Принципы проектирования и дизайна ПО

Лекция №8

Агошков Илья 2016

```
  name_label = TkLabel.new() {text "What is Your
Name?"}
  name_label.pack
  name = TkEntry.new(root).pack
  name.bind("FocusOut") {process_name(name)}
  quest_label = TkLabel.new() {text "What is Your
Quest?"}
  quest_label.pack
  quest = TkEntry.new(root).pack
  quest.bind("FocusOut") {process_quest(quest)}
  Tk.mainloop()
```

Inversion of Control

also known as the

Hollywood Principle

"Don't call us, we'll call you"

class MovieLister...

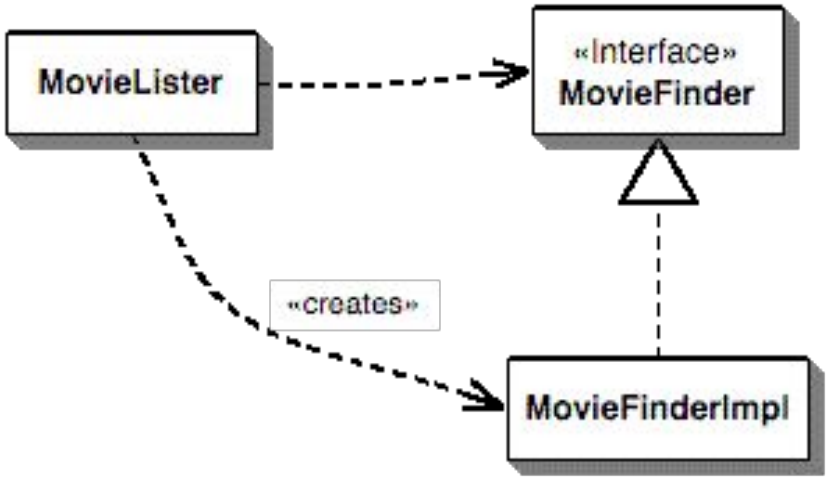
```
 public Collection<Movie> moviesDirectedBy(String director) {
      Collection<Movie> filteredMovies = new ArrayList<>();
      for (Movie m : finder.findAll()) {
          if (m.getDirector().equals(director)) filteredMovies.add(m);
      }
      return filteredMovies;
  }
```

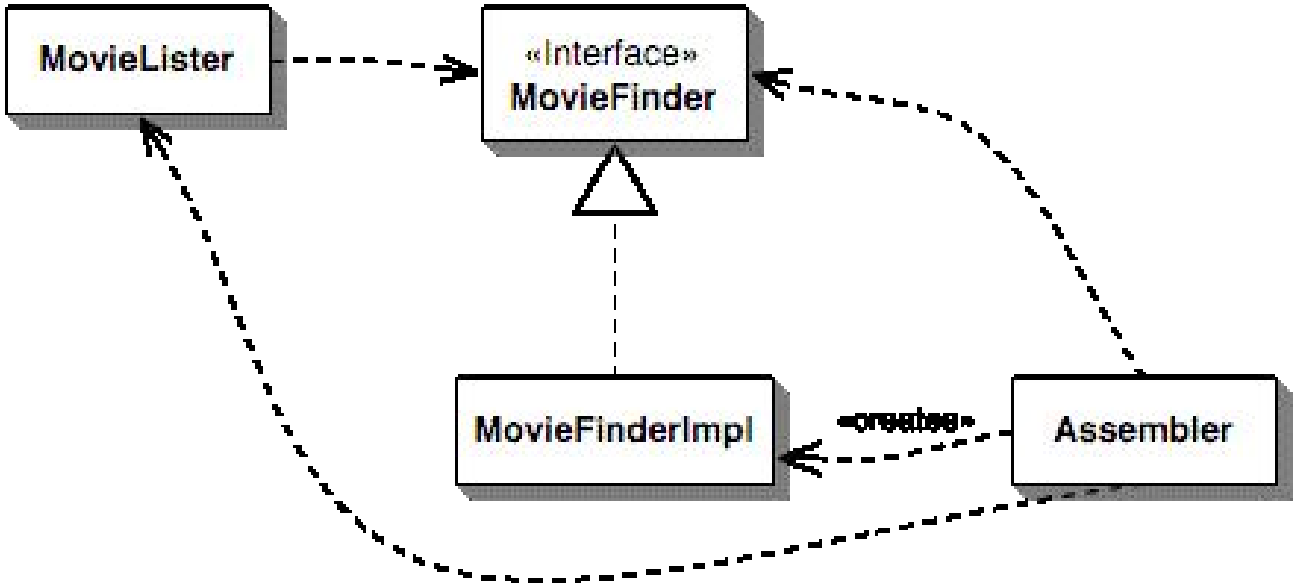
interface MovieFinder

```
public interface MovieFinder {
    List findAll();
}
```

class MovieLister...

```
private MovieFinder finder;
public MovieLister() {
  finder = new ColonDelimitedMovieFinder("movies1.txt");
}
```




MovieLister
«Interface»
MovieFinder
MovieFinderImpl
«creates»
Assembler

PicoContainer

```
private MutablePicoContainer configureContainer() {
    MutablePicoContainer pico = new DefaultPicoContainer();
    Parameter[] finderParams =  {new
ConstantParameter("movies1.txt")};
    pico.registerComponentImplementation(MovieFinder.class,
ColonMovieFinder.class, finderParams);
    pico.registerComponentImplementation(MovieLister.class);
    return pico;
}
```

Service locator

Problems it attempts to solve:

- Resolution of dependencies without calling new in your production code
- Inversion of Control
- Allow for looser coupling
- Interchange wiring for tests

Service locator

- a.k.a. Context
- Better then Singleton
  - if you had static look up of services this is an improvement. It is testable but it is not pretty.
- Hides true dependencies

Avalon

```
public interface Serviceable {
  void service(ServiceManager sm) throws ServiceException;
}

public interface ServiceManager {
  boolean hasService(String key);
  Object lookup(String key) throws ServiceException;
  void release(Object object);
}
```

Avalon

```
public interface Mouse {
  int runReallyFast();
}

public class LittleMouse implements Mouse {
  public LittleMouse() {}
  public int runReallyFast() {
    return (int)((Math.random()*12)+1);
  }
}

public interface Cat {
  public void hunt();
}
```

Avalon

```
public class VeryFastCat implements Serviceable {
  private Mouse jerry;
  private int speed = 10;

  public VeryFastCat() {}

  public void service(ServiceManager sm)
      throws ServiceException {
    jerry = sm.lookup(Mouse.class.getName());
  }

  public void hunt() {
    if (jerry.runRealFast() =< speed)
      System.out.println( "Caught the mouse!" );
    else
```

Spring

hello/MessageService.java

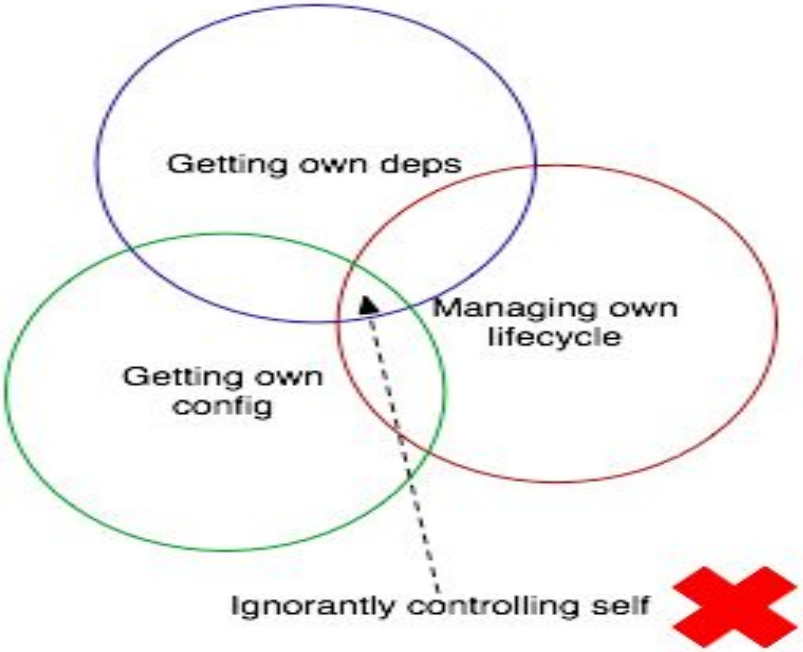
```
package hello;

public interface MessageService {
    String getMessage();
}
```

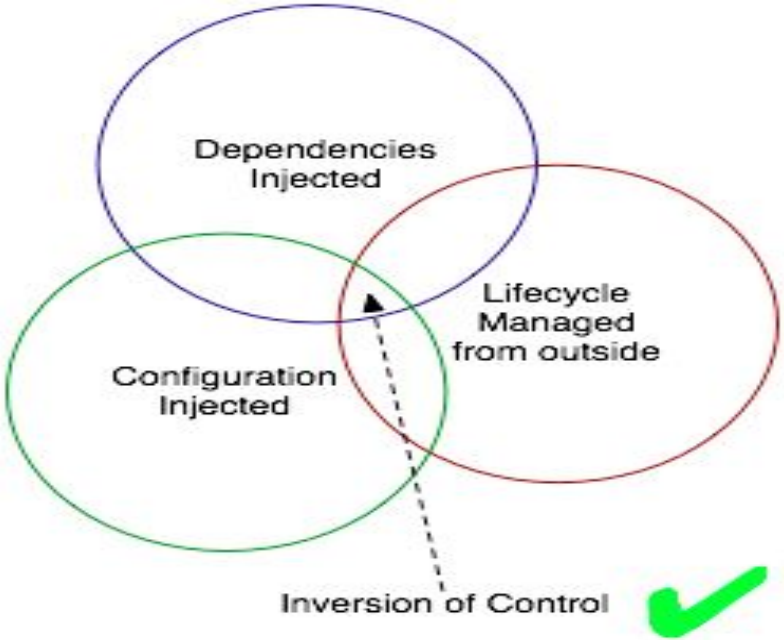
hello/MessagePrinter.java

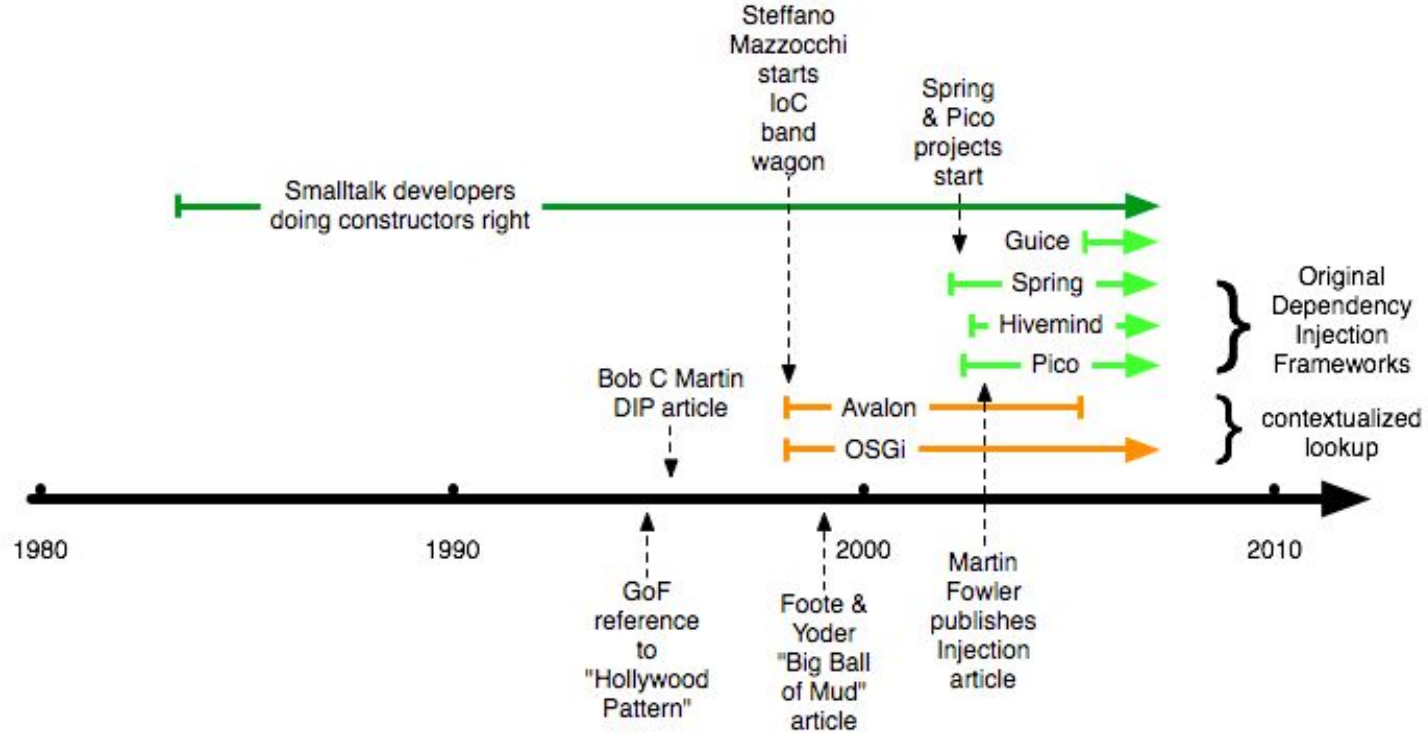
```
package hello;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class MessagePrinter {

    final private MessageService service;

    @Autowired
    public MessagePrinter(MessageService service) {
        this.service = service;
    }

    public void printMessage() {
        System.out.println(this.service.getMessage());
    }
}
```

hello/Application.java

```
package hello;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.*;

@Configuration
@ComponentScan
public class Application {

    @Bean
    MessageService mockMessageService() {
        return new MessageService() {
            public String getMessage() {
              return "Hello World!";
            }
        };
    }

  public static void main(String[] args) {
      ApplicationContext context =
          new AnnotationConfigApplicationContext(Application.class);
      MessagePrinter printer = context.getBean(MessagePrinter.class);
      printer.printMessage();
  }
}
```


Getting own deps
Managing own lifecycle
Getting own config
Ignorantly controlling self


Dependencies Injected
Lifecycle Managed from outside
Configuration Injected
Inversion of Control


Steffano Mazzocchi starts IoC band wagon
Spring & Pico projects start
Smalltalk developers doing constructors right
Guice
Spring
Hivemind
Pico
Original Dependency Injection Frameworks
Bob C Martin DIP article
Avalon
OSGi
contextualized lookup
1980
1990
2000
2010
GoF reference to "Hollywood Pattern"
Foote & Yoder "Big Ball of Mud" article
Martin Fowler publishes Injection article

Spring (xml configuration)

```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">

    <!-- services -->

    <bean id="petStore" class="org.springframework.samples.jpetstore.services.PetStoreServiceImpl">
        <property name="accountDao" ref="accountDao"/>
        <property name="itemDao" ref="itemDao"/>
        <!-- additional collaborators and configuration for this bean go here -->
    </bean>

    <!-- more bean definitions for services go here -->

</beans>
```

```xml
<bean id="exampleBean" class="examples.ExampleBean">
    <!-- setter injection using the nested ref element -->
    <property name="beanOne">
        <ref bean="anotherExampleBean"/>
    </property>

    <!-- setter injection using the neater ref attribute -->
    <property name="beanTwo" ref="yetAnotherBean"/>
    <property name="integerProperty" value="1"/>
</bean>

<bean id="anotherExampleBean" class="examples.AnotherBean"/>
<bean id="yetAnotherBean" class="examples.YetAnotherBean"/>
```

```java
public class ExampleBean {

    private AnotherBean beanOne;
    private YetAnotherBean beanTwo;
    private int i;

    public void setBeanOne(AnotherBean beanOne) {
        this.beanOne = beanOne;
    }

    public void setBeanTwo(YetAnotherBean beanTwo) {
        this.beanTwo = beanTwo;
    }

    public void setIntegerProperty(int i) {
        this.i = i;
    }

}
```

```xml
<bean id="exampleBean" class="examples.ExampleBean">
    <!-- constructor injection using the nested ref element -->
    <constructor-arg>
        <ref bean="anotherExampleBean"/>
    </constructor-arg>

    <!-- constructor injection using the neater ref attribute -->
    <constructor-arg ref="yetAnotherBean"/>

    <constructor-arg type="int" value="1"/>
</bean>

<bean id="anotherExampleBean" class="examples.AnotherBean"/>
<bean id="yetAnotherBean" class="examples.YetAnotherBean"/>
```

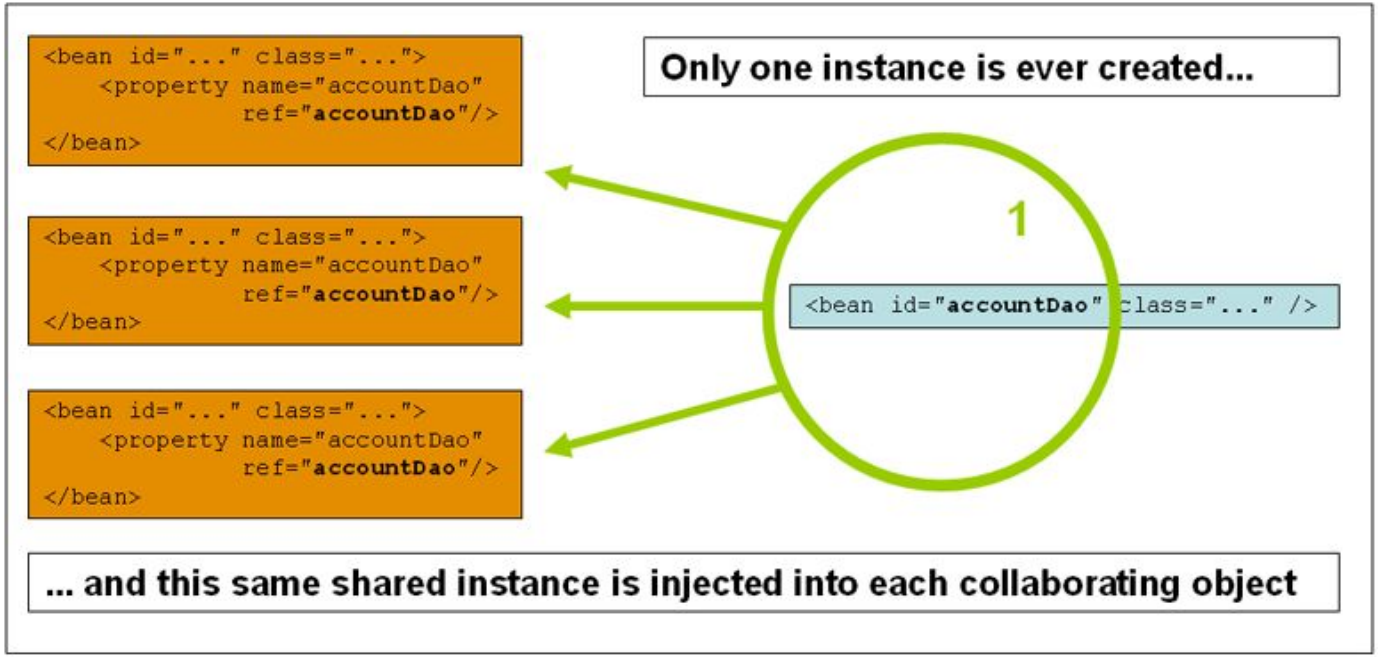
```java
public class ExampleBean {

    private AnotherBean beanOne;
    private YetAnotherBean beanTwo;
    private int i;

    public ExampleBean(
        AnotherBean anotherBean, YetAnotherBean yetAnotherBean, int i) {
        this.beanOne = anotherBean;
        this.beanTwo = yetAnotherBean;
        this.i = i;
    }

}
```

```xml
<bean id="exampleBean" class="examples.ExampleBean" factory-method="createInstance">
    <constructor-arg ref="anotherExampleBean"/>
    <constructor-arg ref="yetAnotherBean"/>
    <constructor-arg value="1"/>
</bean>

<bean id="anotherExampleBean" class="examples.AnotherBean"/>
<bean id="yetAnotherBean" class="examples.YetAnotherBean"/>
```

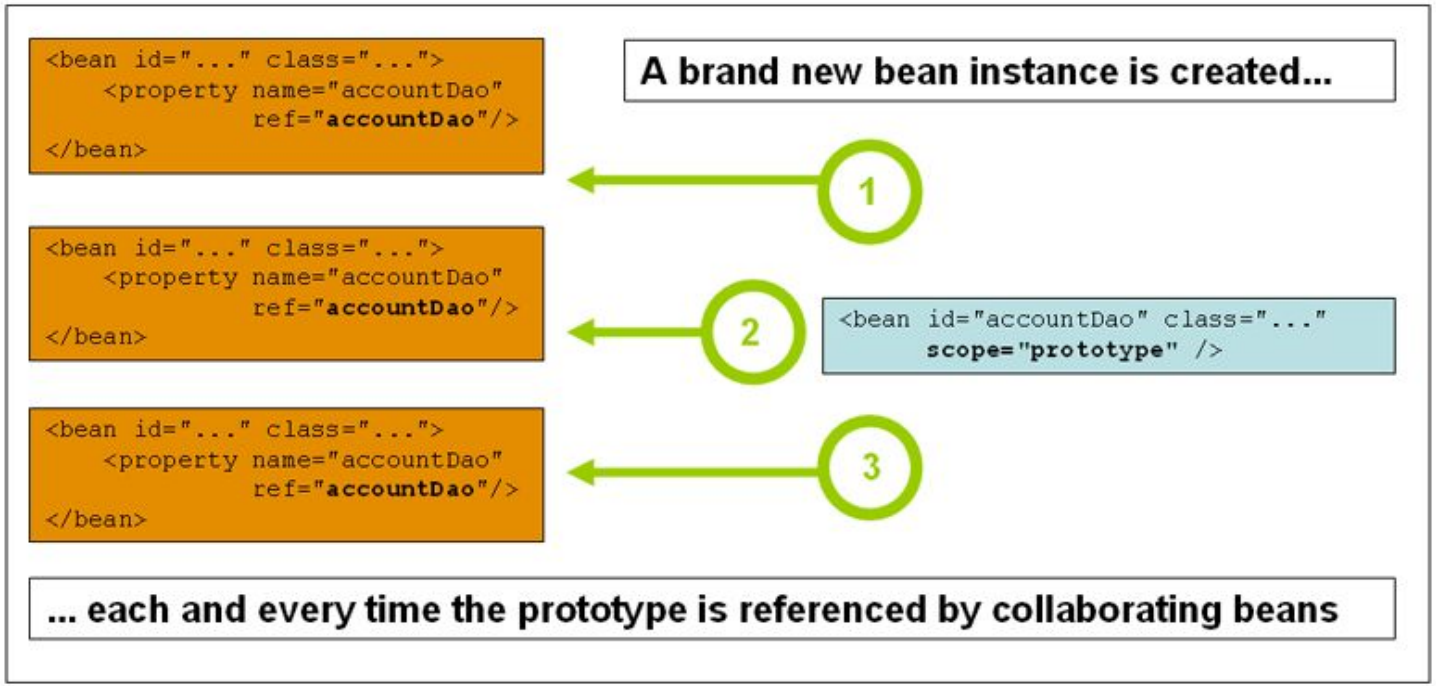
```java
public class ExampleBean {

    // a private constructor
    private ExampleBean(...) {
        ...
    }

    // a static factory method; the arguments to this method can be
    // considered the dependencies of the bean that is returned,
    // regardless of how those arguments are actually used.
    public static ExampleBean createInstance (
        AnotherBean anotherBean, YetAnotherBean yetAnotherBean, int i) {

        ExampleBean eb = new ExampleBean (...);
        // some other operations...
        return eb;
    }

}
```

```xml
<bean id="moreComplexObject" class="example.ComplexObject">
    <!-- results in a setAdminEmails(java.util.Properties) call -->
    <property name="adminEmails">
        <props>
            <prop key="administrator">administrator@example.org</prop>
            <prop key="support">support@example.org</prop>
            <prop key="development">development@example.org</prop>
        </props>
    </property>
    <!-- results in a setSomeList(java.util.List) call -->
    <property name="someList">
        <list>
            <value>a list element followed by a reference</value>
            <ref bean="myDataSource" />
        </list>
    </property>
    <!-- results in a setSomeMap(java.util.Map) call -->
    <property name="someMap">
        <map>
            <entry key="an entry" value="just some string"/>
            <entry key ="a ref" value-ref="myDataSource"/>
        </map>
    </property>
    <!-- results in a setSomeSet(java.util.Set) call -->
    <property name="someSet">
        <set>
            <value>just some string</value>
            <ref bean="myDataSource" />
        </set>
    </property>
</bean>
```

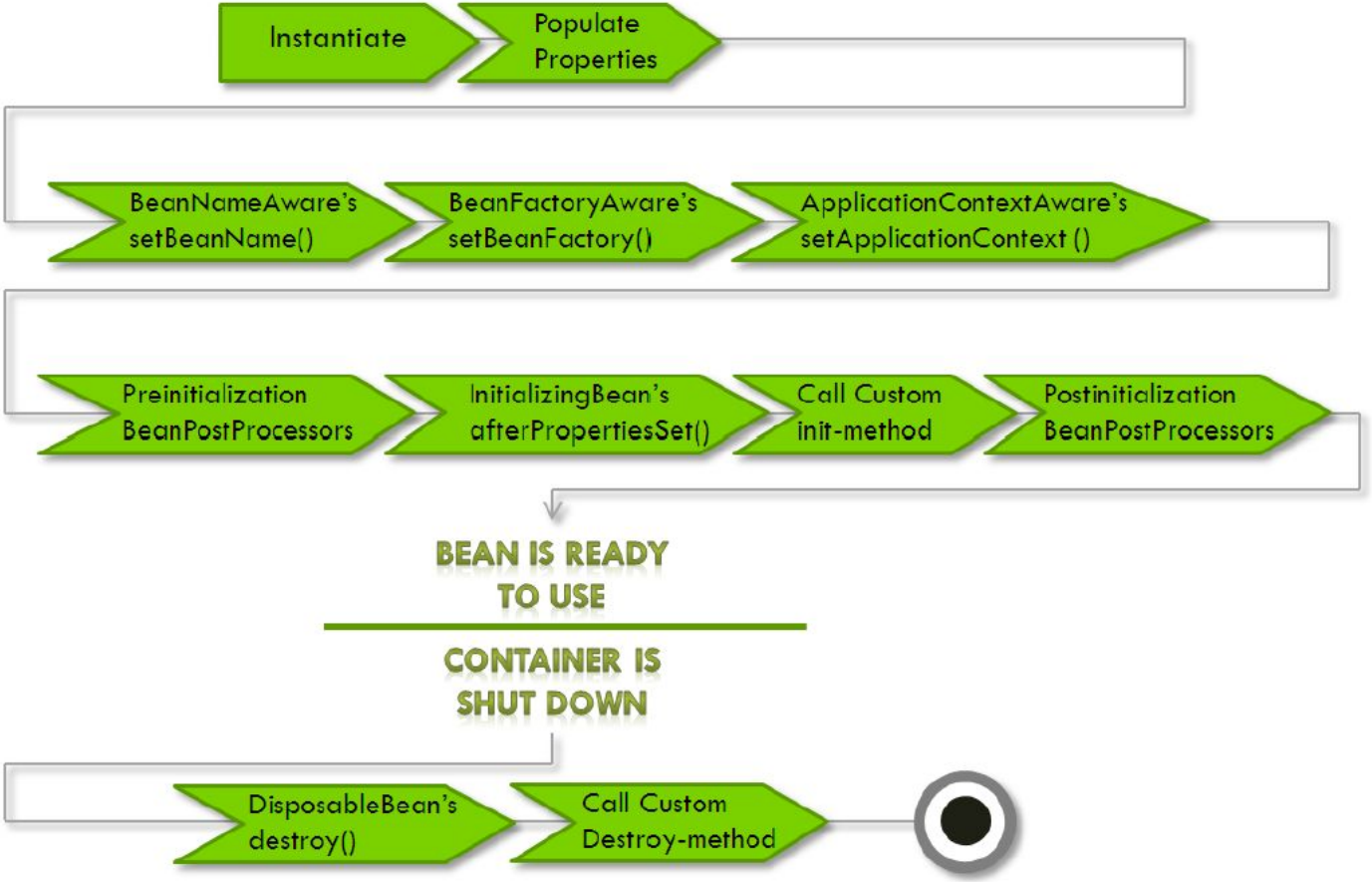
Singleton scope



Prototype scope



Bean lifecycle



More about Spring

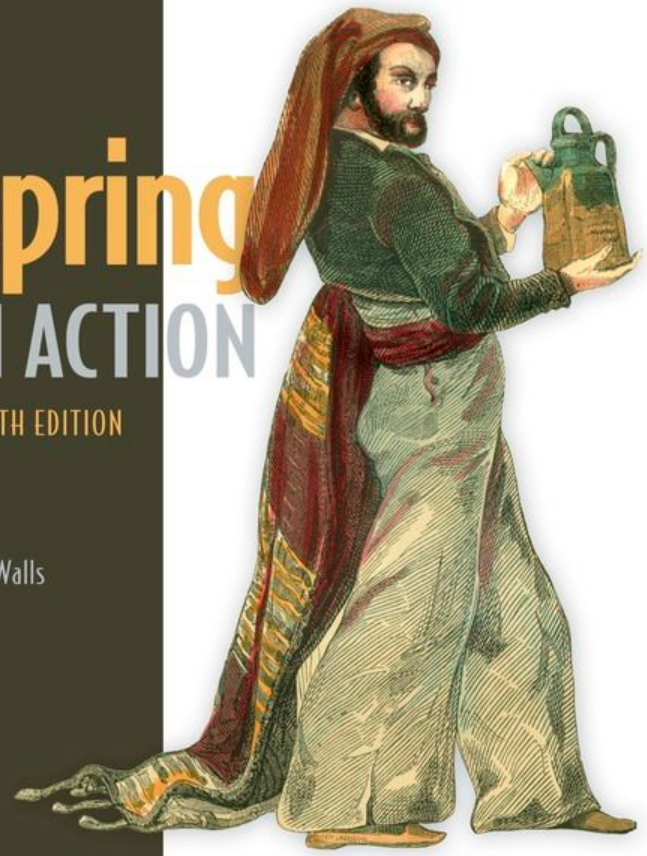
Spring web site:
http://spring.io

Spring beans documentation
http://docs.spring.io/spring/docs/current/spring-framework-reference/html/beans.html

covers Spring 4

Spring IN ACTION

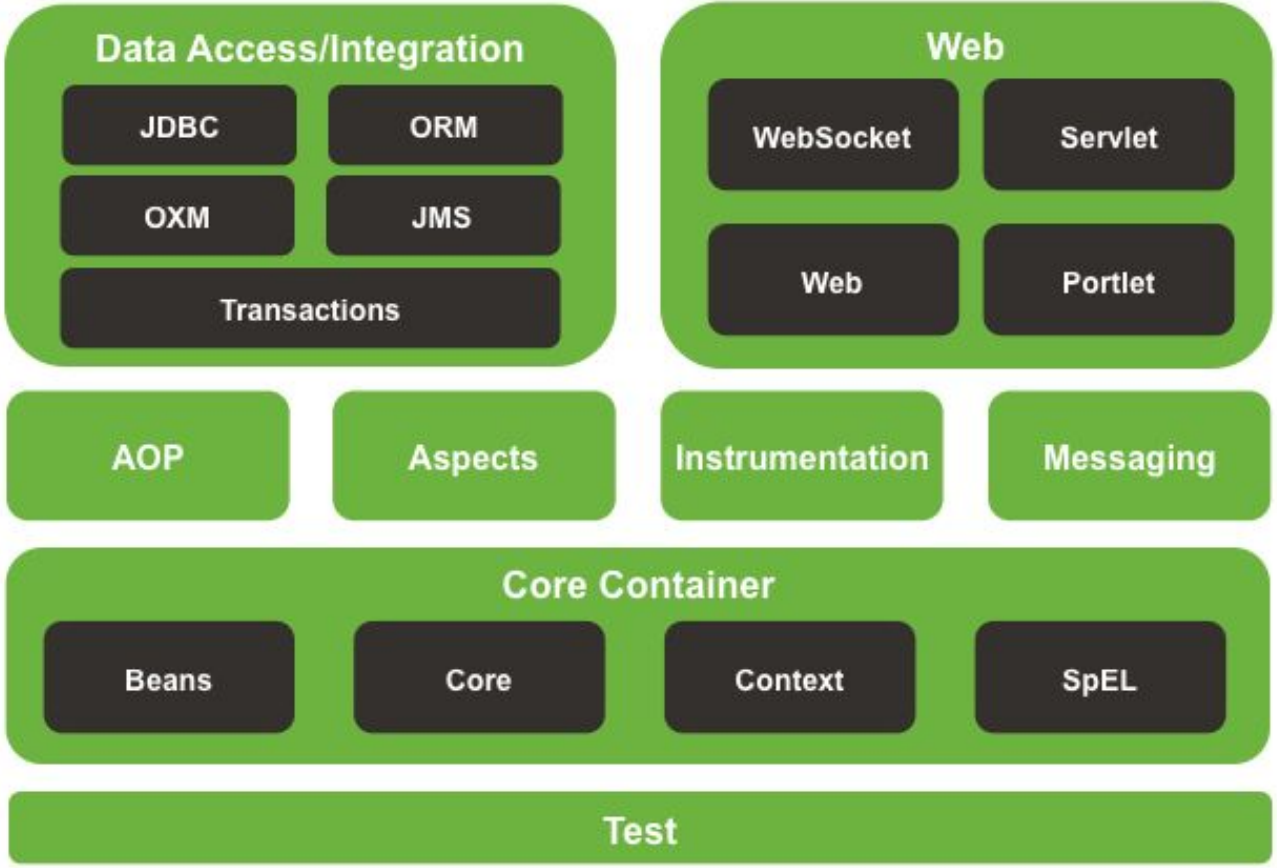
FOURTH EDITION



Craig Walls

MANNING

Spring Framework Runtime



More about DI

http://en.wikipedia.org/wiki/Dependency_injection
http://en.wikipedia.org/wiki/Service_locator_pattern
http://habrahabr.ru/post/166287/

http://www.martinfowler.com/articles/injection.html
http://www.martinfowler.com/bliki/InversionOfControl.html

Dependency Inversion Principle + Inversion of Control + Dependency Injection
http://www.martinfowler.com/articles/dipInTheWild.html