spring

by Pivotal™

Uvod u programski okvir Spring

Web aplikacije u Javi

Sadržaj predavanja

- Uvod u programski okvir Spring
- JavaBeans
- Značajke i mehanizmi Springa
- *Dependency Injection*
- *Aspect oriented programming*
- Spring *templateovi*
- Spring *container*
- Moduli Springa

Uvod u programski okvir Spring

- Programski okvir otvorenog koda (engl. *Open source framework*)
- Osmišljen 2002. od Roda Johnsona
- Koristi osnovne JavaBeans klase (*plain-vanilla JavaBeans*)
- Pojednostavnjuje razvoj i testiranje Java aplikacija i omogućava korištenje labave sprege među komponentama

JavaBeans

- Nazivaju se i POJO klase (engl. *Plain Old Java Objects*)
- Definiraju komponente programskog jezika koje se mogu ponovno iskorištavati
- Implementiraju sučelje „Serializable", imaju konstruktor bez ulaznih parametara te omogućavaju pristup poljima objekata uz pomoć „getter" i „setter" metoda
- Jednostavnost programskog okvira Spring se temelji na njima

JavaBeans

- Primjer:

```
public class Fakultet implements Serializable {

   private String naziv;
   private int brojDiplomanata;

   public String getNaziv() {
      return naziv;
   }

   public void setNaziv(String naziv) {
      naziv = naziv;
   }
   …
}
```

Značajke i mehanizmi Springa

- Jednostavniji Java kôd
- *Dependency Injection*
- *Aspect Oriented Programming*
- Korištenje *templatea*
- Objekti koje koristi aplikacija nalaze se unutar Spring *containera* koji se „brine" za njih tijekom cijelog životnog ciklusa

Dependency Injection

- Omogućava definiranje zavisnosti između objekata pomoću XML konfiguracije u fazi kreiranja objekata
- Još se naziva i „*Inversion of Control*" (IoC)
- Objekti ne moraju kreirati ili dohvaćati instance drugih objekata o kojima ovise
- Ovisnosti se „injektiraju" u one objekte koji ih trebaju, tako da su kreiranim objektima postavljena sva svojstva (engl. *property*) u trenutku kad su kreirani

Dependency Injection

- Primjer korištenja:

```
public interface Student {
    void diplomiraj();
    Fakultet dohvatiFakultet();
}
```

```
public class StudentJave implements Student, Serializable {

   private Fakultet faks;
   private String titula;

   public StudentJave(Fakultet faks) {
      setFakultet(faks);
      titula = "Student Jave";
   }
   …
```

Dependency Injection

```
    public Fakultet dohvatiFakultet() {
        return faks;
    }

    private void setFakultet(Fakultet faks) {
        this.faks = faks;
    }

    public void diplomiraj() {
        faks = null;
        titula = "Java guru";
    }

    public String getTitula() {
        return titula;
    }
}
```

Dependency Injection

```
public class Fakultet implements Serializable {

   private String naziv;
   private int brojDiplomanata;

   public String getNaziv() {
      return naziv;
   }

   public void setNaziv(String naziv) {
      naziv = naziv;
   }
   …
```

Dependency Injection

```
public class Fakultet implements Serializable {

   private String naziv;
   private int brojDiplomanata;

   public String getNaziv() {
      return naziv;
   }

   public void setNaziv(String naziv) {
      naziv = naziv;
   }
   …
```

Dependency Injection

```
    public int getBrojDiplomanata() {
        return brojDiplomanata;
    }

    public void setBrojDiplomanata(int brojDiplomanata) {
        brojDiplomanata = brojDiplomanata;
    }

    public void inkrementirajBrojDiplomanada() {
        brojDiplomanata++;
    }
}
```

Dependency Injection

- Primjer korištenja:

```
<bean id="fakultet" class="pojo.Fakultet">
   <property name="naziv" value="TVZ" />
   <property name="brojDiplomanata" value="0"/>
</bean>

<bean id="student" class="pojo.StudentJave">
   <constructor-arg ref="fakultet" />
</bean>
```

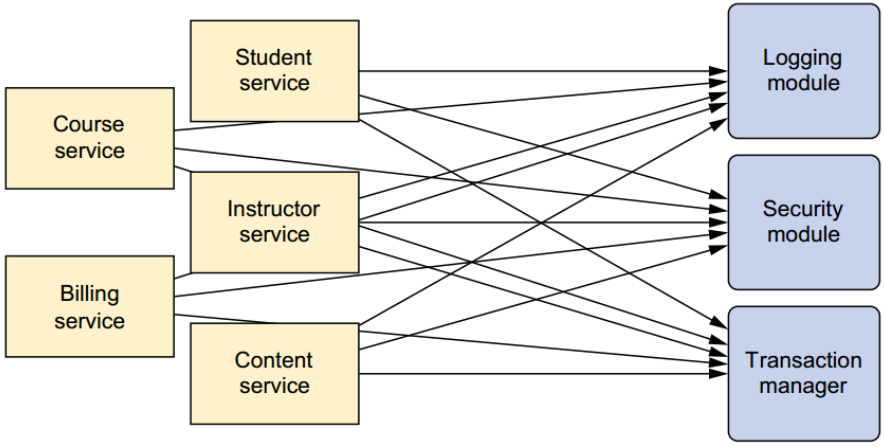
Dependency Injection

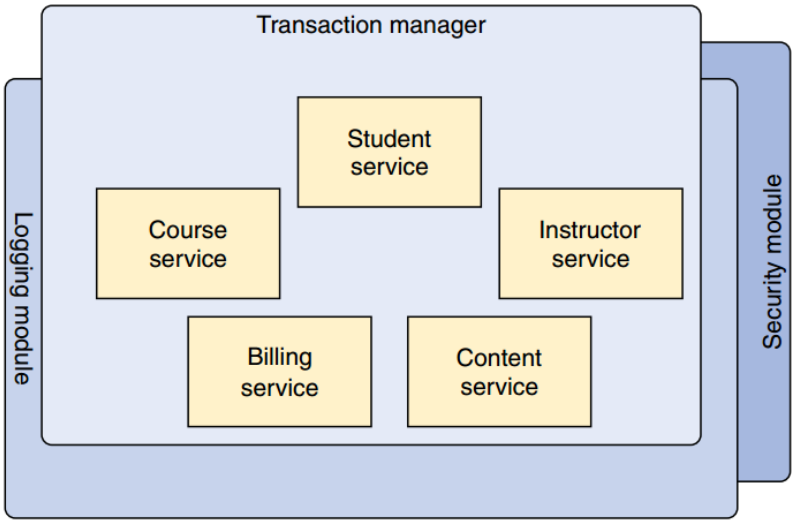
- Dohvaćanje objekata (*beanova*) iz *contexta* moguće je na sljedeći način:

```
ClassPathXmlApplicationContext context =
        new ClassPathXmlApplicationContext("studenti.xml");

Student student = (Student) context.getBean("student");
Fakultet faks = student.dohvatiFakultet();
```

Aspect oriented programming

- Aspektno orijentirano programiranje je programska paradigma koja povećava modularnost odvajanjem funkcionalnosti koje su zajedničke za različite dijelove aplikacije





Aspect oriented programming

- Na primjer, pomoću XML-a je moguće povezati metodu „inkrementirajBrojDiplomanada" koja se poziva na svaki poziv metode „diplomiraj" unutar klase „Student":

```
<aop:config>
    <aop:aspect ref="fakultet">
      <aop:pointcut id="diplomiranje"
    expression="execution(* pojo.Student.diplomiraj(..))" />
      <aop:after pointcut-ref="diplomiranje"
                 method="inkrementirajBrojDiplomanada"/>
    </aop:aspect>
  </aop:config>
```

Spring templateovi

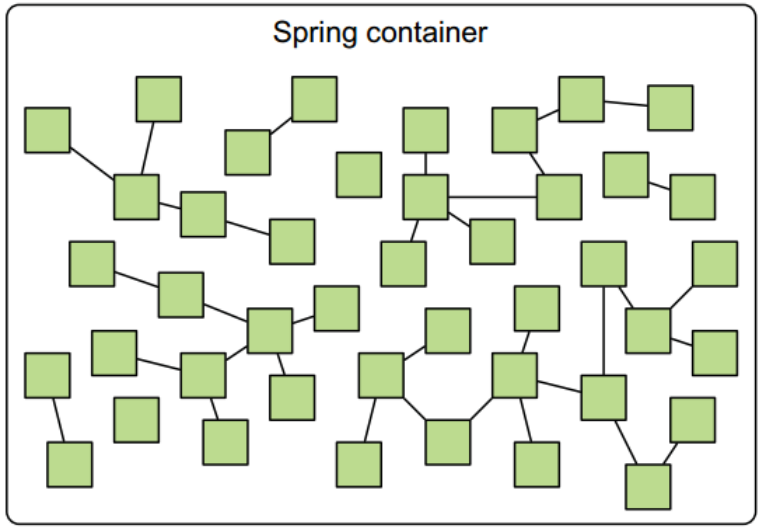
- Postoji mnogo primjera u Javi koji zahtijevaju pisanje istog „boilerplate" koda (kao što je pristupanje bazi podataka, pri čemu je potrebno kreirati i zatvoriti vezu s bazom podataka, inicijalizirati resurse i sl.)
- Umjesto toga moguće je koristiti pripremljene *templateove* koji dobar dio tih nužnih preduvjeta odrađuju automatski
- Primjer toga je Springov „jdbcTemplate"

Spring templateovi

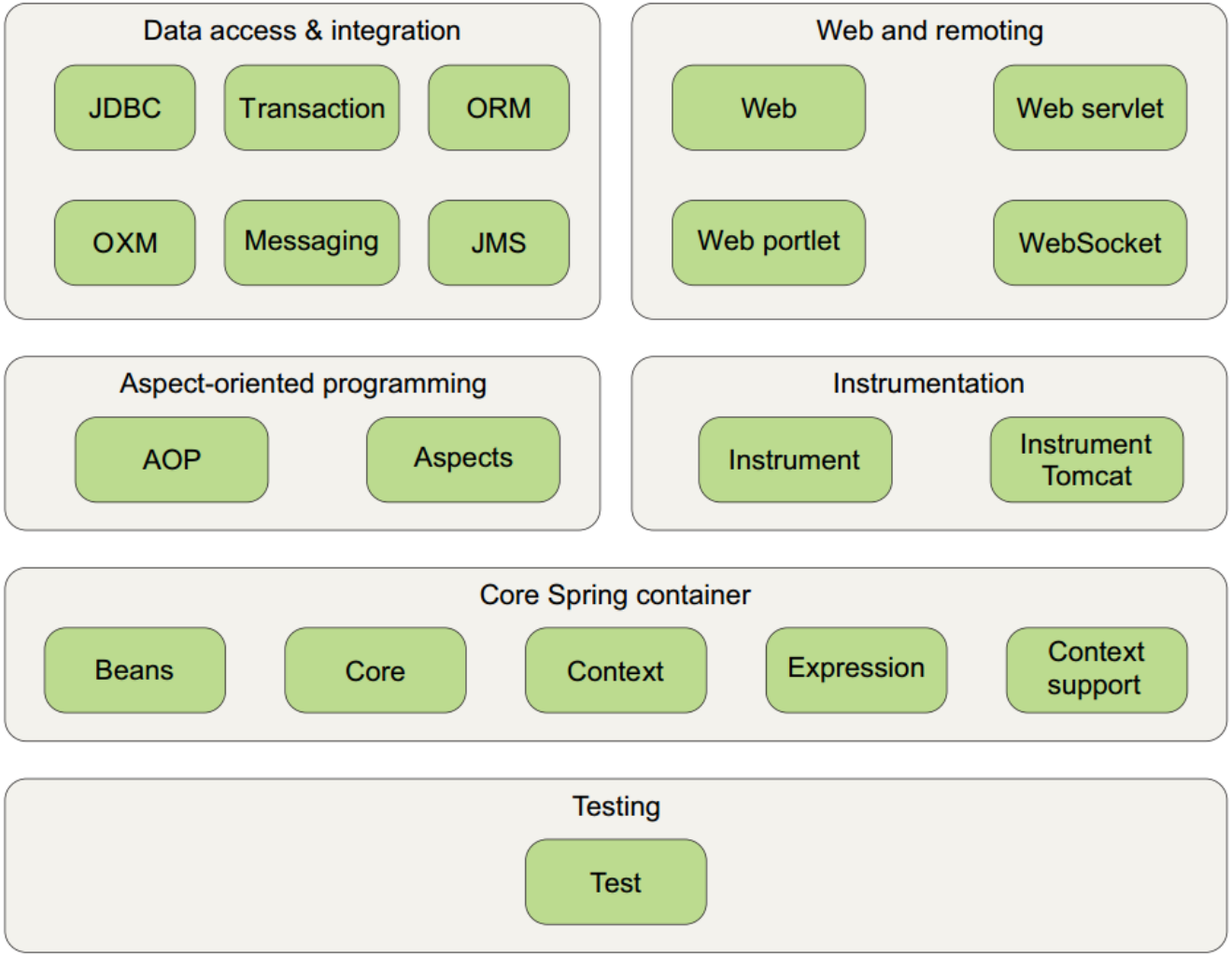
```
public Employee getEmployeeById(long id) {

  return jdbcTemplate.queryForObject(

     "select id, firstname, lastname, salary " +
     "from employee where id=?",

       new RowMapper<Employee>() {
         public Employee mapRow(ResultSet rs, int rowNum)
         throws SQLException {

            Employee employee = new Employee();
            employee.setId(rs.getLong("id"));
            employee.setFirstName(rs.getString("firstname"));
            employee.setLastName(rs.getString("lastname"));
            employee.setSalary(
                rs.getBigDecimal("salary"));

            return employee;
         }
       },id);
}
```

Spring *container*

- Jezgra Springovog *contexta*
- Korištenjem Dependency injectiona upravlja komponentama koje čine aplikaciju kroz kreiranje veza među njima
- Objekti time postaju jasniji, mogu se ponovno iskoristiti i lako testirati



Moduli Springa



Pitanja?