Annotation-based Configuration

"Code with Passion!"

Topics

- Spring Annotations (related to Spring Beans)
- Two different ways of defining Spring Beans
  - *Through configuration*
  - *Through component-scanning*
- *@Component, @Controller, @Repository, @Service*
- *@SpringBootConfiguration, @EnableAutoConfiguration*
- *@Qualifier*
- *Custom qualifier*

Two ways of Defining Spring Beans

Two ways of Defining Spring Beans

- Through Java configuration class
- Through component-scanning

Defining Spring Beans through Configuration

@Configuration and @Bean

- Annotating a class with the *@Configuration* indicates that the class can be used by the Spring DI container as a source of bean definitions

```
@Configuration
public class AppConfig {

    // MyService is interface type
    @Bean
    public MyService myService() {
        return new MyServiceImpl();
    }


    @Bean
    public YourService yourService() {
        return new YourServiceImpl();
    }


}
```

AnnotationConfigApplicationContext

- *@Configuration* class is used as input when instantiating an *AnnotationConfigApplicationContext*

```
public static void main(String[] args) {

    // Read bean configuration defined in the AppConfig.class
    // and perform bean instantiation, configuration, wiring, and assembly
    ApplicationContext ctx =
                        new AnnotationConfigApplicationContext(AppConfig.class);

    // Retrieve MyClass object
    MyService myService = ctx.getBean(MyService.class);
    myService.doStuff();
}
```

Option #1: How to specify dependency

- Specify dependency as an arguiment

```
@Configuration
public class AppConfig {

   @Bean
   public TransferService transferService(AccountRepository accountRepository) {
      return new TransferServiceImpl(accountRepository);
   }

   @Bean
   public AccountRepository accountRepository() {
      return new InMemoryAccountRepository();
   }

}
```

TransferService has a dependency of AccountRepository

Option 2: How to specify dependency

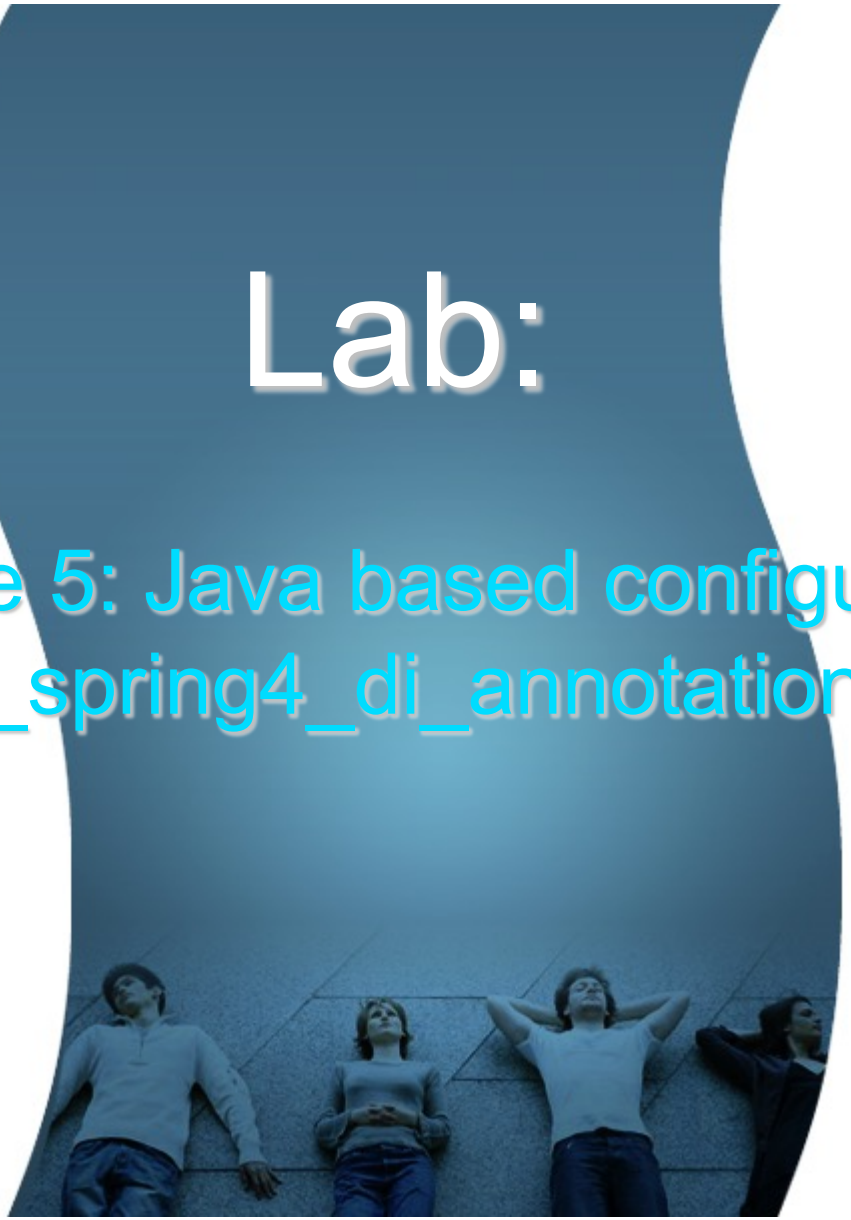
- Call a method that returns a dependency

```
@Configuration
public class AppConfig {

   @Bean
   public TransferService transferService() {
      return new TransferServiceImpl(accountRepository());
   }

   @Bean
   public AccountRepository accountRepository() {
      return new InMemoryAccountRepository();
   }

}
```

TransferService has a dependency of AccountRepository

Lab:

Exercise 5: Java based configuration
4939_spring4_di_annotation.zip



Defining Spring Beans through Component Scanning (@ComponentScan)

@ComponentScan

- No need to declare beans with @Bean annotations in the configuration
  - The beans needs to be annotated with @Component (or specialized annotations from @Component)
- Use *basePackages attribute* to define specific packages to scan
  - If specific packages are not defined, the scanning will occur from the package of the class with this annotation

Component Scan

- The specified package via base-package attribute – *com.jpassion.examples* package in the example below - will be scanned, looking for any *@Component*-annotated (and its stereo-typed annotations - *@Service, @Repository, @Controller*) classes, and those classes will be registered

```
@Configuration
@ComponentScan("com.jpassion.examples")
public class BeanConfiguration {

//    @Bean
//    public CustomerService getCustomerService() {
//        CustomerService customerService = new CustomerServiceImpl();
//        return customerService;
//    }
//
//    @Bean
//    public CustomerDao getCustomerDao() {
//        CustomerDao customerDao = new CustomerDaoImpl();
//        return customerDao;
//    }
}
```

Lab:

Exercise 6: @Service and @Repository Annotations

4939_spring4_di_annotation.zip

@Component & Further Stereotype Annotations (@Repository, @Service, @Controller)

@Component, @Repository, @Service, @Controller

- *@Component* is a generic stereotype for any Spring-managed component

- *@Repository, @Service, and @Controller* are specializations of *@Component* for more specific use cases (We are going to cover these in detail in Spring MVC topics)
  - @Repository – for persistence
  - @Service – for service
  - @Controller – for controller

@Repository, @Service, @Controller

- @Repository
  - A class that is annotated with "@Repository" is eligible for Spring org.springframework.dao.DataAccessException translation.
- @Service
  - A class that is annotated with "@Service" plays a role of business service
- @Controller
  - A class that is annotated with "@Controller" plays a role of controller in the Spring MVC application

@Profile

@Profile

- Spring Profiles provide a way to segregate parts of your application configuration and make it only available in certain environments
- Any @Component or @Configuration can be marked with @Profile to limit when it is loaded

```
@Configuration
@Profile("production")
public class ProductionConfiguration {

    // ...

}
```

- You can then set a spring.profiles.active Environment property to specify which profiles are active
- You can also specify the property in *application.properties* file

```
spring.profiles.active=production,mysql
```

Lab:

Exercise 7: @Profile

4939_spring4_di_annotation.zip

@SpringBootApplication
@EnableAutoConfiguration

@SpringBootApplication

- Composite annotation (Stereo annotation)
- Introduced as part of Spring Boot

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@Configuration
@EnableAutoConfiguration
@ComponentScan
public @interface SpringBootApplication {

/**
 * Exclude specific auto-configuration classes such that they will never be applied.
 * @return the classes to exclude
 */
Class<?>[] exclude() default {};

}
```

@EnableAutoConfiguration

- Enable auto-configuration of the Spring Application Context, attempting to guess and configure beans that you are likely to need
- Introduced as part of Spring Boot
- Auto-configuration classes are usually applied based on your classpath and what beans you have defined
  - If you have tomcat-embedded.jar on your classpath, you are likely to want a TomcatEmbeddedServletContainerFactory (unless you have defined your own EmbeddedServletContainerFactory bean)
- Auto-configuration tries to be as intelligent as possible and will back-away as you define more of your own configuration
  - You can always manually exclude() any configuration that you never want to apply
  - Auto-configuration is always applied after user-defined beans have been registered.

@Autowired

@Autowired

- Can be used in the Java source code for specifying DI requirement

- Places where *@Autowired* can be used
  - Constructor methods (constructor injection) – most preferred
  - Setter methods (setter injection)
  - Arbitrary methods
  - Fields – least preferred

@Autowired at Constructor method

```
public class MovieRecommender {

  private CustomerPreferenceDao customerPreferenceDao;

  @Autowired // used at the constructor – optional from Spring 4.3 if this is only constructor
  public MovieRecommender(
      CustomerPreferenceDao customerPreferenceDao) {
      this.customerPreferenceDao = customerPreferenceDao;
  }

  // ...
}
```

@Autowired at Setter method

```
public class SimpleMovieLister {

  private MovieFinder movieFinder;

  // MovieFinder object gets created and injected by Spring DI container
  @Autowired
  public void setMovieFinder(MovieFinder movieFinder) {
     this.movieFinder = movieFinder;
  }

  // ...
}
```

@Autowired at arbitrary methods

- You can also apply @Autowired annotation to methods with arbitrary names and/or multiple arguments:

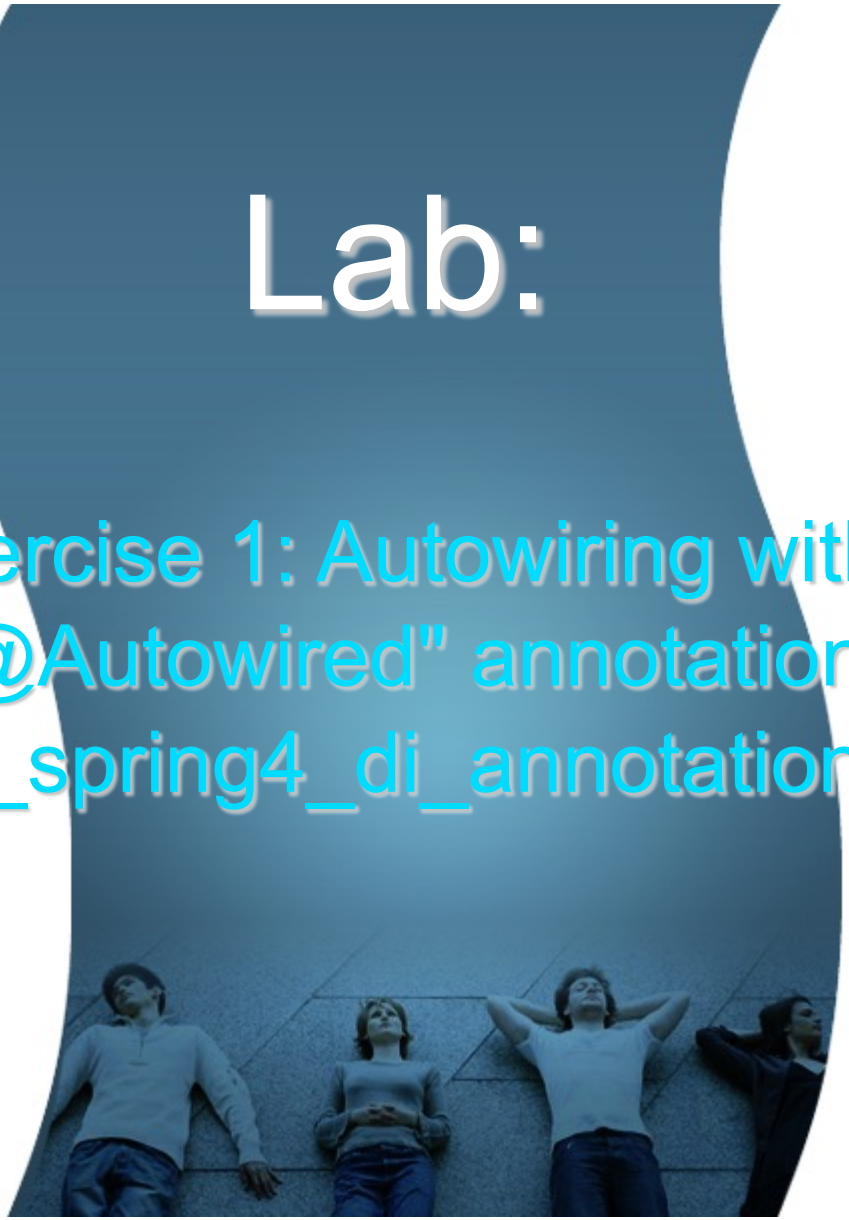
```
public class MovieRecommender {

  private MovieCatalog movieCatalog;
  private CustomerPreferenceDao customerPreferenceDao;

  // MovieCatalog and CustomerPreferenceDao objects are
  // injected automatically
  @Autowired
  public void prepare(MovieCatalog movieCatalog,
                      CustomerPreferenceDao customerPreferenceDao) {
     this.movieCatalog = movieCatalog;
     this.customerPreferenceDao = customerPreferenceDao;
  }

  // ...
}
```

@Autowired at Field

```
public class MovieRecommender {

  @Autowired  // used at the field
  private MovieCatalog movieCatalog;


  // ...
}
```

Lab:

Exercise 1: Autowiring with "@Autowired" annotation
4939_spring4_di_annotation.zip



@Qualifier, @Primary

Fine-tuning @Autowired with Qualifiers

- Because autowiring by type may lead to multiple candidates, it is often necessary to have more control over the selection process

- One way to accomplish this is with Spring's *@Qualifier* annotation

```
public class MovieRecommender {

  // Among the multiple candidates of MovieCatalog type, select
  // the one that has the bean name “main”.
  @Autowired
  @Qualifier("main")
  private MovieCatalog movieCatalog;

  // ...
}
```

Fine-tuning @Autowired with @Qualifier

- The @Qualifier annotation can also be specified on individual constructor arguments or method arguments

```
public class MovieRecommender {

  private MovieCatalog movieCatalog;
  private CustomerPreferenceDao customerPreferenceDao;

  @Autowired
  public void prepare(
                  @Qualifier("main") MovieCatalog movieCatalog,
                  CustomerPreferenceDao customerPreferenceDao) {
     this.movieCatalog = movieCatalog;
     this.customerPreferenceDao = customerPreferenceDao;
  }

  // ...
}
```

Qualifier name is usually bean name

```
@Configuration
public class BeanConfiguration {

    @Bean(name = "myaddress")
    public AddressInterface getMyAddress() {
        AddressInterface address = new MyAddress();
        return address;
    }

    @Bean(name = "youraddress")
    public AddressInterface getYourAddress() {
        AddressInterface address = new YourAddress();
        return address;
    }

    @Bean
    public Person getPerson() {
        Person person = new Person();
        return person;
    }

}
```

There are two candidates to AddressInterface type: MyAddress and YourAddress.

Give name to these candidates so that they can be specified in the selection process

Custom Qualifier

Creating Custom Qualifier Annotation

- You can create your own custom qualifier annotations.

```
// Create custom qualifier annotation called “Genre”
@Target({ElementType.FIELD, ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
@Qualifier
public @interface Genre {

  String value();
}
```

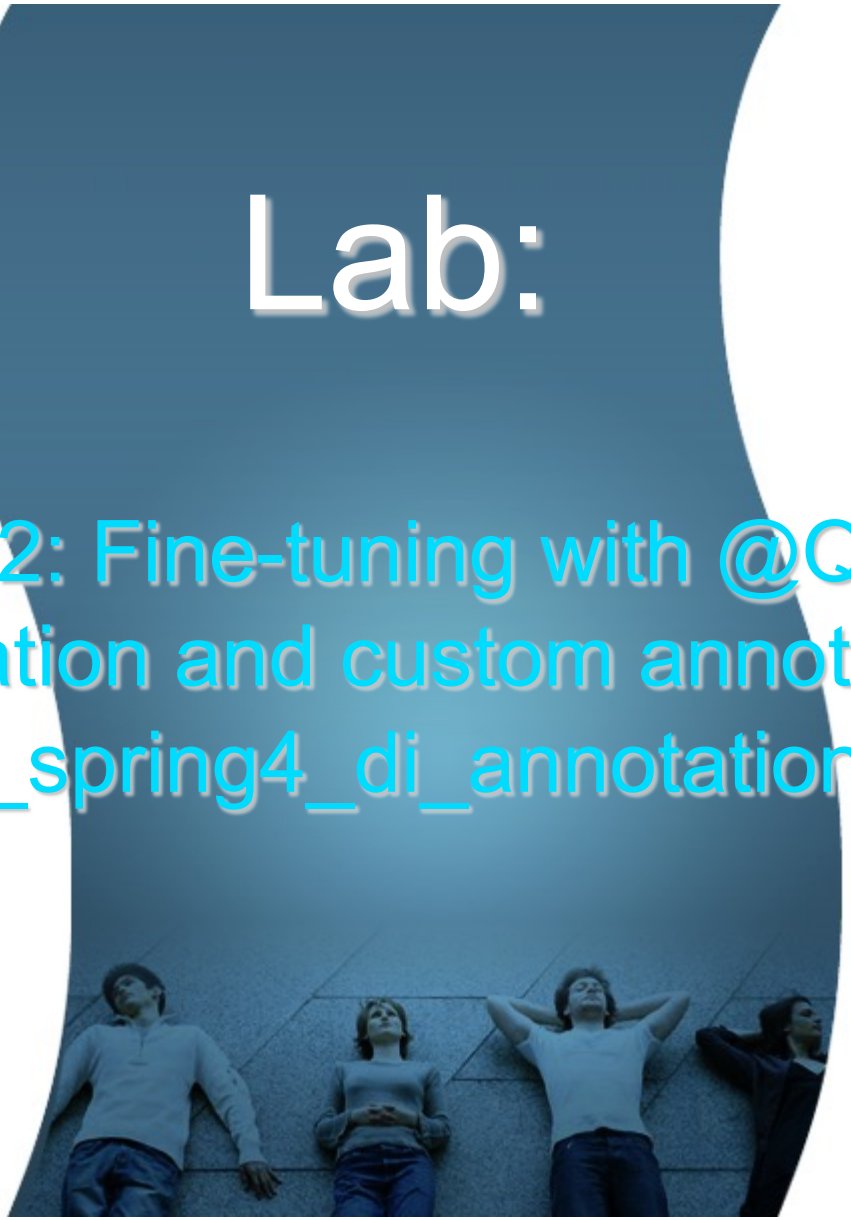
@Autowired with Custom Qualifier

- Then you can provide the custom qualifier annotation on autowired fields and parameters:

```
public class MovieRecommender {

  @Autowired
  @Genre("Action")
  private MovieCatalog actionCatalog;

  private MovieCatalog comedyCatalog;

  @Autowired
  public void setComedyCatalog(
      @Genre("Comedy" ) MovieCatalog comedyCatalog) {
      this.comedyCatalog = comedyCatalog;
  }

  // ...
}
```

Lab:

Exercise 2: Fine-tuning with @Qualifier annotation and custom annotation
4939_spring4_di_annotation.zip



Code with Passion!