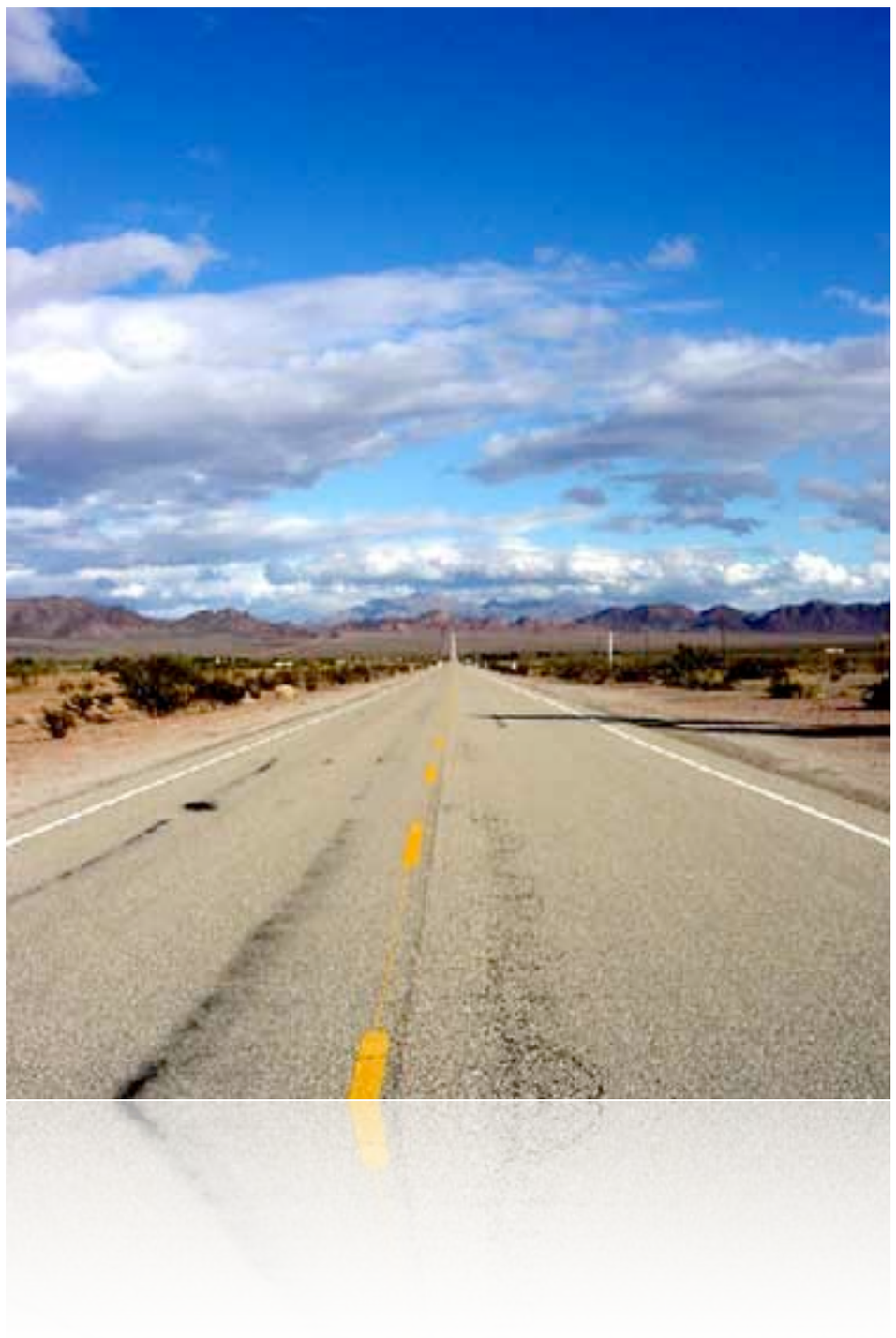
Spring Framework

Christoph Pickl

agenda

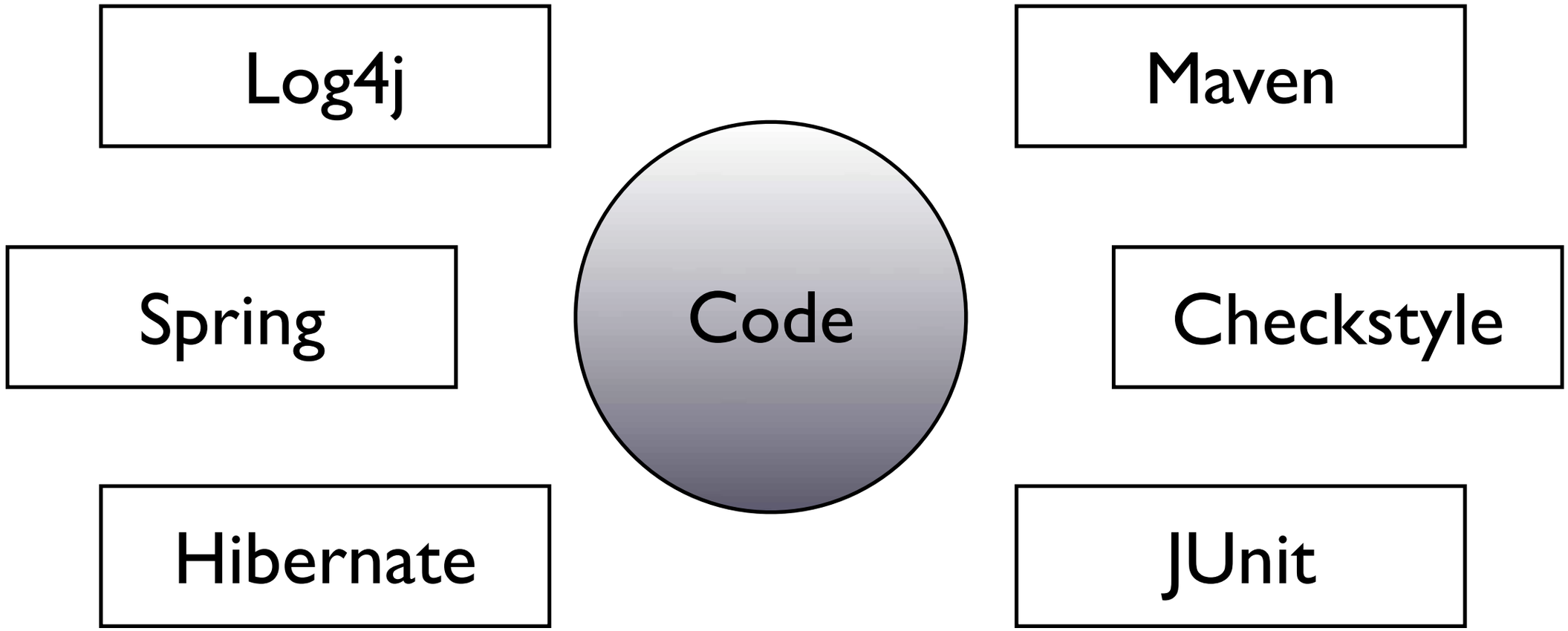
1. short introduction
2. basic declaration
3. medieval times
4. advanced features
5. demo



short introduction

Spring

common tool stack



Spring is a leightweight **inversion of control** and **aspect oriented** container framework

Spring makes developing **J2EE application easier** and **more fun**!

mission statement

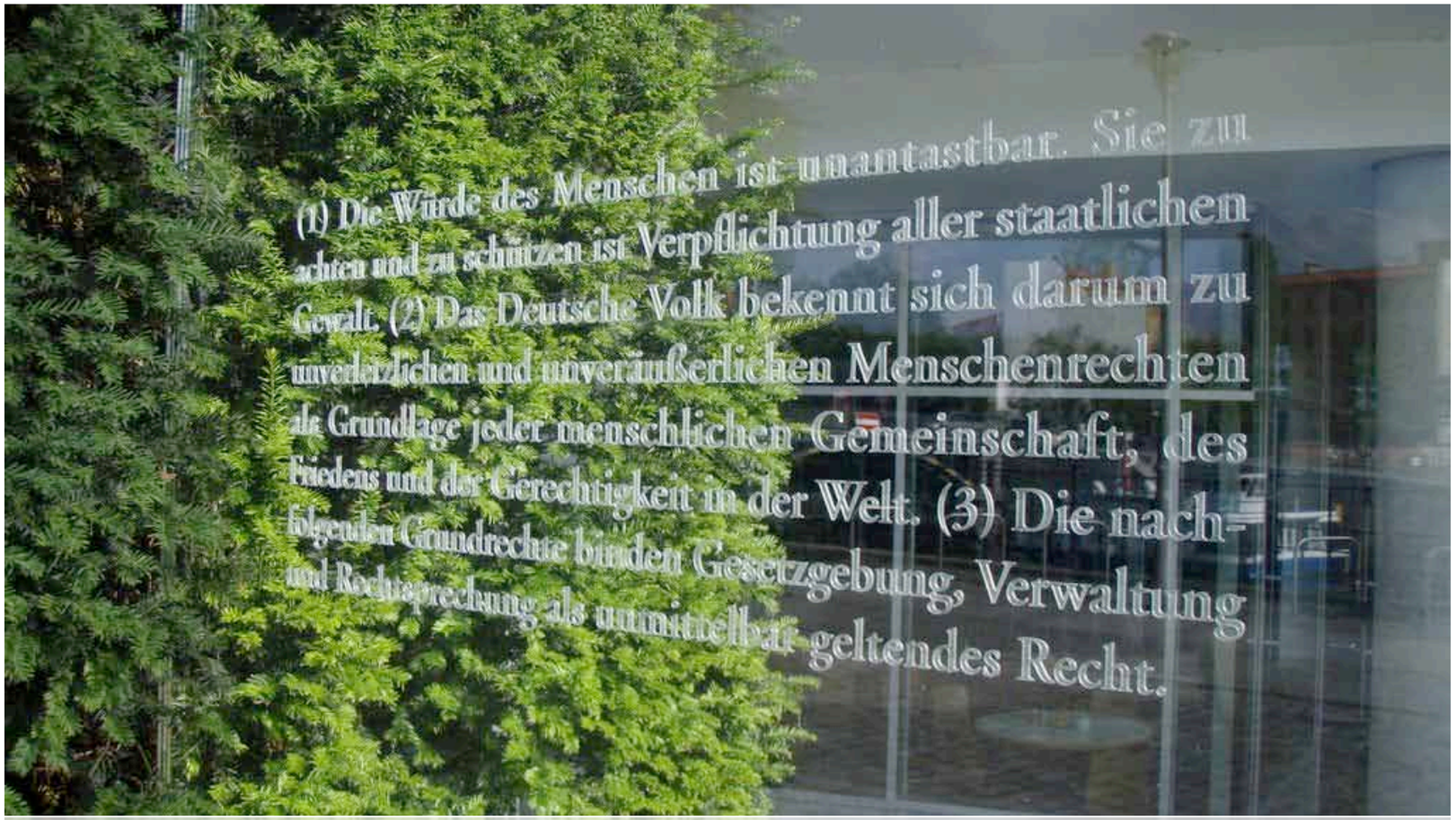
- "*J2EE should be easier to use*"
- "*it is best to program to interfaces, rather than classes [...]*"
- "*JavaBeans offer a great way of configuring applications*"
- "*OO is more important than any implementation technology [...]*"
- "*checked exceptions are overused in Java [...]*"
- "*testability is essential [...]*"

http://www.springframework.org/about

container age

- heavy (EJB) vs lightweight container (Spring)
- born as alternative to Java EE
- POJO development
- focuses on server development
- no compile dependency on the framework

http://martinfowler.com/articles/injection.html



basic declaration

http://de.wikipedia.org/wiki/Menschenwürde

initial setup

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- <!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
     "http://www.springframework.org/dtd/spring-beans.dtd"> -->
<beans
  xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd"
  >

  <!-- define your beans here -->

</beans>
```


http://static.springframework.org/spring/docs/2.0.x/reference/xsd-config.html

declare some beans

```
<!-- this is our first (singleton) bean -->
<bean id="myDao" class="jsug.MyDao" />

<bean id="myCtrl1" class="jsug.MyController1">
    <!-- invokes MyController1.setDao(MyDao) -->
    <property name="dao">
        <ref bean="myDao" />
    </property>
</bean>

<!-- share same dao in second controller -->
<bean id="myCtrl2" class="jsug.MyController2">
    <property name="dao">
        <ref bean="myDao" />
    </property>
</bean>
```

declare easy beans

```
<!-- reusable string bean -->
<bean id="SQL_CREATE" class="java.lang.String">
    <constructor-arg>
        <value>
            CREATE TABLE myTable ( ... );
        </value>
    </constructor-arg>
</bean>

<!-- pass some string values -->
<bean id="somePerson" class="jsug.Person">
    <property name="firstName" value="Christoph" />
    <property name="lastName" value="Pickl" />
</bean>
```

declare more beans

```
<!-- pass a list of beans -->
<bean id="someContainer" class="jsug.SomeContainer">
    <property name="modules">
        <list>
            <ref bean="module1" />
            <ref bean="module2" />
        </list>
    </property>
</bean>
<!-- or even a map -->
<bean id="someOtherContainer" class="jsug.SomeOtherContainer">
    <property name="mailMap">
        <map>
            <entry key="cpickl" value="cpickl@heim.at" />
            <entry key="fmotlik" value="fmotlik@heim.at" />
        </map>
    </property>
</bean>
```

medieval times

early beginning

```
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;

public class App { // implements InitializingBean
   public static void main(String[] args) {
        Resource beanResource =
           new ClassPathResource("/beans.xml");
//         new FileSystemResource("/my/app/beans.xml");

        BeanFactory beanFactory = new XmlBeanFactory(beanResource);

        IBean bean = (IBean) beanFactory.getBean("myBeanId");
        bean.doSomeMethod();
   }
}
```


http://www.ociweb.com/mark/programming/SpringIOC.html

~annotation approach

```
public class App {
    private SomeBean someBean;
    /**
     * @Bean("myBeanId")
     */
    public void setSomeBean(SomeBean someBean) {
        this.someBean = someBean;
    }
}

/**
 * @BeanId("myBeanId")
 * @BeanScope("singleton")
 */
public class SomeBean {
    // ...
}
```

today

```
// single code dependency to spring throughout our whole application
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App {
   public static void main(String[] args) {
      String[] beanDefinitions = { "/beans.xml" };
      new ClassPathXmlApplicationContext(beanDefinitions);
   }

   public App() {
      // spring will invoke this constructor for us
   }
   public void init() {
      // and spring will also invoke this initialize method for us
   }
}
```

evolution

- first attempt: **BeanFactory**
  - direct code dependency :(
- second attempt: **Annotations**
  - yet code is polluted
- third attempt: **Injection only**
  - complete transparent usage

advanced features

Spring

constant injection

```
<?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:util="http://www.springframework.org/schema/util"
  xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-2.0.xsd">

  <bean id=”myBean” class=”jsug.MyBean”>
    <property name=”someProperty”>
      <util:constant static-field=”jsuga.Constants.MY_CONSTANT” />
    </property>
  </bean>

</beans>
```

init methods

```
<bean id="myBean" class="jsug.MyBean" init-method="init">
    <constructor-arg index="0" value="someString" />
    <constructor-arg index="1" ref="someOtherBean" />
    <property name="firstName" value="Christoph" />
</bean>
```

```
public class MyBean {
    private final String someString;
    private String firstName;

    // #1 first of all, the bean will be created with proper constructor args
    public MyBean(String someString, Bean b) { this.someString = someString; }

    // #2 afterwards, all properties will be set via setter methods
    public void setFirstName(String firstName) { this.firstName = firstName; }

    // #3 finally, the initialize method will get invoked
    public void init() { /* initialize bean in here */ }
}
```

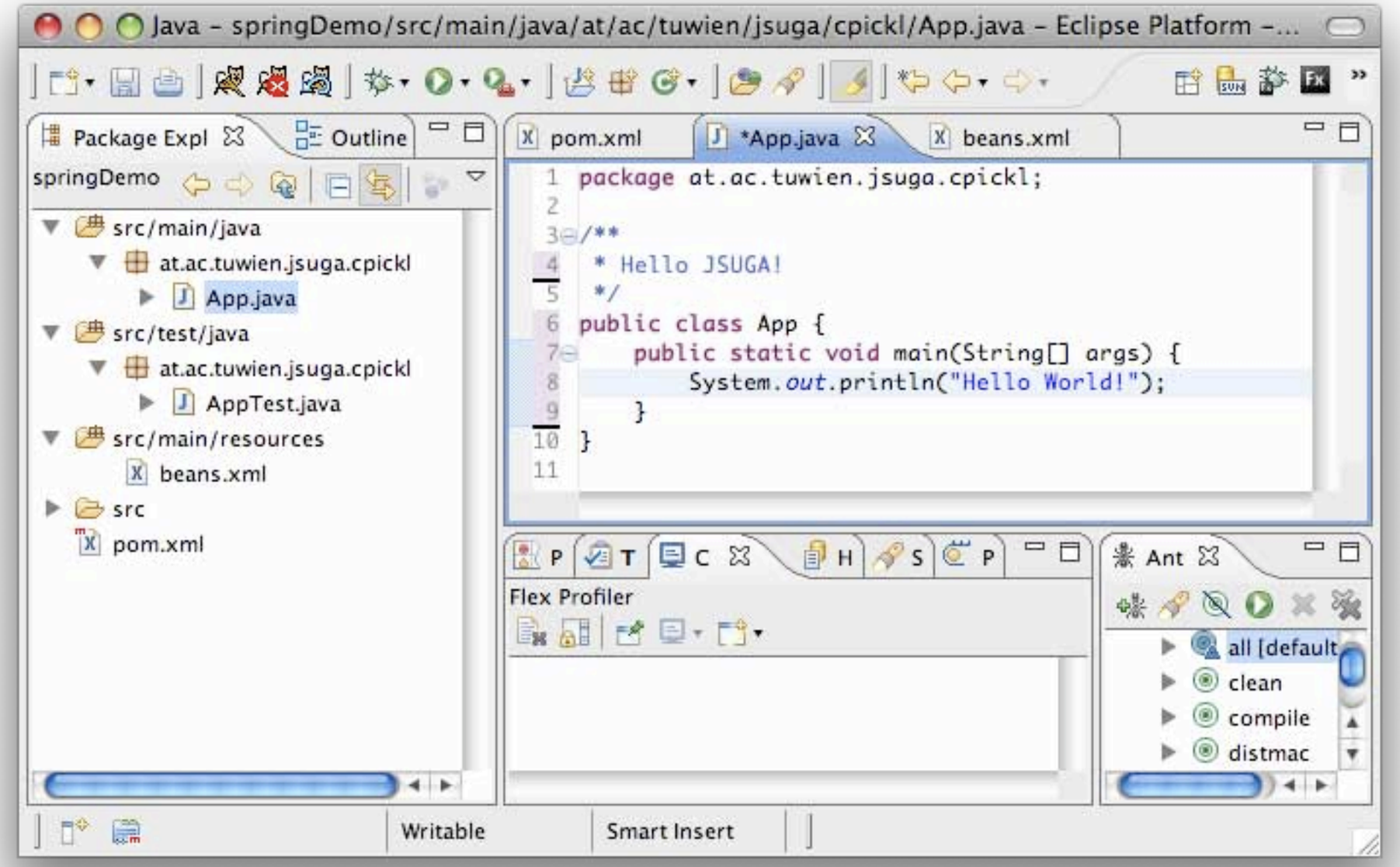
prototyping

```
<!-- share the same dao among all students/persons -->
<bean id="dao" class="jsuga.DaoImpl" scope="singleton" />

<!-- student implements the IPerson interface -->
<bean id="student" class="jsuga.Student" scope="prototype">
    <constructor-arg ref="dao" />
</bean>

<!-- the factory will create (any) persons for us -->
<bean id="personFactory" class="jsuga.PersonFactory">
    <lookup-method name="newPerson" bean="student" />
</bean>
```

... to be continued ...


Java - springDemo/src/main/java/at/ac/tuwien/jsuga/cpickl/App.java - Eclipse Platform -...
Package Expl
Outline
springDemo
src/main/java
at.ac.tuwien.jsuga.cpickl
App.java
src/test/java
at.ac.tuwien.jsuga.cpickl
AppTest.java
src/main/resources
beans.xml
src
pom.xml
pom.xml
*App.java
beans.xml
1 package at.ac.tuwien.jsuga.cpickl;
2
3 /**
4 * Hello JSUGA!
5 */
6 public class App {
7 public static void main(String[] args) {
8 System.out.println("Hello World!");
9 }
10 }
11
Flex Profiler
Ant
all [default
clean
compile
distmac
Writable
Smart Insert

demo

Framework

Web Flow

Web Services

Security (Acegi)

Dynamic Modules (OSGi)

Integration

IDE

JavaConfig

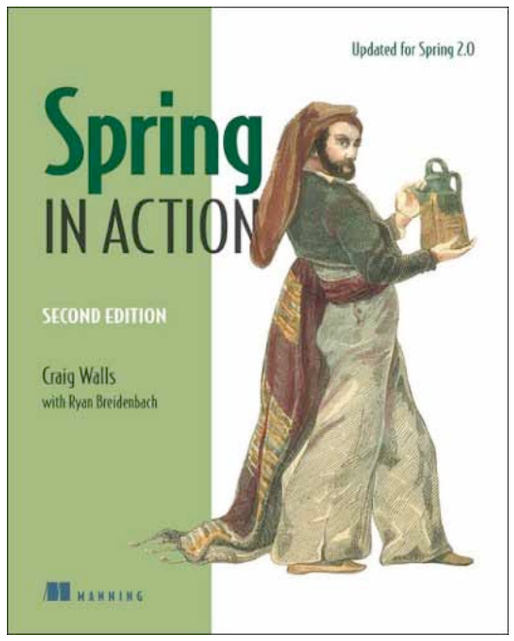
Rich Client

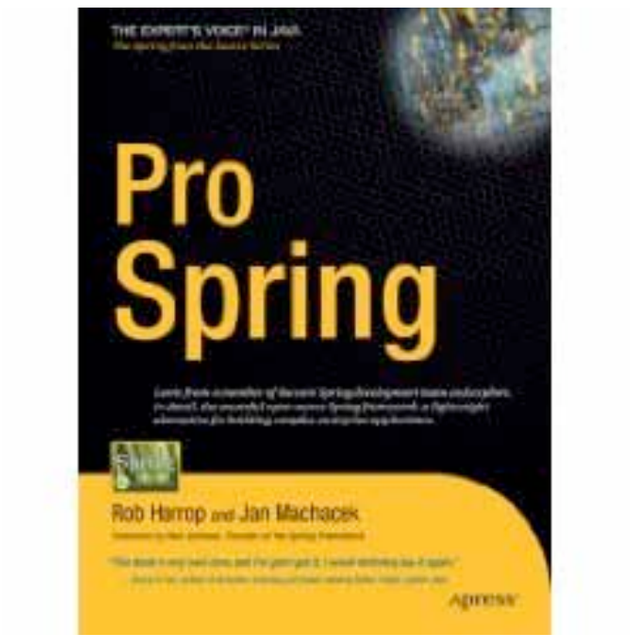
...

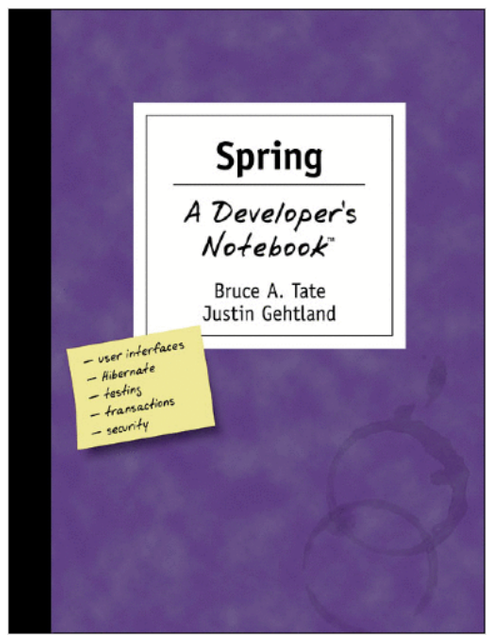
http://www.springframework.org/projects

further reading

- Spring official Website
  http://www.springframework.org/
- Spring Reference Documentation
  http://static.springframework.org/spring/docs/2.5.5/spring-reference.pdf
- Spring 2.5 on the Way to 3.0
  http://www.parleys.com/display/PARLEYS/Home#talk=19759132;title=Spring%202.5%20on%20the%20Way%20to%203.0;slide=1







Spring