Spring Beans

"Code with Passion!"

Java-based Container 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 {

    @Bean
    public MyService myService() {
        return new MyServiceImpl();
    }
}
```

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();
}
```

@Configuration and @Bean

- A case where a bean has a dependency bean

```
@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

@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

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)

- One of basePackageClasses(), basePackages() or its alias value() may be specified to define specific packages to scan
  - If specific packages are not defined 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;
//    }
}
```

No need to manually configure beans

@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
```

@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.

Code with Passion!