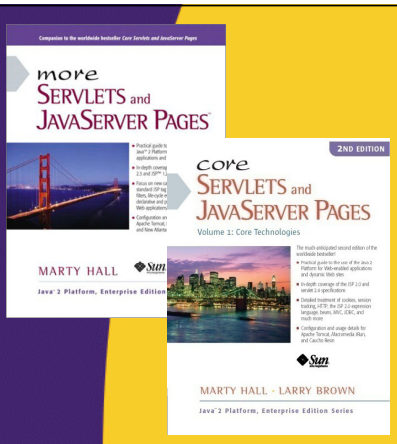
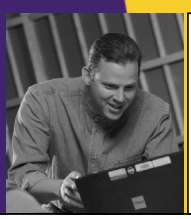
The Spring Framework: Overview and Setup

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

© 2009 Marty Hall





Agenda

- **What is Spring?**
  - And why use it?
- **Main Spring modules**
  - Dependency injection
  - AOP
- **Configuring apps to use Spring**
  - And Eclipse plugin support
- **Simple example**

Overview of Spring

What is Spring?

- **Spring is a framework with many modules**
  - To simplify many different Java EE tasks
- **Core module: dependency injection**
  - Lets you define an XML file that specifies which beans are created and what their properties are
    - Simplifies OOP by promoting loose coupling between classes. Also called "Inversion of Control" (IoC)
    - Small example shown in this lecture; details in next lecture
- **Second most important module: AOP**
  - "Aspect Oriented Programming" refers to the ability to add side effects to a class' method calls without modifying the class' source code
  - Lets you separate business logic from system services

Why Spring?

- **Disenchantment with EJB**
  - Too complicated
  - Objects were technology-specific: tied to EJB
  - Hard to test in isolation
- **Basic Spring philosophy**
  - Avoid tight coupling among classes
- **Approaches to support this philosophy**
  - Use POJOs (Plain Old Java Objects)
  - Add enterprise services declaratively
  - Use Spring to obtain object instances and to manage their lifecyle. Don't make Spring-dependent classes.
    - However, annotations partially violate this principle

Spring Modules

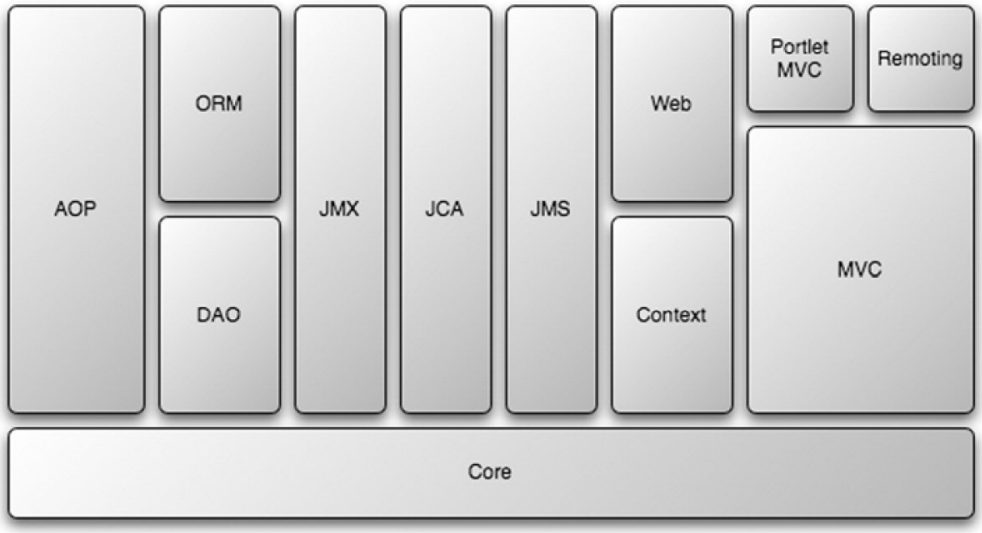
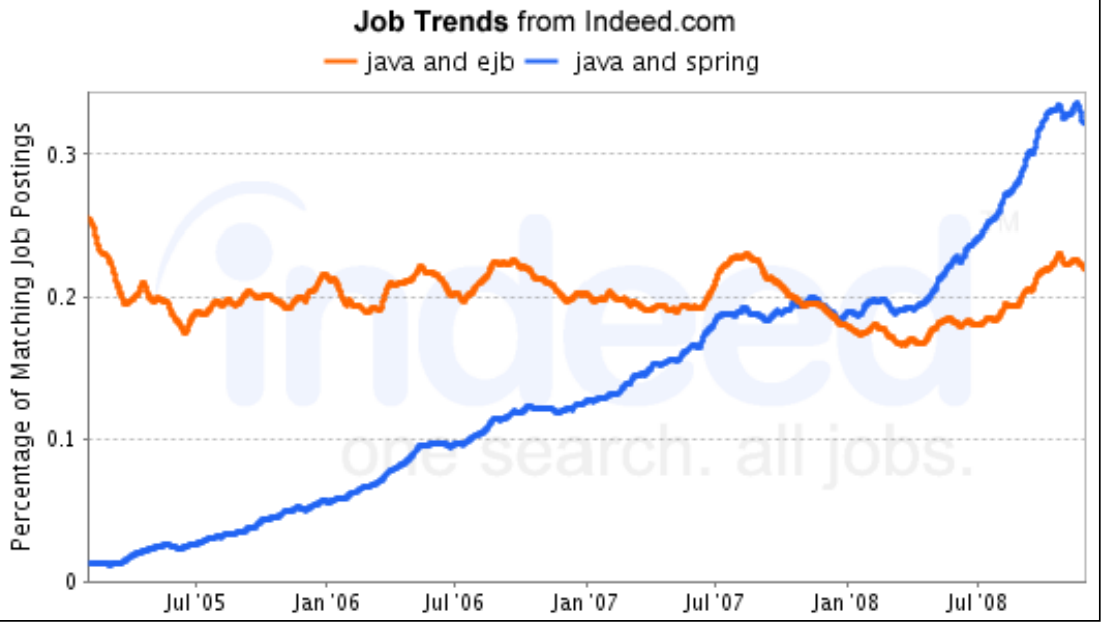


Figure 1.1 The Spring Framework is composed of several well-defined modules built on top of the core container. This modularity makes it possible to use as much or as little of the Spring Framework as is needed in a particular application.

From *Spring in Action, 2nd Edition* by Craig Walls

Use of Spring in Industry

- Claims to compile data from most major jobs sites
  - Data through 12/2008



Installation and Setup

Downloading

- **http://www.springsource.org/download**
  - Choose latest released version
    - 2.5.6 as of 12/2008
  - Choose "Community Download"
    - This is free and open source
    - There is also Enterprise version that comes with paid support
  - Choose version with dependencies
    - Includes required and optional JAR files, documentation, and samples
      - You only have to use the JAR files you want, so there is no harm in downloading the full version
    - spring-framework-2.5.6-with-dependencies.zip as of 12/08
  - Alternative: download spring-blank.zip
    - Eclipse project with two required JAR files, blank applicationContext.xml, and Eclipse Spring IDE nature
    - From coreservlets.com; see link on title page of this tutorial.

Installation

- **Unzip into directory of your choice**
  - I will refer to this later as *spring-install*
- **To add Spring capabilities to projects**
  - Simple Java project
    - Add two JAR files to project class path
      - *spring-install*/dist/spring.jar
      - *spring-install*/lib/jakarta-commons/commons-logging.jar
    - Create empty bean definition file to use as a starting point
      - Usually placed in top-level of class path: src/applicationContext.xml
    - Simple example given in this lecture
  - Dynamic Web Project
    - Add two JAR files to WEB-INF/lib
      - *spring-install*/dist/spring.jar
      - *spring-install*/lib/jakarta-commons/commons-logging.jar
    - Place starting-point bean definition file in WEB-INF
    - Declare listener that loads the bean definition file at app startup
    - Simple example given in next lecture

Documentation

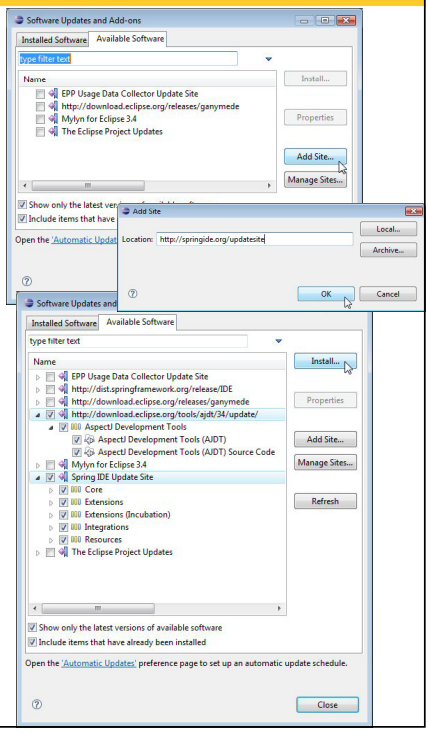
- **Online docs**
  - Reference documentation
    - http://static.springframework.org/spring/docs/2.5.x/reference/
  - API in JavaDoc
    - http://static.springframework.org/spring/docs/2.5.x/api/
  - Tutorials and setup guides
    - http://opensource.atlassian.com/confluence/spring/dashboard.action
  - Documentation for Spring Eclipse plugin
    - http://springide.org/project/wiki/TOC
- **Books**
  - *Spring Recipes* by Gary Mak (APress)
    - Covers Spring 2.5
  - *Spring in Action* by Craig Walls (Manning)
    - Covers Spring 2.0

IDE Support for Spring

- **Spring IDE**
  - Free, open source, Eclipse only. Used in these tutorials.
    - http://springide.org/
- **Skyway Builder**
  - Commercial Eclipse plugin for Spring and Hibernate
    - http://www.skywayperspectives.org/
- **Others for Eclipse**
  - Search Eclipse Plugin Central. Several new ones listed.
    - http://www.eclipseplugincentral.com/
- **Other IDEs**
  - MyEclipse
    - Has their own Spring support. http://myeclipseide.com/
  - Intellij IDEA
    - Has their own Spring support. http://www.jetbrains.com/idea/

Spring IDE: Eclipse Plugin for Spring

- **Idea**
  - Optional plugin adds many useful Spring-aware features to Eclipse.
  - Highly recommended for Eclipse users.
- **Installation**
  - Help → Software Updates → Available Software → Add Site
  - Enter http://springide.org/updatesite
  - Select both the springide.org site *and* the automatically-created site for AspectJ support
  - Click Install



Making a New Spring Project

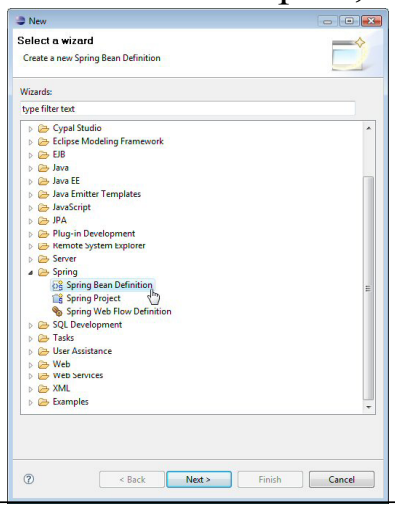
1. **Make new Java project**
   - New → Project → Java → Java Project
2. **Add Spring IDE Eclipse support**
   - R-click project, Spring Tools → Add Spring Project Nature
     - For regular Java projects (not Dynamic Web Projects) you can combine the above two steps by doing New → Project → Spring → Spring Project.
     - You can still use Spring even if you don't have Spring IDE. Adding the project nature just makes Eclipse smarter about editing and graphically displaying certain files.
3. **Add spring.jar and commons-logging.jar**
   - Details on next slide
4. **Put an empty bean definition file in src**
   - Details on upcoming slide

Adding JAR Files to Project

- **Required JAR files**
  - spring.jar (from "dist" folder)
    - If disk space is critical, you can choose among many smaller JAR files. But simplest to just use spring.jar.
  - commons-logging.jar (from "lib/jakarta-commons")
- **Putting JAR files in class path**
  - Make a new project directory (e.g., "lib")
  - Copy *spring-install*/dist/spring.jar and *spring-install*/lib/jakarta-commons/commons-logging.jar to the lib folder.
  - R-click on project, Properties → Libraries → Add JARs
    - Then point at lib/spring.jar and lib/commons-logging.jar
- **For Dynamic Web Projects**
  - Instead of above steps, copy JARs to WEB-INF/lib

Making a Bean Definition File

- **Idea**
  - XML file defines objects (beans) and gives them names. Java code will load file and refer to objects by name.
  - File is most commonly loaded relative to the class path, so is typically placed in src folder
  - File can have any name
    - Common choices are beans.xml and applicationContext.xml
- **Creating with Spring IDE**
  - R-click on src folder → New → Other → Spring → Spring Bean Definition
  - Choose a name and hit "Finish"



Sample Bean Definition File

- **Starting point 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">
  <!-- Your entries will go here -->
</beans>
```

- **Spring IDE is not required**
  - This file can be created and edited manually
    - Grab one from the samples that come with Spring
      - Make sure it does not refer to fancy features you are not using yet
      - Delete everything between <beans …> and </beans>
    - Or, take the empty one from spring-blank

The spring-blank Project

- **Preconfigured Eclipse project**
  - Standard Java project with Spring Project Nature
    - But works with or without Spring IDE plugin
  - spring.jar and commons-logging.jar in lib folder
    - And with those two JARs added to build path
  - Empty applicationContext.xml file in src folder
- **Usage**
  - R-click project, Copy
  - R-click in Project Explorer, Paste, give new name
- **Download from coreservlets.com**
  - http://www.coreservlets.com/Spring-Tutorial/
  - Import into Eclipse with File → Import → General → Existing Projects into Workspace

Simple Spring Example

Goal: Loose Coupling

- **Geometric analysis**
  - You have a program that computes areas for various collections of shapes. You want to avoid code that depends on any particular type of shape, since the varieties of shapes keep changing.
- **Strategy**
  - Define interface or abstract class
    - No dependencies on Spring
  - Make concrete implementations of the interface
    - No dependencies on Spring
  - Declare specific concrete object in bean definition file
  - Load bean definition file and get instance (driver class)
    - No dependency on specific concrete type

Making Spring Project

- **From scratch**
  - File → New → Project → Spring → Spring Project
    - Or, if no Spring IDE, File → New → Java → Java Project
  - Named project spring-intro
  - 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-intro

Interface

```
package coreservlets;

public interface Shape {
  public double getArea();
}
```

- **Notes**
  - No imports of Spring packages
    - No ties in *any* way to Spring
  - Code that only uses the area (or inherited methods like toString) should refer only to Shape
    - But interfaces cannot be used everywhere: code that uses more specific info (e.g., the radius of a Circle that implements Shape) will need to use concrete type

Concrete Class: Rectangle

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

  public Rectangle() {}

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

The driver class will use only getArea and methods inherited from Object, so will refer to Shape, not Rectangle.

The bean definition in the upcoming applicationContext.xml file will not specify any constructor arguments. So, the zero-arg constructor will be used.

Rectangle (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);
  }
}
```

The bean definition in the upcoming applicationContext.xml file will refer to the length and width "properties". That really means to call setLength and setWidth.

Concrete Class: Circle

```
public class Circle implements 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);
  }
}
```

The bean definition in the upcoming applicationContext.xml file will supply a number as a constructor argument. So, this constructor will be used.

Bean Definition File: applicationContext.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.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>
```

Call new Rectangle(), then setLength(10) and setWidth(20). Associate the instance with the name "shape1".

Call new Circle(10). Associate the instance with the name "shape2".

Driver Class

```
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");
    Shape shape1 = (Shape)context.getBean("shape1");
    printInfo(shape1);
    Shape shape2 = (Shape)context.getBean("shape2");
    printInfo(shape2);
  }
```

File is in "src", which is top level of class path.

Names as given in applicationContext.xml.

Driver Class (Continued)

```
 private static void printInfo(Shape shape) {
    System.out.printf("Area of %s is %.2f%n",
                      shape.getClass().getSimpleName(),
                      shape.getArea());
  }
}
```

Since type is declared as Shape, use only methods defined in the interface (getArea) or inherited from Object (getClass).

Output

```
Rectangle with area of 200.00
Circle with area of 314.16
```

Wrap-up

Summary

- **Why Spring?**
  - Spring promotes loose coupling among classes, isolating one class from changes in another.
- **Using Spring in your apps**
  - Add spring.jar and commons-logging.jar to class path
  - Put a bean definition file in your class path
- **Approach**
  - Define interface or abstract class
  - Make concrete implementations of the interface
  - Declare specific concrete object in bean definition file
    - <bean id="someName" class="package.Class">…</bean>
      - Use property or constructor-arg
  - Load bean definition file and get instance

Questions?