüphane'den başka bir şey değildir.
- Spring Boot'ta Bağımlılık Yönetimi ve Otomatik Yapılandırma aynı anda çalışır.
- Bağımlılıkları yönetmeyi bizim için son derece kolaylaştıran otomatik yapılandırmadır.
- Bağımlılıkları pom.xml/build.gradle dosyasına eklenmelidir. Bu eklenen bağımlılıklar daha sonra Maven Central'dan indirilecektir. İndirilen bağımlılıklar, yerel dosya sistemindeki '.m2' klasöründe depolanacaktır. Spring-Boot uygulaması bu bağımlılıklara '.m2' ve alt dizinlerinden erişebilir.

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, veri tabanımızın durumu vs. gibi bilgilere çok kolay bir şekilde endpoint'ler üzerinden erişebiliriz.

Spring Boot uygulama linkimizin sonuna actuator/beans , actuator/health gibi parametreler yazarak endpoint'lere ulaşıp bilgi alabiliriz. Örneğin (uygulama_linki)/actuator/health yazarak uygulamamızın ayakta olup olmadığını kontrol edebiliriz. Uygulamamız ayaktayken şöyle bir json veri döner:

```
{
    "status" : "UP"
}
```

Bu arada büyük uygulamalarda bunu aktif ettiğimizde performansı etkileyebileceğini unutmamak gerekir. Actuator'un bize sunduğu tüm özelliklere ihtiyacımız yoksa şayet, gereksiz olanları application.properties'den değerini false yaparak kapatabiliriz.

Yani kısaca, Spring Boot Actuator, uygulamaların production ortamına hazır özellikleri aktifleştirir ve farklı HTTP endpoint'ler ile etkileşimde bulunmayı sağlayan bir yapı sunar.

Spring Boot Actuator' ün projede aktif olması istenirse aşağıdaki Maven dependency bloğunun pom dosyasına eklenmesi gerekir.

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```