

Spring Framework Configuration



Agenda

- Introduction to Spring Framework Configuration
- Spring Configuration Options
- Spring Framework Stereotype
- Spring Component Scan
- Java Configuration Example
- Using Spring Factory Beans
- Spring Boot Configuration
- Spring Bean Scope

Spring Bean Configuration Options



Review – What is a Spring Bean?

What is a Spring Bean?

- In Spring, the objects that form the backbone of your application and that are managed by the Spring IoC container are called Beans.
- A bean is an object that is instantiated, assembled and otherwise managed by the Spring IoC container.
- Official Documentation:
<https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-introduction>

Spring Configuration Options

Spring Bean Configuration Options:

1. **xml-based:**

- spring bean configuration is defined within an xml file

2. **Java-based:**

- configuration is defined within a java class(es), marked with specific annotations

3. **groovy-based:**

- when configuration is defined within groovy code.

Spring Configuration Options

XML based configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans" ...>

    <bean id="accountService" class="com.wiley.beginningspring.ch2.AccountServiceImpl">
        <property name="accountDao" ref="accountDao"/>
    </bean>

    <bean id="accountDao" class="com.wiley.beginningspring.ch2.AccountDaoInMemoryImpl">
    </bean>

</beans>
```

Requires an xml
config that looks like
this

Bootstrapping code

```
ClassPathXmlApplicationContext applicationContext = new ClassPathXmlApplicationContext("/com/wiley/beginningspring/ch2/ch2-beans.xml");
```

Spring Configuration Options

Java based configuration

```
@Configuration
public class Ch2BeanConfiguration {

    @Bean
    public AccountService accountService() {
        AccountServiceImpl bean = new AccountServiceImpl();
        bean.setAccountDao(accountDao());
        return bean;
    }

    @Bean
    public AccountDao accountDao() {
        AccountDaoInMemoryImpl bean = new AccountDaoInMemoryImpl();
        return bean;
    }
}
```

Java configuration requires a class utilizing the `@Configuration` annotation

Beans are defined with the `@Bean` annotation

Bootstrapping code

```
ApplicationContext applicationContext
    = new AnnotationConfigApplicationContext(Ch2BeanConfiguration.class);
```

Spring Configuration Options

Groovy Bean definition DSL (domain specific language) Configuration

- Introduced in Spring Framework version 4.0
- Allows you to declare beans in Groovy

```
<dependency>
  <groupId>org.codehaus.groovy</groupId>
  <artifactId>groovy</artifactId>
  <version>3.0.9</version>
</dependency>
```

Assume the following Bean definition

```
public class JavaPersonBean {
    private String firstName;
    private String lastName;

    // standard getters and setters
}
```

Crucial!

```
beans {
    javaPersonBean(JavaPersonBean) {
        firstName = 'John'
        lastName = 'Doe'
    }
}
```


Spring Bean Definition Options



Spring Definition Options

Two ways to add bean definitions into an application

1. Configuration based:

- In this case the definition will be based on the configuration type.
 - For xml-config, uses `<bean></bean>` tags
 - For java-config, uses `@Bean`
 - For groovy, uses `beans{ ... }`

2. Annotation based:

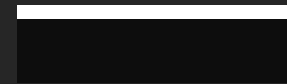
- Class level bean definitions occurring when you mark java classes using specific annotations such as
 - `@Component`
 - `@Service`
 - `@Controller`
 - `@Repository`

picked up via component scan

Which to Use?

- You can use a combination of all methods
 - They will work seamlessly together to define beans in the Spring Context
 - Please note however, the general Industry trend (at present) is to favor **Java based configuration**

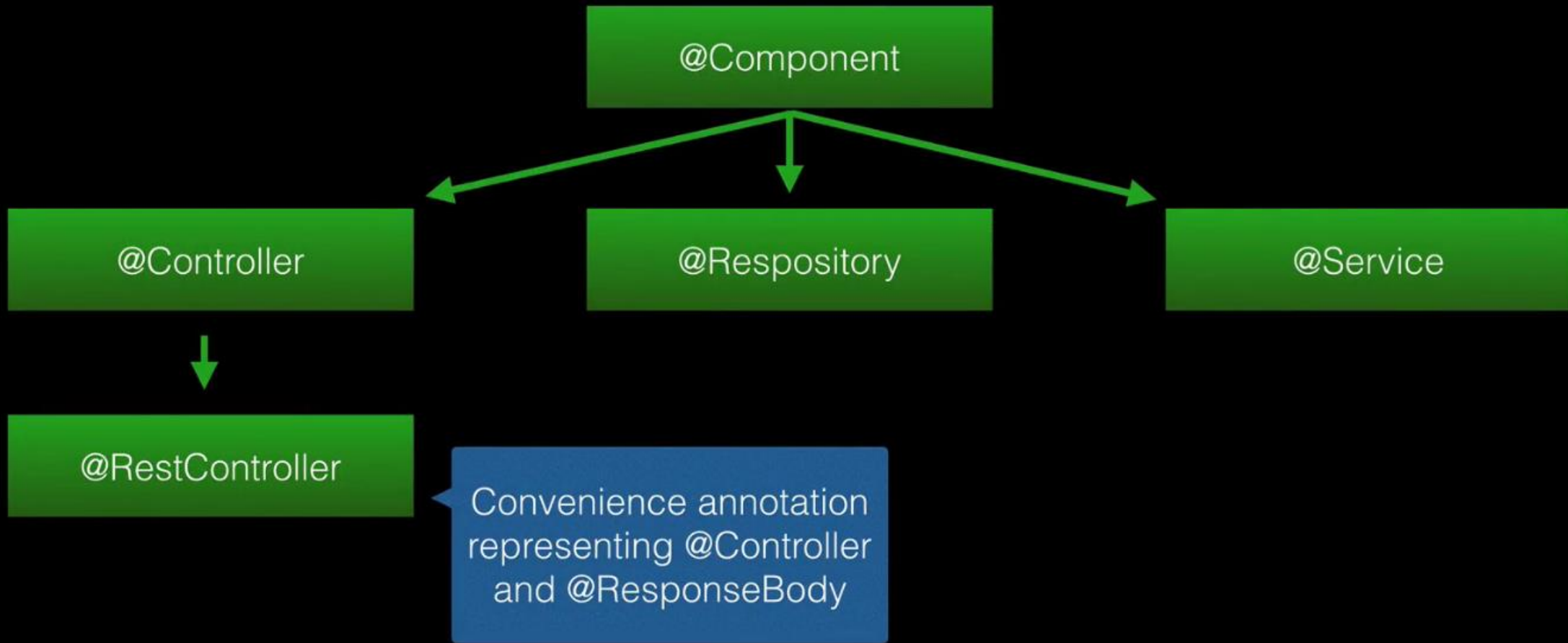
Spring Framework Stereotypes



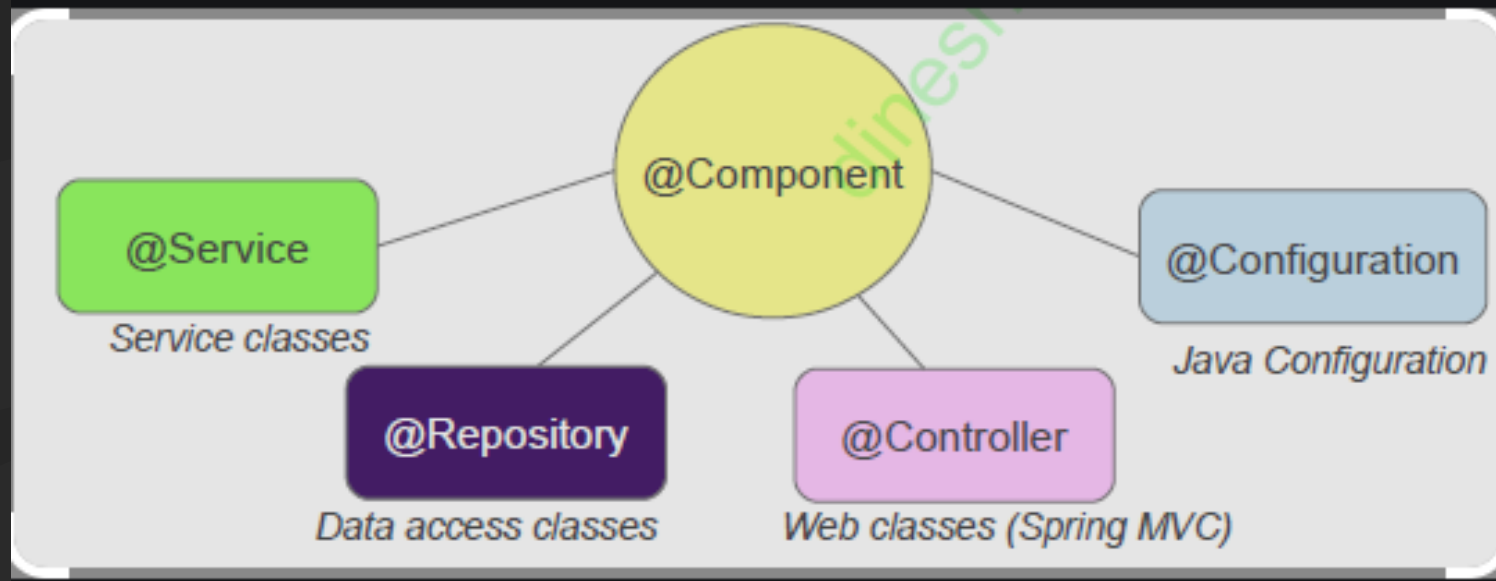
Spring Framework Stereotypes

- Largely used to express the intent of the bean
- Spring Stereotypes are used to define Spring Beans in the Spring Context
- Available Stereotypes - **@Component**, **@Controller**, **@RestController**, **@Repository**, **@Service**
- With **@Component**, **@Repository**, **@Service** and **@Controller** annotations in place and automatic component scanning enabled, Spring will automatically import the beans into the container and inject to dependencies as required.

Spring Framework Stereotype Hierarchy



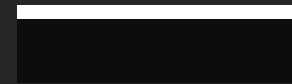
Spring Framework Stereotype Hierarchy



Spring Framework Stereotype Hierarchy

Annotation	Description
@Component	Indicates that an annotated class is a “component” and it will be created as a bean
@Controller	Indicates that an annotated class has the role of a Spring MVC “Controller”
@RestController	Convenience Annotation which extends @Controller, and adds @ResponseBody
@Repository	Indicates that an annotated class is a “Repository”, originally defined by Domain-Driven Design (Evans, 2003) as “a mechanism for encapsulating storage, retrieval, and search behavior which emulates a collection of objects”
@Service	Indicates that an annotated class is a “Service”, originally defined by Domain-Driven Design (Evans, 2003) as “an operation offered as an interface that stands alone in the model, with no encapsulated state.”

Spring Component Scan

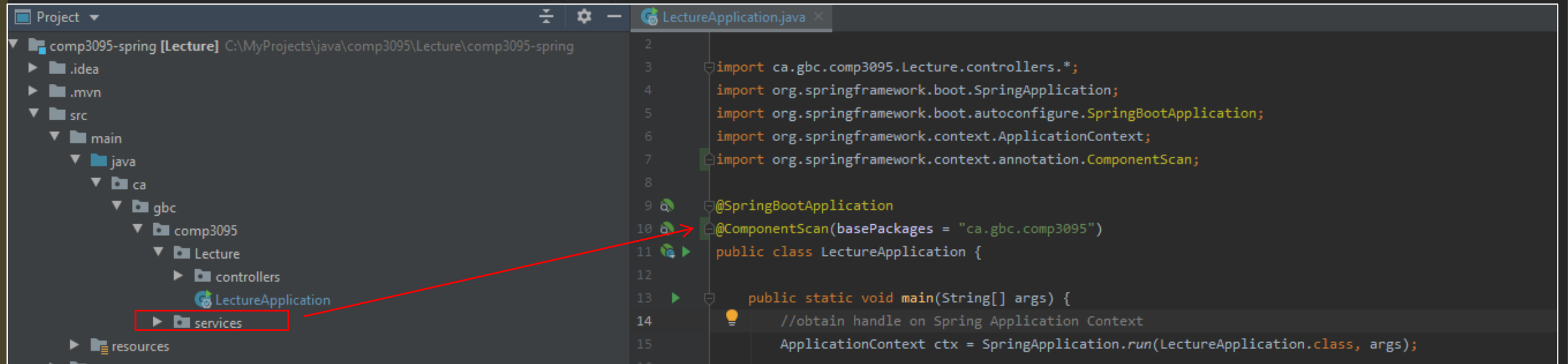


Spring Component Scan

- Is a way for Spring to look at a projects package structure, to determine and locate Spring Beans.
- Specifically, Spring is scanning for any classes that have been annotated, as components, services etc..
- Spring by default, looks at its own package, and scans down from that package, but it does not scan out (this is a common junior developer mistake).

@ComponentScan

Used to specify the package that you want to get scanned by the Spring Context

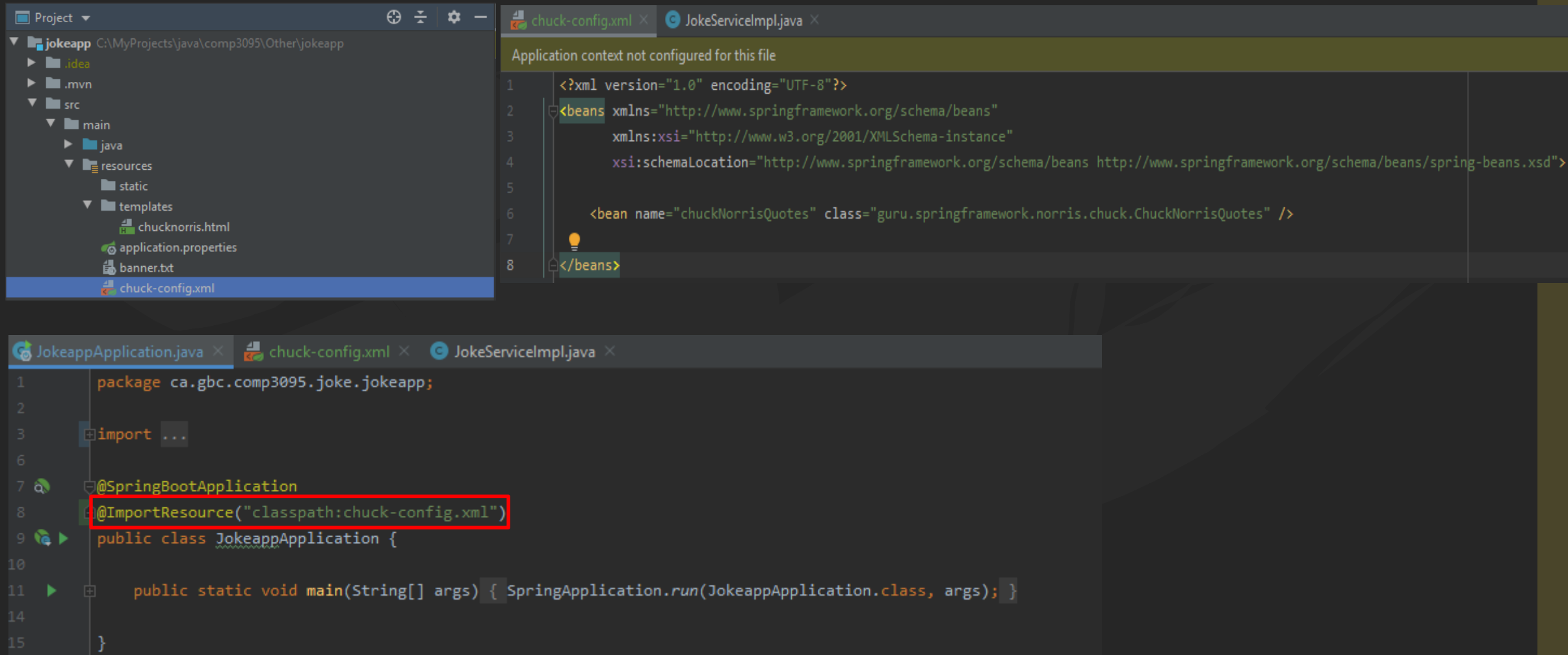


Spring XML vs Java Configuration Example



Spring XML Configuration continued...

- Create Spring Config file (New → XML Configuration → Spring Config) under resources folder.



The screenshot displays an IDE with two main panels. The top panel shows the 'Project' view on the left, highlighting the 'resources' folder under 'main'. The right pane shows the 'chuck-config.xml' file with the following content:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">
5
6     <bean name="chuckNorrisQuotes" class="guru.springframework.norris.chuck.ChuckNorrisQuotes" />
7
8 </beans>
```

The bottom panel shows the 'JokeappApplication.java' file with the following code:

```
1 package ca.gbc.comp3095.joke.jokeapp;
2
3 import ...
4
5
6
7 @SpringBootApplication
8 @ImportResource("classpath:chuck-config.xml")
9 public class JokeappApplication {
10
11     public static void main(String[] args) { SpringApplication.run(JokeappApplication.class, args); }
12
13
14
15 }
```

In the Java code, the `@ImportResource("classpath:chuck-config.xml")` annotation is highlighted with a red rectangle.

Spring Java Bean Configuration

```
package ca.gbc.comp3095.joke.jokeapp.services;

import guru.springframework.norris.chuck.ChuckNorrisQuotes;
import org.springframework.stereotype.Service;

@Service
public class JokeServiceImpl implements JokeService{

    private final ChuckNorrisQuotes chuckNorrisQuotes;

    public JokeServiceImpl(ChuckNorrisQuotes chuckNorrisQuotes) {
        this.chuckNorrisQuotes = chuckNorrisQuotes;
    }

    @Override
    public String getJoke() { return chuckNorrisQuotes.getRandomQuote(); }

}
```

```
package ca.gbc.comp3095.joke.jokeapp.config;

import guru.springframework.norris.chuck.ChuckNorrisQuotes;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

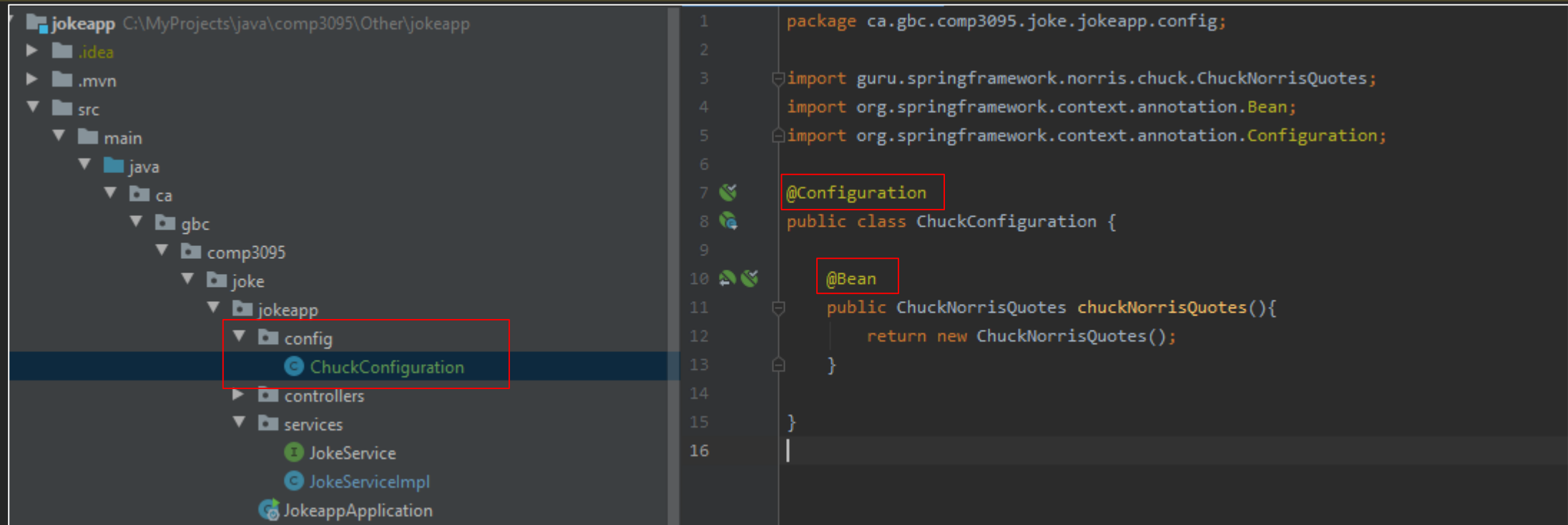
@Configuration
public class ChuckConfiguration {

    @Bean
    public ChuckNorrisQuotes chuckNorrisQuotes(){
        return new ChuckNorrisQuotes();
    }

}
```

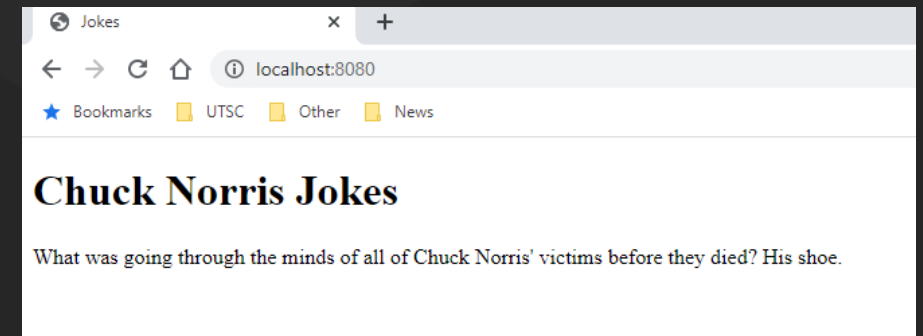
Java Configuration Based

@Configuration continued ...



The screenshot shows an IDE with a project structure on the left and a code editor on the right. The project structure on the left shows a package hierarchy: `ca.gbc.comp3095.joke.jokeapp.config`. The `ChuckConfiguration` class is highlighted in the project structure. The code editor on the right shows the implementation of the `ChuckConfiguration` class, which is annotated with `@Configuration` and `@Bean`.

```
1 package ca.gbc.comp3095.joke.jokeapp.config;
2
3 import guru.springframework.norris.chuck.ChuckNorrisQuotes;
4 import org.springframework.context.annotation.Bean;
5 import org.springframework.context.annotation.Configuration;
6
7 @Configuration
8 public class ChuckConfiguration {
9
10     @Bean
11     public ChuckNorrisQuotes chuckNorrisQuotes(){
12         return new ChuckNorrisQuotes();
13     }
14
15 }
16
```



The screenshot shows a web browser window with the title "Jokes" and the address bar showing "localhost:8080". The page displays "Chuck Norris Jokes" and a single joke: "What was going through the minds of all of Chuck Norris' victims before they died? His shoe."

Jokes

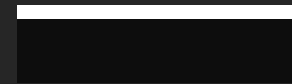
localhost:8080

Bookmarks UTSC Other News

Chuck Norris Jokes

What was going through the minds of all of Chuck Norris' victims before they died? His shoe.

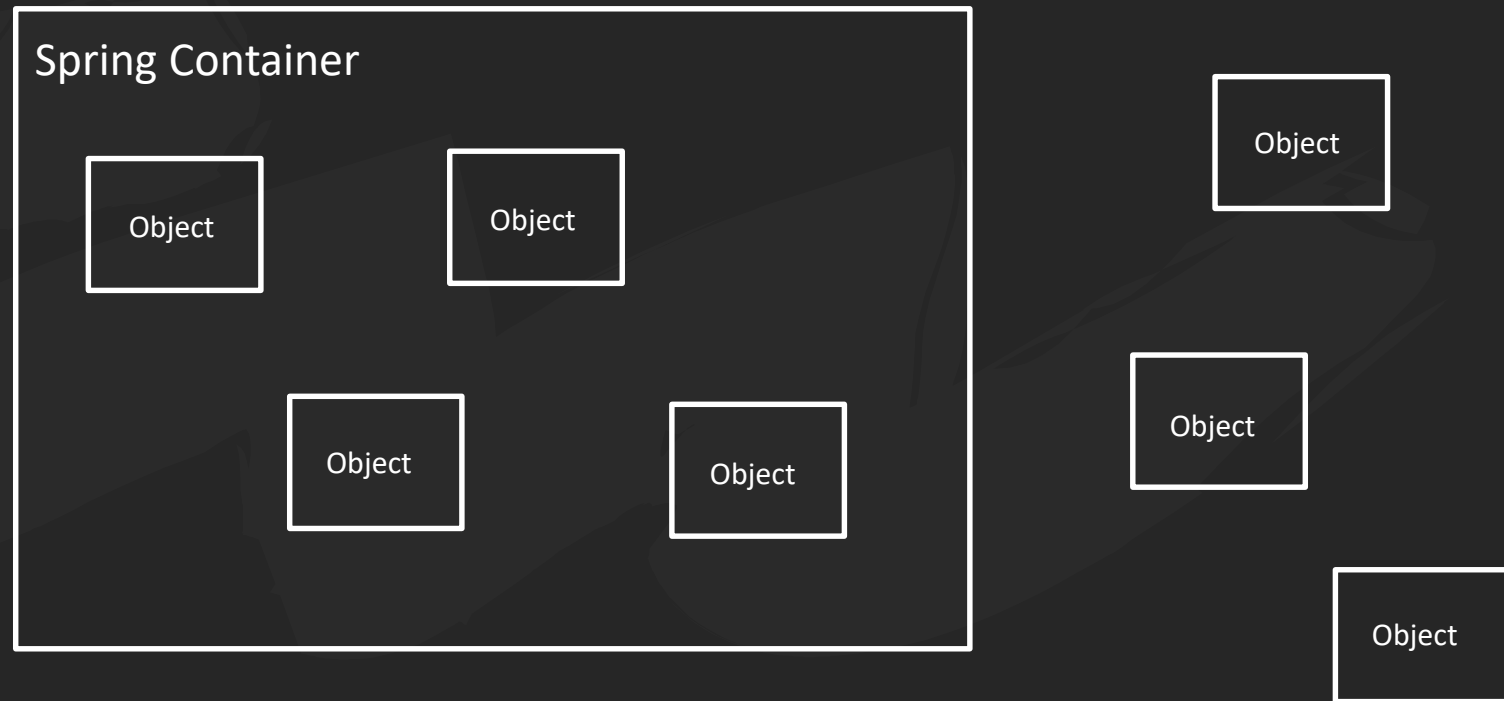
Spring Factory Bean



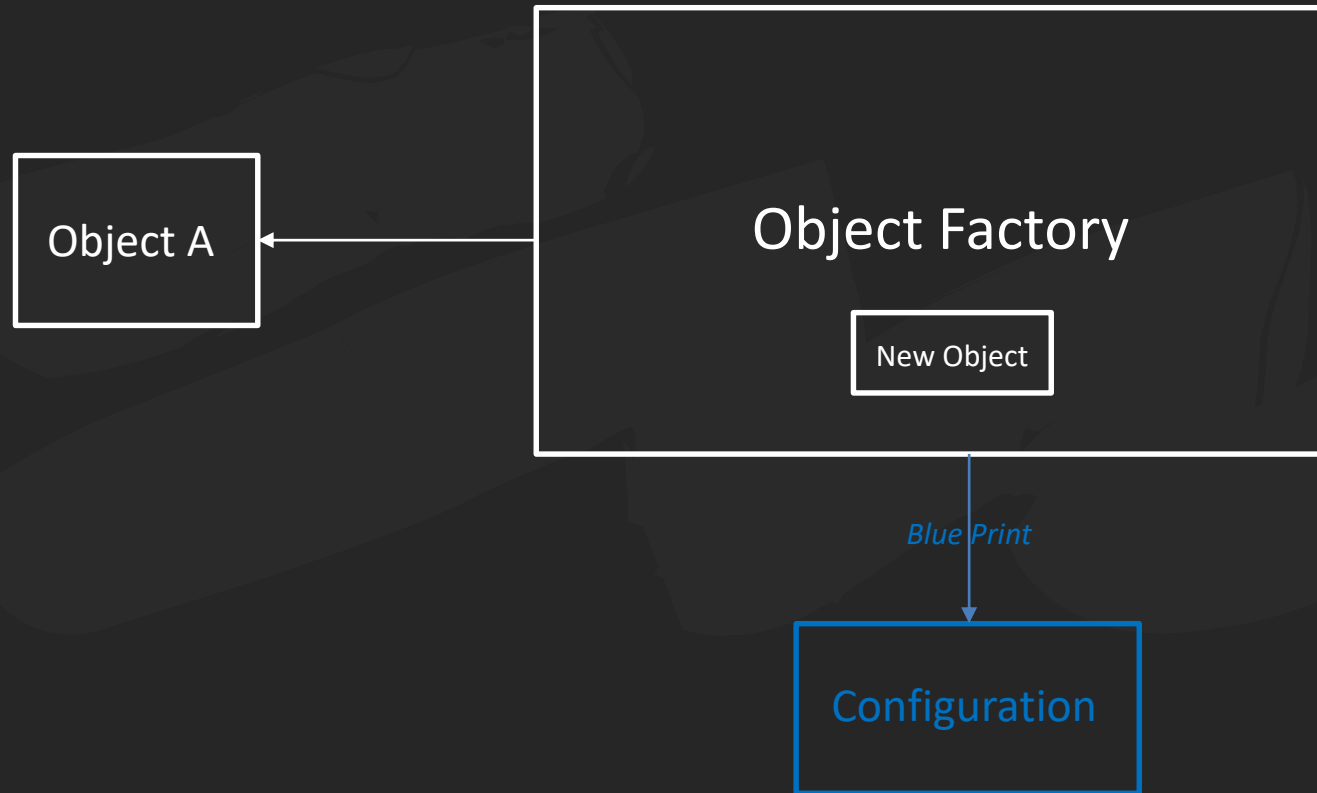
Spring Factory Beans

- There are two kinds of beans in the Spring bean container
 - Ordinary Beans
 - Factory Beans
- Spring FactoryBean, is a special bean in the Spring that can be used as a bean factory (ie. used to create other Spring managed beans).
- It's used to encapsulate interesting object construction logic in a class.
- This is achieved by employing the **Factory Design Pattern**

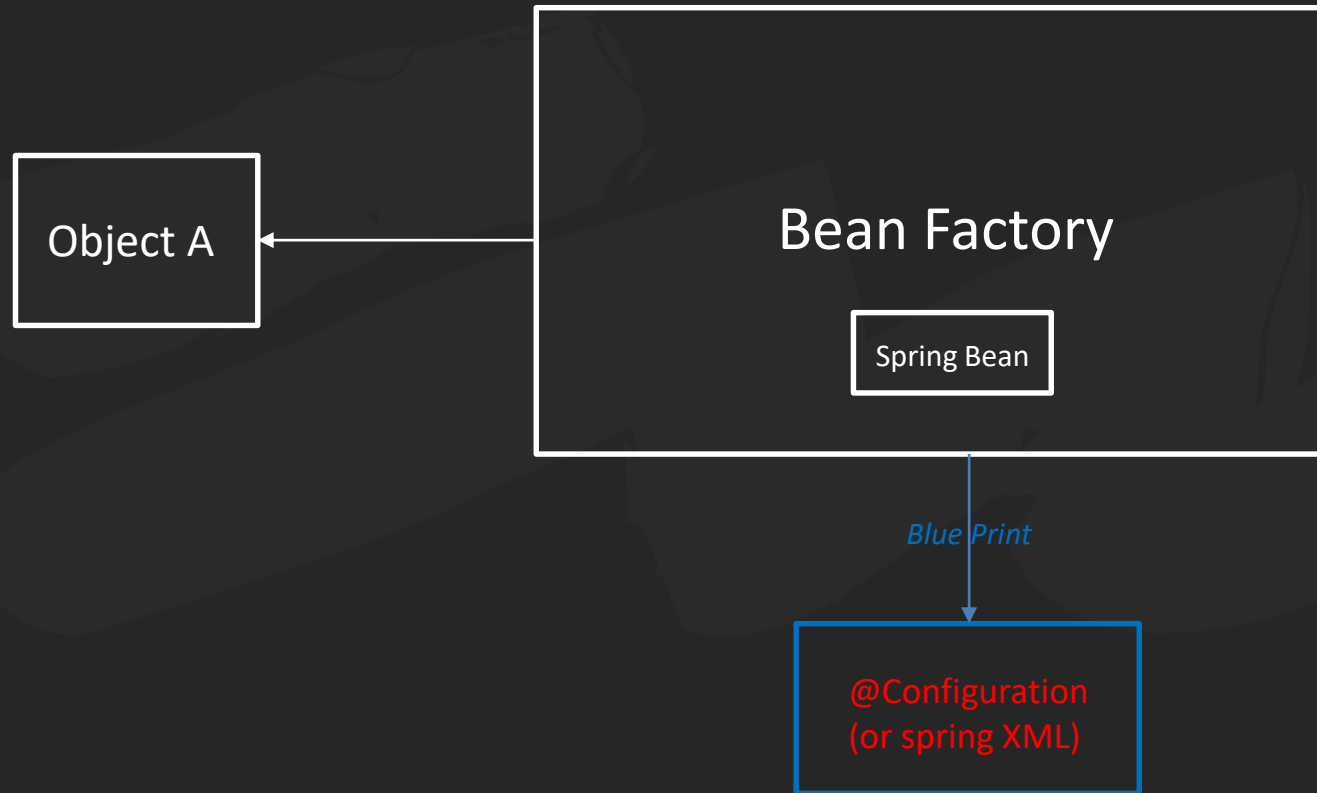
A Spring Container



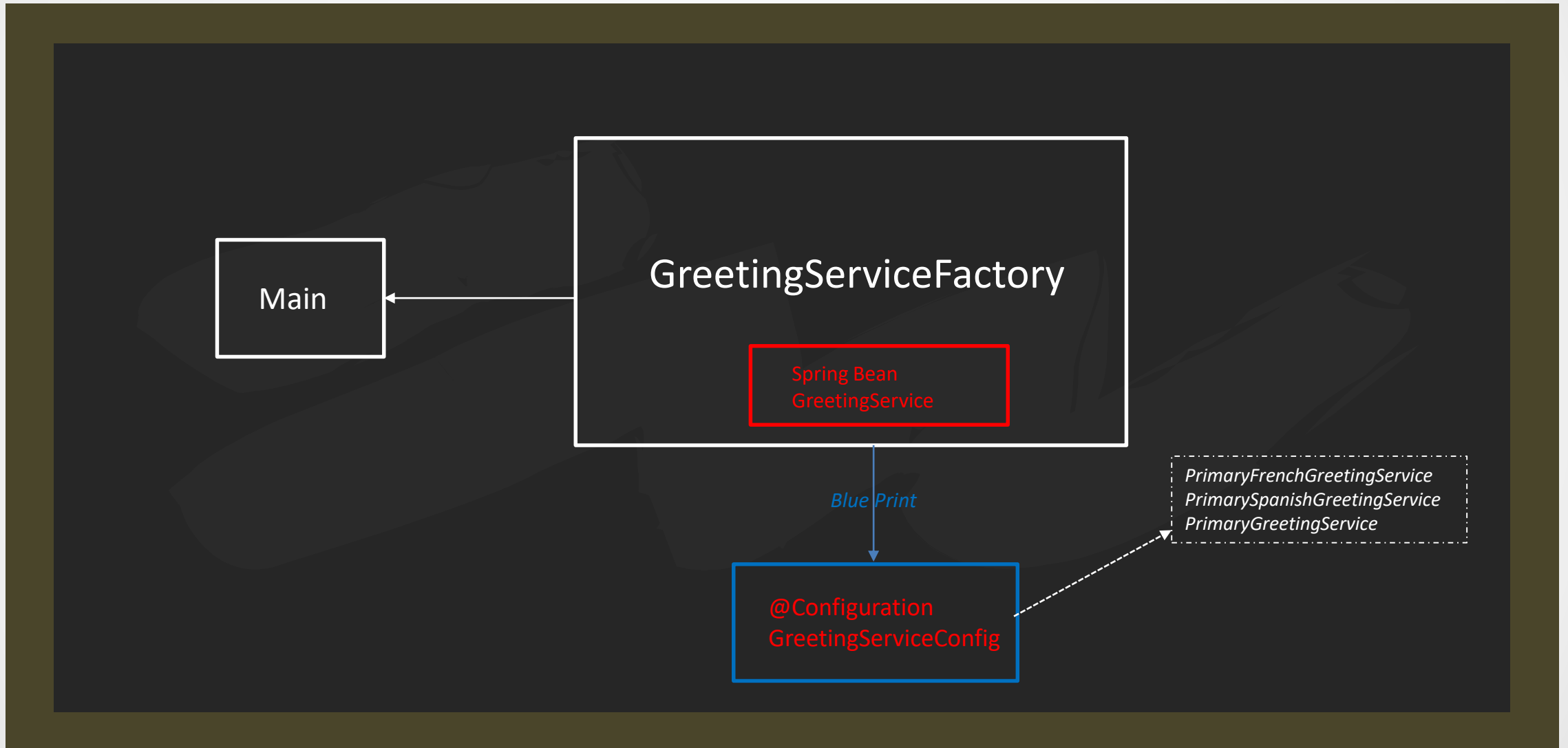
Factory Design Pattern



Spring Bean Factory



Greeting Service - Example Code



```

@Configuration
public class GreetingServiceConfig {

    /**
     * Need to make Factory a Spring Managed Bean
     */

    @Bean
    GreetingServiceFactory greetingServiceFactory(GreetingRepository repository){
        return new GreetingServiceFactory(repository);
    }

    /**
     * we use @Primary to give higher preference to a bean when there are multiple beans of the same type.
     */

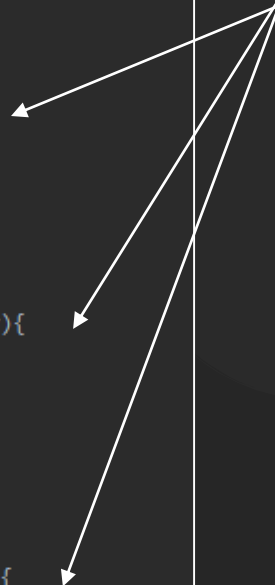
    @Bean
    @Primary
    @Profile({"default", "EN"})
    GreetingService primaryGreetingService(GreetingServiceFactory greetingServiceFactory){
        return greetingServiceFactory.createGreetingService( lang: "EN");
    }

    @Bean
    @Primary
    @Profile("ES")
    GreetingService primarySpanishGreetingService(GreetingServiceFactory greetingServiceFactory){
        return greetingServiceFactory.createGreetingService( lang: "ES");
    }

    @Bean
    @Primary
    @Profile("FR")
    GreetingService primaryFrenchGreetingService(GreetingServiceFactory greetingServiceFactory){
        return greetingServiceFactory.createGreetingService( lang: "FR");
    }
}

```

- Sometimes its easier and cleaner to consolidate the Bean configuration to a central class
- One class to maintain for all your configurations rather than have them spread across different classes.



Spring Boot Configuration



What is Spring Boot?

- Spring boot is an extension of the Spring Framework which eliminated the boilerplate configurations required for setting up a Spring application.
- It takes an opinionated (assertive) view of the Spring platform which paved the way for a faster more efficient development eco-system.
- Example Features
 - Opinionated 'starter' dependencies to simplify build and application configuration
 - Embedded server to avoid complexity in application deployment
 - Metrics, Health check, and externalized configuration
 - Automatic config for Spring functionality – whenever possible

Spring Boot Configuration

- We have been relying quite a bit on the magic of Spring Boot, underneath, Spring Boot has been taking care of a lot of sensible configuration for us, making it easy for us to bring up a application in no time.
- Spring boot can/has been configuring data source (ex H2 in memory database) and more.
- Much of this configuration work has been hidden from us, thanks in large part to Spring Boot.
- There are many configuration options in Spring Boot

Dependency Management of Spring Boot

- Maven or Gradle are supported for curated dependencies.
- Each version of Spring Boot is configured to work with a specific version of the Spring Framework.
- Overriding the Spring Framework Version is NOT recommended
- Other build systems such as Ant, can be used but are not recommended.

Maven Support in Spring Boot

- Maven projects inherit from a Spring Boot Parent POM
- When possible, do NOT specify versions in your POM. Allow the versions to inherit from the parent.
- The Spring Boot Maven Plugin allows for packaging the executable jar

Gradle Support in Spring Boot

- Gradle support depends on a Spring Boot Gradle plugin
- Requires Gradle 3.4 or later
- The Gradle plugin provides support of the curated dependencies, packaging as jar or war, and allows you to run the application from the command line.

Spring Boot Starters



Spring Boot Starters

- Starters are top level dependencies for popular Java libraries
- Will bring in dependencies for the project and related Spring components
- Starter “**spring-boot-starter-data-jpa**” brings in:
 - Hibernate
 - Spring Data JPA – related Spring dependencies

Spring Boot Annotations



Spring Boot Annotations

- **@SpringBootApplication** – main annotation to use (top-level annotations)
- Includes:
 - **@Configuration** - Declares a class as a Spring Configuration class
 - Declare Spring Beans directly here.
 - **@EnableAutoConfiguration** – Enables auto configuration
 - **@ComponentScan** – Scans for components in the **current package** and all child packages.

Disable Spring Boot Annotations

- Auto-configuration will bring a lot of configuration classes in supplied Spring Boot jars
- You can specify classes to exclude with:
 - **`@EnableAutoConfiguration(exclude={DataSourceAutoConfiguration.class})`**

Spring Bean Scope



Spring Bean Scopes

There are 7 Spring Bean Scope that we have to be aware of:

- 1. **Singleton** → (default) Only one instance of the bean is created in the IoC Container
- 2. **Prototype** → a new instance is created each time the bean is requested
- 3. **Request** → a single instance per http request. Only valid in the context of a web-aware Spring ApplicationContext.
- 4. **Session** → a single instance per http session. Only valid in the context of a web-aware Spring ApplicationContext.
- 5. **Global-session** → a single instance per global session. Typically only used in a Portlet context (portlet container). Only valid in the context of a web-aware Spring ApplicationContext
- 6. **Application** → bean is scoped to the lifecycle of a ServletContext. Only valid in the context of a web-aware.
- 7. **Websocket** → scopes a single bean definition to the lifecycle of a WebSocket. Only valid in the context of a web-aware Spring ApplicationContext.

Spring Bean Scopes continued...

- Custom Scope → Spring Scopes are extensible and you can define your own scope by implementing Spring's "Scope" interface
 - We will not be investigating this in the course, but you're welcome to research the Java Documentation on your own.
 - Please note however, you cannot override the built in **Singleton** and **Prototype** Scopes

Spring Singleton Scope

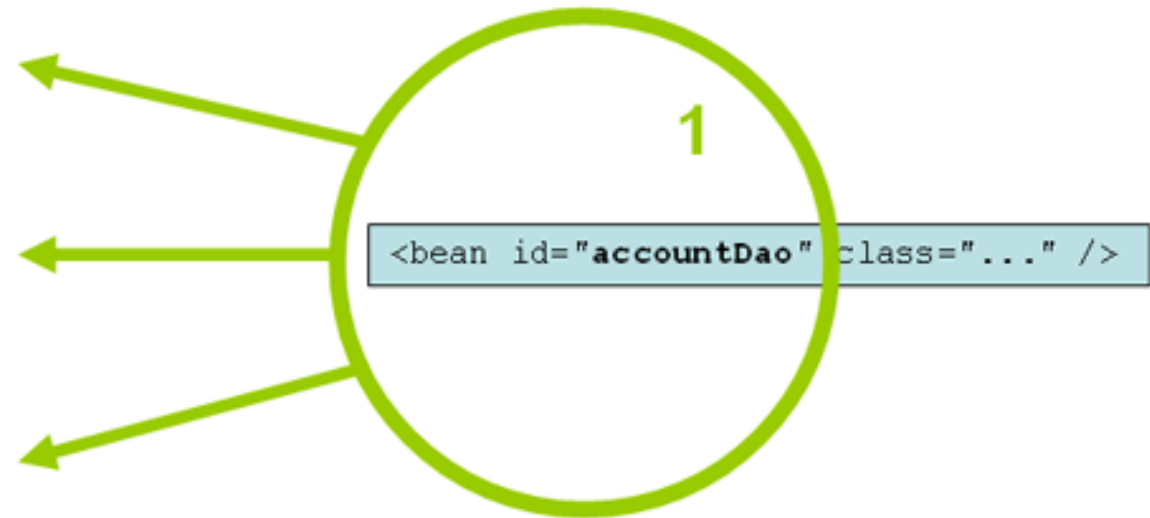
3 Beans

```
<bean id="..." class="...">  
  <property name="accountDao"  
    ref="accountDao"/>  
</bean>
```

```
<bean id="..." class="...">  
  <property name="accountDao"  
    ref="accountDao"/>  
</bean>
```

```
<bean id="..." class="...">  
  <property name="accountDao"  
    ref="accountDao"/>  
</bean>
```

Only one instance is ever created...



... and this same shared instance is injected into each collaborating object

Spring Prototype Scope

3 Beans

```
<bean id="..." class="...">  
  <property name="accountDao"  
    ref="accountDao"/>  
</bean>
```

A brand new bean instance is created...

1

```
<bean id="..." class="...">  
  <property name="accountDao"  
    ref="accountDao"/>  
</bean>
```

2

```
<bean id="accountDao" class="..."  
  scope="prototype" />
```

```
<bean id="..." class="...">  
  <property name="accountDao"  
    ref="accountDao"/>  
</bean>
```

3

... each and every time the prototype is referenced by collaborating beans

Declaring the Bean Scope

- No declaration needed for **Singleton** scope
- Java configuration uses **@Scope** annotation (ex @Scope ("singleton"))

```
public class Person {  
    private String name;  
  
    // standard constructor, getters and setters  
}
```

```
@Bean  
@Scope("singleton")  
public Person personSingleton() {  
    return new Person();  
}
```

```
@Scope(value = ConfigurableBeanFactory.SCOPE_SINGLETON)
```



- In XML configuration is an XML attribute of the “**bean**” tag
 - (ex <bean id="personSingleton" class="ca.gbc.Person" scope="singleton"/>)

Questions?