Spring

and

the IoC Container
Spring overview

**DAO**
Spring JDBC Transaction management

**ORM**
Hibernate JPA TopLink JDO OJB iBatis

**AOP**
Spring AOP AspectJ integration

**JEE**
JMX JMS JCA Remoting EJBs Email

**Web**

**Core**
The IoC container
The IoC container

- IoC means *Inversion of Control* (Dependency Injection)
- The *IoC container* is the core component of the Spring framework
- A *bean* is an object that is managed by the IoC container
- The IoC container is responsible for containing and managing beans
- Spring comes with two types of containers
  - BeanFactory
  - ApplicationContext
The IoC container

Your Java objects (beans)

Spring IoC container

Configured system ready for use!
The BeanFactory

- Provides basic support for dependency injection
- Responsible for
  - Creating and dispensing beans
  - Managing dependencies between beans
- Lightweight – useful when resources are scarce
  - Mobile applications, applets
- *XMLBeanFactory* most commonly used implementation

```java
Resource xmlFile = new ClassPathResource("META-INF/beans.xml");
BeanFactory beanFactory = new XmlBeanFactory(xmlFile);
MyBean myBean = (MyBean) beanFactory.getBean("myBean");
```
The ApplicationContext

• Built on top of the BeanFactory
• Provides more enterprise-centric functionality
  – Internationalization of messages
  – AOP, transaction management
• Preferred over the BeanFactory in most situations
• Most commonly used implementation is the ClassPathXmlApplicationContext

```java
String xmlFilePath = "META-INF/beans.xml";
ApplicationContext context = new ClassPathXmlApplicationContext( xmlFilePath );

MyBean myBean = (MyBean) context.getBean( "myBean" );
```
Convenient container instantiation

- ApplicationContext instances can be created declaratively in web.xml using a ContextLoader

```xml
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath*:META-INF/beans.xml</param-value>
</context-param>

<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

ContextLoaderListener definition. The listener will inspect the contextConfigLocation parameter.
Dependencies

- The container injects dependencies when it creates a bean (the dependency injection principle)
- Setter-based dependency injection most convenient

```java
class DefaultStudentSystem implements StudentSystem {
    private String studentDAO;
    public void setStudentDAO(StudentDAO studentDAO) {
        this.studentDAO = studentDAO;
    }
}
```

```xml
<bean id="studentSystem" class="no.uio.inf5750.service.DefaultStudentSystem">
    <property name="studentDAO" ref bean="studentDAO" />
</bean>

<bean id="studentDAO" class="no.uio.inf5750.dao.HibernateStudentDAO"/>
```
Bean scopes

- A bean definition is a *recipe* for creating instances
  - Many object instances can be created from a single definition
- Spring will manage the *scope* of the beans for you
  - No need for doing it programmatically

<table>
<thead>
<tr>
<th>Scope</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>singleton</td>
<td>Scopes a single bean definition to a single object instance.</td>
</tr>
<tr>
<td>prototype</td>
<td>Scopes a single bean definition to any number of object instances.</td>
</tr>
</tbody>
</table>
The singleton scope

- Only one shared instance will ever be created by the container
- The single bean instance will be stored in a cache and returned for all requests
- Singleton beans are created at container startup-time

```
<bean id="studentDAO"
     class="no.uio.inf5750.dao.hibernate.StudentDAO"
     scope="singleton"/>
```

Spring configuration file (beans.xml)

Bean defined as singleton (not necessary since singleton scope is default)
The singleton scope

- Singleton per container – not by classloader
- Singleton is default scope in Spring

```
<bean id="bean1" class="..." scope="singleton"/>

<bean id="bean2" class="...">
    <property name="bean1">
        <ref bean="bean1"/>
    </property>
</bean>

<bean id="bean3" class="...">
    <property name="bean1">
        <ref bean="bean1"/>
    </property>
</bean>

```

Spring IoC container

The same instance is injected into both beans

```
<bean id="bean1" class="...">
    <property name="bean1">
        <ref bean="bean1"/>
    </property>
</bean>
```

bean1

bean2

bean3
The prototype scope

- A new bean instance will be created for each request
- Use prototype scope for stateful beans – singleton scope for stateless beans

```xml
<bean id="bean1" class="..." scope="prototype"/>

<bean id="bean2" class="...">
  <property name="bean1">
    <ref bean="bean1"/>
  </property>
</bean>

<bean id="bean3" class="...">
  <property name="bean1">
    <ref bean="bean1"/>
  </property>
</bean>
```

Spring IoC container

New instances created for each request

bean1 -> bean2

bean1 -> bean3
Customizing the lifecycle of a bean

- Spring lets you define callback methods which are invoked at bean initialization and destruction.
- The *init* method will be invoked after all properties are set on the bean.

```xml
<bean id="lifecycleBean"
     class="no.uio.inf5750.example.spring.lifecycle.LifecycleBean"
     init-method="init"/>
```

```java
public class LifecycleBean {
    public void init() {
        // do something useful initialization work
    }
}
```
Customizing the lifecycle of a bean

- The *destroy* method will be invoked when the container containing the bean is destroyed (not prototypes)
  - Most relevant in desktop applications
- Default lifecycle methods can be defined in the config

```xml
<bean id="lifecycleBean"
    class="no.uio.inf5750.example.spring.lifecycle.LifecycleBean"
    destroy-method="destroy"/>
```

```java
public class LifecycleBean
{
    public void destroy()
    {
        // do some useful destruction work
    }
}```
Internationalization

- Internationalization (i18n) is the process of decoupling the application from any specific locale
- Makes it possible to display messages in the user’s native language
- The ApplicationContext extends the MessageSource interface which provides i18n functionality
- Most commonly used implementation is the provided ResourceBundleMessageSource
The SaluteService

Spring looks for a bean called `messageSource`

Basename for the resourcebundles to use

MessageSource injected into DefaultSaluteService

gMessage is invoked

`public class DefaultSaluteService implements SaluteService`

`private MessageSource messages;`

`// set-method for messages`

`public String salute()`

```java
    return messages.getMessage( "salute", null, locale );
```

`param1: property key`
`param2: arguments`
`param3: Locale`
The SaluteService

- MessageResource follows the locale resolution and fallback rules of the standard JDK ResourceBundle

- i18n.properties (Fallback)
  - salute=Good morning!
  - farewell=Goodbye!

- i18n_fr_FR.properties
  - salute=Bonjour!
  - farewell=Au revoir!

- i18n_en_GB.properties
  - salute=Good morning!
  - farewell=Goodbye!

- i18n_de_DE.properties
  - salute=Guten Tag!
  - farewell=Auf Wiedersehen!
Resources

• Powerful access to low-level resources
• Avoids direct use of classloaders
• Simplifies exception handling
• Wrappers for regular Java classes
• Several built-in implementations:
  – ClassPathResource
  – FileSystemResource
  – URLResource

public interface Resource
    extends InputStreamSource
{
    boolean exists();
    boolean isOpen();
    URL getURL();
    File getFile();
    Resource createRelative( String p );
    String getFileName();
    String getDescription();
}

public interface InputStreamSource()
{
    InputStream getInputStream();
}
Summary

• IoC Container
  – BeanFactory, ApplicationContext

• Bean scopes
  – Singleton
  – Prototype

• Customization of bean lifecycle
  – Initialization
  – Destruction

• Internationalization
  – MessageSource

• Resources
  – Classpath, Filesystem, URL
Resources

• Lots of books on Spring:
  – Rod Johnson, Juergen Hoeller: *Expert One-on-One J2EE Development without EJB*
  – Justin Gehtland, Bruce A. Tate: *Better, Faster, Lighter Java*
  – Craig Walls and Ryan Breidenbach: *Spring in Action*

• The Spring reference documentation
  – www.springframework.org