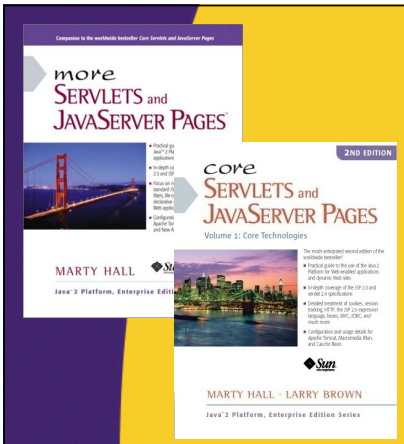
The Spring Framework: Core Capabilities Part 2

Originals of Slides and Source Code for Examples:
http://courses.coreservlets.com/Course-Materials/spring.html

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, Struts, JSF/MyFaces/Facelets, Ajax, GWT, Spring, Hibernate/JPA, Java 5 & 6.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.



For live Spring & Hibernate training, see courses at http://courses.coreservlets.com/.

Taught by the experts that brought you this tutorial. Available at public venues, or customized versions can be held on-site at <u>your</u> organization.

- Courses developed and taught by Marty Hall
  - Java 5, Java 6, intermediate/beginning servlets/JSP, advanced servlets/JSP, Struts, JSF, Ajax, GWT, custom mix of topics
- Courses developed and taught by coreservlets.com experts (edited by Marty)
  - Spring, Hibernate/JPA, EJB3, Ruby/Rails

Contact hall@coreservlets.com for details

Topics in This Section

- Bean naming
- Bean scoping
- Dependency injection

General Approach Review

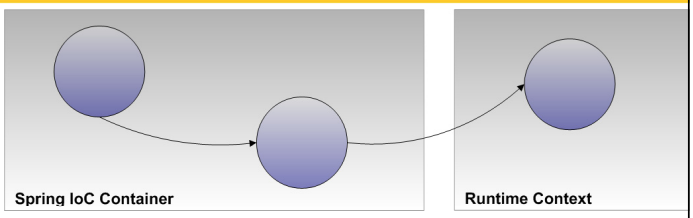
General Approach Review

- Define and create service interfaces
- Implement service interfaces
- Add bean definitions
- Access and use container-managed beans

Choosing a Bean Name

Bean Name

- **Uses**
  - Identify bean collaborators when defining Spring beans
  - Identify beans to be accessed programmatically from the Spring IoC container using the **`BeanFactory`** API



- **Configuration**
  - XML **`bean`** attribute **`id`**
    - W3C XML Schema datatype ID
    - Accepts a single value
  - XML **`bean`** attribute **`name`**
    - One or more values delimited by comma, semicolon, or space
  - Separate XML element **`alias`**

Bean Definition Example

- **Establishes multiple names for clients**
- **Single class implementing multiple interfaces**

```
<bean id="beanId"
      name="beanName01,beanName02;beanName03 beanName04"
      class="coreservlets.Bean"/>

beanFactory.getBean("beanId");
beanFactory.getBean("beanName01");
beanFactory.getBean("beanName02");
beanFactory.getBean("beanName03");
beanFactory.getBean("beanName04");
```

Bean Definition Example

- Adapts names from client contexts

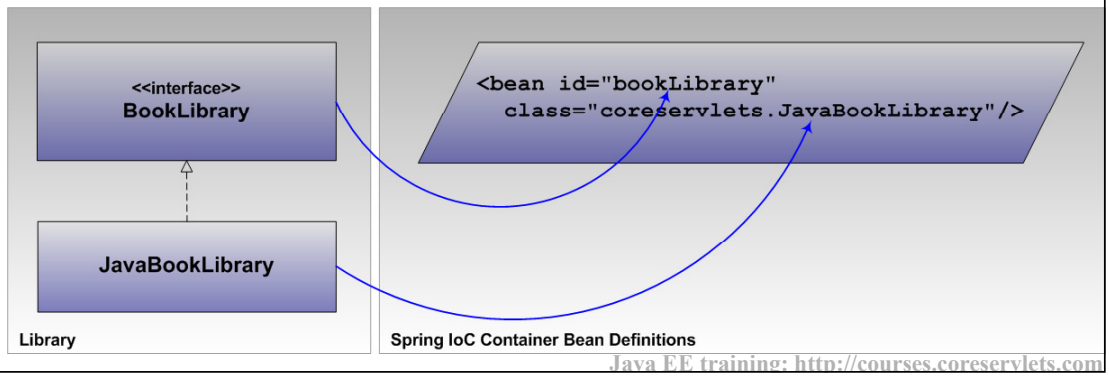
```
<bean id="beanId"
      class="coreservlets.Bean"/>

<alias name="beanId" alias="beanAlias"/>

beanFactory.getBean("beanAlias");
```


Naming Convention

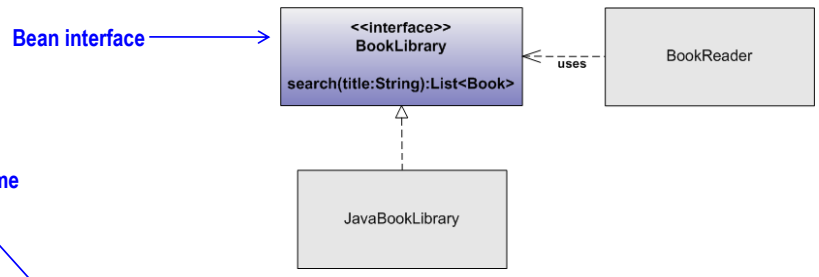
- Establish a convention in order to predict bean names
- Identify objects according to the interface
- Use unqualified class names
- Start name with a lower-case letter



Naming Convention

- **Conventions are guidelines**
  - When necessary, it is okay to depart from the convention
  - This may demonstrate the need to refactor
    - For example, separate or merge responsibilities

Naming Convention Example



Bean name

```
<beans>
  <bean id="bookLibrary"
        class="coreservlets.JavaBookLibrary" />
</beans>
```

Naming Example

- Convention coordinates when accessing the Spring container

Bean name

BeanFactory#getBean name parameter

```
<beans>
  <bean id="bookLibrary"
        class="coreservlets.JavaBookLibrary" />
</beans>

BookLibrary bookLibrary = (BookLibrary)
  beanFactory.getBean("bookLibrary");
```


Naming Convention Example

- Convention coordinates naming within a bean definition document

Bean name

Bean reference

```
<beans>
  <bean id="bookLibrary"
        class="coreservlets.JavaBookLibrary" />

  <bean id="bookReader"
        class="coreservlets.BookReader">
    <constructor-arg ref="bookLibrary" />
  </bean>
</beans>
```

Naming Convention Example

- **Convention coordinates naming across separate bean definition files**

```
<beans>
  <bean id="bookLibrary"
        class="coreservlets.JavaBookLibrary" />
</beans>
```

Bean name

service-context.xml

```
<beans>
  <bean id="bookReader"
        class="coreservlets.BookReader">
      <constructor-arg ref="bookLibrary" />
  </bean>
</beans>
```

Bean reference

client-context.xml

Naming Convention Example

- **Coordinating names across contexts**
  - Test
  - Production

```
<bean id="bookReader">
  <constructor-arg
   ref="bookLibrary"/>
</bean>
```

reference

Library applicationContext.xml

```
<bean id="bookLibrary"
  class="MockBookLibrary" />
```

reference

Test applicationContext.xml

```
<bean id="bookLibrary"
 class="ProductionBookLibrary"/>
```

Production applicationContext.xml

Bean Naming Summary

- **Establish a naming convention**
  - Unqualified class name
  - Name starting with a lower-case letter
  - Name is based on the interface type
- **Use the convention to predict names across contexts**
- **Use the XML `bean id` attribute for declaring the bean name**
- **Rely on the same conventions for referencing other beans**

General Approach Review

- **Define and create service interfaces**
- **Implement services interfaces**
- **Add the bean definitions**
  - Establish bean identifiers using the `id` attribute
    - Aliases can be established using `name` attribute or `alias` element
  - Develop bean names consistently using a convention
- **Access and use container-managed beans**
  - The access and integration method is context-dependent

Bean Scope

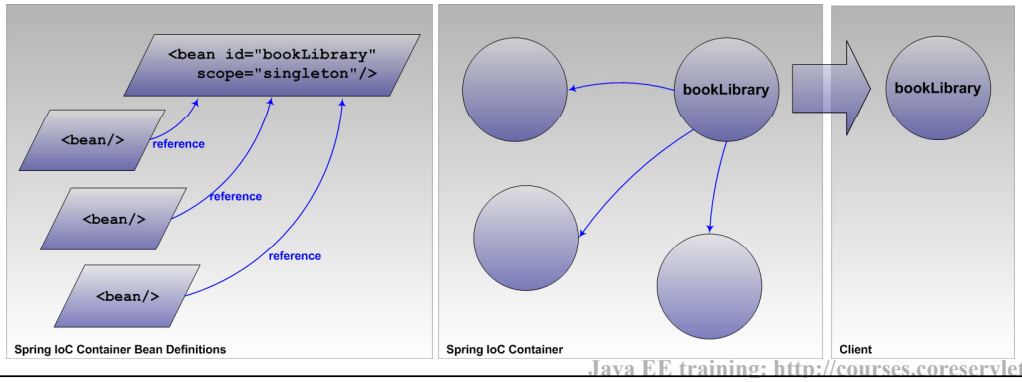
Bean Scope

- **Bean scopes control bean creation and storage**
- **The Spring IoC container defines five bean scopes**
  - `singleton` **(default)**
  - `prototype`
  - `request`*
  - `session`*
  - `globalSession`*

    ****Only available in web application environments***
- **Bean scope configuration is exposed through the XML `bean` attribute `scope`**

Singleton Scope

- **The default bean scope**
  - Explicitly specifying `singleton` is allowed but redundant
- **Caches and distributes the same bean instance**
  - Collaborative references
  - **`BeanFactory`** bean access requests
- **Replaces singleton implementations**



Singleton Scope Example

```
<bean id="bookLibrary" class="coreservlets.JavaBookLibrary"
  scope="singleton"/>

BeanFactory beanFactory =
  new ClassPathXmlApplicationContext(
    "/applicationContext.xml");

Object a = beanFactory.getBean("bookLibrary");
Object b = beanFactory.getBean("bookLibrary");
Object c = beanFactory.getBean("bookLibrary");
Object d = beanFactory.getBean("bookLibrary");

System.out.printf("%s, %s, %s%n",
                  a == b, b == c, c == d);
```

Bean scope

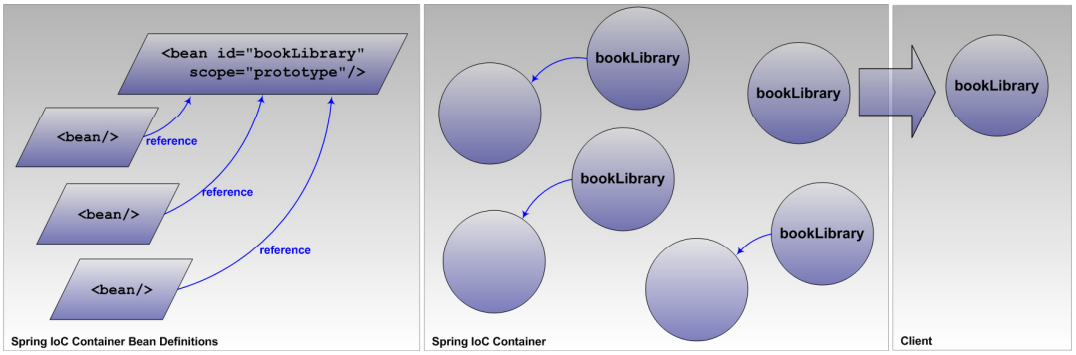
Bean references

Standard output

```
true, true, true
```

Prototype Scope

- **Prototype scope caches and distributes a new bean instance for various types of bean requests**
  - Collaborative references
  - **`BeanFactory`** bean access requests



Prototype Scope Example

```
<bean id="bookLibrary" class="coreservlets.JavaBookLibrary"
  scope="prototype"/>
```

Bean scope

```
BeanFactory beanFactory = new ClassPathXmlApplicationContext(
  "/applicationContext.xml");

Object a = beanFactory.getBean("bookLibrary");
Object b = beanFactory.getBean("bookLibrary");
Object c = beanFactory.getBean("bookLibrary");
Object d = beanFactory.getBean("bookLibrary");

System.out.printf("%s, %s, %s%n",
                  a == b, b == c, c == d);
```

Bean references

Standard output

```
false, false, false
```

Externally-Stored Bean Scopes

- **Java Servlet scope support**

Spring scope name	Java Servlet
request	request
session	session
globalSession	application
n/a	page

General Approach Review

- **Define and create service interfaces**
- **Implement services interfaces**
- **Add the bean definitions**
  - Establish identifiers using the **`id`** attribute
    - Aliases can be established using **`name`** attribute or **`alias`** element
  - Develop bean names consistently using a convention
  - Default to singleton beans
    - Override bean creation and caching using **`scope`** attribute
- **Access and use container-managed beans**
  - The access and integration method is context-dependent

Dependency Injection (DI)

Dependency Injection Target

```
public class BookReader {

  private BookLibrary bookLibrary;

  public BookReader() {
  }

  public BookReader(BookLibrary bookLibrary) {
    this.bookLibrary = bookLibrary;
  }

  public void setBookLibrary(BookLibrary bookLibrary) {
    this.bookLibrary = bookLibrary;
  }
  ...
}
```

Property Setter DI Target

```
public class BookReader {

  private BookLibrary bookLibrary;

  public BookReader() {
  }

  public BookReader(BookLibrary bookLibrary) {
    this.bookLibrary = bookLibrary;
  }

  public void setBookLibrary(BookLibrary bookLibrary) {
    this.bookLibrary = bookLibrary;
  }
  ...
}
```


Property Setter DI Example

```
<beans>
  <bean id="bookLibrary"
        class="coreservlets.JavaBookLibrary"/>

  <bean id="bookReader"
        class="coreservlets.BookReader">
    <property name="bookLibrary" ref="bookLibrary"/>
  </bean>
</beans>
```

Property Setter DI Details

- **Matches on qualified setter methods by name**
- **Based on JavaBean conventions**
  - The property setter has no relationship to a class field
  - Must conform to the capitalized property name prefixed with **`set`**
    - e.g., property **`foo`** as the setter method **`setFoo`**
  - Must be **`public`**
    - Configuration fails with private, protected, or package-private modifiers
  - Must specify the return type **`void`**
    - Incompatible with chaining setter methods
  - Must accept exactly one parameter
  - Must be enclosed by a class that has a no-args constructor
    - No requirements on the access modifier

Property Setter DI Implications

- **Forces the class to be mutable**
- **Includes the property name in the DI declaration**
- **Supports inheritance**

Property Setter DI Bean Definition

- **Configuration is exposed by the XML element type `property`**
  - child to the XML element **`bean`**
  - requires a value for the XML element attribute **`name`**
- **Supports shorthand configuration**
  - **`value`** attribute for directly specifying a value
  - **`ref`** attribute for referring to beans defined elsewhere
- **Supports child references**
  - Collaborators, inner beans, collections, and values

Property Setter DI Process

- **Verify the presence of a no-args constructor**
- **Verify the method signature**
  - **`public`** access modifier
  - **`void`** return type
  - bean-oriented name
- **Map the property into the bean definition**
  - Create the nested XML **`property`** element
  - Identify the property using the **`name`** attribute
  - Identify the value to be passed into the setter method as a parameter value

Property Setter DI Example

```
public class BookReader {

  private BookLibrary bookLibrary;

  public BookReader() {
  }

  public BookReader(BookLibrary bookLibrary) {
    this.bookLibrary = bookLibrary;
  }

  public void setBookLibrary(BookLibrary bookLibrary) {
    this.bookLibrary = bookLibrary;
  }
  ...
}
```

No-args constructor

Bean-oriented name

Single parameter

void return type

public access modifier

Property Setter DI Example

```
<beans>
  <bean id="bookLibrary"
        class="coreservlets.JavaBookLibrary"/>

  <bean id="bookReader"
        class="coreservlets.BookReader">
    <property name="bookLibrary" ref="bookLibrary"/>
  </bean>
</beans>
```

BookReader property

Setter parameter value

Alternative Property Setter DI Example

```
<beans>
  <bean id="bookLibrary"
        class="coreservlets.JavaBookLibrary"/>

  <bean id="bookReader"
        class="coreservlets.BookReader">
    <property name="bookLibrary">
      <ref bean="bookLibrary"/>
    </property>
  </bean>
</beans>
```

BookReader property

Setter parameter value

Constructor DI Target

```
public class BookReader {

  private BookLibrary bookLibrary;

  public BookReader() {
  }

  public BookReader(BookLibrary bookLibrary) {
    this.bookLibrary = bookLibrary;
  }

  public void setBookLibrary(BookLibrary bookLibrary) {
    this.bookLibrary = bookLibrary;
  }
  ...
}
```

Constructor DI Example

```
<beans>
  <bean id="bookLibrary"
        class="coreservlets.JavaBookLibrary"/>

  <bean id="bookReader"
        class="coreservlets.BookReader">
    <constructor-arg ref="bookLibrary"/>
  </bean>
</beans>
```


Constructor DI Details

- **Matches on a constructor using parameter types**
  - Must share matching quantities
  - Must share matching types
  - Must match while preserving parameter ordering
- **Constructor by type matching is ambiguous**
  - Multiple constructors may match the **`constructor-arg`** configuration

Constructor DI Implications

- Supports an immutable class design
- Does not support parameter naming
- Requires multiple constructors for each parameter parameter combination

Constructor DI Bean Definition

- Configuration is exposed by the XML element type `constructor-arg`
  - Child of the XML element `bean`
  - Supports shorthand configuration
    - `value` attribute for directly injecting a value
    - `ref` for referring to beans defined elsewhere
- Supports child references
  - Collaborators, inner beans, collections, and values
- Offers fine-tuned constructor matching options
  - `index`
  - `type`

Constructor DI Example

```
public class BookReader {

  private BookLibrary bookLibrary;

  public BookReader(BookLibrary bookLibrary) {
    this.bookLibrary = bookLibrary;
  }
  ...
}
```

Dependency injection interface

Implied Constructor DI

- Implied index and type

```
<beans>
  <bean id="bookLibrary"
        class="coreservlets.JavaBookLibrary"/>

  <bean id="bookReader"
        class="coreservlets.BookReader">
    <constructor-arg ref="bookLibrary"/>
  </bean>
</beans>
```

Implied parameter index and type

Parameter value

Explicit Constructor DI

- Explicit index and type

```
<beans>
  <bean id="bookLibrary"
        class="coreservlets.JavaBookLibrary"/>

  <bean id="bookReader"
        class="coreservlets.BookReader">
    <constructor-arg index="0"
                     type="coreservlets.BookLibrary"
                     ref="bookLibrary"/>
  </bean>
</beans>
```

Explicit parameter index and type

Implied Constructor Matching

- Implied matches are ambiguous

```
public class Values {
  public Values(Integer integer, String string){
  }
  public Values(String string, Integer integer){
  }
  public Values(String string, String anotherString){
  }
}

<beans>
  <bean class="coreservlets.di.constructor.Values">
    <constructor-arg value="abc" />
    <constructor-arg value="123" />
  </bean>
</beans>
```

Implied Constructor Matching

- Arguments can be automatically re-ordered

```
public class Values {
  public Values(Integer integer, String string){
  }
  public Values(String string, Integer integer){
  }
}

<beans>
  <bean class="coreservlets.Values">
    <constructor-arg value="123" />
    <constructor-arg value="abc" />
  </bean>
</beans>
```


Implied Constructor Matching

- Implied matches are unreliable

```
public class Values {
  public Values(Integer integer, String string){
  }
}
<beans>
  <bean class="coreservlets.Values">
    <constructor-arg value="abc" />
    <constructor-arg value="123" />
  </bean>
</beans>
```

Standard output

```
Exception in thread "main"
org.springframework.beans.factory.UnsatisfiedDependencyException
Failed to convert value of type [java.lang.String] to required
type [java.lang.Integer]
```

Explicit Constructor Matching

- Implied matches can be corrected by specifying the `index` and/or `type`

```
public class Values {
  public Values(Integer integer, String string){
  }
}
<beans>
  <bean class="coreservlets.Values">
    <constructor-arg value="abc" index="1"/>
    <constructor-arg value="123" index="0"/>
  </bean>
</beans>
```


Constructor Matching

- Always specify `index` and `type` when you have overloaded constructors

Constructor DI Alternative

```
public class ValuesFactory {

  private Integer integer;
  private String string;

  public Values newValuesInstance() {
    return new Values(integer, string);
  }

  public void setIntegerValueFromString(String string) {
    this.integerValue = Integer.parseInt(string);
  }

  public void setIntegerValue(Integer integerValue) {
    this.integerValue = integerValue;
  }
  ...
}
```

Factory method

Additional setter

Constructor DI Alternative

```
<bean id="values"
      class="coreservlets.Values"
      factory-bean="valuesFactory"
      factory-method="newValuesInstance" />

<bean id="valuesFactory"
      class="coreservlets.ValuesFactory">
  <property name="stringValueFromInteger" value="123" />
  <property name="integerValueFromString" value="456" />
</bean>
```

Target bean

Factory bean

Standard output

```
stringValue=123 integerValue=456
```

DI Alternatives

- Lookup Method
- Autowiring
- Annotation-driven autowiring

Lookup Method DI

- Overrides or implements a factory method
  - `abstract` keyword is optional
  - `public` modifier is required
  - `final` classes are not supported
- Accommodates special use cases
  - Scope loss between bean collaborators if the dependency specifies a more transient scope
  - Version of the abstract template pattern
- Requires the `cglib` library

Lookup Method DI

```
public abstract class AbstractBookLibraryVisitor
implements BookLibraryVisitor{
  public int visitLibrary(int visitCount){
    Set<Integer>set = new HashSet<Integer>();
    while(visitCount-- > 0){
      set.add(getBookLibrary().hashCode());
    }
    return set.size();
  }
  public abstract BookLibrary getBookLibrary();
}

<bean id="bookLibraryVisitor"
      class="coreservlets.AbstractBookLibraryVisitor">
  <lookup-method name="getBookLibrary" bean="bookLibrary"/>
</bean>
<bean id="bookLibrary" class="coreservlets.JavaBookLibrary"
      scope="prototype"/>
```


Lookup Method DI

```
public class Main {
  public static void main(String[] args) {
    BeanFactory beanFactory = new
      ClassPathXmlApplicationContext(
      "/applicationContext.xml");

    BookLibraryVisitor visitor = (BookLibraryVisitor)
      beanFactory.getBean("bookLibraryVisitor");

    System.out.println("BookLibraryVisitor class: %s%n",
      visitor.getClass().getSimpleName());
  }
}
```

object from cglib generated class

Standard output

```
BookLibraryVisitor class:
AbstractBookLibraryVisitor$$EnhancerByCGLIB$$9fec4bdc
```

Autowiring DI

- Enabled by specifying an autowiring strategy
- Autowiring is implied
- DI resolution may quietly fail
  - Property setter dependency resolution

Autowiring DI

```
public class BookReader {
   public void setBookLibrary(BookLibrary bookLibrary) {
      this.bookLibrary = bookLibrary;
   }
   ...
}

<bean id="BookReader"
  class="coreservlets.BookReader"
  autowire="byName"/>

<bean id="bookLibrary"
  class="coreservlets.JavaBookLibrary"/>
```

Autowiring setting

Dependency resolution

Annotation-Driven Autowiring

- **Uses annotations for specifying dependencies**
- **Reduces XML configuration**
- **Introduces Spring class imports into source code**
  - Violates POJO design principles
- **Decentralizes application configuration**
  - Mitigates the benefits of a central blueprint
- **Implies dependencies**

Annotation-Driven Autowiring Process

- **Declare an empty XML element, `context:annotation-config`**
  - Requirement depends on **`BeanFactory`** implementation
- **Annotate dependency target**
  - Field
  - Method
  - Constructor

Annotation-Driven Autowiring Example

```
<?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-2.5.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-
   2.5.xsd">

   <bean id="annotatedBookReader"
         class="coreservlets.AnnotatedBookReader"/>

   <bean id="bookLibrary"
         class="coreservlets.JavaBookLibrary"/>

   <context:annotation-config/>
</beans>
```


Annotation-Driven Autowiring Field Example

```
import org.springframework.beans.factory.annotation.*;

public class AnnotatedBookReader {

  @Autowired
  private BookLibrary bookLibrary;
  ...
}
```

Annotation-Driven Autowiring Constructor Example

```
import org.springframework.beans.factory.annotation.*;

public class BookReader {

  private BookLibrary bookLibrary;

  @Autowired
  public BookReader(BookLibrary bookLibrary) {
    this.bookLibrary = bookLibrary;
  }
  ...
}
```


Annotation-Driven Autowiring Property Setter Example

```
import org.springframework.beans.factory.annotation.*;

public class BookReader {

  private BookLibrary bookLibrary;

  @Autowired
  public void setBookLibrary(BookLibrary bookLibrary) {
    this.bookLibrary = bookLibrary;
  }
  ...
}
```

Autowired Test Contexts

```
import org.springframework.beans.factory.annotation.*;

public class BookLibraryTest {

  @Autowired
  public BookLibrary bookLibrary;

  @Test
  public void verify() { ... }

  ...
}
```


General Approach Review

- **Define and create service interfaces**
- **Implement services interfaces**
- **Add the bean definitions**
  - Establish identifiers using the **`id`** attribute
    - Aliases can be established using **`name`** attribute or **`alias`** element
  - Develop bean names consistently using a convention
  - Default to singleton beans
    - Override bean creation and caching using **`scope`** attribute
  - Specify bean inter-dependencies using DI mechanisms
    - Property setter, constructor, lookup-method, autowiring
- **Access and use container-managed beans**
  - The access and integration method is context-dependent

Wrap-up

Summary

- **Bean naming**
  - Establish naming standards
  - Use **`id`** and/or **`name`** attributes or separate **`alias`** elements
    - Favor **`id`** attribute
- **Simple bean configuration**
  - Use **`constructor-arg`** elements
- **Complex bean configuration**
  - Use **`property`** elements
  - Consider using a separate factory bean with **`property`** elements
    - Target bean: **`factory-bean`** and **`factory-method`**
    - Factory bean: **`id`** and **`class`**

Summary Continued

- **Transient bean collaborator reference**
  - Add cglib library
  - Use nested **`lookup-method`** element with **`name`** attribute
- **Autowired**
  - May require XML **`context`** element **`annotation-config`**

Questions?