**The dependency inversion principle**

**Constructor injection**

Spring container projede tanımlı interface ve classlar için otomatik olarak nesneler üretir ve ihtiyaç duyduğumuzda bu nesnelere erişip kullanabilir. Böylece herhangi bir nesneye ihtiyaç duyduğumuzda her seferinde new ile manuel olarak nesne üretmekten kurtulmuş oluruz.

Spring container nesne oluşturmak için projeyi tararken “@Components” annotasyonuna göre arama yapar. Nesneyi oluştururken eğer birden fazla kurucu metod varsa “@Autowired” anotasyonu bulunan kurucu metodu çalıştırır.

**Component Scanning**

Spring proje çalıştırıldığında varsayılan olarak ana paket altında “@Component” anotasyonuna sahip olan classlar vb. (beans) tarar ve istediğimiz de erişip kullanabilmemiz için tutar. Eğer ana paket dışında, farklı bir paket içerisinde “@Component” anotasyonuna sahip ögelerimiz varsa, Srpingin ana paket haricinde taramasını istediğimiz bu paketleri belirtmemiz gerekir. Bunu belirtmenin yolu projemizi çalıştırdığımız classda “@SpringBootApplication” anotasyonu içerisine yazmaktır.

ÖRNEK:

@SpringBootApplication(  
 scanBasePackages = {"com.bmu.springcoredemo",  
 "com.bmu.util"}  
)

Böylece projemizi çalıştırdığımızda, Spring belirtmiş olduğumuz paketlerin altında “@Component” anotasyonunu bulur.

**Setter injection:**

Setter injection, setter metodları kullanılarak enjekte etme yöntemidir.

@Autowired  
public void setCoach(Coach theCoach){  
 myCoach=theCoach;  
}

**HANGİSİNİ KULLANMAK DAHA İYİDİR?**

Spring geliştirici tamının öncelikli tavsiyesi Constructor enjection kullanmaktır, fakat isteğe ve ihtiyaca göre setter injection da kullanılabilir. Daha fazla bilgi için spring kılavuzu incelenmelidir.

**Field Injection with Annotations and Autowiring**

Unit testi zorlaştırır, eskimiş bir kullanımdır. Günümüzde yaygın olarak kullanılmaz ve spring geliştirici ekibi tarfından kullanımı önerilmemektedir.

**Kullanımı:** Herhangi bir kurucu veya ayarlayıcı metod tanımlanmaz sadece tanımlanan nesne üzerine “@Autowired” eklenir.

**Annotation Autowiring and Qualifiers**

Eğer spring injection annotation kullanırken, bir arayüzden(interface) türeyen birden fazla sınıf varsa spring hata verir hangisinin enjekte edileceğini belirlemek gerekir.

Bu belirtmeyi “@Qualifier” annotationu ile yaparız, injection metodun parametre kısmında belirtilir. Burada dikkat edilmesi gereken kısım, sınıf adını “Qualifier” annotationunda tanımlarken sınıf adını küçük harf ile başlatmak gerektiğidir.

**@Primary Annotation**

Component olarak işaretlenmiş sınıf sayısı birdan fazla ise springin hangisini enjekte edeceğine karar vermek için kullanılan başka bir yöntem de @primary annotationundur.

Component olarak tanımlı sınıfın başına yazılır ve birden çok component içerisinden yalnızca birine @primary anotasyonu uygulanabilir. Birden fazla @primary anotasyonuna sahip component sınıf olması halinde spring hata verir.

**Hangisini Kullanmak Daha İyi Olur?**

İkisi de temelde aynı işi yapar, aynı anda ikisi de kullanılabilir fakat “@Qalifiers” anotasyonu> “@Primary” anotasyonudur, dolayısıyla herhangi bir component @primary anotasyonuna sahip olsa dahi @Qualifiers anostsyonunda belirtilen component enjekte edilir.

“@Qualifiers” anotasyonu kullanmak daha spesifik ve karmaşa önleyici olduğundan tercih edilmelidir.

**Lazy Initialization Annotation (Temel Başlatma)**

@Lazy anotasyonuna sahip olan beanler spring tarafından program çalıştırıldığında taranır fakat otomatik olarak oluşturulmaz. Ya bir dependency injection(bağımlılık enjeksiyonu) ya da açıkça talep edildiğinde oluşturulur ve kullanılabilir hale gelir.

**Avantajları:**

Sadece ihtiyaç halinde nesne üretilir.

Çok fazla componentin olduğu durumda sistemin çalışmaya başlama süresi kısalır.

**Dezavantajları:**

@RestController isteklerde bulunulmadan önce sorun olmadığındna emin olmalısınız.

Tüm Beanler için yeterli belleğe sahip olduğunuzdan emin olmalısınız.

**Tüm componentleri varsayılan olarak lazy ayarlama**

Application.properties dosyasına

spring.main.lazy-initialization=true

komutu eklenmelidir.

**Bean Scopes:**

Scope bir beanin yaşamdöngüsünü gösterir, ne kadar canlı kalacağı, kaç örneğinin üretileceği ve nasıl paylaşılacağı gibi. **Springin varsayılan scope’u singletondır .**

**Singleton nedir:**

Spring Container varsayılan olarak her bir bean için yalnızca bir örneğini üretir ve tüm dependency injection(bağımlılık örneği)ler aynı beani referans alır.

@Scope anotasyonu sayesinde springin birden fazla nesne örneği oluturması sağlanır.

@Scope(ConfigurableBeanFactory.*SCOPE\_PROTOTYPE*)

Bu anotasyon ile bellekte birden fazla örnek oluşturulması sağlanır.

@Scope(ConfigurableBeanFactory.*SCOPE\_SINGLETON*)

Bu varsayılan ayardır ve bu şekilde tüm beanler aynı bellek alanını kullanır.

(myCoach==anotherCoach)

Bu koşul PROTOTYPE için false döndürürken SINGLETION için true döndürür.

**Configuration Beans with Java Code**

3.parti bir sınıfı kullanılabilir hale getirmek için bu yöntem kullanılır, örneğin bir jar dosyasını spring çatısı altında kullanılabilir bir bean’e dönüştürmek istersek bu yöntem kullanılır.

Configurasyon kullanım:

@Configuration  
public class SportConfig {  
 @Bean  
 public Coach swimCoach(){  
 return new SwimCoach();}  
}

Eğer istersek manuel olarak oluşturduğumuz bean’e farklı bir id verebiliriz be böylece erişmek istediğimizde bu id aracılığıyla erişebiliriz.

Manuel id belirleme:

@Bean("aquatic")  
public Coach swimCoach(){  
 return new SwimCoach();  
}

Manuel id ile injection etme:

@Autowired  
public DemoController(@Qualifier("aquatic") Coach theCoach){  
 System.*out*.println("In constructor: "+getClass().getSimpleName());  
 myCoach=theCoach;  
}