**JAVA SPRING BOOT BOOTCAMP 2. HAFTA ÖDEVİ**

**SORU 1: IOC ve DI nedir?**

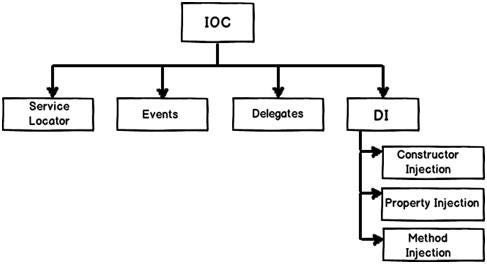
IoC, bir nesnenin kendi bağımlılıklarını oluşturmak yerine, bu bağımlılıkların dışarıdan sağlanmasını ifade eder. Bu prensip, yazılım geliştirme süreçlerinde kontrolün framework veya container gibi dış sistemlere devredilmesini sağlar.

DI, IoC prensibinin uygulanmasını sağlayan bir tekniktir. Nesnelerin bağımlılıkları, kendileri oluşturulmadan önce dışarıdan (constructor, setter ya da field) enjekte edilir.

**DI'nin Yöntemleri:**

1. **Constructor Injection** (En yaygın yöntem)
2. **Setter Injection**
3. **Field Injection**

 IoC’yi uygulamanın birçok yolu vardır, DI ise bunlardan biridir.



*Kaynak: https://medium.com/@kamer-dev/spring-framework-1-ioc-ve-di-3e2da31bc0e5*

DI yaklaşımında ise temel prensip kodumuzda birbirine bağımlı olan bileşenleri tespit edip, bu bağımlılıkları birbirinden ayırıp bağımsız hale getirmek ve bağımlılıklarını dışarıdan enjekte etmektir.

Yazılım geliştirme süreçlerinde modülerliği sağlamak için birçok prensip kullanırız. IoC bunlardan biridir. Kontrolü kodu yazan kişiden alıp başka bir aktöre (mesela framework) teslim etmek demektir. Biz kontrolü teslim ettiğimizde loose-coupling sağlarız. Yani bir nesneyi yaratırken bağımlılıklarını minimuma indirmeyi ve bu bağımlılıklarını da gevşek bağlı yapmaya çalışırız. Peki bu prensipleri uygulamak için ne yaparız sorusunun cevaplarından biri ise dependency injection. Yani bağımlılıkları tespit edip ayırmak ve bu bağlılıkları dışarıdan enjekte etmek.

Özetle uygulamada IOC bir framwork veya container üzerinden gerçekleştirilirken DI, constructor, setter veya field üzerinden yapılır.

**SORU 2: Spring Bean kapsamı nedir?**

Spring Bean, Spring Container tarafından başlatılan setter ve getter yöntemlerine sahip normal bir pojo Java sınıfıdır. Spring container tarafından başlatılırsa Spring bean olarak çağrılır. Kısacası, bir **bean**, Spring konteyneri (container) tarafından oluşturulan, konfigüre edilen ve yönetilen **nesnedir**.  
Spring uygulamalarında kullanılan **bileşenlerin (component)** veya **servislerin (service)** Spring'in denetiminde olması için **bean** olarak tanımlanır.

Spring bean üç şeyle ilgilidir.

1. Spring bean tanımı

2. Spring bean kapsamları

3. Spring bean yaşam döngüsü.

**Bean Oluşturma Yöntemleri**

Spring Boot'ta **bean** oluşturmanın 3 temel yolu vardır:

1. **@Component ile Bean Oluşturma (En Yaygın Yöntem)**

* @Component, Spring’in otomatik olarak algılayacağı bir bean tanımlama yöntemidir.
* @Service, @Repository, ve @Controller anotasyonları da @Component'in özel varyasyonlarıdır.

1. **@Bean ile Bean Oluşturma (Daha Esnek ve Özelleştirilmiş Yöntem)**

* @Bean anotasyonu, genellikle üçüncü parti kütüphanelerden gelen sınıfları veya daha karmaşık nesneleri oluştururken kullanılır.
* @Configuration anotasyonu ile birlikte kullanılır.

1. **XML Konfigürasyonu ile Bean Oluşturma** (Eski Yöntem - Önerilmez)

**Bean Scope (Kapsam) - Bean'in Yaşam Döngüsü**

Spring Boot’ta her bean’in yaşam döngüsü (lifecycle) belirlenebilir.

|  |  |
| --- | --- |
| **Scope** | **Açıklama** |
| singleton | *Tüm uygulama boyunca tek bir nesne oluşturur (Varsayılan).* |
| prototype | *Her çağrıda yeni nesne oluşturur.* |
| request | *Her HTTP isteği başına yeni nesne oluşturur.* |
| session | *Kullanıcı oturumu başına tek bir nesne oluşturur.* |

@Component

@Socope(“…………”) ile tanımlanır.

**Component ve Bean Farkı**

Spring'de @Component ve @Bean aynı amaca hizmet eder gibi görünse de aralarında önemli farklar vardır:

|  |  |  |
| --- | --- | --- |
| **Özellik** | **@Component** | **@Bean** |
| **Kullanım Alanı** | *Sınıf seviyesinde kullanılır. @Service, @Repository gibi özel versiyonları vardır.* | *Yöntem (method) seviyesinde kullanılır ve @Configuration anotasyonu ile birlikte çalışır.* |
| **Otomatik Tespit** | *Spring otomatik olarak tarayıp kaydeder. (IoC Container'a dahil eder.)* | *Manuel olarak tanımlanır ve Spring'e kaydedilir.* |
| **Kapsam** | *Varsayılan olarak singleton olarak çalışır.* | *Varsayılan olarak singleton olarak çalışır ancak @Scope ile değiştirebilir.* |

**Önemli Notlar**

* Singleton ve Prototype tüm uygulamalarda kullanılabilirken, Request, Session ve Application yalnızca web tabanlı projelerde kullanılabilir.
* @Scope anotasyonu @Component, @Service, @Controller gibi anotasyonlarla birlikte çalışır.

**SORU 3: @SpringBootApplication ne yapar?**

@SpringBootApplication anotasyonu, Spring Boot uygulamalarında kullanılan temel bir anotasyondur ve genellikle main metodunun bulunduğu başlatıcı sınıfın üstüne eklenir. Bu anotasyon, Spring Boot'un üç temel anotasyonunun birleşimidir:

**@SpringBootApplication sayesinde:**

1. @Configuration işleviyle bean tanımlamaları yapılabilir.
2. @EnableAutoConfiguration işleviyle gerekli yapılandırmalar otomatik yüklenir.
3. @ComponentScan işleviyle controller, service ve diğer bileşenler otomatik olarak algılanır.

**Dikkat Edilmesi Gereken Noktalar**

* @SpringBootApplication anotasyonunun bulunduğu sınıfın **proje yapısında en üst seviyede yer alması** önerilir. Böylece @ComponentScan tüm bileşenleri kolaylıkla tarayabilir.
* Otomatik yapılandırma özelliği projeyi hızlandırır ancak bazı durumlarda **gereksiz konfigürasyonların yüklenmesine** neden olabilir. Bu durumda @EnableAutoConfiguration(exclude = ...) kullanılarak istenmeyen yapılandırmalar devre dışı bırakılabilir.

**SORU 4: Spring AOP nedir? Nerede ve nasıl kullanılır?**

**Cross Cutting Concern Nedir?**

Concern, uygulamalarımızın belli bölümlerinde core fonksiyonlarımızın dışında yapı ve kurallar dahilinde gerçekleşen **logging, performance, transaction management, security, caching, validation, exception handling**gibi sınıflandırdığımız çatıların her biridir. Cross cutting concern denmesinin sebebi uygulamızın istenilen yerlerinde kullanılması ve kullanılırken o bölgeden soyutlanmış yapılar olmasıdır.

**Aspect Oriented Programming (AOP) Nedir?**

Aspect, uygulamalarımızın **cross cutting concern** kısımlarını işleyen bir programlama paradigmasıdır ve paradigmanın çıkış noktası concernlere çözüm üretmektir. Bu yapının kullanılma motivasyonu, single responsibility ve don’t repeat yourself gibi prensiplere uygun olmasıdır.

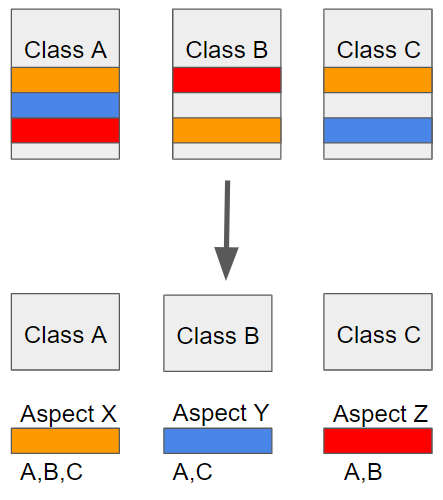
**Faydaları nelerdir?**-Uygulamamızın daha esnek olması ve yönetimi kolaylığı,  
- Tekrar eden kod düzeninden kurtulma,  
- Daha temiz ve anlaşılabilir bir kod,  
- Core logic ve concernlerin birbirinden ayrılması.

**Spring AOP'nin Kullanım Alanları**

* **Loglama**: Uygulama metodlarının giriş-çıkışlarını izlemek.
* **Hata Yönetimi**: Uygulama genelinde hata takibi yapmak.
* **Performans İzleme**: Belirli metodların çalışmasını ölçmek.
* **Güvenlik Kontrolü**: Belirli metodların yalnızca yetkili kullanıcılarca çalıştırılmasını sağlamak.
* **Önbellekleme (Caching)**: Sık kullanılan verileri bellekte saklamak.

**Spring AOP Temel Kavramları**

|  |  |
| --- | --- |
| **Kavram** | **Açıklama** |
| **Aspect** | *Çapraz kesişen işlevleri tanımlayan modüldür. (Örneğin loglama, güvenlik)* |
| **Join Point** | *Aspect'in müdahale edebileceği yerlerdir (metot çağrımı, exception fırlatma vb.)* |
| **Pointcut** | *Hangi Join Point'lerin seçileceğini belirleyen kurallardır.* |
| **Advice** | *Aspect'in ne yapacağını tanımlar (Öncesinde, sonrasında veya hata durumunda çalışabilir).* |
| **Weaving** | *Aspect'lerin uygulama akışına dahil edilme işlemidir. (Compile-time veya Runtime)* |



* **Class A**, **Class B** ve **Class C**'de tekrarlanan loglama, güvenlik ve hata ayıklama gibi işlemler vardı.
* Bu tekrar eden işlemler **Aspect X**, **Aspect Y** ve **Aspect Z**'ye taşındı ve Pointcut kurallarıyla yalnızca belirli sınıflara uygulanarak temiz ve düzenli bir yapı elde edildi.
* Sonuç olarak, Class'lar yalnızca kendi işlevlerine odaklanırken, çapraz kesişen kaygılar tek bir merkezden yönetildi.

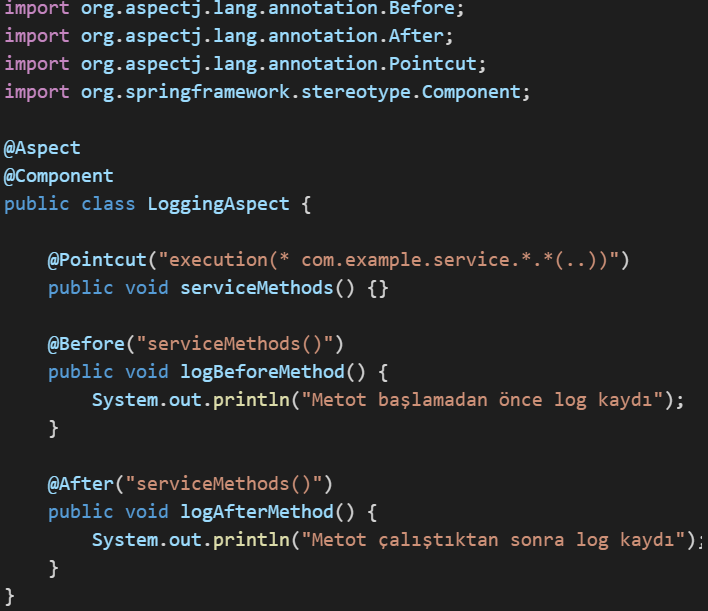
*Kaynak: https://medium.com/@mdemircan/aspect-oriented-programming-nedir-4df7d0a3cd8f*

**Özet**

\* Spring AOP, tekrarlayan kod bloklarını Aspect olarak tanımlayarak yazılımı daha temiz ve düzenli hale getirir.  
\* @Aspect, @Before, @After, @Pointcut gibi anotasyonlarla uygulanır.  
\* Loglama, hata yönetimi ve güvenlik gibi çapraz kesişen kaygılar için güçlü bir çözümdür.

**Kullanım Örneği:**

İlk olarak maven’de bağımlılık eklemesi yapılır.



**SORU 5: Singleton nedir? Nerede kullanılır?**

Singleton tasarım deseni (design pattern), bir sınıftan yalnızca **tek bir örneğin (instance)** oluşturulmasını ve bu örneğe tüm uygulama boyunca aynı noktadan erişilmesini sağlar.

Singleton tasarım deseni (design pattern), bir sınıftan yalnızca **tek bir örneğin (instance)** oluşturulmasını ve bu örneğe tüm uygulama boyunca aynı noktadan erişilmesini sağlar.

**Temel Prensip:**

* Bir sınıf yalnızca tek bir örnek oluşturur ve bu örneğe tüm uygulama boyunca aynı referansla erişilir.

**Singleton Tasarım Deseninin Özellikleri**

1. **Tek Örnek (Single Instance):** Sınıfın yalnızca bir örneği oluşturulur.
2. **Global Erişim (Global Access Point):** Bu örneğe global erişim noktası sağlar.
3. **Kontrollü Oluşturma:** Nesne yalnızca ihtiyaç duyulduğunda oluşturulur (Lazy Initialization).

**Singleton Nasıl Oluşturulur?**

* Singleton deseni genellikle aşağıdaki yöntemlerle oluşturulur:
* Eager Initialization (Uygulama başlarken oluşturma)
* Lazy Initialization (İlk ihtiyaç olduğunda oluşturma)
* Thread Safe Singleton (Çoklu iş parçacığı senaryolarında güvenli Singleton oluşturma)

**Singleton Kullanım Alanları**

Singleton tasarım deseni aşağıdaki senaryolarda idealdir:

\* Veritabanı bağlantıları (Database Connection Pooling): Tüm uygulamanın aynı veritabanı bağlantısını kullanmasını sağlamak.

\* Logger (Günlükleme): Tüm uygulamada tek bir loglama servisi kullanılmasını sağlamak.

\* Yapılandırma Yönetimi (Configuration Management): Uygulama ayarlarını tek bir nesnede toplamak.

\* Önbellekleme (Caching): Verileri tek bir nesnede saklayarak gereksiz veri tabanı erişimlerini önlemek.

\* Cihaz Sürücüleri (Device Drivers): Örneğin yazıcı sürücüsü gibi, tek bir bağlantı yönetilmelidir.

**Özet**

✅ Singleton deseni, uygulama genelinde tek bir nesnenin kullanılmasını sağlar.  
✅ Eager, Lazy ve Thread Safe yöntemleriyle oluşturulabilir.  
✅ Spring Framework'te @Component ve @Bean ile oluşturulan nesneler varsayılan olarak Singleton kapsamındadır.  
✅ Veritabanı bağlantıları, loglama ve önbellekleme gibi kritik yapılarda idealdir.

**SORU 6: Spring Boot Actuator nedir? Nerede kullanılır?**

Spring Boot Actuator, Spring Boot uygulamalarının sağlığını, metriklerini ve çalışma durumunu izlemek için kullanılan güçlü bir araçtır. Actuator, uygulamanın iç durumu hakkında bilgi sağlayan uç noktalar (endpoints) aracılığıyla uygulama yönetimini ve izlemeyi kolaylaştırır.

***Kısaca: Spring Boot Actuator, uygulamanızın ne durumda olduğunu anlamanızı sağlayan bir izleme ve yönetim aracıdır.***

**Spring Boot Actuator'ın Temel Özellikleri**

-**Hazır Uç Noktalar (Endpoints):** Actuator, uygulama hakkında kapsamlı bilgi sağlayan hazır uç noktalar (örneğin /health, /info, /metrics) sunar.

- **Esnek Konfigürasyon:** Hangi uç noktaların etkin olacağını ve nasıl çalışacağını belirlemek için konfigürasyon desteği sağlar.

-**Güvenlik ve Yetkilendirme:** Uç noktaların erişimini kısıtlamak için güvenlik önlemleri ekleyebilirsiniz.

- **Üçüncü Parti Entegrasyonu:** Prometheus, Grafana gibi izleme araçlarıyla kolayca entegre edilebilir.

**En Popüler Spring Boot Actuator Endpoints**

|  |  |
| --- | --- |
| **Uç Nokta** | **Açıklama** |
| */actuator/health* | Uygulamanın sağlık durumunu gösterir. |
| */actuator/info* | Uygulama hakkında özel bilgiler verir (örneğin sürüm, açıklama). |
| */actuator/metrics* | Uygulama metriklerini gösterir (örn: bellek kullanımı, CPU yükü). |
| */actuator/env* | Uygulamanın ortam değişkenlerini gösterir. |
| */actuator/beans* | Spring konteynerine yüklenen Bean'leri listeler. |
| */actuator/mappings* | Tüm HTTP uç noktalarını gösterir. |
| */actuator/logfile* | Uygulama loglarını görüntüler. |

Spring Boot Actuator’u kullanmak için, uygulamanıza Spring Boot Actuator’un maven bağımlılığını eklemek gerekir.

Daha sonra application.properties dosyasına birkaç ayar eklemek gerekir. Örneğin "/actuator/health" URL'sine erişilerek uygulamanın sağlık durumunun sorgulanmasını sağlar.

Uygulama çalışırken, Actuator verilerine REST API aracılığıyla erişilebilir.

**SORU 7: Spring Boot ve Spring arasındaki temel farklar nelerdir?**

|  |  |  |
| --- | --- | --- |
| **Özellik** | **Spring** | **Spring Boot** |
| **Konfigürasyon** | *Konfigürasyon manuel olarak yapılır. XML veya Java tabanlı konfigürasyon gerektirir.* | *@SpringBootApplication ile otomatik konfigürasyon sağlar.* |
| **Bağımlılık Yönetimi** | *Maven veya Gradle'da bağımlılıkların tek tek eklenmesi gerekir.* | *spring-boot-starter-\* gibi hazır bağımlılık setleri kullanır.* |
| **Web Sunucusu** | *Harici Tomcat veya Jetty gibi web sunucuları gereklidir.* | *Dahili (embedded) Tomcat, Jetty veya Undertow ile gelir.* |
| **Geliştirme Hızı** | *Yapılandırma nedeniyle geliştirme süreci daha uzun olabilir.* | *Otomatik konfigürasyon ile hızlı geliştirme imkanı sağlar.* |
| **Kurumsal Projeler** | *Büyük ve karmaşık projelerde daha fazla esneklik sağlar.* | *Microservice tabanlı ve küçük projeler için idealdir.* |
| **Monitoring/Health Check** | *Varsayılan olarak izleme araçları içermez.* | *Spring Boot Actuator ile sağlık durumu ve metrikleri takip edebilir.* |

**Ne Zaman Spring Kullanmalıyım?**

✅ Karmaşık iş mantıkları ve büyük kurumsal projelerde.  
✅ Projede kapsamlı ve özelleştirilmiş konfigürasyon gerekiyorsa.

**Ne Zaman Spring Boot Kullanmalıyım?**

✅ Hızlı geliştirme, küçük projeler veya mikroservis tabanlı yapılar için.  
✅ Varsayılan ayarlarla hızlıca çalışan bir uygulama oluşturmak istediğinizde

**SORU 8: Neden VCS kullanıyoruz?**

**1. Değişikliklerin Takibi**

* VCS, her değişikliğin kaydını tutar ve hangi satırın kim tarafından ne zaman değiştirildiğini gösterir.
* Geriye dönük inceleme yapılmasını sağlar ve kodun önceki sürümlerine kolayca ulaşmayı mümkün kılar.

**2. Ekip Çalışmasını Kolaylaştırma**

* VCS, aynı proje üzerinde çalışan birden fazla geliştiricinin aynı dosya üzerinde çalışmasına olanak tanır.
* Branching ve Merging (dal oluşturma ve birleştirme) gibi özelliklerle farklı geliştirme dalları oluşturulabilir ve bu dallar birleştirilebilir.

**3. Hata Tespiti ve Düzeltme**

* VCS, kod değişikliklerinin hangi commit’te yapıldığını gösterdiği için hata kaynağını bulmayı kolaylaştırır.
* git blame gibi komutlarla hangi satırın kim tarafından ve ne zaman değiştirildiği izlenebilir.

**4. Yedekleme ve Güvenlik**

* VCS, tüm proje geçmişini saklar ve bu sayede dosyaların yanlışlıkla silinmesi veya bozulması durumunda kayıpların önüne geçer.
* Projenin farklı depolarda saklanması (örneğin GitHub, GitLab) sayesinde veri kaybı riski azalır.

**5. Paralel Çalışma (Branching ve Merging)**

* Farklı özelliklerin geliştirilmesini kolaylaştıran branch yapısı sayesinde, aynı anda birden fazla kişi çalışabilir.
* merge işlemiyle bu dallar tek bir projede birleştirilir.

**6. Sürüm Etiketleme (Tagging)**

* VCS, yazılımın belirli sürümlerini (v1.0, v2.0 gibi) etiketleyerek kaydetme imkanı sunar.

**7. Geri Dönüş (Rollback)**

* Yanlış veya hatalı kod değişikliklerinde önceki sürümlere kolayca geri dönmeyi sağlar.

**8. Dokümantasyon ve Takip**

* VCS, yapılan her değişikliğe bir açıklama (commit mesajı) ekleyerek değişikliklerin daha anlaşılır olmasını sağlar.

**SORU 9: SOLID prensipleri nelerdir? Java’da kullanım örnekleri veriniz.**

**1. Single Responsibility Principle (SRP) - Tek Sorumluluk Prensibi**

**Her sınıf yalnızca bir amaca hizmet etmelidir.** Bir sınıfın birden fazla sorumluluğu varsa, bu sorumluluklar farklı sınıflara ayrılmalıdır.

Örnek: 🔹

Bir kütüphane düşünelim. Bir kitap eklemek, silmek ve listelemek gibi işlemleri tek bir sınıfa yazarsak kod karmaşık ve bakımı zor olur. Bunun yerine:

* **KitapYönetimi** → Kitap ekleme, silme ve düzenleme işlemlerini yapar.
* **Raporlama** → Kütüphane istatistikleri gibi raporları oluşturur.

Bu şekilde sorumluluklar ayrılarak daha düzenli ve bakımı kolay bir yapı elde edilir.

**2. Open/Closed Principle (OCP) - Açık/Kapalı Prensibi**

**Bir sınıf genişletilmeye açık, ancak değiştirilmeye kapalı olmalıdır.** Yeni özellik eklenirken mevcut kod değiştirilmemelidir.

Örnek:

Bir **ödeme sistemi** düşünelim. Başlangıçta sadece kredi kartı ödemeleri destekleniyor. Eğer OdemeServisi sınıfına yeni ödeme yöntemleri eklemeye çalışırsak mevcut kodu değiştirmemiz gerekir. Bu SOLID'e aykırıdır. Bunun yerine:

* **Odeme (Arayüz/Interface)** → Ortak ödeme işlemlerini tanımlar.
* **KrediKartıOdeme**, **PayPalOdeme** gibi yeni sınıflar eklenerek genişletilebilir.

Bu yöntemle mevcut kodu değiştirmeden yeni ödeme yöntemleri eklenebilir.

**3. Liskov Substitution Principle (LSP) - Liskov Yerine Geçme Prensibi**

**Alt sınıflar, üst sınıfların yerine kullanılabilir olmalıdır.** Alt sınıflar, üst sınıfın tüm davranışlarını korumalı ve genişletmelidir.

Örnek:

Bir **taşıma sistemi** düşünelim. Arac adlı bir temel sınıf olsun ve bu sınıftan türetilmiş Araba ve Bisiklet sınıfları olsun.

* **Arac** → Hızlanma, fren gibi temel işlevleri tanımlar.
* **Araba** → Arac'ın özelliklerini aynen destekler.
* **Bisiklet** → Eğer Bisiklet sınıfı Arac'ın kurallarına uymuyorsa (örneğin benzine ihtiyaç duymaması gibi) bu SOLID'e aykırı olur.

**4. Interface Segregation Principle (ISP) - Arayüz Ayrımı Prensibi**

**Bir arayüz yalnızca ihtiyacı olan metodları içermelidir.** Büyük arayüzler küçük ve özelleşmiş arayüzlere ayrılmalıdır.

Örnek:

Bir **çalışan yönetim sistemi** düşünelim. Eğer Calisan arayüzü tüm çalışanlar için ortak metotları içeriyorsa bu sorun çıkarabilir. Örneğin:

* **Calisan** → calis() ve yemekYe() metotlarını içeriyorsa, robot çalışanlar yemekYe() metoduna ihtiyaç duymayacaktır.

Bu durumda Calisan arayüzünü şu şekilde bölmeliyiz:

* **Calisabilir** → calis() metodunu içerir.
* **YemekYiyebilir** → yemekYe() metodunu içerir.

Bu yöntemle her sınıf yalnızca ihtiyacı olan metotları içerir.

**5. Dependency Inversion Principle (DIP) - Bağımlılığı Tersine Çevirme Prensibi**

**Üst seviyedeki modüller, alt seviyedeki modüllere doğrudan bağımlı olmamalıdır.** Aralarındaki bağımlılık soyutlamalar (abstraction) üzerinden gerçekleştirilmelidir.

Örnek:

Bir **veritabanı bağlantı sistemi** düşünelim. Eğer VeriServisi doğrudan MySQLVeriTabani sınıfına bağımlı olursa, gelecekte farklı bir veritabanı eklemek zorlaşır. Bunun yerine:

* **VeriTabani (Arayüz/Interface)** → Tüm veritabanı türleri için ortak metotları tanımlar.
* **MySQLVeriTabani**, **PostgreSQLVeriTabani** gibi sınıflar bu arayüzü uygular.

Bu yöntemle VeriServisi doğrudan MySQLVeriTabani sınıfına değil, VeriTabani arayüzüne bağlı olduğu için daha esnek bir yapı elde edilir.

**OCP ve DIP Bağlantısı:**

* **OCP'nin uygulanması**, genellikle **DIP yapısı sayesinde kolaylaşır.**
* **DIP'nin uygulanması**, genellikle **OCP'yi destekler.**

🟡 Ancak şunu da unutmamak gerekir:

* OCP, bir **genişletme stratejisi** iken,
* DIP, bir **bağımlılık yönetimi stratejisidir.**

Yani her OCP çözümü DIP gerektirmez ama iyi bir DIP uygulaması, OCP'nin daha kolay ve sağlıklı uygulanmasını sağlar.

**SORU 10: RAD (Rapid Application Development) Modeli Nedir?**

RAD modeli, geleneksel yazılım geliştirme yöntemlerine kıyasla daha hızlı ve müşteri odaklı bir yaklaşımdır.

Bu modelde, müşteri ihtiyaçlarının hızla değiştiği ve sık sık geri bildirimlerin alındığı projeler için idealdir.

Model:

*✅* ***Prototipleme Odaklı****: RAD modeli, projeyi parçalara ayırarak her parçanın hızlıca prototipini oluşturmayı teşvik eder.  
✅* ***Hızlı Geri Bildirim:*** *Kullanıcıdan sık sık geri bildirim alarak geliştirme süreci optimize edilir.  
✅* ***Esnek Yapı****: Gereksinimler değiştikçe projede hızlı güncellemeler yapılabilir.  
✅* ***Az Belgelenme****: Geleneksel modellere kıyasla daha az dokümantasyon gerektirir.  
✅* ***Hızlı Teslimat:*** *RAD modeli genellikle kısa sürede çalışır ürünler (MVP - Minimum Viable Product) üretmeyi amaçlar.*

**RAD Modelinin Avantajları**

✅ **Daha Hızlı Geliştirme:** RAD modeli sayesinde yazılım geliştirme süreci geleneksel modellere kıyasla daha hızlı ilerler.  
✅ **Esneklik:** Müşteri taleplerindeki değişikliklere kolayca uyum sağlar.  
✅ **Müşteri Odaklı:** Müşteri geri bildirimleri doğrultusunda sürekli iyileştirme yapılır.  
✅ **Erken Çalışır Ürün:** RAD modeli, erken prototipler sayesinde müşteriye hızlıca çalışır ürün sunar.

**RAD Modelinin Dezavantajları**

❗ **Geniş Projelerde Karmaşıklık:** Büyük ve çok modüllü projelerde kontrolün kaybedilmesine yol açabilir.  
❗ **Yoğun Müşteri Katılımı:** RAD modelinin başarılı olması için müşteriyle sık iletişim gereklidir.  
❗ **Teknik Risk:** RAD modeliyle geliştirilen prototiplerin teknik altyapısı bazen yetersiz kalabilir.

**SORU 11: Spring Boot Starter Nedir ve Nasıl Kullanılır?**

Spring Boot Starter'lar, Spring Boot uygulamalarında belirli işlevsellikleri hızla entegre etmek için kullanılan hazır bağımlılık paketleridir. Starter'lar, uygulamaya ihtiyaç duyulan kütüphaneleri tek bir bağımlılık ekleyerek yüklemeyi kolaylaştırır.

**Temel Amaç:** Konfigürasyon ve bağımlılık yönetimini kolaylaştırarak geliştiricinin yalnızca iş mantığına odaklanmasını sağlamaktır.

**Spring Boot Starter'ların Avantajları**

✅ **Kolay Bağımlılık Yönetimi:** Starter'lar, birden fazla kütüphaneyi tek bir bağımlılık ile ekler.  
✅ **Otomatik Yapılandırma:** Starter'lar sayesinde gerekli ayarlar otomatik olarak yapılır, ek yapılandırmaya gerek kalmaz.  
✅ **Standartlaştırma:** Projede hangi kütüphanelerin kullanıldığı daha açık ve düzenli şekilde görünür.  
✅ **Hızlı Başlangıç:** Temel ayarlarla hızlıca çalışır durumda bir uygulama oluşturulabilir.

En yaygın starterlar:

|  |  |
| --- | --- |
| **Starter Paketi** | **Açıklama** |
| spring-boot-starter-web | *Web uygulamaları için gerekli Tomcat, MVC ve Jackson gibi kütüphaneleri ekler.* |
| spring-boot-starter-data-jpa | *JPA (Hibernate) ve veritabanı işlemleri için kullanılır.* |
| spring-boot-starter-security | *Spring Security için gerekli bağımlılıkları ekler.* |
| spring-boot-starter-test | *JUnit, Mockito gibi test araçlarını içerir.* |
| spring-boot-starter-thymeleaf | *HTML şablon motoru olan Thymeleaf için kullanılır.* |
| spring-boot-starter-actuator | *Uygulama sağlığı, metrikler ve yapılandırma bilgileri için kullanılır.* |
| spring-boot-starter-mail | *E-posta gönderimi için kullanılır.* |
| spring-boot-starter-cache | *Önbellekleme (Caching) işlemleri için kullanılır.* |

Backend için önemli starterlar:

// **spring-boot-starter-web** → Web projeleri ve REST API'ler için.

// **spring-boot-starter-data-jpa** → Veritabanı işlemleri için.

// **spring-boot-starter-security** → Güvenlik ve kimlik doğrulama için.

// **spring-boot-starter-validation** → Giriş ve doğrulama kuralları için.

// **spring-boot-starter-actuator** → Uygulama izleme ve sağlık kontrolleri için.

// **spring-boot-starter-test** → Unit ve integration testleri için.

// **spring-boot-devtools** → Geliştirme ortamında hızlı geri bildirim için.

**SORU 12: Caching Nedir ve Spring Boot'ta Nasıl Uygulanır?**

Caching (Önbellekleme), veriye daha hızlı erişim sağlamak amacıyla sık kullanılan verilerin bellekte (RAM) saklanması işlemidir. Bu sayede uygulama, veritabanı veya harici servis gibi yavaş kaynaklara tekrar tekrar başvurmak zorunda kalmaz ve performans artar.

\* **Performans Artışı:** Verinin bellekte tutulması, tekrar eden sorgularda daha hızlı yanıt verir.  
\* **Veritabanı Yükünü Azaltır:** Aynı veriye tekrar tekrar sorgu yapmak yerine önbellekten getirmek kaynak tüketimini düşürür.  
\***Hızlı Erişim:** Veriler daha **hızlı erişilebilen** bellekte saklandığı için yanıt süresi kısalır.

**Spring Boot'ta Caching Nasıl Uygulanır?**

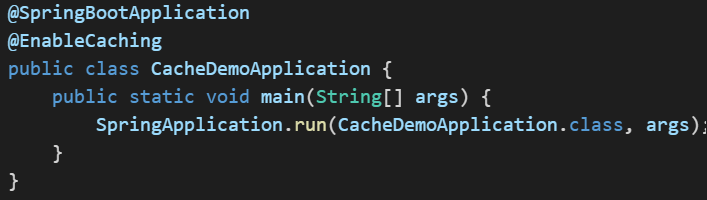
Spring Boot, @Cacheable, @CachePut ve @CacheEvict anotasyonlarıyla güçlü bir caching mekanizması sağlar.

**-Gerekli Bağımlılıkların Eklenmesi**

Spring Boot'ta caching yapmak için aşağıdaki starter'ı eklenir

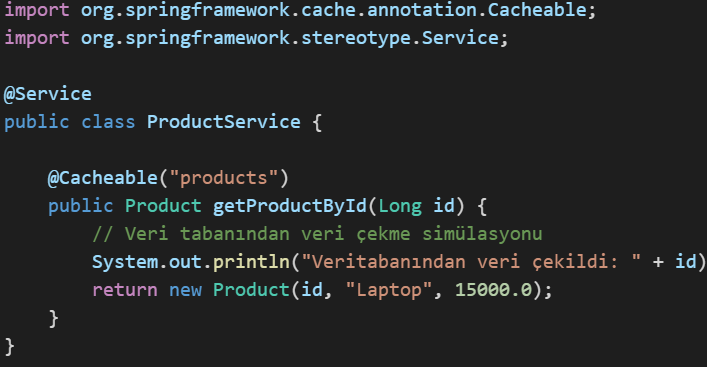
**-@EnableCaching Anotasyonunun Eklenmesi**

Spring Boot uygulamalarında caching özelliğini etkinleştirmek için @EnableCaching anotasyonunu @SpringBootApplication ile birlikte eklenir.



**-@Cacheable Kullanımı (Veri Okuma)**

@Cacheable, belirli bir metodun çıktısını cache'e (önbelleğe) kaydeder. Aynı parametreyle tekrar çağrıldığında metot çalışmaz, direkt cache'deki veri döner.



**-@CachePut Kullanımı (Veri Güncelleme)**

@CachePut, cache'e eklenmiş veriyi günceller. Metot her çağrıldığında çalışır ve cache'i yeniler.

**-@CacheEvict Kullanımı (Veri Silme)**

@CacheEvict, belirli bir cache bilgisini siler. Cache'in güncel tutulmasını sağlar.

**// Cache Yapıları (Opsiyonel)**

Spring Boot, varsayılan olarak **ConcurrentMap** tabanlı caching sağlar. Daha gelişmiş çözümler için şu teknolojiler tercih edilebilir:

* **Redis** (En çok tercih edilen, yüksek performanslı cache çözümü)
* **Ehcache** (Java tabanlı güçlü cache çözümü)
* **Caffeine** (Düşük gecikmeli, yüksek performanslı cache çözümü)

**SORU 13: Logging Nedir, Nasıl, Nerede ve Neden Kullanılır?**

Logging, bir uygulamanın çalışması sırasında meydana gelen olayların, hataların, uyarıların ve bilgi mesajlarının kaydedilmesidir. Bu kayıtlar genellikle dosyalarda, konsolda veya merkezi izleme sistemlerinde saklanır.

Amaç: Uygulamanın çalışma durumunu izlemek, hataları analiz etmek ve sistemin performansını değerlendirmektir.

**Logging Neden Önemlidir?**

**Hata Ayıklama (Debugging):** Hataların kaynağını tespit etmek için gereklidir.  
**Sistem İzleme:** Uygulamanın çalışma sürecini anlamak ve analiz etmek için kullanılır.  
**Performans Takibi:** Uygulama performansını ölçmek ve darboğazları belirlemek için kullanılır.  
**Güvenlik:** Şüpheli aktiviteleri ve saldırıları tespit etmek için kullanılır.  
**Kullanıcı Davranışlarının İzlenmesi:** Kullanıcı işlemlerini analiz ederek daha iyi bir kullanıcı deneyimi sağlanabilir.

**Logging Nerede Kullanılır?**

* **Uygulama Başlatıldığında:** Başarılı başlatma veya hata durumunda bilgilendirme.
* **Veritabanı İşlemlerinde:** CRUD (Create, Read, Update, Delete) işlemlerinde veri manipülasyonunu izlemek.
* **API Çağrılarında:** Dış servislere yapılan istek ve yanıtların kaydedilmesi.
* **Güvenlik Kontrolünde:** Yetkisiz erişim, hatalı giriş denemeleri gibi olayların kaydedilmesi.
* **Performans Ölçümlerinde:** Metotların çalışma sürelerini analiz etmek için.

**Logging Nasıl Yapılır?**

Spring Boot'ta yaygın olarak kullanılan logging framework'leri şunlardır:

* SLF4J (Simple Logging Facade for Java) - En yaygın logging arayüzüdür.
* Logback - Varsayılan Spring Boot logging framework'üdür.
* Log4j/Log4j2 - Alternatif güçlü bir logging framework'üdür.
* Java Util Logging (JUL) - Java'nın standart logging kütüphanesidir.

**Spring Boot'ta Logging Yapılandırması**

Spring Boot varsayılan olarak Logback kullanır. Ek bir bağımlılık eklemeden direkt SLF4J kullanılabilir.

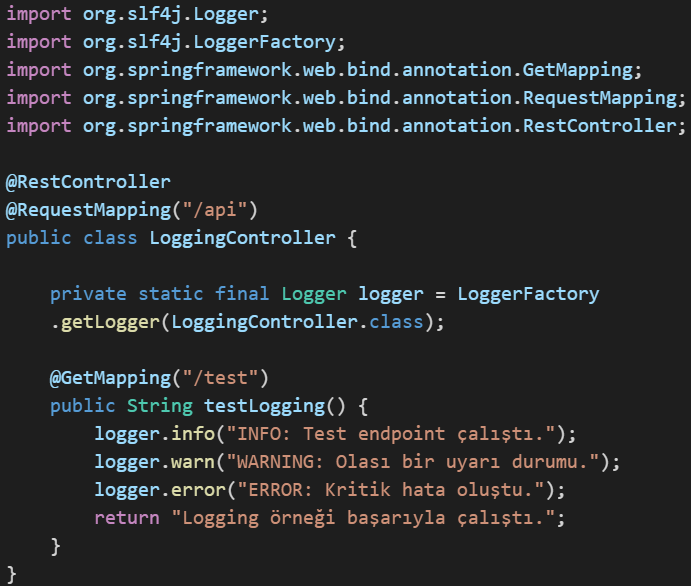
**application.properties Dosyasıyla Yapılandırma**

Spring Boot'ta log seviyelerini şu şekilde belirlenebilir:

logging.level.root=INFO

logging.level.com.example=DEBUG

logging.file.name=logs/app.log



**Log Seviyeleri ve Kullanım Alanları**

|  |  |  |
| --- | --- | --- |
| **Seviye** | **Açıklama** | **Kullanım Alanı** |
| **TRACE** | *En ayrıntılı seviyedir. Debug'dan daha fazla detay içerir.* | *Derin hata ayıklama (low-level debug)* |
| **DEBUG** | *Geliştirme aşamasında hata ayıklama için idealdir.* | *Metotların giriş-çıkışları* |
| **INFO** | *Genel bilgi mesajları içindir.* | *Başarılı işlemler, başlangıç mesajları* |
| **WARN** | *Dikkat gerektiren durumlar için kullanılır.* | *Potansiyel sorunlar, disk dolma uyarısı gibi* |
| **ERROR** | *Hata durumlarında kullanılır.* | *Uygulamanın kritik bölümlerinde sorun yaşandığında* |
| **FATAL** | *Uygulamanın çalışmasını durduracak hatalar içindir.* | *Veri kaybı, sistem çökmesi gibi ciddi durumlarda* |

**SORU 14: Swagger Nedir ve Spring Boot'ta Nasıl Kullanılır?**

Swagger, RESTful web servislerinin belgelenmesi, test edilmesi ve doğrulanması için kullanılan güçlü bir açık kaynaklı araçtır. Swagger, API'leri görselleştirerek geliştiricilere ve kullanıcıya kolay bir arayüz sunar.

**Swagger'ın Avantajları**

✅ **Otomatik Dokümantasyon:** API endpoint'lerini manuel olarak yazmadan otomatik olarak belge oluşturur.  
✅ **Kolay Test İmkanı:** Swagger UI arayüzü üzerinden API'leri doğrudan test edebilirsin.  
✅ **Geliştirici Dostu:** Swagger UI, API'nin girdilerini, çıktısını ve hata durumlarını görselleştirir.  
✅ **Entegrasyon Kolaylığı:** Spring Boot gibi framework'lerle kolayca entegre olur.

**Swagger Entegrasyonu Nasıl Yapılır?**

Spring Boot'ta Swagger entegrasyonu için genellikle **springdoc-openapi** veya **Swagger 2** kütüphanesi kullanılır. Burada en güncel ve tercih edilen yöntem olan springdoc-openapi gösterilecektir.

Maven bağımlılığı yapılmalıdır.

Swagger UI üzerinde API endpoint'leri şu detaylarla görüntülenir:

* **URL:** API adresi (örn: /api/users/hello)
* **Method:** GET, POST, PUT, DELETE gibi HTTP metodları
* **Parameters:** Endpoint'e gönderilebilecek parametreler
* **Responses:** Dönüş değerleri ve hata mesajları

**İyi Bir Swagger Yapısı İçin İpuçları**

✅ Her Endpoint için açıklama ekleyin.

✅ Dönen HTTP status kodlarını açıkça belirtin.

✅ Parametrelerin veri tiplerini ve zorunluluk durumlarını belirtin.

✅ Güvenlik (JWT, OAuth2) gibi konular için Swagger yapılandırmalarını kullanın.