TDP 2017 Web Week 2: Spring and RESTful Services Programming

Version 1.0.0 May 2017



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Legal Information

Copyright (C) Descriptor Systems 2001, 2017

All Rights Reserved

All trademarks are owned by their respective companies

You can contact Descriptor Systems at:
http://www.descriptor.com
jbarnum@descriptor.com

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

TDP 2017 Web Week 2: Spring and RESTful Services Programming

- Introductions
- Objectives

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introductions

- Please introduce yourself:
  - Name:
  - Java Background:
  - Your goal for the course:

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Course Description

Description: This course covers how to write Java Enterprise applications that use Spring, Servlets/JSP, SpringMVC and RESTful services and clients as well as microservices using Spring Boot.

Prerequisites: Basic Java background is assumed.

Audience: Java developers that want to learn the Spring Framework, RESTful services and Spring Boot.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Course Objectives

After taking this course, you will be able to:

- Understand the basics of the Spring Framework
- Code Web applications that use servlets/JSPs and SpringMVC
- Write RESTful clients and services using JAX-RS and Spring MVC REST
- Use Spring Boot to develop microservices

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Sample Agenda

Day 1

Introduction to the Course

Introduction to the Spring Framework

Lab 1: Introduction to Spring

Additional Configuration and Annotations

Lab 2: Annotations

Java Configuration

Lab 3: Java Configuration

Spring Beans

Lab 4: Spring Beans

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Sample Agenda, cont'd

Day 2

XML Dependency Injection

Lab 5: XML Dependency Injection

Intro to JEE

Intro to Servlets

Lab 6: Intro Servlets

Intro JSP

Lab 7: Intro JSP

Introduction to Spring MVC

Lab 8: Spring and the Web

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Sample Agenda, cont'd

Day 3

Spring and JDBC

Lab 9: Spring and JDBC

Transactions in Spring OPTIONAL

Lab 10: Spring Transactions OPTIONAL

Introduction to SOAP

Lab 11: SOAP Hello, World

Introduction to REST

Lab 12: Intro REST

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Sample Agenda, cont'd

Day 4

Intro to JAX-RS REST

REST CRUD

Lab 13: REST JAX-RS

Spring MVC REST

Lab 14: Spring MVC REST

Intro Spring Boot

Lab 15: Intro to Spring Boot

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Sample Agenda, cont'd

Day 5

Additional Spring Boot Topics

Lab 16: More Spring Boot

Review

TDP Exam

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Course Logistics

- Schedule
  - Start time
  - Lunch
  - End time
  - Breaks

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to the Spring Framework

- What is the Spring Framework?
- Spring Architecture
- Spring Hello, World

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is the Spring Framework?

- The **Spring Framework** is an open-source, Java and JEE framework that aims to make developing applications easier
- Spring applications are based on **plain old Java objects** (POJO) also known as JavaBeans
- Objects in a Spring application should not depend on Spring and should be easy to test



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

History of Spring

- The Spring Framework was created in 2003 by Rod Johnson based on material from his book *Expert One-on-One J2EE Design and Development*
- Spring 2 was released in 2006 and added basic support for Java 5 and configuration extensions
- Spring 2.5 adds support for Java 5 annotations
- Spring 3 contains support for RESTful Web services and other new features including **Java-based configuration**
- Spring 4 adds support for Java 8 and removes many old deprecated classes and methods

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

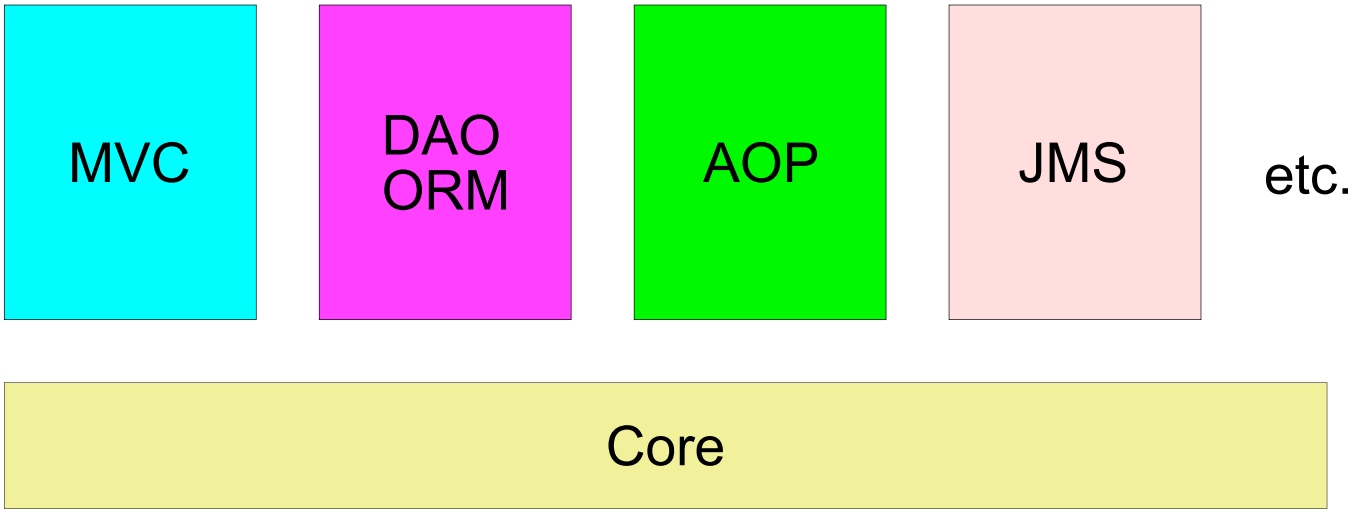
Spring Fundamentals

- Use of Spring beans (POJOs) instead of requiring classes to implement framework interfaces
- Loose coupling between objects
- **Dependency injection** (DI) to give objects resources instead of requiring lookups
- **Aspect-oriented programming** (AOP) to free objects from performing system functions such as logging and transactions
- Spring doesn't re-invent the wheel, but can integrate with other frameworks (e.g. Java Persistence)

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring Modules

- The Spring Framework itself is modular so you can use only the pieces needed by your application
- This makes the Spring runtime lightweight



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Writing Spring Applications

- To write Spring programs, you:
    - Write Spring **bean** classes
    - Configure the **container** so it can manage and initialize the beans
    - Bootstrap the container, retrieve bean references and write application code

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is a Spring Bean?

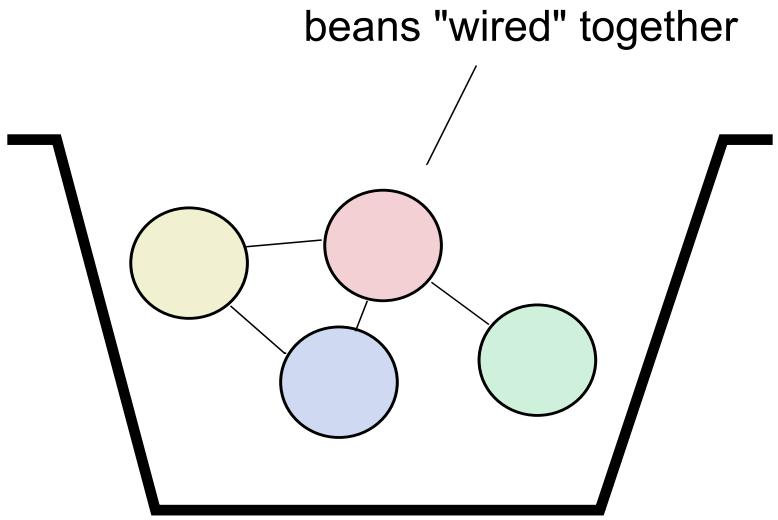
- Unlike in earlier frameworks, Spring **beans** don't need to implement any non-domain interfaces
- These POJOs are like JavaBeans, though they don't need to implement Serializable and such

```
1    @Component
2    public class Student
3    {
4       private int id;
5       . . .
6
7       public int getId() {return id;}
8       public void setId(int i) {id = i;}
9       . . .
10   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Spring Container

- To manage the lifecycle of the beans, Spring provides a lightweight **container**
- Spring apps create the container, then retrieve bean references from the container
- The container can **wire**, or hook up beans to each other via *dependency injection*

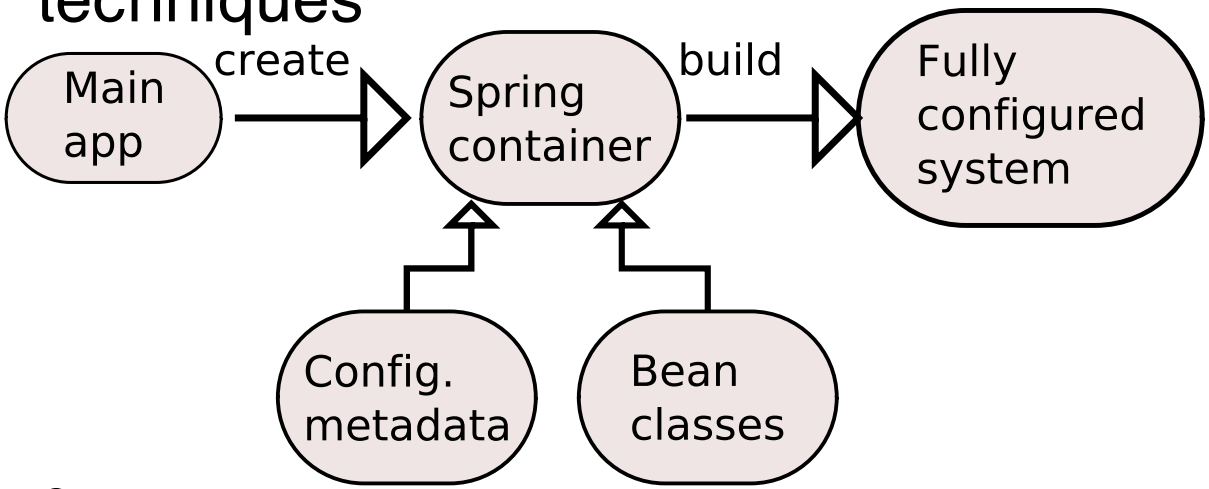


A Spring container

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring the Container

- Spring provides three* ways to configure Spring beans:
    - **Annotations** - Markup bean classes themselves
    - **Java configuration** - Write one or more classes that perform configuration
    - **XML** - Write one or more XML configuration files
- Spring applications can use any mixture of these techniques



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Programming To Interfaces

- To decrease coupling between objects, Spring developers often start by writing an **interface** for some of their beans
- This eases maintenance, and makes beans easier to test

```
1    public interface MyService
2    {
3        public void myMethod();
4    }
5
6    public class MyClass implements MyService
7    {
8       public void myMethod() {...};
9    }
10
11   public class OtherClass
12   {
13      private MyService theService;
14   . . .
15   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Dependent Objects Without Spring

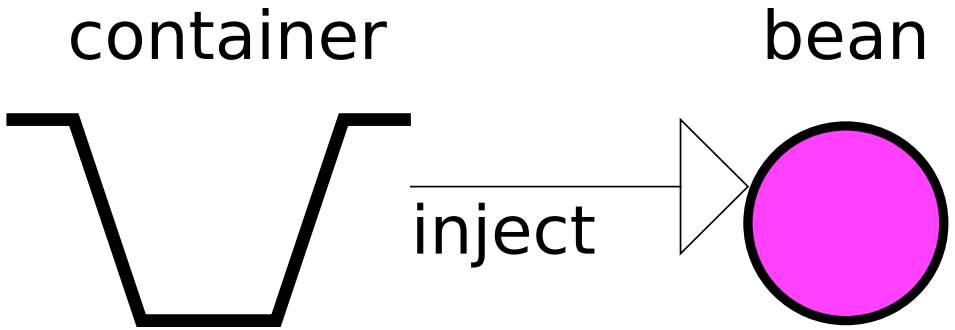
- Most real-world objects use other objects; we refer to this as a **dependency**
- In non-Spring environments, objects typically satisfy their own dependencies

```
1    public class OtherClass
2    {
3       // MyService is an interface
4       private MyService theService;
5
6       public OtherClass()
7       {
8          // MyClass implements MyService
9          theService = new MyClass();
10         . . .
11      }
12   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Dependency Injection

- **Dependency injection** is a technique in which the container instantiates and provides references for beans
- To inject a bean reference, the container directly accesses a field or calls a bean's "set" method or constructor
- DI lessens coupling between beans and decouples beans from services such as JNDI



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring with Annotations

- Spring lets you configure injection using **annotations**

```
1    package sample;
2
3    @Component("myService")
4    public class MyClass implements MyService
5    {
6       public void myMethod() {...};
7    }
```

```
1    package sample;
2
3    @Component("other")
4    public class OtherClass
5    {
6       @Resource(name="myService")
7       private MyService theService;
8       public void setTheService(MyService svc) {...}
9    . . .
10   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring with Java Config

- Java configuration is the newest approach and lets you configure using the same language you use to write the beans themselves
- To use Java config, create one or more Java classes annotated with **@Configuration** containing bean-creation methods annotated with **@Bean**

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring with Java Config, cont'd

```
1    @Configuration
2    public class MyConfig
3    {
4      @Bean
5      public MyService myService()
6      {
7         return new MyClass();
8      }
9
10     @Bean
11     public OtherClass otherClass()
12     {
13       OtherClass other = new OtherClass();
14       other.setTheService(myService());
15       return other;
16     }
17   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring with XML

- XML configuration was the original configuration approach in Spring

```
1     <!-- sample.xml -->
2
3     <?xml version="1.0" encoding="UTF-8"?>
4     <beans ...>
5
6       <bean id="myService" class="sample.MyClass"/>
7
8       <bean id="other" class="sample.OtherClass">
9         <property name="theService" ref="myService" />
10      </bean>
11
12    </beans>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Injecting Simple Values

- You can use annotations, XML or Java config to intialize properties to values

```
1    // using annotations
2    public class MyClass
3    {
4       @Value("1234")
5       private int myInteger;
6       public void setMyInteger(int myInt) {...}
7    }
```

```
1    <!-- using XML -->
2
3    <bean id="myBean" class="sample.MyClass">
4      <property name="myInteger" value="1234" />
5    </bean>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring Annotation-Based Apps

- To enable Spring to scan your beans for annotations, you need to provide at least a minimal XML file

```
1     <!-- sample.xml -->
2
3     <?xml version="1.0" encoding="UTF-8"?>
4     <beans ...>
5
6       <context:component-scan base-package="sample"/>
7
8     </beans>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Creating the Container (Annotation or XML)

- Spring applications with a *main* method can create a Spring container and then retrieve beans from it
- Spring provides several techniques for creating a container - here we retrieve the configuration file from the file system and use it to create the container

```
1    AbstractApplicationContext ctx =
2     new FileSystemXmlApplicationContext("sample.xml");
3
4    OtherClass oc =
5       (OtherClass)ctx.getBean("other");
6
7    . . .
8
9    ctx.close();
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Creating the Container (Java Configuration)

- If you are using Java configuration, you do not need any XML configuration - just reference the configuration class

```
1    AnnotationConfigApplicationContext ctx =
2      new AnnotationConfigApplicationContext(
3         MyConfig.class);
4
5    OtherClass oc =
6       (OtherClass)ctx.getBean("other");
7
8    . . .
9
10   ctx.close();
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Steps for a Simple Spring Application

- As desired, create interfaces for bean classes
- Implement the beans, using dependency injection as required
- Write an XML configuration file (if necessary)
- Write a class with a *main* method that creates a container and retrieves bean instances
- Invoke the beans' methods to perform the application function

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Hello World From Spring

HelloService.java

HelloImpl.java

Greeter.java

HelloApp.java

hello.xml

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    package hello;
2
3    public interface HelloService
4    {
5        public void sayHello();
6    }
```

```
1    package hello;
2
3    import org.springframework.stereotype.Component;
4
5    @Component("helloService")
6    public class HelloImpl implements HelloService
7    {
8        public void sayHello()
9        {
10           System.out.println("Hello, from Spring");
11       }
12   }
```

```java
package hello;

import javax.annotation.Resource;

import org.springframework.stereotype.Component;

@Component("greeter")
public class Greeter
{
    @Resource(name="helloService")
    private HelloService helloService;

    public void sendGreetings()
    {
        helloService.sayHello();
    }
}
```

```
1    package hello;
2
3    import org.springframework.context.support.AbstractApplicationContext;
4    import org.springframework.context.support.FileSystemXmlApplicationContext;
5
6    public class HelloApp
7    {
8        public static void main(String[] args) throws Exception
9        {
10           AbstractApplicationContext factory =
11              new FileSystemXmlApplicationContext("hello.xml");
12
13           Greeter greeter = (Greeter) factory
14                   .getBean("greeter");
15           greeter.sendGreetings();
16
17           ctx.close();
18       }
19   }
```

```
<?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-3.1.xsd
 http://www.springframework.org/schema/context
 http://www.springframework.org/schema/context/spring-context-3.1.xsd">

    <context:component-scan base-package="hello"/>


</beans>
```

Chapter Summary

In this chapter, you learned

- About the basic architecture of the Spring Framework
- How to use Dependency Injection to inject resources into a bean

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Additional Configuration and Annotations

- Injecting Collections
- Autowiring
- Annotations

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring Spring Beans

- Spring provides several techniques so that developers can provide metadata for beans, including:
  - XML
  - Annotations
  - Spring 3 Java Configuration
- XML configuration is the traditional approach

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring XML Namespaces

- XML defines the notion of **namespaces** to partition names and to prevent name collisions
- A namespace specifies a URI and a prefix
- Spring defines several XML namespaces that are used to configure various Spring components

Prefix	URI
aop	http://www.springframework.org/schema/aop
beans	http://www.springframework.org/schema/beans
context	http://www.springframework.org/schema/context
jee	http://www.springframework.org/schema/jee
jms	http://www.springframework.org/schema/jms
lang	http://www.springframework.org/schema/lang
tx	http://www.springframework.org/schema/tx
util	http://www.springframework.org/schema/util

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring XML Namespaces, cont'd

- In the XML, it's also a good idea to specify the **schemaLocation** of each namespace so that your XML-aware editor can validate your entries and provide code completion

spring.xml

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

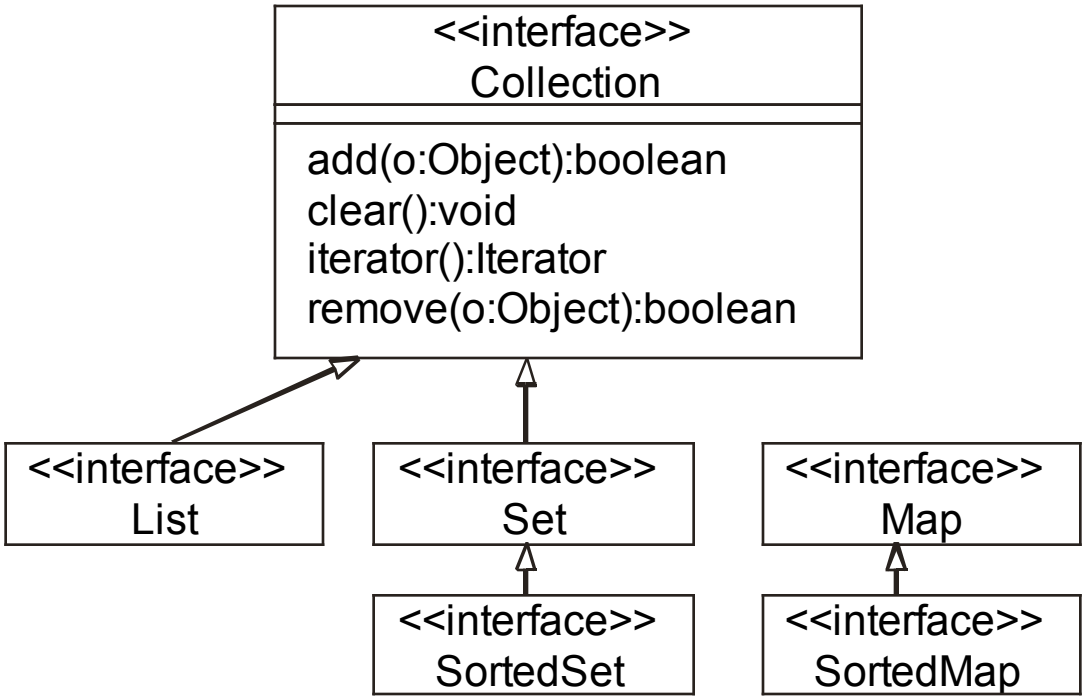
```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:jee="http://www.springframework.org/schema/jee"
    xmlns:jms="http://www.springframework.org/schema/jms"
    xmlns:lang="http://www.springframework.org/schema/lang"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-2.5.xsd
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-2.5.xsd
        http://www.springframework.org/schema/jee
        http://www.springframework.org/schema/jee/spring-jee-2.5.xsd
        http://www.springframework.org/schema/lang
        http://www.springframework.org/schema/lang/spring-lang-2.5.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-2.5.xsd
        http://www.springframework.org/schema/util
        http://www.springframework.org/schema/util/spring-util-2.5.xsd">

    <aop:aspectj-autoproxy/>
    <bean/>
    <context:component-scan base-package=""/>
    <jee:jndi-lookup jndi-name=""/>
    <lang:groovy/>
    <tx:annotation-driven/>
    <util:constant static-field=""/>

</beans>
```

Java Collection Framework Review

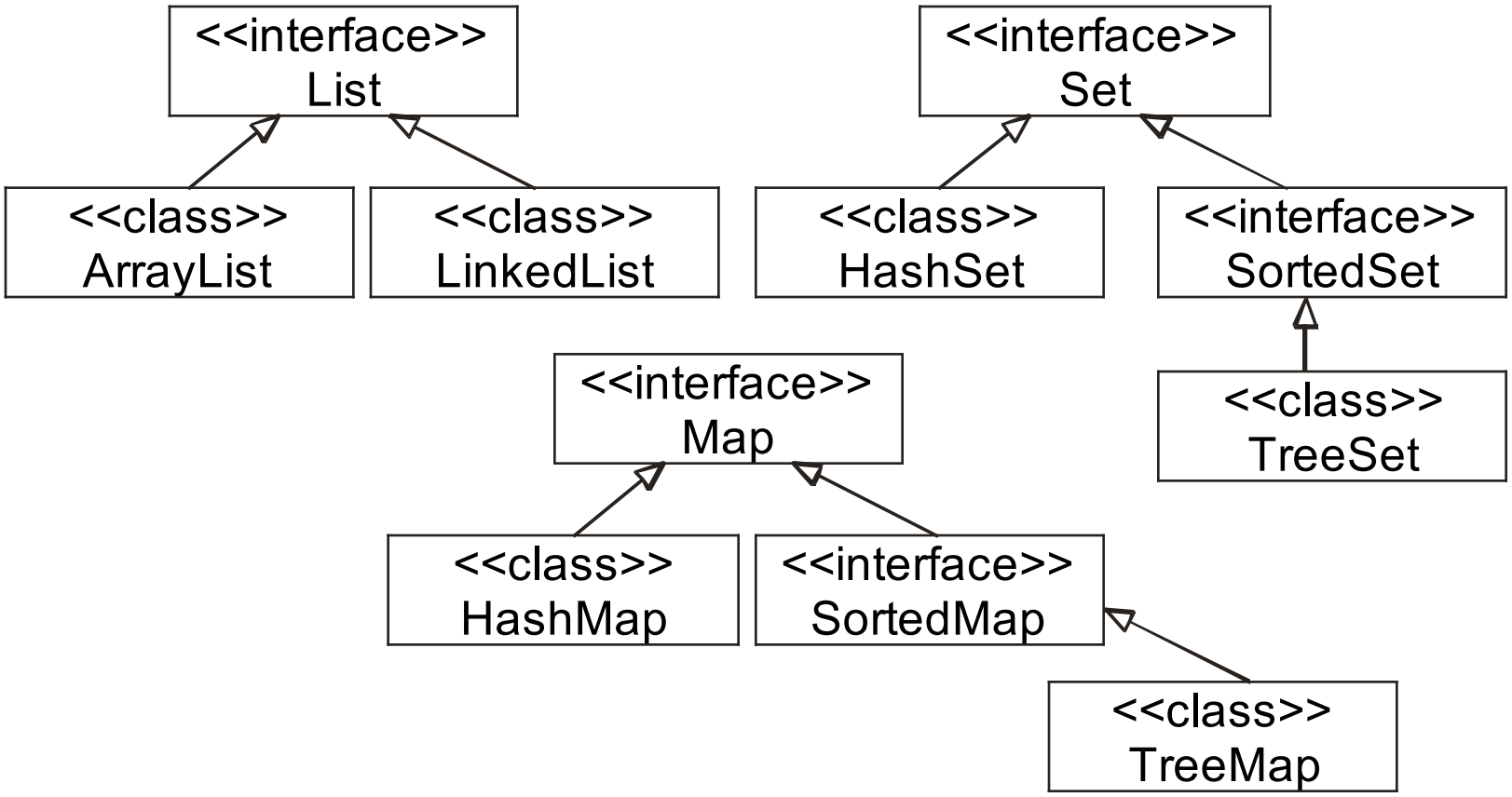
- The *Collection* interface is the base type for the collections in Java
- **Lists** are sequential collections that allow duplicate elements
- **Sets** reject duplicates and are optionally sorted
- **Maps** are key/value collections where the key must be unique



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Java Collection Framework Review, cont'd

- Behind the scenes, collections use well-known data structures to store elements



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

XML: Injecting Collections as Properties

- To inject a collection, you define data in the Spring configuration file and match it to a property in the bean
- The bean property can be an array, any collection interface, or a concrete collection type
- To inject, the container creates the appropriate Java type (array or collection), populates it, then calls the "set" method

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring XML Collection Configuration

- Spring provides these XML configuration elements to define collections:

- **<list>** - duplicates allowed
- **<set>** - no dupes allowed
- **<map>** - name/value pairs of any type
- **<props>** - name/value pairs of strings

- The data in the collection can be either simple data types or references to beans defined elsewhere in the configuration

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

XML: Injecting a List

- You can use the Spring **list** configuration element to inject into any Java collection or array property

```
1    <bean id="student1" class="university.Student">
2      <property name="scores">
3         <list>
4           <value>88</value>
5           <value>100</value>
6           <value>100</value>
7         </list>
8      </property>
9    </bean>
```

```
1    private List<Integer> scores;
2    public void setScores(List<Integer> scores)
3    {
4      this.scores = scores;
5    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

XML: Injecting a Set

- You can use the Spring **set** configuration element to inject into any Java collection or array property
- Spring will filter duplicates before injecting

```
1    <bean id="student2" class="university.Student">
2       <property name="advisors">
3         <set>
4           <ref bean="advisor1"/>
5           <ref bean="advisor2"/>
6           <ref bean="advisor2"/>
7         </set>
8       </property>
9    </bean>
```

```
1    private Set<Advisor> advisors;
2    public void setAdvisors(Set<Advisor> advisors)
3    {
4      this.advisors = advisors;
5    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

XML: Injecting a Map

- Maps are key/value pairs; you can specify either a hard-coded value or reference for either the map key or value

```
1    <bean id="advisor3" class="university.Advisor">
2      <property name="coursesTaught">
3        <map>
4          <entry>
5            <key><value>chem101</value></key>
6            <ref bean="chem101"/>
7          </entry>
8          <entry>
9            <key><value>chem102</value></key>
10           <ref bean="chem102"/>
11         </entry>
12       </map>
13     </property>
14   </bean>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

XML: Injecting java.util.Properties

- A java.util.Properties collection is a special case of a Map where the keys and values are both Strings

```
1    <bean id="calc1" class="university.Course">
2      <property name="materials">
3        <props>
4          <prop key="Calculus for Dummies">Textbook</prop>
5          <prop key="History of Calculus">Article</prop>
6        </props>
7      </property>
8    </bean>
```

```
1    private Properties materials;
2    public void setMaterials(Properties materials)
3    {
4       this.materials = materials;
5    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to XML Autowiring

- Spring configuration files can get quite verbose
- Using **autowiring** can reduce the bulk at the expense of being less explicit
- Autowiring lets you omit some property-injection configuration
- Autowiring only works with injected references, not values
- Autowiring is disabled by default

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

XML Autowiring By Name

- The container examines each bean property and searches for another bean with the same name (or ID) and then injects the reference

```
1    <bean id="theAdvisor" class="university.Advisor"/>
2
3    <bean id="student1" class="university.Student"
4       autowire="byName">
5    </bean>
```

```
1    private Advisor theAdvisor;
2    public void setTheAdvisor(Advisor theAdvisor)
3    {
4      this.theAdvisor = theAdvisor;
5    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

XML Autowiring By Type

- The container examines each bean property and searches for another bean with the same type and then injects the reference

```
1     <bean id="theAdvisor" class="university.Advisor"/>
2
3     <bean id="student1" class="university.Student"
4        autowire="byType">
5     </bean>
```

```
1     private Advisor theAdvisor;
2     public void setTheAdvisor(Advisor theAdvisor)
3     {
4       this.theAdvisor = theAdvisor;
5     }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Should You Autowire using XML?

- Autowiring can cut bulk in the configuration, but is more "magical"
- Many Spring experts recommend against XML autowiring

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Annotations

- **Annotations** were added to Java in JDK5 and let you provide **metadata** in your source code
- You can use annotations to decorate classes, fields or methods
- Java defines several built-in annotations and frameworks like Spring can define their own "custom" annotations
- The Java compiler ignores custom annotations, so they need to be processed by the framework

```
1    @MyCustomAnnotation
2    public class Test
3    {
4      @SuppressWarnings("unused")
5      private int x;
6
7       @Override
8       public String toString() {...}
9    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Annotations and Spring

- Annotations were first added to Spring in version 2.0 and expanded in version 2.5
- Spring defines its own annotations and also uses annotations from other Java specifications (e.g. EJB3 and AOP)

```
1    import org.springframework.stereotype.Component;
2    import javax.annotation.Resource;
3
4    @Component("theStudent")
5    public class Student
6    {
7      @Resource(name="theAdvisor")
8      private Advisor theAdvisor;
9      public void setTheAdvisor(Advisor theAdvisor)
10       {...}
11   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Enabling Annotations

- In Spring, annotations are processed by BeanPostProcessors, so you must configure the post processor(s) in the Spring configuration
- Spring lets you enable ALL annotations or selective enable only the ones you are using

```
<!-- option 1: enable individual BeanPostProcessors -->
<bean class=
  "org.springframework.context.annotation.CommonAnnotationBeanPostProcessor"/>
<bean class=
  "org.springframework.beans.factory.annotation.RequiredAnnotationBeanPostProcessor"


<!-- option 2: enable ALL BeanPostProcessors -->
<context:annotation-config/>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The @Value Annotation

- This is a Spring-specific annotation that lets you inject a simple value
- This annotation is processed by the CommonAnnotationBeanPostProcessor

```
1    public class Student
2    {
3    . . .
4
5      @Value("3.33")
6      public void setGpa(double gpa)
7      {. . .}
8    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The @Required Annotation

- This is a Spring-specific annotation that lets you indicate a that a property MUST be injected during initialization
- If not, Spring throws BeanCreationException
- This annotation is processed by the RequiredAnnotationBeanPostProcessor

```
1    public class Student
2    {
3    . . .
4
5      @Required
6      public void setGpa(double gpa)
7      {. . .}
8    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The @Resource Annotation

- This is a standard JEE annotation that Spring uses to perform dependency injection
- This annotation is processed by the CommonAnnotationBeanPostProcessor

```
1     public class Student
2     {
3     . . .
4
5       @Resource(name="advisor1")
6       public void setTheAdvisor(Advisor theAdvisor)
7       {. . .}
8
9     }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The @Resource Annotation, cont'd

- You can use @Resource to inject several standard Spring objects, including the ApplicationContext (no need to implement ApplicationContextAware lifecycle interface)
- If your Spring program is running in a JEE environment, you can also use @Resource to inject objects that Spring looks up from JNDI

```
1    public class Student
2    {
3      @Resource
4      private ApplicationContext ctx;
5
6      . . .
7    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The @Component Annotation

- This is a Spring-specific annotation that obviates the need to even define beans in the configuration file
- You must write the **context:component-scan** element in the configuration file

```
1     <beans ...>
2
3        <!-- <context:annotation-config/> -->
4        <context:component-scan base-package="university"/>
5
6     </beans>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The @Component Annotation, cont'd

```
1    @Component("advisor1")
2    public class Advisor
3    {
4    . . .
5    }
```

```
1    @Component("student1")
2    public class Student
3    {
4      private Advisor theAdvisor;
5
6      @Resource(name="advisor1")
7      public void setTheAdvisor(Advisor theAdvisor)
8      {. . .}
9    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Autowiring with Annotations

- You can use the **@Autowired** annotation to perform autowiring without doing any XML configuration

```
1     public class MyBean
2     {
3       @Autowired
4       private ServiceBean1 sob1;
5
6       @Autowired
7       public void doSomething(ServiceBean sob2)
8       {
9           . . .
10      }
11
12      @Autowired
13      public MyBean(ServiceBean sob3) { . . . }
14    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The @Qualifier Annotation

- You can use **@Qualifier** with autowiring when there are multiple candidate beans

```
1    @Component("firstImpl")
2    public class FirstImpl implements MyService
3    {
4    . . .
5    }
6    @Component("secondImpl")
7    public class SecondImpl implements MyService
8    {
9    . . .
10   }
11   public class Client
12   {
13     @Autowired @Qualifier("secondImpl")
14     private MyService theService;
15    . . .
16   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Other Annotations

- Spring defines many more annotations, including:

Annotation	Notes
@Scope	Configures bean scope
@Controller	For Spring MVC @Component stereotype
@RequestMapping	For Spring MVC
@RequestParameter	For Spring MVC
@Service	@Component stereotype
@Respository	@Compoment stereotype for DAOs
@Transactional	Configure transactions
@Pointcut	For Spring AOP
@Before	For Spring AOP
@RunWith	For testing

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring and JSR-330

- The **Dependency Injection for Java** (JSR-330) specification defines a standard set of annotations for DI in Java
- JSR-330 defines @Named and @Inject annotatations similar to Spring's @Component and @Autowired
- Spring 3 applications that need to be standards compliant can use the JSR-330 annotations if they include the **javax-inject.jar** in the CLASSPATH

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Annotations vs XML Configuration

- Using annotations can drastically reduce the amount of XML you need to write
- Things to consider:
    - Configuration is scattered across the application, rather in central location(s)
    - Code is less "pure" POJO since it needs additional CLASSPATH entries for the annotations
    - You cannot use annotations to define beans provided by other code (e.g. Spring transaction managers and data sources)

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Java-Based Configuration

- String in Spring 3, you can provide configuration metadata by writing Java code instead of XML, giving you fine-grained control over how the container creates objects
- Spring provides the **@Configuration** and **@Bean** annotations as well as the **AnnotationConfigApplicationContext** application context type

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Java-Based Configuration Example

```
1    @Configuration
2    public class HelloWorldConfig
3    {
4       @Bean
5       public HelloWorld helloWorld()
6       {
7          return new HelloWorld();
8       }
9    }
```

```
1    ApplicationContext ctx =
2      new AnnotationConfigApplicationContext(
3         HelloWorldConfig.class);
4
5    HelloWorld helloWorld =
6       ctx.getBean(HelloWorld.class);
7
8    . . .
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- How to inject collection properties
- About autowiring
- How to use annotations in Spring 2.x or later as an alternative way to provide configuration metadata

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring Java Configuration

- Why Use Java Configuration?
- Writing a Configuration Class
- Using Dependency Injection

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring Spring Beans

- Spring provides several techniques so that developers can provide metadata for beans, including:
  - XML
  - Annotations
  - Java Configuration
- XML configuration is the traditional approach

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Java-Based Configuration

- String in Spring 3, you can provide configuration metadata by writing Java code instead of XML, giving you fine-grained control over how the container creates objects
- Spring provides the **@Configuration** and **@Bean** annotations as well as the **AnnotationConfigApplicationContext** application context type

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Java-Based Configuration Example

```
1    @Configuration
2    public class HelloWorldConfig
3    {
4       @Bean
5       public HelloWorld helloWorld()
6       {
7          return new HelloWorld();
8       }
9    }
```

```
1    AnnotationConfigApplicationContext ctx =
2      new AnnotationConfigApplicationContext(
3         HelloWorldConfig.class);
4
5    HelloWorld helloWorld =
6       ctx.getBean(HelloWorld.class);
7
8    . . .
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Why Use Java Configuration?

- Java configuration has some of the same advantages as XML configuration:
    - Centralized configuration
    - Cleaner, pure-POJO beans (no annotations)
- Main advantage of Java configuration over XML is compiler-time checking for typos and type-safety

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The @Bean Annotation

- You use the **@Bean** annotation on methods that produce Spring beans
- The container calls such methods, then adds the returned reference to its internal bean list
- By default, the new bean's name is taken from the bean-creation method's name
- This annotation has optional elements to let you specify the bean's name, scope and initialize and destroy methods

```
1    @Bean
2    public Student myStudent()
3    {
4      Student s = new Student();
5
6      return s;
7    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Performing Setter Injection

- In the bean-creation method, you can easily call **set()** methods to inject dependencies

```
1     @Bean
2     public Advisor myAdvisor()
3     {
4        Advisor a = new Advisor();
5        a.setAdvisorID(12);
6        a.setName("Sue Smith");
7        a.setSalary(40000);
8
9        return a;
10    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Referencing Other Beans

- If the bean-creation methods are in the same configuration class, you can call them directly to obtain references to other beans

```
1    @Configuration
2    public class MyConfig
3    {
4       @Bean
5       public Student myStudent()
6       {
7           Student s = new Student();
8           s.setAdvisor(myAdvisor());
9           return s;
10      }
11
12      @Bean
13      public Advisor myAdvisor() {....}
14   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Performing Constructor Injection

- In the bean-creation method, you can provide dependencies via **constructor injection** if the bean class supports it

```
1     @Bean
2     public Student myStudent()
3     {
4        Student s = new Student(myAdvisor());
5        s.setName("Harry Wolfe");
6        s.setGpa(3.33);
7        s.setStudentID(45);
8
9        return s;
10    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using Autowiring in a Configuration Class

- Instead of directly calling other bean-creation methods, configuration classes can autowire related beans by type or name

```
1    @Configuration
2    public class StudentConfiguration
3    {
4      @Autowired private Advisor a;  // autowire by type
5
6      @Bean
7      public Student myStudent()
8      {
9        Student s = new Student(a);
10
11       return s;
12     }
13   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Autowiring Bean-Creation Methods

- The bean-creation methods themselves can take advantage of dependency injection by defining arguments that Spring will inject

```
1    @Configuration
2    public class StudentConfig
3    {
4       @Bean
5       public Student myStudent(Advisor a)
6       {
7           Student s = new Student(a);
8           return s;
9       }
10   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Multiple @Configuration Classes

- The **@Configuration** annotation identifies a class as a providing beans
- You can write multiple Configuration classes to modularize the bean definitions

```
1    @Configuration
2    public class StudentConfig
3    {
4       @Bean public Student myStudent() {.....}
5    }
```

```
1    @Configuration
2    public class AdvisorConfig
3    {
4       @Bean public Advisor myAdvisor() {.....}
5    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Referencing Multiple Configuration Classes

- Applications can use **AnnotationConfigApplicationContext** or **AnnotationConfigWebApplicationContext** to reference configuration class(es)

```
1    AnnotationConfigApplicationContext ctx =
2     new AnnotationConfigApplicationContext(
3        StudentConfig.class,
4        AdvisorConfig.class);
5
6    Student s = (Student)ctx.getBean(Student.class);
7    Advisor a = (Advisor)ctx.getBean(Advisor.class);
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Importing a Configuration

- You can use the **@Import** annotation to reference other Configurations
- Then, the "main" program only needs to specify the "root" Configuration

```
1    @Configuration
2    @Import(AdvisorConfig.class)
3    public class StudentConfig
4    {
5    . . .
6    }
```

```
1    AnnotationConfigApplicationContext ctx =
2      new AnnotationConfigApplicationContext(
3        StudentConfig.class);
4
5    Student s = (Student)ctx.getBean(Student.class);
6    Advisor a = (Advisor)ctx.getBean(Advisor.class);
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Importing Other Configuration Types

- You can use the **@ImportResource** annotation to reference configurations defined in XML or Groovy
- You can use the **@ComponentScan** annotation to enable the same functionality as in XML's **context:component-scan** element
- These techniques lets you mix configuration approaches and migrate from XML and/or annotation-based configuration to Java configuration

```
1    @Configuration
2    @ImportResource({"classpath:spring.xml"})
3    @ComponentScan(basePackages="mypackage")
4    public class MyConfig
5    {
6    . . .
7    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Complete Example

Advisor.java

Professor.java

Student.java

AdvisorConfig.java

StudentConfig.java

TestUniversity.java

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    package university;
2
3    import java.util.ArrayList;
4    import java.util.Collection;
5
6    public class Advisor
7    {
8        private int advisorID;
9        private String name;
10       private double salary;
11       private Professor theProf;
12
13       private Collection<Student> students = new ArrayList<Student>();
14
15       public Professor getTheProf()
16       {
17           return theProf;
18       }
19
20       public void setTheProf(Professor theProf)
21       {
22           this.theProf = theProf;
23       }
24
25       public int getAdvisorID()
26       {
27           return advisorID;
28       }
29
30       public void setAdvisorID(int advisorID)
31       {
32           this.advisorID = advisorID;
33       }
34
35       public String getName()
36       {
37           return name;
38       }
39
40       public void setName(String name)
41       {
42           this.name = name;
43       }
44
```

```
46      {
47          return salary;
48      }
49
50      public void setSalary(double salary)
51      {
52          this.salary = salary;
53      }
54
55      public Collection<Student> getStudents()
56      {
57          return students;
58      }
59
60      public void setStudents(Collection<Student> students)
61      {
62          this.students = students;
63      }
64
65  }
```

```
1    package university;
2
3    public class Student
4    {
5        private int studentID;
6        private String name;
7        private double gpa;
8
9        private Advisor advisor;
10
11       public Student() {}
12
13       public Student(Advisor advisor)
14       {
15           this.advisor = advisor;
16       }
17
18       public int getStudentID()
19       {
20           return studentID;
21       }
22
23       public void setStudentID(int studentID)
24       {
25           this.studentID = studentID;
26       }
27
28       public String getName()
29       {
30           return name;
31       }
32
33       public void setName(String name)
34       {
35           this.name = name;
36       }
37
38       public double getGpa()
39       {
40           return gpa;
41       }
42
43       public void setGpa(double gpa)
44       {
```

```
46      }
47
48      public Advisor getAdvisor()
49      {
50          return advisor;
51      }
52
53      public void init()
54      {
55          System.out.println("In init method");
56      }
57
58  }
```

```java
package university.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import university.Advisor;
import university.Professor;

@Configuration
public class AdvisorConfig
{
    @Bean
    public Advisor myAdvisor()
    {
        Advisor a = new Advisor();
        a.setAdvisorID(12);
        a.setName("Sue Smith");
        a.setSalary(40000);

        a.setTheProf(theProfessor());

        return a;
    }

    @Bean
    public Professor theProfessor()
    {
        Professor p = new Professor();
        p.setName("I.M. Mean");
        p.setProfessorID(42);

        return p;
    }
}
```

```java
package university.config;

import javax.annotation.Resource;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Import;

import university.Advisor;
import university.Student;

@Configuration
@Import(AdvisorConfig.class)
public class StudentConfig
{
  @Autowired
  private Advisor a;

//    @Resource(name="myAdvisor")
//    private Advisor a;

    @Bean(name="student01")
//  public Student myStudent(Advisor a)
    public Student myStudent()
    {
        Student s = new Student(a);
        s.setName("Harry Wolfe");
        s.setGpa(3.33);
        s.setStudentID(45);

        return s;
    }
}
```

```java
package university.test;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.support.AbstractApplicationContext;

import university.Student;
import university.config.AdvisorConfig;
import university.config.StudentConfig;

public class TestUniversity
{

    public static void main(String[] args)
    {
        AbstractApplicationContext ctx = new AnnotationConfigApplicationContext(
                StudentConfig.class, AdvisorConfig.class);

        // Student s = (Student)ctx.getBean(Student.class);
        Student s = (Student) ctx.getBean("student01");
        System.out.println("Student name: " + s.getName());
        System.out.println("Advisor name: " + s.getAdvisor().getName());
        System.out.println("Professor name: " +
           s.getAdvisor().getTheProf().getName());

        ctx.close();
    }

}
```

```java
package university;

import java.util.ArrayList;

public class Professor
{
    private int professorID;
    private String name;
    private ArrayList<Advisor> advisors;

    public int getProfessorID()
    {
        return professorID;
    }

    public void setProfessorID(int professorID)
    {
        this.professorID = professorID;
    }

    public String getName()
    {
        return name;
    }

    public void setName(String name)
    {
        this.name = name;
    }

    public ArrayList<Advisor> getAdvisors()
    {
        return advisors;
    }

    public void setAdvisors(ArrayList<Advisor> advisors)
    {
        this.advisors = advisors;
    }
}
```

Using the Groovy DSL for Configuration

- Starting in Spring 4, you can use the Groovy language to define Spring beans and/or to provide configuration metadata
- Steps for using Groovy:
    - Add Groovy dependency to Gradle or Maven build script
    - Optionally use Groovy to create Spring bean classes
    - Optionally use Spring's Groovy DSL to configure beans written in Java or Groovy

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is Groovy?

- Groovy is a dynamic, typing optional language with syntax similar to Java that runs on the JVM
- Some differences from Java:
    - Semicolons optional
    - Delimit strings with single or double quotes (double quoted support **interpolation**)
    - Can omit parenthesis for method calls
    - **print** and **println** shortcuts for System.out.println



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Groovy DSL Example

```
1    // build.gradle
2    dependencies {
3     compile 'org.springframework:spring-context:4.2.6.RELEASE'
4     compile "org.codehaus.groovy:groovy-all:2.1.0"
5    }
```

```
1    // spring.groovy
2    import university.TeachingAssistant
3
4    beans {
5      ta23 (TeachingAssistant) {  // bean-name (bean-type)
6        taID = 465                // set property
7        name = "Sue Mi"           // set property
8      }
9    }
```

```
1    // StudentConfig.java
2    @ImportResource({"classpath:spring.groovy"})
3    public class StudentConfig {. . .}
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- About the advantanges of Spring Java Configuration
- How to use Java Configuration to perform dependency injection

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring Beans

- Scopes
- Lifecycle

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring Spring Beans

- Spring provides several techniques so that developers can provide metadata for beans, including:
  - XML
  - Annotations
  - Spring 3 Java Configuration
- XML configuration is the traditional approach

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Bean Scopes

- In Spring, the notion of scope defines how many objects a Spring container instantiates for a given configured bean
- Basic Spring supports **singleton** and **prototype**
- ApplicationContexts used in Web applications also support **request**, **session** and **global session** scopes

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Singleton Scope

- By default, Spring treats beans as **singletons**, which means that a container instantiates only one instance of a defined bean

```
1    @Component("student5")
2    public class Student
3    {
4       @Resource(name="advisor1")
5       private Advisor theAdvisor;
6    }
7    @Component("advisor1")
8    public class Advisor{...}
```

```
1    Advisor advisor1 = (Advisor)ctx.getBean("advisor1");
2    Student student5 = (Student)ctx.getBean("student5");
3    // prints "Singleton!"
4    if (advisor1 == student5.getTheAdvisor())
5      System.out.println("Singleton!");
6    else
7      System.out.println("Prototype!");
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Prototype Scope

- If you configure a bean as a **prototype**, then a container instantiates a new object whenever a request, e.g. injection or getBean(), is made

```
1    @Component("student5")
2    public class Student
3    {
4       @Resource(name="advisor1")
5       private Advisor theAdvisor;
6    }
7    @Component("advisor1") @Scope("prototype")
8    public class Advisor{...}
```

```
1    Advisor advisor1 = (Advisor)ctx.getBean("advisor1");
2    Student student5 = (Student)ctx.getBean("student5");
3    // prints "Prototype!"
4    if (advisor1 == student5.getTheAdvisor())
5      System.out.println("Singleton!");
6    else
7      System.out.println("Prototype!");
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Singleton vs Prototype Java Config

- By default, beans created during Java configuration are singletons, but you can specify prototype scope on the bean-creation method using **@Scope**

```
1     @Bean
2     @Scope("prototype")
3     public Advisor createAdvisor()
4     {
5       Advisor a = new Advisor();
6       a.setAdvisorID(12);
7       a.setName("Sue Smith");
8       a.setSalary(40000);
9
10      return a;
11    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Bean Lifecycle

- Similar to EJBs, Spring beans have a well-defined creation and destruction lifecycle
- If you wish, you can hook into the lifecycle so that the container calls methods as the bean moves through its lifecycle
- Spring provides serveral ways to configure notification of lifecycle events:
    - Annotation configuration
    - Java configuration
    - Implementing optional Spring interfaces
    - XML configuration

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Lifecycle Annotations

- **@PostConstruct** and **PreDestroy** are standard JEE annotations that Spring uses to let you hook lifecycle events
- These annotations are processed by the CommonAnnotationBeanPostProcessor

```
1    @Component
2    public class Student
3    {
4      @PostConstruct
5      public void setup() {....}
6
7      @PreDestroy
8      public void cleanup() {....}
9    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

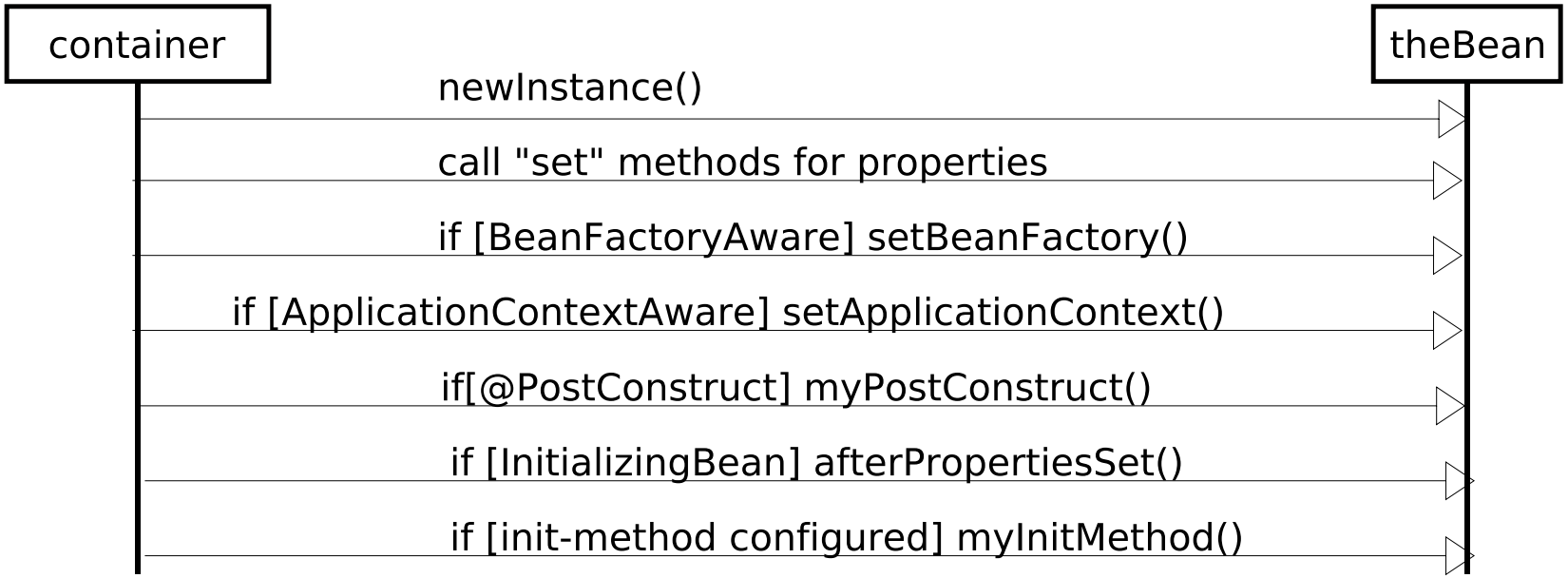
Lifecycle With Java Config

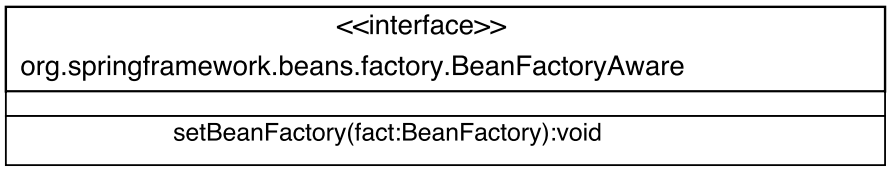
- Optional elements on @Bean let you specify **init** and **destroy** methods

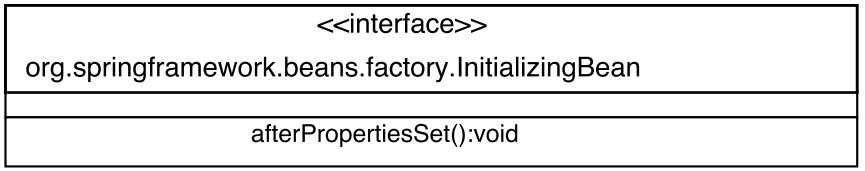
```
1    @Bean(initMethod="init", destroyMethod="cleanup")
2    public Student createStudent()
3    {
4      Student s = new Student();
5
6      return s;
7    }
```

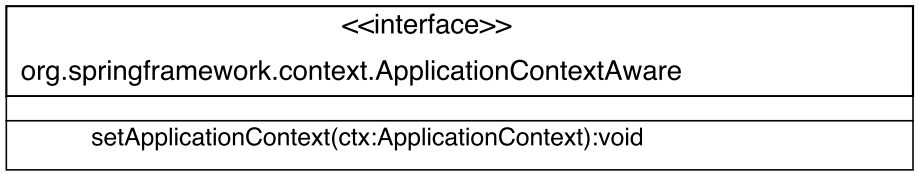

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Creation Lifecycle









Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Creation Lifeycle Example

MyCreationBean.java

TestCreation.java

create.xml

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1   package mypackage;
2
3   import org.springframework.beans.BeansException;
4   import org.springframework.beans.factory.BeanFactory;
5   import org.springframework.beans.factory.BeanFactoryAware;
6   import org.springframework.beans.factory.InitializingBean;
7   import org.springframework.context.ApplicationContext;
8   import org.springframework.context.ApplicationContextAware;
9
10  public class MyCreationBean implements BeanFactoryAware,
11          ApplicationContextAware, InitializingBean
12  {
13      private int prop1;
14      private String prop2;
15
16      public int getProp1()
17      {
18          return prop1;
19      }
20
21      public void setProp1(int prop1)
22      {
23          this.prop1 = prop1;
24      }
25
26      public String getProp2()
27      {
28          return prop2;
29      }
30
31      public void setProp2(String prop2)
32      {
33          this.prop2 = prop2;
34      }
35
36      @Override
37      public void setBeanFactory(BeanFactory factory) throws BeansException
38      {
39          System.out.println("In setBeanFactory");
40      }
41
42      @Override
43      public void setApplicationContext(ApplicationContext ctx)
44              throws BeansException
```

5 - 11A : MyCreationBean.java

```
46          System.out.println("In setApplicationContext");
47      }
48
49      @Override
50      public void afterPropertiesSet() throws Exception
51      {
52          System.out.println("In afterPropertiesSet, prop1=" + prop1);
53      }
54
55      public void myInitMethod()
56      {
57          System.out.println("In myInitMethod");
58      }
59
60      @PostConstruct
61      public void myPostConstructMethod()
62      {
63          System.out.println("In myPostConstructMethod");
64      }
65  }
```

```
1    package main;
2
3    import mypackage.MyCreationBean;
4
5    import org.springframework.context.ApplicationContext;
6    import org.springframework.context.support.FileSystemXmlApplicationContext;
7
8    public class TestCreation
9    {
10       public static void main(String[] args)
11       {
12           ApplicationContext ctx =
13               new FileSystemXmlApplicationContext("create.xml");
14
15           MyCreationBean mcb = (MyCreationBean)ctx.getBean("myCreateBean");
16       }
17
18   }
```

```
<?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:aop="http://www.springframework.org/schema/aop"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:jee="http://www.springframework.org/schema/jee"
    xmlns:jms="http://www.springframework.org/schema/jms"
    xmlns:lang="http://www.springframework.org/schema/lang"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-3.1.xsd
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.1.xsd
        http://www.springframework.org/schema/jee
        http://www.springframework.org/schema/jee/spring-jee-3.1.xsd
        http://www.springframework.org/schema/lang
        http://www.springframework.org/schema/lang/spring-lang-3.1.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-3.1.xsd
        http://www.springframework.org/schema/util
        http://www.springframework.org/schema/util/spring-util-3.1.xsd">

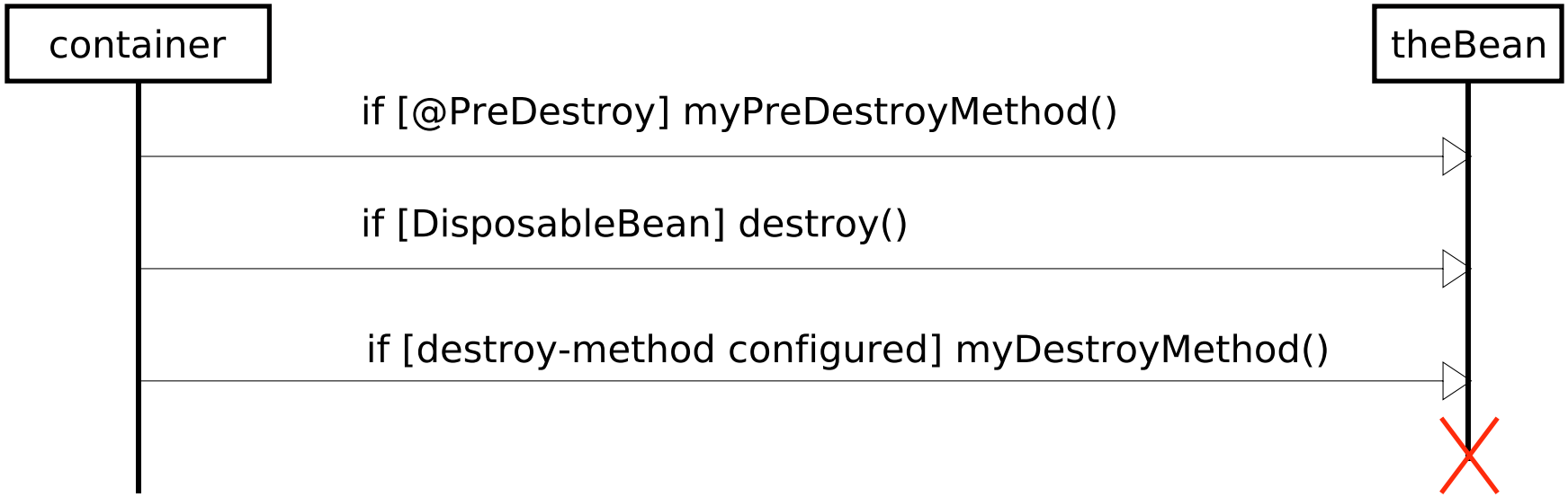
    <context:annotation-config/>
    <context:component-scan base-package="mypackage"/>

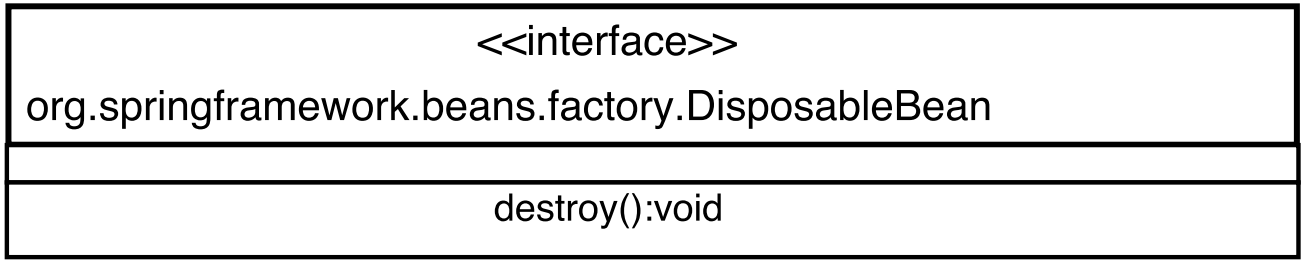
    <bean id="myCreateBean" class="mypackage.MyCreationBean"
       init-method="myInitMethod">
     <property name="prop1" value="12"/>
     <property name="prop2" value="abcd"/>
    </bean>

</beans>
```

Destruction Lifecycle

- This only applies to singleton-scoped beans since the container "forgets" about prototypes after creation





Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Destruction Lifecycle Example

MyDestroyBean.java

TestDestroy.java

destroy.xml

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```java
package mypackage;

import org.springframework.beans.factory.DisposableBean;

public class MyDestroyBean implements DisposableBean
{

    @Override
    public void destroy() throws Exception
    {
        System.out.println("In destroy");
    }

    public void myDestroyMethod()
    {
        System.out.println("In myDestroyMethod");
    }

    @PreDestroy
    public void myPreDestroyMethod()
    {
        System.out.println("In myPreDestroyMethod");
    }

}
```

5 - 13A : MyDestroyBean.java

```
1    package main;
2
3    import mypackage.MyDestroyBean;
4
5    import org.springframework.context.support.AbstractApplicationContext;
6    import org.springframework.context.support.FileSystemXmlApplicationContext;
7
8    public class TestDestroy
9    {
10       public static void main(String[] args)
11       {
12           AbstractApplicationContext ctx =
13               new FileSystemXmlApplicationContext("destroy.xml");
14
15           MyDestroyBean mdb = (MyDestroyBean)ctx.getBean("myDestroyBean");
16           ctx.close();
17       }
18
19   }
```

```
<?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:aop="http://www.springframework.org/schema/aop"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:jee="http://www.springframework.org/schema/jee"
    xmlns:jms="http://www.springframework.org/schema/jms"
    xmlns:lang="http://www.springframework.org/schema/lang"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-3.1.xsd
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.1.xsd
        http://www.springframework.org/schema/jee
        http://www.springframework.org/schema/jee/spring-jee-3.1.xsd
        http://www.springframework.org/schema/lang
        http://www.springframework.org/schema/lang/spring-lang-3.1.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-3.1.xsd
        http://www.springframework.org/schema/util
        http://www.springframework.org/schema/util/spring-util-3.1.xsd">

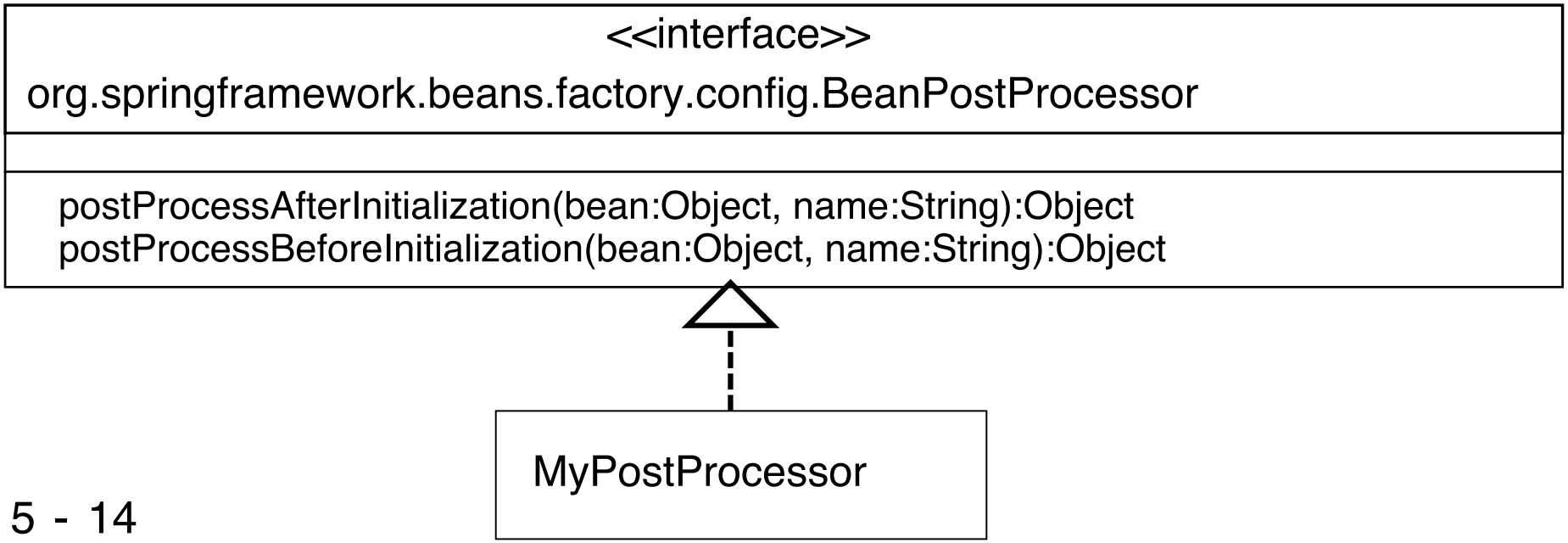
    <context:annotation-config/>
    <context:component-scan base-package="mypackage"/>

    <bean id="myDestroyBean" class="mypackage.MyDestroyBean"
       destroy-method="myDestroyMethod">
    </bean>

</beans>
```

BeanPostProcessors

- A **BeanPostProcessor** can hook certain lifecycle events on **all** of the beans in a container
- You must register a BeanPostProcessor - in an application context you can register via annotations or XML, but in a bean factory, you must register programmatically



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Coding A BeanPostProcessor

- Since a BeanPostProcessor "sees" ALL beans in the container, if it only wants to operate on certain beans, it must be able to distinguish them
- Distinguishing strategies include using a **marker** interface or by using bean naming conventions
- The post-processor methods can return either the input object or substitute another object with the same interface

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

BeanPostProcessor Example

Encrypted.java

EncryptedBean.java

MyBeanPostProcessor.java

TestBeanPostProcessor.java

beanpp.xml

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    package mypackage;
2
3    public interface Encrypted
4    {
5        public String getText();
6        public void setText(String text);
7    }
```

```java
package mypackage;

import org.springframework.stereotype.Component;

@Component
public class EncryptedBean implements Encrypted
{
    private String text = "Uryyb, jbeyq";

    public String getText()
    {
        return text;
    }

    public void setText(String text)
    {
        this.text = text;
    }
}
```

```java
package mypackage;

import org.springframework.beans.BeansException;
import org.springframework.beans.factory.config.BeanPostProcessor;
import org.springframework.stereotype.Component;

@Component
public class MyBeanPostProcessor implements BeanPostProcessor
{

    @Override
    public Object postProcessAfterInitialization(Object bean, String name)
            throws BeansException
    {
        System.out.println("In postProcessAfterInitialization");
        if (bean instanceof Encrypted)
        {
            Encrypted eb = (Encrypted)bean;
            String s1 = eb.getText();
            String s2 = decrypt(s1);
            eb.setText(s2);
            System.out.println("Decrypted '" + s1 + "' to '" + s2 + "'");
        }

        return bean;
    }

    @Override
    public Object postProcessBeforeInitialization(Object bean, String name)
            throws BeansException
    {
        return bean;
    }

    private String decrypt(String s)
    {
        StringBuilder out = new StringBuilder(s.length());
        char[] ca = s.toCharArray();
        for (char c : ca)
        {
            if (c >= 'a' && c <= 'm')
                c += 13;
            else if (c >= 'n' && c <= 'z')
                c -= 13;
```

```
46                  c += 13;
47              else if (c >= 'N' && c <= 'Z')
48                  c -= 13;
49              out.append(c);
50          }
51          return out.toString();
52      }
53
54      public static void main(String[] args)
55      {
56          MyBeanPostProcessor mbpp = new MyBeanPostProcessor();
57          String s1 = mbpp.decrypt("Hello, world");
58          String s2 = mbpp.decrypt(s1);
59          System.out.println(s1);
60          System.out.println(s2);
61      }
62  }
```

```
1    package main;
2
3    import mypackage.EncryptedBean;
4
5    import org.springframework.context.support.AbstractApplicationContext;
6    import org.springframework.context.support.FileSystemXmlApplicationContext;
7
8    public class TestBeanPostProcessor
9    {
10
11       public static void main(String[] args)
12       {
13           AbstractApplicationContext ctx =
14               new FileSystemXmlApplicationContext("beanpp.xml");
15
16           EncryptedBean eb = (EncryptedBean)ctx.getBean("encryptedBean");
17           System.out.println("Main: " + eb.getText());
18       }
19
20   }
```

```xml
<?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:aop="http://www.springframework.org/schema/aop"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:jee="http://www.springframework.org/schema/jee"
    xmlns:jms="http://www.springframework.org/schema/jms"
    xmlns:lang="http://www.springframework.org/schema/lang"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-3.1.xsd
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.1.xsd
        http://www.springframework.org/schema/jee
        http://www.springframework.org/schema/jee/spring-jee-3.1.xsd
        http://www.springframework.org/schema/lang
        http://www.springframework.org/schema/lang/spring-lang-3.1.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-3.1.xsd
        http://www.springframework.org/schema/util
        http://www.springframework.org/schema/util/spring-util-3.1.xsd">

    <context:annotation-config/>
    <context:component-scan base-package="mypackage"/>

</beans>
```

Factory Methods

- Some Java classes define a **static** method that creates and initializes instances
- If you wish to use such a class as a Spring bean configured in XML, then you must tell the container to use the static method

```
1    <bean id="bean1" factory-method="getInstance"
2      class="mypackage.MyClassWithFactoryMethod" />
```

```
1    public static MyClassWithFactoryMethod getInstance()
2    {
3     . . .
4    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Factory Classes

- Some Java classes use a separate **factory class** to create instances
- The factory itself defines **non-static** methods that create and return object references
- If you wish to use such a factory class to create Spring beans configured in XML, then you must configure the factory class and its factory method

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Factory Classes, cont'd

```
1    public class StudentFactory
2    {
3      public Student getStudent()
4      {
5          return new Student(12, 3.44);
6      }
7    }
```

```
1    <bean id="myFactory"
2      class="mypackage.StudentFactory"/>
3
4    <bean id="student1" factory-bean="myFactory"
5        factory-method="getStudent" />
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- About names and IDs for beans
- How bean scopes work
- About the lifecycle of beans

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

XML Dependency Injection

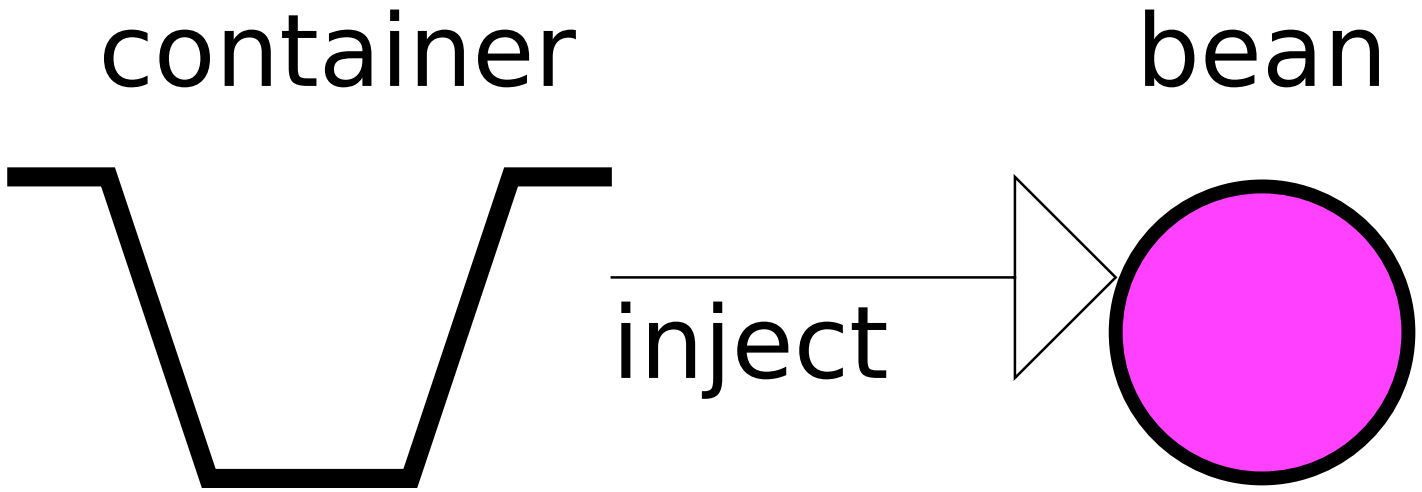
- Setter Injection
- Constructor Injection

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Dependency Injection

- **Dependency injection** is a technique in which the container instantiates and provides references for beans
- To inject a reference, the container either calls a bean's "set" method or constructor
- DI lessens coupling between beans and decouples beans from services such as JNDI



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring Dependency Injection

- Spring provides several techniques so that developers can configure DI, including:
    - XML
    - Annotations
    - Spring 3 Java Configuration
- This chapter covers XML DI configuration

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Inversion of Control

- IoC is an architectural pattern in which the flow of control of a system is inverted in comparison to procedural programming
- Sometimes referred to as "The Hollywood Principle"
- Dependency injection is a form of IoC in which a container provides resources instead of the traditional practice of performing lookups

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Sample Application for this Chapter

Student.java

Advisor.java

TestDI.java

spring.xml

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    package university;
2
3    public class Student
4    {
5        private int id;
6        private String name;
7        private double gpa;
8        private Advisor theAdvisor;
9
10       public Student(String name, Advisor advisor)
11       {
12           this.name = name;
13           this.theAdvisor = advisor;
14       }
15
16       public Student(int id, double gpa)
17       {
18           this.id = id;
19           this.gpa = gpa;
20       }
21
22       public Student() {}
23
24       public Advisor getTheAdvisor()
25       {
26           return theAdvisor;
27       }
28
29       public void setTheAdvisor(Advisor theAdvisor)
30       {
31           this.theAdvisor = theAdvisor;
32       }
33
34       public int getId()
35       {
36           return id;
37       }
38
39       public void setId(int id)
40       {
41           this.id = id;
42       }
43
44       public String getName()
```

```
46          return name;
47      }
48
49      public void setName(String name)
50      {
51          this.name = name;
52      }
53
54      public double getGpa()
55      {
56          return gpa;
57      }
58
59      public void setGpa(double gpa)
60      {
61          this.gpa = gpa;
62      }
63
64      @Override
65      public String toString()
66      {
67          return "ID: " + id + " name: " + name + " GPA: " + gpa;
68      }
69  }
```

```java
package university;

public class Advisor
{
    private String name;
    private double salary;

    public String getName()
    {
        return name;
    }

    public void setName(String name)
    {
        this.name = name;
    }

    public double getSalary()
    {
        return salary;
    }

    public void setSalary(double salary)
    {
        this.salary = salary;
    }

    @Override
    public String toString()
    {
        return "Name: " + name + " salary: " + salary;
    }
}
```

```
1    package main;
2
3    import org.springframework.context.ApplicationContext;
4    import org.springframework.context.support.FileSystemXmlApplicationContext;
5
6    import university.Student;
7
8    public class TestDI
9    {
10
11       public static void main(String[] args)
12       {
13           ApplicationContext ctx =
14               new FileSystemXmlApplicationContext("spring.xml");
15
16           Student s0 = (Student)ctx.getBean("student0");
17           System.out.println(s0);
18
19           Student s1 = (Student)ctx.getBean("student1");
20           System.out.println(s1);
21
22           Student s2 = (Student)ctx.getBean("student2");
23           System.out.println(s2);
24
25           Student s3 = (Student)ctx.getBean("student3");
26           System.out.println(s3);
27
28           Student s4 = (Student)ctx.getBean("student4");
29           System.out.println(s4);
30
31           Student s5 = (Student)ctx.getBean("student5");
32           System.out.println(s5);
33       }
34   }
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<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">

   <bean id="advisor1" class="university.Advisor"/>

   <bean id="student0" class="university.Student">
     <property name="name" value="Harry"/>
     <property name="id" value="77"/>
     <property name="gpa" value="3.14"/>
     <property name="theAdvisor" ref="advisor1"/>
   </bean>

   <bean id="student1" class="university.Student">
     <constructor-arg value="Joe"/>
     <constructor-arg ref="advisor1"/>
   </bean>

   <bean id="student2" class="university.Student">
     <constructor-arg ref="advisor1"/>
     <constructor-arg value="Sue"/>
   </bean>

   <bean id="student3" class="university.Student">
     <constructor-arg value="12"/>
     <constructor-arg value="3.33"/>
   </bean>

   <bean id="student4" class="university.Student">
     <constructor-arg value="3.33" index="1"/>
     <constructor-arg value="12" index="0"/>
   </bean>

   <bean id="student5" class="university.Student">
     <constructor-arg value="3.33" index="1"/>
     <constructor-arg value="12" index="0"/>
     <property name="name" value="George"/>
     <property name="theAdvisor" ref="advisor1">
     </property>
   </bean>
</beans>
```

Setter Injection

- The JavaBean specification defines a **property** as data exposed via get/set methods
- Spring can use the "set" method on JavaBean-style properties to inject simple values and bean references



Student bean

```
public class Student
{
   public void setId(int id) {...}
   public void setName(String name) {...}
   public void setTheAdvisor(Advisor advisor) {...}
   public void setGpa(double gpa) {...}
  . . .
}
```

spring.xml

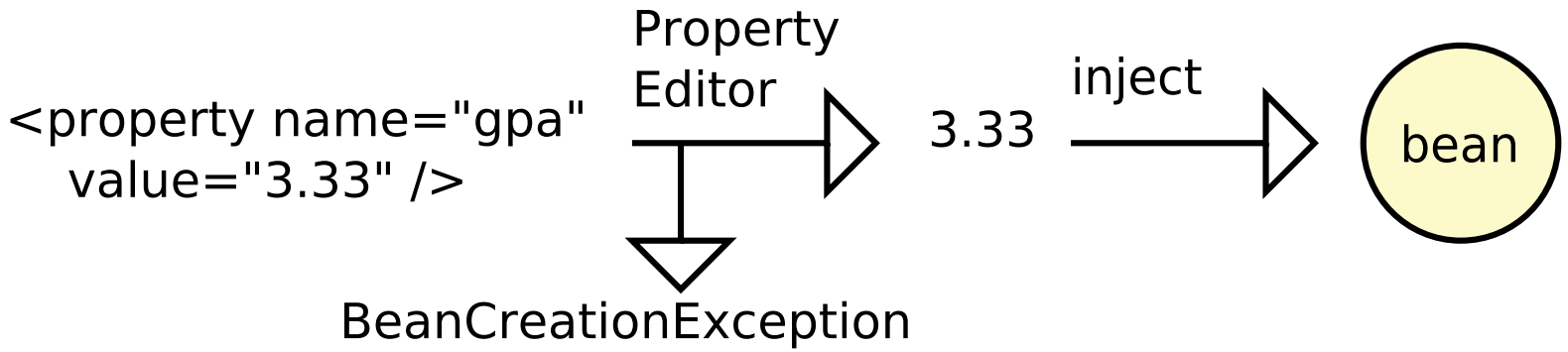
```
<bean id="advisor1" class="university.Advisor"/>

<bean id="student0" class="university.Student">
  <property name="name" value="Harry"/>
  <property name="id" value="77"/>
  <property name="gpa" value="3.14"/>
  <property name="theAdvisor" ref="advisor1"/>
</bean>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Property Conversions

- In the XML configuration, values are defined as character strings
- Spring converts the character strings as required to match the property type using standard JavaBean **property editors**



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Constructor Injection

- Instead of, or in addition to using **setter** injection, Spring supports **constructor** injection that provides properties to a bean during instantiation
- The bean must define an appropriate constructor

Student bean

```
public Student(int id, double gpa){...}
```

spring.xml

```
<bean id="student3" class="university.Student">
  <constructor-arg value="12"/>
  <constructor-arg value="3.33"/>
</bean>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Constructor Injection Resolution

- When you specify constructor arguments, Spring examines the bean for a matching constructor
- Spring attempts to use the types used in the XML file, but note that **values** don't have a type (they are character strings in XML)

Student bean

```
public Student(String name, Advisor advisor) {...}
```

spring.xml

```
<bean id="student1" class="university.Student">
  <constructor-arg value="Joe"/>
  <constructor-arg ref="advisor1"/>
</bean>

<bean id="student2" class="university.Student">
  <constructor-arg ref="advisor1"/>
  <constructor-arg value="Sue"/>
</bean>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Constructor Injection Resolution, cont'd

- To aid resolution, you can use the optional **type** and/or **index** attributes in the XML
- Note that if you specify **constructor-args** in the same order as they appear in the bean's constructor, this is often not necessary

Student bean

```
public Student(int id, double gpa){...}
```

spring.xml

```
<bean id="student3" class="university.Student">
  <constructor-arg value="12"/>
  <constructor-arg value="3.33"/>
</bean>

<bean id="student4" class="university.Student">
  <constructor-arg value="3.33" index="1"/>
  <constructor-arg value="12" index="0"/>
</bean>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Mixing Injection Types

- It's OK to use both setter and constructor injection on a bean
- Spring will first create the bean using the constructor arguments, then invoke the set method(s)

Student bean

```
public Student(int id, double gpa) {...}
public void setName(String name) {...}
public void setTheAdvisor(Advisor advisor) {...}
```

spring.xml

```
<bean id="student5" class="university.Student">
  <constructor-arg value="3.33" index="1"/>
  <constructor-arg value="12" index="0"/>
  <property name="name" value="George"/>
  <property name="theAdvisor" ref="advisor1">
  </property>
</bean>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Setter vs Constructor Injection

- The Spring documentation favors setter injection over constructor injection since it's simpler and more intuitive
- Constructor injection does allow you to guarantee intialization (especially if you ONLY allow constructor injection)
- In Spring 2.5 and later, you can use the **@Required** annotation on a "set" method to guarantee initialization





Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- How to inject values and references via **setter** injection
- How to inject values and references via **constructor** injection

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to JEE

- What is JEE?
- JEE Development Process
- Overview of JEE Technologies

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is JEE?

- Java Enterprise Edition is a set of specifications for server-side Java programming
- Vendors implement JEE by creating **containers** within an application server
- From the Oracle Web site:

> Java Platform, Enterprise Edition (Java EE) builds on the solid foundation of Java Platform, Standard Edition (Java SE) and is the industry standard for implementing enterprise-class service-oriented architecture (SOA) and next-generation web applications.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JEE Version History

- J2EE 1.2 (December 12, 1999)
- J2EE 1.3 (September 24, 2001)
- J2EE 1.4 (November 11, 2003)
- Java EE 5 (May 11, 2006)
- Java EE 6 (December 10, 2009)
- Java EE 7 (May 28, 2013)

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Benefits of JEE

- Portability across platforms
- Portability across vendors
- Containers provide high-level services
- Many implementations from which to choose
- Wide array of tools for development, testing and deployment

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JEE Vendors

A partial list of JEE vendors/products:

- IBM WebSphere
- Oracle/BEA WebLogic
- Oracle GlassFish
- JBoss/Wildfly
- Apache Tomcat (Web container only)
- Apache Geronimo

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JEE Development Tools

A partial list of JEE development tools:

- IBM Rational Application Developer (RAD)
- Eclipse
- Oracle Netbeans
- Oracle JDeveloper
- JetBrains IntelliJ

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JEE Technologies

A partial list of the Application Programming Interfaces (APIs) defined by JEE

- Servlets
- JavaServer Pages
- JavaServer Faces
- Enterprise JavaBeans
- Java Persistence API
- Java Naming and Directory Interface
- JDBC
- Java Messaging Service
- Java SOAP Web Services
- Java RESTful Services

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Deployment Descriptors

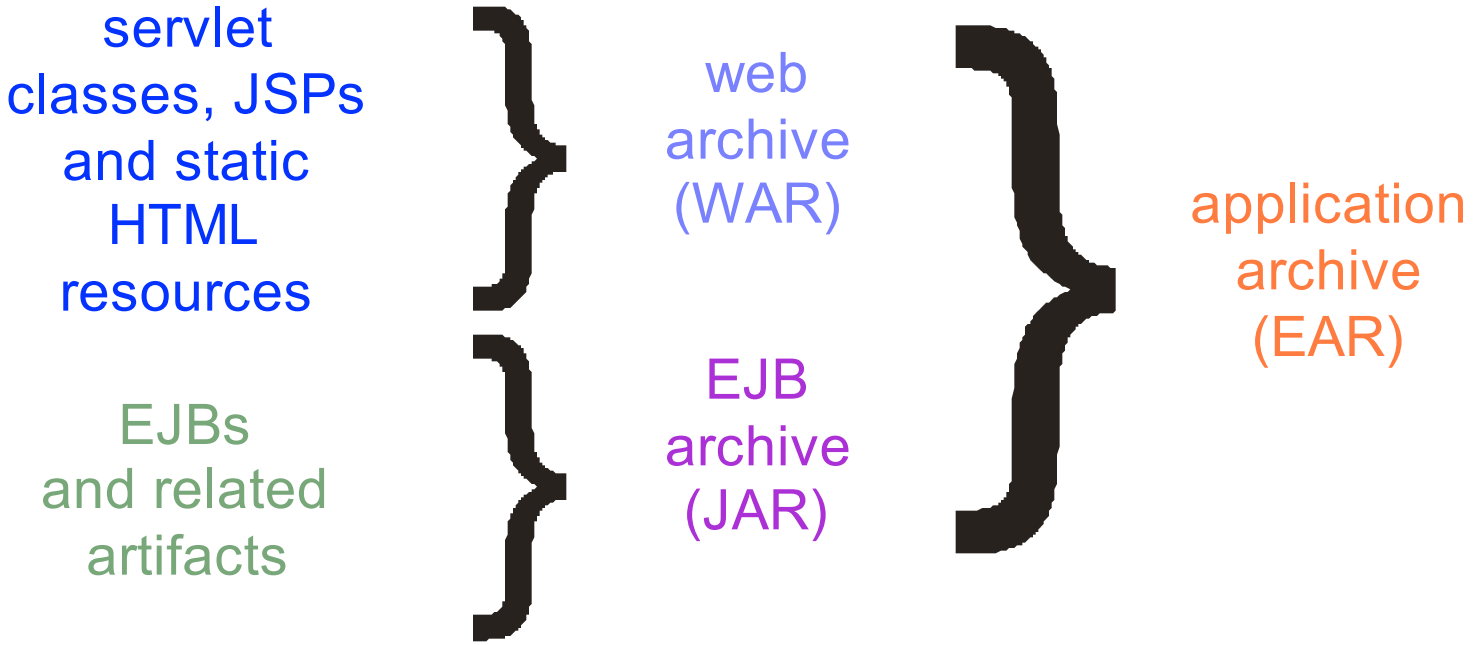
- Many JEE technologies use XML-based **deployment descriptors** for configuration
- Many container vendors provide high-level tools to make it easier to work with these deployment descriptors
- One focus of JEE Version 5 is to reduce the requirement to write deployment descriptors

```
<?xml version="1.0"?>
<web-app>
  <servlet>
    <servlet-name>HelloCount</servlet-name>
    <servlet-class>HelloCount</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>HelloCount</servlet-name>
    <url-pattern>HelloCount</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>InputCount.html</welcome-file>
  </welcome-file-list>
</web-app>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JEE Development and Packaging

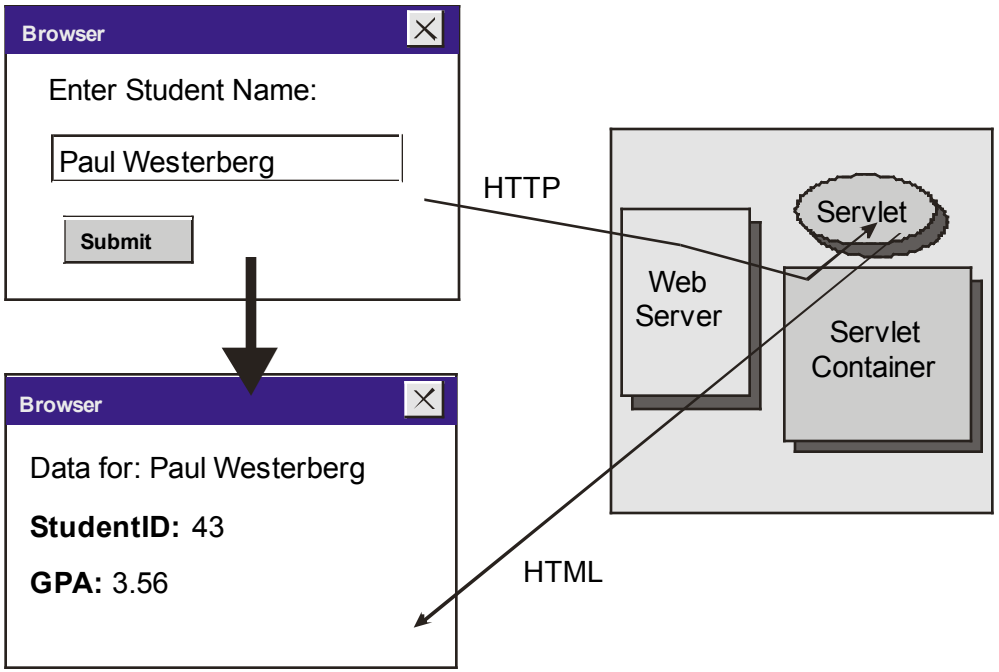
- Since JEE is a Java technology, programmers write JEE Java code and compile it in the normal fashion
- JEE provides various ways to package complete applications to make them easier to deploy



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Servlets

- Servlets are a Java replacement for traditional CGI programs and let you add programmability to your Web sites
- Servlets perform well and are portable across container vendors and platforms
- A common use of servlets is to respond to HTML forms



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to JSPs

- Like servlets, JSPs let you add programmability to Web sites, but JSPs are more focused on presentation rather than Java coding
- Upon first access, the container converts JSPs into servlets, so they have the same performance and portability characteristics

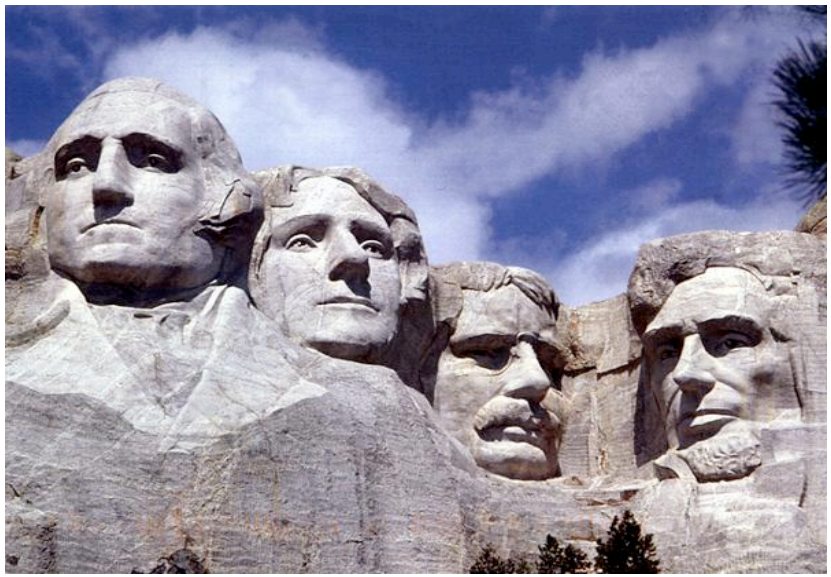
```
1    <%@ taglib uri="http://java.sun.com/jsp/jstl/core"
2       prefix="c"%>
3    <html>
4      <body>
5        <c:forEach begin="0" end="5">
6          <p>Hello, World</p>
7        </c:forEach>
8      </body>
9    </html>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to JSF

- JavaServer Faces is a **component-based** framework for developing Java-based Web applications
- JSF supports a "rapid application development" programming style, especially coupled with a visual development tool
- JSF is a standard part of Java Enterprise Edition (JEE)



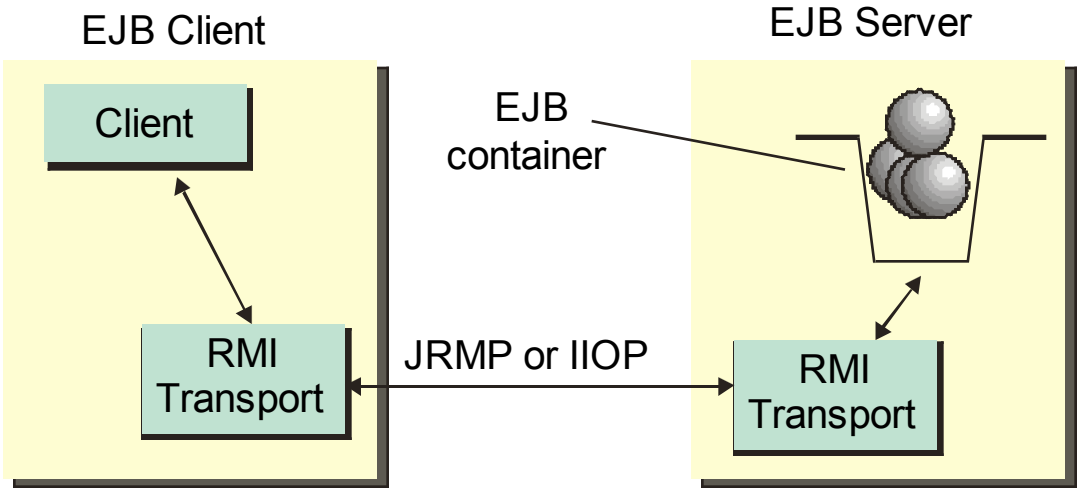


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to EJBs

- Enterprise Java Beans are distributed components that run within the bounds of an EJB container
- The container provides services such as pooling, transactions, security and persistence
- From the EJB specification:

Enterprise JavaBeans is an architecture for component-based computing. Enterprise beans are components of transaction-oriented enterprise applications.



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to CDI

- The **Contexts and Dependency Injection** specification provides a programming model for **domain** objects in Java Enterprise Edition
- CDI is a standard part of JEE 6 and later and provides code **annotations** so that programs can request references to objects that provide a useful service
- CDI was inspired by the Spring Framework

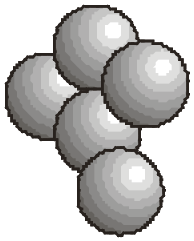
```
public class Test
{
   @Inject
   private MyService service;
    . . .
}
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introducing the Java Persistence API

- **Java Persistence** is the official Oracle API for object/relational mapping
- JPA is a required part of Java Enterprise Edition, Version 5

Object Space





Relational Space

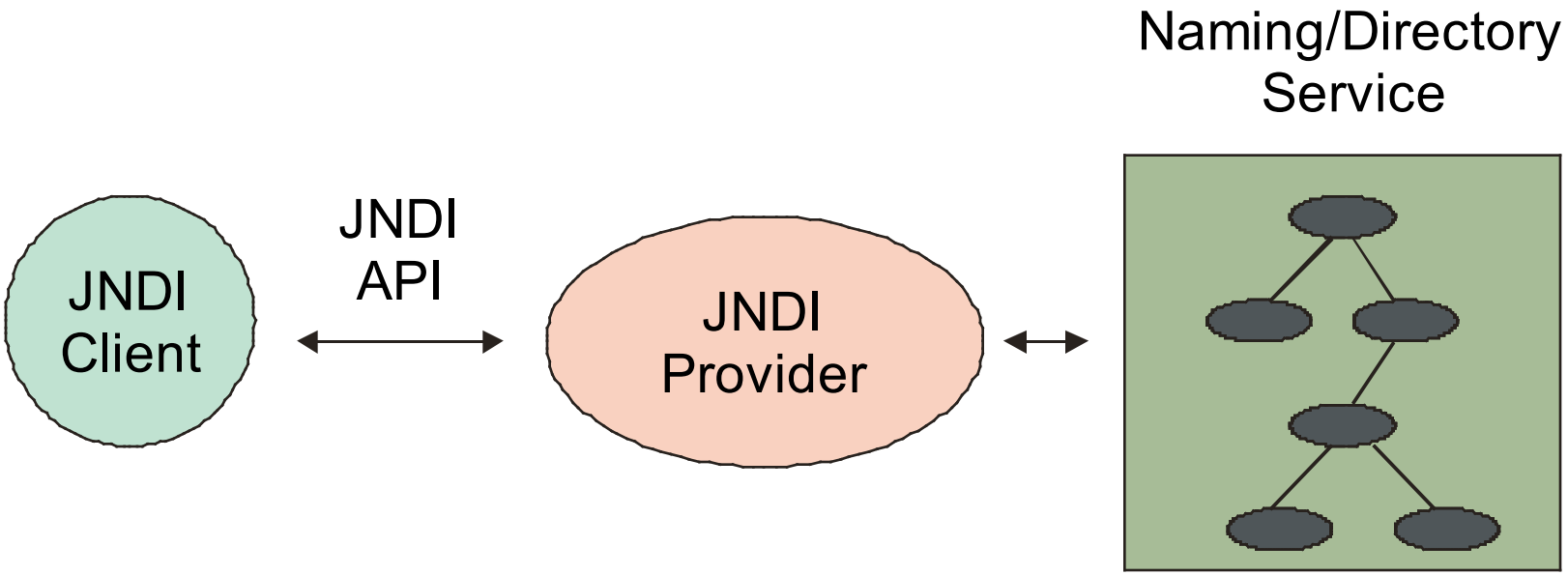
STUDENT

STUDENTID	NAME	DATE
5	Jess Settles	1998-12-02
6	George Peeples	1966-05-19
7	Ben McGilmer	1965-03-13

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to JNDI

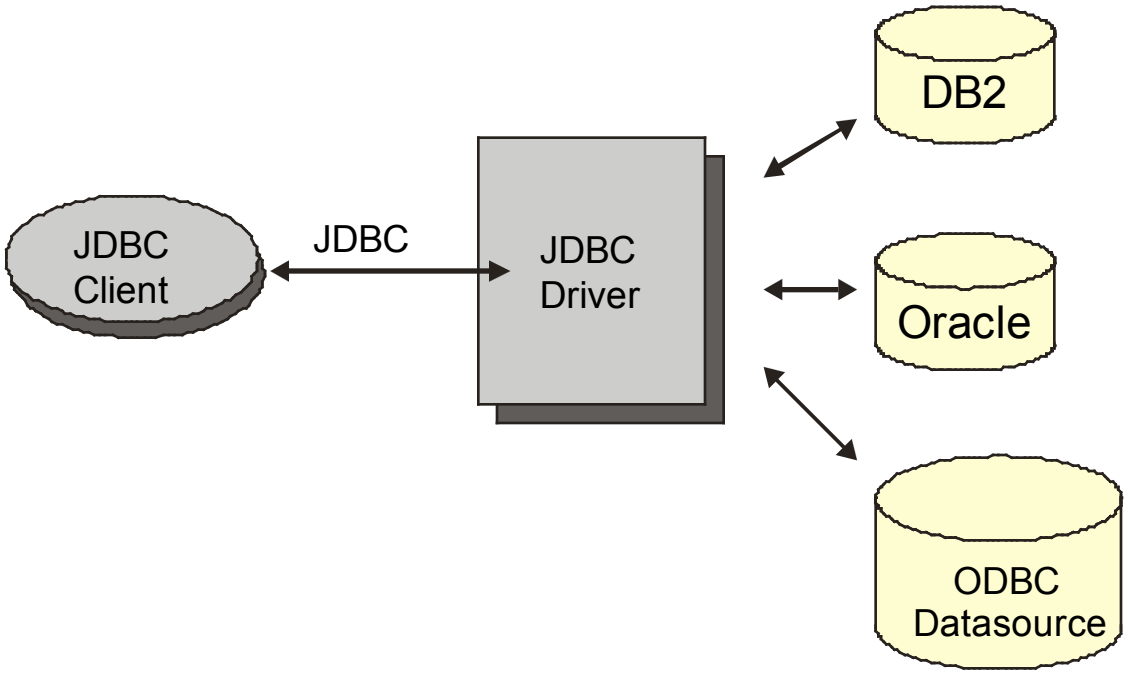
- The Java Naming and Directory Interface provides a standard way to access naming services such as LDAP
- All JEE compliant containers must provide a JNDI implementation since other JEE technologies such as EJBs use JNDI



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to JDBC

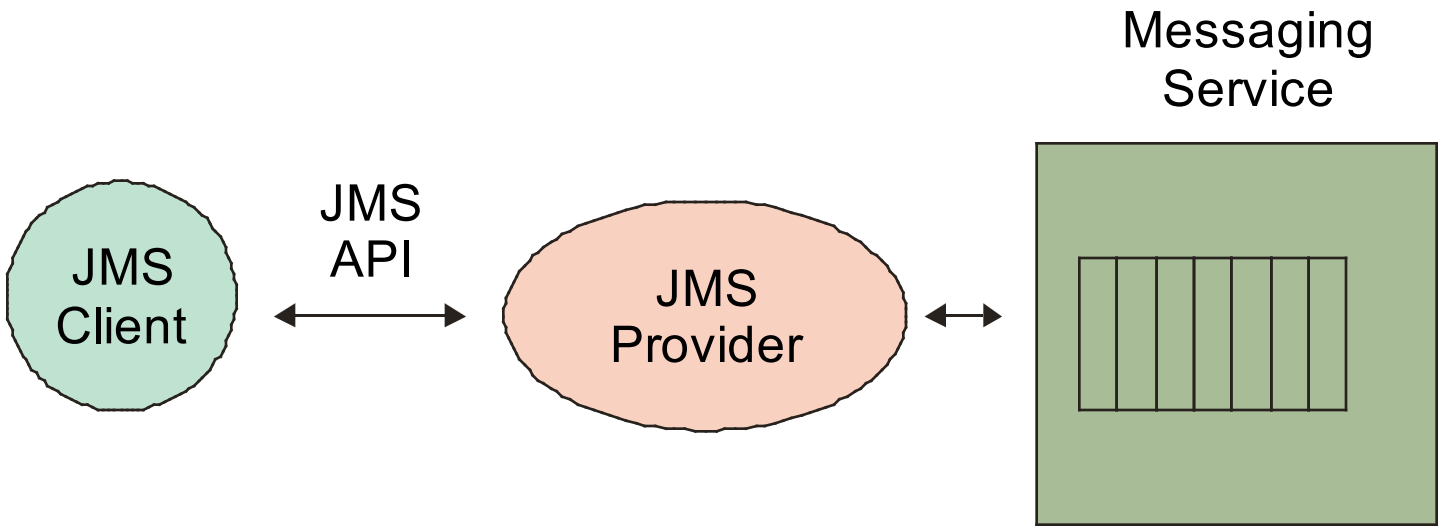
- The JDBC API provides a standard way to access relational data so that your Java programs can remain independent of the actual database product
- To use JDBC, you must obtain a JDBC driver for your particular database



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to JMS

- The Java Messaging Service provides a standard way to access message-oriented middleware (MOM) products such as IBM's WebSphere MQ in a product-independent fashion
- To use JMS, you must obtain a JMS provider for your particular MOM product



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

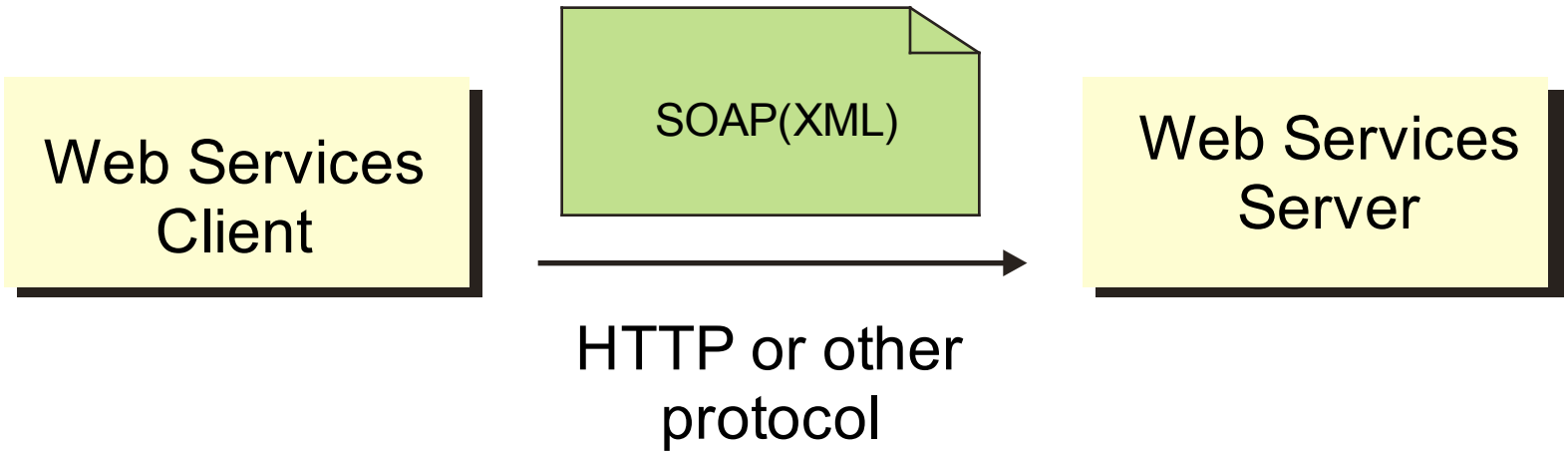
Introduction to JTA

- The Java Transaction API provides a standard interface to transaction managers such as those included with database products like IBM's DB2 or Oracle
- Most JEE programmers do not use JTA directly, but instead use **declarative** transactions and let the container handle transactions

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to SOAP Web Services

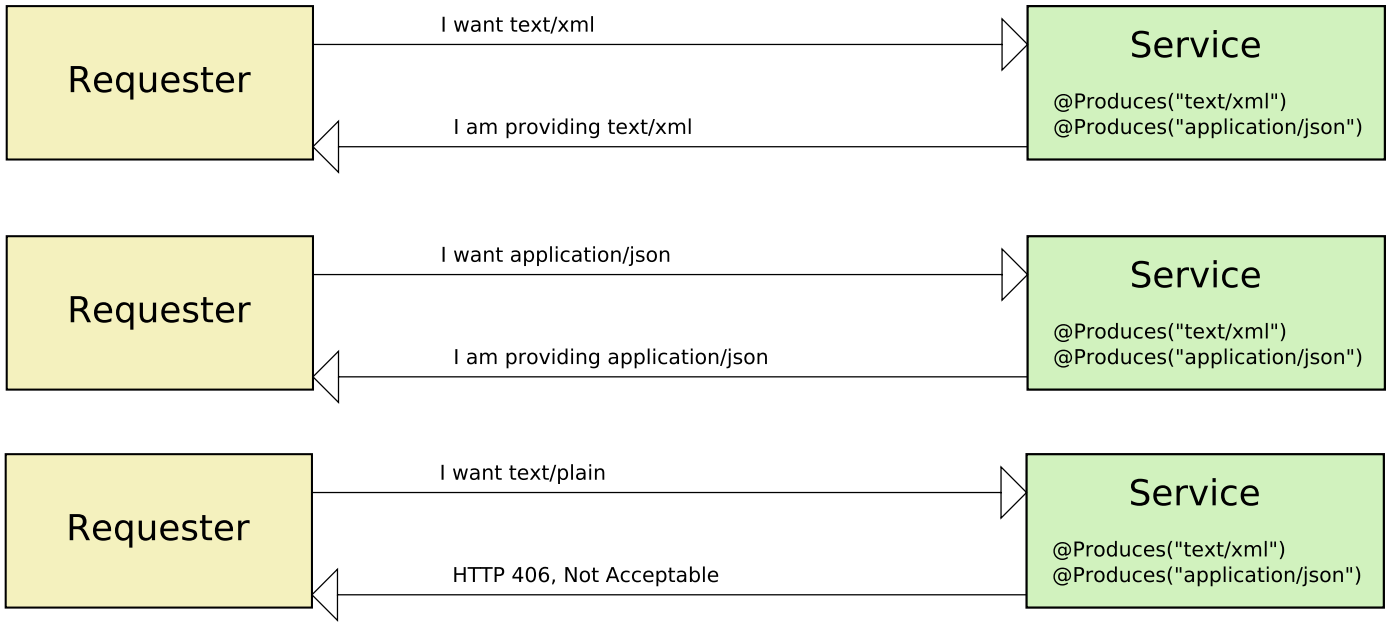
- Web services use the SOAP protocol to allow for remote procedure calls and document exchange
- JEE defines the JAX-WS API so that JEE developers can use Web services in a vendor-independent fashion



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to REST

- **Representational State Transfer (REST)** is an architectural style that emphasizes stateless transfer of resource representations using standard HTTP methods
- JEE defines the JAX-RS API so that JEE developers can develop RESTful applications



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to JavaMail

- The JavaMail API provides a standard way for Java applications to send and receive Internet email without having to program the SMTP protocol directly

```
1    Properties p = System.getProperties();
2    p.put("mail.host", "localhost" );
3
4    MimeMessage message =
5          new MimeMessage ( Session.getInstance(p, null ) );
6    message.setFrom
7           ( new InternetAddress ( "me@mycompany.com" ) );
8    message.setRecipients(Message.RecipientType.TO,
9          InternetAddress.parse ( "you@yourcompany.com" ) );
10
11   message.setSubject ( "Great Subject" );
12   message.setText( "The best message ever!" )
13   Transport.send(message);
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- What JEE is all about
- Fundamentals of several JEE technologies

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Servlets

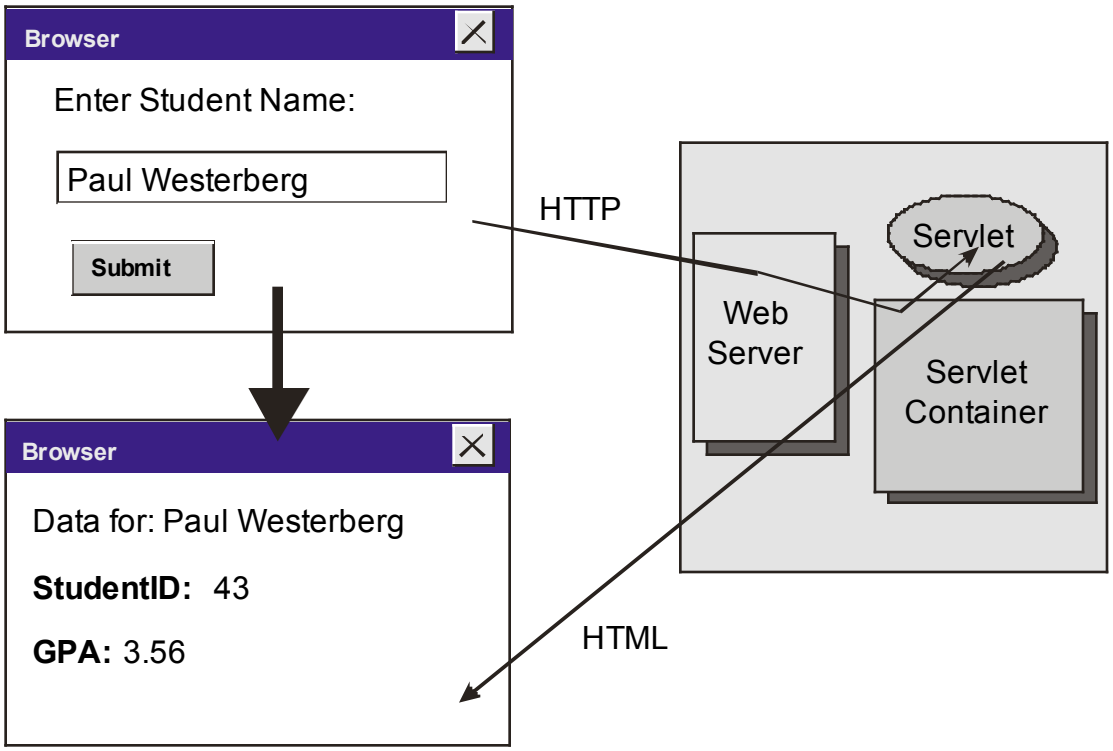
- Why Use Servlets?
- The Servlet Container
- The Servlet API
- The Servlet Lifecycle

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is a Servlet?

- A servlet is a Java program that's invoked by a Web server -- servlets typically generate HTML
- Servlets can take the place of traditional CGI programs



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Why Use Servlets?

- Servlets scale better than CGI programs where the Web server spawns a separate process for each request. In contrast, servlets can stay in memory once they are loaded
- Since servlets are written in Java, they are portable across platforms
- Unlike other proprietary schemes, servlets are widely supported by application servers and thus are portable across vendors

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Servlet Containers

- Servlet support is not included with web servers like Apache. To support servlets, you must install a a container for the servlet objects
- There are several popular servlet containers including:
  - Apache Tomcat
  - IBM WebSphere
  - Oracle WebLogic

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

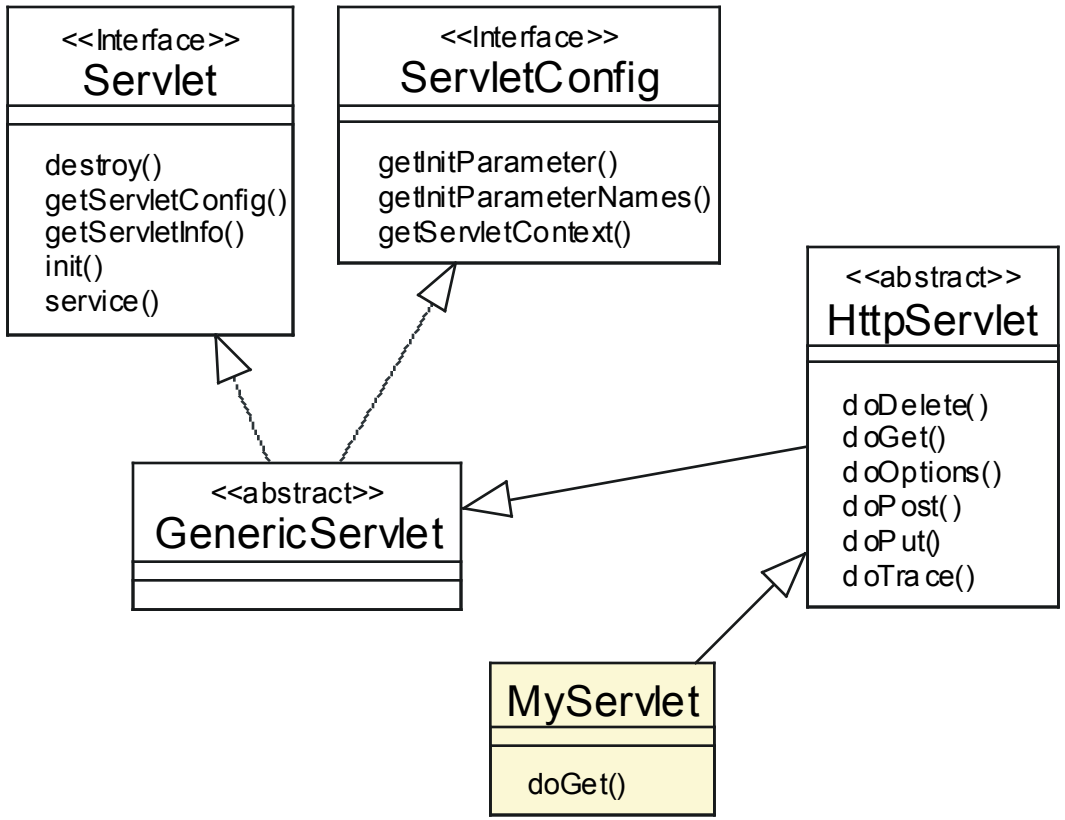
The Servlet API

- Sun has defined a set of standard interfaces that you must implement when you write a servlet. The container calls the methods in the interfaces at well defined times
- The container also implements a standard set of interfaces that the servlet can call if required

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Writing a Servlet

- You can extend the HttpServlet abstract class to implement your servlet
- We will examine these interfaces in more detail later in the course



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A Sample Servlet

```
 1    import java.io.*;
 2    import javax.servlet.*;
 3    import javax.servlet.http.*;
 4    import javax.servlet.annotation.*;
 5
 6    @WebServlet("/MyServlet")
 7    public class MyServlet extends HttpServlet
 8    {
 9        public void doGet ( HttpServletRequest request
10                            , HttpServletResponse response)
11            throws ServletException, IOException
12        {
13            response.setContentType ( "text/html" );
14            PrintWriter out = response.getWriter();
15            out.println ("<html><body>" );
16            for ( int i=0; i < 5; i++ )
17                out.println ("<p>Hello, <b>world</b></p>" );
18            out.println ("</body></html>" );
19        }
20   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- The advantages of using servlets
- About servlet containers
- How to write a simple servlet

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to JavaServer Pages

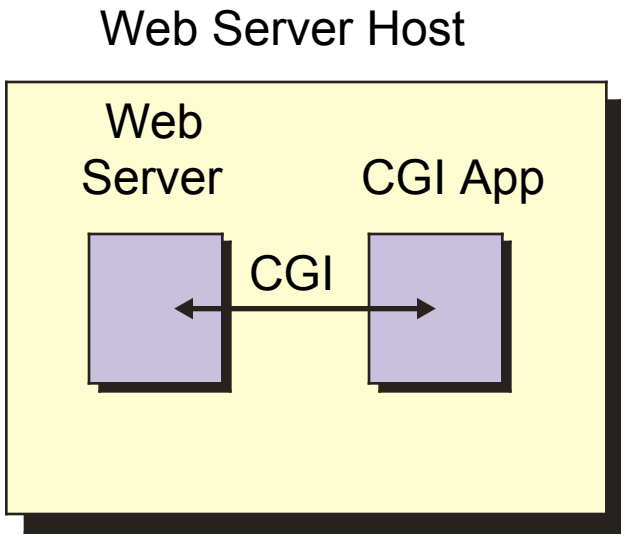
- CGI Alternatives
- Servlets and JSPs
- JSP Version History
- Intro to JSP Expression Language

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is CGI?

- The *Common Gateway Interface* is a standard protocol that web servers like Apache use to invoke programs on the web server host
- Many CGI programs are written in the Perl language
- The web server passes request arguments to the CGI program on the command line or via environment variables



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

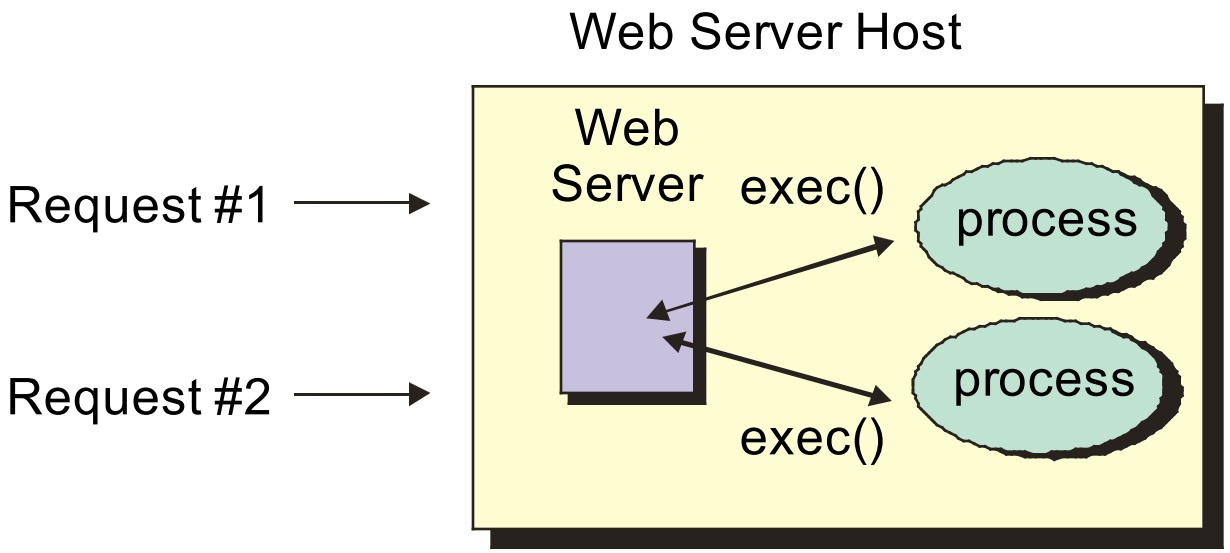
A Sample Perl CGI Script

```
1    #! /usr/bin/perl
2    require "cgi-lib.pl";
3
4    if ( &ReadParse ( *input ) )
5    {
6        print "Content-type: text/html\n\n";
7        print ( "<html><body>\n" );
8
9        @arglist = split ( "\0", $input{"HelloCount"} );
10       $printcount = $arglist[0];
11
12       for ( $i = 0; $i < $printcount; $i++ )
13       {
14           print ( "<p>Hello</p>\n" );
15       }
16
17       print ( "</html></body>\n" );
18   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What's Wrong with CGI?

- CGI, especially in older implementations, does not scale well since the web server runs each request in a separate process
- Since many CGI scripts are written in interpreted languages such as Perl, performance may suffer
- Perl, though powerful, is not particularly easy to learn or maintain



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

CGI Enhancements and Alternatives

- Mod_perl
- PHP
- Microsoft Active Server Pages (ASP)
- ColdFusion
- Java2 Enterprise Edition (includes servlets and JavaServer Pages)
- Others

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Microsoft's ASP

- An ASP consists of HTML with embedded snippets of code
- ASP uses VBScript as its default scripting language
- ASP runs on Microsoft web servers and on other web servers via Chili!Soft
- Latest version known as ASP.NET

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A Sample ASP

```
1    <html>
2      <body>
3      <%
4        For i = 1 to 5
5          response.write ( "Hello, <b>world</b>" )
6        Next
7      %>
8      </body>
9    </html>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

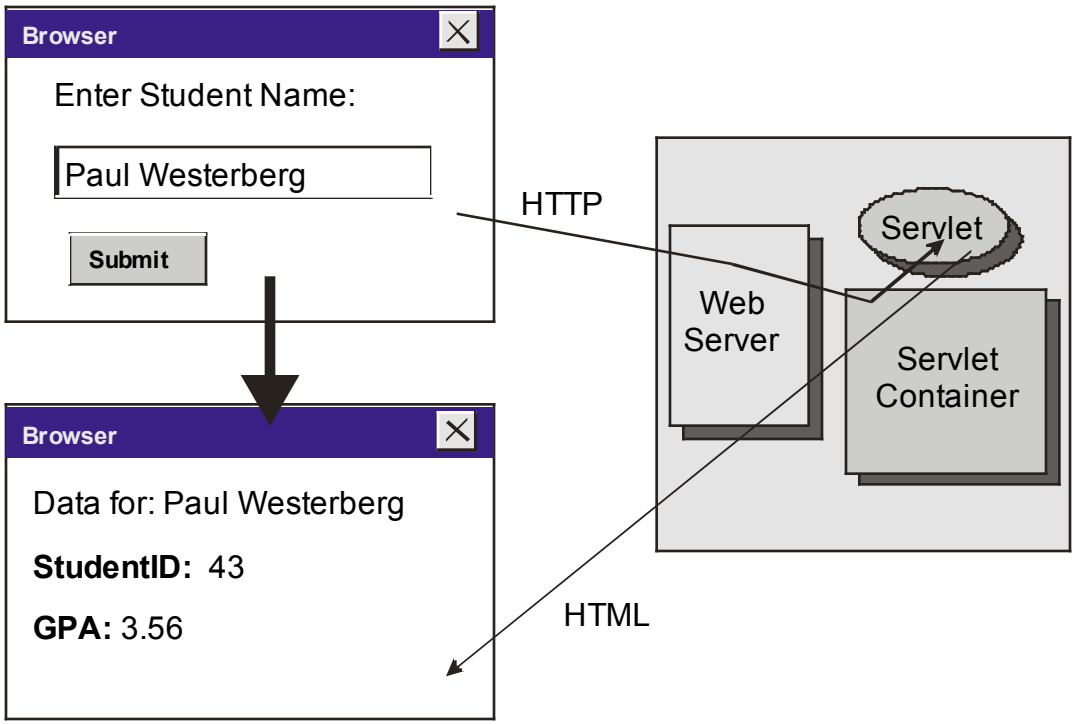
What is Java2 Enterprise Edition?

- J2EE is a set of technologies for server-side Java programming. It includes:
  - Servlets
  - JavaServer Pages
  - JDBC
  - Remote Method Invocation (RMI)
  - Enterprise JavaBeans (EJB)
- There are many vendors implementing all or part of J2EE

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Servlets

- A servlet is a Java program that's invoked by a Web server -- servlets typically generate HTML
- Servlets can take the place of traditional CGI programs



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

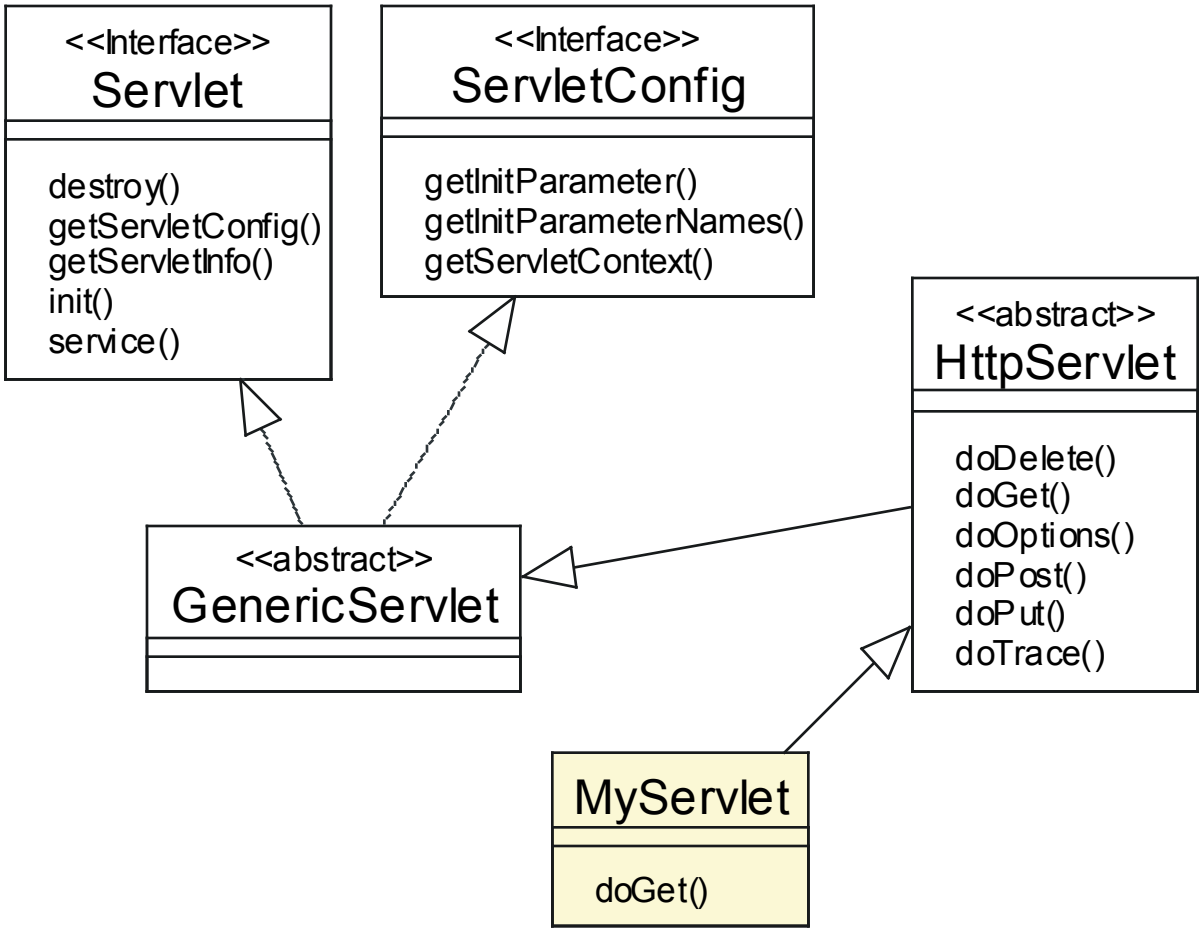
Why Use Servlets?

- Servlets scale better than CGI programs where the Web server spawns a separate process for each request. In contrast, servlets stay in memory once they are loaded
- Since servlets are written in Java, they are portable
- Unlike other proprietary schemes, servlets are widely supported by application servers and popular web servers

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Writing a Servlet

- You can extend the HttpServlet abstract class to implement your servlet



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A Simple Servlet

```
1     import java.io.*;
2     import javax.servlet.*;
3     import javax.servlet.http.*;
4
5     public class MyServlet extends HttpServlet
6     {
7         public void doGet (
8                     HttpServletRequest request
9                   , HttpServletResponse response )
10            throws ServletException, IOException
11        {
12            response.setContentType ( "text/html" );
13            PrintWriter out = response.getWriter();
14
15            out.println ("<html><body>" );
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A Simple Servlet, cont'd

```
16            for ( int i=0; i < 5; i++ )
17            {
18                 out.println (
19                    "Hello, <b>world</b>" );
20            }
21             out.println ("</body></html>" );
22         }
23    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

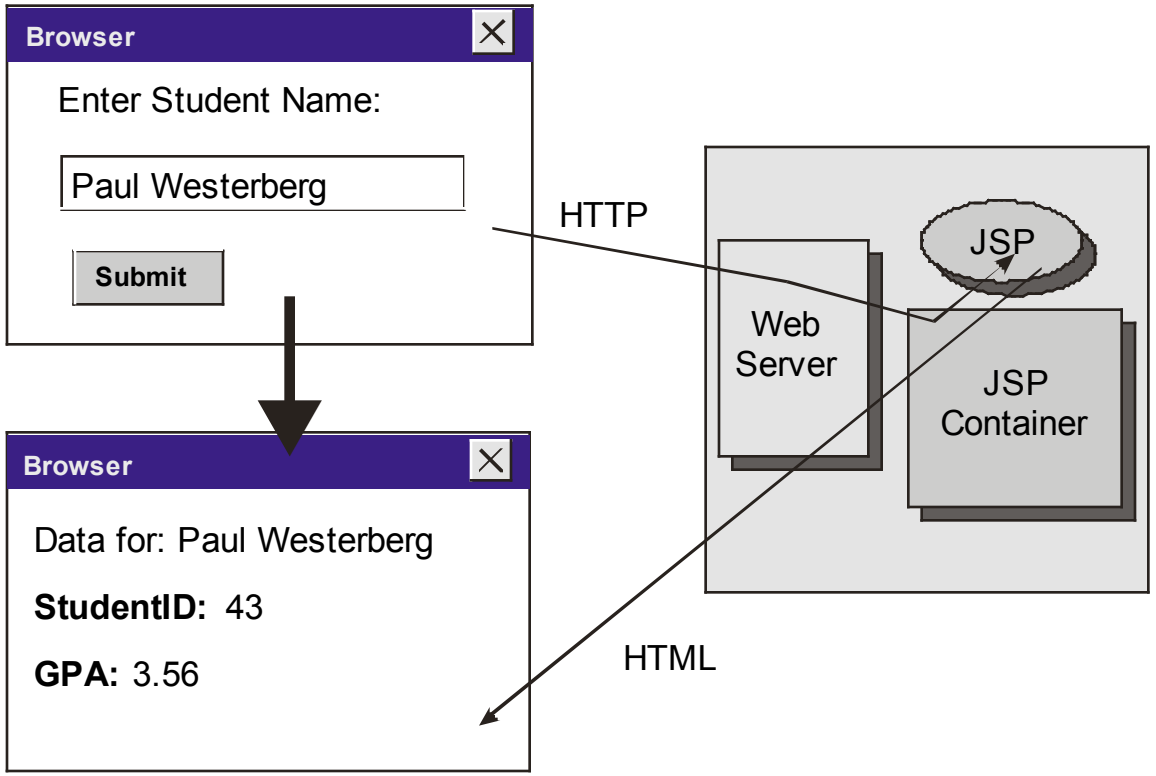
Generating HTML in a Servlet

- Servlets are portable, object-oriented and perform well, but they are not particulary suited to heavy-duty HTML generation:
  - It's hard to find the actual HTML amid all the Java code
  - If you decide to change the HTML, you need to recompile the servlet
  - It's hard for web-page designers who are not Java programmers to create and maintain servlets
- Is there better way to generate HTML?

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to JavaServer Pages

- JavaServer Pages are typically HTML with embedded program snippets (Java or **JSP actions**)
- Upon first use, the container converts the JSP into a servlet and then compiles the servlet



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Why Use JSPs?

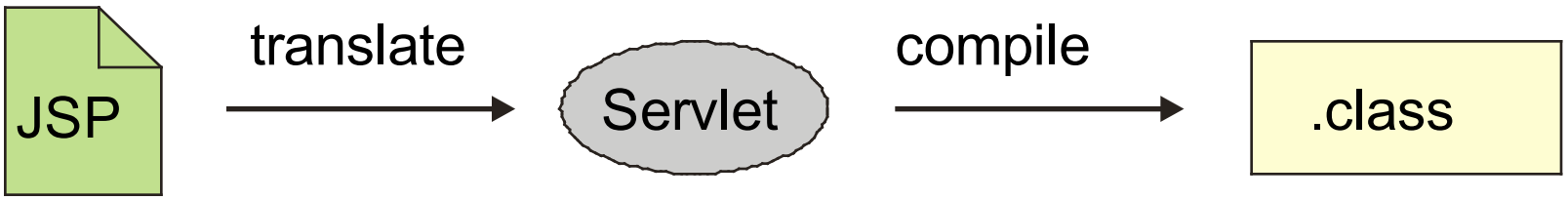
- JSPs are easier than servlets for non-programmers to work with since they ideally contain mostly plain HTML

```
1    <jsp:useBean id="now" class="java.util.Date"/>
2    <html>
3      <body>
4        <h3>My First Page</h3>
5        <p>
6          The <b>current</b> time is: ${now}
7        </p>
8      </body>
9    </html>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSP Compilation

- Upon first use, the container translates JSPs into servlets, then compiles and deploys the servlet
- Since the container converts JSPs into servlets, they perform well and are portable
- Most container vendors provide a *pre-compilation* facility



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSP Version History

- Version 1.0 introduced JSPs and defined basic syntax of **directives** and **Java scripting elements**
- Version 1.1 added **action elements** and **tag libraries**
- Version 2.0 added an **expression language** and a simpler way to write tag libraries

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

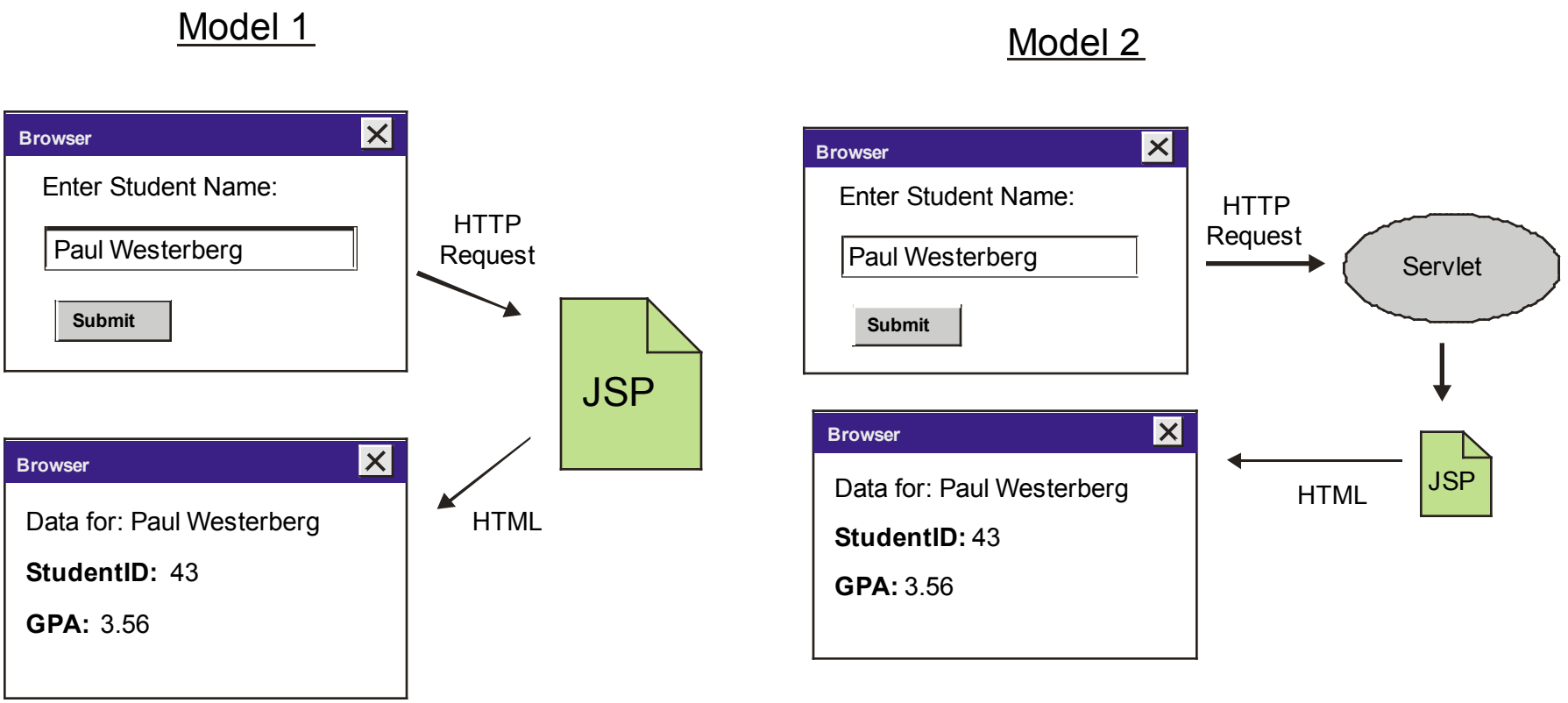
JSP Version History, cont'd

```
1    <!-- DIRECTIVE (JSP 1.0)-->
2    <%@ taglib uri="http://java.sun.com/jsp/jstl/core"
3      prefix="c" %>
4
5    <!-- JSP STANDARD ACTION (JSP 1.1)-->
6    <jsp:useBean id="now" class="java.util.Date"/>
7
8    <!-- JSP EXPRESSION IN TEMPLATE TEXT (JSP 2.0)-->
9    <p>Time: ${now}</p>
10
11   <!-- CUSTOM ACTION (JSP 1.1)-->
12   <c:if test="${empty param.gpa}">
13     <p>GPA is required</p>
14   </c:if>
15
16   <!-- SCRIPTING ELEMENT (JSP 1.0) -->
17   <%
18     for(int i=0; i<5; i++)
19       out.println("Hello, world");
20   %>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Model 1 versus Model 2

- In a **Model 1** architected application, JSPs respond directly to HTML forms
- In a **Model 2** (aka MVC) application, servlets process forms and then transfer control to JSPs



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Intro to the JSP Expression Language

- The JSP **expression language** is a language, based on Java, that makes it easy for JSP authors to perform calculations and access data defined in *JavaBeans*
- Prior to JSP 2.0, you could only use the expression language in conjunction with the *JSP Standard Tag Library* (JSTL)

```
<p>
  The total is: ${item.quantity * item.price}
</p>
<p>
  The total is:
    <c:out value="${item.quantity * item.price}"/>
</p>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- About alternatives to traditional CGI
- The basics of writing servlets and JSPs

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Spring MVC

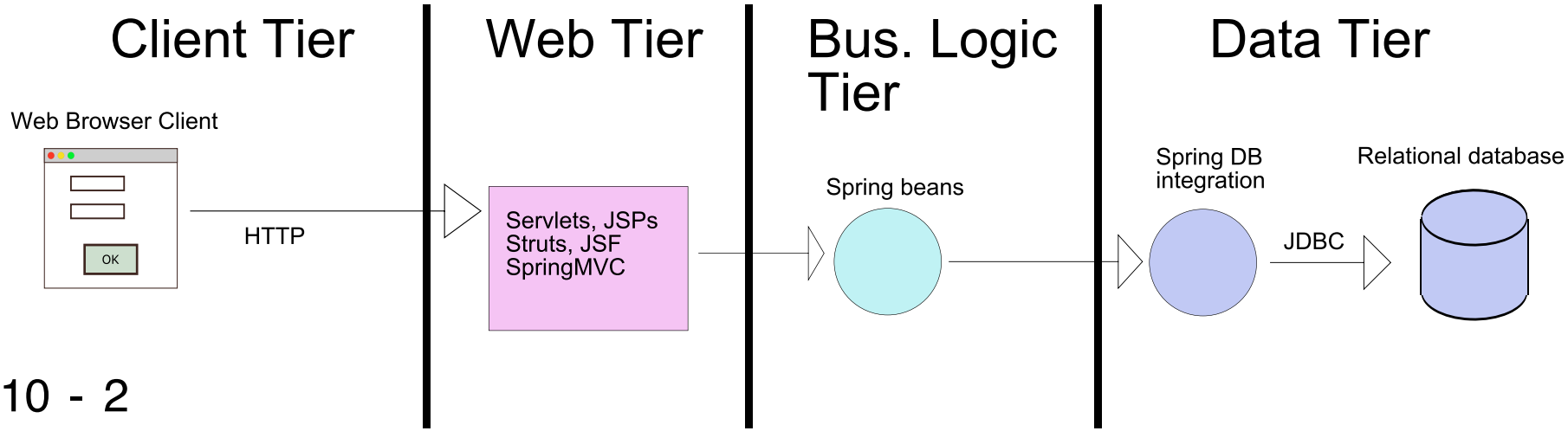
- MVC Fundamentals
- Spring MVC Architecture
- Spring MVC Sample Application

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring and the Web

- There are three basic strategies for fronting Spring applications with a Web front end:
  - Use standard JEE servlets and JSPs as the Web tier
  - Use a JEE Web framework such as Struts or JSF as the Web tier
  - Use the SpringMVC framework as the Web tier



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Spring WebApplicationContext

- If you simply want to use a Spring business layer from a servlet/JSP, Struts or JSF Web app, Spring provides a **context listener** and a **WebApplicationContext**

```
1    web.xml fragment
2    ----------------
3
4    <context-param>
5      <param-name>contextConfigLocation</param-name>
6      <param-value>/WEB-INF/spring.xml</param-value>
7    </context-param>
8    <listener>
9      <display-name>ContextLoaderListener</display-name>
10     <listener-class>
11      org.springframework.web.context.ContextLoaderListener
12     </listener-class>
13   </listener>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Spring WebApplicationContext, cont'd

```
1    Servlet class fragment
2    ----------------------
3
4    protected void doGet(HttpServletRequest request,
5     HttpServletResponse response)
6       throws ServletException, IOException
7    {
8      ServletContext servletContext = getServletContext();
9      WebApplicationContext ctx =
10       WebApplicationContextUtils.
11          getRequiredWebApplicationContext(
12           servletContext);
13
14     Student s1 = (Student)ctx.getBean("student1");
15
16     response.setContentType("text/html");
17     PrintWriter out = response.getWriter();
18     out.println("Student name: " + s1.getName() + "<br>");
19   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

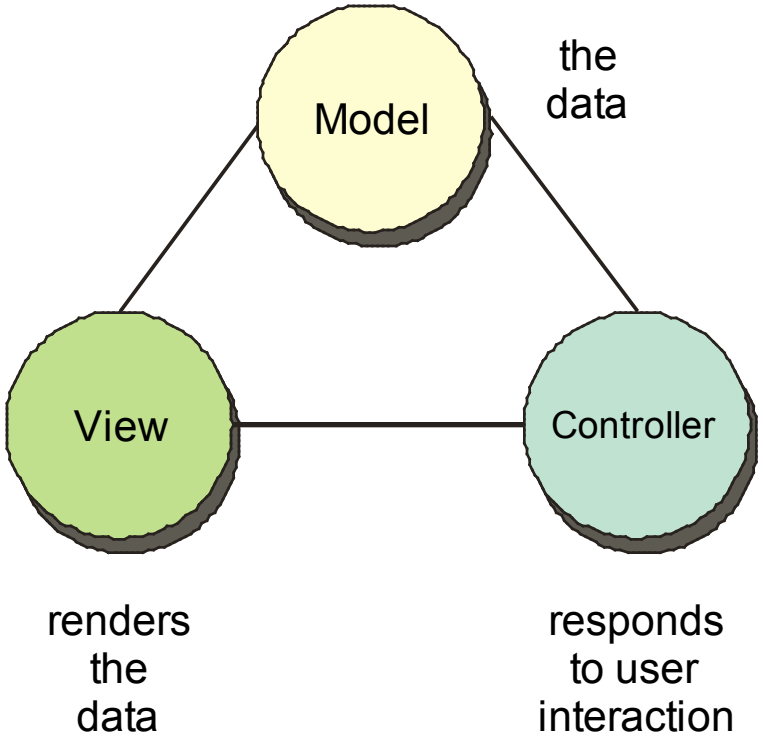
Introduction to Spring MVC

- Spring MVC is a Spring module that makes it fairly easy to create sophisticated Web front ends that use the Model-View-Controller design pattern
- You can using Spring MVC as an alternative to standard servlets/JSPs, Struts, JSF or other JEE Web frameworks
- Version 2.5 and later support using **annotations** to configure how Spring MVC maps *controllers* and maps requests

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is the MVC Design Pattern?

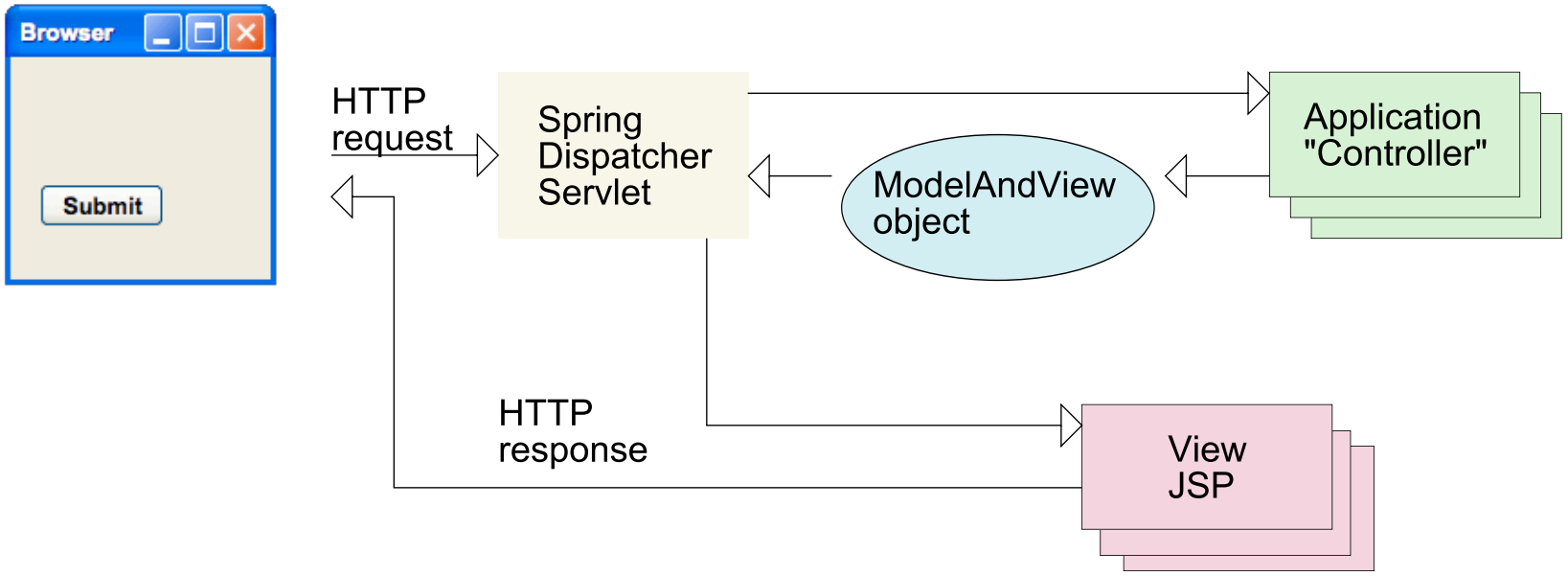
- The **Model-View-Controller** (MVC) architecture originated in the 1980s in the Smalltalk language, where it was used to build graphical user interfaces
- MVC is effective since it decomposes applications into components that you can develop, debug and maintain separately



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring MVC Architecture

- Spring MVC supplies a **front controller** servlet that acts as a single point of entry to an application
- Developers write "command controllers" that respond to requests from client



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

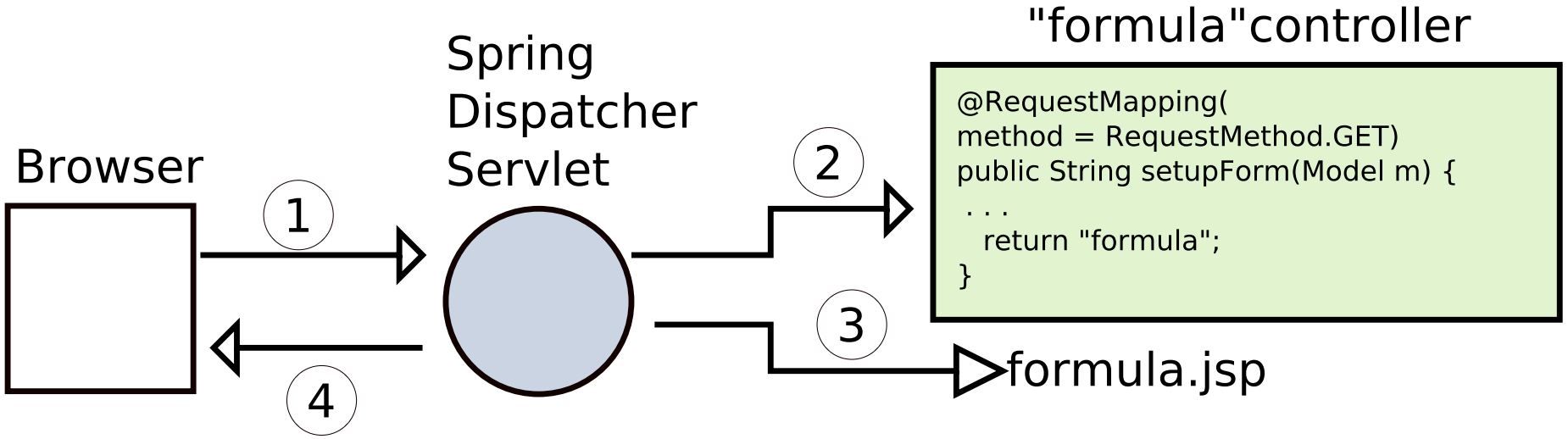
Spring MVC Controllers

- An application **controller** (not to be confused with a *front* controller), is an object that responds to a request, typically by populating a model and specifying a view
- In Spring 2.5 and later, controllers can be POJOs that you configure using annotations

```
1    @Controller
2    @RequestMapping("/formula")
3    public class QuadraticController
4    {
5      . . .
6    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

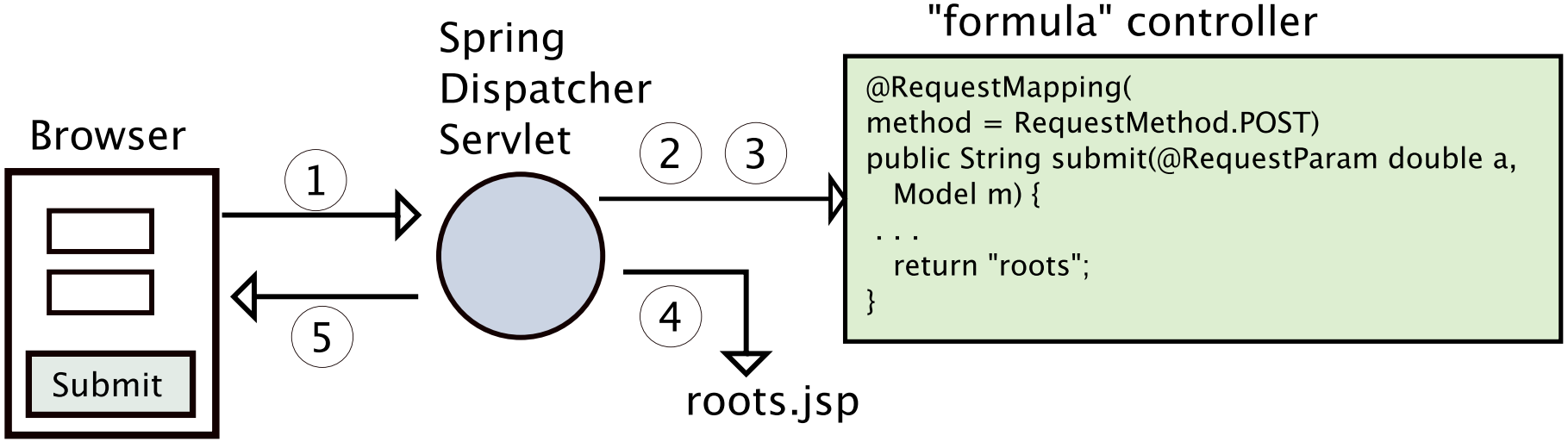
Form Processing Flow: Phase 1



1. Browser sends GET /formula.html
2. DispatcherServlet finds controller with "formula" URL and invokes a "setup" method configured with @RequestMapping(method = RequestMethod.GET). The "setup" method optionally initializes a model and returns a String with a JSP name.
3. The DispatcherServlet executes the JSP, initializing form fields from model
4. DispatcherServlet sends response with form back to browser

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Form Processing Flow: Phase 2



1. Browser sends POST /formula.html
2. DispatcherServlet maps HTTP request parameters to "submit" method parameters
3. DispatcherServlet invokes "submit" method annotated with @RequestMapping(method = RequestMethod.POST). The "submit" method processes request, populates model and specifies view JSP
4. DispatcherServlet forwards request to view JSP - the JSP accesses the model and generates a response
5. DispatcherServlet returns response to browser

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

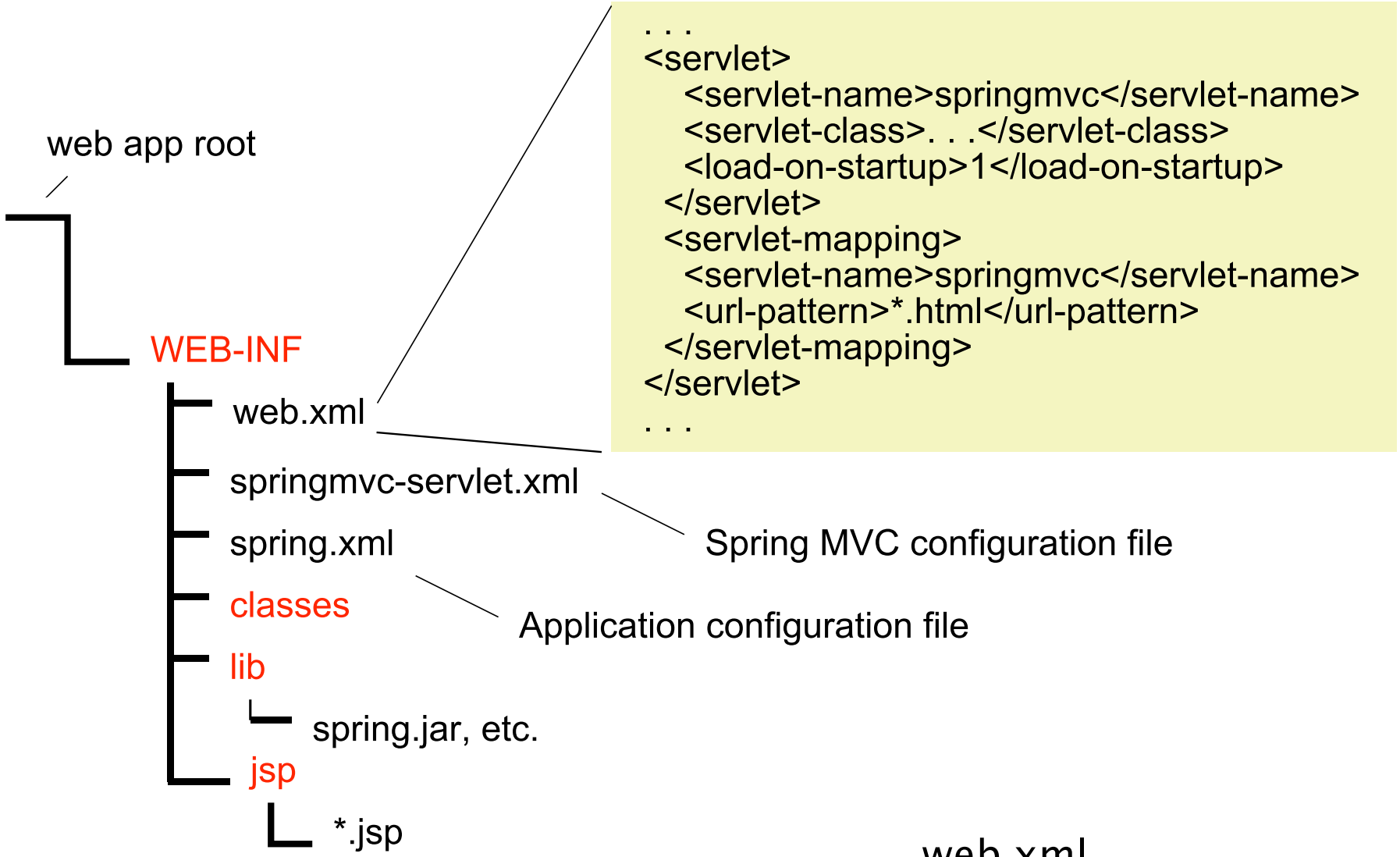
Spring MVC Development Steps

For a Spring SimpleFormController app:

- Configure the Web application:
  - Configure Spring MVC Dispatcher servlet in web.xml
  - Add Spring MVC JARS to WEB-INF/lib
- Create an HTML input form, optionally using Spring MVC custom actions
- Create an application **controller** class to process the form
- Write Spring configuration XML file(s)

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring the Web Application



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    id="WebApp_ID" version="3.0">

    <listener>
        <listener-class>org.springframework.web.util.Log4jConfigListener</listener-class>
    </listener>
    <listener>
        <display-name>ContextLoaderListener</display-name>
        <listener-class>
            org.springframework.web.context.ContextLoaderListener
        </listener-class>
    </listener>
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring.xml</param-value>
    </context-param>
    <servlet>
        <servlet-name>springmvc</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>springmvc</servlet-name>
        <url-pattern>*.html</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

A Service Bean

- In the normal fashion, you can write Spring beans that provide application services
- As always, it's a good idea to write an interface first for application service beans

$$ax^2 + bx + c = 0$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

SolveQuadratic.java

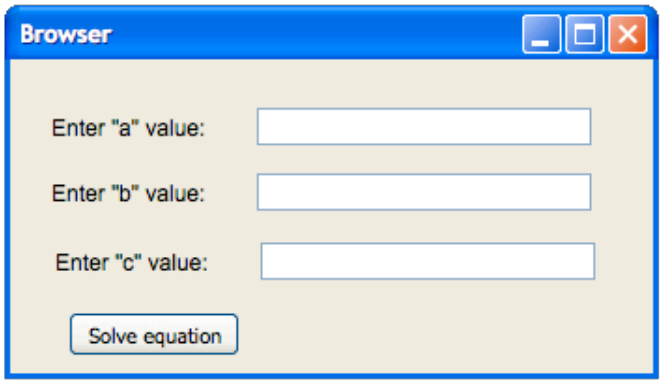
SolveQuadraticImpl.java

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    package equations;
2
3    public interface SolveQuadratic
4    {
5        public double getRoot1(double a, double b, double c);
6        public double getRoot2(double a, double b, double c);
7    }
```

```
1    package equations;
2
3    @Component("solveQuadratic")
4    public class SolveQuadraticImpl implements SolveQuadratic
5    {
6
7        public double getRoot1(double a, double b, double c)
8        {
9            return (-b + Math.sqrt(b * b - 4 * a * c)) / 2 * a;
10       }
11
12       public double getRoot2(double a, double b, double c)
13       {
14           return (-b - Math.sqrt(b * b - 4 * a * c)) / 2 * a;
15       }
16   }
```

A Spring MVC Input Form



formula.jsp

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
<html>
  <head>
    <title>Solve The Quadratic Formula</title>
  </head>
  <body>
    <h3>Solve The Quadratic Formula</h3>
    <p>
      The Quadratic Formula solves equations of the form   
      <img src="equation.png"> using this formula:
    </p>
    <p>
      <img src="QuadraticFormula.jpg">
    </p>
    <form action="formula.html" method="post">
      <table>
        <tr>
          <td>Enter <i>a</i> value:</td>
          <td><input type="text" name="a" value="${a}" /></td>
        </tr>
        <tr>
          <td>Enter <i>b</i> value:</td>
          <td><input type="text" name="b" value="${b}" /></td>
        </tr>
        <tr>
          <td>Enter <i>c</i> value:</td>
          <td><input type="text" name="c" value="${c}" /></td>
        </tr>
        <tr>
          <td>
            <input type="submit" value="Solve equation">
          </td>
        </tr>
      </table>
    </form>
  </body>
</html>
```

A Spring MVC Result Page

```
1     <html>
2       <head>
3         <title>Equation Solved</title>
4       </head>
5       <body>
6          <h3>Equation Solved</h3>
7          <p>For input values:</p>
8
9           a=${a}<br>
10          b=${b}<br>
11          c=${c}<br>
12
13         <p><b>Root 1:</b> ${root1}</p>
14         <p><b>Root 2:</b> ${root2}</p>
15      </body>
16    </html>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring Controller Annotation Processing

- Since our controller uses annotations, we must enable annotation processing in the Spring MVC configuration file

```
1    Spring MVC configuration file
2    ----------------------------
3
4    <context:component-scan
5         base-package="controllers" />
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is a ViewResolver?

- So that controllers do not need to hard-code the file names of JSPs, SpringMVC provides **ViewResolver** classes that map logical strings into actual file names

```
1    Spring MVC configuration file
2    ----------------------------
3
4    <bean id="viewResolver"
5      class=
6    "org.springframework.web.servlet.view.InternalResourceViewResolver">
7      <property name="viewClass"
8        value="org.springframework.web.servlet.view.JstlView" />
9      <property name="prefix" value="/WEB-INF/jsp/" />
10     <property name="suffix" value=".jsp" />
11   </bean>
```

For example, if a controller method returns "roots" as a view logical string, this resolver will map to a JSP named "/WEB-INF/jsp/roots.jsp"

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A Spring MVC Command Controller

QuadraticEquationController.java

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```java
package controllers;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.ApplicationContext;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;

import equations.SolveQuadratic;

@Controller
@RequestMapping("/formula")
public class QuadraticEquationController
{
    @Autowired private ApplicationContext ctx;

    @RequestMapping(method = RequestMethod.GET)
    public String setupForm(Model model)
    {
        model.addAttribute("a", "1");
        model.addAttribute("b", "3");
        model.addAttribute("c", "-4");

        return "formula";
    }

    @RequestMapping(method = RequestMethod.POST)
    public String submitForm(@RequestParam double a, @RequestParam double b,
            @RequestParam double c, Model model)
    {
        SolveQuadratic solver = (SolveQuadratic)ctx.getBean("solveQuadratic");

        model.addAttribute("a", a);
        model.addAttribute("b", b);
        model.addAttribute("c", c);

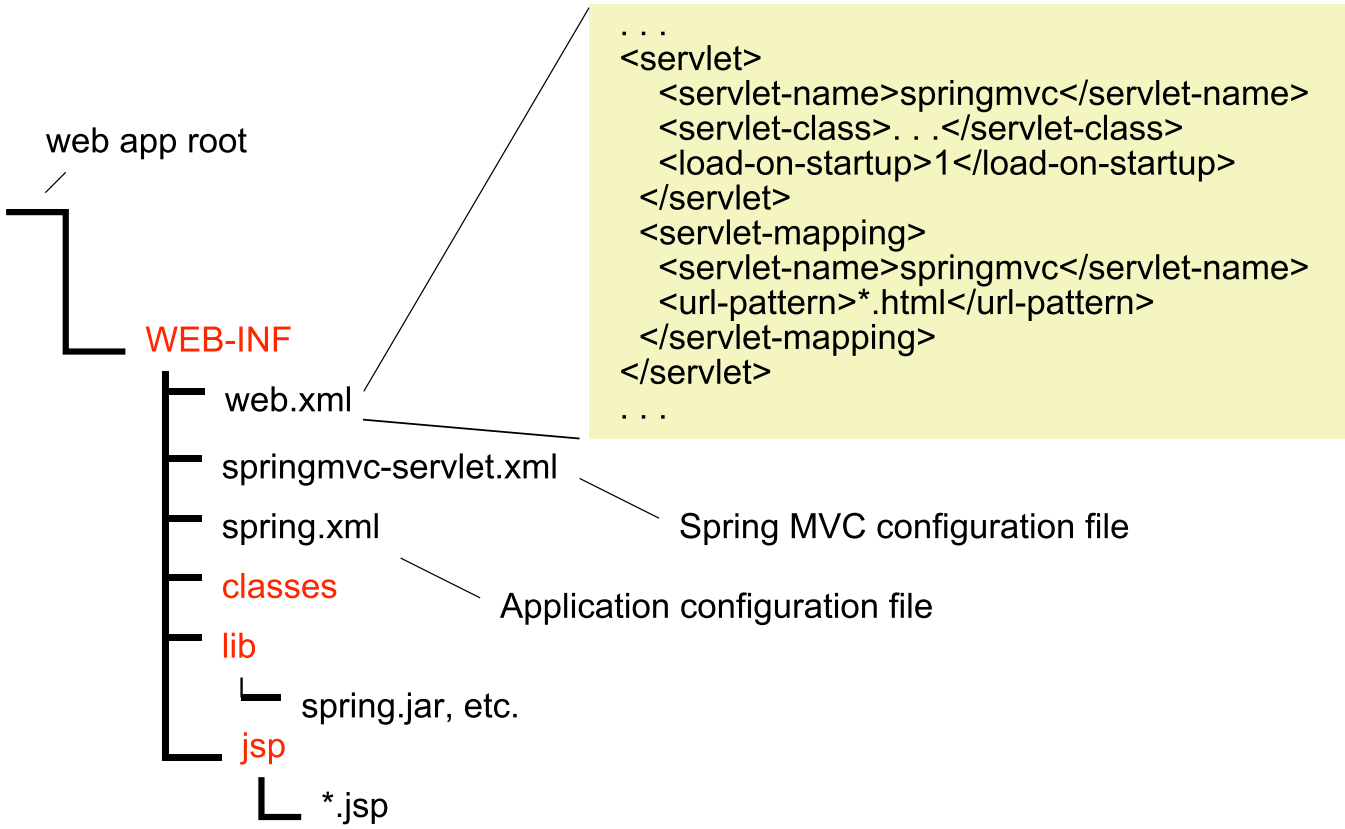
        model.addAttribute("root1", solver.getRoot1(a, b, c));
        model.addAttribute("root2", solver.getRoot2(a, b, c));

        return "roots";
    }
}
```

The Spring Configuration File

- By default, the Spring MVC servlet looks in the WEB-INF folder for a file with name xxxxx-servlet.xml where xxxxx is the servlet name in web.xml

springmvc-servlet.xml



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
<?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:aop="http://www.springframework.org/schema/aop"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:jee="http://www.springframework.org/schema/jee" xmlns:jms="http://www.springframework.org/schema/jms"
    xmlns:lang="http://www.springframework.org/schema/lang" xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-2.5.xsd
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-2.5.xsd
        http://www.springframework.org/schema/jee
        http://www.springframework.org/schema/jee/spring-jee-2.5.xsd
        http://www.springframework.org/schema/lang
        http://www.springframework.org/schema/lang/spring-lang-2.5.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-2.5.xsd
        http://www.springframework.org/schema/util
        http://www.springframework.org/schema/util/spring-util-2.5.xsd">

    <context:component-scan base-package="controllers" />

    <bean id="viewResolver"
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="viewClass"
            value="org.springframework.web.servlet.view.JstlView" />
        <property name="prefix" value="/WEB-INF/jsp/" />
        <property name="suffix" value=".jsp" />
    </bean>

</beans>
```

MVC With Java Configuration

- If you use Java configuration for Spring MVC, you can avoid writing any Spring XML
- Recipe:
    - Write a configuration class annotated with @Configuration, @ComponentScan and @EnableWebMvc
    - In the configuration class, write bean-creation methods to configure a **view resolver** and any other application beans
    - Configure **contextClass** and **contextConfigLocation** in web.xml

javaconfg-web.xml

QuadraticConfig.java

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:web="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    id="WebApp_ID" version="3.0">
    <servlet>
        <servlet-name>springmvc</servlet-name>
        <servlet-class>
          org.springframework.web.servlet.DispatcherServlet
          </servlet-class>
        <init-param>
            <param-name>contextClass</param-name>
            <param-value>
              org.springframework.web.context.support.AnnotationConfigWebApplicationContext
            </param-value>
        </init-param>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>config.QuadraticConfig</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>springmvc</servlet-name>
        <url-pattern>*.html</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

```
1    package config;
2
3    import org.springframework.context.annotation.Bean;
4    import org.springframework.context.annotation.ComponentScan;
5    import org.springframework.context.annotation.Configuration;
6    import org.springframework.web.servlet.ViewResolver;
7    import org.springframework.web.servlet.config.annotation.EnableWebMvc;
8    import org.springframework.web.servlet.view.InternalResourceViewResolver;
9
10   import equations.SolveQuadratic;
11   import equations.SolveQuadraticImpl;
12
13   @Configuration
14   @EnableWebMvc
15   @ComponentScan("controllers")
16   public class QuadraticConfig
17   {
18       @Bean
19       public ViewResolver configureViewResolver()
20       {
21           InternalResourceViewResolver viewResolve =
22               new InternalResourceViewResolver();
23           viewResolve.setPrefix("/WEB-INF/jsp/");
24           viewResolve.setSuffix(".jsp");
25
26           return viewResolve;
27       }
28
29       @Bean
30       public SolveQuadratic solveQuadratic()
31       {
32           return new SolveQuadraticImpl();
33       }
34   }
```

Using Thymeleaf Instead of JSP

- **Thymeleaf** is a template engine that many Spring developers prefer instead of JSPs for the view:
    - Supports HTML5 out of the box and lets you better preview pages in a browser (JSPs often need to run tag libraries)
    - There is a Spring/Thymeleaf integration library



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Thymeleaf/Spring Recipe

- Steps for using Thymeleaf:
    - Load the Thymeleaf libraries into the project
    - Write HTML5, XML-parseable Thymeleaf **templates** instead of JSPs
    - Configure Thymeleaf template-engine and view-resolver beans

formula.html

roots.html

QuadraticThymeleafConfig.java

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
<!DOCTYPE html>
<html>
  <head>
    <title>Solve The Quadratic Formula</title>
  </head>
  <body>
    <h3>Solve The Quadratic Formula</h3>
    <p>
      The Quadratic Formula solves equations of the form   
      <img src="equation.png" /> using this formula:
    </p>
    <p>
      <img src="QuadraticFormula.jpg" />
    </p>
    <form action="formula.html" method="post">
      <table>
        <tr>
          <td>Enter <i>a</i> value:</td>
          <td><input type="text" name="a" th:value="${a}"/></td>
        </tr>
        <tr>
          <td>Enter <i>b</i> value:</td>
          <td><input type="text" name="b"  th:value="${b}"/></td>
        </tr>
        <tr>
          <td>Enter <i>c</i> value:</td>
          <td><input type="text" name="c"  th:value="${c}" /></td>
        </tr>
        <tr>
          <td>
            <input type="submit" value="Solve equation" />
          </td>
        </tr>
      </table>
    </form>
  </body>
</html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Equation Solved</title>
  </head>
  <body>
     <h3>Equation Solved</h3>
     <p>For input values:</p>

      <div>
         a=<span th:text="${a}"></span>
      </div>
      <div>
         b=<span th:text="${b}"></span>
      </div>
      <div>
         c=<span th:text="${c}"></span>
      </div>

     <p><b>Root 1:</b> <span th:text="${root1}"></span></p>
     <p><b>Root 2:</b> <span th:text="${root2}"></span></p>
  </body>
</html>
```

```
1   package config;
2
3   import org.springframework.context.annotation.Bean;
4   import org.springframework.context.annotation.ComponentScan;
5   import org.springframework.context.annotation.Configuration;
6   import org.springframework.web.servlet.config.annotation.DefaultServletHandlerConfigurer;
7   import org.springframework.web.servlet.config.annotation.EnableWebMvc;
8   import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;
9   import org.thymeleaf.spring4.SpringTemplateEngine;
10  import org.thymeleaf.spring4.view.ThymeleafViewResolver;
11  import org.thymeleaf.templateresolver.ServletContextTemplateResolver;
12
13  import equations.SolveQuadratic;
14  import equations.SolveQuadraticImpl;
15
16  /*
17   *   Note that we extend WebMvcConfigurerAdaptor
18   *   so we can override configureDefaultServletHandling().
19   *   That lets us configure the Spring MVC servlet so
20   *   it loads static resources (e.g. images) from the
21   *   WebContent folder (context root).
22   */
23
24  @Configuration
25  @EnableWebMvc
26  @ComponentScan("controllers")
27  public class QuadraticThymeleafConfig extends WebMvcConfigurerAdapter
28  {
29      @Bean
30      public ServletContextTemplateResolver templateResolver()
31      {
32          ServletContextTemplateResolver resolver =
33             new ServletContextTemplateResolver();
34
35          resolver.setPrefix("/WEB-INF/templates/");
36          resolver.setSuffix(".html");
37          resolver.setTemplateMode("HTML5");
38          resolver.setCacheable(false);
39
40          return resolver;
41      }
42
43      @Bean
44      SpringTemplateEngine templateEngine()
```

```
46          SpringTemplateEngine templateEngine = new SpringTemplateEngine();
47
48          templateEngine.setTemplateResolver(templateResolver());
49
50          return templateEngine;
51      }
52
53      @Bean
54      public ThymeleafViewResolver viewResolver()
55      {
56          ThymeleafViewResolver tvr = new ThymeleafViewResolver();
57          tvr.setTemplateEngine(templateEngine());
58
59          return tvr;
60      }
61
62      @Override
63      public void configureDefaultServletHandling(
64              DefaultServletHandlerConfigurer configurer)
65      {
66          configurer.enable();
67      }
68
69      @Bean
70      public SolveQuadratic solveQuadratic()
71      {
72          return new SolveQuadraticImpl();
73      }
74  }
```

Chapter Summary

In this chapter, you learned:

- Fundamental Model-View-Controller concepts
- The basics of the Spring MVC module

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring and JDBC

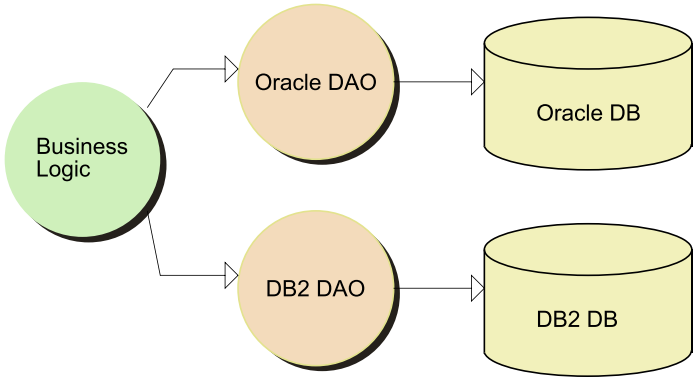
- The DAO Design Pattern
- JdbcTemplate and SimpleJdbcTemplate
- JdbcDaoSupport

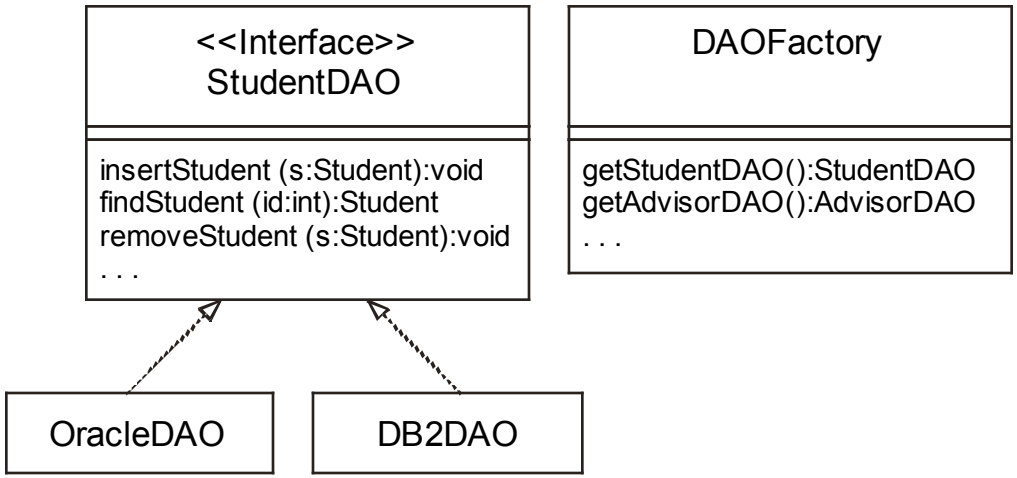
Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The DAO Design Pattern

- Spring supports and encourages the JEE DAO design pattern and simplifies coding, even if you you use basic JDBC





Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What's Wrong with JDBC?

- Even if you use DAOs, JDBC involves a lot of low-level, tedious and repetitive coding
- Managing JDBC connections is prone to errors
- OR/M solutions like Hibernate help with some of these issues

```
1    Connection con = null;
2    try {
3      con = dataSource.getConnection();
4      . . .
5    }
6    catch (SQLException exc) {
7    . . . }
8    finally {
9      try {
10       if (con != null) con.close();
11     catch (Exception e) {}
12   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introducing the Spring JDBC Module

- Spring provides the following support for database access:
    - Classes that retrieve an underlying JDBC **data source**
    - **Template** classes that simplify database coding, including templates for basic JDBC, Hibernate and JPA
    - **Support** classes that you can subclass when writing your DAOs
    - An improved exception hierarchy that's based on unchecked (runtime) exceptions

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The JdbcTemplate Class

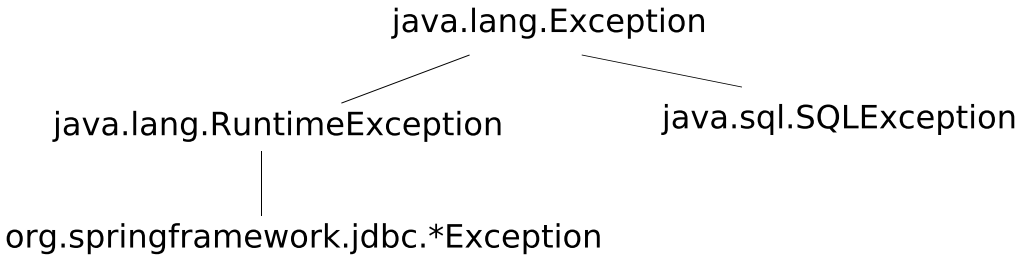
- This is the workhorse class for programs that use Spring's basic JDBC support
- It provides methods to let you query, update, insert and delete rows in a table

org.springframework.jdbc.core.JdbcTemplate
batchUpdate(sql:String[]):int[] call(csc:CallableStatementCreator, parms:List):Map createConnectionProxy(con:Connection):Connection execute(sql:String):void extractOutputParameters(cs:CallableStatement, parms:List):Map getFetchSize():int getMaxRows():int getQueryTimeout():int getSingleColumnRowMapper(cls:Class):RowMapper handleWarnings(warning:SQLWarning):void isIgnoreWarnings():boolean isResultsMapCaseInsensitive():boolean isSkipResultsProcessing():boolean isSkipUndeclaredResults():boolean queryForInt(sql:String, args:Object[]):int queryForList(sql:String):List queryForLong(sql:String):long queryForMap(sql:String):Map queryForObject(sql:String, arg1:Class):Object queryForRowSet(arg0:String):SqlRowSet setFetchSize(size:int):void setIgnoreWarnings(b:boolean):void setMaxRows(rows:int):void setQueryTimeout(timeOut:int):void setResultsMapCaseInsensitive(b:boolean):void setSkipResultsProcessing(b:boolean):void setSkipUndeclaredResults(b:boolean):void update(sql:String):int

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring JDBC Exceptions

- JDBC provides only the single SQLException and it's a **checked** exception, which can require messy nested **try-catch** blocks
- Spring provides many more precisely named **unchecked** exceptions -- exception handling is simpler and more intuitive



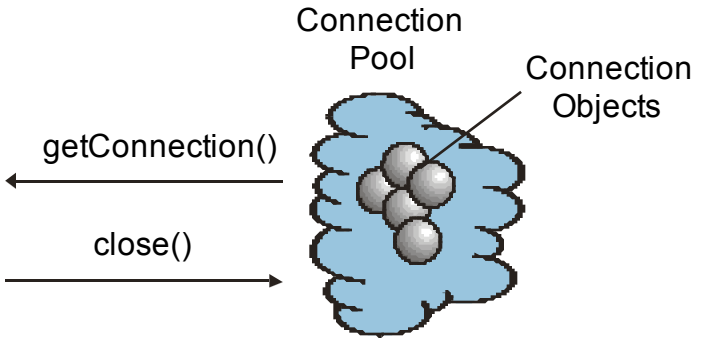
Selected Spring JDBC Exceptions (in org.springframework.jdbc)

BadSqlGrammarException
CannotGetJdbcConnectionException
IncorrectResultSetColumnCountException
InvalidResultSetAccessException
JdbcUpdateAffectedIncorrectNumberOfRowsException
LobRetrievalFailureException
SQLWarningException
UncategorizedSQLException

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JDBC Connection Review

- Non-Spring JDBC applications can obtain a connection in one of two ways:
  - Using the DriverManager interface
  - Using a DataSource
- The DriverManager approach is appropriate for standalone Java programs or for testing of enterprise applications
- The DataSource approach requires a JEE runtime that provides a JNDI service, but allows for **pooled** connections



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Working with Data Sources in Spring

- Spring lets you treat both JDBC approaches in a similar fashion
- Spring provides:
    - A DriverManagerDataSource class that uses the JDBC DriverManager approach
    - A JndiObjectFactoryBean class or a <jee:jndi-lookup> configuration element that locate JNDI-based DataSources
- Since you configure these in the Spring configuration file, you can easily switch between the two approaches

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring a DriverManager Source (XML)

- The Spring DriverManagerDataSource wraps the JDBC DriverManager and is appropriate for standalone programs
- You can inject the data source bean into your DAOs using XML, annotations or Java configuration

```
1    <bean id="studentDataSource"
2      class="org.springframework.jdbc.datasource.DriverManagerDataSource">
3      <property name="driverClassName"
4        value="org.apache.derby.jdbc.EmbeddedDriver"/>
5      <property name="url"
6        value="jdbc:derby:c:\springclass\db\StudentDB"/>
7      <property name="username"
8        value="SA"/>
9      <property name="password"
10       value="SA"/>
11   </bean>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring a DriverManager (Java Config)

- With Java configuration, you can use the **DataSourceBuilder** to create and initialize a driver-manager DataSource

```
1     @Bean
2     public DataSource studentDataSource()
3     {
4       DataSourceBuilder dsb =
5           DataSourceBuilder.create();
6       dsb.url("jdbc:derby:c:\springclass\db\StudentDB");
7       dsb.driverClassName(
8          "org.apache.derby.jdbc.EmbeddedDriver");
9       dsb.username("SA");
10      dsb.password("SA");
11      return dsb.build();
12    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring a JNDI-Based DataSource (XML)

- You can use <jee:jndi-lookup> to create a data source via a JNDI lookup
- If your "client" defines a **resource-reference**, then you should specify **resource-ref="true"** - Spring will prepend *java:comp/env* to the provided JNDI name

```
<jee:jndi-lookup id="studentDataSource"
  jndi-name="studentDS" resource-ref="true">
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring JNDI (Java Config)

- With Java configuration, you can use the **JndiDataSourceLookup** to create and initialize a JNDI-based DataSource

```
1    @Bean
2    public DataSource studentDataSource()
3    {
4      JndiDataSourceLookup dsLookup =
5          new JndiDataSourceLookup();
6      dsLookup.setResourceRef(true);
7      DataSource dataSource = dsLookup.getDataSource(
8          "StudentDS");
9      return dataSource;
10   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Writing a DAO

- DAO classes normally implement a DAO interface, and use injection to obtain the data source

```
1    @Repository
2    public class StudentDAOJdbcImpl implements StudentDAO
3    {
4        private DataSource studentDataSource;
5        private JdbcTemplate template;
6
7        @Autowired
8        public void setDataSource(DataSource ds)
9        {
10           studentDataSource = ds;
11           template = new JdbcTemplate(ds);
12       }
13   . . .
14   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JdbcTemplate Query Methods

- The JdbcTemplate provides numerous ways to query for various types

```
<T> List<T> query(String sql, RowMapper<T> rm, Map<String,?> args)
<T> List<T> query(String sql, RowMapper<T> rm, Object... args)
<T> List<T> query(String sql, RowMapper<T> rm, SqlParameterSource args)
int queryForInt(String sql, Map<String,?> args)
int queryForInt(String sql, Object... args)
int queryForInt(String sql, SqlParameterSource args)
List<Map<String,Object>> queryForList(String sql, Map<String,?> args)
List<Map<String,Object>> queryForList(String sql, Object... args)
List<Map<String,Object>> queryForList(String sql, SqlParameterSource args)
long queryForLong(String sql, Map<String,?> args)
long queryForLong(String sql, Object... args)
long queryForLong(String sql, SqlParameterSource args)
Map<String,Object> queryForMap(String sql, Map<String,?> args)
Map<String,Object> queryForMap(String sql, Object... args)
Map<String,Object> queryForMap(String sql, SqlParameterSource args)
<T> T queryForObject(String sql, Class<T> requiredType, Map<String,?> args)
<T> T queryForObject(String sql, Class<T> requiredType, Object... args)
<T> T queryForObject(String sql, Class<T> requiredType, SqlParameterSource args)
<T> T queryForObject(String sql, RowMapper<T> rm, Map<String,?> args)
<T> T queryForObject(String sql, RowMapper<T> rm, Object... args)
<T> T queryForObject(String sql, RowMapper<T> rm, SqlParameterSource args)
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Example: Querying Row Count

- A common DAO operation is to return the count of rows in a table

```
1    @Repository
2    public class StudentDAOJdbcImpl implements StudentDAO
3    {
4        // initialized in setDataSource()
5        private JdbcTemplate template;
6
7    . . .
8
9      public int getStudentCount()
10     {
11       return template.queryForObject(
12          "SELECT COUNT(*) FROM student", Integer.class);
13     }
14   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Mapping Rows to Objects

- Spring defines **row mappers**, which are interfaces you can implement so a template can convert a result-set row to an object

```
1     private class StudentRowMapper implements
2       RowMapper<Student>  {
3       public Student mapRow(ResultSet rs, int rowNum)
4        throws SQLException  {
5           Student s = new Student();
6           s.setGpa(rs.getDouble("gpa"));
7           s.setName(rs.getString("name"));
8           s.setStudentID(rs.getInt("studentID"));
9           return s;
10       }
11    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Example: Query All

```
1    public class StudentDAOJdbcImpl
2      implements StudentDAO
3    {
4        // initialized in setDataSource()
5        private JdbcTemplate template;
6    . . .
7        private class StudentRowMapper
8           implements RowMapper<Student>
9        { . . . // shown earlier }
10
11       public Collection<Student> findAllStudents()
12       {
13           List<Student> students = template
14              .query("SELECT * FROM student", mapper);
15           return students;
16       }
17   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Example: Find Student By ID

```
1    public class StudentDAOJdbcImpl
2      implements StudentDAO
3    {
4
5    . . .
6
7        public Student findStudentByID(int studentID)
8        {
9            return template.queryForObject(
10             "SELECT * FROM student WHERE studentID=?",
11              mapper, studentID);
12       }
13   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JdbcTemplate Insert, Update and Delete

```
int update(String sql, Map<String,?> args)
int update(String sql, Object... args)
int update(String sql, SqlParameterSource args)

int[] batchUpdate(String sql, List<Object[]> batchArgs)
int[] batchUpdate(String sql, List<Object[]> batchArgs, int[] argTypes)
int[] batchUpdate(String sql, Map<String,?>[] batchValues)
int[] batchUpdate(String sql, SqlParameterSource[] batchArgs)
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Example: Insert

```
1    public class StudentDAOJdbcImpl
2      implements StudentDAO
3    {
4        // initialized in setDataSource()
5        private JdbcTemplate template;
6
7    . . .
8
9        public int insertStudent(Student s)
10       {
11         return template.update(
12           "INSERT INTO student
13           (studentId,name,gpa) VALUES(?,?,?)",
14           s.getStudentID(), s.getName(), s.getGpa());
15       }
16   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Complete DAO Example

Student.java

StudentDAO.java

StudentDAOJdbcImpl.java

DisplayStudents.java

FindStudent.java

InsertStudent.java

spring.xml

web.xml

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```java
package university;

public class Student
{
    private int studentID;
    private String name;
    private double gpa;

    public double getGpa()
    {
        return gpa;
    }

    public void setGpa(double gpa)
    {
        this.gpa = gpa;
    }

    public String getName()
    {
        return name;
    }

    public void setName(String name)
    {
        this.name = name;
    }

    public int getStudentID()
    {
        return studentID;
    }

    public void setStudentID(int studentID)
    {
        this.studentID = studentID;
    }
}
```

```java
package dao;

import java.util.Collection;

import university.Student;

public interface StudentDAO
{
    public int getStudentCount();
    public Collection<Student> findAllStudents();
    public Student findStudentByID(int studentID);
    public int insertStudent(Student s);
    public int insertStudentReturnId(Student s);
}
```

```
1   package dao;
2
3   import java.sql.ResultSet;
4   import java.sql.SQLException;
5   import java.util.Collection;
6   import java.util.HashMap;
7   import java.util.List;
8   import java.util.Map;
9
10  import javax.sql.DataSource;
11
12  import org.springframework.beans.factory.annotation.Autowired;
13  import org.springframework.jdbc.core.JdbcTemplate;
14  import org.springframework.jdbc.core.RowMapper;
15  import org.springframework.jdbc.core.simple.SimpleJdbcInsert;
16  import org.springframework.stereotype.Repository;
17
18  import university.Student;
19
20  @Repository("studentDAO")
21  public class StudentDAOJdbcImpl implements StudentDAO
22  {
23      private DataSource studentDataSource;
24      private JdbcTemplate template;
25      private StudentRowMapper mapper;
26
27      @Autowired
28      public void setDataSource(DataSource ds)
29      {
30          studentDataSource = ds;
31          template = new JdbcTemplate(ds);
32          mapper = new StudentRowMapper();
33      }
34
35      private class StudentRowMapper implements RowMapper<Student>
36      {
37          public Student mapRow(ResultSet rs, int rowNum) throws SQLException
38          {
39              Student s = new Student();
40              s.setGpa(rs.getDouble("gpa"));
41              s.setName(rs.getString("name"));
42              s.setStudentID(rs.getInt("studentID"));
43              return s;
44          }
```

```
46
47      public int getStudentCount()
48      {
49          return template.queryForObject("SELECT COUNT(*) FROM student",
50                  Integer.class);
51      }
52
53      public Collection<Student> findAllStudents()
54      {
55          List<Student> students = template.query("SELECT * FROM student",
56                  mapper);
57          return students;
58      }
59
60      public Student findStudentByID(int studentID)
61      {
62          Student s = template.queryForObject(
63                  "SELECT * FROM student WHERE studentID=?",
64                  mapper, studentID);
65          return s;
66      }
67
68      public int insertStudent(Student s)
69      {
70          return template.update(
71                 "INSERT INTO student (name,gpa) VALUES(?,?)",
72                  s.getName(), s.getGpa());
73      }
74
75      // This insertStudent version uses the SimpleJdbcInsert
76      // class to retrieve an autogenerated key
77      public int insertStudentReturnId(Student s)
78      {
79          SimpleJdbcInsert inserter = new SimpleJdbcInsert(studentDataSource);
80          inserter.setTableName("student");
81          inserter.usingGeneratedKeyColumns("StudentID");
82          Map<String, Object> parms = new HashMap<String, Object>();
83          parms.put("name", s.getName());
84          parms.put("gpa", s.getGpa());
85
86          Number id = inserter.executeAndReturnKey(parms);
87          return id.intValue();
88      }
89  }
```

```
1   package servlets;
2
3   import java.io.IOException;
4   import java.io.PrintWriter;
5   import java.util.Collection;
6
7   import javax.servlet.ServletContext;
8   import javax.servlet.ServletException;
9   import javax.servlet.annotation.WebServlet;
10  import javax.servlet.http.HttpServletRequest;
11  import javax.servlet.http.HttpServletResponse;
12
13  import org.springframework.web.context.WebApplicationContext;
14  import org.springframework.web.context.support.WebApplicationContextUtils;
15
16  import university.Student;
17  import dao.StudentDAO;
18
19
20  @SuppressWarnings("serial")
21  @WebServlet("/displayStudents")
22  public class DisplayStudents extends javax.servlet.http.HttpServlet
23  {
24      @Override
25      protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOExceptio
26      {
27          response.setContentType("text/html");
28          PrintWriter out = response.getWriter();
29
30          ServletContext servletContext = getServletContext();
31          WebApplicationContext ctx =
32              WebApplicationContextUtils.getRequiredWebApplicationContext(
33                  servletContext);
34
35          try
36          {
37              StudentDAO dao = (StudentDAO)ctx.getBean("studentDAO");
38              out.println("There are " + dao.getStudentCount() + " students");
39
40              Collection<Student> students = dao.findAllStudents();
41              out.println("<table border='1'>");
42              for(Student s : students)
43              {
44                  out.println("<tr>");
```

```
46                  out.println("<td>" + s.getName() + "</td>");
47                  out.println("<td>" + s.getGpa() + "</td>");
48                  out.println("</tr>");
49              }
50              out.println("</table>");
51          }
52          catch (Exception e)
53          {
54              out.println("Something went wrong: " + e);
55          }
56
57  //        Connection con = null;
58  //        try
59  //        {
60  //            Context ctx = new InitialContext();
61  //            DataSource ds =
62  //                (DataSource)ctx.lookup("java:comp/env/studentDS");
63  //            con = ds.getConnection();
64  //            Statement st = con.createStatement();
65  //            ResultSet rs = st.executeQuery("SELECT COUNT(*) FROM student");
66  //            rs.next();
67  //            out.println("There are " + rs.getInt(1) + " students");
68  //        }
69  //        catch (NamingException e)
70  //        {
71  //            e.printStackTrace();
72  //        }
73  //        catch (SQLException e)
74  //        {
75  //            e.printStackTrace();
76  //        }
77  //        finally
78  //        {
79  //            try
80  //            {
81  //               if (con != null) con.close();
82  //            } catch (Exception e1) {}
83  //        }
84
85
86      }
87  }
```

```
1   package servlets;
2
3   import java.io.IOException;
4   import java.io.PrintWriter;
5
6   import javax.servlet.ServletContext;
7   import javax.servlet.ServletException;
8   import javax.servlet.annotation.WebServlet;
9   import javax.servlet.http.HttpServletRequest;
10  import javax.servlet.http.HttpServletResponse;
11
12  import org.springframework.dao.EmptyResultDataAccessException;
13  import org.springframework.web.context.WebApplicationContext;
14  import org.springframework.web.context.support.WebApplicationContextUtils;
15
16  import university.Student;
17  import dao.StudentDAO;
18
19  @SuppressWarnings("serial")
20  @WebServlet("/findStudent")
21  public class FindStudent extends javax.servlet.http.HttpServlet
22  {
23      protected void doGet(HttpServletRequest request,
24              HttpServletResponse response) throws ServletException, IOException
25      {
26          response.setContentType("text/html");
27          PrintWriter out = response.getWriter();
28
29          ServletContext servletContext = getServletContext();
30          WebApplicationContext ctx = WebApplicationContextUtils
31                  .getRequiredWebApplicationContext(servletContext);
32
33          try
34          {
35              int studentID = Integer.parseInt(request.getParameter("studentID"));
36              StudentDAO dao = (StudentDAO) ctx.getBean("studentDAO");
37              Student s = dao.findStudentByID(studentID);
38              if (s != null)
39              {
40                  out.println("<p>Student ID: " + s.getStudentID() + "</p>");
41                  out.println("<p>Name: " + s.getName() + "</p>");
42                  out.println("<p>GPA: " + s.getGpa() + "</p>");
43              }
44          }
```

```
46          {
47              out.println("Invalid number entered. Press 'Back' and try again.");
48          }
49          catch (EmptyResultDataAccessException e)
50          {
51              out.println("Student not found");
52          }
53          catch (Exception e)
54          {
55              out.println("Something went wrong: " + e);
56          }
57      }
58  }
```

```
1   package servlets;
2
3   import java.io.IOException;
4   import java.io.PrintWriter;
5
6   import javax.servlet.ServletContext;
7   import javax.servlet.ServletException;
8   import javax.servlet.annotation.WebServlet;
9   import javax.servlet.http.HttpServletRequest;
10  import javax.servlet.http.HttpServletResponse;
11
12  import org.springframework.web.context.WebApplicationContext;
13  import org.springframework.web.context.support.WebApplicationContextUtils;
14
15  import university.Student;
16  import dao.StudentDAO;
17
18  @SuppressWarnings("serial")
19  @WebServlet("/insertStudent")
20  public class InsertStudent extends javax.servlet.http.HttpServlet
21  {
22      protected void doPost(HttpServletRequest request,
23              HttpServletResponse response) throws ServletException, IOException
24      {
25          response.setContentType("text/html");
26          PrintWriter out = response.getWriter();
27
28          ServletContext servletContext = getServletContext();
29          WebApplicationContext ctx = WebApplicationContextUtils
30                  .getRequiredWebApplicationContext(servletContext);
31
32          String name = request.getParameter("name");
33          String s = request.getParameter("gpa");
34
35          double gpa = Double.parseDouble(s);
36
37          try
38          {
39              StudentDAO dao = (StudentDAO) ctx.getBean("studentDAO");
40
41              Student student = new Student();
42              student.setGpa(gpa);
43              student.setName(name);
44              int rows = dao.insertStudent(student);
```

```
46          }
47          catch (Exception e)
48          {
49              out.println("Something went wrong: " + e);
50          }
51      }
52  }
```

```xml
<?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:aop="http://www.springframework.org/schema/aop"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:jee="http://www.springframework.org/schema/jee"
    xmlns:jms="http://www.springframework.org/schema/jms"
    xmlns:lang="http://www.springframework.org/schema/lang"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-4.1.xsd
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-4.1.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-4.1.xsd
        http://www.springframework.org/schema/jee
        http://www.springframework.org/schema/jee/spring-jee-4.1.xsd
        http://www.springframework.org/schema/lang
        http://www.springframework.org/schema/lang/spring-lang-4.1.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-4.1.xsd
        http://www.springframework.org/schema/util
        http://www.springframework.org/schema/util/spring-util-4.1.xsd">

    <context:component-scan base-package="dao" />

    <bean id="studentDataSource"
        class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="org.hsqldb.jdbcDriver" />
        <property name="url" value="jdbc:hsqldb:hsql://localhost/" />
        <property name="username" value="sa" />
        <property name="password" value="" />
    </bean>

</beans>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="ht
  <display-name>SpringJdbcTemplateWeb</display-name>
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring.xml</param-value>
    </context-param>
    <listener>
        <display-name>ContextLoaderListener</display-name>
        <listener-class>
            org.springframework.web.context.ContextLoaderListener
        </listener-class>
    </listener>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

Chapter Summary

In this chapter, you learned:

- The basics of Spring's JDBC support
- How to use SimpleJdbcTemplate and SimpleJdbcDaoSupport

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Web Services

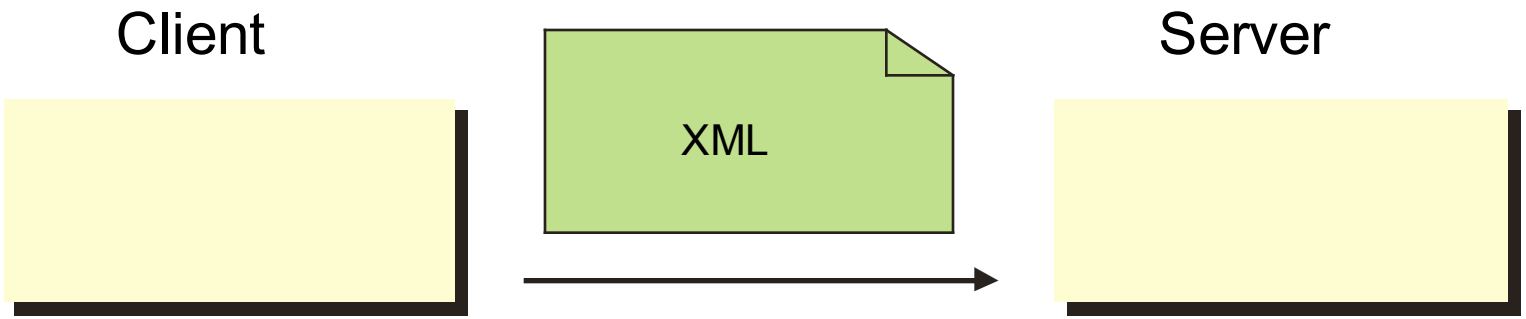
- What are Web Services?
- SOAP
- WSDL
- UDDI

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What are Web Services?

- Web services let you create **interoperable** distributed applications that use XML



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Benefits of Web Services

- Interoperability between heterogeneous systems
- Loosely-coupled applications that use standard tools, formats and protocols
- Wide industry support for runtimes and tools

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

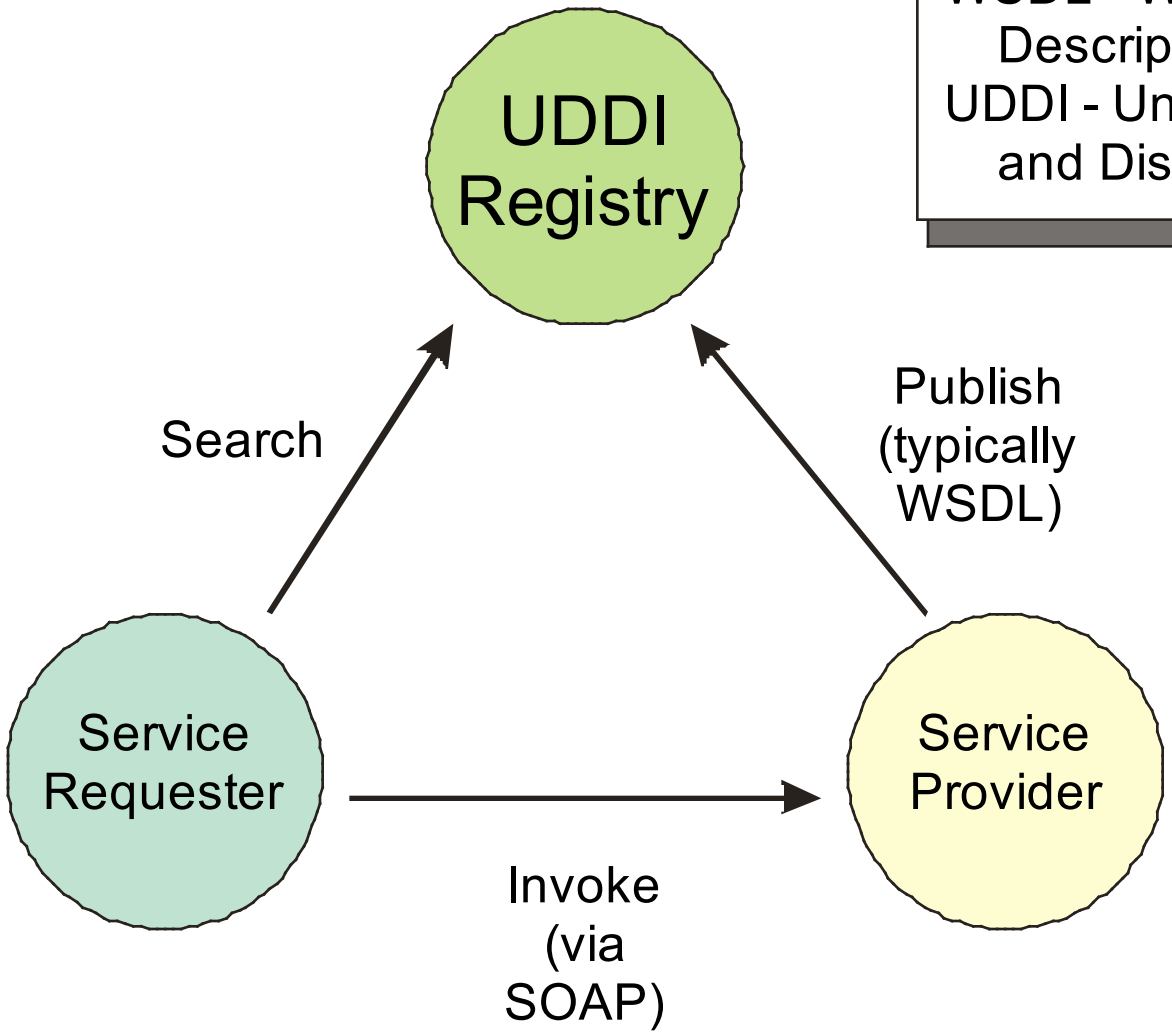
Web Services Challenges

- Interoperability can still be difficult
- Evolving standards (e.g. SOAP 1.1 to SOAP 1.2)
- Complexity due to many WS-* standards (e.g. WS-Security, WS-Addressing, WS-Policy, etc.)

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Service Oriented Architectures

SOAP
WSDL - Web Services Description Language
UDDI - Universal Description and Discovery Interface



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is SOAP?

- SOAP is a low-level XML-based protocol for exchanging messages
- SOAP is transport-neutral, but many implementations use HTTP
- SOAP is administered by the World Wide Web Consortium (http://www.w3.org)

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Example SOAP Message

- This example uses the SOAP 1.1 specification, which is currently in most wide use

```
1    <soapenv:Envelope
2       xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
3       xmlns:q0="http://ws.cdyne.com/WeatherWS/">
4      <soapenv:Body>
5        <q0:GetCityWeatherByZIP>
6          <q0:ZIP>52403</q0:ZIP>
7        </q0:GetCityWeatherByZIP>
8      </soapenv:Body>
9    </soapenv:Envelope>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Example RPC Response

- This example shows the SOAP 1.1 response to the **GetCityWeatherByZIP** message

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetCityWeatherByZIPResponse xmlns="http://ws.cdyne.com/WeatherWS/">
      <GetCityWeatherByZIPResult>
        <Success>true</Success>
        <ResponseText>City Found</ResponseText>
        <State>IA</State>
        <City>Cedar Rapids</City>
        <WeatherStationCity>Cedar Rapids</WeatherStationCity>
        <WeatherID>9</WeatherID>
        <Description>Partly Sunny</Description>
        <Temperature>1</Temperature>
        <RelativeHumidity>85</RelativeHumidity>
        <Wind>NW15</Wind>
        <Pressure>30.39F</Pressure>
      </GetCityWeatherByZIPResult>
    </GetCityWeatherByZIPResponse>
  </soap:Body>
</soap:Envelope>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is WSDL?

- The Web Services Description Language is an XML vocabulary for describing the interface to a web service
- WSDL is currently at version 2, but WSDL 1.1 is the most widely used version
- Like SOAP, WSDL is administered by the W3C
- The WSDL for a web service should contain enough detail so a client can issue a request to the web service

weather.wsdl

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
<wsdl:definitions xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" xr
  <wsdl:types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://ws.cdyne.com/WeatherWS/">
      <s:element name="GetWeatherInformation">
        <s:complexType />
      </s:element>
      <s:element name="GetWeatherInformationResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="GetWeatherInformationResult" type="tns:ArrayOfWeatherDescript
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:complexType name="ArrayOfWeatherDescription">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="unbounded" name="WeatherDescription" type="tns:WeatherDescription" />
        </s:sequence>
      </s:complexType>
      <s:complexType name="WeatherDescription">
        <s:sequence>
          <s:element minOccurs="1" maxOccurs="1" name="WeatherID" type="s:short" />
          <s:element minOccurs="0" maxOccurs="1" name="Description" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="PictureURL" type="s:string" />
        </s:sequence>
      </s:complexType>
      <s:element name="GetCityForecastByZIP">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="ZIP" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="GetCityForecastByZIPResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="GetCityForecastByZIPResult" type="tns:ForecastReturn" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:complexType name="ForecastReturn">
        <s:sequence>
          <s:element minOccurs="1" maxOccurs="1" name="Success" type="s:boolean" />
          <s:element minOccurs="0" maxOccurs="1" name="ResponseText" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="State" type="s:string" />
```

```
46            <s:element minOccurs="0" maxOccurs="1" name="WeatherStationCity" type="s:string" />
47            <s:element minOccurs="0" maxOccurs="1" name="ForecastResult" type="tns:ArrayOfForecast" />
48          </s:sequence>
49        </s:complexType>
50        <s:complexType name="ArrayOfForecast">
51          <s:sequence>
52            <s:element minOccurs="0" maxOccurs="unbounded" name="Forecast" nillable="true" type="tns:Forecast" />
53          </s:sequence>
54        </s:complexType>
55        <s:complexType name="Forecast">
56          <s:sequence>
57            <s:element minOccurs="1" maxOccurs="1" name="Date" type="s:dateTime" />
58            <s:element minOccurs="1" maxOccurs="1" name="WeatherID" type="s:short" />
59            <s:element minOccurs="0" maxOccurs="1" name="Desciption" type="s:string" />
60            <s:element minOccurs="1" maxOccurs="1" name="Temperatures" type="tns:temp" />
61            <s:element minOccurs="1" maxOccurs="1" name="ProbabilityOfPrecipiation" type="tns:POP" />
62          </s:sequence>
63        </s:complexType>
64        <s:complexType name="temp">
65          <s:sequence>
66            <s:element minOccurs="0" maxOccurs="1" name="MorningLow" type="s:string" />
67            <s:element minOccurs="0" maxOccurs="1" name="DaytimeHigh" type="s:string" />
68          </s:sequence>
69        </s:complexType>
70        <s:complexType name="POP">
71          <s:sequence>
72            <s:element minOccurs="0" maxOccurs="1" name="Nighttime" type="s:string" />
73            <s:element minOccurs="0" maxOccurs="1" name="Daytime" type="s:string" />
74          </s:sequence>
75        </s:complexType>
76        <s:element name="GetCityWeatherByZIP">
77          <s:complexType>
78            <s:sequence>
79              <s:element minOccurs="0" maxOccurs="1" name="ZIP" type="s:string" />
80            </s:sequence>
81          </s:complexType>
82        </s:element>
83        <s:element name="GetCityWeatherByZIPResponse">
84          <s:complexType>
85            <s:sequence>
86              <s:element minOccurs="1" maxOccurs="1" name="GetCityWeatherByZIPResult" type="tns:WeatherReturn" />
87            </s:sequence>
88          </s:complexType>
89        </s:element>
```

```
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="Success" type="s:boolean" />
            <s:element minOccurs="0" maxOccurs="1" name="ResponseText" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="State" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="City" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="WeatherStationCity" type="s:string" />
            <s:element minOccurs="1" maxOccurs="1" name="WeatherID" type="s:short" />
            <s:element minOccurs="0" maxOccurs="1" name="Description" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="Temperature" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="RelativeHumidity" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="Wind" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="Pressure" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="Visibility" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="WindChill" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="Remarks" type="s:string" />
          </s:sequence>
        </s:complexType>
        <s:element name="ArrayOfWeatherDescription" nillable="true" type="tns:ArrayOfWeatherDescription" />
        <s:element name="ForecastReturn" nillable="true" type="tns:ForecastReturn" />
        <s:element name="WeatherReturn" type="tns:WeatherReturn" />
      </s:schema>
    </wsdl:types>
    <wsdl:message name="GetWeatherInformationSoapIn">
      <wsdl:part name="parameters" element="tns:GetWeatherInformation" />
    </wsdl:message>
    <wsdl:message name="GetWeatherInformationSoapOut">
      <wsdl:part name="parameters" element="tns:GetWeatherInformationResponse" />
    </wsdl:message>
    <wsdl:message name="GetCityForecastByZIPSoapIn">
      <wsdl:part name="parameters" element="tns:GetCityForecastByZIP" />
    </wsdl:message>
    <wsdl:message name="GetCityForecastByZIPSoapOut">
      <wsdl:part name="parameters" element="tns:GetCityForecastByZIPResponse" />
    </wsdl:message>
    <wsdl:message name="GetCityWeatherByZIPSoapIn">
      <wsdl:part name="parameters" element="tns:GetCityWeatherByZIP" />
    </wsdl:message>
    <wsdl:message name="GetCityWeatherByZIPSoapOut">
      <wsdl:part name="parameters" element="tns:GetCityWeatherByZIPResponse" />
    </wsdl:message>
    <wsdl:message name="GetWeatherInformationHttpGetIn" />
    <wsdl:message name="GetWeatherInformationHttpGetOut">
      <wsdl:part name="Body" element="tns:ArrayOfWeatherDescription" />
    </wsdl:message>
```

```
136       <wsdl:part name="ZIP" type="s:string" />
137     </wsdl:message>
138     <wsdl:message name="GetCityForecastByZIPHttpGetOut">
139       <wsdl:part name="Body" element="tns:ForecastReturn" />
140     </wsdl:message>
141     <wsdl:message name="GetCityWeatherByZIPHttpGetIn">
142       <wsdl:part name="ZIP" type="s:string" />
143     </wsdl:message>
144     <wsdl:message name="GetCityWeatherByZIPHttpGetOut">
145       <wsdl:part name="Body" element="tns:WeatherReturn" />
146     </wsdl:message>
147     <wsdl:message name="GetWeatherInformationHttpPostIn" />
148     <wsdl:message name="GetWeatherInformationHttpPostOut">
149       <wsdl:part name="Body" element="tns:ArrayOfWeatherDescription" />
150     </wsdl:message>
151     <wsdl:message name="GetCityForecastByZIPHttpPostIn">
152       <wsdl:part name="ZIP" type="s:string" />
153     </wsdl:message>
154     <wsdl:message name="GetCityForecastByZIPHttpPostOut">
155       <wsdl:part name="Body" element="tns:ForecastReturn" />
156     </wsdl:message>
157     <wsdl:message name="GetCityWeatherByZIPHttpPostIn">
158       <wsdl:part name="ZIP" type="s:string" />
159     </wsdl:message>
160     <wsdl:message name="GetCityWeatherByZIPHttpPostOut">
161       <wsdl:part name="Body" element="tns:WeatherReturn" />
162     </wsdl:message>
163     <wsdl:portType name="WeatherSoap">
164       <wsdl:operation name="GetWeatherInformation">
165         <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Gets Information for each WeatherID</wsdl:docu
166         <wsdl:input message="tns:GetWeatherInformationSoapIn" />
167         <wsdl:output message="tns:GetWeatherInformationSoapOut" />
168       </wsdl:operation>
169       <wsdl:operation name="GetCityForecastByZIP">
170         <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Allows you to get your City Forecast Over the
171         <wsdl:input message="tns:GetCityForecastByZIPSoapIn" />
172         <wsdl:output message="tns:GetCityForecastByZIPSoapOut" />
173       </wsdl:operation>
174       <wsdl:operation name="GetCityWeatherByZIP">
175         <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Allows you to get your City's Weather, which
176         <wsdl:input message="tns:GetCityWeatherByZIPSoapIn" />
177         <wsdl:output message="tns:GetCityWeatherByZIPSoapOut" />
178       </wsdl:operation>
179     </wsdl:portType>
```

```
    <wsdl:operation name="GetWeatherInformation">
      <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Gets Information for each WeatherID</wsdl:docu
      <wsdl:input message="tns:GetWeatherInformationHttpGetIn" />
      <wsdl:output message="tns:GetWeatherInformationHttpGetOut" />
    </wsdl:operation>
    <wsdl:operation name="GetCityForecastByZIP">
      <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Allows you to get your City Forecast Over the
      <wsdl:input message="tns:GetCityForecastByZIPHttpGetIn" />
      <wsdl:output message="tns:GetCityForecastByZIPHttpGetOut" />
    </wsdl:operation>
    <wsdl:operation name="GetCityWeatherByZIP">
      <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Allows you to get your City's Weather, which
      <wsdl:input message="tns:GetCityWeatherByZIPHttpGetIn" />
      <wsdl:output message="tns:GetCityWeatherByZIPHttpGetOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="WeatherHttpPost">
    <wsdl:operation name="GetWeatherInformation">
      <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Gets Information for each WeatherID</wsdl:docu
      <wsdl:input message="tns:GetWeatherInformationHttpPostIn" />
      <wsdl:output message="tns:GetWeatherInformationHttpPostOut" />
    </wsdl:operation>
    <wsdl:operation name="GetCityForecastByZIP">
      <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Allows you to get your City Forecast Over the
      <wsdl:input message="tns:GetCityForecastByZIPHttpPostIn" />
      <wsdl:output message="tns:GetCityForecastByZIPHttpPostOut" />
    </wsdl:operation>
    <wsdl:operation name="GetCityWeatherByZIP">
      <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Allows you to get your City's Weather, which
      <wsdl:input message="tns:GetCityWeatherByZIPHttpPostIn" />
      <wsdl:output message="tns:GetCityWeatherByZIPHttpPostOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="WeatherSoap" type="tns:WeatherSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="GetWeatherInformation">
      <soap:operation soapAction="http://ws.cdyne.com/WeatherWS/GetWeatherInformation" style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
```

```
      <soap:operation soapAction="http://ws.cdyne.com/WeatherWS/GetCityForecastByZIP" style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetCityWeatherByZIP">
      <soap:operation soapAction="http://ws.cdyne.com/WeatherWS/GetCityWeatherByZIP" style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="WeatherSoap12" type="tns:WeatherSoap">
    <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="GetWeatherInformation">
      <soap12:operation soapAction="http://ws.cdyne.com/WeatherWS/GetWeatherInformation" style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetCityForecastByZIP">
      <soap12:operation soapAction="http://ws.cdyne.com/WeatherWS/GetCityForecastByZIP" style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetCityWeatherByZIP">
      <soap12:operation soapAction="http://ws.cdyne.com/WeatherWS/GetCityWeatherByZIP" style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output>
```

```
271       </wsdl:output>
272     </wsdl:operation>
273   </wsdl:binding>
274   <wsdl:binding name="WeatherHttpGet" type="tns:WeatherHttpGet">
275     <http:binding verb="GET" />
276     <wsdl:operation name="GetWeatherInformation">
277       <http:operation location="/GetWeatherInformation" />
278       <wsdl:input>
279         <http:urlEncoded />
280       </wsdl:input>
281       <wsdl:output>
282         <mime:mimeXml part="Body" />
283       </wsdl:output>
284     </wsdl:operation>
285     <wsdl:operation name="GetCityForecastByZIP">
286       <http:operation location="/GetCityForecastByZIP" />
287       <wsdl:input>
288         <http:urlEncoded />
289       </wsdl:input>
290       <wsdl:output>
291         <mime:mimeXml part="Body" />
292       </wsdl:output>
293     </wsdl:operation>
294     <wsdl:operation name="GetCityWeatherByZIP">
295       <http:operation location="/GetCityWeatherByZIP" />
296       <wsdl:input>
297         <http:urlEncoded />
298       </wsdl:input>
299       <wsdl:output>
300         <mime:mimeXml part="Body" />
301       </wsdl:output>
302     </wsdl:operation>
303   </wsdl:binding>
304   <wsdl:binding name="WeatherHttpPost" type="tns:WeatherHttpPost">
305     <http:binding verb="POST" />
306     <wsdl:operation name="GetWeatherInformation">
307       <http:operation location="/GetWeatherInformation" />
308       <wsdl:input>
309         <mime:content type="application/x-www-form-urlencoded" />
310       </wsdl:input>
311       <wsdl:output>
312         <mime:mimeXml part="Body" />
313       </wsdl:output>
314     </wsdl:operation>
```

```
      <http:operation location="/GetCityForecastByZIP" />
      <wsdl:input>
        <mime:content type="application/x-www-form-urlencoded" />
      </wsdl:input>
      <wsdl:output>
        <mime:mimeXml part="Body" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetCityWeatherByZIP">
      <http:operation location="/GetCityWeatherByZIP" />
      <wsdl:input>
        <mime:content type="application/x-www-form-urlencoded" />
      </wsdl:input>
      <wsdl:output>
        <mime:mimeXml part="Body" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="Weather">
    <wsdl:port name="WeatherSoap" binding="tns:WeatherSoap">
      <soap:address location="http://ws.cdyne.com/WeatherWS/Weather.asmx" />
    </wsdl:port>
    <wsdl:port name="WeatherSoap12" binding="tns:WeatherSoap12">
      <soap12:address location="http://ws.cdyne.com/WeatherWS/Weather.asmx" />
    </wsdl:port>
    <wsdl:port name="WeatherHttpGet" binding="tns:WeatherHttpGet">
      <http:address location="http://ws.cdyne.com/WeatherWS/Weather.asmx" />
    </wsdl:port>
    <wsdl:port name="WeatherHttpPost" binding="tns:WeatherHttpPost">
      <http:address location="http://ws.cdyne.com/WeatherWS/Weather.asmx" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

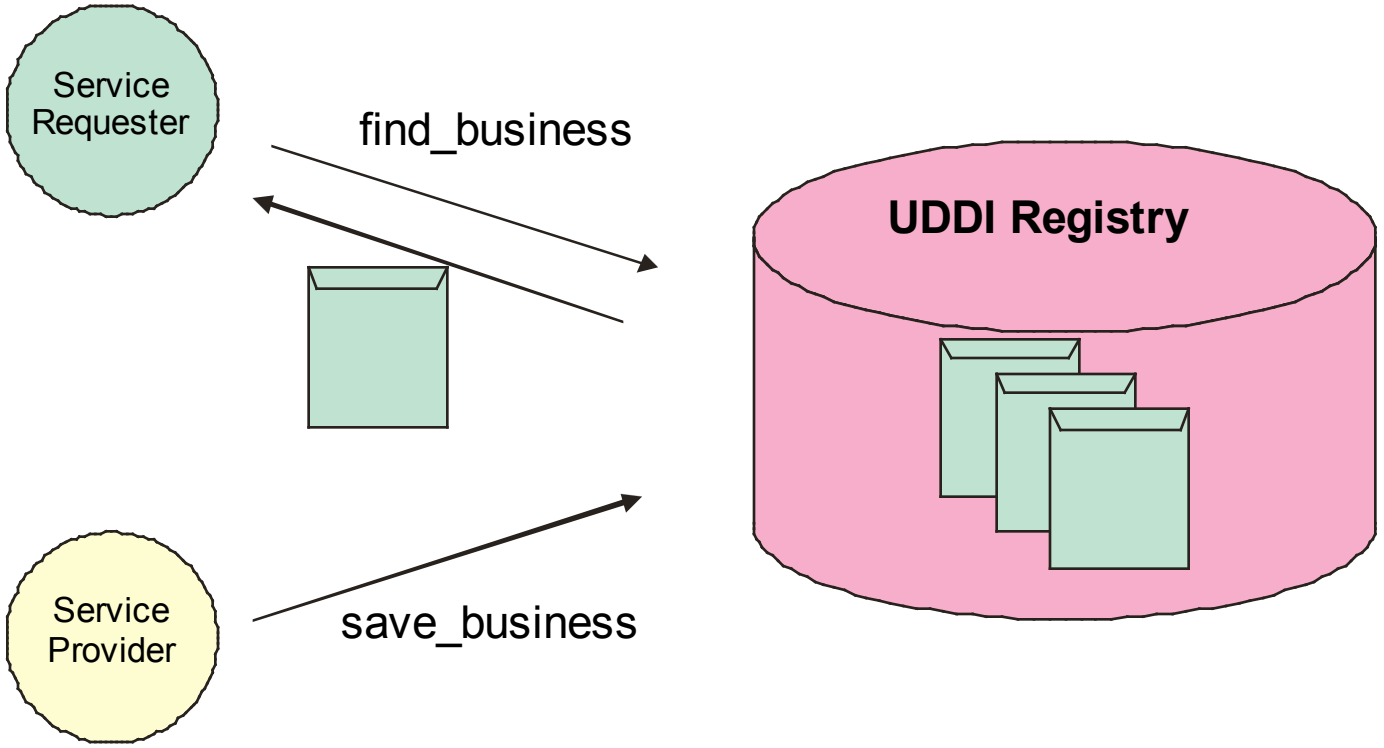
What is UDDI?

- The Universal Description and Discovery Interface defines a registry so that web service providers can publish their services and web service consumers can locate the services
- UDDI registries come in two flavors: the public UDDI Business Registry (UBR) and private registries
- UDDI is administered by the OASIS standards body

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using UDDI

- UDDI defines a set of web services that let providers publish their service and consumers search for services
- It's possible (and likely) that WSDL is stored in the registry, but it's not required



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Java APIs for Web Services

Sun has defined several APIs so that Java programs can work with web services:

- JAX-RPC - XML Remote Procedure Calls
- JAX-WS - The successor to JAX-RPC in JEE Version 5
- SAAJ - SOAP API for Attachments in Java
- JAX-M - XML Messaging
- JAX-R - XML Registries
- JAX-RS - The Java API for RESTful Web Services

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Toolkits for Web Services

There are numerous Web services programming toolkits for Java and other languages, including:

- Sun Glassfish (Java)
- Apache Axis (Java)
- IBM Rational Application Developer (RAD)
- BEA WebLogic Server and Workshop
- Spring Framework Web Services
- Microsoft Visual Studio.NET

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- The basics of web services and service-oriented architectures
- About SOAP, WSDL and UDDI

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to REST

- What is Representational State Transfer?
- Principles of REST
- Introduction to Java REST frameworks

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is REST?

- **Representational State Transfer** is an architecture for distributed systems that emphasizes statelessness
- REST was introduced by Roy Fielding, who was one of the authors of HTTP

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

REST vs SOAP

- REST provides a straightforward, simple architecture for providing highly scalable services (note that the Web itself is RESTful)
- SOAP provides a more complex but perhaps more enterprise-class architecture that includes additional standards such as WSDL, WS-Security and so forth

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Principles of REST

- Every resource should have a unique ID
- Resources can have multiple representations
- Use links to connect resources
- Use standard HTTP methods
- Clients and servers communicate statelessly

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Resource IDs

- The Internet provides a resource-identification technique known as a **Uniform Resource Identifier** (URI), a.k.a URL
- RESTful applications use URIs to assign unique IDs to application resources
- Examples:
    - http://www.mydomain.com/students/5847
    - http://www.mydomain.com/students?id=44
    - http://www.mydomain.com/id=17

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Resources With Multiple Representations

- For flexibility, applications that provide data can support different data formats, e.g. HTML, XML, CSV, JSON
- Clients can then request the resource in the format they like best, perhaps by specifying an HTTP **accept** header

```
//XML representation
<student>
  <ID>17</ID>
  <name>Sue Smith</student>
  <gpa>3.45</gpa>
</student>

//JSON representation
{"student":{"id":17, "name":"Sue Smith, "gpa":3.45}}
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Use Links to Connect Resources

- HTML supports **hyperlinking** so that resources can reference other resources
- You can also perform linking in other data formats such as XML

```
<student xmlns:xlink="http://www.w3.org/1999/xlink">
  <ID>17</ID>
  <name>Sue Smith</student>
  <gpa>3.45</gpa>
  <advisor
    xlink:href="http://www.mydomain.com/advisors/156"/>
</student>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Use Standard HTTP Methods

- HTTP supports **methods** that map to the four standard database Create, Retrieve, Update and Delete (CRUD) actions
- Note that only GET and POST can be sent directly from a browser so PUT, DELETE actions must be done programmatically (e.g. via AJAX)

HTTP Method	Database CRUD Operation
POST	CREATE
GET	SELECT (Retrieve)
PUT	UPDATE or CREATE
DELETE	DELETE

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Stateless Communication

- RESTful applications do **not** require the server to maintain communication state based on client requests
- This makes servers more scalable and allows the server to efficiently use techniques like caching, clustering and instance pooling

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

REST in Java

- JEE developers can use two basic approaches for writing RESTful services:
  - Write basic servlets
  - Use a REST framework
- Java RESTful frameworks include:
  - Restlets
  - JAX-WS REST
  - JAX-RS
  - Spring MVC REST

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is Restlet?

- Restlet is a lightweight, open-source REST framework for Java
- Restlet applications can run within a servlet container or as standalone Java applications
- Restlet was created by Jerome Louvel in 2005

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A Simple Restlet Resource

```
1    public class HelloServerResource extends ServerResource
2    {
3        @Get
4        public String represent()
5        {
6            return "hello, world";
7        }
8    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JAX-WS and REST

- JAX-WS is the standard Web services API for Java Enterprise Edition 5 and later
- JAX-WS provides support for creating Web services with no SOAP envelope, i.e. RESTful services

```
1    @WebServiceProvider(serviceName = "HWService")
2    @BindingType(value = HTTPBinding.HTTP_BINDING)
3    public class MyProvider implements Provider<Source>
4    {
5      public Source invoke(Source source)
6      {
7         String replyElement =
8            new String("<p>hello world</p>");
9         StreamSource reply = new StreamSource(
10           new StringReader(replyElement));
11        return reply;
12     }
13   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JAX-RS

- JAX-RS is an API for providing RESTful services in Java
- JAX-RS went through the Java Community Process as JSR-311 and is a part of JEE Version 6



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A JAX-RS Service

- To use JAX-RS, you create **resource** classes, using annotations to configure them

```
1    @Path("/helloworld")
2    public class HelloWorldResource
3    {
4        @GET
5        @Produces("text/plain")
6        public String getMessage(
7                @DefaultValue("")
8                @QueryParam("name") String name)
9        {
10           return "Hello World, " + name;
11       }
12   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring MVC REST

- The popular Spring Framework's MVC module includes support for RESTful services

```
1    @Controller
2    @RequestMapping("rest")
3    public class StudentController
4    {
5      @RequestMapping(value = "/student/{id}",
6        method = RequestMethod.GET)
7        public @ResponseBody
8         Student getStudent(@PathVariable("id") int id)
9      {
10       Student s = findStudentByID(id);
11
12       return s;
13     }
14   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- About the fundamentals of representational state transfer
- The basics of Java REST frameworks

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to JAX-RS

- Resource Classes
- Annotations
- Paths
- Context

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to JAX-RS

- JAX-RS is an API for providing RESTful services in Java
- JAX-RS went through the Java Community Process as JSR-311 and is a part of JEE Version 6



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JAX-RS Annotations

@Path specifies a relative URI for a resource class or method
@ApplicationPath specifies a relative URI for application class
@GET, @PUT, @POST, @DELETE and @HEAD specify the HTTP method
@Produces specifies response Media Type(s)
@Consumes specifies accepted request Media Type(s)
@PathParam binds a parameter based on URI
@QueryParam binds a parameter based on HTTP query parameter
@MatrixParam binds a parameter based on HTTP matrix parameter
@HeaderParam binds a parameter based on HTTP header
@CookieParam binds a parameter based on cookie
@FormParam binds a parameter based on form value
@DefaultValue specifies a default value
@Context binds a field or parameter to an environmental value

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Writing a Resource Class

- A **resource class** defines methods that respond to RESTful GET, POST or PUT methods, decorated with JAX-RS annotations
- Resource classes must provide a public constructor that JAX-RS can use, either a zero-argument constructor or a constructor with arguments annotated with @Context

```
1    @Path("/student")
2    public class RosterService
3    {
4      @GET
5      @Produces("text/xml")
6      public String studentAsXml(
7       @QueryParam("studentID") String id)
8      {
9       . . .
10     }
11   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Writing an Application Class

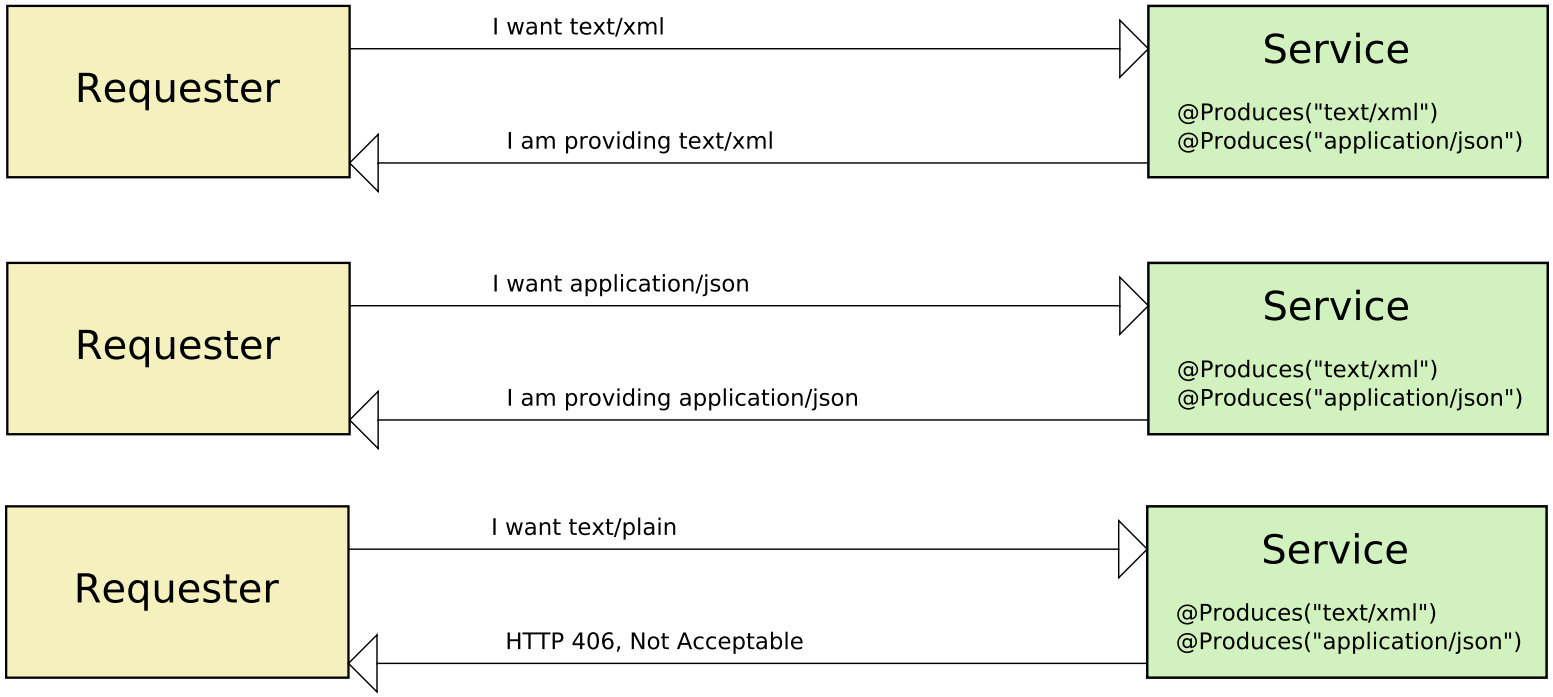
- An **application class** configures the resource classes (and providers) that make up a JAX-RS application
- Such classes extend **javax.ws.rs.core.Application** and typically override the **getClasses** method

```
1    @ApplicationPath("/roster/")
2    public class RosterApplication extends Application
3    {
4      public Set<Class<?>> getClasses()
5      {
6        Set<Class<?>> s = new HashSet<Class<?>>();
7        s.add(RosterService.class);
8        return s;
9      }
10   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Content Negotiation

- The client and server must agree on a data **representation**, which is usually a standard format such as text, XML or JSON using a **MediaType**
- Ultimately, the service decides on which representations it will provide



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Negotiation via HTTP Accept Header

- The HTTP specification allows for an **Accept** header, in which the user agent client provide a list of MIME types
- For GET requests, you use the **@Produces** annotation and the **@Consumes** annotation for POST and PUT requests
- JAX-RS will map requests to methods based on the Accept header and the annotations

```
curl –H "Accept: application/json, application/xml"
 http://localhost:8080/TestRESTWeb/rest/advisor/12

{"id":12,"name":"Sue Joens","salary":50000.0}
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Content Negotiation Example #1

```
1    // JavaScript Ajax requester
2
3    req.open("GET", url, true);
4    req.setRequestHeader("Accept","application/json");
5    req.onreadystatechange = callback;
6    req.send(null);
```

```
1    // JAX-RS resource class
2
3    @GET
4    @Produces("application/json")
5    public String studentAsJSON(
6       @QueryParam("studentID") String id)
7    {
8    . . .
9    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Content Negotiation Example #2

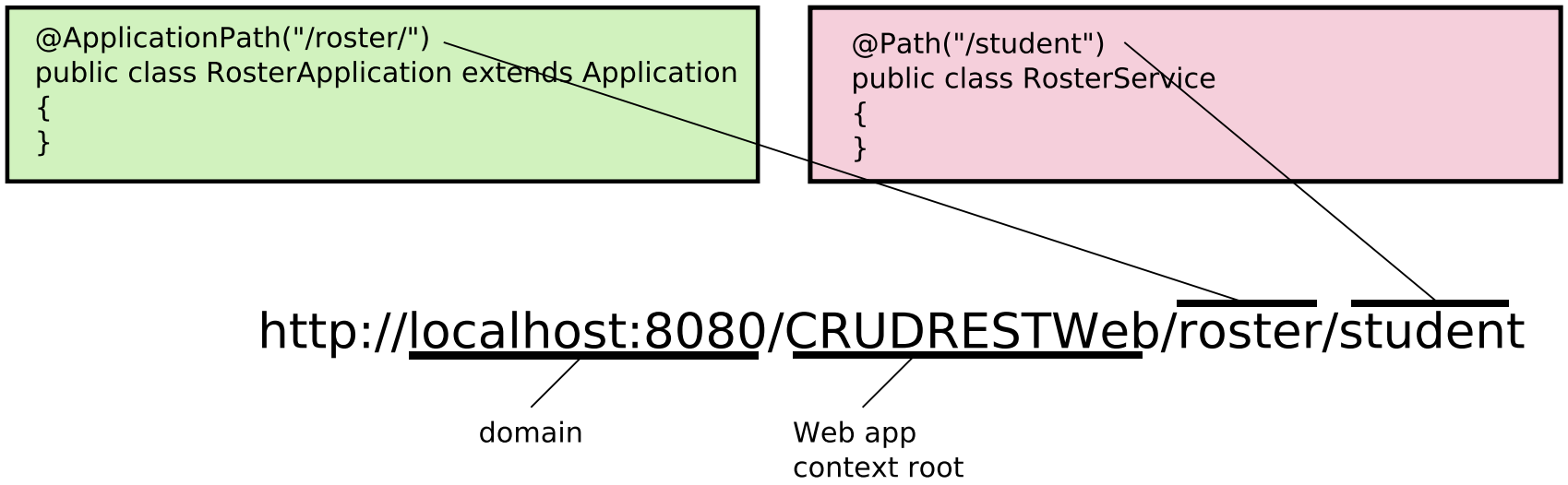
```
1    // JavaScript Ajax requester
2
3    req.open("POST", url, true);
4    req.setRequestHeader("Content-type","text/xml");
5    req.onreadystatechange = callback;
6    req.send(xml);
```

```
1    // JAX-RS resource class
2
3    @POST
4    @Consumes("text/xml")
5    public String createStudentFromXml(String xml)
6    {
7    . . .
8    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Assigning a URI Path

- To make methods available to respond to REST requests, you use the **Path** annotation to assign URIs
- You also can assign an overall application URI



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Accessing Request Parameters

- HTTP **GET** requests can contain parameters as part of the URL
- JAX-RS methods can bind to such parameters using the **@QueryParam** annotation

```
1    // JavaScript Ajax client
2
3    // url like http://...../roster/student?studentID=12
4    var url="roster/student?studentID=" + escape(studentID);
```

```
1    // JAX-RS resource
2    @GET
3    @Produces("text/xml, application/json")
4    public String studentAsXml(
5       @QueryParam("studentID") String id)
6    {
7     . . .
8    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Accessing URI Parameters

- Instead of, or in addition to, HTTP GET request parameters, RESTful applications can use parameters from the URI itself
- JAX-RS methods can bind to such parameters using the **@PathParam** annotation

```
1    // JAX-RS resource
2    @Path("/student/{studentId}")
3    public class StudentService
4    {
5        @GET
6        @Produces("text/xml, application/json")
7        public Response find(@PathParam("studentId")int id)
8        {
9            . . .
10       }
11   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The @Context Annotation

- You can use the **javax.ws.rs.core.Context** annotation to decorate fields or method parameters to **inject** context information of the following types:
    - **Application** - Reference to the Application subclass
    - **UriInfo** - Request URI information
    - **HttpHeaders** - HTTP headers
    - **Request** - Content negotiation information
    - **SecurityContext** - User principal, etc.
    - **ServletConfig**, **ServletContext**, **HttpServletRequest**, **HttpServletResponse** - Standard JEE objects

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

@Context Example

```
1    @Path("/student")
2    public class RosterService
3    {
4      @Context
5       HttpServletRequest request;
6
7      @GET
8      @Produces("text/xml, application/json")
9      public String studentAsXml(
10            @Context HttpHeaders headers)
11     {
12       System.out.println("User-agent: " +
13          headers.getRequestHeader("user-agent"));
14       HttpSession session = request.getSession();
15
16        . . .
17      }
18   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Mapping Java Objects to Content Types

- Many JAX-RS implementations, including Jersey, can automatically map Java objects to/from common types such as XML or JSON
- You must decorate the Java classes with **JAX-B** annotations

```
1    // JAX-RS resource
2    @GET
3    @Produces("text/xml, application/json")
4    public Response find(@PathParam("studentId")int id)
5    {
6        Student s = . . .;
7        return Response.ok(s).build();
8    }
9
10   @XmlRootElement
11   public class Student{. . .}
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Writing a JAX-RS Application

- Create one or more **resource** classes that produce or consume RESTful data and annotate them with JAX-RS annotations
- Create a class that extends **javax.ws.rs.core.Application** and overrides the **getClasses** method
- Configure JAX-RS libraries into Web application (if not using a full JEE application server)
- Deploy Web application
- Write client(s) that invoke the service

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Complete Example

Student.java

Roster.java

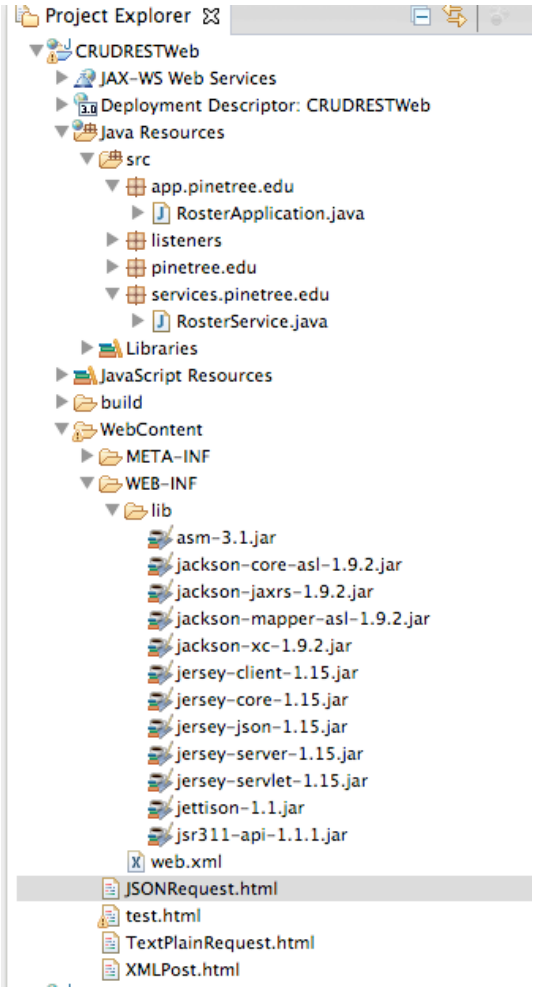
MyContextListener.java

RosterApplication.java

RosterService.java

GETRequest.html

XMLPost.html



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```java
package pinetree.edu;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class Student
{
    private int studentID;
    private String name;
    private double gpa;

    @Override
    public int hashCode()
    {
        final int prime = 31;
        int result = 1;
        result = prime * result + studentID;
        return result;
    }

    @Override
    public boolean equals(Object obj)
    {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Student other = (Student) obj;
        if (studentID != other.studentID)
            return false;
        return true;
    }

    public int getStudentID()
    {
        return studentID;
    }

    public void setStudentID(int studentID)
    {
        this.studentID = studentID;
    }
```

```
46      public String getName()
47      {
48          return name;
49      }
50
51      public void setName(String name)
52      {
53          this.name = name;
54      }
55
56      public double getGpa()
57      {
58          return gpa;
59      }
60
61      public void setGpa(double gpa)
62      {
63          this.gpa = gpa;
64      }
65
66      @Override
67      public String toString()
68      {
69          return "Student [studentID=" + studentID + ", name=" + name + ", gpa="
70                  + gpa + "]";
71      }
72
73  }
```

```
1    package pinetree.edu;
2
3    import java.util.ArrayList;
4    import java.util.List;
5
6    import javax.xml.bind.annotation.XmlRootElement;
7
8    @XmlRootElement
9    public class Roster
10   {
11       private List<Student> students = new ArrayList<Student>();
12
13       public List<Student> getStudents()
14       {
15           return students;
16       }
17
18       public void setStudents(List<Student> students)
19       {
20           this.students = students;
21       }
22   }
```

```
1   package listeners;
2
3   import javax.servlet.ServletContextEvent;
4   import javax.servlet.ServletContextListener;
5
6   import pinetree.edu.Roster;
7   import pinetree.edu.Student;
8
9   public class MyContextListener implements ServletContextListener
10  {
11      @Override
12      public void contextDestroyed(ServletContextEvent evt)
13      {
14      }
15
16      @Override
17      public void contextInitialized(ServletContextEvent evt)
18      {
19          Roster r = new Roster();
20          Student s = new Student();
21          s.setGpa(3.44);
22          s.setName("Sam B. Jones");
23          s.setStudentID(12);
24          r.getStudents().add(s);
25
26          s = new Student();
27          s.setGpa(3.45);
28          s.setName("Susan Q. Smith");
29          s.setStudentID(13);
30          r.getStudents().add(s);
31
32          s = new Student();
33          s.setGpa(2.98);
34          s.setName("Xiang Lu");
35          s.setStudentID(14);
36          r.getStudents().add(s);
37
38          System.out.println("Context initialized");
39
40          evt.getServletContext().setAttribute("roster", r);
41      }
42  }
```

```java
package app.pinetree.edu;

import java.util.HashSet;
import java.util.Set;

import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;

import services.pinetree.edu.RosterService;

@ApplicationPath("/roster/")
public class RosterApplication extends Application
{
    @Override
    public Set<Class<?>> getClasses()
    {
        Set<Class<?>> s = new HashSet<Class<?>>();
        s.add(RosterService.class);
        return s;
    }

}
```

```
1    package services.pinetree.edu;
2
3    import java.net.InetAddress;
4    import java.net.URI;
5
6    import javax.servlet.ServletContext;
7    import javax.servlet.http.HttpServletRequest;
8    import javax.ws.rs.Consumes;
9    import javax.ws.rs.GET;
10   import javax.ws.rs.HeaderParam;
11   import javax.ws.rs.POST;
12   import javax.ws.rs.Path;
13   import javax.ws.rs.Produces;
14   import javax.ws.rs.QueryParam;
15   import javax.ws.rs.core.Context;
16   import javax.ws.rs.core.Response;
17
18   import pinetree.edu.Roster;
19   import pinetree.edu.Student;
20
21   @Path("/student")
22   public class RosterService
23   {
24       @Context
25       ServletContext context;
26
27       @GET
28       @Produces("text/xml, application/json")
29       public Response getStudent(@QueryParam("studentID")
30          String id, @HeaderParam("Accept") String acceptHeader)
31       {
32           System.out.println("Received GET request for student: " +
33              id + " requested type: " + acceptHeader);
34           Response retVal = null;
35
36           retVal = findStudent(id);
37
38           return retVal;
39       }
40
41       private Response findStudent(String id)
42       {
43           Response retVal;
44           int studentID = Integer.parseInt(id);
```

```
46
47          Student found = null;
48          for (Student s : roster.getStudents())
49          {
50              if (s.getStudentID() == studentID)
51              {
52                  System.out.println("Found a student: " + s);
53                  found = s;
54                  break;
55              }
56          }
57
58          if (found == null)
59          {
60              System.out.println("No student found");
61              retVal =
62                Response.status(Response.Status.NOT_FOUND).build();
63          }
64          else
65          {
66              retVal = Response.ok(found).build();
67          }
68          return retVal;
69      }
70
71      @POST
72      @Consumes("text/xml")
73      public Response createStudentFromXml(Student s,
74              @Context final HttpServletRequest request)
75          throws Exception
76      {
77          Response retVal =
78            Response.status(Response.Status.BAD_REQUEST).build();
79
80          Roster roster = (Roster) context.getAttribute("roster");
81
82          if (s != null)
83          {
84              System.out.println("Received student XML " + s);
85              if (!roster.getStudents().contains(s))
86              {
87                  roster.getStudents().add(s);
88
89                  String newStudentURI = "http://" +
```

```
91                    ":" + request.getLocalPort() + request.getContextPath() +
92                    "/roster/student?studentID=" + s.getStudentID();
93
94                 System.out.println(
95                    "Created student resource with URI: " +
96                         newStudentURI);
97
98                 // the Response.created() method sets the HTTP
99                 // Location header to the given URI
100                retVal = Response.created(
101                   URI.create(newStudentURI)).build();
102             }
103         }
104
105         return retVal;
106     }
107 }
```

```
1    <html>
2      <head>
3        <title>XML POST</title>
4        <script>
5
6          function sendRequest()
7          {
8            var url = "roster/student";
9            sendAsyncPostRequest(url);
10         }
11
12         function callback()
13         {
14           if (req.readyState == 4)
15           {
16               if (req.status == 200)
17               {
18               var text = req.responseText;
19               alert(text);
20               }
21               else
22               {
23                   alert("Problem: " + req.status + ", " + req.statusText);
24               }
25           }
26         }
27
28         function sendAsyncPostRequest(url)
29         {
30           req = getXMLHttpRequest();
31           if (req)
32           {
33           var studentID = document.getElementById("studentid").value;
34           var name = document.getElementById("name").value;
35           var gpa = document.getElementById("gpa").value;
36           var xml = "<student><name>" + name + "</name>" +
37                                 "<studentID>" + studentID + "</studentID>" +
38                                 "<gpa>" + gpa + "</gpa>" +
39                     "</student>";
40
41             req.open("POST", url, true);
42             req.setRequestHeader("Content-type","text/xml");
43             req.onreadystatechange = callback;
44             req.send(xml);
```

```
46        }
47
48        function getXMLHttpRequest()
49        {
50          var xRequest=null;
51          if (window.XMLHttpRequest)
52          {
53              xRequest=new XMLHttpRequest();
54          }
55          else if (typeof ActiveXObject != "undefined")
56          {
57              xRequest=new ActiveXObject("Microsoft.XMLHTTP");
58          }
59          return xRequest;
60        }
61      </script>
62    </head>
63    <body>
64      <form>
65        <p>
66          Student ID:   <input type="text" name="studentID" id="studentid">
67        </p>
68        <p>
69          Name:   <input type="text" name="name" id="name">
70        </p>
71        <p>
72          GPA:   <input type="text" name="gpa" id="gpa">
73        </p>
74        <p>
75          <input type="button" value="Insert Student" onclick="sendRequest()">
76        </p>
77      </form>
78    </body>
79  </html>
```

```
<html>
  <head>
    <title>JSON Request</title>
    <script>

      function sendRequest()
      {
        var studentID =
            document.getElementById("studentid").value;
        var mediaType =
            document.getElementById("mediaType").value;
        var url = "roster/student?studentID=" +
            escape(studentID);
        sendAsyncGetRequest(url, mediaType);
      }

      function showResponse(text)
      {
        if (text.trim().length == 0)
          text = "No student with that ID";

        var resp = document.getElementById('output');

        resp.firstChild.nodeValue=text;
      }

      function callback()
      {
        if (req.readyState == 4)
        {
            if (req.status == 200)
            {
                var text = req.responseText;

                showResponse(text);
            }
            else
            {
                alert("Problem: " + req.status + ", " +
                   req.statusText);
            }
        }
      }

```

```
46        {
47          req = getXMLHttpRequest();
48          if (req)
49          {
50            req.open("GET", url, true);
51            req.setRequestHeader("Accept",mediaType);
52            req.onreadystatechange = callback;
53            req.send(null);
54          }
55        }
56
57        function getXMLHttpRequest()
58        {
59          var xRequest=null;
60          if (window.XMLHttpRequest)
61          {
62              xRequest=new XMLHttpRequest();
63          }
64          else if (typeof ActiveXObject != "undefined")
65          {
66              xRequest=new ActiveXObject("Microsoft.XMLHTTP");
67          }
68          return xRequest;
69        }
70      </script>
71    </head>
72    <body>
73      <h3>Service Response</h3>
74      <div id="output">Nothing yet</div>
75      <form>
76        <p>
77          Student ID:
78          <input type="text" name="studentID" id="studentid">
79        </p>
80        <p>
81          Media Type (text/xml or application/json):
82          <input type="text" name="mediaType" id="mediaType">
83        </p>
84        <p>
85          <input type="button" value="Send Request"
86             onclick="sendRequest()">
87        </p>
88      </form>
89    </body>
```

Chapter Summary

In this chapter, you learned:

- The fundamentals of the JAX-RS framework
- How to write **resource** classes and **application** classes
- How JAX-RS applications negotiate content representation types

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

RESTful CRUD Services

- Create
- Retrieve
- Update
- Delete

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Richardson Maturity Model

- According to the **Richardson Maturity Model**, which evaluates how RESTful an application is, apps that correctly implement CRUD are at **Level 2**

```
Level 0  Use a single verb, typically POST, and a single URI
         for all interactions. Most SOAP based services are
         like this

Level 1  Use a single verb, typically GET, but multiple URIs.
         These types of services often violate the idempotency
         of GET

Level 2  Use multiple verbs and multiple URIs. This includes
         CRUD RESTful apps

Level 3  Use Hypermedia as the Engine of Application State
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

RESTful CRUD Service Overview

- Many applications use the four **CRUD** operations: **Create**, **Retrieve**, **Update** and **Delete**
- The REST community has come up with guidelines for how to implement CRUD using standard HTTP facilities

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

HTTP Review

- HTTP provides several **methods** as well as standard **return codes** and **headers**

```
HTTP Methods
------------
GET, POST,
PUT, DELETE,
HEAD, PATCH,
OPTIONS, TRACE,
CONNECT


HTTP Headers
------------
Accept
Content-Type
Status
...
```

```
HTTP Status Codes
-----------------
200 OK
201 Created
204 No Content
400 Bad Request
404 Not Found
409 Conflict
500 Internal Server Error
...
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Media Types

- The IANA defines **media types (MIME)** that services can use to specify data types

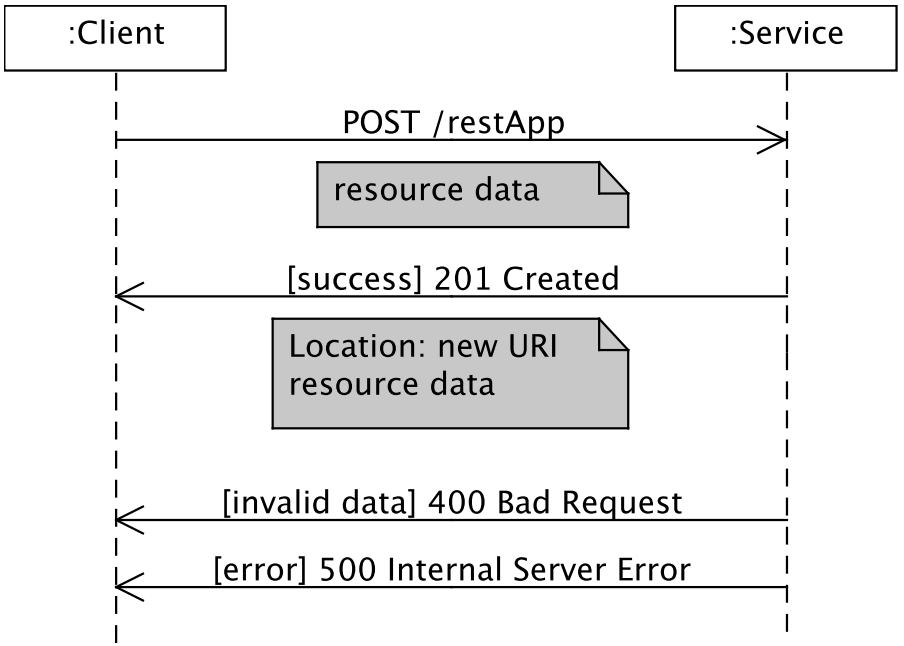
```
Common Media Types
------------------

application/json
text/plain
text/xml
application/vnd.*     -- vendor specific
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Implementing Create

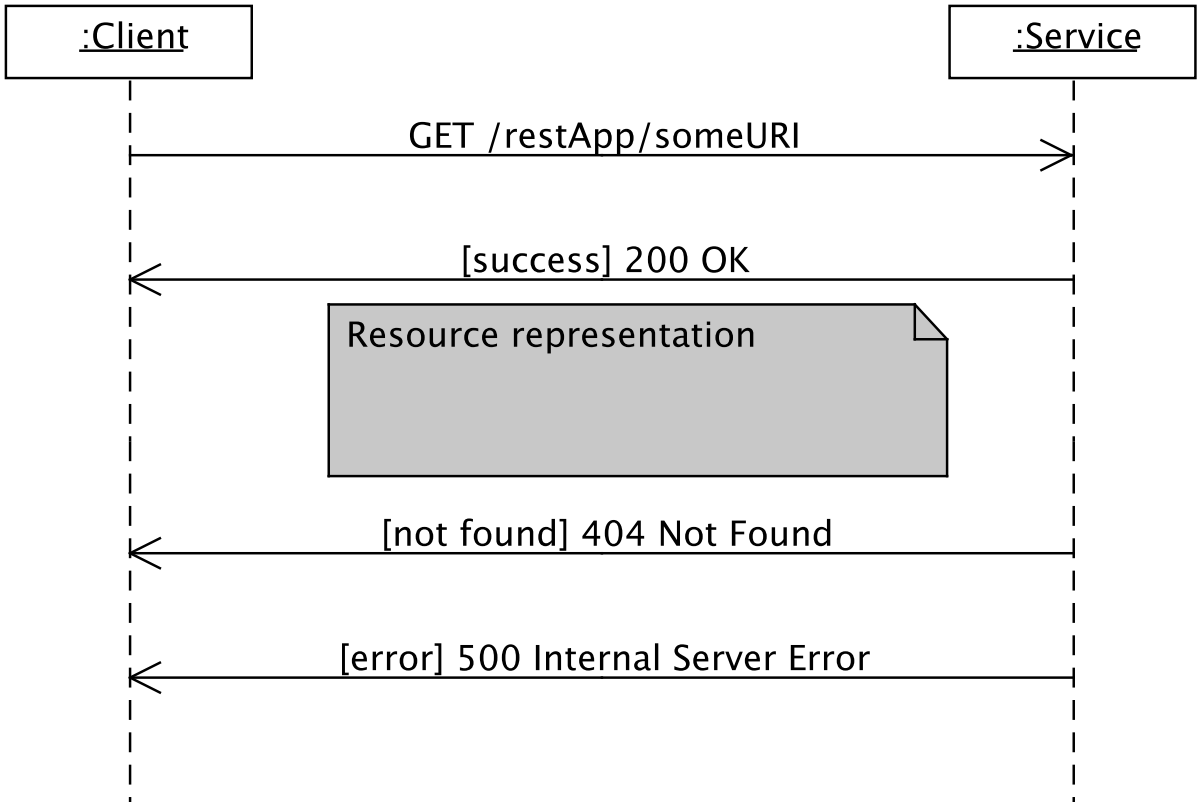
- Client should **POST** a resource representation and will receive a possibly modified resource representation
- Service returns response code **201 Created** on success or **400 Bad Request** or **500 Internal Server Error**



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Implementing Retrieve

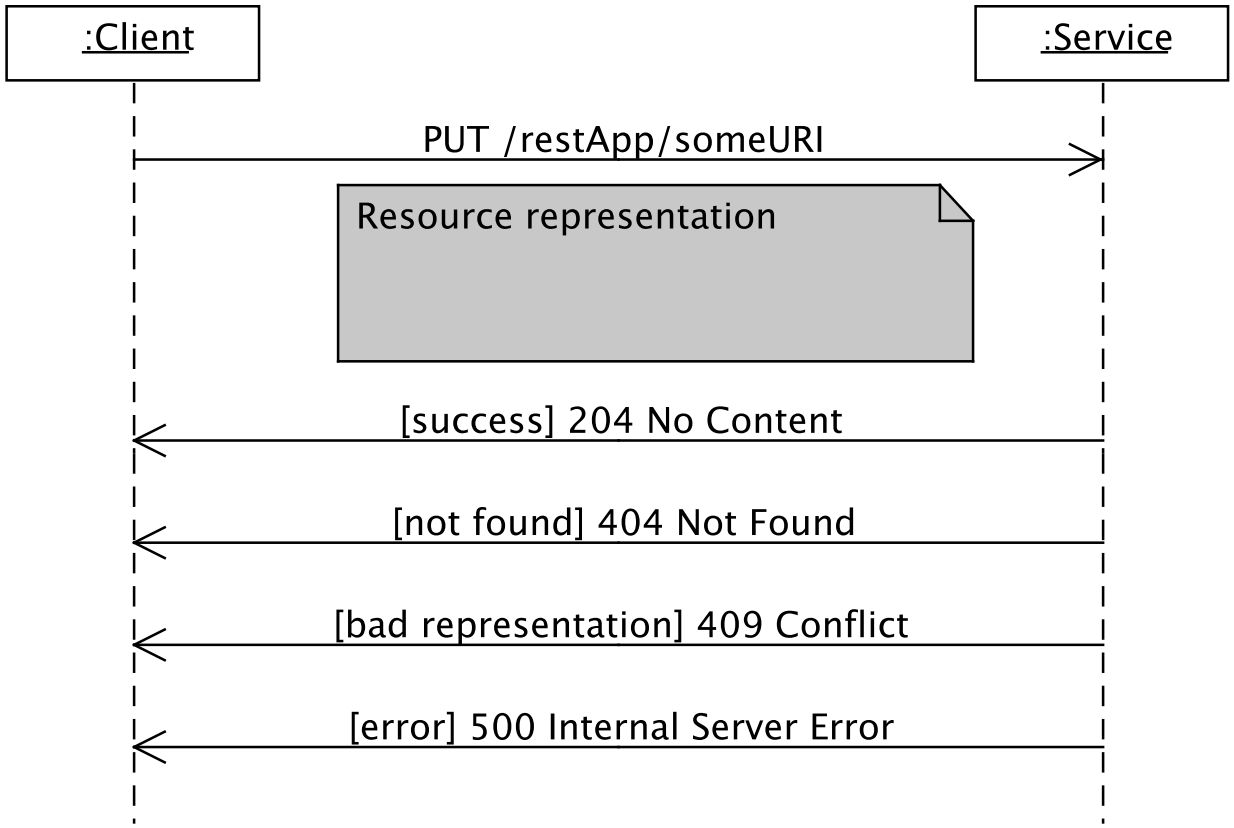
- Client should **GET** to a URI for the requested resource
- Service returns requested resource and **200 OK** or **404 Not Found** or **500 Internal Server Error**



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Implementing Update

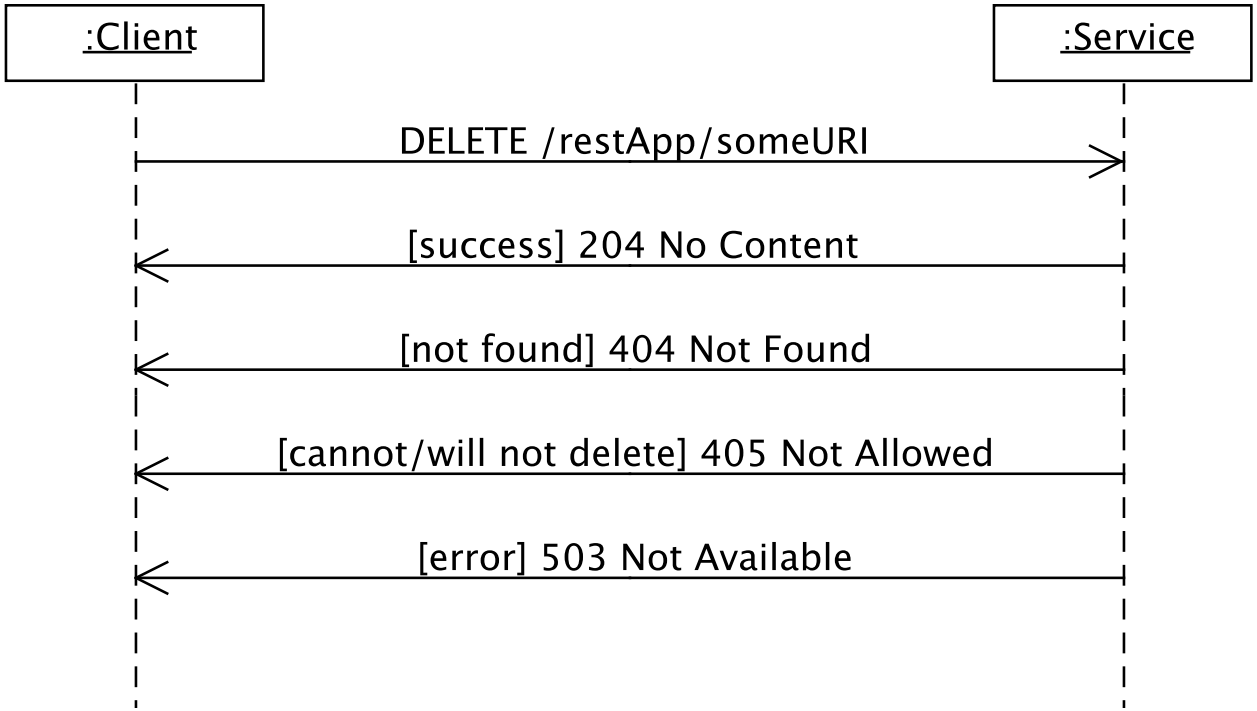
- Client should **PUT** a complete resource representation to the resource's URI
- Service updates the requested resource and returns **204 No Content**, **404 Not Found**, **409 Conflict** or **500 Internal Server Error**



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Implementing Delete

- Client uses HTTP **DELETE** specifying the resource's URI
- Service returns **204 No Content** on success or **404 Not Found**, **405 Method Not Allowed** or **503 Service Unavailable**



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Complete JAX-RS CRUD Example

Student.java

MyContextListener.java

StudentApplication.java

StudentService.java

GetClient.java

PostClient.java

PutClient.java

DeleteClient.java

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```java
package edu.bigcollege;

import java.io.Serializable;

import javax.xml.bind.annotation.XmlRootElement;

@SuppressWarnings("serial")
@XmlRootElement
public class Student implements Serializable
{
    private int studentId;
    private String name;
    private double gpa;

    public Student()
    {
    }

    public Student(int studentId, String name, double gpa)
    {
        super();
        this.studentId = studentId;
        this.name = name;
        this.gpa = gpa;
    }

    public int getStudentId()
    {
        return studentId;
    }

    public void setStudentId(int studentId)
    {
        this.studentId = studentId;
    }

    public String getName()
    {
        return name;
    }

    public void setName(String name)
    {
        this.name = name;
```

```
46
47      public double getGpa()
48      {
49          return gpa;
50      }
51
52      public void setGpa(double gpa)
53      {
54          this.gpa = gpa;
55      }
56
57      @Override
58      public int hashCode()
59      {
60          final int prime = 31;
61          int result = 1;
62          result = prime * result + studentId;
63          return result;
64      }
65
66      @Override
67      public boolean equals(Object obj)
68      {
69          if (this == obj)
70              return true;
71          if (obj == null)
72              return false;
73          if (getClass() != obj.getClass())
74              return false;
75          Student other = (Student) obj;
76          if (studentId != other.studentId)
77              return false;
78          return true;
79      }
80
81      @Override
82      public String toString()
83      {
84          StringBuilder builder = new StringBuilder();
85          builder.append("Student [studentId=").append(studentId)
86                  .append(", name=").append(name).append(", gpa=").append(gpa)
87                  .append("]");
88          return builder.toString();
89      }
```

```java
package edu.bigcollege.listeners;

import java.util.ArrayList;
import java.util.List;

import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;

import edu.bigcollege.Student;

public class MyContextListener implements ServletContextListener
{

    @Override
    public void contextDestroyed(ServletContextEvent ctx)
    {
    }

    @Override
    public void contextInitialized(ServletContextEvent ctx)
    {
        System.out.println("Context initialized");
        List<Student> students = new ArrayList<Student>();

        Student c = new Student();
        c.setGpa(3.44);
        c.setName("Sue Smith");
        c.setStudentId(12);
        students.add(c);

        c = new Student();
        c.setGpa(3.87);
        c.setName("Bill Jones");
        c.setStudentId(44);
        students.add(c);

        ctx.getServletContext().setAttribute("students", students);
    }

}
```

```java
package edu.bigcollege.service;

import java.util.HashSet;
import java.util.Set;

import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;

@ApplicationPath("/rest/")
public class StudentApplication extends Application
{
    @Override
    public Set<Class<?>> getClasses()
    {
        Set<Class<?>> s = new HashSet<Class<?>>();
        s.add(StudentService.class);

        return s;
    }
}
```

```
1   package edu.bigcollege.service;
2
3   import java.net.URI;
4   import java.net.URISyntaxException;
5   import java.util.List;
6   import java.util.Random;
7
8   import javax.servlet.ServletContext;
9   import javax.ws.rs.Consumes;
10  import javax.ws.rs.DELETE;
11  import javax.ws.rs.GET;
12  import javax.ws.rs.POST;
13  import javax.ws.rs.PUT;
14  import javax.ws.rs.Path;
15  import javax.ws.rs.PathParam;
16  import javax.ws.rs.Produces;
17  import javax.ws.rs.core.Context;
18  import javax.ws.rs.core.Response;
19  import javax.ws.rs.core.Response.ResponseBuilder;
20  import javax.ws.rs.core.Response.Status;
21
22  import edu.bigcollege.Student;
23
24  @Path("students")
25  public class StudentService
26  {
27      @Context
28      ServletContext servletContext;
29
30      @SuppressWarnings("unchecked")
31      @DELETE
32      @Path("{id}")
33      public Response delete(@PathParam("id") int id)
34      {
35          ResponseBuilder rb = null;
36
37          System.out.println("In delete: " + id);
38
39          Student found = findStudent(id);
40          if(found != null)
41          {
42              List<Student> students = (List<Student>) servletContext
43                      .getAttribute("students");
44
```

```
46              rb = Response.noContent();
47          }
48          else
49          {
50              rb = Response.status(Status.NOT_FOUND);
51          }
52
53          return rb.build();
54      }
55
56      @PUT
57      @Consumes({ "text/xml", "application/json" })
58      public Response update(Student s)
59      {
60          System.out.println("In update: " + s);
61          ResponseBuilder rb = null;
62
63          if(s != null)
64          {
65              Student found = findStudent(s.getStudentId());
66
67              if(found != null)
68              {
69                  found.setGpa(s.getGpa());
70                  found.setName(s.getName());
71                  rb = Response.noContent();
72              }
73              else
74              {
75                  rb = Response.status(Status.NOT_FOUND);
76              }
77          }
78          else
79          {
80              rb = Response.status(Status.CONFLICT);
81          }
82
83          return rb.build();
84      }
85
86
87      @SuppressWarnings("unchecked")
88      @POST
89      @Consumes({ "text/xml", "application/json" })
```

```
91      // representation in response body
92      @Produces({ "text/xml", "application/json" })
93      public Response insert(Student s)
94      {
95          System.out.println("In insert: " + s);
96          ResponseBuilder rb = null;
97
98          if(s != null)
99          {
100             List<Student> students = (List<Student>) servletContext
101                     .getAttribute("students");
102
103             Random r = new Random();
104             s.setStudentId(Math.abs(r.nextInt()));
105
106             students.add(s);
107
108             try
109             {
110                 rb = Response.created(new URI("/students/" +
111                     s.getStudentId()));
112                 // marshal the student into the response body
113                 rb.entity(s);
114             }
115             catch (URISyntaxException e)
116             {
117                 System.out.println("ERROR inserting: " +
118                   e.getMessage());
119                 rb = Response.serverError();
120             }
121         }
122         else
123         {
124             rb = Response.status(Status.BAD_REQUEST);
125         }
126
127         return rb.build();
128     }
129
130     @SuppressWarnings("unchecked")
131     @GET
132     @Path("{id}")
133     @Produces({ "text/xml", "application/json" })
134     public Response getStudent(@PathParam("id") int id)
```

```
136         Response ret = null;
137 
138         List<Student> students = (List<Student>) servletContext
139                 .getAttribute("students");
140 
141         Student theStudent = null;
142         for (Student c : students)
143         {
144             if (c.getStudentId() == id)
145             {
146                 theStudent = c;
147                 break;
148             }
149         }
150 
151         if (theStudent != null)
152         {
153             ret = Response.ok(theStudent).build();
154         }
155         else
156         {
157             ret = Response.status(Response.Status.NOT_FOUND).build();
158         }
159 
160         return ret;
161     }
162 
163     @SuppressWarnings("unchecked")
164     private Student findStudent(int id)
165     {
166         List<Student> students = (List<Student>) servletContext
167                 .getAttribute("students");
168 
169         Student found = null;
170         for(Student temp : students)
171         {
172             if(temp.getStudentId() == id)
173             {
174                 found = temp;
175                 break;
176             }
177         }
178         return found;
179     }
```

```
181  }
```

```
1    package client;
2
3    import java.io.BufferedReader;
4    import java.io.InputStreamReader;
5    import java.net.HttpURLConnection;
6    import java.net.URL;
7
8    public class GetClient
9    {
10
11       public static void main(String[] args) throws Exception
12       {
13           URL url = new URL(
14             "http://localhost:9081/TestRestCRUDWeb/rest/students/12");
15           HttpURLConnection conn =
16             (HttpURLConnection) url.openConnection();
17           conn.setRequestMethod("GET");
18           conn.setRequestProperty("Accept", "application/json");
19
20           if (conn.getResponseCode() != 200)
21           {
22               throw new RuntimeException("Failed : HTTP error code : "
23                       + conn.getResponseCode());
24           }
25
26           BufferedReader br = new BufferedReader(new InputStreamReader(
27                   (conn.getInputStream())));
28
29           String output;
30           System.out.println("Output from Server .... \n");
31           while ((output = br.readLine()) != null)
32           {
33               System.out.println(output);
34           }
35
36           conn.disconnect();
37
38       }
39   }
```

```
1   package client;
2
3   import java.io.BufferedReader;
4   import java.io.InputStreamReader;
5   import java.io.OutputStream;
6   import java.net.HttpURLConnection;
7   import java.net.URL;
8
9   public class PostClient
10  {
11      public static void main(String[] args) throws Exception
12      {
13          URL url =
14            new URL(
15             "http://localhost:9081/TestRestCRUDWeb/rest/students");
16          HttpURLConnection conn =
17            (HttpURLConnection) url.openConnection();
18          conn.setDoOutput(true);
19          conn.setRequestMethod("POST");
20          conn.setRequestProperty("Content-Type", "application/json");
21
22          String input = "{\"name\":\"Harry Wolfe\",\"gpa\":2.22}";
23
24          OutputStream os = conn.getOutputStream();
25          os.write(input.getBytes());
26          os.flush();
27
28          if (conn.getResponseCode() != HttpURLConnection.HTTP_CREATED)
29          {
30              throw new RuntimeException("Failed : HTTP error code : "
31                      + conn.getResponseCode());
32          }
33
34          BufferedReader br = new BufferedReader(new InputStreamReader(
35                  (conn.getInputStream())));
36
37          String output;
38          System.out.println("Output from Server .... \n");
39          while ((output = br.readLine()) != null)
40          {
41              System.out.println(output);
42          }
43
44          System.out.println("Location header: "
```

```
46
47              conn.disconnect();
48          }
49      }
```

```
1    package client;
2
3    import java.io.OutputStream;
4    import java.net.HttpURLConnection;
5    import java.net.URL;
6
7    public class PutClient
8    {
9        public static void main(String[] args) throws Exception
10       {
11           URL url =
12             new URL(
13               "http://localhost:9081/TestRestCRUDWeb/rest/students");
14           HttpURLConnection conn =
15             (HttpURLConnection) url.openConnection();
16           conn.setDoOutput(true);
17           conn.setRequestMethod("PUT");
18           conn.setRequestProperty("Content-Type", "application/json");
19
20           String input =
21             "{\"name\":\"Joe Blow\",\"gpa\":4.56, \"studentId\":12}";
22
23           OutputStream os = conn.getOutputStream();
24           os.write(input.getBytes());
25           os.flush();
26
27           if (conn.getResponseCode() !=
28              HttpURLConnection.HTTP_NO_CONTENT)
29           {
30               throw new RuntimeException("Failed : HTTP error code : "
31                       + conn.getResponseCode());
32           }
33
34           conn.disconnect();
35       }
36   }
```

```
1    package client;
2
3    import java.net.HttpURLConnection;
4    import java.net.URL;
5
6    public class DeleteClient
7    {
8        public static void main(String[] args) throws Exception
9        {
10           URL url =
11             new URL(
12               "http://localhost:9081/TestRestCRUDWeb/rest/students/44");
13           HttpURLConnection conn =
14             (HttpURLConnection) url.openConnection();
15           conn.setRequestMethod("DELETE");
16
17           conn.connect();
18
19           if (conn.getResponseCode() != HttpURLConnection.HTTP_NO_CONTENT)
20           {
21               throw new RuntimeException("Failed : HTTP error code : "
22                       + conn.getResponseCode());
23           }
24
25           conn.disconnect();
26       }
27   }
```

Chapter Summary

In this chapter, you learned:

- The fundamentals of REST applications that create, retrieve, update and delete resources
- How to implement a RESTful CRUD application using JAX-RS

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring and REST

- Spring Support for REST
- Message Converters
- Annotations

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring Support for REST

- Starting in Spring 3, Spring makes it easy to write SpringMVC **controllers** that work with REST resources
- If you are writing your application with Spring, than using Spring REST is easier and more natural than using JAX-RS

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

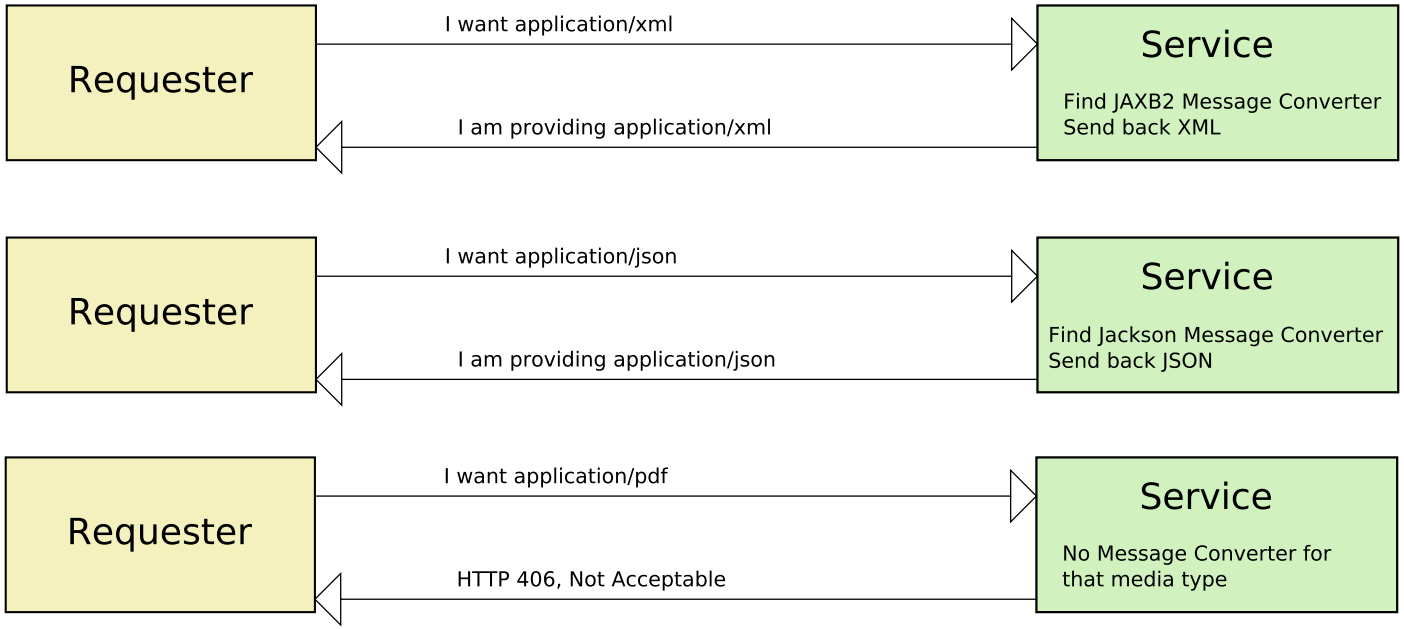
Message Converters

- Spring uses **message converters** to convert Java objects to/from a **resource representation**, including the following:
    - **SpringHttpMessageConverter** - transforms String objects to/from *text/plain*
    - **Jaxb2RootElementHttpMessageConverter** - transforms JAXB-annotated objects to/from *application/xml*
    - **MappingJacksonHttpMessageConverter** - transforms object to/from *application/json*

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Content Negotiation

- The client and server must agree on a data **representation**, which is usually a standard format such as text, XML or JSON using a **MediaType**
- Ultimately, the service decides on which representations it will provide



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Negotiation via HTTP Accept Header

- The HTTP specification allows for an **Accept** header, in which the user agent client provide a list of media types
- Spring parses the Accept header and then finds a message converter that can work with that media type

```
curl -H "Accept: application/xml, application/json"
 http://localhost:9080/TestSpringRESTWeb/rest/advisor/12
```

```
<!-- response -->
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<advisor>
  <id>12</id>
  <name>Sue Joens</name>
  <salary>50000.0</salary>
</advisor>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring REST Annotations

- Spring provides several annotations to configure controllers to work with REST:
    - **@RequestMapping** - assigns a URL and an HTTP method to a processing method
    - **@ResponsStatus** - sets the HTTP status code
    - **@RequestBody** - transforms REST request into Java object(s)
    - **@ResponseBody** - transforms Java object into a REST response
    - **@PathVariable** - initializes a method parameter from part of the request URL
    - **@RequestParam** - initializes a method parameter from an HTTP *request parameter*

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A Simple RESTful Controller

- This controller method returns either XML or JSON depending on the client's **Accept** header

```
1    @Controller
2    @RequestMapping("rest")
3    public class MyService
4    {
5      // respond to URLs like ..../rest/advisor/12
6      @RequestMapping(value = "/advisor/{id}",
7        method = RequestMethod.GET)
8      public @ResponseBody
9        Advisor getAdvisor(@PathVariable("id") int id)
10     {
11       Advisor a = findAdvisorByID(id);
12       if(a == null) throw new ResourceNotFoundException():
13       return a;
14     }
15   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Responding to POST Requests

- POST requests cause the REST service to create the resource and assign its RESTful URL

```
1    @RequestMapping(value = "/advisor",
2      method=RequestMethod.POST)
3    @ResponseStatus(HttpStatus.CREATED)
4    public @ResponseBody Advisor createAdvisor(
5      @RequestBody Advisor a, HttpServletResponse response)
6    {
7        int newID = insertAdvisor(a);
8        a.id = newID;
9
10       response.setHeader("Location", "/advisor/" + newID);
11       return a;
12   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Responding to PUT Requests

- PUT requests cause the REST service to update the RESTful resource

```
1    @RequestMapping(value = "/advisor/{id}",
2      method = RequestMethod.PUT)
3    public ResponseEntity<String> putAdvisor(
4      @PathVariable("id") long id, @RequestBody Advisor a)
5    {
6      ResponseEntity<String> retVal =
7        new ResponseEntity<String>(HttpStatus.NO_CONTENT);
8
9        boolean ok = updateAdvisor(a);
10
11       if(!ok)
12         retVal = new
13           ResponseEntity<String>(HttpStatus.NOT_FOUND);
14
15       return retVal;
16   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring a Spring REST Application

- You configure a RESTful Spring application in the same fashion as for a "normal" Spring MVC application

```
1    <!-- springmvc-servlet.xml -->
2    <beans . . .>
3
4      <context:component-scan
5         base-package="com.goliath" />
6
7      <mvc:annotation-driven />
8    . . .
9    </beans>
```

MyConfig.java

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1   package com.goliath.config;
2
3   import org.springframework.context.annotation.ComponentScan;
4   import org.springframework.context.annotation.Configuration;
5   import org.springframework.web.servlet.config.annotation.DefaultServletHandlerConfigurer;
6   import org.springframework.web.servlet.config.annotation.EnableWebMvc;
7   import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;
8
9   @Configuration
10  @EnableWebMvc
11  @ComponentScan("com.goliath.services")
12  public class MyConfig extends WebMvcConfigurerAdapter
13  {
14      @Override
15      public void configureDefaultServletHandling(
16              DefaultServletHandlerConfigurer configurer)
17      {
18          configurer.enable();
19      }
20  }
```

Chapter Summary

In this chapter, you learned:

- How to use Spring MVC controllers, annotations and message converters to write RESTful CRUD services in Spring.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Spring Boot

- What is Spring Boot?
- Starter Projects
- Autoconfiguration
- Building and Running

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is Spring Boot?

- Spring Boot is an add-on project to Spring that provides:
    - A generator for starter projects that defines common configurations (e.g SpringMVC with JPA)
    - Easier configuration with no XML
    - Ability to run Web applications without deploying to an application server
    - An optional **command-line** interface (CLI)

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

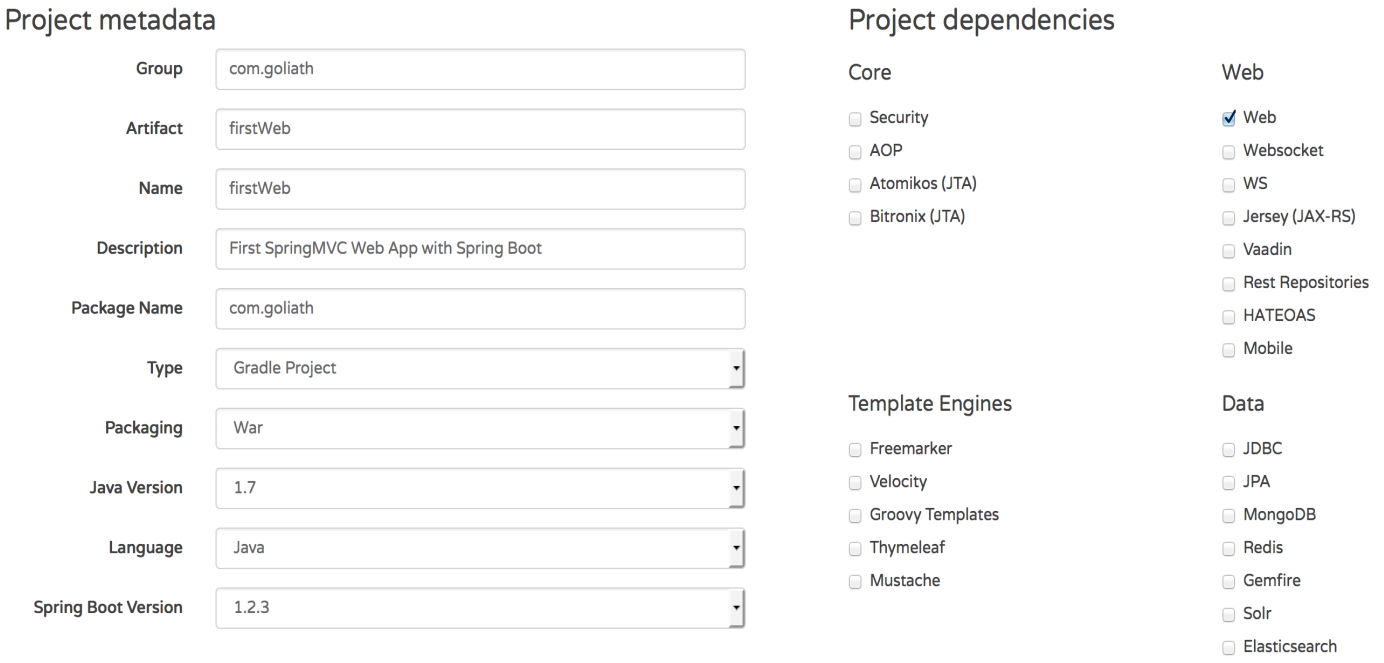
Building Spring Boot Projects

- Spring Boot supports both Maven and Gradle and can generate starter build scripts for both tools
- You can read more about Maven at: https://maven.apache.org/
- You can read more about Gradle at: https://gradle.org/

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Generating a Starter Project

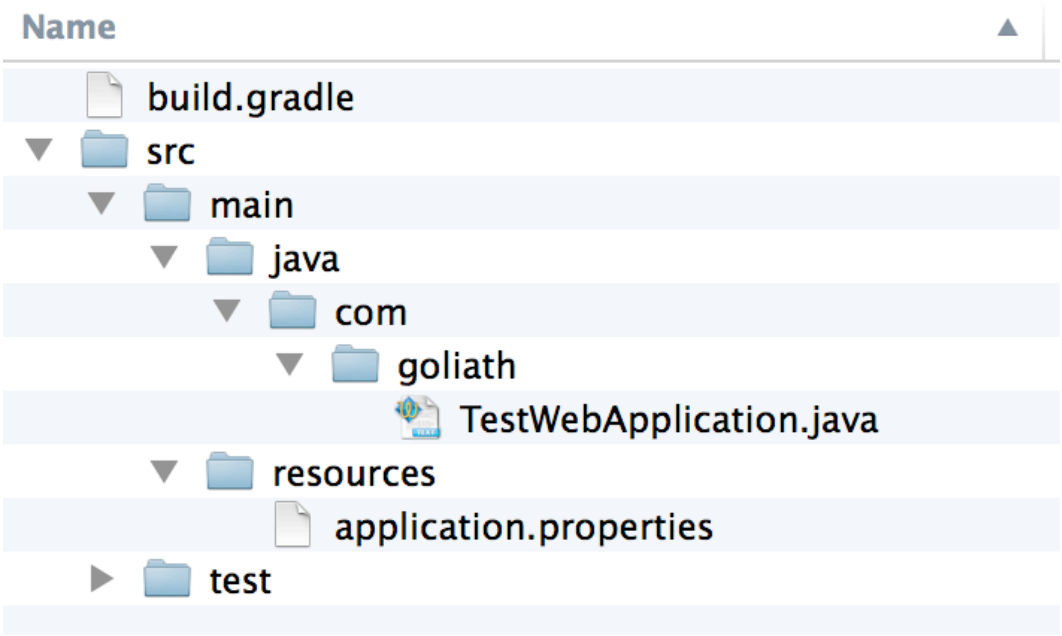
- Spring Boot provides the **Initializr** to generate projects with commonly used features
- You can access the Initializr on the Web at: http://start.spring.io or via the Spring Boot Command-Line Interface



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Gradle Project Structure

- Once you unzip the project generated by the Initializr, you will have a directory structure:



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Generated Build Script

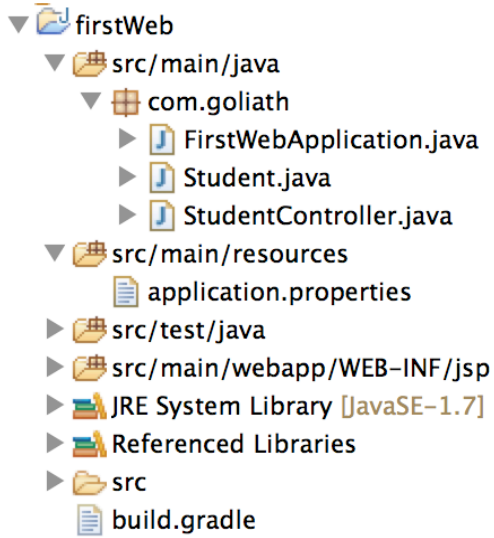
- The Initializr generated a Gradle build script that specifies project metadata, **plugins**, **repositories** and **dependencies**

```
1    ...
2    apply plugin: 'java'
3    apply plugin: 'eclipse'
4    apply plugin: 'idea'
5    apply plugin: 'spring-boot'
6    apply plugin: 'io.spring.dependency-management'
7
8    repositories {
9        mavenCentral()
10   }
11
12   dependencies {
13       compile(
14         "org.springframework.boot:spring-boot-starter-web")
15   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Generating an Eclipse Project

- The project generated by Initialzr does not contain support for an IDE out of the box, but the build script contains **eclipse** and **idea** plugins
- For Eclipse, run **gradle eclipse**, then import the project into your workspace
- You can then create the Java classes, Web pages and so forth that comprise your application



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The 'main' Program

- The Initializr generated a **main()** method class so you can run the application, even if it's a Web application
- The **@SpringBootApplication** annotation configures the class to use Spring Boot's autoconfiguration facility so we don't need any XML configuration

```
1    @SpringBootApplication
2    public class FirstWebApplication {
3
4        public static void main(String[] args) {
5            SpringApplication.run(
6              FirstWebApplication.class, args);
7        }
8    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring Boot Autoconfiguration

- Without Spring Boot, applications need to explicitly configure beans such as data sources, Spring MVC view mappings and so forth
- Spring Boot automatically configures default beans based on what it finds on the application's CLASSPATH
- You can override the defaults if needed, but if the defaults are OK, you can run an application with little or no Spring configuration

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring SpringMVC JSP Views

- It's traditional in Spring MVC to place JSPs in the WEB-INF folder
- For this to work with Spring Boot, you can update the **application.properties** to specify the exact location

```
1    spring.mvc.view.prefix: /WEB-INF/jsp/
2    spring.mvc.view.suffix: .jsp
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using Thymeleaf Instead of JSP

- **Thymeleaf** is a template engine that many Spring developers prefer instead of JSPs for the view:
  - Supports HTML5 out of the box and lets you better preview pages in a browser (JSPs often need to run tag libraries)
  - There is a Spring/Thymeleaf integration library

- If you specify the *thymeleaf* starter dependency, Spring Boot autoconfigures Thymeleaf and expects view templates to be in the src/main/resources/templates folder



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Running the Project

- The Spring Boot plugin provides the **bootRun** task to run Web applications on an embedded Tomcat instance
- This is an easy and fast way to run your application without needing to load it onto an application server



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Complete Example

notes.txt

build.gradle

application.properties

FirstWebApplication.java

Student.java

StudentController.java

studentList.jsp

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
- Download, install gradle and put its "bin" directory
  in PATH

- Create the project using Web-based Spring Boot Initializr:
     Metadata:
        Group:  com.goliath
        Artifact: firstWeb
        Name: firstWeb
        Description: First SpringMVC Web App with Spring Boot
        Package Name: com.goliath
        Type: Gradle Project
        Packaging: JAR
        Java Version: 1.7
        Language: Java
        Spring Boot Version: 1.2.3

   Project dependencies:
        Web

- Generate Project, then unzip

- In terminal (command prompt), change to project directory

- Generate Eclipse project:

   gradle eclipse

- Import "existing project" into Eclipse

- In com.goliath package, create class Student.

- In com.goliath package, create StudentController

- Add to application.properties:

spring.view.prefix: /WEB-INF/jsp/
spring.view.suffix: .jsp

- Create source folder: src/main/webapp/WEB-INF/jsp

- In the new folder, create studentList.jsp

- In gradle.build, add plugin:
```

```
46
47  - In gradle.build, add dependencies:
48
49      providedRuntime("org.apache.tomcat.embed:tomcat-embed-jasper")
50      providedRuntime("javax.servlet:jstl")
51
52  - In terminal:
53
54     gradle bootRun
55
56  - In browser, navigate to http://localhost:8080/studentList
57
58
```

```
buildscript {
    ext {
        springBootVersion = '1.2.3.RELEASE'
    }
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath("org.springframework.boot:spring-boot-gradle-plugin:${springBootVersion}")
        classpath("io.spring.gradle:dependency-management-plugin:0.5.0.RELEASE")
    }
}

apply plugin: 'java'
apply plugin: 'eclipse'
apply plugin: 'idea'
apply plugin: 'spring-boot'
apply plugin: 'io.spring.dependency-management'
apply plugin: 'war'


jar {
    baseName = 'firstWeb'
    version = '0.0.1-SNAPSHOT'
}
sourceCompatibility = 1.7
targetCompatibility = 1.7

repositories {
    mavenCentral()
}


dependencies {
    compile("org.springframework.boot:spring-boot-starter-web")
    testCompile("org.springframework.boot:spring-boot-starter-test")
    providedRuntime("org.apache.tomcat.embed:tomcat-embed-jasper")
    providedRuntime("javax.servlet:jstl")
}


eclipse {
    classpath {
         containers.remove('org.eclipse.jdt.launching.JRE_CONTAINER')
```

```
46      }
47  }
48
49  task wrapper(type: Wrapper) {
50      gradleVersion = '2.3'
51  }
```

```
1    spring.view.prefix: /WEB-INF/jsp/
2    spring.view.suffix: .jsp
```

```java
package com.goliath;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class FirstWebApplication {

    public static void main(String[] args) {
        SpringApplication.run(FirstWebApplication.class, args);
    }
}
```

```java
package com.goliath;

public class Student
{
    private int studentID;
    private String name;
    private double gpa;

    public Student()
    {
    }

    public Student(int studentID, String name, double gpa)
    {
        this.studentID = studentID;
        this.name = name;
        this.gpa = gpa;
    }

    public int getStudentID()
    {
        return studentID;
    }

    public void setStudentID(int studentID)
    {
        this.studentID = studentID;
    }

    public String getName()
    {
        return name;
    }

    public void setName(String name)
    {
        this.name = name;
    }

    public double getGpa()
    {
        return gpa;
    }

}
```

```
46      {
47          this.gpa = gpa;
48      }
49
50  }
```

```java
package com.goliath;

import java.util.ArrayList;
import java.util.List;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
public class StudentController
{

    @RequestMapping(value = "/studentList", method = RequestMethod.GET)
    public String getStudents(Model model)
    {
        List<Student> students = new ArrayList<Student>();

        students.add(new Student(12, "Sue Smith", 3.44));
        students.add(new Student(87, "Harry Wolfe", 2.68));
        students.add(new Student(947, "Jerry Olde", 3.12));

        model.addAttribute("students", students);

        return "studentList";
    }
}
```

```jsp
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<DOCTYPE html>
<html lang="en">
  <head>
    <title>Student List</title>
  </head>
  <body>
    <h1>Student List</h1>
      <ul>
        <c:forEach items="${students}" var="aStudent">
          <li>${aStudent.name}</li>
        </c:forEach>
      </ul>
  </body>
</html>
```

Chapter Summary

In this chapter, you learned:

- What Spring Boot is all about
- How to configure, write and run a Spring Boot Web application

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Additional Spring Boot Topics

- Starter Dependencies
- Auto Configuration
- Deploying a WAR

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Dependent Libraries

- Most large-scale Java applications use third-party libraries, which are typically packaged as JAR files
- Many libraries themselves depend on other libraries, leading to **transitive dependencies**
- Build tools such as Gradle and Maven can satisfy transitive dependencies

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Traditional Approaches for Dependencies

- Libraries typically have multiple versions, with newer versions having additional functionality and/or bug fixes
- Gradle and Maven let you specify specific library version numbers
- One challenge with transitive dependencies is ensuring that the versions "play well together" - traditionally, this job falls to the developer

```
compile("org.springframework:spring-web:4.1.2.RELEASE")
compile("org.springframework.data:spring-data-jpa:1.7.0.RELEASE")
compile("org.hibernate:hibernate-entitymanager:jar:4.2.2.Final")
runtime("com.h2database:h2:1.3.64")
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring Boot Starter Dependencies

- Spring Boot provides **starter dependencies** for both Gradle and Maven that are curated versions of common libraries that work well together
- Instead of specifying individual library versions, you specify a Spring Boot version and Spring Boot provides the dependent versions

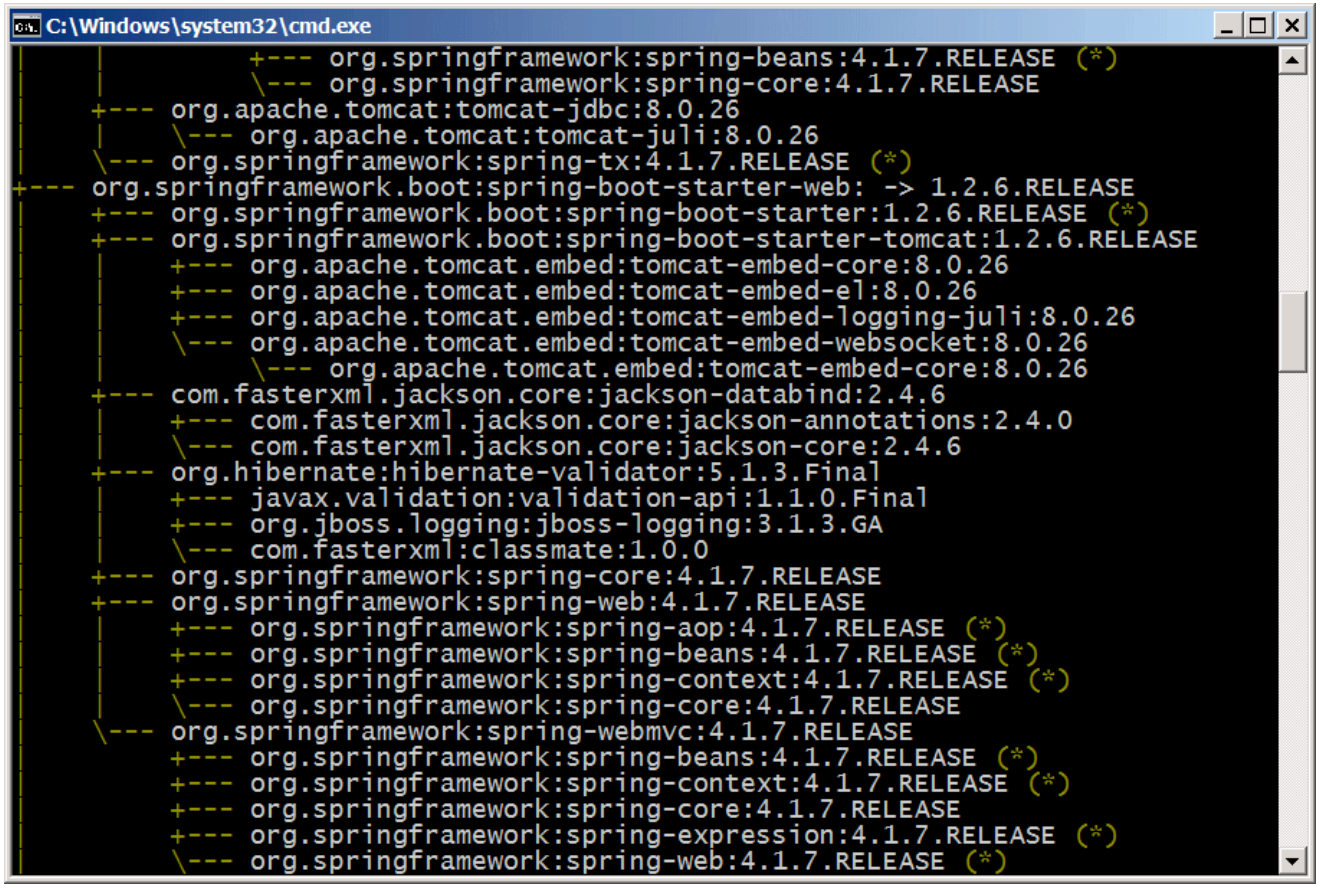
```
compile('org.springframework.boot:spring-boot-starter-jdbc')
compile('org.springframework.boot:spring-boot-starter-web')
runtime('com.h2database.h2')
```

- You can see a list of available starters using the CLI's *spring init --list* command

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Viewing Dependencies

- Gradle and Maven scripts let you view libraries so you can see what the starter dependencies brought in and the library versions



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Eliminating Starter Dependencies

- Starter dependencies bring in commonly used transitive dependencies
- For example, **starter-web** brings in Spring MVC and embedded Tomcat
- You can remove dependencies that your app doesn't need to reduce the size of the executable JAR or WAR

```
compile('org.springframework.boot:spring-boot-starter-web') {
//      exclude group: 'org.hibernate'
        exclude module: 'hibernate-validator'
}
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using a Specific Library

- Due to organizational policy, you might wish to use specific versions of some libraries other than what starters provide
- In Gradle, to use a newer version, simply specify it, but to use an older version, first exclude the older version, then specify the newer

```
compile('org.springframework.boot:spring-boot-starter-jdbc') {
  exclude module: 'snakeyaml'
}

compile('org.yaml:snakeyaml:1.8')
```

```
C:\Users\jbarnum\advspring\workspace\lab02>gradlew dependencies | find "snake"
|    |    |    \--- org.yaml:snakeyaml:1.14 -> 1.8
|    |    \--- org.yaml:snakeyaml:1.14 -> 1.8
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Auto Configuration

- In traditional Spring applications, developers needed to provide extensive configuration either using XML, annotations or Java configuration
- Spring Boot eliminates the need for most configuration, freeing the developer to concentrate on application code

```
1    <bean id="studentDataSource"
2      class=
3    "org.springframework.jdbc.datasource.DriverManagerDataSource">
4      <property name="driverClassName"
5        value="org.hsqldb.jdbcDriver"/>
6      <property name="url"
7        value="jdbc:hsqldb:hsql://localhost/"/>
8      <property name="username"
9        value="SA"/>
10     <property name="password"
11       value=""/>
12   </bean>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

How Auto Configuration Works

- Essentially, Spring Boot makes configuration decisions based on what it finds on the application's **classpath**
- For example, if you have a dependency for the HSQLDB database, Spring Boot will automatically configure a database connection
- For auto config to work, your application's "main" class should be annotated with **@SpringBootApplication**

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Overriding Auto Config Externally

- You can override many auto configuration defaults by adding to the **application.properties** file
- For example, you can specify database connection information:

```
spring.datasource.url=jdbc:hsqldb:hsql://localhost/
spring.datasource.username=SA
spring.datasource.password=
spring.datasource.driver-class-name=org.hsqldb.jdbcDriver
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Executable JAR Deployment

- By default, *Initializr* generates build scripts that create a "fat" executable JAR that you can run to bootstrap the application
- If it's a Web application, executing the JAR launches an embedded Tomcat
- You can configure many server and Tomcat options via properties files

```
server.port=80

# enable SSL
server.port=443
server.ssl.key-store=classpath:mykeystore.jks
server.ssl.key-store-password=duke
server.ssl.key-password=mypassword
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Building a WAR

- Executable JARs are easy, but some organizations use a traditional **application server** such as WebSphere or WebLogic
- Gradle scripts can use the **war** plugin to generate a WAR that you can deploy to an app server

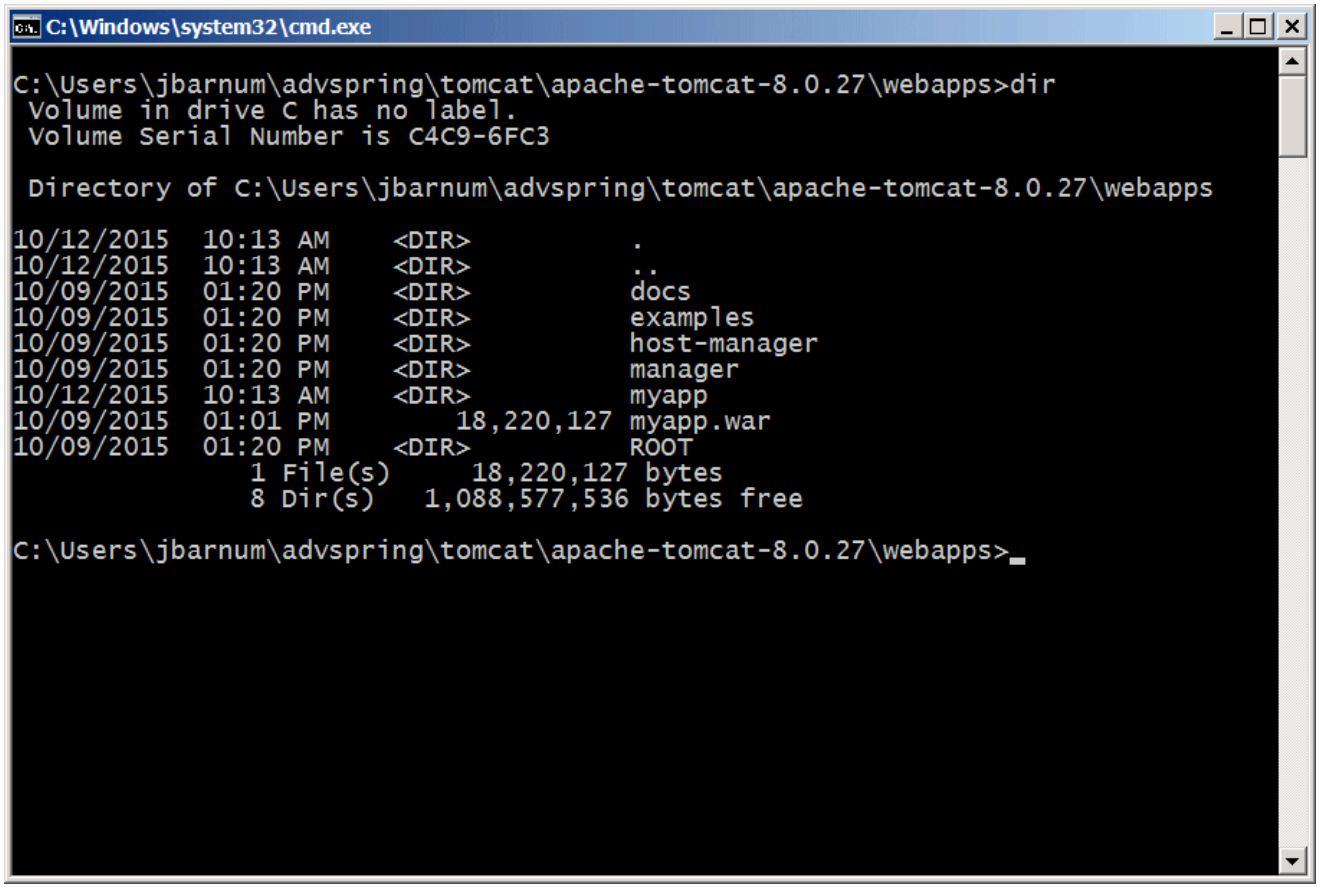
```
apply plugin: 'war'

war {
    baseName = 'myapp'
    version = '0.0.1-SNAPSHOT'
}
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Deploying the WAR

- Deploying WARs is application-server specific - for Tomcat, just drop the WAR into TOMCAT_HOME/webapps



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- About Spring Boot starter dependencies and how to override them
- About Spring Boot auto configuration and how to override it
- How to deploy Spring Boot applications as executable JARs or WARs

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Appendix A: Transactions in Spring

- What is a Transaction?
- Transaction Scope and Propagation

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Transactions

- Transactions let programs define units of work consisting of steps that must all complete or none complete
- If you use transactions properly, your programs will be more robust and reliable

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A Program with No Transactions

- This program does not use transactions and is subject to integrity problems if something goes wrong

```
1    // transfer $100 from savings to checking
2
3    long balance = savings.getBalance();
4    balance -= 100;
5    savings.setBalance ( balance );
6
7    balance = checking.getBalance();
8    balance += 100;
9    checking.setBalance()
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A Program with Transactions

- By adding calls to a *transaction service*, the system will not be corrupted if something goes wrong
- This is an example of program-demarcated transactions

```
1     // transfer $100 from savings to checking
2
3     ts.begin();
4     long balance = savings.getBalance();
5     balance -= 100;
6     savings.setBalance ( balance );
7
8     balance = checking.getBalance();
9     balance += 100;
10    checking.setBalance()
11    ts.commit();
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

ACID Transactions

Atomic Transactions should complete entirely or not at all

Consistent The system state after the transaction should meet "invariants", for example `balance > 0` or `customerID is unique`

Isolated Other transactions cannot "see" intermediate results. Implies locking

Durable Updates to data should survive a system crash, perhaps use transaction logs to recover after a restart

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

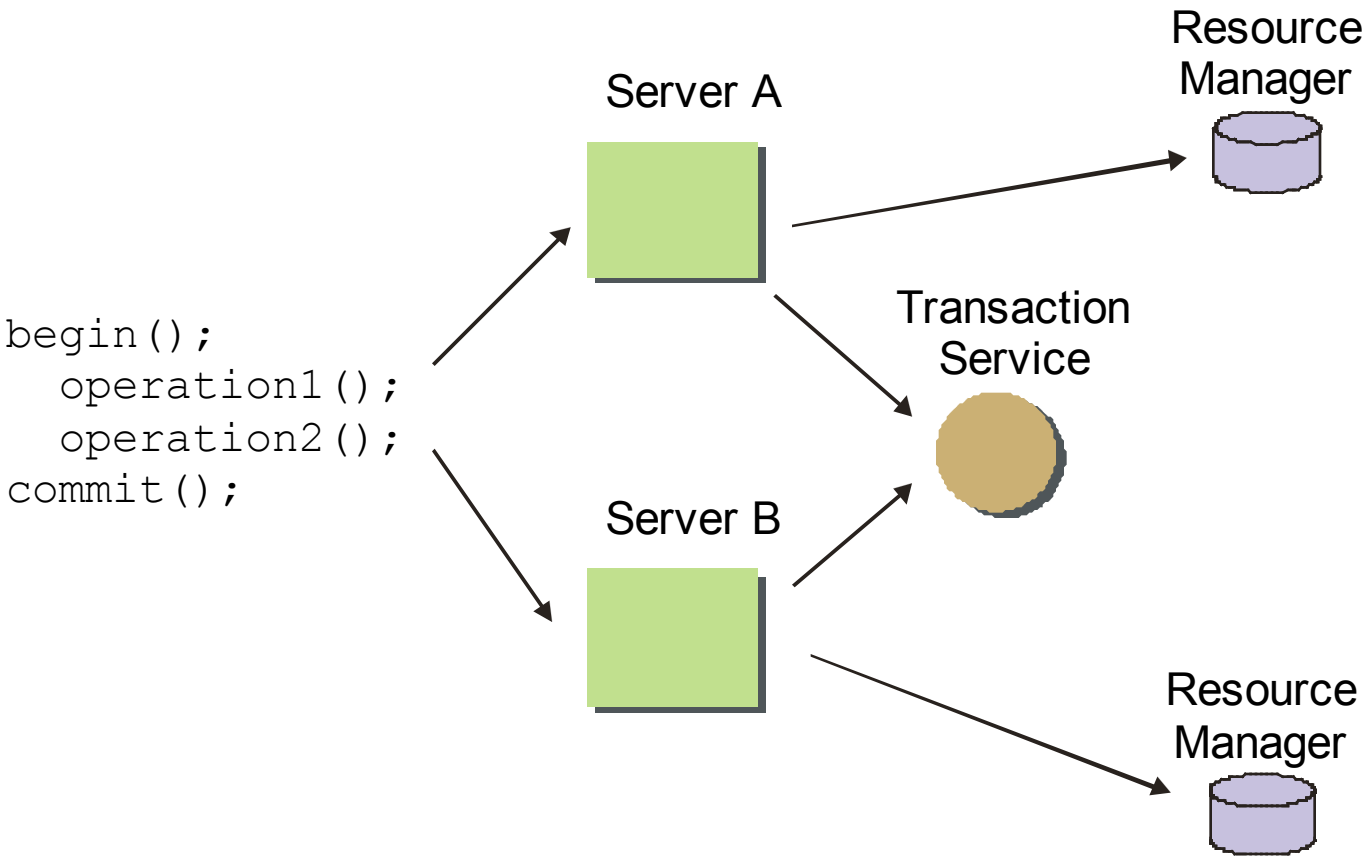
Global and Local Transactions

- **Local** transactions are those associated with a single resource manager, e.g. a relational database
- **Global** transactions are under the control of a **transaction manager**, typically provided an application server
- Global transactions allow for a distributed, two-phase commit across multiple transaction managers
- Unfortunately, local and global transactions have different programing models in Java, thus making it difficult to switch

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Distributed Transactions

- Some transaction managers allow for transactions to be work across different resource managers that adhere to industry standards for transactions



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring and Transactions

- Spring provides the following support for transactions:
    - **Transaction managers** that abstract the underlying transaction architecture (i.e. local vs global)
    - An exception hierarchy based on **unchecked** exceptions
    - Either **programmatic** or **declarative** transaction demarcation

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring Transaction Managers

- Spring provides **transaction managers** that you can configure via dependency injection, thus insulating applications from the actual transaction implementation and programing model
- The Spring transaction managers delegate to the underlying resource manager
- Spring provides several transaction managers, including:
    - DataSourceTransactionManager (basic JDBC)
    - HibernateTransactionManager
    - JpaTransactionManager
    - JtaTransactionManager
    - WebLogicJtaTransactionManger
    - WebSphereUowTransactionManager

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring a DataSource Transaction Manager (XML)

- This is a local transaction manager that works with a database

```
1    <jee:jndi-lookup id="acctDataSource"
2      jndi-name="accountDS"
3      resource-ref="true" />
4
5    <bean id="transactionManager"
6      class=
7    "org.springframework.jdbc.datasource.DataSourceTransactionManager">
8      <property name="dataSource" ref="acctDataSource" />
9    </bean>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring a DataSource Transaction Manager (Java Config)

```
1    @Configuration
2    @EnableTransactionManagement
3    public class MyConfig
4    {
5      @Bean
6      public DataSource acctDataSource()
7      {
8        JndiDataSourceLookup dsLookup = new JndiDataSourceLookup(
9        dsLookup.setResourceRef(true);
10       return  dsLookup.getDataSource("AccountDS");
11     }
12
13     @Bean
14     public DataSourceTransactionManager transactionManager()
15     {
16       return new DataSourceTransactionManager(acctDataSource())
17     }
18   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring a JTA Transaction Manager (XML)

- This transaction manager uses a application server's JTA service and thus typically supports global transactions (e.g. two-phase commit)
- This transaction manager can **enlist** multiple underlying resource managers (e.g. DataSource, Hibernate or JPA transaction managers)

```
<bean id="transactionManager" class=
"org.springframework.transaction.jta.JtaTransactionManager" />
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

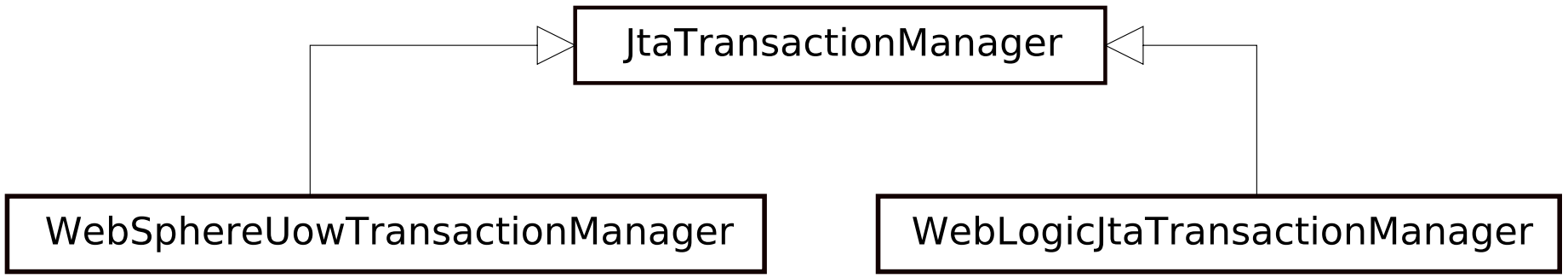
Configuring a JTA Transaction Manager (Java Config)

```
1    @Configuration
2    @EnableTransactionManagement
3    public class MyConfig
4    {
5      . . .
6
7      @Bean
8      public JtaTransactionManager transactionManager()
9      {
10       return new JtaTransactionManager();
11     }
12   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using a Vendor-Specific Transaction Manager

- If your Spring application is running on BEA (Oracle) WebLogic or IBM WebSphere, Spring provides specialized transaction managers tailored for those application servers
- The tailored transaction managers support the full range of Spring transactional capability including transaction suspension



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Transaction Demarcation and Management

- Applications that use transactions must define where a transaction should start and where it should commit or rollback
- We refer to that process as **transaction demarcation**
- Spring lets you demarcate and configure transactions via configuration (declarative) or in the code itself (programmatic)
- This chapter covers only **declarative** transaction management

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

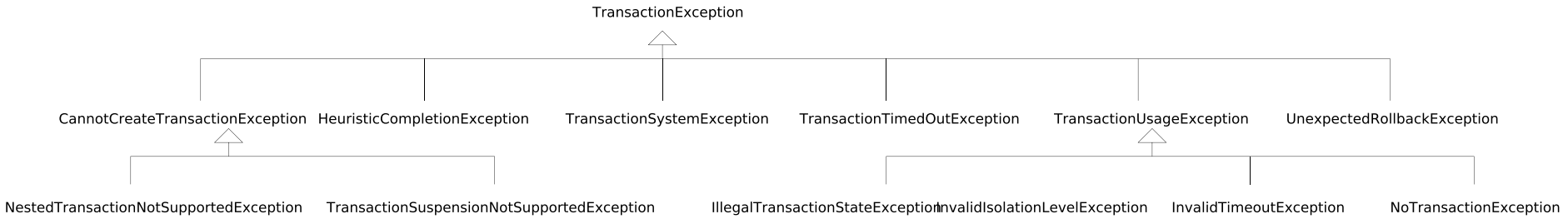
Using Declarative Transaction Management

- According to the Spring docs, most Spring applications use declarative transactions since it affects code the least
- Spring lets you configure transactions either via XML configuration or via annotations in the code
- Behind the scenes, Spring uses **aspect oriented programming (AOP) proxies** to implement declarative transactions

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring Transaction Exception Handling

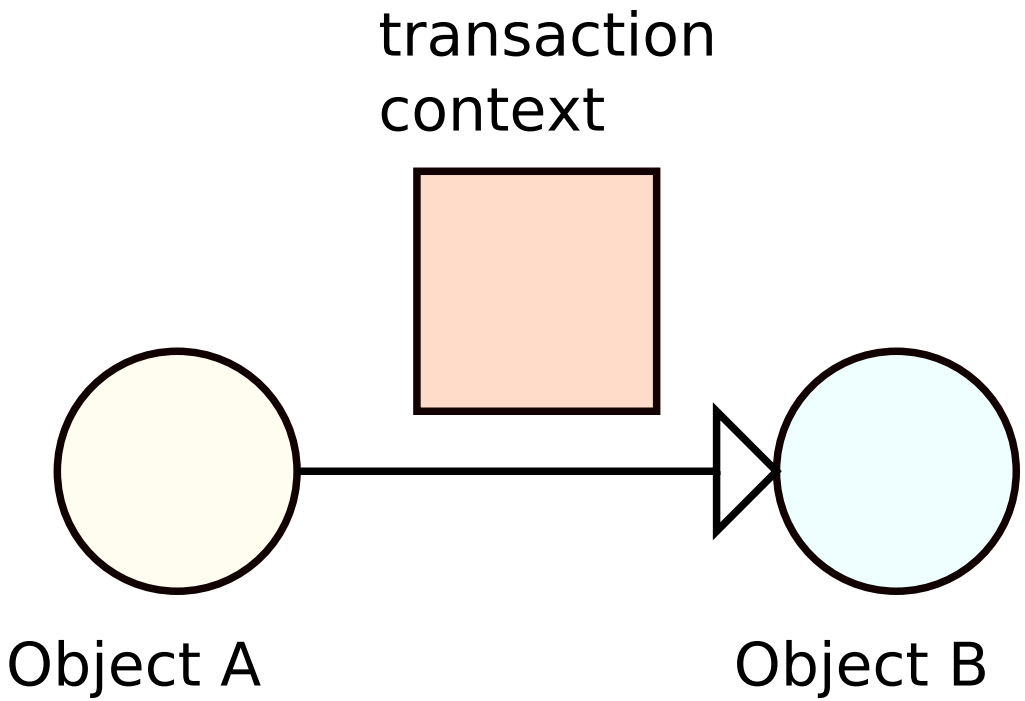
- Since transaction exceptions are generally not recoverable, Spring provides **unchecked** exceptions that you can catch at a global level in your program and report the problem
- This saves you from writing messy try-catch blocks when using programmatic transaction management



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Transaction Propagation

- Transaction **propagation** governs on how transactions flow from object to object
- Spring provides several **propagation attributes**, based on a similar notion in Enterprise JavaBeans



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Transaction Propagation Attributes

Required If there is no existing transaction, start one. If transaction already exists, use existing

RequiresNew If there is no existing transaction, start one. If transaction exists, suspend it and start new

Mandatory If there is no existing transaction, throw exception. If transaction exists, use it

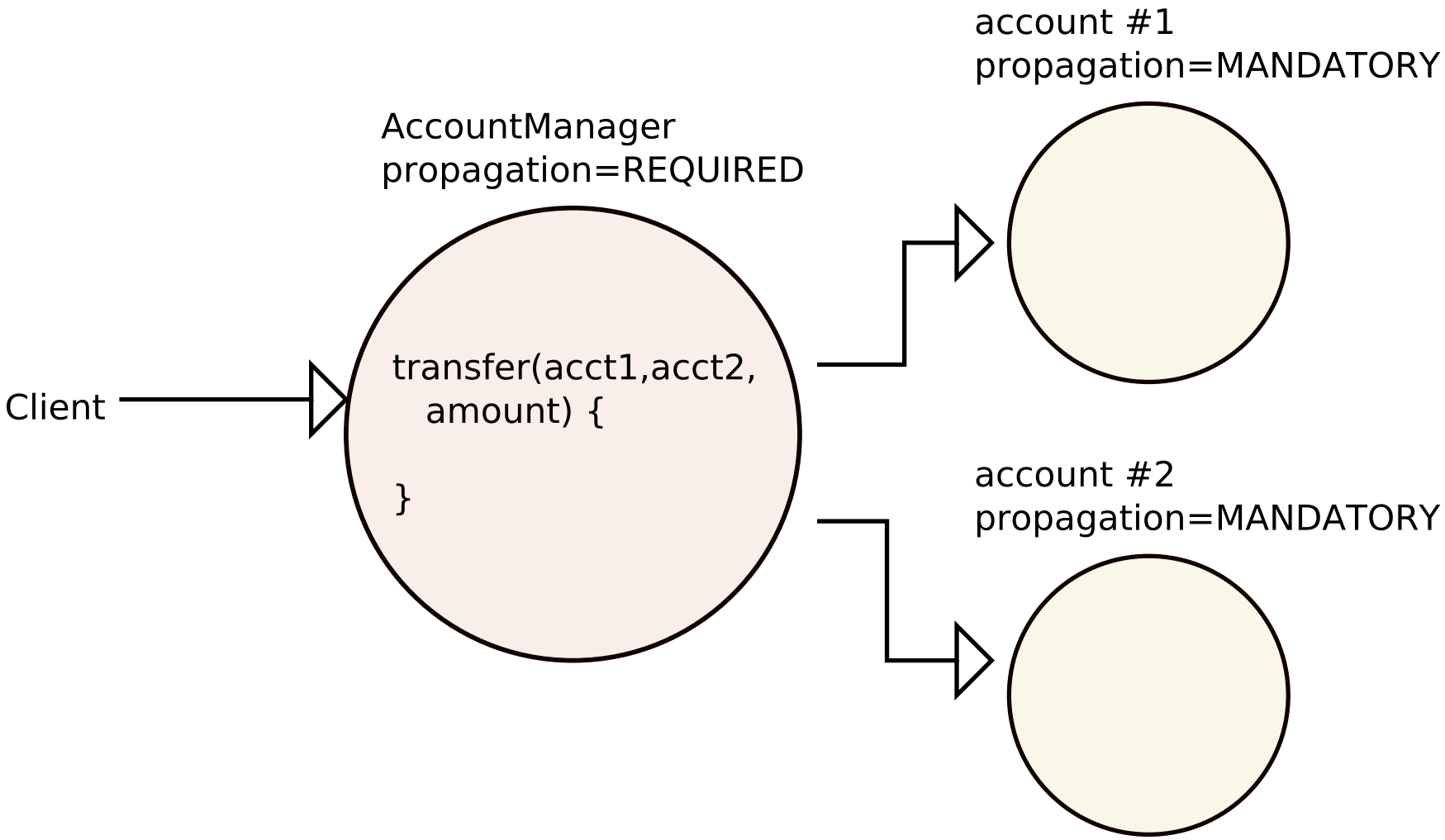
Supports If there is no existing transaction, run method without transaction. If transaction exists, use it

NotSupported If there is no existing transaction, run method without a transaction. If transaction exists, suspend existing transaction then run method with no transaction

Never If there is no existing transaction, run method without a transaction. If transaction exists, throw RemoteException

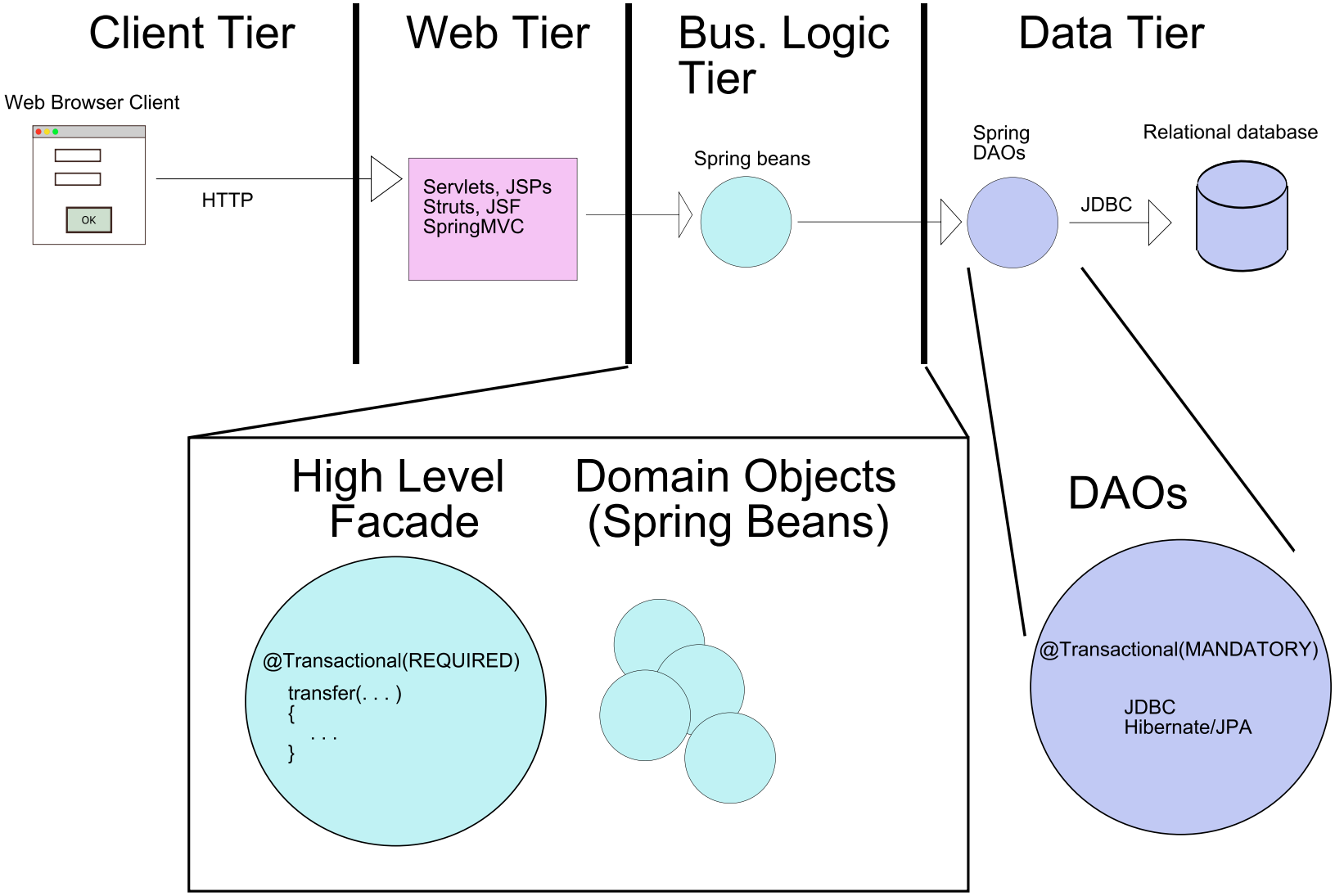
Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Transaction Propagation Example



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A Typical Architecture



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Exceptions and Rolling Back

- Spring automatically rolls back a transaction if an **unchecked** exception occurs, but not for **checked** exceptions
- You can change this behavior on a method-by-method basis during configuration

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Transaction Timeouts

- Most resource managers define a **timeout** after which a running transaction is automatically aborted (rolled back)
- Spring lets you configure this timeout value
- The default is whatever default the resource manager has, or zero (no timeout) for resource managers that don't support timeouts

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Transaction Isolation

- Completely isolated transactions are robust but complete isolation can impact performance
- Spring lets you set an **isolation level** that relaxes isolation at the risk of allowing transactions to "see" intermediate results of other uncommitted transactions

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Transaction Isolation Levels

- The TransactionDefinition interface defines several constants for isolation:

- **ISOLATION_DEFAULT** -- Use resource manager default
- **ISOLATION_READ_COMMITED** -- *Phantom Read* and *Non-repeatable Read* inconsistencies may occur
- **ISOLATION_READ_UNCOMMITED** -- *Dirty Read*, *Phantom Read* and *Non-repeatable Read* inconsistencies may occur
- **ISOLATION_REPEATABLE_READ** -- *Phantom Read* inconsistencies may occur
- **ISOLATION_SERIALIZABLE** -- Transaction is completed isolated

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Read Only Transactions

- In some cases, a resource manager can optimize access if the transaction will not modify data
- Spring lets you configure transactions as **read-only** as a hint to the resource manager

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Specifying Transaction Attributes

- Spring lets you specify transaction propagation attributes either via:
    - Annotations (Spring 2.x or later)
    - XML configuration
- It's generally simpler to use annotations, so that's what we cover in this chapter

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Transaction Annotations

- Instead of, or in addition to using XML configuration, Spring 2.x supports configuring transactions by annotations in the code itself
- You must use the **annotation-driven** element in the configuration file for transactional annotations to work

```
1     Spring configuration file:
2     -------------------------
3     <beans ...
4       xmlns:tx="http://www.springframework.org/schema/tx"
5        ...>
6
7       <tx:annotation-driven
8         transaction-manager="transactionManager" />
9
10    </beans>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The @Transactional Annotation

```
1    @Transactional(
2      propagation=Propagation.MANDATORY)
3    public class MyBean
4    {
5      @Transactional(readOnly=true,
6        rollbackFor=MyException.class)
7      public void myMethod() throws MyException
8      {}
9    }
```

Element	Type	Default
propagation	enum on Propagation type	PROPAGATION_REQUIRED
isolation	enum on Isolation type	ISOLATION_DEFAULT
readOnly	boolean	false
timeout	int (in seconds)	resource manager default
rollbackFor	array of Class objects	none
noRollbackFor	array of Class objects	none

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Complete Declarative Transaction Example

Account.java

AccountManager.java

AccountManagerImpl.java

AccountDAO.java

JdbcAccountDAO.java

NoMoneyException.java

TestTransactions.java

web.xml

AccountConfig.java

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```java
package acct;

public class Account
{
    private Long accountNumber;
    private double balance;
    private String holderName;

    public Long getAccountNumber()
    {
        return accountNumber;
    }
    public void setAccountNumber(Long accountNumber)
    {
        this.accountNumber = accountNumber;
    }
    public double getBalance()
    {
        return balance;
    }
    public void setBalance(double balance)
    {
        this.balance = balance;
    }
    public String getHolderName()
    {
        return holderName;
    }
    public void setHolderName(String holderName)
    {
        this.holderName = holderName;
    }

    @Override
    public String toString()
    {
        return "Acct#: " + accountNumber + ", holder: " + holderName + ", balance: " + balance;
    }
}
```

```
1    package acct;
2
3    public interface AccountManager
4    {
5        public void transfer(int acct1, int acct2, double amount)
6            throws NoMoneyException;
7        public String getAccountInfo(int accountNumber);
8    }
```

```java
package acct;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Transactional;

@Component("accountMgr")
public class AccountManagerImpl
    implements AccountManager
{
    private AccountDAO accountDAO;

    @Autowired
    public void setAccountDAO(AccountDAO accountDAO)
    {
        this.accountDAO = accountDAO;
    }

    @Transactional(rollbackFor=NoMoneyException.class, timeout=10,
            propagation=Propagation.REQUIRED)
    public void transfer(int acct1, int acct2, double amount)
        throws NoMoneyException
    {
        Account a1 = accountDAO.getAccountByID(acct1);
        Account a2 = accountDAO.getAccountByID(acct2);

        double a1Balance = a1.getBalance();
        if (a1Balance - amount < 0)
            throw new NoMoneyException();

        double a2Balance = a2.getBalance();
        a1Balance -= amount;
        a2Balance += amount;
        a1.setBalance(a1Balance);
        a2.setBalance(a2Balance);
        accountDAO.updateAccount(a1);
        accountDAO.updateAccount(a2);
    }

    @Transactional(readOnly=true)
    public String getAccountInfo(int accountNumber)
    {
        Account a = accountDAO.getAccountByID(accountNumber);
```

```
46          }
47      }
```

```
1    package acct;
2
3    public interface AccountDAO
4    {
5        public Account getAccountByID(int accountNumber);
6        public void updateAccount(Account a);
7    }
```

```
1   package acct;
2
3   import java.sql.ResultSet;
4   import java.sql.SQLException;
5
6   import javax.sql.DataSource;
7
8   import org.springframework.beans.factory.annotation.Autowired;
9   import org.springframework.jdbc.core.JdbcTemplate;
10  import org.springframework.jdbc.core.RowMapper;
11  import org.springframework.stereotype.Repository;
12  import org.springframework.transaction.annotation.Propagation;
13  import org.springframework.transaction.annotation.Transactional;
14
15  @Repository("accountDao")
16  public class JdbcAccountDAO implements AccountDAO
17  {
18      private static class AccountRowMapper implements
19              RowMapper<Account>
20      {
21          public Account mapRow(ResultSet rs, int index) throws SQLException
22          {
23              Account acct = new Account();
24              acct.setAccountNumber(rs.getLong("ACCT_NUM"));
25              acct.setBalance(rs.getDouble("BALANCE"));
26              acct.setHolderName(rs.getString("HOLDERNAME"));
27              return acct;
28          }
29      }
30
31      private JdbcTemplate template;
32      private AccountRowMapper mapper;
33
34      @Autowired
35      public void setDataSource(DataSource ds)
36      {
37          template = new JdbcTemplate(ds);
38          mapper = new AccountRowMapper();
39      }
40
41      @Transactional(propagation=Propagation.MANDATORY)
42      public void updateAccount(Account a)
43      {
44          String SQL = "UPDATE account SET balance=?,holdername=? WHERE acct_num=?";
```

```
46                      a.getAccountNumber());
47          }
48
49          @Transactional(propagation=Propagation.MANDATORY)
50          public Account getAccountByID(int accountNumber)
51          {
52              String SQL = "SELECT * FROM account WHERE ACCT_NUM=?";
53              return template.queryForObject(SQL,
54                      mapper, accountNumber);
55          }
56      }
```

```java
package acct;

@SuppressWarnings("serial")
public class NoMoneyException extends Exception
{
    public NoMoneyException()
    {
        super("Insufficient funds for transfer");
    }
}
```

```
1   package servlets;
2
3   import java.io.IOException;
4   import java.io.PrintWriter;
5
6   import javax.servlet.ServletContext;
7   import javax.servlet.ServletException;
8   import javax.servlet.annotation.WebServlet;
9   import javax.servlet.http.HttpServletRequest;
10  import javax.servlet.http.HttpServletResponse;
11
12  import org.springframework.web.context.WebApplicationContext;
13  import org.springframework.web.context.support.WebApplicationContextUtils;
14
15  import acct.AccountManager;
16
17  @SuppressWarnings("serial")
18  @WebServlet("/TestTransactions")
19  public class TestTransactions extends javax.servlet.http.HttpServlet
20  {
21      protected void doGet(HttpServletRequest request,
22              HttpServletResponse response) throws ServletException, IOException
23      {
24          response.setContentType("text/html");
25          PrintWriter out = response.getWriter();
26
27          ServletContext servletContext = getServletContext();
28          WebApplicationContext ctx = WebApplicationContextUtils
29                  .getRequiredWebApplicationContext(servletContext);
30          try
31          {
32              AccountManager mgr = (AccountManager)ctx.getBean("accountMgr");
33
34              out.println("Before transfer: <br>");
35              out.println(mgr.getAccountInfo(1) + "<br>");
36              out.println(mgr.getAccountInfo(2) + "<br>");
37
38              mgr.transfer(1, 2, 1);
39
40              out.println("After transfer: <br>");
41              out.println(mgr.getAccountInfo(1) + "<br>");
42              out.println(mgr.getAccountInfo(2) + "<br>");
43          }
44          catch (Exception e)
```

```
46                 out.println("Something went wrong: " + e);
47             }
48         }
49     }
```

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xsi:schemaL
  <display-name>TestTransactionsTomcatWeb</display-name>
  <resource-ref>
    <res-ref-name>jdbc/account</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
    <res-sharing-scope>Shareable</res-sharing-scope>
  </resource-ref>
  <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
  </listener>
  <context-param>
    <param-name>contextClass</param-name>
    <param-value>org.springframework.web.context.support.AnnotationConfigWebApplicationContext</param-value>
  </context-param>
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>config.AccountConfig</param-value>
  </context-param>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

```
1   package config;
2
3   import javax.sql.DataSource;
4
5   import org.springframework.context.annotation.Bean;
6   import org.springframework.context.annotation.ComponentScan;
7   import org.springframework.context.annotation.Configuration;
8   import org.springframework.jdbc.datasource.DataSourceTransactionManager;
9   import org.springframework.jdbc.datasource.lookup.JndiDataSourceLookup;
10  import org.springframework.transaction.annotation.EnableTransactionManagement;
11
12  @Configuration
13  @ComponentScan("acct")
14  @EnableTransactionManagement
15  public class AccountConfig
16  {
17      @Bean
18      public DataSource acctDataSource()
19      {
20        JndiDataSourceLookup dsLookup =
21            new JndiDataSourceLookup();
22        dsLookup.setResourceRef(true);
23        DataSource dataSource = dsLookup.getDataSource(
24            "jdbc/account");
25        return dataSource;
26      }
27
28        @Bean
29        public DataSourceTransactionManager transactionManager()
30        {
31          return new DataSourceTransactionManager(acctDataSource());
32        }
33  }
```

Chapter Summary

In this chapter, you learned:

- The basics of how transactions work
- About Spring transaction managers
- How to configure and control transactions in Spring

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum