

# Injekcija zavisnosti u Spring radnom okviru

---

# Sadržaj

---

- Šta je injekcija zavisnosti
- Injekcija zavisnosti u Springu
- Spring konfiguracija

# Zavisnost (engl. *dependency*)

- Kada objekat klase A koristi usluge objekta klase B, kaže se da je objekat klase B zavisnost (engl. *dependency*) objekta klase A

```
public class MyController {
    private MyService myService;

    public MyController(MyService myService) {
        this.myService = myService;
    }
    ...
    public void setMyService(MyService myService) {
        this.myService = myService
    }

    public void doSomething() {
        this.myService.doSomethingForMe();
    }
}
```

# Injekcija zavisnosti

---

- Injekcija zavisnosti (engl. *Dependency Injection* - DI) je operacija kojom se potrebna zavisnost injektuje posmatranom objektu od strane trećeg objekta ili radnog okvira
- Objekat kome se injektuje zavisnost nema odgovornost instanciranja objekta koji mu se injektuje
- Uspostavljanje reference na objekat od kog posmatrani zavisi vrši se pozivom konstruktora ili odgovarajuće set metode

# Vrste injekcije zavisnosti

---

- Pomoću obeležja klase
  - javno ili privatno obeležje
  - korišćenje privatnog obeležja za injekciju zavisnosti je loša praksa jer otežava testiranje
- Pomoću set metode
- Pomoću konstruktora
  - optimalna opcija

# Konkretne klase nasuprot interfejsa

---

- Injekcija zavisnosti može biti realizovana pomoću konkretnih klasa, apstraktnih klasa ili interfejsa
- Po pravilu, injekciju zavisnosti preko konkretnih klasa treba izbegavati
- Preferira se injekcija zavisnosti pomoću interfejsa
  - Omogućava da interpreter u vreme izvršenja programa odluči koja će se implementacija injektovati
  - U skladu je sa *Interface Segregation* SOLID principom
  - Čini sistem lakšim za testiranje

# Inverzija kontrole (engl. *Inversion of Control*)

---

- Tehnika koja omogućava injekciju zavisnosti u vreme izvršenja programa
- Zavisnosti nisu predefinisane, već se određuju u vreme izvršenja programa
- IoC i DI jesu povezani ali ne predstavljaju isti pojam
- Spring kontejner preuzima kontrolu i (između ostalog) izvršava injekciju zavisnosti

# Injekcija zavisnosti bez IoC

```
MyController.java
5
6 @Controller
7 public class MyController {
8
9     private SayHelloService sayHelloService;
10
11     public void doGreeting() {
12         System.out.println(this.sayHelloService.sayHello());
13     }
14
15     public void setSayHelloService(SayHelloService sayHelloService) {
16         this.sayHelloService = sayHelloService;
17     }
18 }
19
SayHelloServiceImpl.java
4
5 @Service
6 public class SayHelloServiceImpl implements SayHelloService {
7
8     @Override
9     public String sayHello() {
10         return "Hello World";
11     }
12 }
13
SpringdiApplication.java
1 package ns.inforce.springdi;
2
3 import ...
4
5 @SpringBootApplication
6 public class SpringdiApplication {
7
8     public static void main(String[] args) {
9         ApplicationContext ctx = SpringApplication.run(SpringdiApplication.class, args);
10
11         MyController myController = (MyController) ctx.getBean( name: "myController");
12         SayHelloService sayHelloService = (SayHelloService) ctx.getBean( name: "sayHelloServiceImpl");
13
14         myController.setSayHelloService(sayHelloService);
15
16         myController.doGreeting();
17     }
18 }
19 }
```

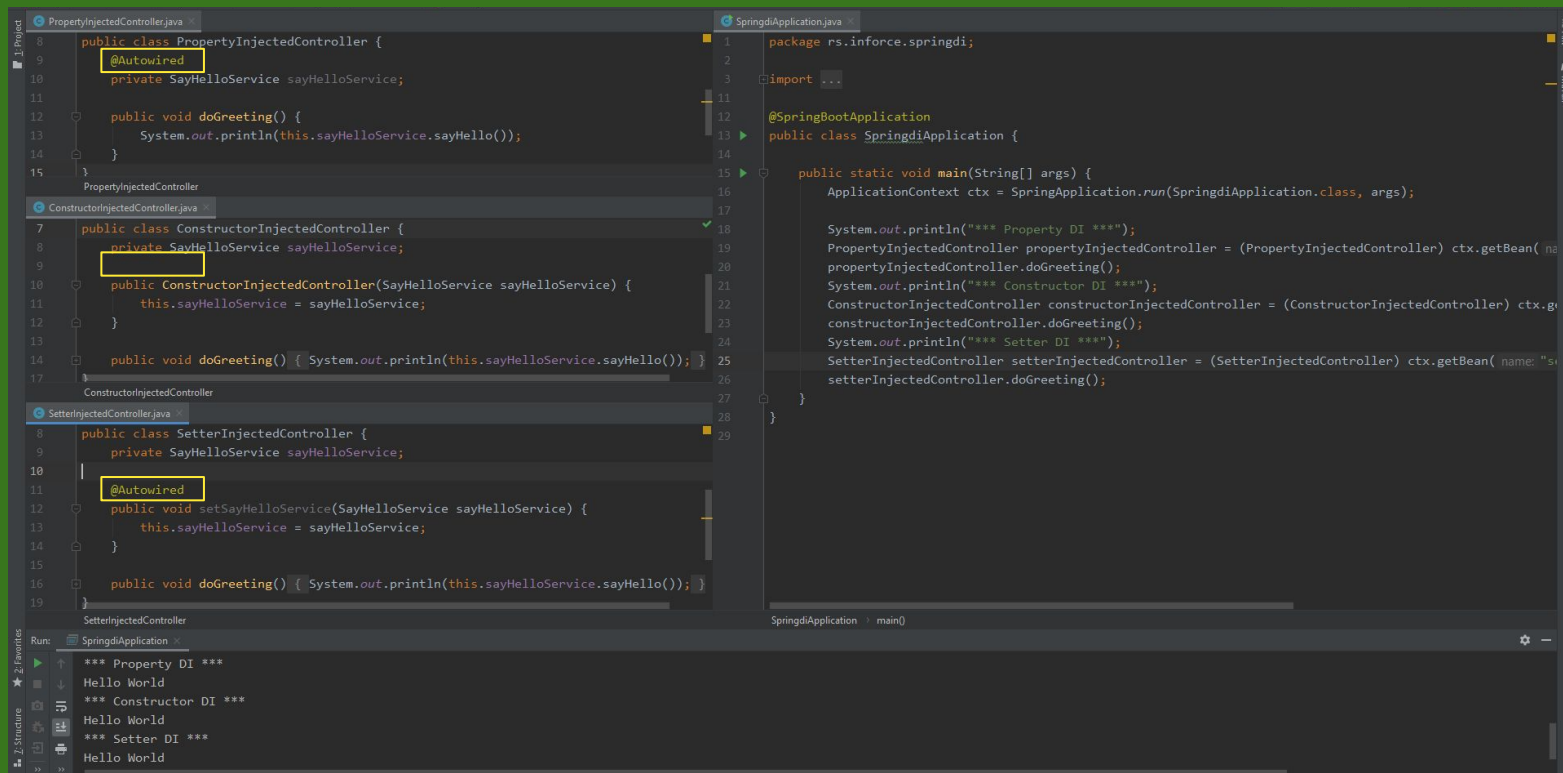


# Spring i injekcija zavisnosti

---

- Anotacija `@Autowired` ukazuje da Spring IoC kontejner treba da izvrši injekciju zavisnosti
- Injekcija se može izvršiti korišćenjem
  - promenljive deklarisanе u klasi - introspekcijom se detektuje i izvrši inicijalizacija
  - set metode
  - konstruktora - optimalna opcija

# Primer injekcije zavisnosti



The screenshot displays an IDE with three Java files on the left and a main application file on the right. The bottom panel shows the output of running the application.

**PropertyInjectedController.java**

```
1 public class PropertyInjectedController {
2     @Autowired
3     private SayHelloService sayHelloService;
4
5     public void doGreeting() {
6         System.out.println(this.sayHelloService.sayHello());
7     }
8 }
```

**ConstructorInjectedController.java**

```
1 public class ConstructorInjectedController {
2     private SayHelloService sayHelloService;
3
4     public ConstructorInjectedController(SayHelloService sayHelloService) {
5         this.sayHelloService = sayHelloService;
6     }
7
8     public void doGreeting() { System.out.println(this.sayHelloService.sayHello()); }
9 }
```

**SetterInjectedController.java**

```
1 public class SetterInjectedController {
2     private SayHelloService sayHelloService;
3
4     @Autowired
5     public void setSayHelloService(SayHelloService sayHelloService) {
6         this.sayHelloService = sayHelloService;
7     }
8
9     public void doGreeting() { System.out.println(this.sayHelloService.sayHello()); }
10 }
```

**SpringdiApplication.java**

```
1 package rs.inforce.springdi;
2
3 import ...
4
5 @SpringBootApplication
6 public class SpringdiApplication {
7
8     public static void main(String[] args) {
9         ApplicationContext ctx = SpringApplication.run(SpringdiApplication.class, args);
10
11         System.out.println("*** Property DI ***");
12         PropertyInjectedController propertyInjectedController = (PropertyInjectedController) ctx.getBean("propertyInjectedController");
13         propertyInjectedController.doGreeting();
14
15         System.out.println("*** Constructor DI ***");
16         ConstructorInjectedController constructorInjectedController = (ConstructorInjectedController) ctx.getBean("constructorInjectedController");
17         constructorInjectedController.doGreeting();
18
19         System.out.println("*** Setter DI ***");
20         SetterInjectedController setterInjectedController = (SetterInjectedController) ctx.getBean("setterInjectedController");
21         setterInjectedController.doGreeting();
22     }
23 }
```

**Run Output:**

```
*** Property DI ***
Hello World
*** Constructor DI ***
Hello World
*** Setter DI ***
Hello World
```

# Injekcija zavisnost za više implementacija

- Pretpostavimo da imamo više klasa koje implementiraju interfejs koji se injektuje
- Spring kontejner ne zna koji *bean* da injektuje
- Nekoliko rešenja
  - imenovanje promenljive koja se injektuje
  - korišćenje anotacije `@Qualifier`
  - korišćenje anotacije `@Primary`

```
*****  
APPLICATION FAILED TO START  
*****
```

Description:

```
Parameter 0 of constructor in rs.inforce.springdi.controllers.ConstructorInjectedController required a single bean, but 2 were found:  
- sayHelloServiceImpl: defined in file [C:\qhSpring\springdi\target\classes\rs\inforce\springdi\services\SayHelloServiceImpl.class]  
- sayHelloServiceRsImpl: defined in file [C:\qhSpring\springdi\target\classes\rs\inforce\springdi\services\SayHelloServiceRsImpl.class]
```

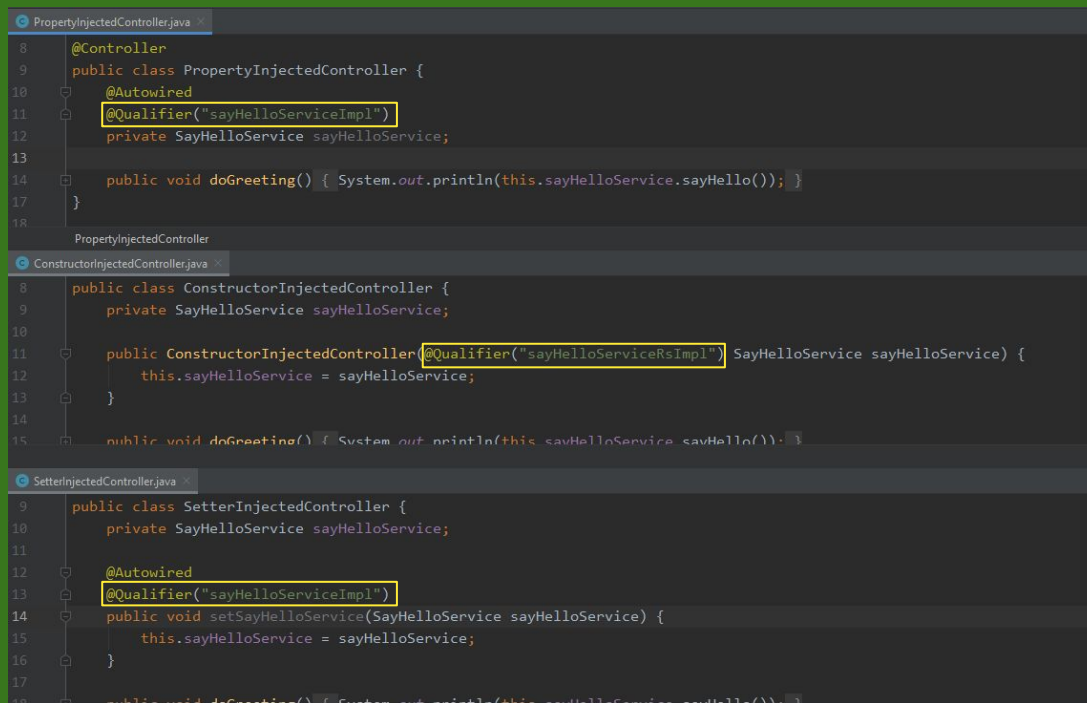
Action:

Consider marking one of the beans as `@Primary`, updating the consumer to accept multiple beans, or using `@Qualifier` to identify the bean that should be consumed

```
SayHelloServiceRsImpl.java  
1 package rs.inforce.springdi.services;  
2  
3 import org.springframework.stereotype.Service;  
4  
5 @Service  
6 public class SayHelloServiceRsImpl implements SayHelloService {  
7  
8     @Override  
9     public String sayHello() {  
10         return "Zdravo svete";  
11     }  
12 }  
13  
SayHelloServiceImpl.java  
1 package rs.inforce.springdi.services;  
2  
3 import org.springframework.stereotype.Service;  
4  
5 @Service  
6 public class SayHelloServiceImpl implements SayHelloService {  
7  
8     @Override  
9     public String sayHello() {  
10         return "Hello World";  
11     }  
12 }
```

# Anotacije @Qualifier

- Određuje koja implementacija će biti injektovana u zavisnu komponentu



```
PropertyInjectedController.java
8  @Controller
9  public class PropertyInjectedController {
10     @Autowired
11     @Qualifier("sayHelloServiceImpl")
12     private SayHelloService sayHelloService;
13
14     public void doGreeting() { System.out.println(this.sayHelloService.sayHello()); }
15 }

ConstructorInjectedController.java
8  public class ConstructorInjectedController {
9     private SayHelloService sayHelloService;
10
11     public ConstructorInjectedController(@Qualifier("sayHelloServiceRsImpl") SayHelloService sayHelloService) {
12         this.sayHelloService = sayHelloService;
13     }
14
15     public void doGreeting() { System.out.println(this.sayHelloService.sayHello()); }
16 }

SetterInjectedController.java
9  public class SetterInjectedController {
10     private SayHelloService sayHelloService;
11
12     @Autowired
13     @Qualifier("sayHelloServiceImpl")
14     public void setSayHelloService(SayHelloService sayHelloService) {
15         this.sayHelloService = sayHelloService;
16     }
17
18     public void doGreeting() { System.out.println(this.sayHelloService.sayHello()); }
```

# Anotacije @Primary

- Definiše implementaciju interfejsa kao primarnu za injektovanje

```
SayHelloServicePrimaryImpl.java
6  @Service
7  @Primary
8  public class SayHelloServicePrimaryImpl implements SayHelloService {
9
10
11     @Override
12     public String sayHello() {
13         return "Hello World from Primary";
14     }
15 }
16

PropertyInjectedController.java
6
7  @Controller
8  public class PropertyInjectedController {
9      @Autowired
10     private SayHelloService sayHelloService;
11
12     public void doGreeting() { System.out.println(this.sayHelloService.sayHello()); }
13
14
15 }
16
```

```
*** Property DI ***
Hello World from Primary
*** Constructor DI ***
Zdravo svete
*** Setter DI ***
Hello World
```

# Spring profil

The screenshot displays a Spring application with three profiles: 'EN', 'RS', and 'Primary'. Each profile has a corresponding implementation of the `SayHelloService` interface. The application is configured to use the 'RS' profile, as indicated by the `application.properties` file and the output log.

```
application.properties
spring.profiles.active=RS
```

```
 SayHelloServiceImpl.java
 5
 6 @Service
 7 @Profile("EN")
 8 public class SayHelloServiceImpl implements SayHelloService {
 9     @Override
10     public String sayHello() { return "Hello World"; }
11 }
 SayHelloServicePrimaryImpl.java
 6 @Service
 7 @Profile("EN")
 8 public class SayHelloServicePrimaryImpl implements SayHelloService {
 9     @Override
10     public String sayHello() { return "Hello World from Primary"; }
11 }
 SayHelloServiceRsImpl.java
 2
 3 import org.springframework.context.annotation.Profile;
 4 import org.springframework.stereotype.Service;
 5
 6 @Service
 7 @Profile("RS")
 8 public class SayHelloServiceRsImpl implements SayHelloService {
 9
10     @Override
11     public String sayHello() { return "Zdravo svete"; }
12 }
 Run: SpringBootApplication
 *** Property DI ***
 Zdravo svete
 *** Constructor DI ***
 Zdravo svete
 *** Setter DI ***
 Zdravo svete
```

```
PropertyInjectedController.java
 7 @Controller
 8 public class PropertyInjectedController {
 9     @Autowired
10     private SayHelloService sayHelloService;
11
12     public void doGreeting() { System.out.println(this.sayHelloService.sayHello()); }
13 }
 ConstructorInjectedController.java
 7 @Controller
 8 public class ConstructorInjectedController {
 9     private SayHelloService sayHelloService;
10
11     public ConstructorInjectedController(SayHelloService sayHelloService) {
12         this.sayHelloService = sayHelloService;
13     }
14 }
 SetterInjectedController.java
11
12 @Autowired
13 public void setSayHelloService(SayHelloService sayHelloService) {
14     this.sayHelloService = sayHelloService;
15 }
16
17 public void doGreeting() { System.out.println(this.sayHelloService.sayHello()); }
```

# Profile default

- Ako nije definisan profil, kreiraće se bean-ovi za default profil

```
SayHelloServiceImpl.java
1 package rs.inforce.springdi.services;
2
3 import org.springframework.context.annotation.Profile;
4 import org.springframework.stereotype.Service;
5
6 @Service
7 @Profile({"EN", "default"})
8 public class SayHelloServiceImpl implements SayHelloService {
9     @Override
10    public String sayHello() { return "Hello World"; }
13 }
14
```

SayHelloServiceImpl

```
application.properties
1 #spring.profiles.active=RS
2
```

```
*** Property DI ***
Hello World
*** Constructor DI ***
Hello World
*** Setter DI ***
Hello World
```

# Rezime

---

- Spring IoC kontejner vrši injekciju zavisnosti
- Preferira se injekcija interfejsa
- Kvalifikatori omogućavaju da se odredi koja će se komponenta injektovati
- Spring profili omogućavaju da se za različit profil instanciraju različite komponente