



Distance Learning System

Java Web development

Spring framework

Šta je Spring framework

- Spring framework je multifunkcionalni Java framework čiji je primarni cilj olakšavanje kreiranja Java Enterprise aplikacija
- Ipak, iako je pomoću Springa moguće kreirati bilo koju vrstu aplikacija, on je naročito koristan za kreiranje web aplikacija



<http://spring.io/>

Spring struktura

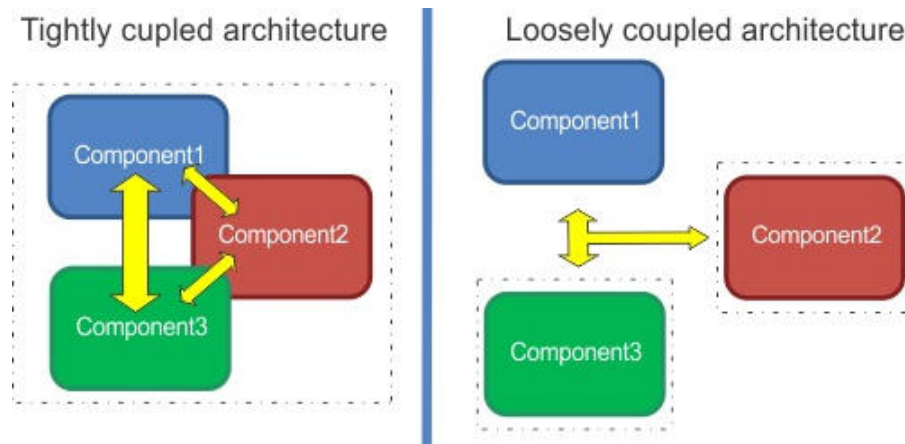
- Spring Framework se sastoji iz iz nekoliko delova:
 - AOP i instrumentacija
 - Core kontejner
 - Pristup podacima/integracija
 - Web
 - Test

Koncepti: AOP, IOC i DI

- Kompletna struktura Spring framework-a, bazirana je na tri koncepta:
 - Aspect Oriented Programming (AOP)
 - Inversion of Control (IOC)
 - Dependency Injection (DI)

IoC i Dependency injection

- Aplikacije kreirane korišćenjem Spring Frameworka su sačinjene od komponenti. U realnim sistemima ovakve komponente će morati da sarađuju međusobno, tj. moraće postojati određene zavisnosti među komponentama sistema. Ove zavisnosti, ako se na kontrolišu na pravi način, mogu dovesti do veoma čvrsto spregnutih komponenti. Što ne želimo da se dogodi.



Primer jako povezanih komponenti

- Naš cilj, prilikom kreiranja bilo koje aplikacije, pa i veb-aplikacije, potrebno je da bude nastojanje da se napravi sistem sa što nezavisnijim komponentama.
- Sa druge strane, nije realno napraviti ni sistem u kome će komponente biti potpuno izolovane, jer ovakav sistem ne bi imao svrhu.
- Dakle, potrebno je da komponente zavise jedna od druge, ali nije dobro ukoliko one razmišljaju i o dobavljanju ovih komponenti.
- Komponente koje predstavljaju zavisnosti drugih komponenti je potrebno da budu dostavljene komponentama kojima su potrebne.
- Objašnjeni koncept je esencija programerskog šablona koji se zove **Inversion of Control**. Spring Framework je okvir koji podržava IoC i to kroz tehniku koja se naziva **Dependency injection**.

Primer zavisnosti i IoC-a



- Jaka povezanost

```
public class ClassA {  
    private ClassB classB;  
    public ClassA() {  
        classB = new ClassB();  
    }  
}
```

- Primena IOC šablona

```
public class ClassA {  
    private ClassB classB;  
    public ClassA(ClassB classB) {  
        this.classB = classB;  
    }  
}
```

Ubacivanje zavisnosti pomoću getera i setera

- Osim pomoću konstruktora, zavisnosti je moguće ostvariti i pomoću getera i setera

```
public class ClassA{  
    private ClassB classB;  
    // a setter method to inject the dependency.  
    public void setClassB(ClassB classB) {  
        this.classB = classB;  
    }  
    // a getter method to return classB  
    public ClassB getClassB() {  
        return classB;  
    }  
}
```


Tipizacija zavisnosti

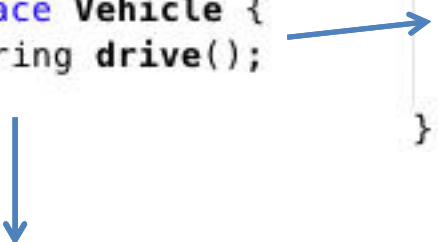
- U prethodnim slajdovima, kao zavisnost je korišćena direktna implementacija
- Praksa je da se za zavisnost koristi samo apstrakcija, kako bi se ostvarila fleksibilnost u pridruživanju zavisnosti

1

```
public interface Vehicle {  
    public String drive();  
}
```

2

```
public class Bike implements Vehicle {  
    public String drive() {  
        return "driving a bike";  
    }  
}
```



2

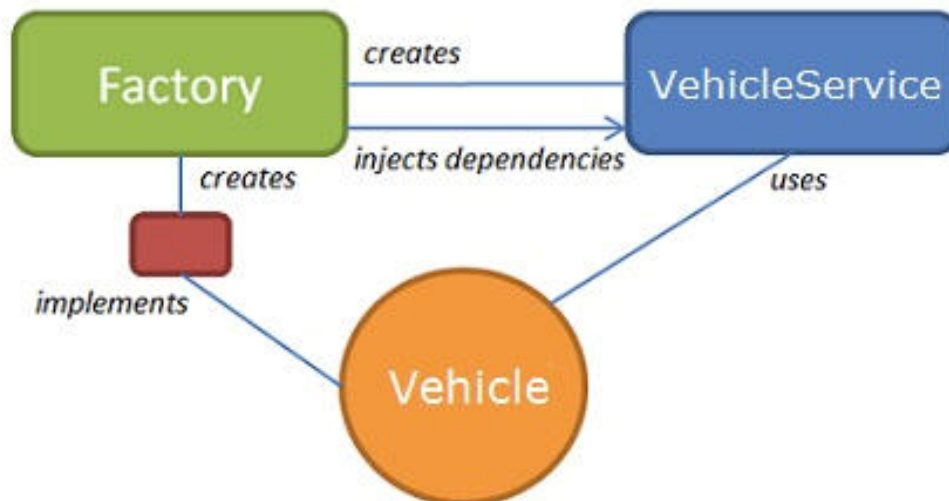
```
public class Car implements Vehicle {  
    public String drive() {  
        return "driving a car";  
    }  
}
```

3

```
public class VehicleService {  
    private Vehicle vehicle;  
    public VehicleService(Vehicle vehicle) {  
        this.vehicle = vehicle;  
    }  
}
```

Tipizacija zavisnosti

- Primer sa prethodnog slajda može se predstaviti na sledeći način

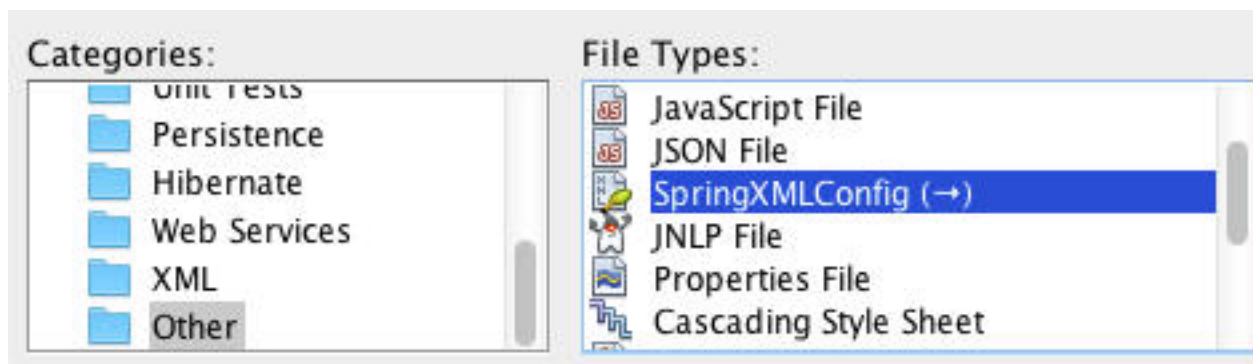


Kreiranje zavisnosti u spring frameworku

- U Springu se zavisnosti mogu ostvariti na dva načina – putem Java konfiguracionih klasa ili xml fajlova
- Neophodno je samo kreirati i označiti odgovarajuću klasu i ona može postati zavisnost
- Da bi klasa bila ispravna zavisnost, mora da bude validan Spring Bean (Java Bean ili POJO)

Kreiranje i korišćenje Spring konfiguracionog xml fajla

- Spring može kompletan sistem zavisnosti formirati na osnovu informacija iz xml fajla
- Ovaj xml fajl mora biti eksplicitno učitán ukoliko ručno pravimo aplikaciju
- Ovi konfiguracioni podaci nazivaju se još i: **kontekst**
- Unutar NetBeans-a postoji čarobnjak za kreiranje spring xml kontekstnog fajla



Korišćenje čarobnjaka za kreiranje Spring xml konfiguracionog fajla

Name and Location

File Name:

NetBeans čarobnjak omogućava da odaberemo koje ćemo prostore imena koristiti u našoj Spring aplikaciji, odnosno, konfiguracionom fajlu

Ove prostore imena možemo tretirati kao pakete u Java aplikaciji

Spring namespaces

- ☐ aop - <http://www.springframework.org/schema/aop>
- ☐ c - <http://www.springframework.org/schema/c>
- ☒ context - <http://www.springframework.org/schema/context>
- ☐ flow - <http://www.springframework.org/schema/webflow-config>
- ☐ jee - <http://www.springframework.org/schema/jee>
- ☐ jms - <http://www.springframework.org/schema/jms>
- ☐ lang - <http://www.springframework.org/schema/lang>
- ☐ osgi - <http://www.springframework.org/schema/osgi>
- ☐ p - <http://www.springframework.org/schema/p>
- ☐ tx - <http://www.springframework.org/schema/tx>
- ☐ util - <http://www.springframework.org/schema/util>

Sadržaj Spring konfiguracionog fajla

- Inicijalni sadržaj Spring konfiguracionog fajla, može izgledati ovako:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
               http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
           http://www.springframework.org/schema/context
               http://www.springframework.org/schema/context/spring-context-4.0.xsd"
">
</beans>
```

Učitavanje Spring konfiguracionog fajla u Javu

- Da bi Java uzela u obzir Spring konfiguraciju definisanu xml fajlom, ovaj fajl treba učitati na neki od načina
- U web aplikaciji, fajl se učitava automatski, ali u konzolnoj, moramo ga ručno učitati
- Najlakši način da se učitava jeste korišćenje metode **ClassPathXmlApplicationContext**

```
public class FirstSpringApp {  
    public static void main(String[] args) {  
        ApplicationContext context  
            = new ClassPathXmlApplicationContext("config.xml");  
    }  
}
```

Kreiranje i registrovanje Spring bean-a

- Spring bean može biti bilo koja klasa
- Da bi registrovali bean, dodajemo klasi anotaciju @Component, ili je označavamo u konfiguracionom xml fajlu
- Na taj način, imamo mogućnost deklarativno da utičemo na rad programa (konfigurisanjem xml fajla)

```
public interface MyInterface {  
    public void f();  
}
```

→

```
public class MyClass implements MyInterface {  
    public void f(){  
        System.out.println("Hello");  
    }  
}
```

↘

```
public class MyClass1 implements MyInterface {  
    public void f() {  
        System.out.println("Hello my class1");  
    }  
}
```

```
<bean  
    id="myBean"  
    class="firstspringapp.MyClass"  
>
```

The diagram illustrates the process of creating and registering a Spring bean. It shows a Java interface `MyInterface` with a method `f()`. Two classes, `MyClass` and `MyClass1`, implement this interface. `MyClass` prints "Hello", while `MyClass1` prints "Hello my class1". An XML snippet shows how to register `MyClass` as a Spring bean with the ID `myBean`.

Korišćenje bean-a unutar Java aplikacije

- U Javi nas "ne interesuje" koja će implementacija biti upotrebljena za objekat. Važno nam je samo da je interfejs ispoštovan

```
ApplicationContext context  
    = new ClassPathXmlApplicationContext("config.xml");  
MyInterface mc = (MyInterface)context.getBean("myBean");  
mc.f();
```

Konfigurisanje zavisnosti

- Moguće je povezati objekte u samom spring konfiguracionom fajlu kroz elemente **property** ili **constructor-arg**

```
<bean id="classA" class="ClassA">
  <property name="classB" ref="classB" />
</bean>
<bean id="classB" class="ClassB" />
```

```
<!-- Definition for classA bean -->
<bean id="classA" class="ClassA">
  <constructor-arg ref="classB"/>
</bean>
```

```
<!-- Definition for classB bean -->
<bean id="classB" class="ClassB">
</bean>
```

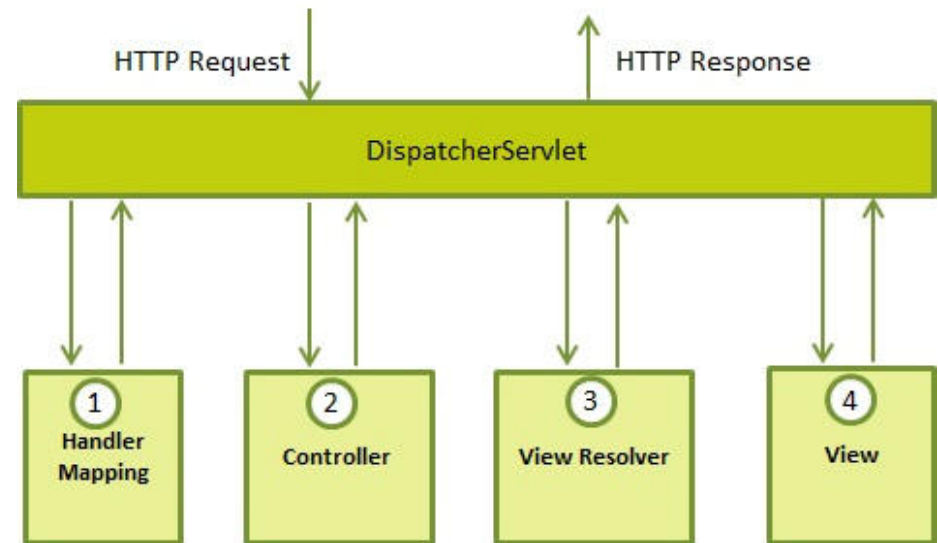
Vežba 1

(jwex 062014 SpringConsoleApp)

- Kreira se konzolna aplikacija za parsiranje gps koordinata
- Za parsiranje se koriste dve klase: AndroidParser i iPhoneParser
- AndroidParser parsira podatke u sledećem formatu: lat|lon
- iPhoneParser parsira podakte u sledećem formatu lon lat
- Obe klase moraju implementirati interfejs ICoordinateParser
- Interfejs ICoordinateParser sadrži jedan metod: parse
- Potrebno je kreirati spring konzolnu aplikaciju unutar koje će u konfiguraciji moći da se odredi sa kojim će uređajem raditi aplikacija
- Rešenje treba da minimalizuje upotrebu Java koda

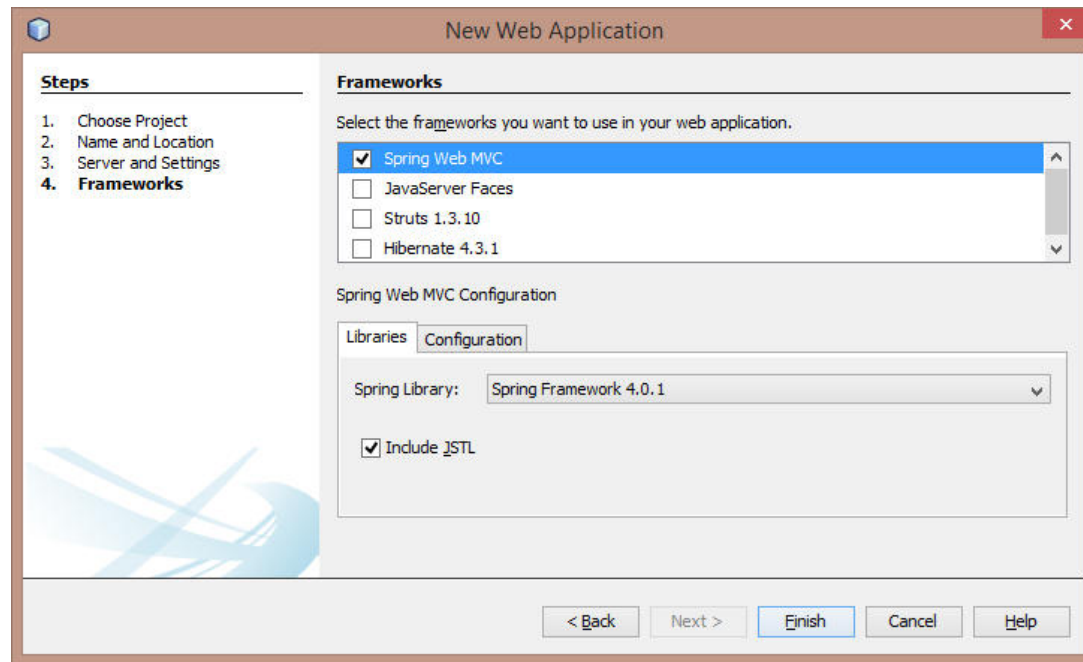
Spring MVC

- Spring Web MVC je framework za kreiranje veb-aplikacija. On se zasniva na postojanju jednog centralnog servleta koji se naziva DispatcherServlet.
- On je zadužen za prosleđivanje zahteva do kontrolera, i zapravo i sam je jedan kontroler, ali takozvani Front Controller



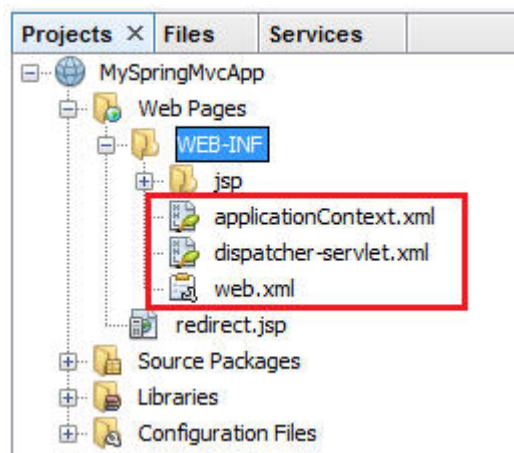
Kreiranje spring mvc projekta (jwex062014 MySpringMVCAApp)

- NetBeans sadrži opciju ugrađivanja spring mvc šablona u web aplikaciju:



Struktura spring mvc projekta

- Po kreiranju projekta, spring generiše konfiguracione fajlove: **applicationContext.xml** i **dispatcher-servlet.xml**
- Takođe, standardno se generiše web.xml fajl koji referencira application context i dispatcher servelet.



Usmeravanje zahteva na spring servlet

- U fajlu web.xml, treba navesti rutu i servlet koji će ovu rutu obrađivati
- Ovaj postupak nema veze sa springom, već je deo same java ee aplikacije

```
<servlet>
  <servlet-name>dispatcher</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>dispatcher</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>
```

Konfigurisanje kontroler klasa

- Podrazumevano, spring netbeans šablon kreira jedan kontroler unutar konfiguracionog fajla
- Kontrolere takođe možemo definisati kroz sopstvene klase
- Ovako definisane kontrolere navodimo pojedinačno navođenjem naziva u konfiguracionom fajlu, ili anotacijama u samim klasama
 - Da bi spring znao koje klase da tretira kao kontrolere, mora biti naveden paket u kome se one nalaze
- Pored kontrolera, u konfiguraciji se navodi i podrazumevano ponašanje za učitavanje pogleda

```
<context:component-scan base-package="myspringmvcapp" />
<bean id="viewResolver"
      class="org.springframework.web.servlet.view.InternalResourceViewResolver"
      p:prefix="/WEB-INF/jsp/"
      p:suffix=".jsp" />
```

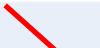

Kreiranje kontrolera

- Sve klase koje se nalaze u navedenom paketu (u primeru `myspringmvcapp`), a pri tom sadrže anotaciju `@Controller`, biće uzete u obzir prilikom mapiranja kontrolera u aplikaciji
- Svaki metod kontrolera odazivaće se na zahtev koji odgovara njegovom mapiranju
- Mapiranja metoda vrše se anotacijom `@RequestMapping`
- Ako anotaciju `@RequestMapping` postavimo na sam kontroler, tada će se ona uzimati u obzir prilikom parsiranja tražene putanje iz zahteva

```
@Controller
public class MyFirstController {
    @RequestMapping("/myfirstpage")
    public String myPage() {
        return "myfirstview";
    }
}
```

Kreiranje kontrolera

```
@Controller
@RequestMapping("/somethingelse")
public class MyFirstController {
    @RequestMapping("/myfirstpage")
    public String myPage() {
        return "myfirstview";
    }
}
```



← → ↻ localhost:8080/MySpringMvcApp/myfirstpage

Hello World!



```
@Controller
public class MyFirstController {
    @RequestMapping("/myfirstpage")
    public String myPage() {
        return "myfirstview";
    }
}
```

← → ↻ localhost:8080/MySpringMvcApp/somethingelse/myfirstpage

Hello World!

Preuzimanje parametara

- MVC je u stanju da preuzme parametre poslate na nekoliko načina
- POST parametri se preuzimaju standardno, iz tela zahteva
- GET parametri se preuzimaju na jedan od dva ili oba načina
 - Parsiranjem delova putanje
 - /id/category (/10/25)
 - Parsiranjem standardne parametar liste
 - /?id=id&cat=category (/?id=10&cat=25)

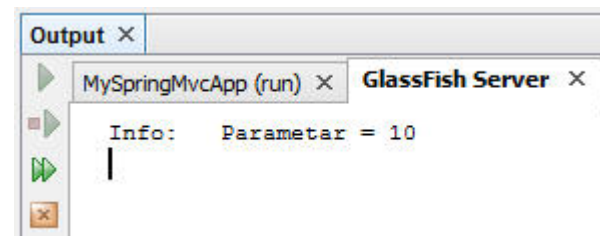
Preuzimanje get parametara

- Get parametri se mogu preuzeti na standardan način, i tada se koristi anotacija `@RequestParam` ispred parametra metode
- Da bi se izbegao nedostatak parametra, može se postaviti njegovo svojstvo **required** na vrednost `false`

```
@Controller
public class MyFirstController {
    @RequestMapping("/myfirstpage")
    public String myPage(@RequestParam(required = false) Integer id) {
        System.out.println("Parametar = " + id);
        return "myfirstview";
    }
}
```

← → ↻

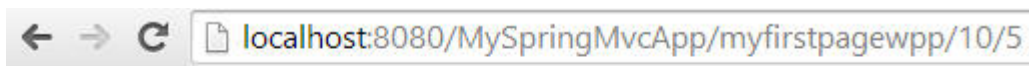
Hello World!



Preuzimanje path parametara

- Parametri putanje su parametri koji se dobijaju raslojavanjem direktorijuma putanje
- Na primer, umesto da korisnik piše: `?id=10&category=5` može napisati `/10/5`
- Ovaj sistem nije logičan serveru, i zato se mora naglasiti da će putanja biti tretirana na ovaj način
- Kod springa je ovo transparentno

```
@RequestMapping("/myfirstpagewpp/{id}/{cat}")  
public String myPageWithPathParameters(@PathVariable Integer id, @PathVariable Integer cat) {  
    System.out.println("Parametar: id = " + id + ", category = " + cat);  
    return "myfirstview";  
}
```



localhost:8080/MySpringMvcApp/myfirstpagewpp/10/5

Hello World!



Output x
MySpringMvcApp (run) x GlassFish Server x
Info: Parametar: id = 10, category = 5

Preuzimanje POST parametara


- Jedan od načina da se preuzmu POST parametri jeste takođe korišćenje anotacije @RequestParam

```
@RequestMapping("/formpage")
public String formpage(
    @RequestParam(required = false) String username,
    @RequestParam(required = false) String password
) {
    System.out.println("Parametar: username = " + username
        + ", password = " + password);
    return "form";
}
```

Mapiranje parametara forme na model

- U springu je moguće direktno mapiranje bean-a na parametre forme, pa se tako može izvršiti implicitno transportovanje podataka iz forme na server i obrnuto

```
@RequestMapping("/modelformpage")  
public String modelformpage(@ModelAttribute User user) {  
    System.out.println("Parametar: username = " + user.getUsername()  
        + ", password = " + user.getPassword());  
    return "modelform";  
}
```



```
<form method="post" action="">  
    <input type="text" name="username" value="${user.username}" />  
    <input type="text" name="password" value="${user.password}" />  
    <input type="submit" value="Send params" />  
</form>
```

Uvođenje izvora podataka

- Iako je moguće koristiti standardno upravljanje podacima, najčešće se u kombinaciji sa springom koriste objektno relacioni maperi
- Većina podešavanja vezanih za konekciju ka bazi i mapiranje, nalazi se u spring konfiguracionim fajlovima, reprezentovana kroz bean-ove

```
<bean id="dataSource"
    class="org.springframework.jdbc.datasource.DriverManagerDataSource"
    p:driverClassName="com.mysql.jdbc.Driver"
    p:url="jdbc:mysql://localhost/sakila"
    p:username="root"
    p:password=""
/>
```


Uvođenje izvora podataka

- Kada je izvor podataka jednom definisan u konfiguraciji, može se koristiti bez dodatnog definisanja.

```
@Autowired
DataSource dataSource;

@RequestMapping("/dbdrivenpage")
public String dbdrivenpage() throws SQLException {
    Connection conn = dataSource.getConnection();
    Statement st = conn.createStatement();
    ResultSet rs = st.executeQuery("select * from actor");
    while(rs.next()){
        System.out.println(rs.getString(2) + " " + rs.getString(3) + " " + rs.getString(4));
    }
    return "dbdrivenview";
}
```