



Configuration Classes

1. Introduction to configuration classes
2. Additional techniques



1. Intro to Configuration Classes

- Overview of configuration classes
- Defining a simple configuration class
- Location of configuration classes
- Defining beans in a Spring Boot app class

Overview of Configuration Classes

- A configuration class is a special class in Spring Boot
 - Creates and initializes bean objects
 - You can use the beans elsewhere in your application
- Here's how to do it:
 - Define a class and annotate with `@Configuration`
 - Implement methods annotated with `@Bean`, to create and return bean objects

Defining a Simple Configuration Class

- Here's a simple example of a configuration class

```
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;
```

```
@Configuration
```

```
public class ConfigSimple {
```

```
    @Bean
```

```
    public MyBean myBean() {  
        MyBean b = new MyBean();  
        b.setField1(42);  
        b.setField2("wibble");  
        return b;  
    }
```

```
    // Etc.
```

```
}
```

ConfigSimple.java

Location of Configuration Classes

- Configuration classes are special kinds of "components"
- When a Spring Boot app starts, it automatically scans for components (and hence configuration classes)
 - It looks in the package of the app class, plus subpackages
- You can tell it to look elsewhere, if you like

```
@SpringBootApplication( scanBasePackages={"mypackage1", "mypackage2"} )  
public class Application {  
    ...  
}
```

Defining Beans in a Spring Boot App Class (1 of 2)

- The application class is itself a configuration class
- Because `@SpringBootApplication` incorporates:
 - `@Configuration`
 - `@EnableAutoConfiguration`
 - `@ComponentScan`
- This means you can define `@Bean` methods in your application class - see next slide...

Defining Beans in a Spring Boot App Class (2 of 2)

- Example:

```
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ApplicationContext ctx = SpringApplication.run(Application.class, args);
        ...
    }

    @Bean
    public MyBean myBean() {
        MyBean b = new MyBean();
        b.setField1(42);
        b.setField2("wibble");
        return b;
    }
    // etc.
}
```

A large, light gray play button icon is positioned on the left side of the slide. It consists of a white right-pointing triangle centered within a series of concentric circles in varying shades of gray.

2. Additional Techniques

- Customizing bean names
- Looking-up named beans
- Injecting dependencies
- Controlling instantiation
- Importing configurations

Customizing Bean Names

- The bean name is same as method name, by default
 - You can specify a different bean name, if you like
 - Set the `name` property in the `@Bean` annotation
 - Specify a single name, or an array of names

```
@Configuration
public class ConfigAdvanced {

    @Bean(name = "cool-bean")
    public SimpleService mySimpleService1() {...}

    @Bean(name = {"subsystemA-bean", "subsystemB-bean", "subsystemC-bean"})
    public SimpleService mySimpleService2() {...}

    ...
}
```

ConfigAdvanced.java

Looking-Up Named Beans

- To look up a bean by its name:
 - Call `getBean()` and specify the bean name you want


```
private static void demoAdvancedJavaConfig(ApplicationContext ctx) {  
  
    // Lookup 1st bean via its name.  
    SimpleService refBean1 = ctx.getBean("cool-bean", SimpleService.class);  
    refBean1.doSomething();  
  
    // Lookup 2nd bean via its various aliases.  
    SimpleService refBean2a = ctx.getBean("subsystemA-bean", SimpleService.class);  
    SimpleService refBean2b = ctx.getBean("subsystemB-bean", SimpleService.class);  
    SimpleService refBean2c = ctx.getBean("subsystemC-bean", SimpleService.class);  
    refBean2a.doSomething();  
    refBean2b.doSomething();  
    refBean2c.doSomething();  
    ...  
}
```

Application.java

Injecting Dependencies

- You can wire-up dependencies between beans like so:

```
@Configuration
public class ConfigAdvanced {
    ...
    @Bean(name="bankservice-bean")
    public BankService myBankService() {
        return new BankServiceImpl(myBankRepository());
    }
    @Bean(name="bankrepository-bean")
    public BankRepository myBankRepository() {
        return new BankRepositoryImpl();
    }
    ...
}
```



ConfigAdvanced.java

Controlling Instantiation

- You can set the scope of a bean

```
@Configuration
public class ConfigAdvanced {

    @Bean(name="proto-bean")
    @Scope("prototype")
    public SimpleService mySimpleService3() { ... }

    ...
}
```

ConfigAdvanced.java

- You can specify a singleton bean is instantiated lazily

```
@Configuration
public class ConfigAdvanced {

    @Bean(name="lazy-bean")
    @Lazy
    public SimpleService mySimpleService4() { ... }

    ...
}
```

ConfigAdvanced.java

Importing Configurations

- A configuration class can import another configuration class via `@Import`

```
@Configuration
@Import(SomeOtherConfig.class)
public class MyConfig {
    ...
}
```

- A configuration class can import an XML configuration file (if you have any!) via `@ImportResource`

```
@Configuration
@ImportResource("SomeOtherConfig.xml")
public class MyConfig {
    ...
}
```

A large, light gray play button icon is positioned on the left side of the slide. It consists of a white right-pointing triangle centered within a series of concentric circles in varying shades of gray.

Summary

- Introduction to components and beans
- Autowiring