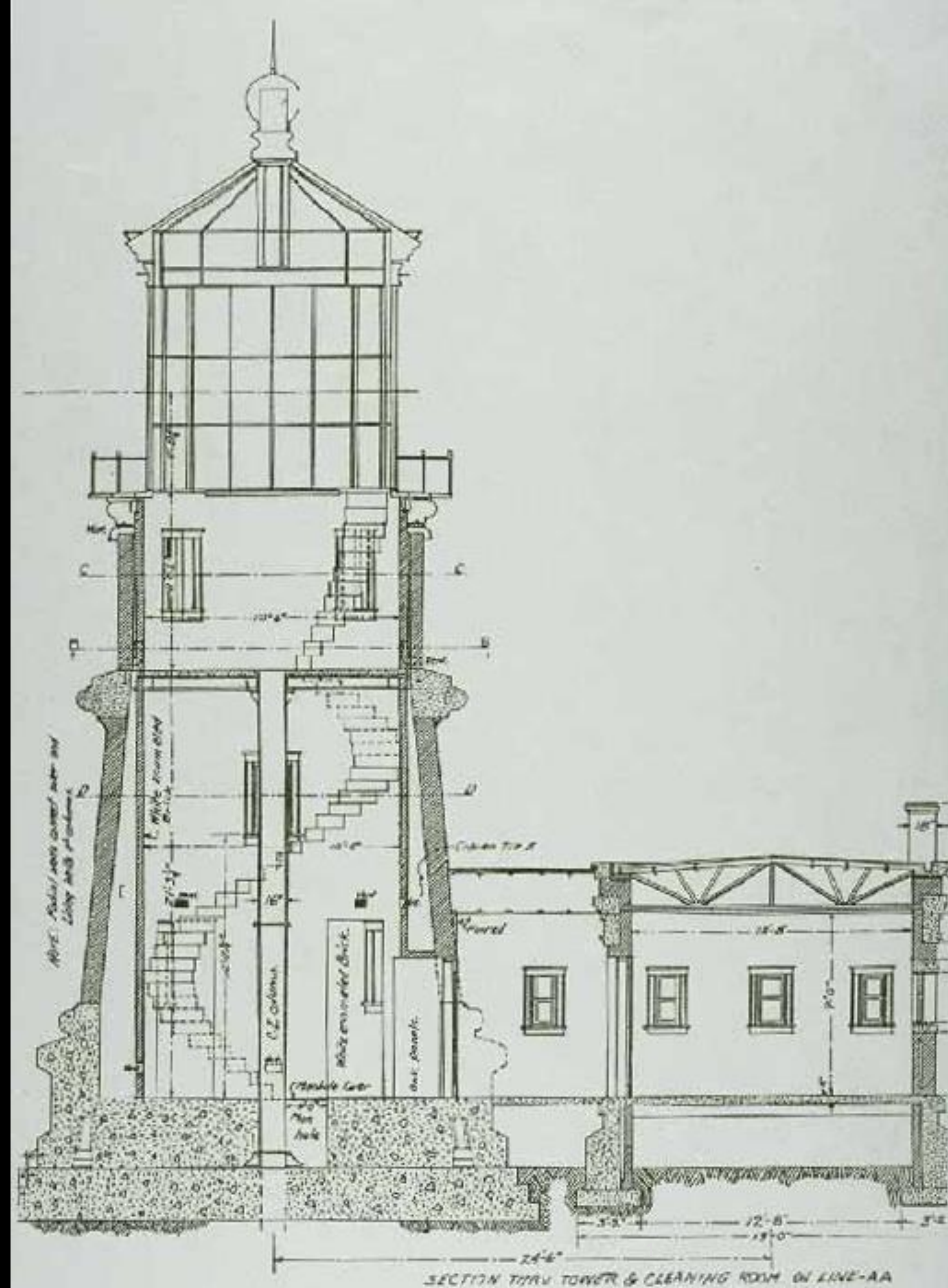
Software Architecture & Design



© Copyright 2017-2019 - Donnacha Forde - All rights reserved.

Spring

An Introduction to the Spring Framework

Contents

- An Introduction to Spring
- Inversion of Control and Dependency Injection
- Code Walkthrough – REST Interfaces with SpringBoot
- Demo – Deploying to the Cloud

© Copyright 2017-2018 - Donnacha Forde - All rights reserved.

Introduction

The Spring Framework

© Copyright 2017-2018 - Donnacha Forde - All rights reserved.

Spring Introduction

- What is Spring?
- What is it not?
- Characteristics
- What did it introduce?

© Copyright 2017-2018 - Donnacha Forde - All rights reserved.

Spring Introduction

- What is Spring?
- What is it not?
- Characteristics
- What did it introduce?

- Spring is a Java Framework
  - For Building Enterprise Applications
  - It provides Infrastructure – Allow developer to concentrate on logic
- Spring is synonymous with the Spring Framework
  - Spring == The Spring Framework
- Spring is Open-Source
  - Interface21→SpringSource→VMware→Pivotal
  - SpringSource.org & SpringSource.com →Spring.io

© Copyright 2017-2018 - Donnacha Forde - All rights reserved.

Spring Introduction

- What is Spring?
- What is it not?
- Characteristics
- What did it introduce?

- Spring is **not** a server/container
  - Requires a runtime (servlet) container ^ Or Reactive
    - i.e. Tomcat
  - Runs on JEE
    - JEE incorporates Tomcat
    - e.g. JBoss → JBossWeb
- SpringBoot now incorporates servlet/tomcat container
  - Jar incorporates servlet engine
  - Framework is opinionated

© Copyright 2017-2018 - Donnacha Forde - All rights reserved.

Spring Introduction

- What is Spring?
- What is it not?
- Characteristics
- What did it introduce?

- Spring manages lifecycle management of objects
  - Relationships between objects are ‘injected’
  - No need to explicitly create objects/bean (springbeans) with the ‘`new`’ operator
  - Sometimes referred to as Inversion of Control (IOC)
- Spring promotes a ‘dynamic programming model’
  - Application = Code + Metadata (POJO + XML*)
  - Objects are loosely coupled → Promotes ease of modification
- Spring is a form of ‘generic programming’
  - Code is abstracted to the point where ‘logic’ is removed & exists only in metadata

© Copyright 2017-2018 - Donnacha Forde - All rights reserved.

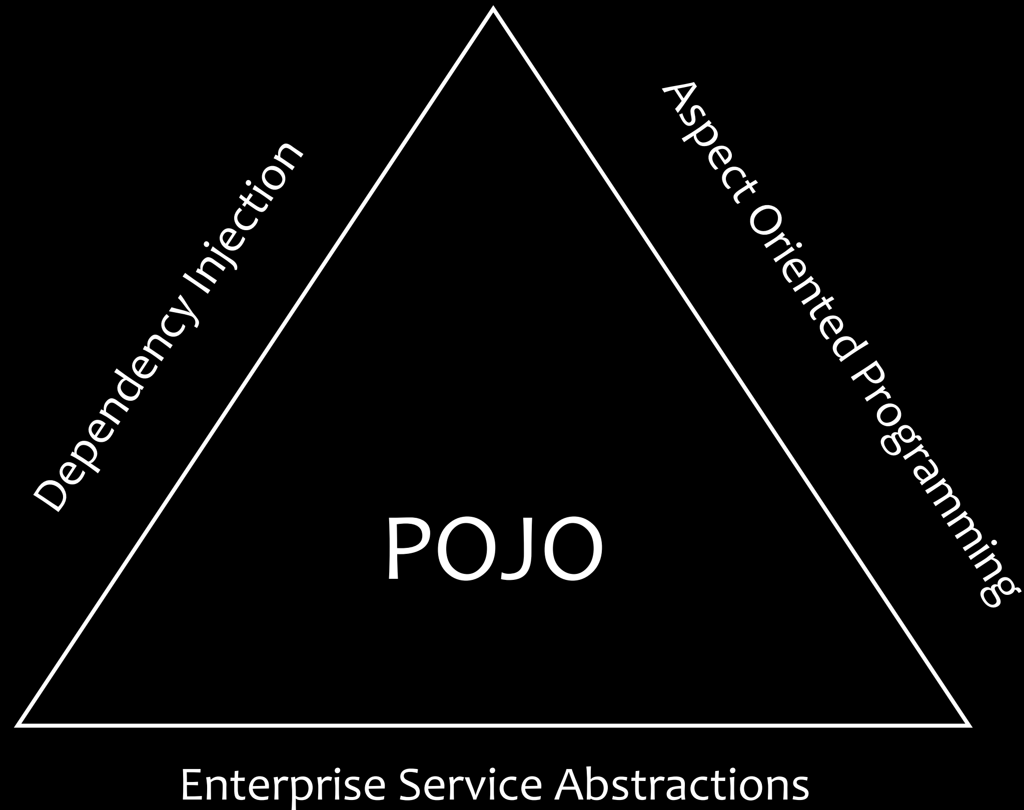
Spring Introduction

- What is Spring?
- What is it not?
- Characteristics
- What did it introduce?

- Spring = POJOs + Metadata
  - Dynamic Programming Model
  - Relationships between objects are 'injected'
- Spring = DI + AOP + Wrappers
  - Dependency Injection (DI)
  - Aspect Oriented Programming (AOP)
  - Wrapper API around enterprise services
    - Goal = Let Spring '*deal with the plumbing*'

© Copyright 2017-2018 - Donnacha Forde - All rights reserved.

Spring Introduction

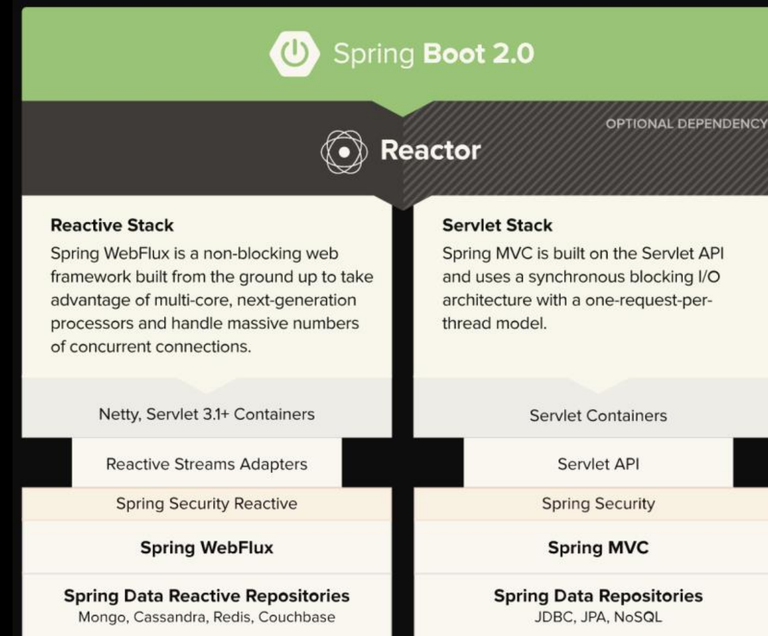


© Copyright 2017-2018 - Donnacha Forde - All rights reserved.

Spring Products

- Spring Framework
- Spring Web Flow
- Spring Web Services
- Spring Security (Acegi Security)
- Spring Dynamic Modules For OSGi(tm) Service Platforms
- Spring Batch
- Spring Integration
- Spring LDAP
- Spring IDE
- Spring Modules
- Spring JavaConfig
- Spring BlazeDS Integration
- Spring Rich Client
- Spring .NET
- Spring BeanDoc

- Spring without J2EE
  - Spring dm Server
    - OSGi Server. + Spring Dynamic Modules
  - Spring tc Server
    - Tomcat + Spring Framework
  - Reactive



© Copyright 2017-2018 - Donnacha Forde - All rights reserved.

Spring Evaluation

Advantages

- Non-invasive programming model – i.e. POJO
- Good Framework
  - Better API than provider
  - Decent OO abstractions
- Abstraction = Flexibility
  - Agnostic about runtime application server and provider (api/technology)
  - Swap, Plug & Play technology providers
- Modularization – No coded dependencies
  - Dynamically loaded/injected
  - Objects/Classes referenced by 'names' (Reflection)
  - Facilitates TDD/Unit Testing– i.e. easier to incorporate JUnit & Mocks
- Resource Usage
  - No runtime
  - HelloWorld = ~14Mb
- Ease of Development
  - Normal Java
  - Promotes POJO
  - Metadata
  - Spring IDE – Eclipse (STS)

Disadvantages

- Abstraction = Metadata*
  - More intelligence in 'configuration'
  - Abstractions don't replace underlying api/technology
- Abstraction = Complexity -or- Simplicity?
- Need to 'buy into' loosely coupled, metadata driven approach.
  - Debugging is more difficult
  - Dynamically bound system so runtime-only context
- Knowing 'Spring' is about annotations & configuration, not programming

© Copyright 2017-2018 - Donnacha Forde - All rights reserved.

Inversion of Control & Dependency Injection

An Analysis of the Basic Constructs

© Copyright 2017-2018 - Donnacha Forde - All rights reserved.

Java Code

```java
public class RewardNetworkImpl implements RewardNetwork
{
    private AccountRepository accountRepository;
    private RestaurantRepository restaurantRepository;
    private RewardRepository rewardRepository;
    .
    .


/** ctor */
public RewardNetworkImpl(AccountRepository accountRepository,
                         RestaurantRepository restaurantRepository,
                         RewardRepository rewardRepository) {
    this.accountRepository = accountRepository;
    this.restaurantRepository = restaurantRepository;
    this.rewardRepository = rewardRepository;
}
```

Member var declarations

ctor arg list

Assignment/Initialization

© Copyright 2017-2018 - Donnacha Forde - All rights reserved.

Metadata Definitions - XML

```xml
<bean id="rewardNetwork" class="rewards.internal.RewardNetworkImpl">
    <constructor-arg ref="accountRepository"/>
    <constructor-arg ref="restaurantRepository"/>
    <constructor-arg ref="rewardRepository"/>
</bean>

<!-- Loads accounts from the data source -->
<bean id="accountRepository" class="rewards.internal.account.JdbcAccountRepository">
    <property name="dataSource" ref="dataSource"/>
</bean>

<!-- Loads restaurants from the data source -->
<bean id="restaurantRepository" class="rewards.internal.restaurant.JdbcRestaurantRepository">
    <property name="dataSource" ref="dataSource"/>
</bean>

<!-- Records reward confirmation records in the data source -->
<bean id="rewardRepository" class="rewards.internal.reward.JdbcRewardRepository">
    <property name="dataSource" ref="dataSource"/>
</bean>
```

ctor args

Bean declarations

© Copyright 2017-2018 - Donnacha Forde - All rights reserved.

Spring: Hello World!

```
public class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println("Hello World!");
    }
}
```

Message is fixed

Message ‘rendering’ is hard-coded

- What improvements could be made?
- What issues could you spot?

© Copyright 2017-2018 - Donnacha Forde - All rights reserved.

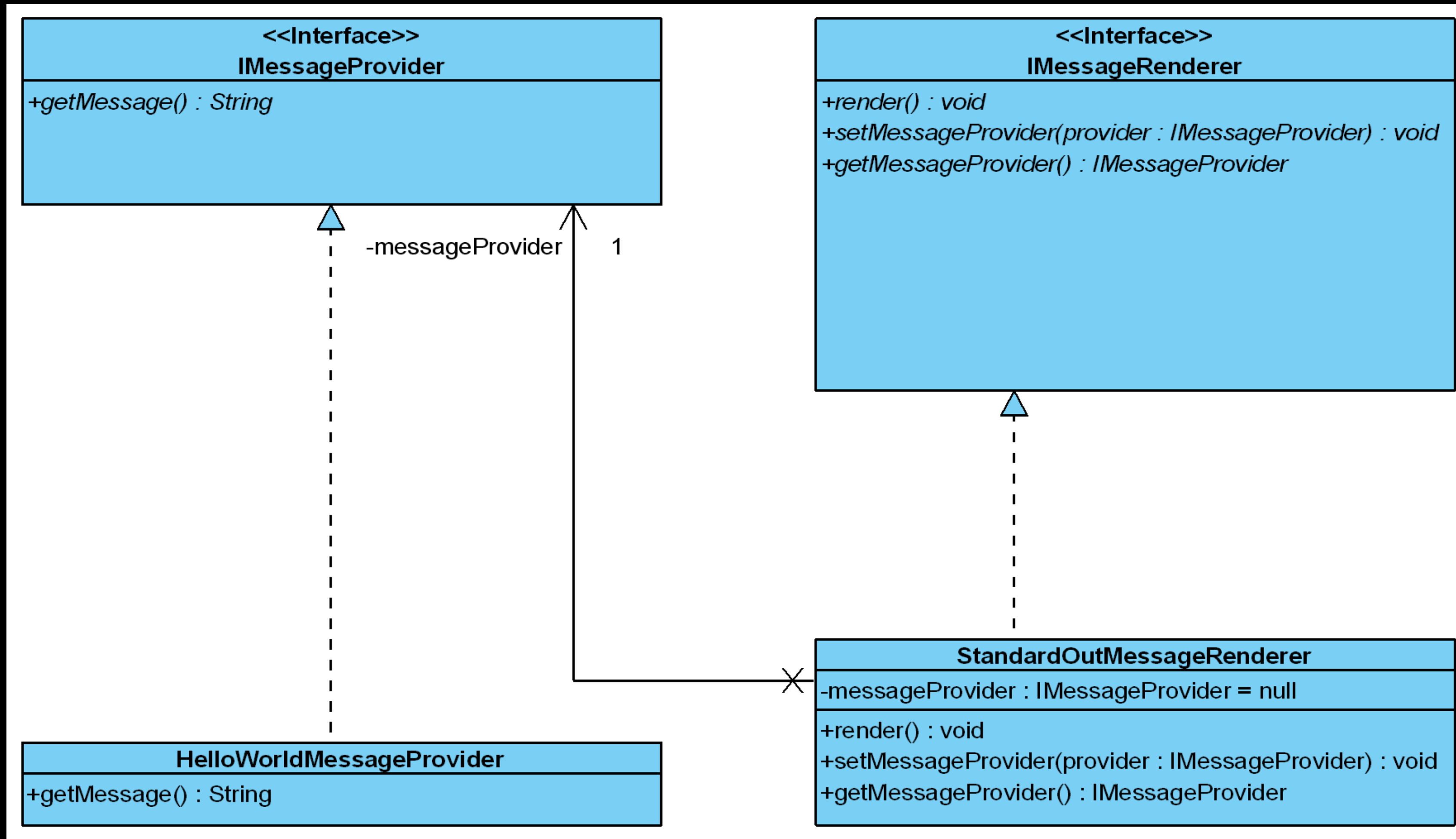
Spring: Hello World!

Message Provision

Message Rendering

© Copyright 2017-2018 - Donnacha Forde - All rights reserved.

Spring: Hello World! Decoupled



© Copyright 2017-2018 - Donnacha Forde - All rights reserved.

Spring: Hello World!

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

Interface defines Provider purpose

```
public class HelloWorldMessageProvider implements IMessageProvider
{
 @Override
    public String getMessage()
    {
        return "Hello World!";
    }
}
```

Concrete class provides specific message

© Copyright 2017-2018 - Donnacha Forde - All rights reserved.

Spring: Hello World!

```java
public interface IMessageRenderer
{
    public void render();


    public void setMessageProvider(IMessageProvider provider);
    public IMessageProvider getMessageProvider();
}
```

```java
public class StandardOutMessageRenderer implements IMessageRenderer
{
    private IMessageProvider messageProvider = null;

    @Override
    public void render(){
        if (messageProvider == null) {
            throw new RuntimeException("You must set the property messageProvider of class:"  + StandardOutMessageRenderer.class.getName());
        }

        System.out.println(messageProvider.getMessage());
    }


    @Override
    public void setMessageProvider(IMessageProvider provider) {
        this.messageProvider = provider;
    }

    @Override
    public IMessageProvider getMessageProvider() {
        return this.messageProvider;
    }
}
```

Interface defines Renderer purpose

Bean method to support injection*

Concrete class handles specific type of rendering

*No longer required when using @Autowired

© Copyright 2017-2018 - Donnacha Forde - All rights reserved.

Spring: Hello World!

```
public class HelloWorldSpring
{
    public static void main(String[] args) throws Exception
    {
        // get the bean factory
        BeanFactory factory = getBeanFactory();

        IMessageRenderer mr = (IMessageRenderer) factory.getBean("renderer");
        mr.render();
}

    private static BeanFactory getBeanFactory() throws Exception
    {
        // get the bean factory
        DefaultListableBeanFactory factory = new DefaultListableBeanFactory();

        // create a definition reader
        PropertiesBeanDefinitionReader rdr = new PropertiesBeanDefinitionReader(factory);

        // load the configuration options
        Properties props = new Properties();
        props.load(new FileInputStream("./beans.properties"));

        rdr.registerBeanDefinitions(props);

        return factory;
    }
```

Objects are dynamically loaded

Metadata is loaded & dependencies 'injected'

Dependency is defined in metadata

```
# The MessageRenderer
renderer.class=com.emc.smc.eval.StandardOutMessageRenderer
renderer.messageProvider(ref)=provider

# The MessageProvider
provider.class=com.emc.smc.eval.HelloWorldMessageProvider
```

© Copyright 2017-2018 - Donnacha Forde - All rights reserved.

Code Walkthrough

Using SpringBoot to build a REST Service

© Copyright 2017-2018 - Donnacha Forde - All rights reserved.

Demo

Deploying services to Cloud Foundry

© Copyright 2017-2018 - Donnacha Forde - All rights reserved.

Conclusions

- Code is modularized
- Code is highly decoupled
- Code modules are interchangeable, without recompilation
- Facilitates very late dynamic binding

© Copyright 2017-2018 - Donnacha Forde - All rights reserved.

Q&A

Discussion Time

Recommended Reading

- Spring docs
    - https://spring.io/docs
- Spring Guides
    - https://spring.io/guides
- Spring Projects
    - https://spring.io/projects
- Spring Javadoc
    - https://docs.spring.io/spring/docs/current/javadoc-api/

© Copyright 2017-2018 - Donnacha Forde - All rights reserved.

Thank You

© Copyright 2017-2018 - Donnacha Forde - All rights reserved.