Week 6 Day 1

Intro to Spring

Reflections API

REVATURE

Allows us to peak into java classes, and modify their states and behaviors at runtime

- Provided in java.lang.reflect package
- Provides information about the class
  - Methods names
  - Method return types
  - Annotations
- Used with many frameworks such as Junit and Spring
- Spring uses it to know which classes to control and to configure your app

Spring Intro

REVATURE

- Family of frameworks
  - Create loosely coupled apps
- Inversion of control container
  - Uses dependency injection
- Enables developers to build java applications with POJO's
  - Spring manages the dependencies between our services
  - Allows developers to focus on business logic

Spring: Modules

REVATURE

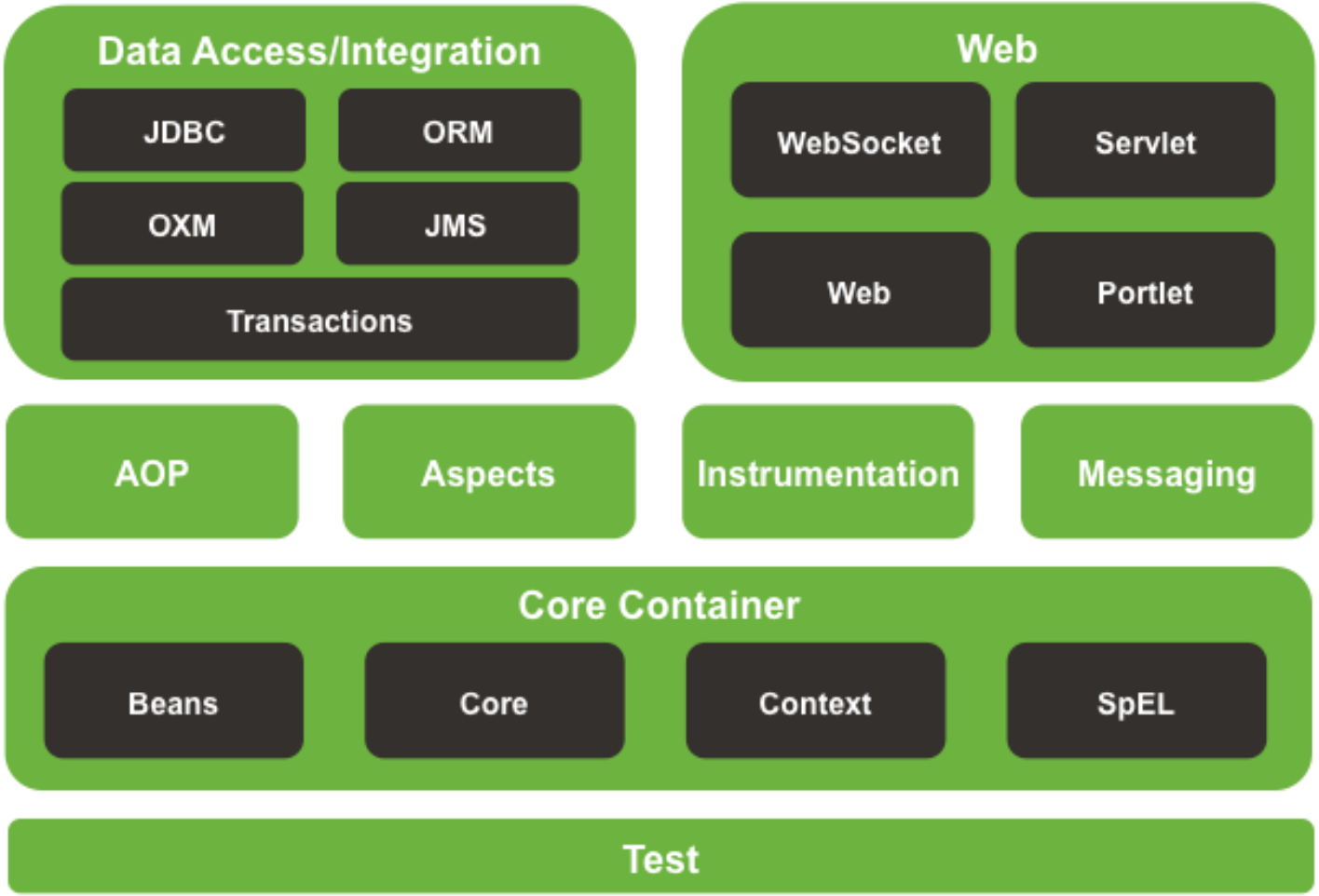

Image source: https://medium.com/@prabhur24/why-spring-101-on-spring-start-3be672f9d75

Spring: Inversion of Control

REVATURE

Design pattern where control over certain components is inverted to achieve loose coupling

- Giving control of modules/classes to spring
- Spring uses an IOC Container
- Gathers this information from beans.xml

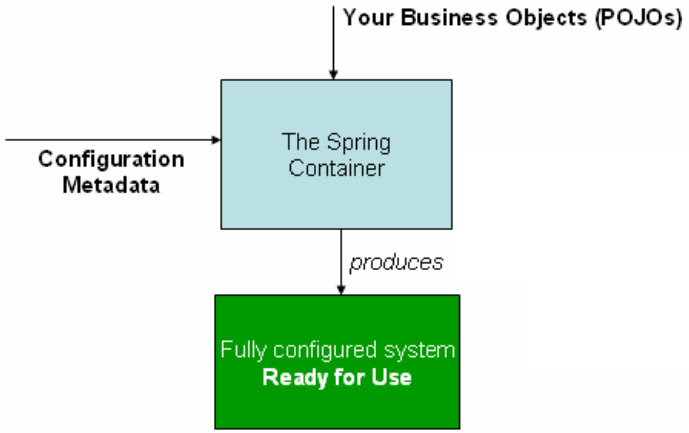

Image source: https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/beans.html

Spring: Inversion of Control Containers

REVATURE

- BeanFactory
  - Most basic IOC Container
- ApplicationContext
  - Implements BeanFactory and extends its features
  - Integration with Spring AOP
  - Event propagation
  - Message resource handling
  - Application layer specific context

Spring: Dependency Injection

REVATURE

Remove dependencies by providing configurations in external files

- Loosely couples your code
- Easier to test (mocking capabilities/inject mocks)
- Implement in a wider variety of environments
- Don’t need to manually create/pass objects your classes depend on

Spring: Constructor Injection

REVATURE

- Constructor Injection
  - Injected through constructors
  - More secure
  - Enables ability for immutable objects

```
public class DIExample {
    private Dependency injectMe;

    //Spring passes the Depedency object for us
    public DIExample(Dependency d){
        this.injectMe = d;
    }
}
```

Spring: Setting Injection

REVATURE

- Setter Injection
  - Injected through our setters after a no-args constructor is called
  - Allows for partial dependencies
  - More flexible
  - Used to resolve circular dependencies

```
public class DIExample {
    private Dependency injectMe;

    public DIExample(){

    }

    //Spring sets the dependency object for us here
    public void setInjectMe(Dependency d){
        this.injectMe = d;
    }
}
```

Spring: Bean Life Cycle

REVATURE

1. Instantiated
2. Bean properties set
3. Associated interfaces are made aware
4. Bean is made aware of associated interfaces
5. Custom methods are invoked
6. Bean is ready
7. Bean is marked for removal, and destroy method is invoked
8. Custom destroy methods invoked
9. Bean is destroyed

Spring: Bean Scopes

REVATURE

Scope is a sub-section of a larger application

- Singleton
- Prototype
- Request
- Session
- Global
- Application

Spring: Configuration

REVATURE

Two ways to configure our beans

- XML files
- Annotations and Configuration class

```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"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">

 <!-- We can define our beans here -->

 <bean id="bookController" class="it.mamino84.example.spring.BookController"></bean>
</beans>
```

```java
package com.concretepage;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan("com.concretepage")
public class OuterConfig {
    @Bean
    public OuterTask getOuterTask() {
            return new OuterTask("Outer Task");
    }

    @Configuration
    public static class InnerConfig {
        @Bean
        public InnerTask getInnerTask() {
            return new InnerTask("Inner Task");
        }
    }
}
```

Image source: https://medium.com/quick-code/xml-or-nnotations-this-is-the-dilemma-6c00cfbf8746

Image source: https://www.concretepage.com/spring-5/spring-nested-configuration-classes

Spring: Annotations

REVATURE

- @Autowired
- @Bean
- @Component
- @ComponentScan
- @Configuration
- @Inject
- @Qualifier
- @Required
- @Resource

Spring: Autowiring

REVATURE

Autowiring allows us to tell Spring to find the beans and register itseld

- “no”: no autowiring
- “byType”: look at the class’ property datatypes
- “constructor”: uses the constructor with the most distinct dependencies
- “byName”: finds beans in the container with the same name

Spring: Stereotypes

REVATURE

Tells Spring what kind of components it is creating

- @Component
- @Repository
- @Service
- @Controller