Spring Framework - Basics

Web Application Development

Zsolt Tóth

University of Miskolc

2017
Table of Contents

1. Instantiation
2. Dependency Injection
3. Spring Core
Static Type vs. Dynamic Type

Collection<User> users = new ArrayList<User>();

Static Type
- Type Declaration
- Defines
  - Methods
  - Intention
  - Limits Behavior
- Could be
  - Class
  - Abstract Class
  - Interface

Dynamic Type
- Instantiation
- Actual Type
- Determine Behavior
- instanceof Static Type
  - implements
  - extends
- Can be Instantiated.
Inheritance and Implementation

**Inheritance**
- Specification
- Class Extension
  - Fields
  - Methods
- @Override
- Only One Parent
  - abstract
- final

**Implementation**
- Interface
- Behavior Only
  - Return Type
  - Name
  - Parameters
  - Exceptions
  - ? JavaDoc
- Public Methods and Services
- Abstract Methods
  - Java 8 default
Abstraction

Liskov Substitution Principle

- General \leftrightarrow\ Abstract
- Expected Behavior
- Various Implementation
- Static Type should be
  - as abstract as possible.
  - as specific as necessary.

```java
Collection<User> users = new ArrayList<User>();
List<User> users = new ArrayList<User>();
ArrayList<User> users = new ArrayList<User>();
```
new Operator

- Creates an Object
  - Memory Allocation
  - Invoke Constructor
  - Initialize Object
- Costly
- Issues
  - Hard Code Dynamic Type
  - When use?
  - Where use?
  - Testing

```java
class Rectangle{
    private Point p1, p2;

    Rectangle(Point p1, Point p2){
        this.p1 = p1;
        this.p2 = p2;
    }
}
```
new Operator

Rectangle r = new Rectangle(0,0,1,1);
/* where
Rectangle(double plx,double ply,...){
this.p1 = new Point(plx,ply); ... 
}
*/

//or
Point p1 = new Point(0,0);
Point p2 = new Point(1,1);
Rectangle r = new Rectangle(p1,p2);

//or
Rectangle r = new Rectangle(
    new Point(0,0),
    new Point(1,1);
);
Instantiation vs. Usage

**Instantiation**
- Costly
  - Resource Allocation
  - Memory Allocation
- Takes Time
  - Establish Network Connection
- Could be Difficult
  - Tons of Parameters
  - JDBC
- Defines Dynamic Type

**Usage**
- Use Object via Methods
- Depends on Expected Behavior
  - Static Type
  - Interface Required
- Should not Create Object.
- Dynamic Type could be Changed
- Reusability

Object creation and usage should be separated!
Consequences

Pro
- Instantiation could be Centralized.
- Testing Become Easier.
  - See "Software Projects & Testing" subject.
- Increase Reusability.
- More General Code

Question
- How will we get the dependencies?
Table of Contents

1. Instantiation
2. Dependency Injection
3. Spring Core
Participants

**service** an object that will be used

**client** object that uses the **service**

**interface** defines how the **client** uses the **service**

**injector** creates the **service** and gives it into the **client**.

**Goal**

Separation of the creation of the **client**’s dependencies from the **client**’s behavior.
Types of Dependency Injection

Field Injection
- Dependency Injection Frameworks know
- Should not be used.
- Reflection

Constructor Injection
- Dependencies are Passed as Constructor Arguments
- Cannot Create object when a Dependency is Missing
- Mandatory Dependencies

Setter Injection
- Setter Methods are Invoked After Constructor Used
- Optional Dependencies
**Constructor Injection**

- Mandatory Dependencies
- Part of Client Initialization
- Block the Client’s Functionality
- Examples
  - Database Access

```java
class Client{
    private MandatoryService ms;
    private OptionalService os;
    public Client(MandatoryService ms){
        this.ms = ms;
    }
}
```
Setter Injection

- Optional Dependencies
- Not Required for the Client
- Extends, Enhance its Functions
- Examples
  - Logging
  - Publishing Event

```java
class Client{
    private MandatoryService ms;
    private OptionalService os;
    public setOptionalService(OptionalService os){
        this.os = os;
    }
}
```
Typical Service Categories

- **Storage**
  - Database Access Objects

- **Business Logic**
  - Other Functions
  - Micro Services

- **3rd Party Services**
  - System Integration
  - "Distributed Systems" subject in MSc.
  - Social Media
  - Currency Change
  - www.programmableweb.com
Consequences

+ Flexibility and Configurability
+ Refactoring
+ Increase Abstractness
+ Cleaner Code
+ Independent Development
+ Facilitates Testing and Mocking

- Configuration Required
- Difficult to Understand
- Dependency Injection Framework is Required
Table of Contents

1. Instantiation
2. Dependency Injection
3. Spring Core
Spring Framework

- Java
- Current Version 4.3
- Open Source
- Inversion of Control
- Framework
  - Core, AOP, Data
  - Web, WebMVC
  - Integration, Cloud

<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-core</artifactId>
<version>4.3.9.RELEASE</version>
</dependency>
Spring Projects

Spring Framework

Data Access / Integration
- JDBC
- ORM
- OXM
- JMS

Transactions

Web (MVC / Remoting)
- WebSocket
- Servlet
- Web
- Portlet

AOP
Aspects
Instrumentation
Messaging

Core Container
- Beans
- Core
- Context
- SpEL

Test
Core Elements

Core
- Basic Tasks
- Resource Finding

Beans
- Bean Definition
- XML Mapping
  - `spring-beans-x.x.x.xsd`

Context
- ApplicationContext

SpEL
- Spring Expression Language
- Text Evaluation
- XML based Bean Definition
  - `#{string expr.}`

#{systemProperties['user']}
#{T(java.lang.Math).random()}
ApplicationContext

- interface
- Bean factory methods
- Load file resources
- Publish events
- Resolve messages
- Support internationalization.
- Context inheritance

Super Interfaces
- BeanFactory
- ResourceLoader

Implementations
- Annotation Config Application Context
- ClassPath Xml Application Context
- Xml Web Application Context
BeanFactory

- interface
- Bean Container
- Registry of Application Components
- Implementations (27)
  - XML
  - Annotation

Bean

- Object
- String ID
- Dependency
- Types
  - Prototype
  - Singleton

containsBean(String)
getBean(String)
getBean(Class<T>)
isPrototype(String)
isSingleton(String)
isTypeMatch(String,Class<T>)
Bean Life-cycle

1. BeanNameAware’s setBeanName
2. BeanClassLoaderAware’s setBeanClassLoader
3. BeanFactoryAware’s setBeanFactory
4. EnvironmentAware’s setEnvironment
5. EmbeddedValueResolverAware’s setEmbeddedValueResolver
6. ResourceLoaderAware’s setResourceLoader (only applicable when running in an application context)
7. ApplicationEventPublisherAware’s setApplicationEventPublisher (only applicable when running in an application context)
8. MessageSourceAware’s setMessageSource (only applicable when running in an application context)
Bean Life-cycle

9. ApplicationContextAware’s setApplicationContext (only applicable when running in an application context)

10. ServletContextAware’s setServletContext (only applicable when running in a web application context)

11. postProcessBeforeInitialization methods of BeanPostProcessors

12. InitializingBean’s afterPropertiesSet

13. a custom init-method definition

14. postProcessAfterInitialization methods of BeanPostProcessors
Application Context - XML

- AbstractXmlApplicationContext
  - ClassPathXmlApplicationContext
  - FileSystemXmlApplicationContext

- XML
  - Text based
  - Human Readable
    - Security
  - Reconfigurable

Location

- $basedir/src/main/resources
- WEB-INF/
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="..."
xmlns:xsi="..."
xmlns:p="..."
xmlns:context="..."
xmlns:aop="..."
xmlns:mvc="..."
xsi:schemaLocation="...">
  <import resource="si-config-measurement.xml"/>
  ...
</beans>
<bean id="wifiDistanceCalculator" class="<packages>.VectorIntersectionWiFiRSSIDistance"/>

<bean id="measurementDistanceCalculator" class="<packages>.MeasurementDistanceCalculatorImpl">
  <constructor-arg ref="wifiDistanceCalculator"/>
  <constructor-arg name="wifiDistanceWeight" value="1.0"/>
  <constructor-arg name="magnetometerDistanceWeight" value="0.5"/>
  <constructor-arg name="gpsDistanceWeight" value="0.0"/>
</bean>
<bean id="positioningService"
class="<packages>.KNNSimplePositioning">
  <constructor-arg name="distanceCalculator"
    ref="measurementDistanceCalculator"/>
  <constructor-arg name="measurementGateway"
    ref="MeasurementGateway"/>
  <constructor-arg name="k" value="3"/>
</bean>
</beans>
Application Context - Annotation

Annotation
- Metadata
- No Direct Effect
- Usage
  - Information for the compiler
  - Compile-time and deployment-time processing
  - Runtime processing

Sprint Annotations
- Java Based
- XML is not required.
- Not Readable
- Configuration ???
- Annotations
  - @Component
  - @Configuration
  - @Bean
Application Context - Annotation

@Component
- Meta Annotation
- Candidates for Auto-detection
- Logical Component Name

@Configuration
- @Component
- Contains @Bean methods
- Describe Application Context

@Bean
- Method
- Returns with a Bean
- Bean Name
  - Method Name
  - Specified names
  - Alias
- Optional Parameters
  - autowire : Autowire
  - initMethod : String
  - destroyMethod : String
  - name : String
  - value : String
@Configuration
public class ExampleConfiguration {

@Bean(name = "greetingsService")
public GreetingsService initGreetingsService() {
    return new GreetingsHU();
}
}