selsoft
build better, deliver faster

Enterprise Application Development with Spring

Chapter 7: Java-based Configuration

Instructor

Akın Kaldıroğlu

Expert for Agile Software Development and Java

Topics

- **Java-based Confguration**
  - @Bean and @Configuration
  - @ComponentScan and @Import
- **Java's DI Mechanisms**
  - Support for JSR-250 & @Resource
  - Support for JSR-330 & @Inject & @Named

Java-Based Configuration

selsoft
build better, deliver faster

Java-based Configuration

- **Spring** allows configuring the container using Java code.
- Main artifacts for Java-based configuration is **`@Bean`** and **`@Configuration`**.
- **Spring** also supports Java's standard injection mechanisms:
  - As part of JSR-250 **`@Resource`**
  - As part of JSR-330 standard DI annotations such as **`@Inject`**

selsoft

build better, deliver faster

@Bean, @Configuration, @ComponentScan and @Import

@Bean - I

- `org.springframework.context.annotation.Bean` is an annotation used for methods.
- `@Bean` makes a method a factory to produce a bean to be managed by the **Spring** container.
    - No need to annotate the classes whose instances will be created in `@Bean`-annotated methods with `@Component`.
- `@Bean` provides the same semantics as `</bean>` in XML file.

@Bean - II

- `@Bean`-annotated methods can be declared in any Spring `@Component` in which case a bean object produces another bean object.
- However, they are most often used with `@Configuration` beans.
- Because it is best to put together factory methods into separate configuration classes annotated with `@Configuration`.

@Configuration - I

- **`org.springframework.context.annotation.Configuration`** is an annotation used for classes.
- **`@Configuration`** indicates that a class declares **`@Bean`** methods.
- It is processed by the container to generate bean definitions and service requests for those beans at runtime.
- **`@Configuration`** classes are typically bootstrapped using either **`AnnotationConfigApplicationContext`** or its web-capable variant, **`AnnotationConfigWebApplicationContext`**.

@Configuration - II

- `Configuration` has two attributes:
  - `value` is a `String` and represents `Configuration` name.
  - `proxyBeanMethods` is a `boolean` which is `true` in default.

Bootstraping @Configuration Classes - I

- There are several ways to register configuration classes:
  - **`AnnotationConfigApplicationContext`** is used to bootstrap a stand-alone context that uses annotations.

```
AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext();
ctx.register(AppConfig.class);
ctx.refresh();
MyBean myBean = ctx.getBean(MyBean.class);
```

  - The XML configuration file can be used to register configuration classes.

```
<beans>
   <context:annotation-config/>
   <bean class="org.javaturk.MyConfig"/>
</beans>
```

  - Through component scan.

AnnotationConfigApplicationContext - I

- `org.springframework.context.annotation.AnnotationConfigApplicationContext` is a class that implements both `BeanFactory` and `ApplicationContext` interfaces.
- It accepts component classes such as `@Component` beans and others produced by `@Configuration`-annotated classes.
- `AnnotationConfigApplicationContext` also accepts plain types and JSR-330 compliant classes using `javax.inject` annotations.

AnnotationConfigApplicationContext - II

- **`AnnotationConfigApplicationContext`** can register component classes and beans.
- It can also scan packages.
- It has constrcutors and methods to do these.

@ComponentScan

- `org.springframework.context.annotation.ComponentScan` is an annotation that provides component scanning directive for use with all `@Component` classes.
    - `@Configuration` is a kind of `@Component` and therefore is subject to classpath scanning.
- `@ComponentScan` has an attributes for packages to scan.
- In default it starts scanning from the package the `@ComponentScan` resides.

@Import

- `org.springframework.context.annotation.Import` is an annotation to import one or more component classes.

ConfigurationExample

- `org.javaturk.spring.di.ch07.configuration.ConfigurationExample`

greeting17

- `org.javaturk.spring.di.ch07.greeting.greeting17.Application`

greeting18

- `org.javaturk.spring.di.ch07.greeting.greeting18.Application`
  - Uses **`@ComponentScan`** to scan all beans annotated with **`@Component`**.
  - Notice that no XML configuration file is used.

@Bean and @Configuration

- The `@Bean` annotation doesn't have any attribute for profile, scope, lazy, depends-on or primary.
- `@Scope`, `@Lazy`, `@DependsOn`, `@Primary` and `@Qualifier` annotations should be used with `@Bean` to get the necessary effect.
- If `@Lazy` is used with `@Configuration` then all beans produced with `@Bean` methods will be initialized lazily.

@Profile

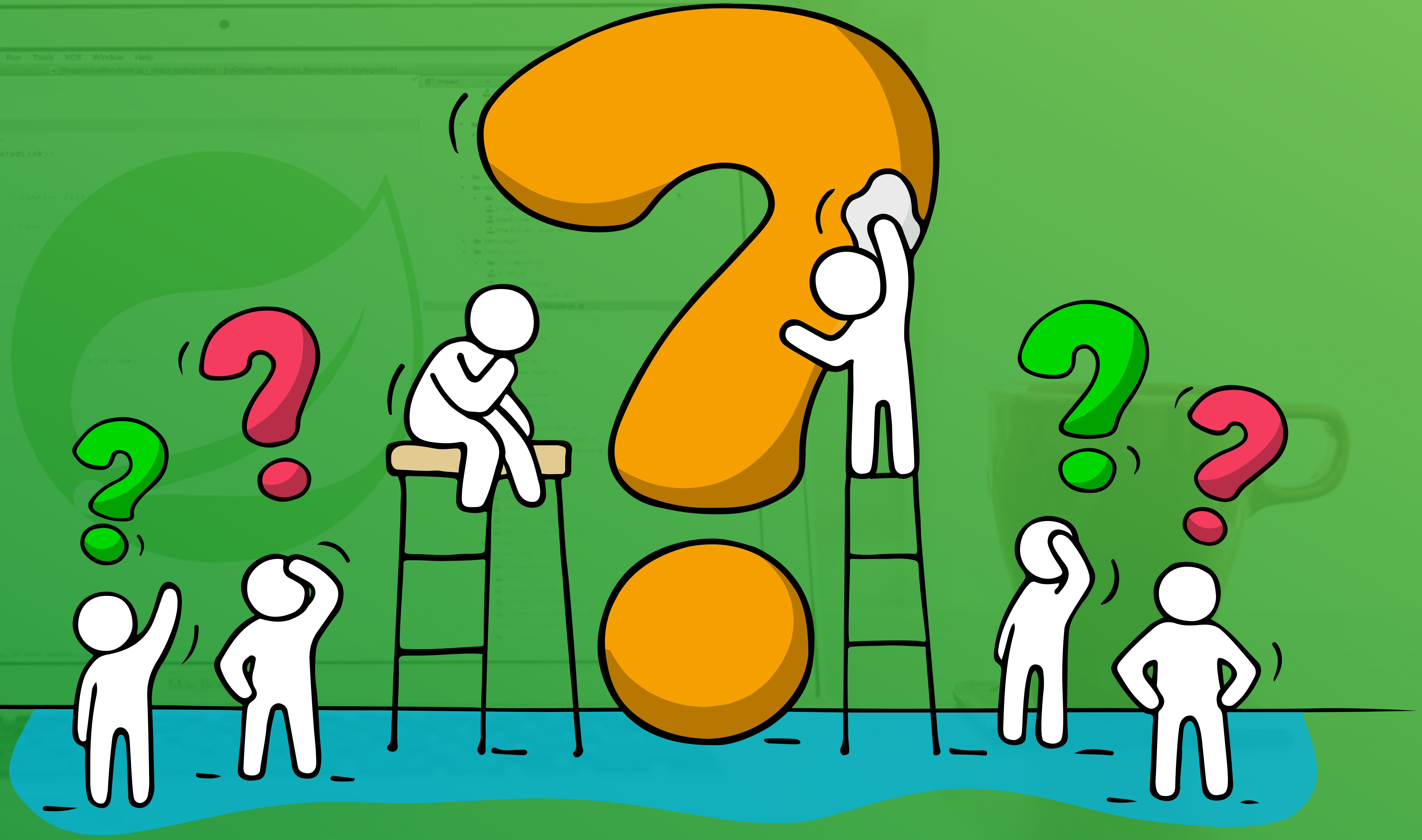
- A class that is annotated with `@Configuration` can have a `@Profile` annotation too.
- In this case all of the `@Bean` methods and `@Import` are associated with specified profiles.

Exercise

Exercise

- `org.javaturk.spring.di.ch07.ex.calculator.conf.Test`
- Use `@Configuration` and `@Beans` to create beans.

Time for Questions!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

selsoft
build better, deliver faster

Lite @Bean

Lite @Bean - I

- The factory `@Bean` methods can be defined in a `@Component` or a regular class in which case they are called **lite `@Bean`** methods.

- While `@Bean` methods in `@Configuration` classes can produce beans to handle inter-bean dependencies, lite `@Bean` methods cannot declare inter-bean dependencies, their functionality is valid only in their classes, producing necessary beans and values only for the state of their classes.
  - It is also called **lite mode** vs. **full mode**.

- Lite mode can be used in order to make components the factory of its dependencies.

Lite @Bean - II

- If a class with `@Component` annotation has its own `@Bean` methods i.e. using lite mode then for injections its own `@Bean` methods are called.
- If the injection is made into a constructor then those `@Bean` methods must be static due to the fact that the object itself has not been created yet.
  - Otherwise `org.springframework.beans.factory.BeanCurrentlyInCreationException` with the message *Error creating bean with name 'Xxx': Requested bean is currently in creation: Is there an unresolvable circular reference?* is thrown.

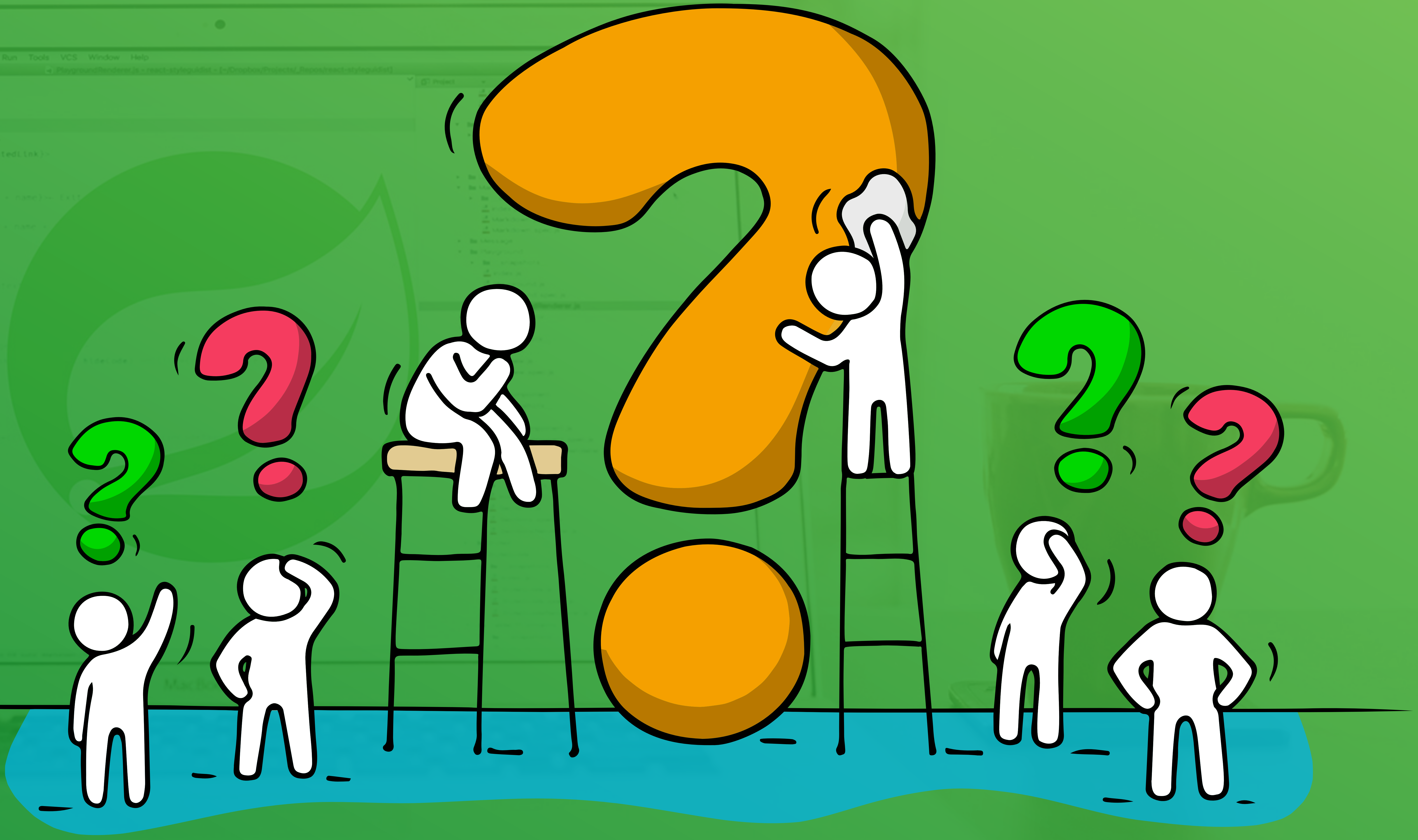
Lite @Bean - III

- For field and property injections instance methods with `@Bean` can be used.

LiteBeanExample

- `org.javaturk.spring.di.ch07.liteBean.LiteBeanExample`

Time for Questions!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

Java's DI Mechanisms

selsoft

build better, deliver faster

Java’s Standard DI Annotations

- **Spring** supports Java’s standard injection methods:
  - As part of JSR-250 `@Resource`
  - As part of JSR-330 standard DI annotations such as `@Inject` and `@Named`

Support for JSR-250

JSR-250 Annotations

- **Spring** 2.5 added support for JSR-250 annotations:
  - `@Resource` will be introduced here.
  - `@PostConstruct` and `@PreDestroy` will be introduced later in lifecycle management.
- These are in `javax.annotation` package which is part of Java EE and was also part of `java.xml.ws` module of JDK.
- Starting version 11, this module is not part of the JDK anymore so its artifacts should be added to the project separately.

Support for JSR-250

@Resource

@Resource - I

- `javax.annotation.Resource` is an annotation used on fields and property setter methods for injection.
- `@Resource` takes several attributes one of which is `name`.
- **Spring** takes the value of `name` attribute as the bean name to be injected.
- If no name is specified, the default name is derived from the field name or the property name if a setter method is annotated.

@Resource - II

- All of the **Spring**'s qualification mechanisms work well with `@Resource`.
- The main use case to use `@Resource` with **Spring** might be having a piece of Java code that had already used `@Resource` and reusing it in a new project where **Spring** is utilized.
- Other than that use case there is no need to use `@Resource` in a project that uses **Spring**.

greeting19

- `org.javaturk.spring.di.ch07.greeting.greeting19.Application`
  - Observe the injection of beans using `@Resource`
  - Observe how injected beans are resolved through naming convention, `@Qualifier` and custom qualifier.

Support for JSR-330

JSR-330 - I

- JSR-330 is a specification for Dependency Injection in Java EE.
  - More generally it is called **Context and Dependency Injection** (**CDI**).
- It has been led by Rod Johnson of SpringSource which was the name of the company for **Spring** framework at 2009) and Bob Lee of Google.
- Its main annotations **`@Inject`** and **`@Named`** in **`javax.inject`** package.
  - CDI 2.0 is part of Java EE 8 and 3.0 will be part of Jakarta EE 9.0.

JSR-330 - II

- Weld (https://weld.cdi-spec.org/) is the reference implementation of DI for Java EE platform.
    - There are some other implementations such as Apache Commons Inject (https://commons.apache.org/sandbox/commons-inject/index.html).
- As of now the DI spec is implemented as Weld 3.1.5.
    - 3.0 is being implemented as Weld 4.
- **Spring** 3.0 added support for JSR-330 annotations.

JSR-299

- JSR-299 is another specification for **Context and Dependency Injection** (**CDI**) for Java.
- It has been led by Gavin King of RedHat.
- JSR-299 is built on the top of JSR-330 and adds some advanced features.

selsoft
build better, deliver faster

Support for JSR-330

@Inject and @Named

@Inject

- `@Inject` can be used instead of `@Autowired`.
- It has no attribute and it can be used at field, constructor and method level.
- `@Inject` injects any Java object which is a POJOs.
  - There is no need to mark POJOs to be injected by `@Inject`.
- Only configuration needed for DI to work is a `beans.xml` file in `META-INF` folder in the root of the packages.

@Named

- `@Named` does the same functionality of `@Component`.
- It has an attribute called `value` of type `String` which designates the string-based qualifier.
- It is also used to qualify beans for injection.
  - In this usage it has the same functionality of `Qualifier` of **Spring**.
- In fact `javax.inject.Qualifier` can be used to build custom qualifiers.

@Inject & @Named

- The main use case to use JSR-330's injection mechanism with **Spring** might be as with `@Resource`, having a piece of Java code that had already used JSR-330's injection annotations and reusing it in a new project where **Spring** is utilized.
- Other than that use case there is no need to use JSR-330's injection mechanisms in a project that uses **Spring**.
- That's because **Spring** provides all kinds of DI structures.

HelloWorldjavaCDI

- `org.javaturk.cdi.hello1`, `hello2` and `hello3`.
  - This example is built totally using Java’s standard injection mechanism.

InjectEample

- **`org.javaturk.spring.di.ch07.inject.InjectExample`**
  - Observe the injection of beans using **`@Inject`**.
  - Notice injected beans and values can be produced by a configuration object.

selsoft

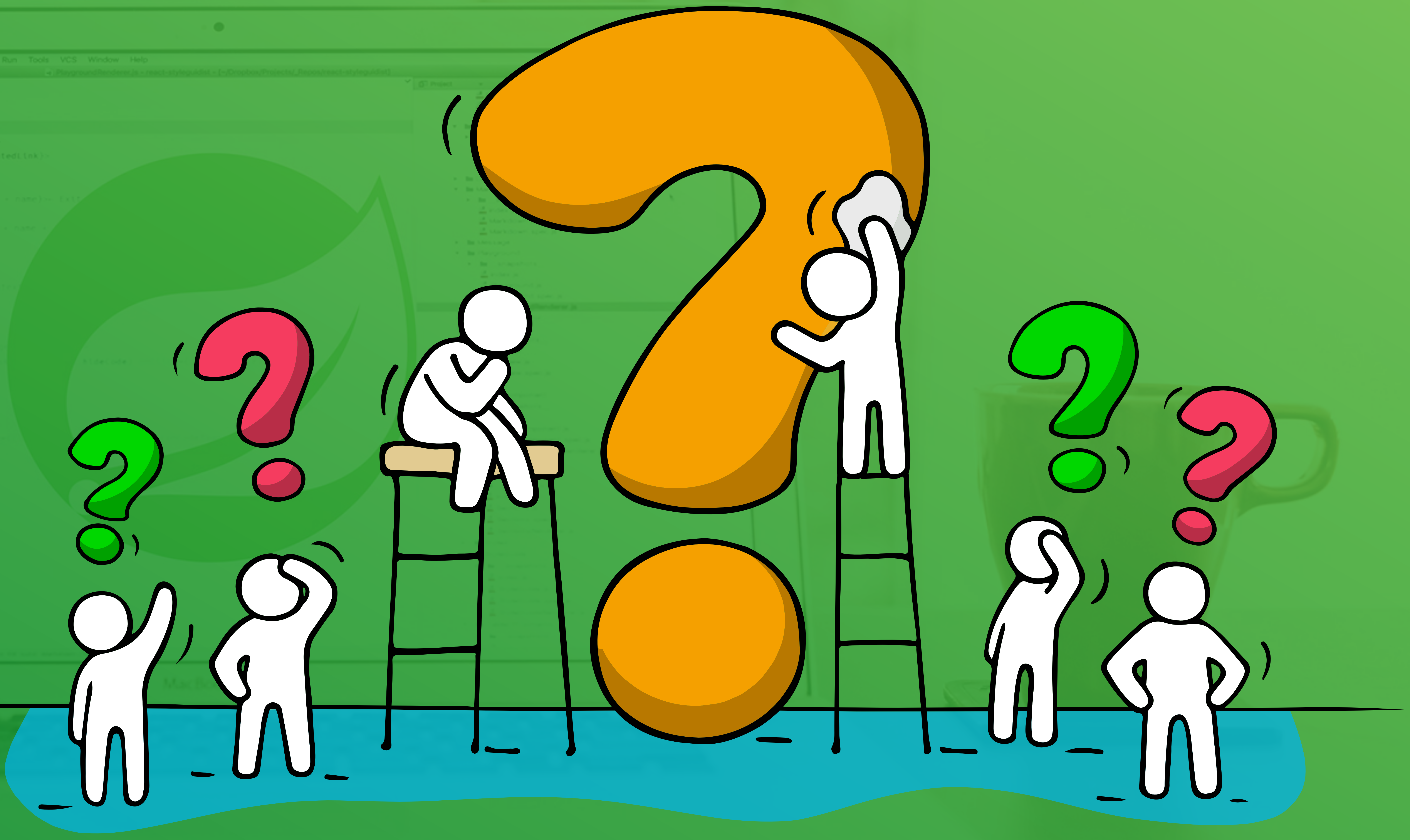
build better, deliver faster

Exercise

Exercise

- `org.javaturk.spring.di.ch07.ex.calculator.inject.Test`
- Use `@Inject` and `@Named` to inject beans.

Soru ve Cevap Zamanı!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr