Configuration Classes

1. Defining a config class and bean methods
2. Locating config classes and bean methods
3. Configuration techniques
4. Configuring bean dependencies

Demo project: `demo-configuration-classes`

1. Defining a Config Class and Bean Methods

- Overview of configuration classes
- Defining a simple configuration class
- Accessing a bean

Demo package: `demo.simpleconfiguration`

Overview of Configuration Classes

- A configuration class is a special "factory" class in Spring Boot
  - Creates and initializes bean objects

- How to define a configuration class:
  - Annotate class with `@Configuration`
  - Annotate methods with `@Bean` and create/return objects

Defining a Simple Configuration Class

- Here's a simple configuration class:

```java
@Configuration
public class MyConfig {

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

MyConfig.java

- This example creates a bean:
    - Type of bean is `MyBean`
    - Name of bean is "`myBean`"

Accessing a Bean

- You can access beans as normal:

```
ApplicationContext ctx = SpringApplication.run(Application.class, args);

MyBean bean = ctx.getBean("myBean", MyBean.class);
System.out.println(bean);                                    Application.java
```

- You can also autowire beans as normal:

```
@Component
public class SomeComponent {

    @Autowired
    MyBean bean;
    …
}                                       SomeComponent.java
```

2. Locating Config Classes and Bean Methods

- Location of configuration classes
- Specifying different configuration locations
- Defining beans in the "application" class

Demo package: `demo.configurationlocation`

Location of Configuration Classes

- Configuration classes are special kinds of "components"

- When a Spring Boot application starts…
  - It scans for components and configuration classes
  - It looks in the "application" class package, plus sub-packages

Specifying Different Configuration Locations

- You can tell Spring Boot to look in alternative packages to find components and configuration classes

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

- See the following packages in the demo app:
  - `demo.configurationlocation.`**`main`** - app class
  - `demo.configurationlocation.`**`config`** - config class

olsen software

Defining Beans in the "Application" Class (1 of 2)

- The `@SpringBootApplication` annotation is equivalent to:
  - `@Configuration`
  - `@EnableAutoConfiguration`
  - `@ComponentScan`

```
@SpringBootApplication
public class Application {
     …
}
```

- This means the application class is also a "configuration" class
  - You can define `@Bean` methods in your application class
  - See example on next slide…

olsen software

Defining Beans in the "Application" Class (2 of 2)

```
@SpringBootApplication(scanBasePackages="demo.configurationlocation.config")
public class Application {

    @Bean
    public LocalDateTime timestamp1() {
        return LocalDateTime.of(1997, 7, 2, 1, 5, 30);
    }

    @Bean
    public LocalDateTime timestamp2() {
        return LocalDateTime.of(1997, 7, 2, 1, 20, 0);
    }
    …
}
```

Application.java

3. Configuration Techniques

- Customizing bean names
- Looking-up named beans
- Lazily instantiating a singleton bean
- Setting the scope of a bean

Demo package: `demo.techniques`

Customizing Bean Names

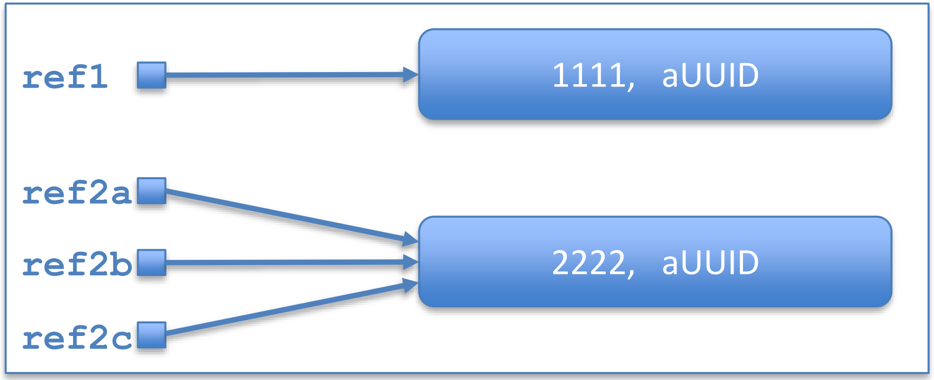
- By default the bean name is the same as method name
    - You can specify different bean name(s), if you like

```
@Configuration
public class MyConfig {

    @Bean(name="cool-bean")
    public MyBean bean1() { return new MyBean(1111, aUUID); }

    @Bean(name = {"subsystemA-bean", "subsystemB-bean", "subsystemC-bean"})
    public MyBean bean2() { return new MyBean(2222, aUUID); }
    …
}                                                               MyConfig.java
```

Looking-Up Named Beans

- Call `getBean()` and specify the bean name you want:

```java
// Lookup 1st bean via its name.
MyBean ref1 = ctx.getBean("cool-bean", MyBean.class);

// Lookup 2nd bean via its various aliases.
MyBean ref2a = ctx.getBean("subsystemA-bean", MyBean.class);
MyBean ref2b = ctx.getBean("subsystemB-bean", MyBean.class);
MyBean ref2c = ctx.getBean("subsystemC-bean", MyBean.class);        Application.java
```



Lazily Instantiating a Singleton Bean

- You can set a bean to be lazily instantiated as follows:

```
@Configuration
public class MyConfig {

    @Bean(name="lazy-bean")
    @Lazy
    public MyBean bean3() { return new MyBean(3333, aUUID); }
    …
}
```

MyConfig.java

- Spring Boot will instantiate the bean "just in time"

Setting the Scope of a Bean

- You can set the scope of a bean as follows:

```
@Configuration
public class MyConfig {

    @Bean(name="proto-bean")
    @Scope("prototype")
    public MyBean bean4() { return new MyBean(4444, aUUID); }
    …
}                                                       MyConfig.java
```

- Spring Boot will instantiate a new bean every time you autowire or call `ctx.getBean()`

olsen software

4. Configuring Bean Dependencies

- Overview
- Configuring dependencies - technique 1
- Configuring dependencies - technique 2

Demo package: `demo.dependencies`

Overview

- Consider the following Java classes:

```
public class TransactionManager {

  DataSource dataSource;

  public void setDataSource(DataSource ds) {
    this.dataSource = ds;
  }
  …
}                    TransactionManager.java
```

```
public class DataSource {

  private String connectionString;
  private int maxPoolSize;
    …
}                    DataSource.java
```

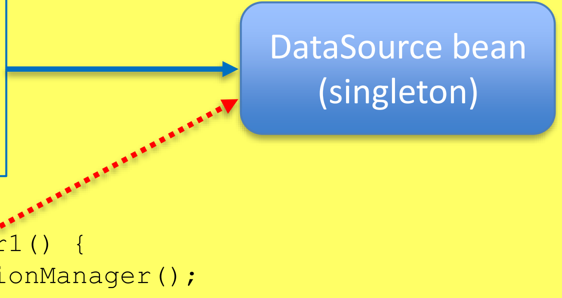
- Note:
  - The `TransactionManager` references a `DataSource`

olsen software

Configuring Dependencies - Technique 1

- You can configure dependencies as follows:

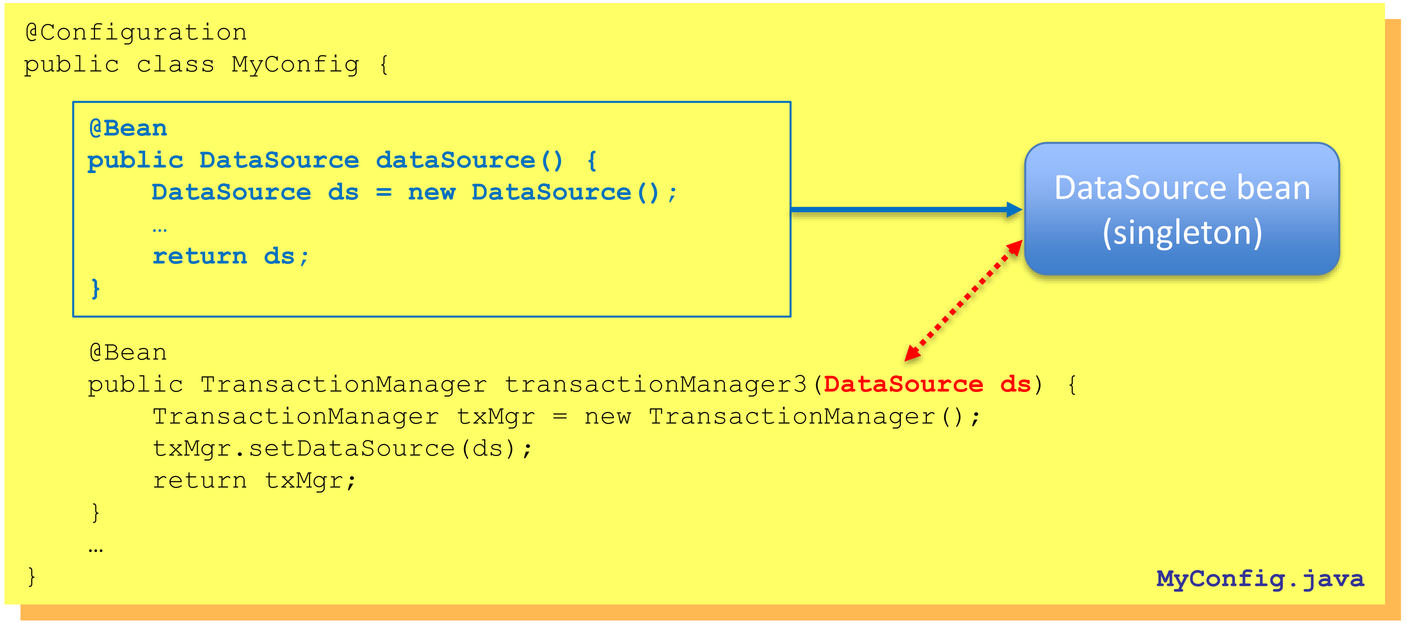
```java
@Configuration
public class MyConfig {

    @Bean
    public DataSource dataSource() {
        DataSource ds = new DataSource();
        …
        return ds;
    }

    @Bean
    public TransactionManager transactionManager1() {
        TransactionManager txMgr = new TransactionManager();
        txMgr.setDataSource(dataSource());
        return txMgr;
    }
    …
}
```



MyConfig.java

Configuring Dependencies - Technique 2

- Here's another way to configure dependencies:



```
@Configuration
public class MyConfig {

    @Bean
    public DataSource dataSource() {
        DataSource ds = new DataSource();
        …
        return ds;
    }

    @Bean
    public TransactionManager transactionManager3(DataSource ds) {
        TransactionManager txMgr = new TransactionManager();
        txMgr.setDataSource(ds);
        return txMgr;
    }
    …
}
```

olsen software

Summary

- Defining a config class and bean methods
- Locating config classes and bean methods
- Configuration techniques
- Configuring bean dependencies