Spring Boot Nedir?

Spring Boot, bağımsız, üretim seviyesinde Spring tabanlı uygulamaların minimal çabayla programlanması için kullanılan açık kaynaklı bir Java framework'üdür. Spring Boot, Spring Java platformunun bir uzantısı olarak yapılandırma sorunlarını en aza indirmek amacıyla geliştirilmiştir. Spring ekibinin "önerilen görüş" yaklaşımı ile birçok yapılandırma önceden ayarlanmıştır ve Spring platformu ile üçüncü parti kütüphanelerin en iyi şekilde kullanımı sağlanır.

Spring Boot, mikro hizmetler, web uygulamaları ve diğer Java tabanlı projeler geliştirmek için yaygın olarak kullanılmaktadır. Kullanım kolaylığı ve sağlamlığı nedeniyle tercih edilir.

Özellikler

- Tomcat, Jetty veya Undertow gibi web uygulama sunucuları yerleşik olarak gelir.

Kod örneği (Tomcat):

"pom.xml":

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.ornek</groupId>
  <artifactId>spring-boot-ornek</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.5.2</version>
  </parent>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
```

```
    </dependency>
  </dependencies>
</project>
```

“Uygulama.java”:

```
package com.ornek;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Uygulama {
  public static void main(String[] args) {
    SpringApplication.run(Uygulama.class, args);
  }
}
```

- Maven ve Gradle gibi yapı araçları için başlangıç POM dosyaları sağlar. Örneğin, yukarıdaki pom.xml dosyası, Spring Boot uygulamasını başlatmak için gereken en temel yapılandırmadır. Spring Boot bağımlılıkları, “spring-boot-starter-parent" ile belirlenir.

- Spring uygulamasının otomatik olarak yapılandırılmasını sağlar.

Kod örneği:

“pom.xml”:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.ornek</groupId>
    <artifactId>demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
```

```
    <packaging>jar</packaging>

    <name>demo</name>
    <description>Spring Boot için örnek proje</description>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.7.0</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

“DemoUygulamasi.java” (Ana Uygulama Sınıfı):

```
package com.ornek.demo;

import org.springframework.boot.SpringApplication;
```

```java
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DemoUygulamasi {

    public static void main(String[] args) {
        SpringApplication.run(DemoUygulamasi.class, args);
    }
}
```

“MerhabaController.java” (RestController Sınıfı):

```java
package com.ornek.demo;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class MerhabaController {

    @GetMapping("/merhaba")
    public String merhabaDe() {
        return "Merhaba, Dünya!";
    }
}
```

Bu kodları kullanarak Spring Boot uygulamasını başlatan kişi, tarayıcıda "http://localhost:8080/merhaba" adresine giderek "Merhaba, Dünya!" mesajını görebilir. Bu örnek, Spring Boot'un otomatik yapılandırma özelliklerini kullanmayı sağlar.

- Metrikler, sağlık kontrolleri ve dışsallaştırılmış konfigürasyon gibi üretime hazır işlevsellikler sunar.

Kod örneği:

```java
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.actuate.health.Health;
import org.springframework.boot.actuate.health.HealthIndicator;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Bean;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
public class Uygulama {

    public static void main(String[] args) {
        SpringApplication.run(Uygulama.class, args);
    }

    @Bean
    @ConfigurationProperties(prefix = "ornek")
    public OrnekAyarlar ornekAyarlar() {
        return new OrnekAyarlar();
    }

    @Bean
    public HealthIndicator ozelSaglikKontrolcusu() {
        return () -> Health.up().withDetail("mesaj", "Uygulama
sağlıklı").build();
    }
}

@RestController
class MetrikController {

    private final OrnekAyarlar ornekAyarlar;

    public MetrikController(OrnekAyarlar ornekAyarlar) {
        this.ornekAyarlar = ornekAyarlar;
    }

    @GetMapping("/metrikler")
    public String metrikler() {
        return "Metrikler: " + ornekAyarlar.getMetrikler();
    }
}

class OrnekAyarlar {
    private String metrikler = "ornek metrikler";
```

```
    public String getMetrikler() {
        return metrikler;
    }

    public void setMetrikler(String metrikler) {
        this.metrikler = metrikler;
    }
}
```

- Kod üretimine gerek kalmadan çalışır.

- XML konfigürasyonuna ihtiyaç duymaz. **Spring Boot, konfigürasyonların çoğunu anotasyonlar ve özellik dosyaları (application.properties veya application.yml) ile yapar, bu da XML konfigürasyonlarına ihtiyaç duyulmadığını gösterir.**

- Java'ya ek olarak opsiyonel Kotlin ve Apache Groovy desteği sunar.

Kod örneği (Kotlin desteğiyle ilgili):

"pom.xml":

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.ornek</groupId>
    <artifactId>demo-kotlin</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>demo-kotlin</name>
    <description>Spring Boot için Kotlin örnek proje</description>
```

```xml
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.7.0</version>
        <relativePath/>
    </parent>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.jetbrains.kotlin</groupId>
            <artifactId>kotlin-stdlib-jdk8</artifactId>
        </dependency>
        <dependency>
            <groupId>org.jetbrains.kotlin</groupId>
            <artifactId>kotlin-reflect</artifactId>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.jetbrains.kotlin</groupId>
                <artifactId>kotlin-maven-plugin</artifactId>
                <version>1.5.21</version>
                <executions>
                    <execution>
                        <goals>
                            <goal>compile</goal>
                            <goal>test-compile</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

</project>
```

Ana Uygulama Sınıfı:

```
package com.ornek.demo

import org.springframework.boot.autoconfigure.SpringBootApplication
import org.springframework.boot.runApplication

@SpringBootApplication
class DemoKotlinUygulamasi

fun main(args: Array<String>) {
    runApplication<DemoKotlinUygulamasi>(*args)
}
```

RestController Sınıfı:

```
package com.ornek.demo

import org.springframework.web.bind.annotation.GetMapping
import org.springframework.web.bind.annotation.RestController

@RestController
class MerhabaController {

    @GetMapping("/merhaba")
    fun merhabaDe(): String {
        return "Merhaba, Dünya!"
    }
}
```

DispatcherServlet'in Bootstrap Edilmesi

Spring Boot, DispatcherServlet'in manuel olarak yapılandırılmasını gerektirmez. Uygulamanın konfigürasyonuna göre otomatik olarak yapılandırma yapar.

Kod örneği:

“pom.xml”:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.ornek</groupId>
    <artifactId>demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>demo</name>
    <description>Spring Boot için örnek proje</description>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.7.0</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
```

```
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

“DemoUygulamasi.java” (Ana Uygulama Sınıfı):

```
package com.ornek.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DemoUygulamasi {

    public static void main(String[] args) {
        SpringApplication.run(DemoUygulamasi.class, args);
    }
}
```

“MerhabaController.java” (RestController Sınıfı):

```
package com.ornek.demo;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class MerhabaController {

    @GetMapping("/merhaba")
    public String merhabaDe() {
        return "Merhaba, Dünya!";
    }
}
```

Bu kodları kullanarak Spring Boot uygulamasını başlatan kişi, tarayıcıda "http://localhost:8080/merhaba" adresine giderek "Merhaba, Dünya!" mesajını görebilir. Bu örnek, Spring Boot'un otomatik yapılandırma özelliklerini kullanmayı sağlar.

SpringBootServletInitializer

Spring Boot, WebApplicationInitializer'ın bir özelleşmesi olan SpringBootServletInitializer sınıfına sahiptir. Bu sınıf, geliştiricinin kendi WebApplicationInitializer sınıfını oluşturma ihtiyacını ortadan kaldırır.

Kod örneği:

"pom.xml":

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.ornek</groupId>
    <artifactId>demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>war</packaging> <!-- WAR paketi olarak belirtiliyor -->

    <name>demo</name>
    <description>Spring Boot için örnek proje</description>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.7.0</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
```

```xml
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-tomcat</artifactId>
            <scope>provided</scope> <!-- Gömülü Tomcat kullanılmayacak -->
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

“DemoUygulamasi.java” (Ana Uygulama Sınıfı):

```java
package com.ornek.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import
org.springframework.boot.web.servlet.support.SpringBootServletInitializer;
import org.springframework.boot.builder.SpringApplicationBuilder;

@SpringBootApplication
public class DemoUygulamasi extends SpringBootServletInitializer {

    public static void main(String[] args) {
        SpringApplication.run(DemoUygulamasi.class, args);
    }

    @Override
```

```
    protected SpringApplicationBuilder configure(SpringApplicationBuilder
application) {
        return application.sources(DemoUygulamasi.class);
    }
}
```

“MerhabaController.java” (RestController Sınıfı):

```
package com.ornek.demo;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class MerhabaController {

    @GetMapping("/merhaba")
    public String merhabaDe() {
        return "Merhaba, Dünya!";
    }
}
```

Açıklama

- **pom.xml:** Projenin WAR olarak paketlenmesi için “<packaging>war</packaging>” etiketi kullanılmıştır. Ayrıca, “spring-boot-starter-tomcat" bağımlılığı provided kapsamına alınmıştır, bu da uygulamanın kendi Tomcat sunucusunu sağlamayacağı anlamına gelir.
- **Ana Uygulama Sınıfı (DemoUygulamasi)**: SpringBootServletInitializer sınıfından miras alır ve configure metodunu override eder. Bu, uygulamanın bir servlet konteyner içinde çalıştırılması için gereklidir.
- **RestController Sınıfı (MerhabaController)**: Basit bir REST endpoint sağlar ve /merhaba URL'sine gelen GET isteklerine yanıt verir.

Bu yapılandırma ile, uygulamanızı bir WAR dosyası olarak paketleyebilir ve geleneksel bir servlet konteyner (örneğin, Tomcat, Jetty) içinde çalıştırabiliriz. SpringBootServletInitializer sayesinde, WebApplicationInitializer sınıfını manuel olarak oluşturmamıza gerek kalmayacaktır.

Konfigürasyon Özellikleri

Spring Boot uygulaması için konfigürasyon özellikleri application.properties veya application.yml dosyasında belirtilebilir. Bu dosyada belirtilen özellikler arasında server.port ve spring.application.name gibi özellikler bulunabilir.

Kod örneği:

"pom.xml":

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.ornek</groupId>
    <artifactId>demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>demo</name>
    <description>Spring Boot için örnek proje</description>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.7.0</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
    </dependencies>

    <build>
```

```
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

“DemoUygulamasi.java” (Ana Uygulama Sınıfı):

```
package com.ornek.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DemoUygulamasi {

    public static void main(String[] args) {
        SpringApplication.run(DemoUygulamasi.class, args);
    }
}
```

“MerhabaController.java” (RestController Sınıfı):

```
package com.ornek.demo;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class MerhabaController {

    @GetMapping("/merhaba")
    public String merhabaDe() {
        return "Merhaba, Dünya!";
    }
}
```

“application.properties”:

```
server.port=8081
spring.application.name=DemoUygulamasi
```

“application.yml” (Alternatif):

```
server:
  port: 8081

spring:
  application:
    name: DemoUygulamasi
```

Açıklama

- **“pom.xml” Dosyası**: Gerekli bağımlılıkları içerir. spring-boot-starter-web bağımlılığı, web uygulaması için gerekli olan Spring bileşenlerini sağlar.
- **Ana Uygulama Sınıfı**: @SpringBootApplication anotasyonu ile işaretlenmiştir. Bu anotasyon, @Configuration, @EnableAutoConfiguration ve @ComponentScan anotasyonlarını bir araya getirir. Uygulamanın ana sınıfı olarak DemoUygulamasi adında bir sınıf oluşturulmuştur. Bu sınıfın main metodu, Spring Boot uygulamasını başlatır.
- **RestController Sınıfı**: MerhabaController adlı bir Controller sınıfı oluşturulmuştur. Bu sınıf, /merhaba URL'sine gelen GET isteklerine yanıt verir ve "Merhaba, Dünya!" mesajını döner.
- **“application.properties" Dosyası**: Uygulamanın sunucu portunu 8081 olarak ve uygulama adını DemoUygulamasi olarak ayarlar.
- **“application.yml” Dosyası**: application.properties dosyasının alternatifidir ve aynı ayarları YAML formatında belirtir.

Bu yapılandırma ile, Spring Boot uygulaması http://localhost:8081 adresinde çalışır ve uygulamanın adı DemoUygulamasi olarak ayarlanır. Bu dosyalarda belirtilen konfigürasyon özellikleri, uygulamanın çalışma zamanında nasıl davranacağını belirler.

Otomatik Konfigürasyon

“@SpringBootApplication” anotasyonu, Spring Boot uygulamasının üçüncü taraf kütüphaneleri ve sınıf yolunda bulunan özellikleri otomatik olarak yapılandırmasını sağlar. Bu anotasyon, üç Spring spesifik anotasyonu birleştirir: “@SpringBootConfiguration”, “@EnableAutoConfiguration” ve “@ComponentScan”.

- **@SpringBootConfiguration**: Spring spesifik @Configuration anotasyonunun bir özelleşmesidir. Bu anotasyon, sınıfı Spring Boot uygulaması için konfigürasyon sınıfı olarak işaretler.
- **@EnableAutoConfiguration**: Spring Boot otomatik konfigürasyonunu etkinleştiren Spring spesifik bir anotasyondur.

Kod örneği:

“pom.xml”:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.ornek</groupId>
    <artifactId>demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>demo</name>
    <description>Spring Boot için örnek proje</description>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.7.0</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>

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

```
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

“DemoUygulamasi.java” (Ana Uygulama Sınıfı):

```
package com.ornek.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DemoUygulamasi {

    public static void main(String[] args) {
        SpringApplication.run(DemoUygulamasi.class, args);
    }
}
```

“MerhabaController.java” (RestController Sınıfı):

```
package com.ornek.demo;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class MerhabaController {

    @GetMapping("/merhaba")
```

```
    public String merhabaDe() {
        return "Merhaba, Dünya!";
    }
}
```

“application.properties”:

```
server.port=8081
spring.application.name=DemoUygulamasi
```

Anotasyonlar ve Açıklamaları

@SpringBootApplication

- Bu anotasyon, aşağıdaki üç ana anotasyonu birleştirir ve Spring Boot uygulamasının ana sınıfını işaretler:
    - **@SpringBootConfiguration**: Bu anotasyon, Spring spesifik @Configuration anotasyonunun bir özelleşmesidir ve sınıfı Spring Boot uygulaması için bir konfigürasyon sınıfı olarak işaretler.
    - **@EnableAutoConfiguration**: Bu anotasyon, Spring Boot'un otomatik konfigürasyonunu etkinleştirir. Spring Boot, sınıf yolunda bulunan bağımlılıkları tarar ve uygun konfigürasyonları otomatik olarak uygular.
    - **@ComponentScan**: Bu anotasyon, belirtilen paketi ve alt paketlerini tarayarak Spring bileşenlerini (örneğin, @Component, @Service, @Repository, @Controller) otomatik olarak tespit eder ve uygular.

@SpringBootConfiguration

- @Configuration anotasyonunun bir özelleşmesidir. Spring Boot uygulamasının konfigürasyon sınıfını belirtir.

@EnableAutoConfiguration

- Spring Boot'un otomatik konfigürasyonunu etkinleştirir. Spring Boot, sınıf yolundaki bağımlılıkları tarar ve uygun konfigürasyonları otomatik olarak uygular.

@ComponentScan

- Spring bileşenlerini otomatik olarak tarar ve uygular.

Açıklama

- **pom.xml Dosyası**: Projenin bağımlılıklarını ve yapılandırmasını içerir.
- **Ana Uygulama Sınıfı (DemoUygulamasi)**: @SpringBootApplication anotasyonu ile işaretlenmiştir ve Spring Boot uygulamasının ana giriş noktasıdır.
- **RestController Sınıfı (MerhabaController)**: Basit bir REST Controllersidir ve /merhaba URL'sine gelen GET isteklerine yanıt verir.
- **application.properties Dosyası**: Uygulama yapılandırma özelliklerini içerir; burada, sunucu portu ve uygulama adı belirtilmiştir.

Bu yapılandırma ile, kullanıcının Spring Boot uygulaması otomatik olarak yapılandırılır ve gerekli bileşenler taranarak uygulanır. Uygulama başlatıldığında, http://localhost:8081/merhaba adresine giderek "Merhaba, Dünya!" mesajı görüntülenebilir.

Actuator

Spring Boot Actuator, Spring Boot uygulamasının izlenmesi ve yönetilmesi için işlevsellik sağlar. Bu araç, üretime hazır birçok özelliği geliştiricinin kendi implementasyonlarını yapmasına gerek kalmadan sunar. Maven yapı aracı olarak kullanıldığında, spring-boot-starter-actuator bağımlılığı pom.xml dosyasında belirtilebilir.

Kod örneği:

“pom.xml”:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
```

```
    <groupId>com.ornek</groupId>
    <artifactId>demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>demo</name>
    <description>Spring Boot için örnek proje</description>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.7.0</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>

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

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

“DemoUygulamasi.java” (Ana Uygulama Sınıfı):

```
package com.ornek.demo;
```

```java
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DemoUygulamasi {

    public static void main(String[] args) {
        SpringApplication.run(DemoUygulamasi.class, args);
    }
}
```

“MerhabaController.java” (RestController Sınıfı):

```java
package com.ornek.demo;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class MerhabaController {

    @GetMapping("/merhaba")
    public String merhabaDe() {
        return "Merhaba, Dünya!";
    }
}
```

“application.properties”:

```
server.port=8081
spring.application.name=DemoUygulamasi

# Actuator endpoint'lerini etkinleştirmek için
management.endpoints.web.exposure.include=*
```

Açıklama

- **“pom.xml" Dosyası**: Gerekli bağımlılıkları içerir. spring-boot-starter-actuator bağımlılığı, Spring Boot Actuator özelliklerini sağlar.
- **Ana Uygulama Sınıfı (DemoUygulamasi.java)**: @SpringBootApplication anotasyonu ile işaretlenmiştir ve Spring Boot uygulamasının ana giriş noktasıdır.
- **RestController Sınıfı (MerhabaController)**: Basit bir REST Controllersidir ve /merhaba URL'sine gelen GET isteklerine yanıt verir.
- **“application.properties" Dosyası**: Uygulama yapılandırma özelliklerini içerir; burada, sunucu portu, uygulama adı ve Actuator endpoint'lerinin etkinleştirilmesi belirtilmiştir.

Actuator Endpoint'leri

Spring Boot Actuator, varsayılan olarak çeşitli endpoint'ler sağlar. Bu endpoint'ler uygulamanın durumu hakkında bilgi verir. Örneğin:

- /actuator/health: Uygulamanın sağlık durumu.
- /actuator/info: Uygulama hakkında genel bilgiler.
- /actuator/metrics: Uygulamanın metrikleri.
- /actuator/env: Uygulamanın ortam değişkenleri.

Kullanıcı uygulamayı başlattıktan sonra, bu endpoint'lere erişebilir ve uygulamanın durumunu izleyebilir. Örneğin, http://localhost:8081/actuator/health adresine giderek uygulamanın sağlık durumu kontrol edilebilir.

Spring Framework Modülleri ile Entegrasyon

Spring Boot, mevcut Spring Framework modülleri ile entegrasyon sağlar.

- **Spring Security**: Spring Boot, Spring Security modülü ile entegre olabilir. Entegrasyonun en basit yolu, yapılandırma dosyasında starter bağımlılığı belirtmektir. Maven kullanıldığında, pom.xml dosyasına spring-boot-starter-security bağımlılığı eklenebilir.

Kod örneği:

“pom.xml”:

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.ornek</groupId>
    <artifactId>demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>demo</name>
    <description>Spring Boot için örnek proje</description>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.7.0</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

“DemoUygulamasi.java” (Ana Uygulama Sınıfı):

```java
package com.ornek.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DemoUygulamasi {

    public static void main(String[] args) {
        SpringApplication.run(DemoUygulamasi.class, args);
    }
}
```

“GuvenlikYapilandirma.java” (Güvenlik Yapılandırma Sınıfı):

```java
package com.ornek.demo;

import org.springframework.context.annotation.Bean;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecu
rity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityCo
nfigurerAdapter;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

@EnableWebSecurity
public class GuvenlikYapilandirma extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
                .antMatchers("/merhaba").permitAll() // Bu endpoint herkes
tarafından erişilebilir
                .anyRequest().authenticated() // Diğer tüm istekler kimlik
doğrulaması gerektirir
```

```
                .and()
            .formLogin() // Form tabanlı giriş sayfası
                .and()
            .httpBasic(); // Temel kimlik doğrulama
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

“MerhabaController.java” (RestController Sınıfı):

```
package com.ornek.demo;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class MerhabaController {

    @GetMapping("/merhaba")
    public String merhabaDe() {
        return "Merhaba, Dünya!";
    }

    @GetMapping("/guvenli")
    public String guvenliBolgede() {
        return "Bu güvenli bir bölgedir!";
    }
}
```

“application.properties”:

```
server.port=8081
spring.application.name=DemoUygulamasi
```

Açıklama

- **“pom.xml" Dosyası**: Gerekli bağımlılıkları içerir. spring-boot-starter-web bağımlılığı, web uygulaması için gerekli olan Spring bileşenlerini sağlar. spring-boot-starter-security bağımlılığı ise Spring Security özelliklerini ekler.
- **Ana Uygulama Sınıfı (DemoUygulamasi.java)**: @SpringBootApplication anotasyonu ile işaretlenmiştir ve Spring Boot uygulamasının ana giriş noktasıdır.
- **Güvenlik Yapılandırma Sınıfı (GuvenlikYapilandirma.java)**: @EnableWebSecurity anotasyonu ile işaretlenmiştir ve WebSecurityConfigurerAdapter sınıfından miras alır. Bu sınıf, HTTP güvenlik yapılandırmasını sağlar. /merhaba endpoint'ine herkese açık erişim izni verilirken, diğer tüm istekler kimlik doğrulaması gerektirir.
- **RestController Sınıfı (MerhabaController.java)**: İki endpoint sağlar: /merhaba (herkese açık) ve /guvenli (kimlik doğrulaması gerektiren).
- **“application.properties" Dosyası**: Uygulama yapılandırma özelliklerini içerir; burada, sunucu portu ve uygulama adı belirtilmiştir.

Bu yapılandırma ile, Spring Boot uygulaması güvenlik özellikleri ile donatılmış olur. “http://localhost:8081/merhaba” adresine kimlik doğrulaması yapmadan erişebilirken, “http://localhost:8081/guvenli” adresine erişmek için kimlik doğrulaması yapılması gerekecektir. Spring Security, uygulamanın güvenliğini sağlamak için form tabanlı giriş ve temel kimlik doğrulama gibi çeşitli mekanizmaları otomatik olarak entegre eder.

Uygulama Sunucuları

Spring Boot, varsayılan olarak yerleşik web sunucuları (örneğin Tomcat) sağlar. Ancak, Spring Boot ayrıca bağımsız bir WildFly uygulama sunucusunda WAR dosyası olarak da dağıtılabilir. Maven kullanıldığında, oluşturulan WAR dosyasının otomatik olarak dağıtılmasını sağlayan wildfly-maven-plugin Maven eklentisi bulunmaktadır.

Kod örneği:

“pom.xml”:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.ornek</groupId>
    <artifactId>demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>war</packaging> <!-- War dosyası olarak paketleme -->

    <name>demo</name>
    <description>Spring Boot WildFly için örnek proje</description>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.7.0</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-tomcat</artifactId>
            <scope>provided</scope> <!-- WildFly içinde Tomcat bağımlılığını
sağlar -->
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>3.2.3</version>
                <configuration>
                    <failOnMissingWebXml>false</failOnMissingWebXml>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.wildfly.plugins</groupId>
                <artifactId>wildfly-maven-plugin</artifactId>
                <version>2.0.2.Final</version>
                <configuration>
```

```
                    <wildfly-home>/path/to/wildfly</wildfly-home> <!-- WildFly
kurulum yolu -->
                    <hostname>localhost</hostname>
                    <port>9990</port>
                    <username>admin</username>
                    <password>password</password>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

“DemoUygulamasi.java” (Ana Uygulama Sınıfı):

```
package com.ornek.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.builder.SpringApplicationBuilder;
import
org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

@SpringBootApplication
public class DemoUygulamasi extends SpringBootServletInitializer {

    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder
application) {
        return application.sources(DemoUygulamasi.class);
    }

    public static void main(String[] args) {
        SpringApplication.run(DemoUygulamasi.class, args);
    }
}
```

“MerhabaController.java” (RestController Sınıfı):

```java
package com.ornek.demo;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class MerhabaController {

    @GetMapping("/merhaba")
    public String merhabaDe() {
        return "Merhaba, Dünya!";
    }
}
```

“application.properties”:

```
spring.application.name=DemoUygulamasi
```

Açıklama

- **“pom.xml" Dosyası**: Uygulamanızın bir WAR dosyası olarak paketlenmesini sağlar. spring-boot-starter-tomcat bağımlılığı provided scope ile belirtilir, böylece Tomcat bağımlılığı WildFly içinde sağlanır. wildfly-maven-plugin, WildFly sunucusuna WAR dosyasını otomatik olarak dağıtmak için kullanılır.
- **Ana Uygulama Sınıfı (DemoUygulamasi.java)**: SpringBootServletInitializer sınıfını genişletir ve configure metodunu override eder. Bu, Spring Boot uygulamasının bir WAR dosyası olarak dağıtılmasını sağlar.
- **RestController Sınıfı (MerhabaController.java)**: Basit bir REST Controllersidir ve /merhaba URL'sine gelen GET isteklerine yanıt verir.
- **“application.properties” Dosyası**: Uygulama adı belirtilmiştir.

Dağıtım

Kullanıcı, uygulamayı WildFly sunucusuna dağıtmak için aşağıdaki adımları izlemelidir:

1. WildFly kurulum dizininizi belirtin (wildfly-home).
2. WildFly yönetici kullanıcı adını ve şifresini belirtin (username ve password).
3. Maven komutunu çalıştırarak WAR dosyasını paketleyin ve WildFly sunucusuna dağıtın:

mvn clean package wildfly:deploy

Bu komut, WAR dosyasını oluşturur ve belirtilen WildFly sunucusuna otomatik olarak dağıtır. WildFly sunucusu çalışıyorsa, uygulama belirtilen URL'de erişilebilir olacaktır.

Kontrolün Tersine Çevrilmesi (Inversion of Control/IOC) Nedir?

Yazılım mühendisliğinde, kontrolün tersine çevrilmesi (Inversion of Control/IOC), bir bilgisayar programının özel olarak yazılmış bölümlerinin genel bir çerçeveden kontrol akışını alması için bir tasarım prensibidir. "Inversion" terimi tarihseldir: bu tasarıma sahip bir yazılım mimarisi, prosedürel programlamaya kıyasla kontrolü tersine çevirir. Prosedürel programlamada, bir programın özel kodu, genel görevleri yerine getirmek için yeniden kullanılabilir kütüphaneleri çağırır, ancak kontrolün tersine çevrilmesi ile, çerçeve özel kodu çağırır.

Kontrolün tersine çevrilmesi (IOC), GUI ortamlarının yükselişinden beri uygulama geliştirme çerçeveleri tarafından yaygın bir şekilde kullanılmıştır ve hem GUI ortamlarında hem de web sunucusu uygulama çerçevelerinde kullanılmaya devam edilmektedir. Kontrolün tersine çevrilmesi, çerçevenin, uygulama programcısı tarafından tanımlanan yöntemlerle çerçeveyi genişletmesini sağlar.

Olay odaklı programlama genellikle IOC kullanılarak uygulanır, böylece özel kod sadece olayları işlemekle ilgilenmek zorundadır, olay döngüsü ve olayların, mesajların dağıtımı çerçeve veya çalışma zamanı ortamı tarafından işlenir. Web sunucusu uygulama çerçevelerinde, dağıtım genellikle yönlendirme olarak adlandırılır ve işleyicilere endpointdenir.

"Kontrolün tersine çevrilmesi" ifadesi, ayrıca Java programcı topluluğunda "IOC konteynerleri" ile birlikte bağımlılıkların enjekte edilme desenlerini belirtmek için

ayrıca kullanılmıştır. Bu farklı anlamda, "kontrolün tersine çevrilmesi", çerçevenin, uygulama nesneleri tarafından kullanılan bağımlılıkların uygulanmasını kontrol etmesine atıfta bulunur ve orijinal anlamı olan çerçevenin kontrol akışını (uygulama kodunun yürütme zamanı kontrolüne, örneğin: geri çağrılar) kontrol etmesine değil.

Bir örnekle, geleneksel programlama ile, bir uygulamanın ana işlevi, kullanılabilir komutların bir listesini görüntülemek için bir menü kitaplığına işlev çağrıları yapabilir ve kullanıcının birini seçmesini isteyebilir. Kitaplık bu durumda işlev çağrısının değerini seçilen seçenek olarak döndürecektir ve ana işlev bu değeri ilişkili komutu yürütmek için kullanır. Bu tarz metin tabanlı arayüzlerde yaygındı. Örneğin, bir e-posta istemcisi, yeni postaları yüklemek, mevcut postaya cevap vermek, yeni posta oluşturmak vb. için komutlar içeren bir ekran gösterebilir ve program yürütme, kullanıcının bir tuşa basarak bir komut seçmesini bekleyene kadar engellenirdi.

Öte yandan, kontrolün tersine çevrilmesi ile, program, pencere sistemleri, menülerin kontrolü, fareyi kontrol etme vb. gibi yaygın davranışsal ve grafiksel unsurları bilen bir yazılım çerçevesi kullanılarak yazılabilir. Özel kod, çerçevenin gereksinimlerini karşılar, örneğin bir menü öğeleri tablosunu sağlar ve her öğe için bir kod alt rutini kaydeder, ancak kullanıcının eylemlerini izleyen ve bir menü öğesi seçildiğinde alt rutini çağıran çerçeve olacaktır. E-posta istemcisi örneğinde, çerçeve klavye ve fare girişlerini takip edebilir ve kullanıcı tarafından herhangi bir şekilde çağrılan komutları izleyebilir ve aynı anda ağ arayüzünü izleyebilir, yeni mesajların gelip gelmediğini bulabilir ve bazı ağ etkinliği algılandığında ekranı yenileyebilir. Aynı çerçeve, bir elektronik tablo programı veya bir metin düzenleyici için iskelet olarak kullanılabilir. Tersine, çerçeve Web tarayıcıları, elektronik tablolar veya metin düzenleyiciler hakkında hiçbir şey bilmez; bunların işlevselliğini uygulamak özel koda bağlıdır.

Kontrolün tersine çevrilmesi (IOC), yeniden kullanılabilir kodun ve problem özgü kodun, bir uygulamada birlikte çalışsalar da bağımsız olarak geliştirildiği güçlü bir anlam taşır. Geri çağrılar, zamanlama yöneticileri, olay döngüleri ve şablon yöntemi, kontrolün tersine çevrilmesi ilkesini takip eden tasarım kalıplarına örnektir, ancak terim genellikle nesne yönelimli programlama bağlamında kullanılır. (Bağımlılık enjeksiyonu, Java çerçevelerinin popülerleştirdiği "bağımlılıkların uygulamalar tarafından kullanılan uygulamaların denetimi üzerindeki kontrolü tersine çevirme" ayrı, belirli bir fikrin örneğidir.)

Kontrolün tersine çevrilmesi bazen "Hollywood Prensibi” olarak da adlandırılır: 'Bizi arama, biz seni ararız'.

Arka Plan

Kontrolün tersine çevrilmesi (IOC) bilgisayar biliminde yeni bir terim değildir. Martin Fowler, terimin etimolojisini 1988'e kadar götürüyor, ancak Michael Jackson'ın 1970'lerdeki Jackson Yapısal Programlama metodolojisinde tanımladığı program tersine çevirme kavramıyla yakından ilişkilidir. Bir alttan yukarı çözümleyici, bir üstten aşağı çözümleyiciyi tersine çevrilmiş olarak görülebilir: birinde kontrol çözümleyicideyken, diğerinde alıcı uygulamada bulunur.

Terim, Michael Mattsson tarafından bir tezde (uygulama kodunun çerçeve kodunu çağırması yerine) kullanılmış ve daha sonra Stefano Mazzocchi tarafından oradan alınmıştır ve 1999'da eski bir Apache Yazılım Vakfı projesi olan Avalon'da popülerleştirilmiştir, burada bir üst nesnenin ek olarak bir çocuk nesnenin bağımlılıklarını kontrol etmesine atıfta bulunmuştur. Terim, 2004'te Robert C. Martin ve Martin Fowler tarafından daha da popüler hale getirildi. Fowler, terimin kökenlerini 1980'lere kadar götürüyor.

Tanım

Geleneksel programlamada, iş mantığının akışı, birbirine statik olarak bağlı olan nesneler tarafından belirlenir. Kontrolün tersine çevrilmesi ile, akış, program yürütme sırasında oluşturulan nesne grafiğine bağlıdır. Böyle bir dinamik akış, soyutlamalar aracılığıyla tanımlanan nesne etkileşimleri tarafından mümkün hale getirilir. Bu çalışma zamanı bağlama, bağımlılık enjeksiyonu veya bir hizmet yerelizatörü gibi mekanizmalar aracılığıyla sağlanır. IOC'da, kod statik olarak derleme sırasında da bağlanabilir, ancak kodu doğrudan referansla değil, dış yapılandırmadan açıklamasını okuyarak yürütmesi için bulur.

Bağımlılık enjeksiyonunda, bir bağımlı nesne veya modül, çalışma zamanında ihtiyaç duyduğu nesneye bağlıdır. Program yürütme sırasında hangi belirli nesnenin bağımlılığı karşılayacağı genellikle statik analiz kullanılarak derleme sırasında bilinemez. Burada nesne etkileşimi terimleriyle tanımlansa da, ilke, nesne yönelimli programlamadan başka programlama metodolojilerine de uygulanabilir.

Çalışan programın nesneleri birbirine bağlanabilmesi için nesnelerin uyumlu arayüzlere sahip olmaları gerekir. Örneğin, sınıf A, davranışı arayüz I'ye delegeler, bu arayüz sınıf B tarafından uygulanır; program A ve B'yi örnekler ve sonra B'yi A'ya enjekte eder.

Kullanım

1) XDE için Mesa Programlama ortamı, 1985
2) Visual Basic (klasik), 1991
3) HTML DOM olayları
4) Spring Framework
5) ASP.NET Core
6) Şablon yöntemi deseni

Kod örneği:

HTML DOM olayları

Web tarayıcıları HTML'deki DOM olayları için kontrolün tersine çevrilmesini uygular. Uygulama geliştiricisi, bir geri çağrıyı kaydetmek için document.addEventListener() kullanır.

```
<!doctype html>
<html lang="tr">
<head>
     <meta charset="utf-8">
     <title>DOM Seviye 2</title>
</head>
<body>
     <h1>DOM Seviye 2 Olay İşleyicisi</h1>
     <p><large><span id="output"></span></large></p>

     <script>
          var kayitliOlayDinleyici = function () {
               document.getElementById("output").innerHTML = "<large>Kayıtlı
dinleyici çağrıldı.</large>";
```

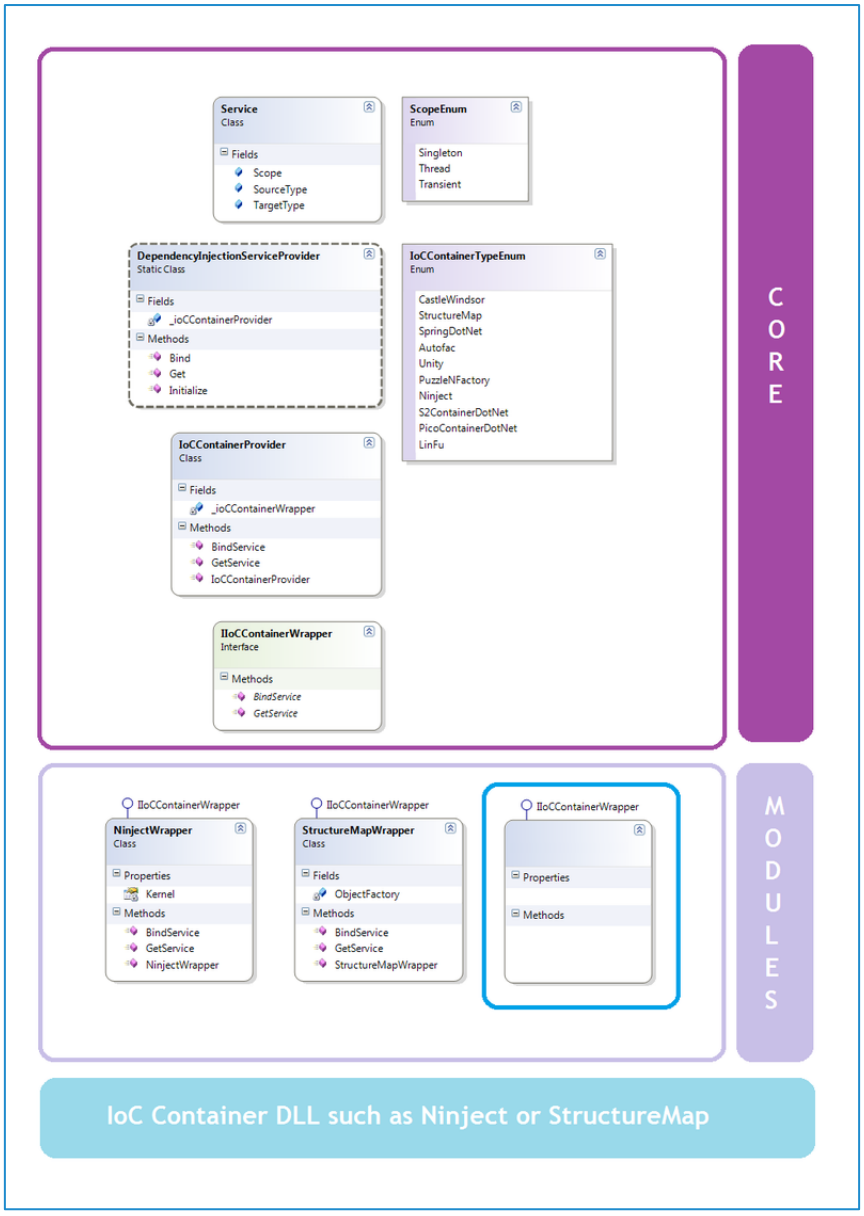
```
          }
          document.addEventListener( "click", kayitliOlayDinleyici, true );

          document.getElementById("output").innerHTML = "<large>Olay
işleyicisi kaydedildi. Sayfaya tıklarsanız, web tarayıcınız olay işleyicisini
çağıracaktır.</large>"
     </script>
</body>
</html>
```

Web Uygulama Çerçeveleri

Bu kod örneği, ASP.NET Core web uygulaması için bir web uygulaması ana bilgisayarını oluşturur, bir endpoint'i kaydeder ve ardından kontrolü çerçeveye aktarır.

```
var builder = WebApplication.CreateBuilder(args);

var uygulama = builder.Build();

uygulama.MapGet("/", () => "Merhaba Dünya!");

uygulama.Run();
```

Bağımlılık Enjeksiyonu (Dependency Injection) Nedir?



Bağımlılık enjeksiyonu, program bileşimini kolaylaştırmak için genellikle 'kapsayıcılar' (containers) olarak bilinen özel çerçevelerle birlikte kullanılır.

Yazılım mühendisliğinde, bağımlılık enjeksiyonu, bir nesnenin veya işlevin, onun içsel olarak oluşturulması yerine gereksinim duyduğu diğer nesneleri veya işlevleri alması için bir programlama tekniğidir. Bağımlılık enjeksiyonu, nesnelerin oluşturulması ve kullanılmasıyla ilgili kaygıları ayırmayı amaçlar, böylece gevşek bir şekilde bağlı programlara yol açar. Kalıp, bir nesnenin veya işlevin belirli bir hizmeti kullanmak istemesi durumunda, bu hizmetleri nasıl oluşturacağını bilmesi gerekmeyeceğini sağlar. Bunun yerine, alıcı 'istemci'ye (nesne veya işlev), bunun farkında olmadığı dış kodlar (bir 'enjektör') tarafından bağımlılıkları sağlanır. Bağımlılık enjeksiyonu, örtük bağımlılıkları açık hale getirir ve aşağıdaki sorunları çözmeye yardımcı olur:

- Bir sınıf, bağımlı olduğu nesnelerin oluşturulmasından bağımsız olabilir mi?
- Bir uygulama ve kullandığı nesneler farklı yapılandırmaları destekleyebilir mi?

Bağımlılık enjeksiyonu genellikle bağımlılık tersine çevirme ilkesiyle uyumlu olacak şekilde kodu tutmak için kullanılır.

Bağımlılık enjeksiyonu, statik tipli dillerde, bir istemcinin yalnızca kullandığı hizmetlerin arabirimlerini bildirmesi gerektiği anlamına gelir, somut uygulamalarını değil. Bu, çalışma zamanında hangi hizmetlerin kullanılacağını değiştirmeyi daha kolay hale getirir ve yeniden derleme yapmadan mümkün kılar.

Uygulama çerçeveleri genellikle bağımlılık enjeksiyonunu kontrolün tersine çevirme ile birleştirir. Kontrolün tersine çevrilmesiyle, çerçeve önce bir nesneyi (örneğin bir Controller) oluşturur ve ardından ona kontrol akışını iletir. Bağımlılık enjeksiyonu ile, çerçeve ayrıca uygulama nesnesi tarafından (genellikle yapıcı yöntemin parametrelerinde) bildirilen bağımlılıkları oluşturur ve nesnelere bağımlılıkları iletilir.

Bağımlılık enjeksiyonu, "bağımlılıkların uygulanımlarının kontrolünü tersine çevirme" fikrini uygular, bu nedenle bazı Java çerçeveleri kavramı genellikle "kontrolün tersine çevrilmesi" olarak adlandırır (akışın tersine çevrilmesi ile karıştırılmamalıdır).

Beş yaşındaki çocuklar için bağımlılık enjeksiyonu

Buzdolabından kendi başlarına şeyler alıp çıkardıklarında, sorunlar çıkarabilirler. Kapıyı açık bırakabilirler, anne veya babanın istemediği bir şeyi alabilirler. Hatta dolapta olmayan veya son kullanma tarihi geçmiş bir şey arıyor olabilirler.

Yapmaları gereken, "Öğle yemeğiyle birlikte bir şey içmek istiyorum" gibi bir ihtiyacı belirtmek ve sonrasında zaten onların içeceği şeyi öğle yemeğine oturduklarında biz onlara verebiliriz.

John Munsch, 28 Ekim 2009.

Bağımlılık enjeksiyonu dört rol içerir: **hizmetler, istemciler, arabirimler ve enjektörler.**

Hizmetler ve istemciler: Bir hizmet, kullanışlı işlevselliği içeren herhangi bir sınıftır. Bunun karşılığında, bir istemci, hizmetleri kullanan herhangi bir sınıftır. Bir istemcinin gereksinim duyduğu hizmetler, istemcinin bağımlılıklarıdır.

Herhangi bir nesne bir hizmet veya istemci olabilir; adlar sadece nesnelerin enjeksiyondaki rolüyle ilgilidir. Aynı nesne hatta hem bir istemci (enjekte edilmiş hizmetleri kullanır) hem de bir hizmet (diğer nesnelere enjekte edilir) olabilir. Enjeksiyon sırasında, hizmet istemcinin durumunun bir parçası haline getirilir ve kullanılmak üzere hazır hale gelir.

Arayüzler: İstemciler, bağımlılıklarının nasıl uygulandığını değil, yalnızca adlarını ve API'yi bilirler. Örneğin, e-postaları almak için bir hizmet, arka planda IMAP veya POP3 protokollerini kullanabilir, ancak bu ayrıntı, yalnızca bir e-posta alınmasını isteyen çağrı kodu için ilgisiz olabilir. Uygulama detaylarını yoksayarak, istemciler bağımlılıkları değiştiğinde değişmek zorunda kalmazlar.

Enjektörler: Enjektör, bazen bir araya getirici, konteyner, sağlayıcı veya fabrika olarak da adlandırılır, hizmetleri istemciye tanıtır.

Enjektörlerin rolü karmaşık nesne grafiklerini oluşturmak ve bağlamaktır, nesneler hem istemci hem de hizmet olabilir. Enjektör kendisi birçok nesnenin bir araya gelerek çalışabileceği, ancak istemci olmaması gereken bir nesne olmalıdır, çünkü bu durumda döngüsel bir bağımlılık oluşturur.

Bağımlılık enjeksiyonu, nesnelerin nasıl oluşturulduğunu nesnelerin nasıl kullanıldığından ayırdığı için, çoğu nesne tabanlı dillerde bulunan new anahtar kelimesinin öneminin azalmasına neden olur. Çünkü çerçeve hizmetleri oluşturmayı ele alırken, programcı genellikle yalnızca programın alanında varlık temsil eden değer nesnelerini doğrudan oluşturur (örneğin, bir iş uygulamasındaki bir “Çalışan“ nesnesi veya bir alışveriş uygulamasındaki bir “Sipariş” nesnesi).

Analoji: Bir benzetme olarak, arabalar insanları bir yerden bir yere taşımanın yararlı işlevini yerine getiren hizmetler olarak düşünülebilir. Araba motorları benzin, dizel veya elektrik gerektirebilir, ancak bu ayrıntı sürücü için önemli değildir - sürücü, sadece onları hedeflerine götürebilip götüremeyeceklerine bakar.

Arabalar, pedalları, direksiyon tekerlekleri ve diğer kontroller aracılığıyla birbirine benzer bir arabirim sunar. Bu nedenle, montaj hattında hangi motorun "enjekte" edildiği artık önemli değildir ve sürücüler istedikleri gibi arabalar arasında geçiş yapabilirler.

Avantajlar ve Dezavantajlar

Avantajlar

Bağımlılık enjeksiyonunun temel bir faydası, sınıfların ve bağımlılıklarının arasındaki azaltılmış bağlantıdır.

Bir istemcinin bağımlılıklarının nasıl uygulandığını bilmediği için, programlar daha yeniden kullanılabilir, test edilebilir ve bakımı yapılabilir hale gelir.

Bu ayrıca esnekliği artırır: bir istemci, içsel arabirimi destekleyen herhangi bir şey üzerinde hareket edebilir.

Daha genel olarak, bağımlılık enjeksiyonu, herhangi bir bağımlılığın oluşturulmasının tek bir bileşen tarafından işlenmesi nedeniyle gereksiz kodu azaltır.

Son olarak, bağımlılık enjeksiyonu, eş zamanlı gelişimi sağlar. İki geliştirici, birbirlerini kullanacak sınıfları bağımsız olarak geliştirebilir, ancak yalnızca sınıfların iletişim kuracakları arabirimi bilmeleri gerekir. Eklentiler genellikle, asıl ürünün geliştiricileriyle hiç konuşmayan üçüncü taraflar tarafından geliştirilir.

Test Bağımlılık enjeksiyonunun birçok faydası, özellikle birim testleri için son derece önemlidir.

Örneğin, bağımlılık enjeksiyonu, bir sistemin yapılandırma detaylarını yapılandırma dosyalarına harici olarak dışsallaştırarak, sistemin yeniden yapılandırılmasını yeniden derleme yapmadan sağlar. Farklı durumlar için farklı bileşen uygulamaları gerektiren ayrı yapılandırmalar yazılabilir.

Benzer şekilde, bağımlılık enjeksiyonu kod davranışında herhangi bir değişiklik gerektirmediğinden, bu, bir refactoring olarak mevcut kodlara uygulanabilir. Bu, istemcilerin daha bağımsız hale gelmesini sağlar ve test edilmesi daha kolay hale gelir, test edilmeyen veya mock nesneleri kullanan izole birim testleri kullanarak.

Bu test kolaylığı, bağımlılık enjeksiyonunu kullanırken fark edilen ilk faydadır.

Dezavantajlar

Bağımlılık enjeksiyonunun eleştirmenleri şunları iddia eder:

- Açık varsayılanlar mevcut olduğunda, istemcilerin yapılandırma ayrıntılarını talep eden istemciler oluşturur, bu da yük oluşturabilir.
- Kodun davranışını yapının oluşturulmasından ayırarak, kodun izlenmesini zorlaştırır.
- Genellikle yansıma veya dinamik programlama ile uygulanır, bu da IDE otomasyonunu engeller.
- Genellikle daha fazla ön geliştirme çabası gerektirir.
- Bir çerçeveye bağımlılığı teşvik eder.

Bağımlılık Enjeksiyonunun Türleri

Bir istemcinin enjekte edilmiş hizmetleri alabileceği üç temel yol vardır:

- Yapıcı enjeksiyon, bağımlılıkların bir istemcinin sınıf yapıcısı aracılığıyla sağlandığı yerdir.
- Ayarlayıcı enjeksiyon, istemci bir yapılandırıcı yöntem aracılığıyla bağımlılıkları kabul ettiğinde istemci, enjektörlerin istemciyi kullanılmadan önce tüm bağımlılıkların enjekte edildiğinden ve geçerli olduğundan emin olmasını sağlar.
- Arabirim enjeksiyonu, bağımlılığın arabirimi bir enjektör yöntemi sağladığında, bu yöntem bağımlılığını herhangi bir istemciye enjekte edecek şekilde tasarlanmıştır.

Bağımlılık enjeksiyonu olmadan

Aşağıdaki Java örneğinde, "Musteri" sınıfı, yapıcı metotta başlatılan bir "Servis" üye değişkenini içerir. İstemci, doğrudan kodlanmış bir bağımlılık oluşturarak hangi hizmeti kullandığını doğrudan oluşturur ve kontrol eder.

```
public class Musteri {
    private Servis servis;

    Musteri() {
        // Bağımlılık sabit olarak tanımlanmıştır.
        this.servis = new OrnekServis();
    }
}
```

Yapıcı Metot (Constructor) Enjeksiyonu

Bağımlılık eklemenin en yaygın biçimi, bir sınıfın bağımlılıklarını yapıcı metot aracılığıyla talep etmesidir. Bu, gerekli bağımlılıklar olmadan örneklenemeyeceği için istemcinin her zaman geçerli bir durumda olmasını sağlar.

```
public class Musteri {
    private Servis servis;

    // Bağımlılık bir yapıcı metot aracılığıyla enjekte edilir.
    Musteri(Servis servis) {
        if (servis == null) {
            throw new IllegalArgumentException("servis null olmamalidir");
        }
        this.servis = servis;
    }
}
```

Ayarlayıcı (Setter) Enjeksiyonu

Bağımlılıkları bir yapıcı metot yerine ayarlayıcı bir yöntemle kabul ederek istemciler enjektörlerin bağımlılıklarını istedikleri zaman değiştirmelerine izin verebilir. Bu esneklik sunar ancak istemci kullanılmadan önce tüm bağımlılıkların enjekte edildiğinden ve geçerli olduğundan emin olmayı zorlaştırır.

```
public class Musteri {
    private Servis servis;

    // Bağımlılık bir ayarlayıcı yöntem aracılığıyla enjekte edilir.
    public void servisAyarla(Servis servis) {
        if (servis == null) {
            throw new IllegalArgumentException("servis null olmamalidir");
        }
        this.servis = servis;
    }
}
```

Arayüz (Interface) Enjeksiyonu

Arayüz enjeksiyonu ile bağımlılıklar istemcilerinden tamamen habersizdir, ancak yine de yeni istemcilere referans gönderip alırlar.

Bu şekilde bağımlılıklar enjektör haline gelir. Önemli olan, enjeksiyon yönteminin bir arayüz aracılığıyla sağlanmasıdır.

İstemciyi ve bağımlılıklarını tanıtmak için hala bir montajcıya ihtiyaç vardır. Birleştirici, istemciye bir referans alır, onu bu bağımlılığı ayarlayan ayarlayıcı arayüzüne aktarır ve onu, kendisine bir referansı istemciye geri ileten bağımlılık nesnesine iletir.

Arayüz enjeksiyonunun bir değere sahip olması için, bağımlılığın kendisine bir referansı geri iletmenin yanı sıra bir şeyler yapması gerekir. Bu, diğer bağımlılıkları çözmek için bir fabrika veya alt montajcı gibi davranabilir, böylece bazı ayrıntıları ana montajcıdan soyutlayabilir. Bağımlılığın onu kaç istemcinin kullandığını bilmesi için referans sayma olabilir. Bağımlılık bir istemci koleksiyonunu sürdürüyorsa, daha sonra bunların hepsine kendisinin farklı bir örneğini enjekte edebilir.

```
public interface ServisAyarlayici {
    void servisAyarla(Servis servis);
}

public class Musteri implements ServisAyarlayici {
    private Servis servis;

    @Override
    public void servisAyarla(Servis servis) {
        if (servis == null) {
            throw new IllegalArgumentException("servis null olmamalidir");
        }
        this.servis = servis;
    }
}

public class ServisEnjektoru {
    private final Set<ServisAyarlayici> musteriler = new HashSet<>();

    public void enjekteEt(ServisAyarlayici musteri) {
        this.musteriler.add(musteri);
        musteri.servisAyarla(new OrnekServis());
```

```
    }

    public void degistir() {
        for (Musteri musteri : this.musteriler) {
            musteri.servisAyarla(new BaskaOrnekServis());
        }
    }
}

public class OrnekServis implements Servis {}

public class BaskaOrnekServis implements Servis {}
```

Montaj (Assembly)

Bağımlılık eklemeyi uygulamanın en basit yolu, hizmetleri ve istemcileri manuel olarak düzenlemektir; bu genellikle yürütmenin (execution) başladığı programın kökünde gerçekleştirilir.

```
public class Program {

    public static void main(String[] args) {
        // Servisi oluştur.
        Servis servis = new OrnekServis();

        // Servisi müşteriye enjekte et.
        Musteri musteri = new Musteri(servis);

        // Nesneleri kullan.
        System.out.println(musteri.selamla());
    }
}
```

Çerçeveler

Tabanlar, Ninject veya StructureMap gibi konteynırlar (containers), Bağımlılık Enjeksiyonu ve kontrolün tersine çevrilmesini (Inversion of Control/IOC) elde etmek için nesne yönelimli programlama dillerinde yaygın olarak kullanılır. Geniş projeler için manuel bağımlılık enjeksiyonu sık sık zahmetli ve hatalı olabilir, bu da işlemi otomatikleştiren çerçevelerin kullanımını teşvik eder. Manuel bağımlılık enjeksiyonu, inşa kodu uygulamaya özgü olmadığında ve genel hale geldiğinde bir bağımlılık enjeksiyonu çerçevesine dönüşür. Bu araçlar faydalı olmakla birlikte, bağımlılık enjeksiyonunu gerçekleştirmek için gerekli değildir.

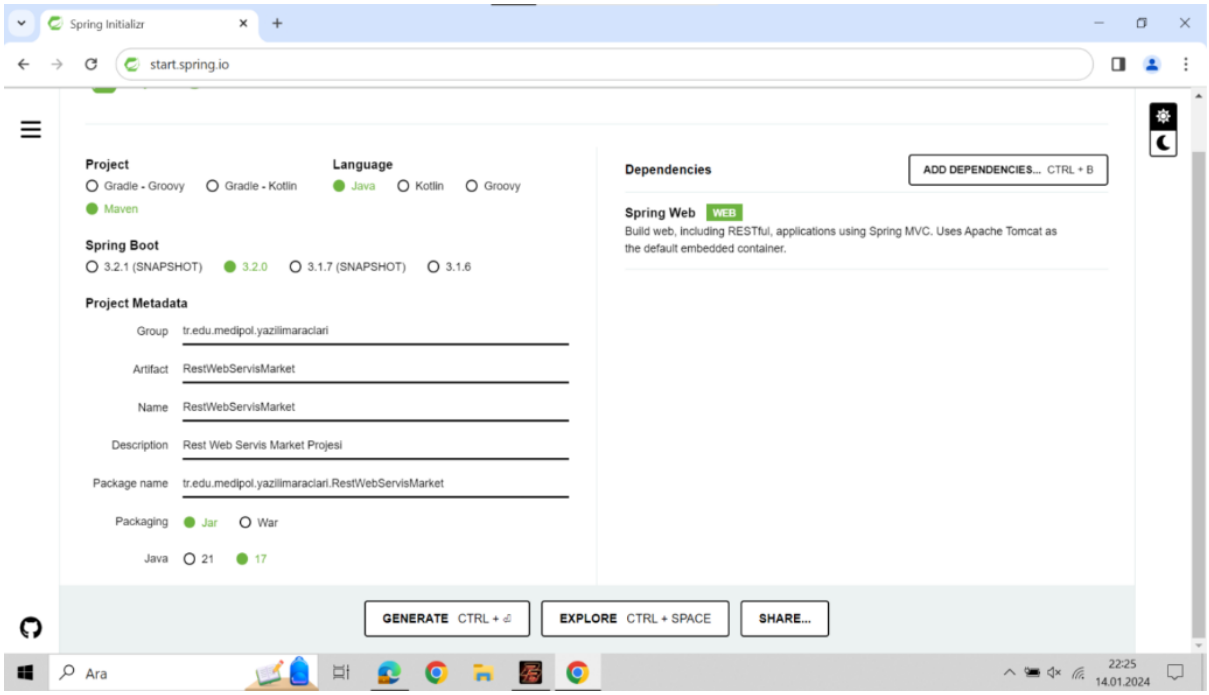
Spring gibi bazı çerçeveler, program bileşimini planlamak için harici yapılandırma dosyalarını kullanabilir:

```
import org.springframework.beans.factory.BeanFactory;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Enjektor {

    public static void main(String[] args) {
        // Hangi somut servisin kullanılacağına dair ayrıntılar, programdan
ayrı olarak yapılandırmada saklanır.
        BeanFactory beanFactory = new
ClassPathXmlApplicationContext("Beans.xml");
        Musteri musteri = (Musteri) beanFactory.getBean("musteri");
        System.out.println(musteri.selamla());
    }
}
```

Potansiyel olarak uzun ve karmaşık bir nesne grafiği ile bile kodda belirtilen tek sınıf bu durumda “Musteri” sınıfıdır. “Musteri”, Spring ile çalışmak için herhangi bir değişiklik geçirmemiş ve hala bir POJO olarak kalır. Spring'e özgü açıklamaları ve çağrıları birçok sınıf arasında yayılmadan tutarak sistem yalnızca Spring'e gevşek bir bağımlılık gösterir.

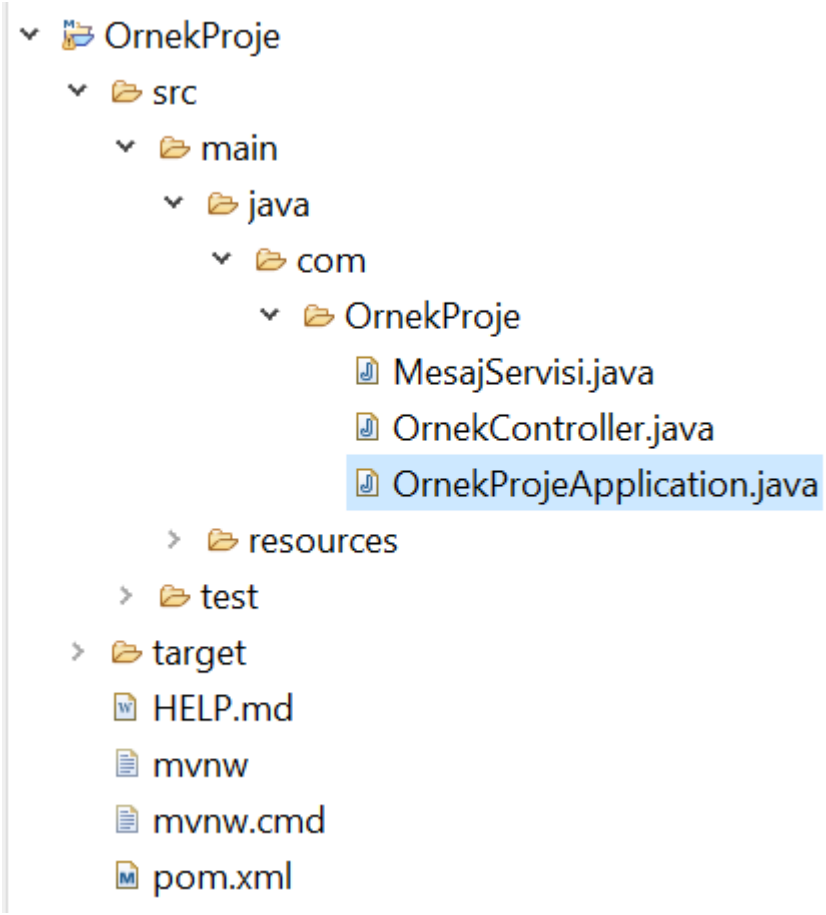
Spring Boot ile Adım Adım Proje Oluşturma - Örnek Proje – Mesaj Servisi (Inversion of Control ve Dependency Injection Prensipleriyle)

1- Projeyi oluşturmak için (https://start.spring.io/) adresine giderek gerekli ayarları yapıyoruz. Proje tipi olarak Maven tercih ediyoruz. Proje adını, paket adını ve Java sürümünü belirleyip projeye "Spring Web" bağımlılığını (dependency) ekliyoruz.



2- Sonrasında “GENERATE” butonuna basarak bilgisayarımıza indirdiğimiz zip dosyasının içinden belirttiğimiz proje ismindeki klasörü çıkarıp kullandığımız IDE üzerinden bu klasörü import ediyoruz. (Sizin belirlediğiniz proje adına göre zip dosyasının ismi farklılık gösterebilir.)

3- **Proje yapısı:** Projede oluşturacağımız sınıfların ve “pom.xml” dosyasının dizini aşağıdaki şekilde olmalıdır. (Ekran görüntüsü Eclipse IDE üzerinden alınmıştır.)



4- Aşağıdaki kod örneğindeki gibi bir RestController sınıfı oluşturuyoruz.

“OrnekController.java”:

```
package com.ornek.ornekproje.controller;

import com.ornek.ornekproje.service.MesajServisi;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
```

```
public class OrnekController {

    private final MesajServisi mesajServisi;

    @Autowired
    public OrnekController(MesajServisi mesajServisi) {
        this.mesajServisi = mesajServisi;
    }

    @GetMapping("/mesaj")
    public String mesajGetir() {
        return mesajServisi.mesajOlustur();
    }
}
```

5- Aşağıdaki kod örneğindeki gibi bir “MesajServisi” sınıfı oluşturuyoruz.

“MesajServisi.java”:

```
package com.ornek.ornekproje.service;

import org.springframework.stereotype.Service;

@Service
public class MesajServisi {

    public String mesajOlustur() {
        return "Merhaba, Spring Boot ile Dependency Injection örneği!";
    }
}
```

6- Aşağıdaki kod örneğindeki gibi ana sınıfı düzenliyoruz.

“OrnekProjeApplication.java”:

```
package com.ornek.ornekproje;

import org.springframework.boot.SpringApplication;
```

```java
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class OrnekProjeApplication {

    public static void main(String[] args) {
        SpringApplication.run(OrnekprojeApplication.class, args);
    }
}
```

Uygulamayı çalıştırmak için “OrnekProjeApplication” sınıfını çalıştırıyoruz. Spring Boot uygulaması başlayacaktır. Tarayıcıda “http://localhost:8080/mesaj” adresine gidince şu mesajı görüntülüyoruz: "Merhaba, Spring Boot ile Dependency Injection örneği!".

IOC ve DI Açıklaması

Inversion of Control (IOC): Uygulama bileşenlerinin kontrolünü tersine çevirir. Bu, framework'ün (Spring) uygulama kodunu yönetmesi anlamına gelir. Kontrol tersine çevrilir çünkü uygulama framework'ü çağırmaz, framework uygulama kodunu çağırır.

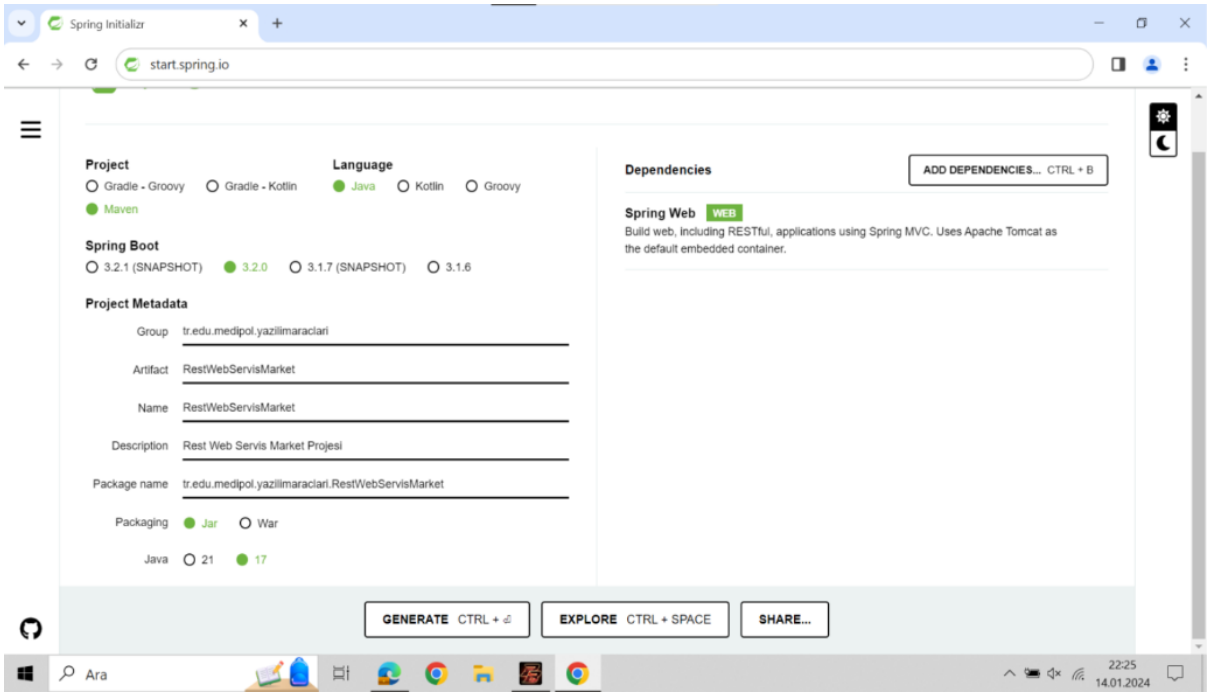
Dependency Injection (DI): Bir sınıfın bağımlılıklarını dışarıdan sağlar. Bu, nesne oluşturma ve bağımlılıkları enjekte etme işleminin framework tarafından yapıldığı anlamına gelir. Yukarıdaki örnekte, “`OrnekController`” sınıfına “`MesajServisi`” enjekte edilir.

Spring, DI'yi gerçekleştirmek için çeşitli yollar sunar, en yaygın olanı ise “`@Autowired`” anotasyonudur. Spring'in ihtiyaç duyulan bağımlılığı otomatik olarak enjekte etmesini sağlar.

Spring Boot ile Adım Adım Proje Oluşturma - Market Rest Web Servis (Inversion of Control ve Dependency Injection Prensipleriyle)

NOT: "Konu sonunda örnek küçük bir projeye yer verilmelidir. Bu proje 10 sayfalık konu anlatımının içinde değildir" talimatı doğrultusunda yapılmış bir örnek "Market REST Web Servis" projesidir.

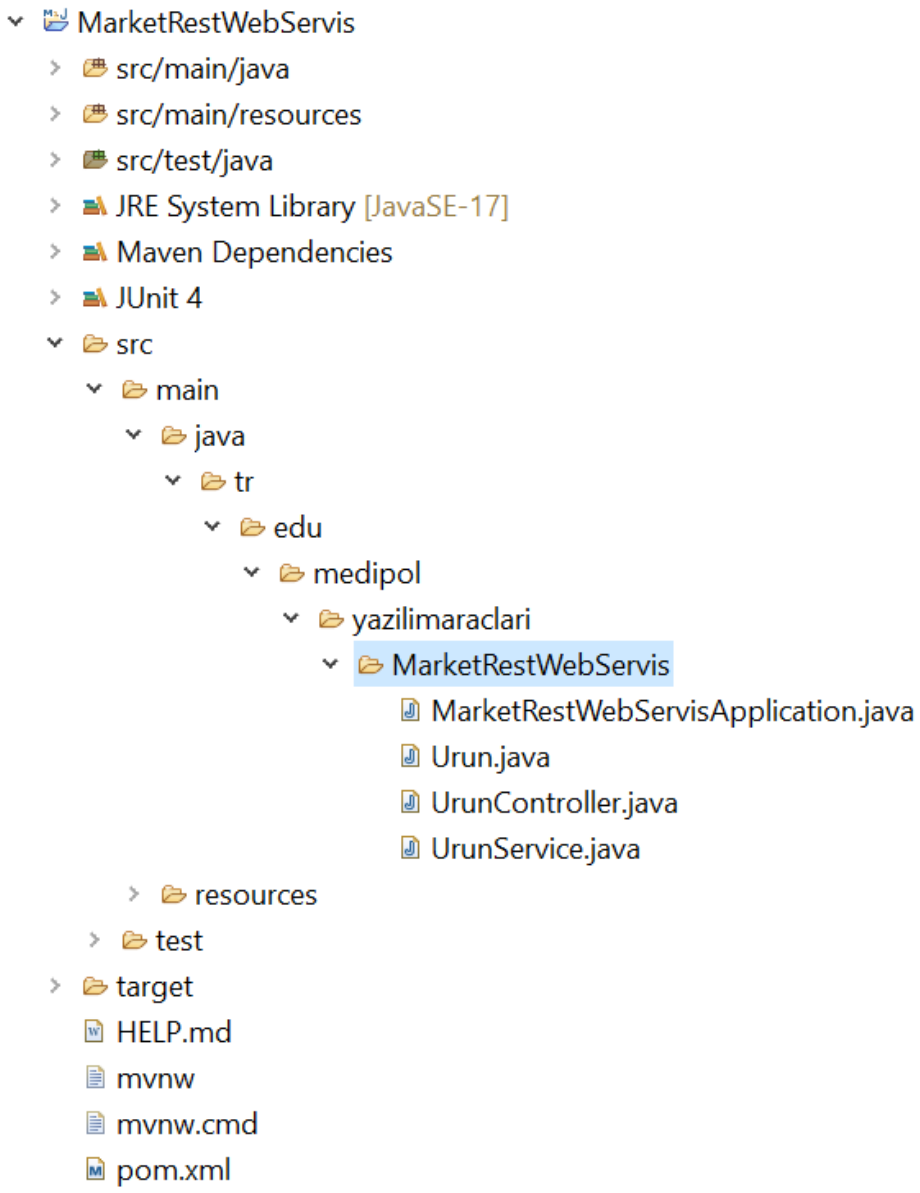
1- Projeyi oluşturmak için (https://start.spring.io/) adresine giderek gerekli ayarları yapıyoruz. Proje tipi olarak Maven tercih ediyoruz. Proje adını, paket adını ve Java sürümünü belirleyip projeye "Spring Web" bağımlılığını (dependency) ekliyoruz.



2- Sonrasında "GENERATE" butonuna basarak bilgisayarımıza indirdiğimiz zip dosyasının (benim örneğimde MarketRestWebServis.zip) içinden belirttiğimiz proje ismindeki (benim örneğimde MarketRestWebServis) klasörü çıkarıp

kullandığımız IDE üzerinden bu klasörü import ediyoruz. (Sizin belirlediğiniz proje adına göre zip dosyasının ismi farklılık gösterebilir.)

3- **Proje yapısı:** Projemdeki sınıfların ve "pom.xml" dosyasının dizini aşağıdaki şekildedir. (Ekran görüntüsü Eclipse IDE üzerinden alınmıştır.)



4- Aşağıdaki kod örneğindeki gibi bir RestController sınıfı oluşturuyoruz.

“UrunController.java”:

```java
package tr.edu.medipol.yazilimaraclari.MarketRestWebServis;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/urunler")
public class UrunController {

    private final UrunService urunService;

    @Autowired
    public UrunController(UrunService urunService) {
        this.urunService = urunService;
    }

    @PostMapping("/ekle")
    public String urunEkle(@RequestBody Urun urun) {
        urunService.urunEkle(urun);
        return "Ürün eklendi: " + urun.getUrunAdi();
    }

    @GetMapping("/listele")
    public List<Urun> urunleriListele() {
        return urunService.urunleriListele();
    }

    @DeleteMapping("/sil/{urunAdi}")
    public String urunSil(@PathVariable String urunAdi) {
        return urunService.urunSil(urunAdi);
    }
}
```

HTTP POST isteklerine cevap verecek şekilde işaretlenmiş bir metot ekleyerek ürün ekleme servisini oluşturulmuştur. Bu servis, gelen JSON verilerini kullanarak yeni bir ürün eklemeyi sağlamaktadır.

HTTP GET isteklerine cevap verecek şekilde işaretlenmiş bir metot ekleyerek ürün listeleme servisini oluşturulmuştur. Bu servis, mevcut ürün listesini döndürmektedir.

HTTP DELETE isteklerine cevap verecek şekilde işaretlenmiş bir metot ekleyerek ürün silme servisini oluşturulmuştur. Bu servis, belirtilen ürün ID'sine sahip ürünü listeden kaldırmaktadır.

5- Aşağıdaki kod örneğindeki gibi bir “UrunService” sınıfı oluşturuyoruz.
“UrunService.java”:

```
package tr.edu.medipol.yazilimaraclari.MarketRestWebServis;

import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;

@Service
public class UrunService {

    private static List<Urun> urunListesi = new ArrayList<>();

    public void urunEkle(Urun urun) {
        urunListesi.add(urun);
    }

    public List<Urun> urunleriListele() {
        return urunListesi;
    }

    public String urunSil(String urunAdi) {
        for (Urun urun : urunListesi) {
            if (urun.getUrunAdi().equals(urunAdi)) {
                urunListesi.remove(urun);
                return "Ürün silindi: " + urunAdi;
            }
```

```
        }
        return "Ürün bulunamadı: " + urunAdi;
    }
}
```

6- Aşağıdaki kod örneğindeki gibi bir “Urun” sınıfı oluşturuyoruz.

“Urun.java”:

```
package tr.edu.medipol.yazilimaraclari.MarketRestWebServis;

public class Urun {

    private String urunAdi;

    public Urun() {
    }

    public Urun(String urunAdi) {
        this.urunAdi = urunAdi;
    }

    public String getUrunAdi() {
        return urunAdi;
    }

    public void setUrunAdi(String urunAdi) {
        this.urunAdi = urunAdi;
    }
}
```

7- Projenin ana sınıfını aşağıdaki şekilde ayarlıyoruz.

“MarketRestWebServisApplication.java”:

```
package tr.edu.medipol.yazilimaraclari.MarketRestWebServis;

import org.springframework.boot.SpringApplication;
```

```java
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MarketRestWebServisApplication {

    public static void main(String[] args) {
        SpringApplication.run(MarketRestWebServisApplication.class, args);
    }
}
```

8- “pom.xml” dosyası aşağıdaki şekildedir.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.2.2</version> <!-- Update to the latest Spring Boot version
-->
        <relativePath/> <!-- lookup parent from repository -->
    </parent>

    <groupId>tr.edu.medipol.yazilimaraclari</groupId>
    <artifactId>MarketRestWebServis</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>MarketRestWebServis</name>
    <description>Market Rest Web Servis projesi</description>

    <properties>
        <java.version>17</java.version>
        <jacoco.version>0.8.8</jacoco.version> <!-- Update to the latest
Jacoco version -->
    </properties>

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

```
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <executions>
                    <execution>
                        <goals>
                            <goal>repackage</goal>
                        </goals>
                        <configuration>

<mainClass>tr.edu.medipol.yazilimaraclari.MarketRestWebServis.MarketRestWebSer
visApplication</mainClass>
                        </configuration>
                    </execution>
                </executions>
            </plugin>

         <plugin>
            <groupId>org.jacoco</groupId>
            <artifactId>jacoco-maven-plugin</artifactId>
            <version>0.8.8</version>
            <executions>
                <execution>
                    <goals>
                        <goal>prepare-agent</goal>
                    </goals>
                    <configuration>
                        <excludes>

<exclude>**/MarketRestWebServisApplication.class</exclude>
                        </excludes>
                    </configuration>
                </execution>
                <execution>
                    <id>report</id>
                    <phase>prepare-package</phase>
```

```
                    <goals>
                        <goal>report</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>

        </plugins>
    </build>

</project>
```

Bu şekilde Spring Boot kullanarak basit bir RESTful web servisi oluşturulmuştur ve Inversion of Control (IOC) ile Dependency Injection (DI) prensiplerini uygulamaktadır. Projede, ürünleri ekleme, listeleme ve silme işlevselliği sunulmaktadır. “`UrunController`” sınıfı, “`/urunler`” altında HTTP POST istekleri için ürün ekleyen, HTTP GET istekleri için mevcut ürünleri listeleyen ve HTTP DELETE istekleri için ürünleri adlarına göre silen endpoint'ler tanımlamaktadır. “`UrunService`” sınıfı ise, ürünlerin yönetimini sağlayan ve bellekte bir ürün listesi tutan bir servistir. Bu servis, ürün ekleme, listeleme ve silme işlemlerini gerçekleştirmektedir. `Urun` sınıfı, ürün nesnesinin yapısını tanımlamakta olup ürünün adını tutan bir alan içermektedir. “`UrunController`” sınıfı, “`UrunService`” bağımlılığını constructor (yapıcı metot) aracılığıyla alarak Dependency Injection uygulamaktadır. Bu, Spring'in IOC container'ı tarafından yönetilir ve nesnelerin bağımlılıklarını otomatik olarak enjekte eder. Son olarak, “`MarketRestWebServisApplication`” sınıfı, Spring Boot uygulamasını başlatan ana sınıftır. Bu sınıf, “`SpringApplication.run`” metodunu çağırarak uygulamanın çalışmasını sağlamaktadır. Uygulama çalıştırıldığında, “`localhost:8080/urunler`” adresinden ürün ekleme, listeleme ve silme işlemleri yapılabilir.