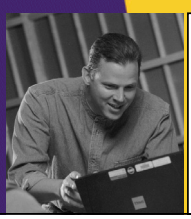
The Core Spring Module:
Defining Beans and Dependency Injection

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

© 2009 Marty Hall

For live Java training, please see training courses at http://courses.coreservlets.com/. Servlets, JSP, Struts, JSF, Ajax, GWT, Java 5, Java 6, Spring, Hibernate, JPA, and customized combinations of topics.



Taught by the author of *Core Servlets and JSP*, *More Servlets and JSP*, and this tutorial. Available at public venues, or customized versions can be held on-site at your organization. Contact hall@coreservlets.com for details.

Agenda

- **Setting bean properties**
- **Supplying constructor arguments**
- **Using factory methods**
- **Dependency injection**
  - Supplying other beans as properties or constructor args
- **Bean scopes**

Overview

Main Capabilities of Core Spring Module

- **Bean definition file**
  - Objects whose implementations are likely to change are defined in XML file. Java code does not need to refer to any specific implementation
    - You use the <bean> tag to define object's name and class
    - You use nested <property> or <constructor-arg> elements to give startup values to the object
- **Container based on bean definition file**

```
ApplicationContext context =
    new ClassPathXmlApplicationContext("/bean-file.xml");
```

- **You get object instances from the container**

```
(InterfaceType)context.getBean("bean-name")
```


Dependency Injection

- **Spring is useful when**
  - You have objects whose implementations change often
    - These objects are defined in bean definition file, isolating Java code from changes in the implementation
    - You supply initialization values via constructors or setters
- **Spring is even more useful when**
  - The initialization values are other beans. That is, your Spring-managed objects depend on other bean values.
  - Supplying these values in bean definition file is called "dependency injection"
    - Because Java code doesn't have to depend explicitly on specific concrete values
    - Instead, bean values are passed in ("injected") at run time
    - Also called "Inversion of Control" (IoC)

Basic Approach

- **Define interface or abstract class**
  - No dependencies on Spring
- **Make concrete implementations of interface**
  - No dependencies on Spring
- **Declare concrete object in bean defn. file**

```
<bean id="bean-name" class="package.SpecificClass">
  <property …/> or <constructor-arg…/>
</bean>
```

- **Instantiate container from bean defn. file**

```
ApplicationContext context =
    new ClassPathXmlApplicationContext("/bean-file.xml");
```

- **Get bean instance(s)**

```
GeneralType bean =
  (GeneralType)context.getBean("bean-name");
```


Simple Properties

Setting Properties: Basics

- **<property name="foo" value="bar"/>**
  - When you declare a bean (but don't use constructor-arg), it calls the zero-argument constructor when the object is instantiated. You use <property…/> tags to specific what setter methods are called after the constructor runs.

```
<bean id="some-name" class="package.SomeClass">
    <property name="someProp" value="some-value"/>
</bean>
```

  - This means that when getBean is called, the zero argument SomeClass constructor is called, then setSomeProp is called.
  - Simple type conversions will be performed, so setSomeProp can take String, int, Integer, double, Double, etc.

Bean Properties

- **Idea**
  - The bean definition file refers to a "bean property", which is a shortcut for a setter method name.
    - The instance variable name is irrelevant
  - In general, you come up with the bean property name by dropping "set" from the setter method name, then changing the next letter to lowercase.
    - But if the first two letters after "set" are uppercase, Java assumes it is an acronym, and the bean property name does not start with a lowercase letter
- **Examples**

Setter Method Name	Bean Property Name
setFirstName	firstName
setURL	URL (not uRL)

Fancy Property Features (Covered Later)

- **ref: to refer to bean declared elsewhere**

```
<bean id="bean1" …>…</bean>
<bean id="bean2" …>
  <property name="blah" ref="bean1"/>
</bean>
```

- **Nested <bean>: to supply new bean as value**

```
<property name="blah">
  <bean…>…</bean>
</property>
```

- **list: to pass a List to the setter method**

```
<property name="blah">
  <list>…</list>
</property>
```

- **map: to pass a Map to the setter method**

```
<property name="blah">
  <map>…</map>
</property>
```


Bean Definition Files

- **Basic format**

```
<?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.xsd">
  <bean…>…</bean>
  <bean…>…</bean>
</beans
```

- **Creating**
  - With Spring IDE
    - R-click on src folder → New → Other → Spring → Spring Bean Definition
  - By copying applicationContext.xml from spring-blank
    - Download from coreservlets.com Spring Tutorial
  - By hand

Storing Bean Definition Files

- **Most common: top level of class path**
  - Instantiate container with
    new ClassPathXmlApplicationContext("/file.xml");
- **Subdirectory of class path**
  - Instantiate container with
    new ClassPathXmlApplicationContext("/dir/file.xml");
- **Anywhere on file system**
  - Instantiate container with
    new FileSystemXmlApplicationContext("/usr/hall/file.xml");
- **In WEB-INF (Web applications only)**
  - Instantiate container with special listener when app loads.
  - Access container with static methods in WebApplicationContextUtils

Getting Bean Instances

- **Instantiate the container**

```
ApplicationContext context =
    new ClassPathXmlApplicationContext
                ("/applicationContext.xml");
```

  - Notes
    - Instantiating the container is expensive
    - Container should be instantiated once only
- **Get instance from the container**

```
SomeType myBean =
    (SomeType)context.getBean("bean-name");
```

  - Notes
    - You normally treat bean as the abstract type
    - Instantiating a bean is inexpensive
    - You often instantiate beans many times
      - By default, calling getBean on the same name multiple times returns the same instance. See later section on bean scopes.

Example: Shapes

- **Goal**
  - Test out various geometric calculations. Top-level code should not change with types of shapes change.
- **Approach**
  - Abstract class: Shape
    - abstract getArea method, concrete printInfo method
  - Concrete classes
    - Rectangle, Circle, RightTriangle
  - Bean definition file

```
<bean id="shape1" class="coreservlets.Rectangle">
  <property name="length" value="10"/>
  <property name="width" value="20"/>
</bean>
```

  - Driver class
    - Gets application context, calls getBean, casts to Shape

Making Shapes Project

- **From scratch**
  - File → New → Project → Spring → Spring Project
    - Or, if no Spring IDE, File → New → Java → Java Project
  - Named project spring-core
  - R-clicked on project, made new folder called lib
    - Copied *spring-install*/dist/spring.jar and *spring-install*/lib/jakarta-commons/commons-logging.jar to lib
    - R-clicked on project, Properties → Libraries → Add JARs
      - Then pointed at lib/spring.jar and lib/commons-logging.jar
  - R-clicked src folder and New → Other → Spring → Spring Bean Definition
    - If no Spring IDE, copied sample applicationContext.xml file
- **By copying existing project**
  - Copied spring-blank
  - Renamed copy to spring-core

Abstract Class

```
package coreservlets;

public abstract class Shape {
  public abstract double getArea();

  public void printInfo() {
    System.out.printf("%s with area of %,.2f%n",
                      getClass().getSimpleName(),
                      getArea());
  }
}
```


Concrete Class 1

```
public class Rectangle extends Shape {
  private double length, width;

  public Rectangle() {}

  public Rectangle(double length, double width) {
    setLength(length);
    setWidth(width);
  }
```

If you instantiate an object from the bean definition file, it is common to use zero-arg constructor and then to invoke setter methods (via <property> tags).

If you instantiate a Rectangle from Java code, you would expect a constructor like this.

Concrete Class 1 (Continued)

```
public double getLength() {
  return(length);
}

public void setLength(double length) {
  this.length = length;
}

public double getWidth() {
  return(width);
}

public void setWidth(double width) {
  this.width = width;
}

public double getArea() {
  return(length * width);
}
}
```

You can use <property name="length" …/> or <property name="width"…/>. Property names are based on setter method names, not on instance variable names.

Concrete Class 2

```
public class Circle extends Shape {
  private double radius = 1.0;

  public Circle() {}

  public Circle(double radius) {
    setRadius(radius);
  }

  public double getRadius() {
    return(radius);
  }

  public void setRadius(double radius) {
    this.radius = radius;
  }

  public double getArea() {
    return(Math.PI * radius * radius);
  }
}
```

Concrete Class 3

```
public class RightTriangle extends Shape {
  private double sideA, sideB;

  public RightTriangle() {}

  public RightTriangle(double sideA, double sideB) {
    setSideA(sideA);
    setSideB(sideB);
  }

  public double getSideA(…) {…}
  public void setSideA(double sideA) {…}
  public double getSideB(…) {…}
  public void setSideB(double sideB) {…}
  public double getHypotenuse() {…}

  public double getArea() {
    return(0.5 * sideA * sideB);
  }
}
```


Bean Definition File

```
<?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.xsd">

  <bean id="shape1" class="coreservlets.Rectangle">
    <property name="length" value="10"/>
    <property name="width" value="20"/>
  </bean>

  …
</beans>
```

Driver Class (Instantiate Container and Get Instances)

```
package coreservlets;

import org.springframework.context.*;
import org.springframework.context.support.*;

public class ShapeTest {
  public static void main(String[] args) {
    ApplicationContext context =
      new ClassPathXmlApplicationContext
                          ("/applicationContext.xml");
    for(int i=1; i<=4; i++) {
      Shape shape = (Shape)context.getBean("shape" + i);
      shape.printInfo();
    }
    …
  }
}
```

Driver class is usually only one that imports Spring-related packages.

Output

- **Executing in Eclipse**
  - Right-click inside main and choose Run As → Java Application
- **Output for value "shape1"**

```
Informational messages about container starting
…
Rectangle with area of 200.00
```

Constructor Arguments

Constructor Args: Basics

- **<constructor-arg value="…"/>**
  - Instead of calling the zero-arg constructor and then calling setter methods (via <property…/>), you can supply constructor arguments.

```
<bean id="some-name" class="package.SomeClass">
  <constructor-arg value="some-value"/>
</bean>
```

  - There are three philosophies on when to use <property…/> vs. when to use <constructor-arg…/>
    - Design the class the way you would have if you weren't planning on using Spring. There may not even be a zero-arg constructor in that case.
    - Use constructor-arg only for immutable classes that have no setter methods. Use property otherwise.
    - Pick and choose depending on how simple it is.

Constructor Args: Problems

- **Issues**
  - Spring doesn't always pass args in the order they are listed
    - For instance, for public Test(int n, String s), you could do

```
<bean id="bean-name" class="package.Test">
  <constructor-arg value="Hello"/>
  <constructor-arg value="3"/>
</bean>
```

Listed 1st, but will be passed as 2nd arg to constructor.
Listed 2nd, but will be passed as 1st arg to constructor.

  - Spring does type conversion
    - For instance, the following could match a constructor that takes a String, an int, or a double
      - <constructor-arg value="5"/>
    - What if there are multiple constructors?
- **Resulting problem**
  - There is often ambiguity about which constructor you mean, and which args go in which order
    - This is one reason some developers prefer <property…/>

Constructor Args: Solutions

- **If there are multiple constructor arguments with compatible types**
  - Use index to specify which argument is which
    - For instance, for public Test(int n, String s), you could do

```
<bean id="bean-name" class="package.Test">
  <constructor-arg value="3" index="0"/>
  <constructor-arg value="4" index="1"/>
</bean>
```

- **If there are multiple constructors with same number of arguments**
  - Use both index and type

```
<constructor-arg value="3"
                 index="0" type="int"/>
<constructor-arg value="4"
                 index="1" type="java.lang.String"/>
```

Constructor Args: Fancy Features (Covered Later)

- **ref: to refer to bean declared earlier**

```
<bean id="bean1" …>…</bean>
<bean id="bean2" …>
  <constructor-arg ref="bean1"/>
</bean>
```

- **Nested <bean>: to supply new bean as value**

```
< constructor-arg>
  <bean…>…</bean>
</ constructor-arg>
```

- **list: to pass a List to the constructor**

```
< constructor-arg>
  <list>…</list>
</ constructor-arg>
```

- **map: to pass a Map to the constructor**

```
< constructor-arg>
  <map>…</map>
</ constructor-arg>
```


Example: Shapes

- **Goal**
  - Test out various geometric calculations. Top-level code should not change with types of shapes change.
- **Approach**
  - Abstract class: Shape
    - abstract getArea method, concrete printInfo method
  - Concrete classes
    - Rectangle, Circle, RightTriangle
  - Bean definition file

```
<bean id="shape2" class="coreservlets.Circle">
  <constructor-arg value="10"/>
</bean>
```

  - Driver class
    - Gets application context, calls getBean, casts to Shape

Abstract and Concrete Classes

- **Abstract class: Shape**
  - Shown in previous section. Has abstract getArea and concrete printInfo methods.
- **Concrete classes**
  - Shown in previous section.
    - Rectangle, Circle, RightTriangle
  - Circle takes a double (the radius) as a constructor argument.

Bean Definition File

```
<?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.xsd">

  <bean id="shape1" class="coreservlets.Rectangle">
    <property name="length" value="10"/>
    <property name="width" value="20"/>
  </bean>
  <bean id="shape2" class="coreservlets.Circle">
    <constructor-arg value="10"/>
  </bean>
  …
</beans>
```

Driver Class (Instantiate Container and Get Instances)

```
package coreservlets;

import org.springframework.context.*;
import org.springframework.context.support.*;

public class ShapeTest {
  public static void main(String[] args) {
    ApplicationContext context =
      new ClassPathXmlApplicationContext
                              ("/applicationContext.xml");
    for(int i=1; i<=4; i++) {
      Shape shape = (Shape)context.getBean("shape" + i);
      shape.printInfo();
    }
    …
  }
}
```


Output

- **Executing in Eclipse**
  - Right-click inside main and choose Run As → Java Application
- **Output for values "shape1" and "shape2"**

```
Informational messages about container starting
…
Rectangle with area of 200.00
Circle with area of 314.16
```

Factory Methods

Factory Methods: Basics

- **<bean … factory-method="makeBean">**
  - Sometimes you don't know the specific type of bean you will need. You need to run some logic to determine this. So, instead of calling constructor, you call a method that returns an object.

```
<bean id="some-name" class="package.HelperClass"
      factory-method="makeSomeBean">
</bean>
```

  - This means that when getBean is called, the static method HelperClass.makeSomeBean() is invoked (with no arguments), and the output of that method is the bean.
    - Note that "class" is not the class of the bean, but rather of the helper class that contains the static factory method.
    - There are also instance factory methods, but are less common so won't be shown here.

Factory Methods: Arguments

- **You use the mis-named "constructor-arg" to supply values to the factory method**

```
<bean id="some-name" class="package.HelperClass"
        factory-method="makeSomeBean">
  <constructor-arg value="Hola"/>
</bean>
```

  - This means that the static method call HelperClass.makeSomeBean("Hola") determines the output of getBean.
    - Note that no constructor is called directly (although the static method might use a constructor internally). So "constructor-arg" is a bad choice of name.
    - Again, "class" is not the type of the bean, but the name of the class that contains the static method.

Example: Shapes

- **Goal**
  - Test out various geometric calculations as before.
- **Approach**
  - Abstract class: Shape
    - abstract getArea method, concrete printInfo method
  - Concrete classes
    - Rectangle, Circle, RightTriangle
  - Bean definition file

```
<bean id="shape3" class="coreservlets.ShapeMaker"
        factory-method="randomShape1">
</bean>
<bean id="shape4" class="coreservlets.ShapeMaker"
        factory-method="randomShape2">
  <constructor-arg value="100"/>
</bean>
```

  - Driver class
    - Gets application context, calls getBean, casts to Shape

Abstract and Concrete Classes

- **Abstract class: Shape**
  - Shown in previous section. Has abstract getArea and concrete printInfo methods.
- **Concrete classes**
  - Shown in previous section.
    - Rectangle, Circle, RightTriangle

Helper Class (with Factory Methods)

```
public class ShapeMaker {
  public static Shape randomShape1() {
    return(randomShape2(10));
  }

  public static Shape randomShape2(double size) {
    double d = Math.random();
    if (d < 0.333) {
      return(new Circle(size));
    } else if (d < 0.666) {
      return(new Rectangle(size, size*2));
    } else {
      return(new RightTriangle(size, size*2));
    }
  }
}
```

Bean Definition File

```
<?xml version="1.0" encoding="UTF-8"?>
<beans …>
  <bean id="shape1" class="coreservlets.Rectangle">
    <property name="length" value="10"/>
    <property name="width" value="20"/>
  </bean>
  <bean id="shape2" class="coreservlets.Circle">
    <constructor-arg value="10"/>
  </bean>
  <bean id="shape3" class="coreservlets.ShapeMaker"
        factory-method="randomShape1">
  </bean>
  <bean id="shape4" class="coreservlets.ShapeMaker"
        factory-method="randomShape2">
    <constructor-arg value="100"/>
  </bean>
  …
</beans>
```


Driver Class (Instantiate Container and Get Instances)

```
package coreservlets;

import org.springframework.context.*;
import org.springframework.context.support.*;

public class ShapeTest {
  public static void main(String[] args) {
    ApplicationContext context =
      new ClassPathXmlApplicationContext
                              ("/applicationContext.xml");
    for(int i=1; i<=4; i++) {
      Shape shape = (Shape)context.getBean("shape" + i);
      shape.printInfo();
    }
    …
  }
}
```

Output

- **Executing in Eclipse**
  - Right-click inside main and choose Run As → Java Application
- **Output for values "shape1" … "shape4"**

```
Informational messages about container starting
…
Rectangle with area of 200.00
Circle with area of 314.16
RightTriangle with area of 100.00
Circle with area of 31,415.93
```


Dependency Injection

Supplying Other Beans as Properties or Constructor Args

Dependency Injection: Basics

- **<property … ref="existing-bean-name"/>**
- **<property><bean…/></property>**
  - Also applies to constructor-arg
- **Idea**
  - Suppose that you have a class that performs operations on Shapes. You don't want the code to depend on any particular Shape or collection of Shapes.
  - So, the main class should never call "new" on a particular Shape subclass. Instead, the bean definition file should create the objects that the main class depends on (i.e., the "dependencies"), and pass them into ("inject" them) the main class via <property…/> or <constructor-arg…/>

Dependency Injection: Passing in Collections

- **You can also supply a List or Map of beans**
  - <property name="propName">
    <list>
    <ref local="existing-bean-name"/>
    <bean…>…</bean>
    …
    </list>
    </property>
  - <property name="propName">
    <map>
    <entry key="…"><bean-or-ref…/></entry>
    …
    </map>
    </property>

Example: ShapeList

- **Goals**
  - Have a class that can take one or more shapes and perform the following operations
    - Find the smallest Shape
    - Find the largest Shape
    - Find the sum of the areas of all the shapes
  - Be able to test this class with various different shapes, without changing the Java code
- **Approach**
  - Instantiate the ShapeList from bean definition file
  - Supply a single Shape or a List of Shapes

ShapeList

```
public class ShapeList {
  private List<Shape> shapes;

  public ShapeList(Shape shape) {
    shapes = Arrays.asList(shape);
  }

  public ShapeList(List<Shape> shapes) {
    this.shapes = shapes;
  }

  public List<Shape> getShapes() {
    return shapes;
  }

  public void setShapes(List<Shape> shapes) {
    this.shapes = shapes;
  }
```

To avoid tying ShapeList to any particular Shape or set of Shapes, these dependencies will be injected (passed in) via the bean definition file.

ShapeList (Continued)

```
public double getTotalArea() {
   double total = 0.0;
   for(Shape shape: shapes) {
     total = total + shape.getArea();
   }
   return(total);
 }

 public Shape getSmallestShape() {
   Shape smallestShape = null;
   double smallestArea = Double.MAX_VALUE;
   for(Shape shape: shapes) {
     double area = shape.getArea();
     if (area < smallestArea) {
       smallestArea = area;
       smallestShape = shape;
     }
   }
   return(smallestShape);
 }
```


ShapeList (Continued)

```
public Shape getBiggestShape() {
   Shape biggestShape = null;
   double biggestArea = 0;
   for(Shape shape: shapes) {
     double area = shape.getArea();
     if (area > biggestArea) {
       biggestArea = area;
       biggestShape = shape;
     }
   }
   return(biggestShape);
 }
 public void printInfo() {
   System.out.printf("ShapeList has %s entries%n",
                     shapes.size());
   System.out.printf("  Smallest: ");
   getSmallestShape().printInfo();
   System.out.printf("  Biggest: ");
   getBiggestShape().printInfo();
   System.out.printf("  Total area: %,.2f%n", getTotalArea());
 }
```

Dependencies (Classes Used by ShapeList)

- **Abstract class: Shape**
  - Shown in previous section. Has abstract getArea and concrete printInfo methods.
- **Concrete classes**
  - Shown in previous sections.
    - Rectangle, Circle, RightTriangle

Bean Definition File

```
<?xml version="1.0" encoding="UTF-8"?>
<beans …>
 …
 <bean id="shapeList1" class="coreservlets.ShapeList">
    <constructor-arg ref="shape1"/>
  </bean>
  <bean id="shapeList2" class="coreservlets.ShapeList">
    <constructor-arg>
      <list>
        <ref local="shape2"/>
        <bean class="coreservlets.RightTriangle">
          <property name="sideA" value="5"/>
          <property name="sideB" value="10"/>
        </bean>
        <bean class="coreservlets.Circle">
          <constructor-arg value="25"/>
        </bean>
      </list>
    </constructor-arg>
  </bean>  …
</beans>
```

Shape names defined earlier in this file.

New, un-named shapes. Sometimes called "inner beans".

Driver Class (Instantiate Container and Get Instances)

```
import org.springframework.context.*;
import org.springframework.context.support.*;

public class ShapeTest {
  public static void main(String[] args) {
    ApplicationContext context =
      new ClassPathXmlApplicationContext
                              ("/applicationContext.xml");
    for(int i=1; i<=4; i++) {
      Shape shape = (Shape)context.getBean("shape" + i);
      shape.printInfo();
    }
    for(int i=1; i<=2; i++) {
      ShapeList shapes =
        (ShapeList)context.getBean("shapeList" + i);
      shapes.printInfo();
    }
  }
}
```


Output

- **Executing in Eclipse**
  - Right-click and choose Run As → Java Application
- **Output**

```
Informational messages about container starting …
Rectangle with area of 200.00
Circle with area of 314.16
RightTriangle with area of 100.00
Circle with area of 31,415.93
ShapeList has 1 entries
  Smallest: Rectangle with area of 200.00
  Biggest: Rectangle with area of 200.00
  Total area: 200.00
ShapeList has 3 entries
  Smallest: RightTriangle with area of 25.00
  Biggest: Circle with area of 1,963.50
  Total area: 2,302.65
```

Bean Scopes

Basics

- **<bean … scope="singleton"> (or no scope)**
  - Every time you call getBean on the same name, you get the same instance. Default if no scope specified.
    - <property…/> and <constructor-arg…/> only invoked the first time (if bean of that name not already instantiated)
    - You get the same instance per container. If you re-instantiate the container, then you get new instance
- **<bean … scope="prototype">**
  - Every time you call getBean, you get new instance
    - <property> and <constructor-arg> invoked every time
- **Other scopes**
  - scope="request" and scope="session"
    - Valid only in Web apps. See next tutorial section.
  - scope="globalSession"
    - Valid only in portal apps.

Bean Definition File

```
<?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.xsd">

  <!--  Default scope is singleton -->
  <bean id="rectangle1" class="coreservlets.Rectangle">
    <property name="length" value="5"/>
    <property name="width" value="10"/>
  </bean>

  <bean id="rectangle2" class="coreservlets.Rectangle"
        scope="prototype">
    <property name="length" value="5"/>
    <property name="width" value="10"/>
  </bean>

</beans>
```


Driver Class

```
public class ScopeTest {
  public static void main(String[] args) {
    System.out.println("Singleton beans from same context");
    singletonTest1("rectangle1");
    System.out.println("Prototype beans from same context");
    singletonTest1("rectangle2");
    System.out.println
                ("Singleton beans from different contexts");
    singletonTest2("rectangle1");
  }
```

Driver Class (Continued)

```
public static void compareRectangles(Rectangle r1,
                                      Rectangle r2) {
   System.out.print("  r1: ");
   r1.printInfo();
   System.out.print("  r2: ");
   r2.printInfo();
   System.out.printf("  r1 == r2: %s%n", r1 == r2);
 }
```


Driver Class (Continued)

```
public static void singletonTest1(String beanName) {
   ApplicationContext context =
     new ClassPathXmlApplicationContext
                          ("/scope-test.xml");
   Rectangle r1 =
     (Rectangle)context.getBean(beanName);
   Rectangle r2 =
     (Rectangle)context.getBean(beanName);
   compareRectangles(r1, r2);
   r1.setLength(50);
   compareRectangles(r1, r2);
 }
```

Driver Class (Continued)

```
public static void singletonTest2(String beanName) {
   ApplicationContext context1 =
     new ClassPathXmlApplicationContext
                          ("/scope-test.xml");
   Rectangle r1 =
     (Rectangle)context1.getBean(beanName);
   ApplicationContext context2 =
     new ClassPathXmlApplicationContext
                          ("/scope-test.xml");
   Rectangle r2 =
     (Rectangle)context2.getBean(beanName);
   compareRectangles(r1, r2);
   r1.setLength(50);
   compareRectangles(r1, r2);
 }
```


Output 1

- **Singleton scope, one container instance**

```
Singleton beans from same context
  r1: Rectangle with area of 50.00
  r2: Rectangle with area of 50.00
  r1 == r2: true
  r1: Rectangle with area of 500.00
  r2: Rectangle with area of 500.00
  r1 == r2: true
```

Output 2

- Prototype scope, one container instance

```
Prototype beans from same context
  r1: Rectangle with area of 50.00
  r2: Rectangle with area of 50.00
  r1 == r2: false
  r1: Rectangle with area of 500.00
  r2: Rectangle with area of 50.00
  r1 == r2: false
```


Output 3

- Singleton scope, two container instances

```
Singleton beans from different contexts
  r1: Rectangle with area of 50.00
  r2: Rectangle with area of 50.00
  r1 == r2: false
  r1: Rectangle with area of 500.00
  r2: Rectangle with area of 50.00
  r1 == r2: false
```

Wrap-up

Summary

- **Java classes**
  - Make interface or abstract class
  - Make concrete implementations
- **Bean definition file**
  - Use <bean> to declare objects
    - Always use name and class
    - Sometimes use scope or factory-method
  - Use <property> or <constructor-arg> for init values
    - Simple values
    - ref to refer to previously-defined beans
    - Nested <bean> definitions
    - <list> or <map> (containing any of the above)
- **Driver class**
  - Make new ClassPathXmlApplicationContext
  - Call getBean

Questions?