selsoft

build better, deliver faster

Enterprise Application Development with Spring

Chapter 4: Spring IoC Container

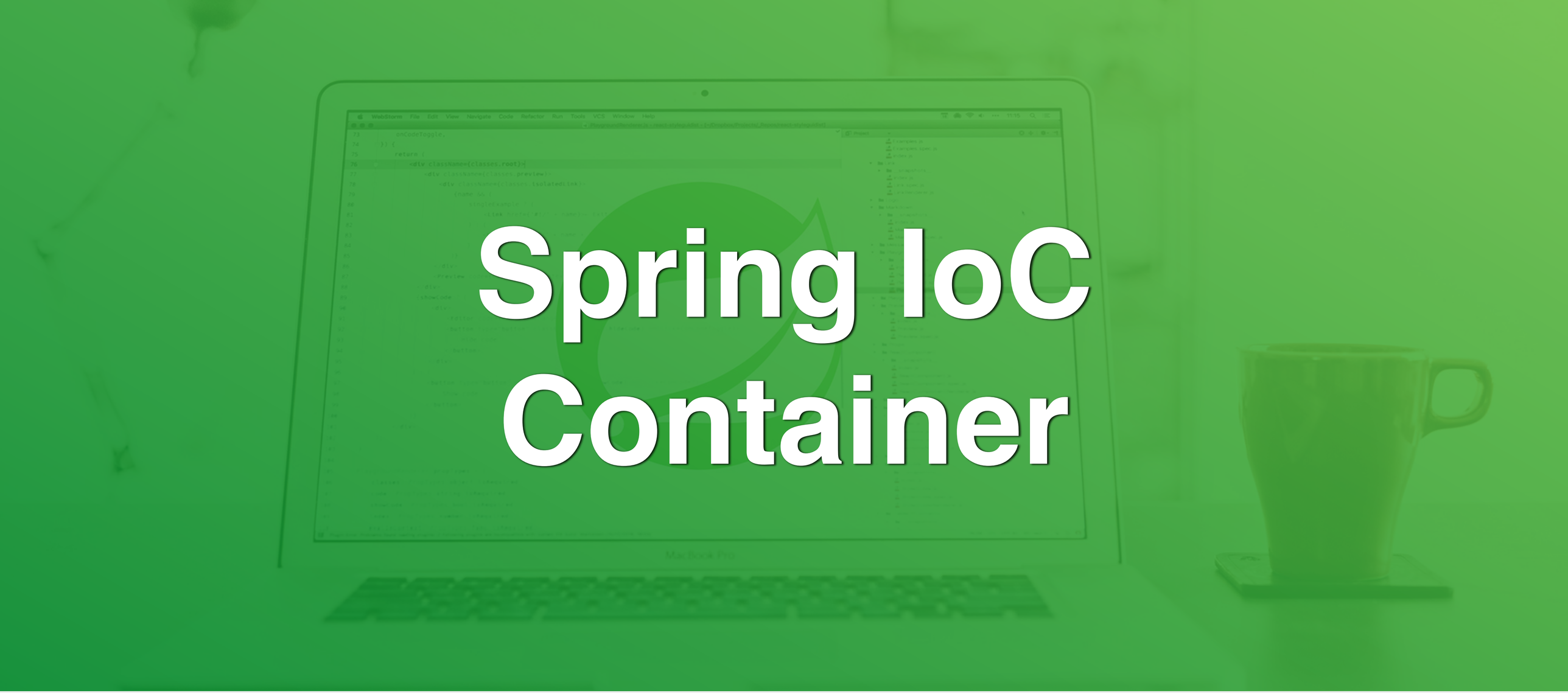
Instructor

Akın Kaldıroğlu

Expert for Agile Software Development and Java

Topics

- **Spring IoC Container**
  - Main Objects
- **Configuration Metadata**



Spring IoC Container

selsoft
build better, deliver faster

What is Container?

- Container is an abstract cup that provides services for the objects.
- Depending on the nature and type of the container it provides such as life cycle management, resource and dependency injection, AOP, transaction management, etc.
- So the objects live in that container and get those services via configuration or API calls on the objects of the container.
- Web container such as Tomcat provides services for servlet objects to handle HTTP requests.

Spring IoC Container

- **Spring** framework has its own container that works using IoC and provides DI mechanisms to create objects and manage their life cycles and dependencies.
- **Spring** IoC container provides other ways to create objects and manage dependencies such as factory methods where the application pulls objects.
- But IoC and DI is at the core of **Spring** framework and **Spring** uses it anywhere applicable.

Main Objects

Bean - I

- Objects that are managed by the **Spring** IoC container are called beans.
- **A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IoC container.**
- Beans and the dependencies among them are reflected in the configuration metadata used by the container.
- All objects or beans in the application that are not created by **Spring** are not managed by **Spring**.

Bean - II

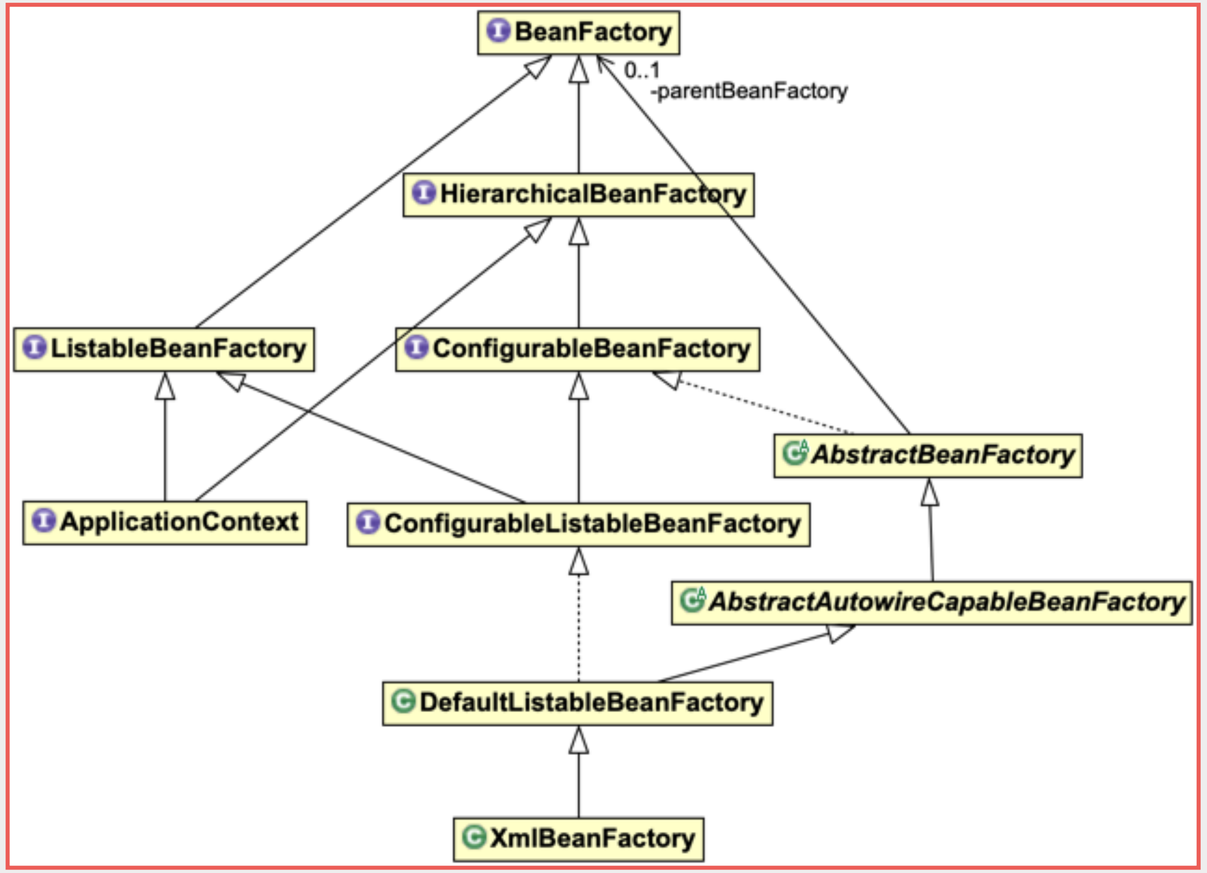
- Bean in this sense is a little bit different from JavaBean.
- A JavaBean must have a default constructor while bean in the context of **Spring** does not have to because dependency injection can be done through the constructor of the depended object.

Main Objects

BeanFactory

BeanFactory - I

- `org.springframework.beans.BeanFactory` is the root interface for accessing the **Spring** IoC container.
- **Spring**'s Dependency Injection functionality is implemented using `BeanFactory` and its subtypes.



BeanFactory - II

- `BeanFactory` provides basic functionality to manage beans in **Spring**.
  - Managing includes first creating objects and then injecting them to each other to satisfy their dependencies.
- Other objects down in the herarchy adds some other necessary functionalties necessary for a full-fledged dependency injection mechanism

BeanFactory - III

- **`BeanFactory`** provides methods to query a bean either by its name or by its type.

```
Object getBean(String name)
T getBean(Class<T> requiredType)
T getBean(String name, Class<T> requiredType)
```

- Notice that if the bean is queried by only a **`String`** argument the returned object would be **`java.lang.Object`** instead of the specific type.
  - In this case the object needs to be cast to its type.

BeanFactory - IV

- If **`BeanFactory`** couldn't find a bean that is looked up it throws **`NoSuchBeanDefinitionException`**.

- If **`BeanFactory`** finds that there are more than one bean defined for a specific type it throws **`NoUniqueBeanDefinitionException`**. beause **`BeanFactory`** provides no methods to return all beans instances for a specific type.

BeanFactoryExample

- `org.javaturk.spring.di.ch04.BeanFactoryExample`

Main Objects

ListableBeanFactory

ListableBeanFactory - I

- **`org.springframework.beans.factory.ListableBeanFactory`** is a sub interface of **`BeanFactory`** and adds some enumeration capabilities for beans.
- It allows to get bean definitions and beans of a specific type as collections.

```
String[]                getBeanDefinitionNames()
String[]                getBeanNamesForType(Class<?> type)
<T> Map<String,T>       getBeansOfType(Class<T> type)
```

ListableBeanFactory - II

- If **`BeanFactory`** finds that there are more than one bean defined for a specific type it throws **`NoUniqueBeanDefinitionException`**.
- So in case of having more than one bean definition for a specific type use **`ListableBeanFactory`** instead of **`BeanFactory`** to get all beans of that type.

ListableBeanFactoryExample

- `org.javaturk.spring.di.ch03.ListableBeanFactoryExample`

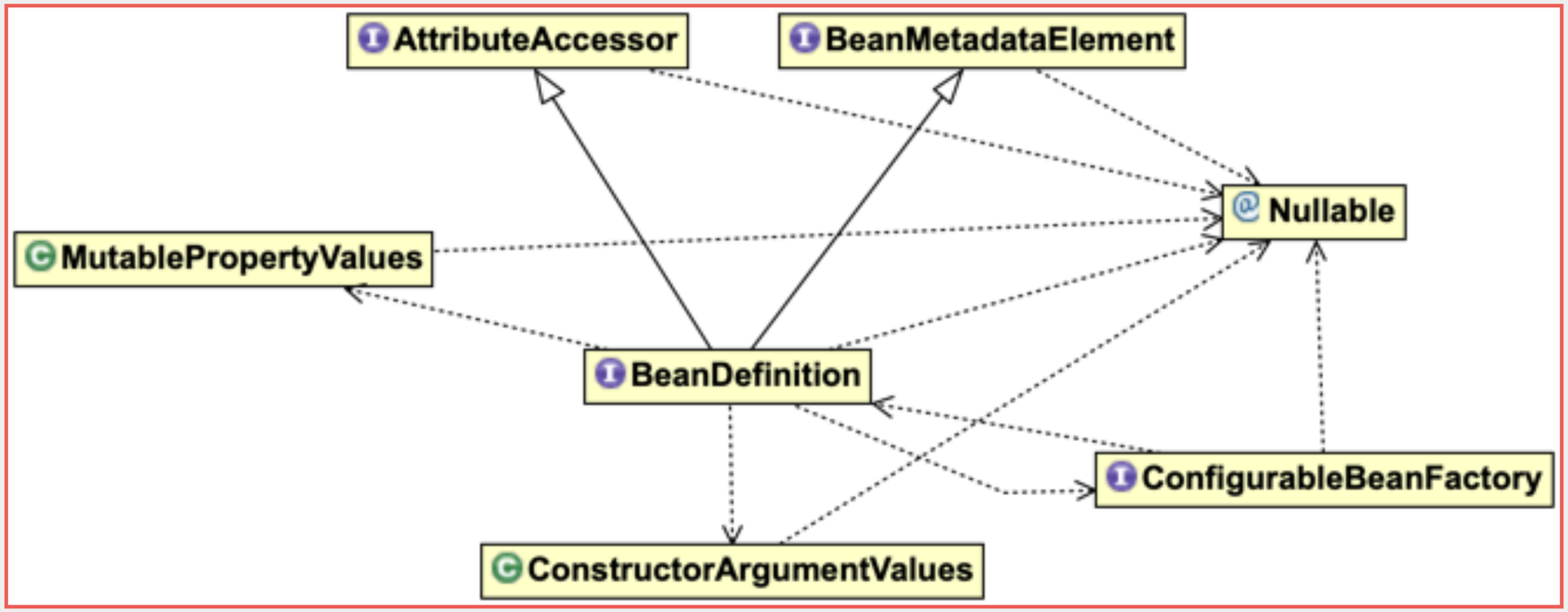
Main Objects

BeanDefinition & Bean Definition Files

BeanDefinition - I

- **Spring** container represents bean definitions by `org.springframework.beans.factory.config.BeanDefinition` objects.
- It is an interface and describes a bean instance, which has property values, constructor argument values, and further information supplied by concrete implementations.

BeanDefinition - II



Bean Definition Files - I

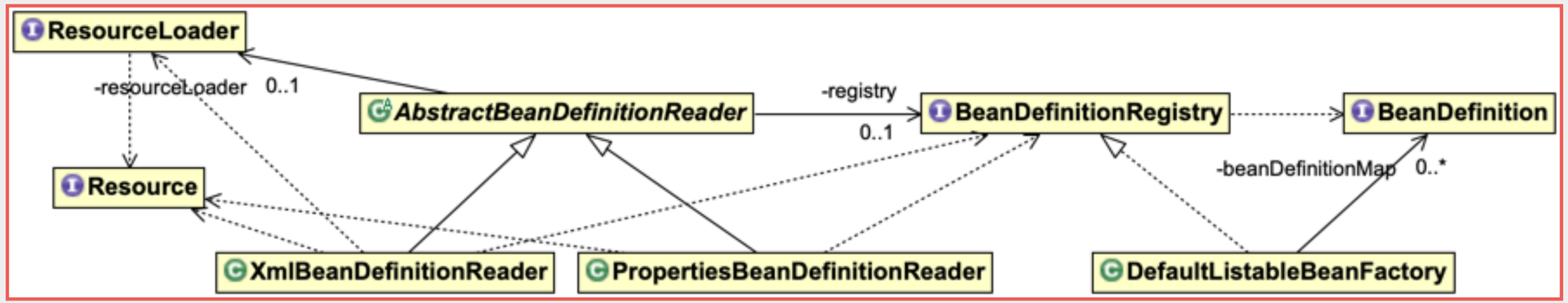
- Beans can be defined either in XML files or properties files.
- **`XmlBeanDefinitionReader`** and **`PropertiesBeanDefinitionReader`** can be used for this purpose.
- They are both in **`org.springframework.beans.factory.support`** package and extends **`AbstractBeanDefinitionReader`**.
- They both load bean definition files via **`loadBeanDefinitions()`** methods.

Bean Definition Files - II

- They are typically used with `DefaultListableBeanFactory`, which is a `BeanDefinitionRegistry`.

```
PropertiesBeanDefinitionReader(BeanDefinitionRegistry registry)
```

```
XmlBeanDefinitionReader(BeanDefinitionRegistry registry)
```



Main Objects

DefaultListableBean Factory

DefaultListableBeanFactory - I

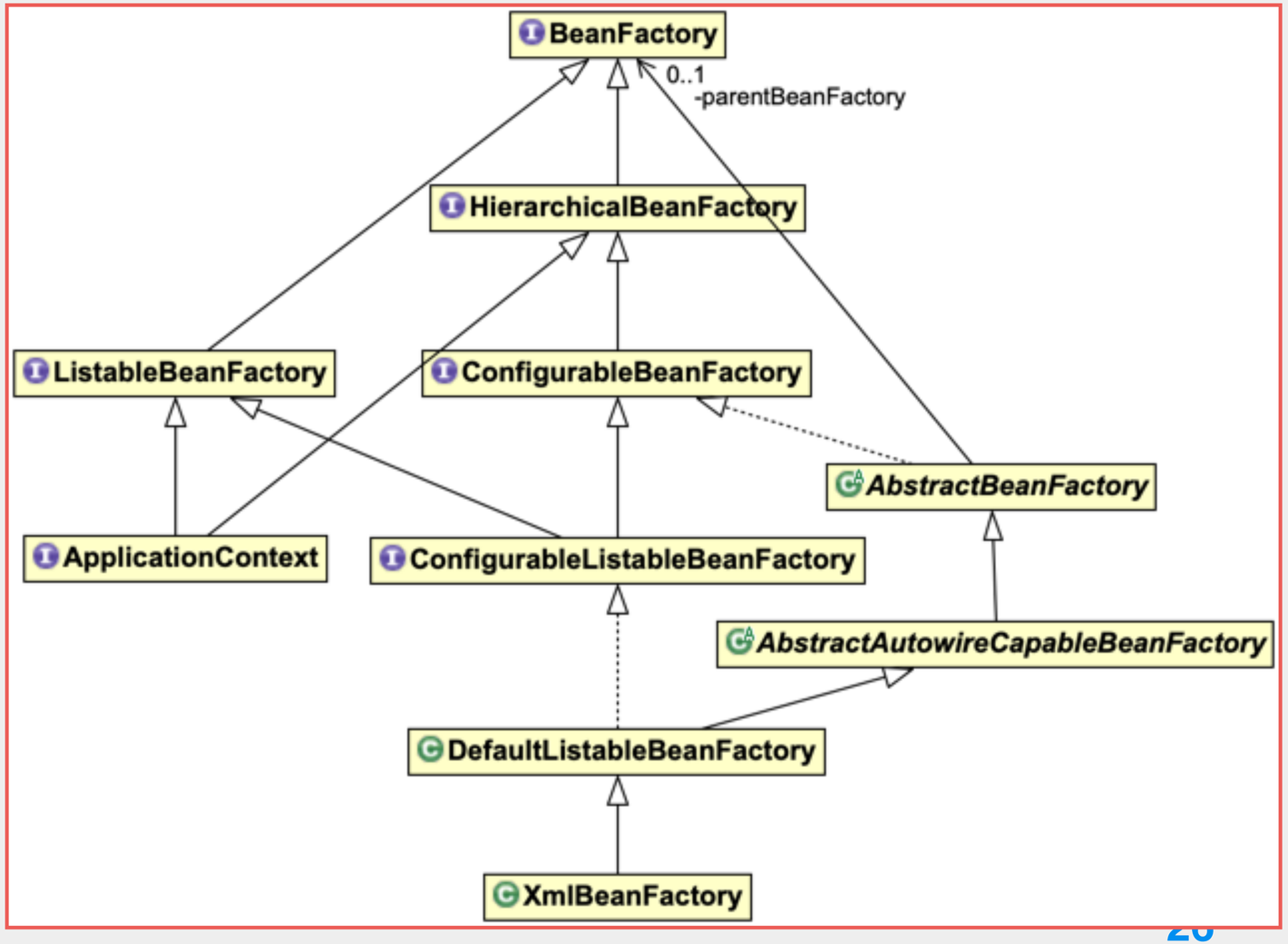
- `org.springframework.beans.factory.support.DefaultListableBeanFactory` is a sub interface of `BeanFactory` and `ListableBeanfactory`

- It is a full-fledged bean factory based on bean definition metadata that provides many functionalities on beans and dependencies.

```
DefaultListableBeanFactory()

DefaultListableBeanFactory(BeanFactory parentBeanFactory)
```

DefaultListableBeanFactory - II



- `BeanFactory` hiearchy is as shown.
- `org.springframework.beans.factory.xml.XMLBeanFactory` is deprecated.

DefaultListableBeanFactoryExample

- `org.javaturk.spring.di.ch03.DefaultListableBeanFactoryExample`

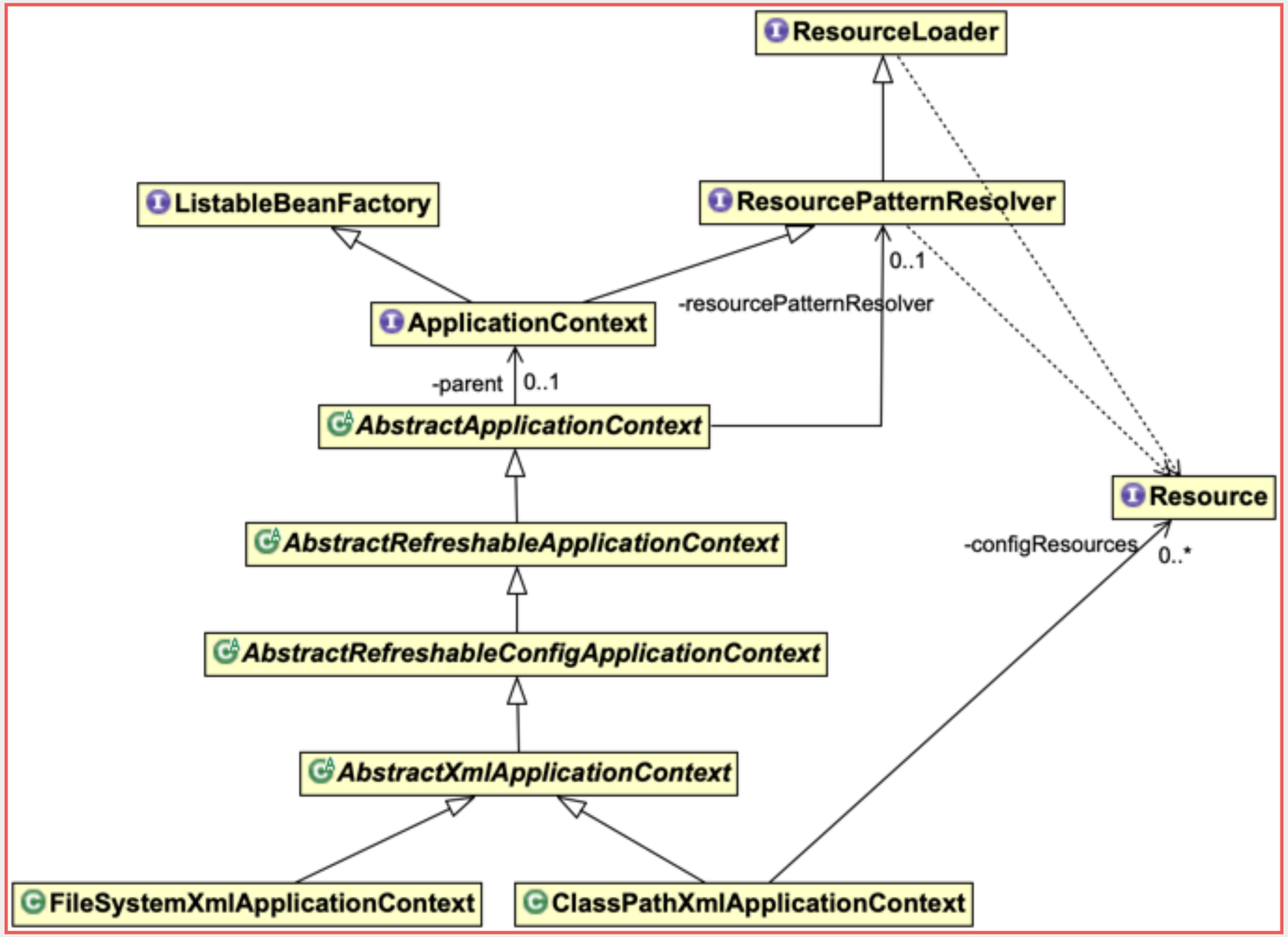
Main Objects

ApplicationContext

ApplicationContext - I

- `org.springframework.context.ApplicationContext` is a sub interface of `BeanFactory` and adds more application-specific functionalities such as loading the configuration for beans because it is a `ResourceLoader`.
- It is the parent of application-layer specific contexts such as the `org.springframework.web.context.WebApplicationContext` for web applications.

ApplicationContext - II



ApplicationContextExample

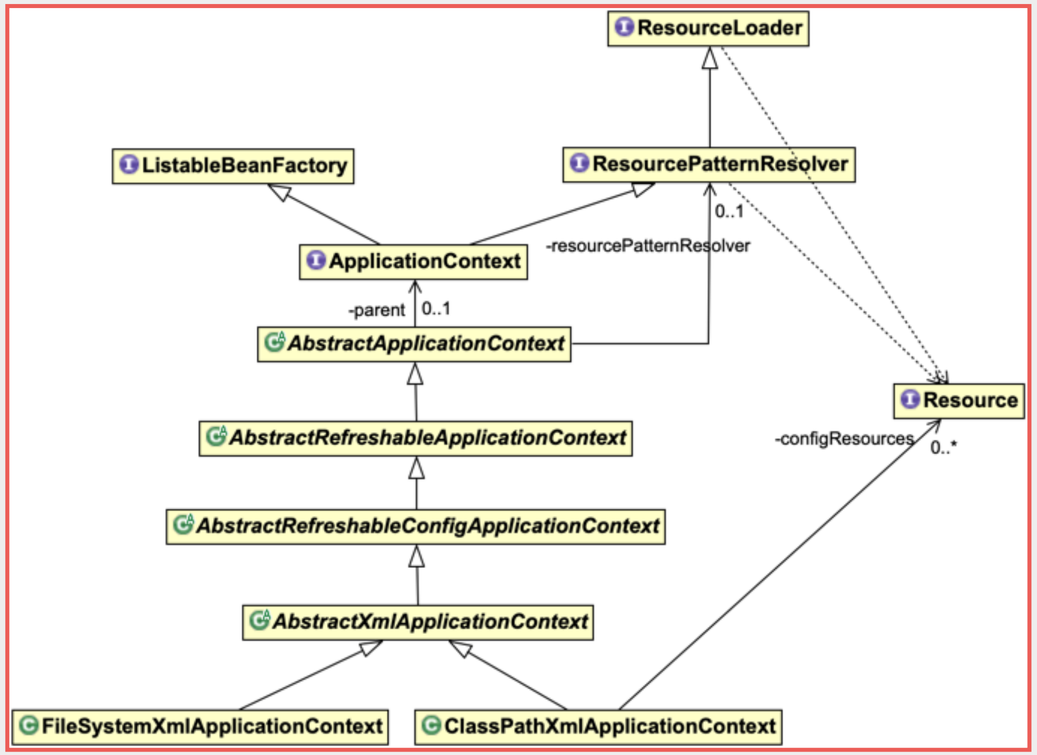
- `org.javaturk.spring.di.ch03.ApplicationContextExample`

ApplicationContext - III

- **`ApplicationContext`** is used more frequently than **`BeanFactory`**.
  - The terms **application context** or **context** refers to the configuration of Spring IoC that **`ApplicationContext`** represents.
- It also provides event publication and message resource handling for internationalization.

ApplicationContext - IV

- **`ApplicationContext`** has two main implementations for stand-alone applications, **`ClassPathXmlApplicationContext`** and **`FileSystemXmlApplicationContext`**
- In other kinds of applications such as web different context objects and ways of configuration are available.



ApplicationContext - V

- They are both standalone XML application contexts that take bean definitions from classpath or file system, respectively.
  - **`ClassPathXMLApplicationContext`** takes the context definition files in classpath using **`packageName/resource.xml`** style.
  - **`SystemFilePathXMLApplicationContext`** takes the context definition files in file system using absolute path with **`file:`** prefix or relative path when used without any prefix.

```
ApplicationContext context = new ClassPathXmlApplicationContext(“org/javaturk/spring/beans.xml”);

ApplicationContext context = new FileSystemXmlApplicationContext(“file:/Users/akin/beans.xml”);
```

ApplicationContext - VI

- Methods to get bean instances from **`ApplicationContext`** are the ones inherited from **`BeanFactory`**:

```
Object getBean(String name)
T getBean(Class<T> requiredType)
T getBean(String name, Class<T> requiredType)
```

- To get all beans instances specific to a type when there is a possibility that many beans may have been defined use **`ListebleBeanFactory`** instead.

```
<T> Map<String,T>getBeansOfType(Class<T> type)
String[] getBeanNamesForType(Class<?> type)
```

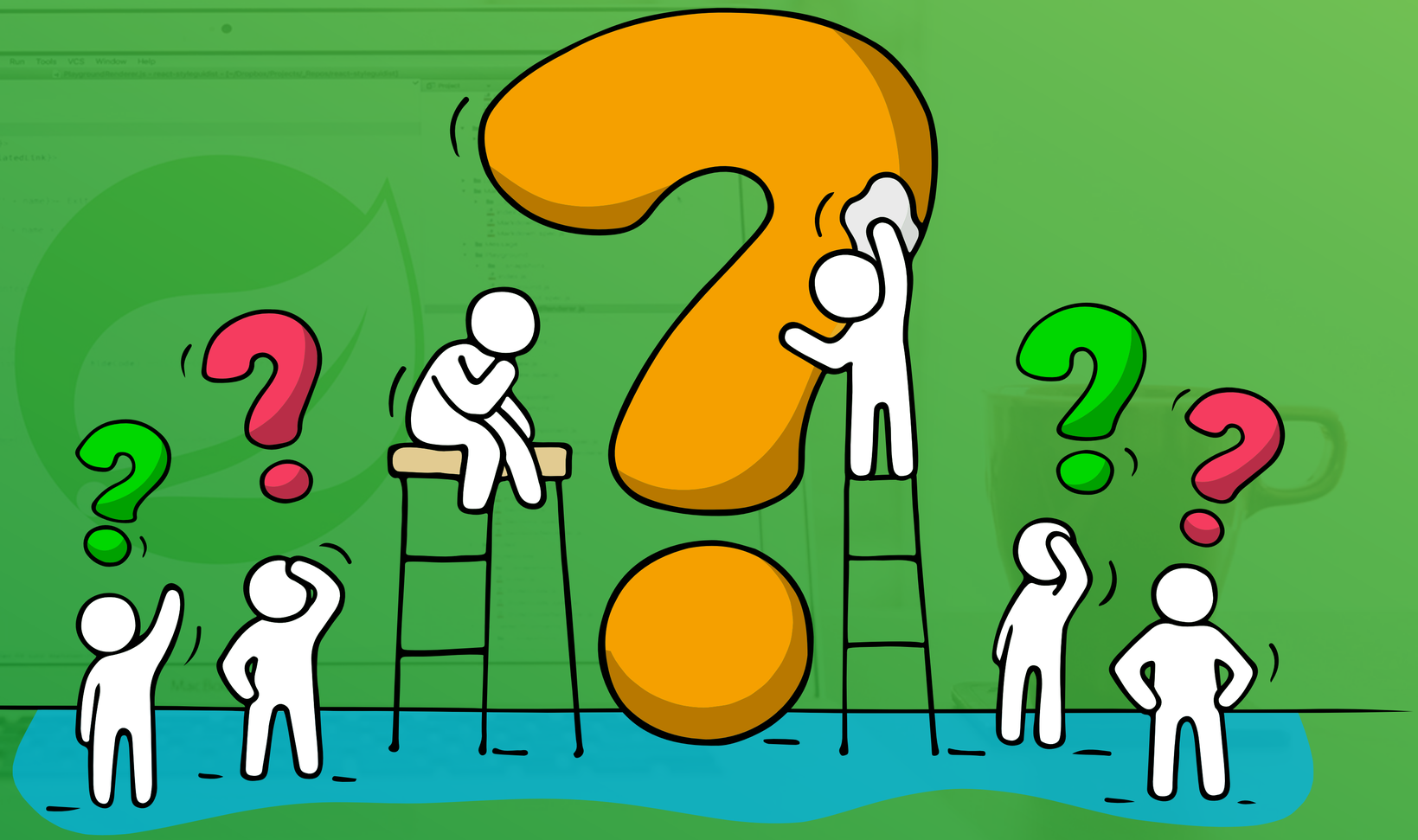
ClassPathXmlApplicationContextExample

- `org.javaturk.spring.di.ch03.ClassPathXmlApplicationContextExample`

FileSystemXmlApplicationContextExample

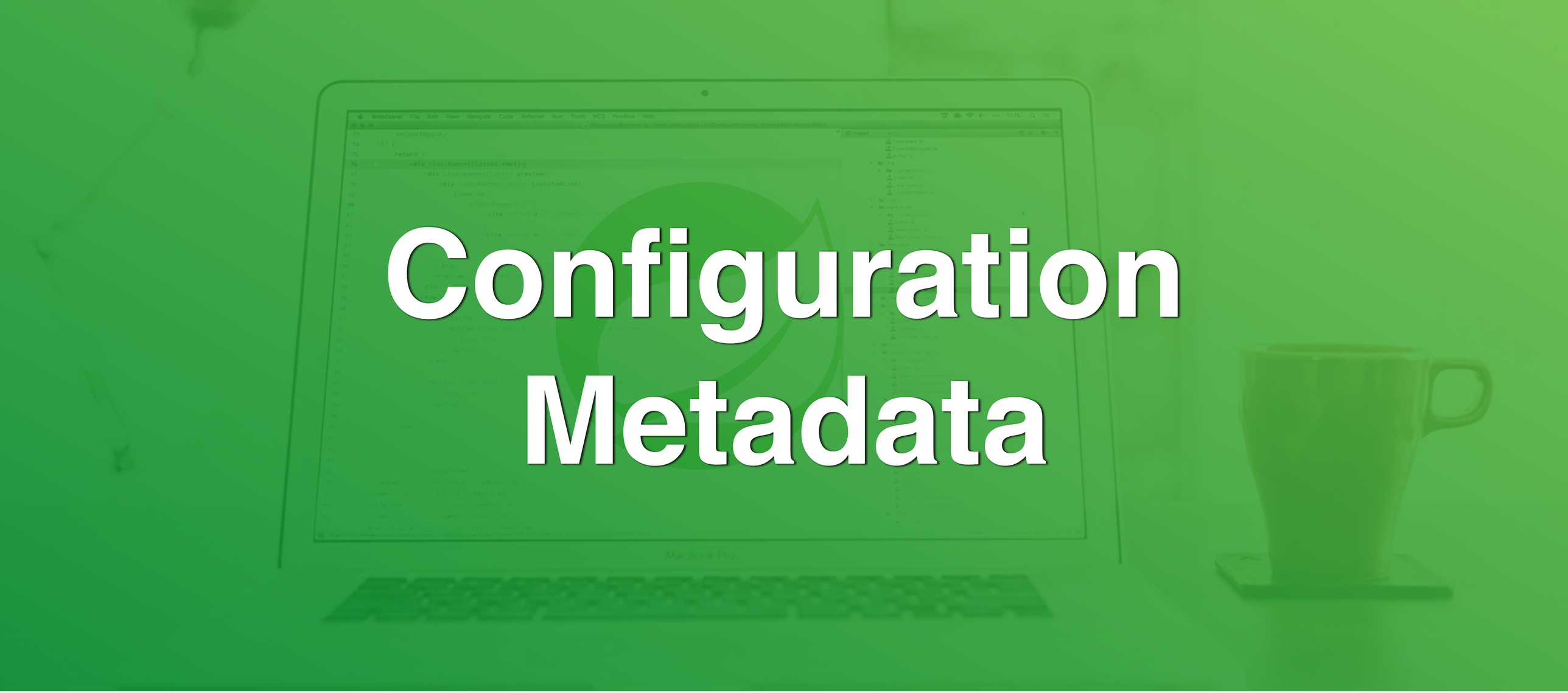
- `org.javaturk.spring.di.ch03.FileSystemXmlApplicationContextExample`

Time for Questions!



selsoft
build better, deliver faster

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

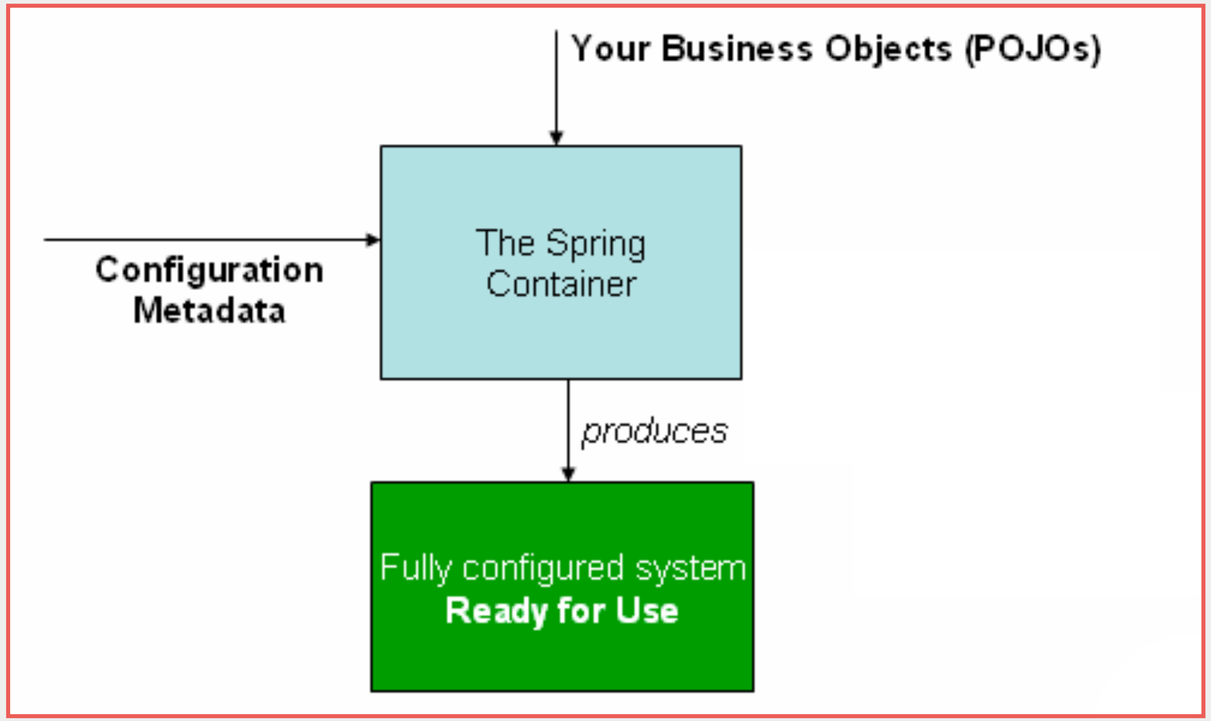


Configuration Metadata

selsoft
build better, deliver faster

Configuration Metadata - I

- The **Spring** IoC container gets its instructions to instantiate, configure, and assemble beans by reading **configuration metadata**.
- **Spring** configuration consists of at least one and typically more than one bean definition that the container manages.



Configuration Metadata - II

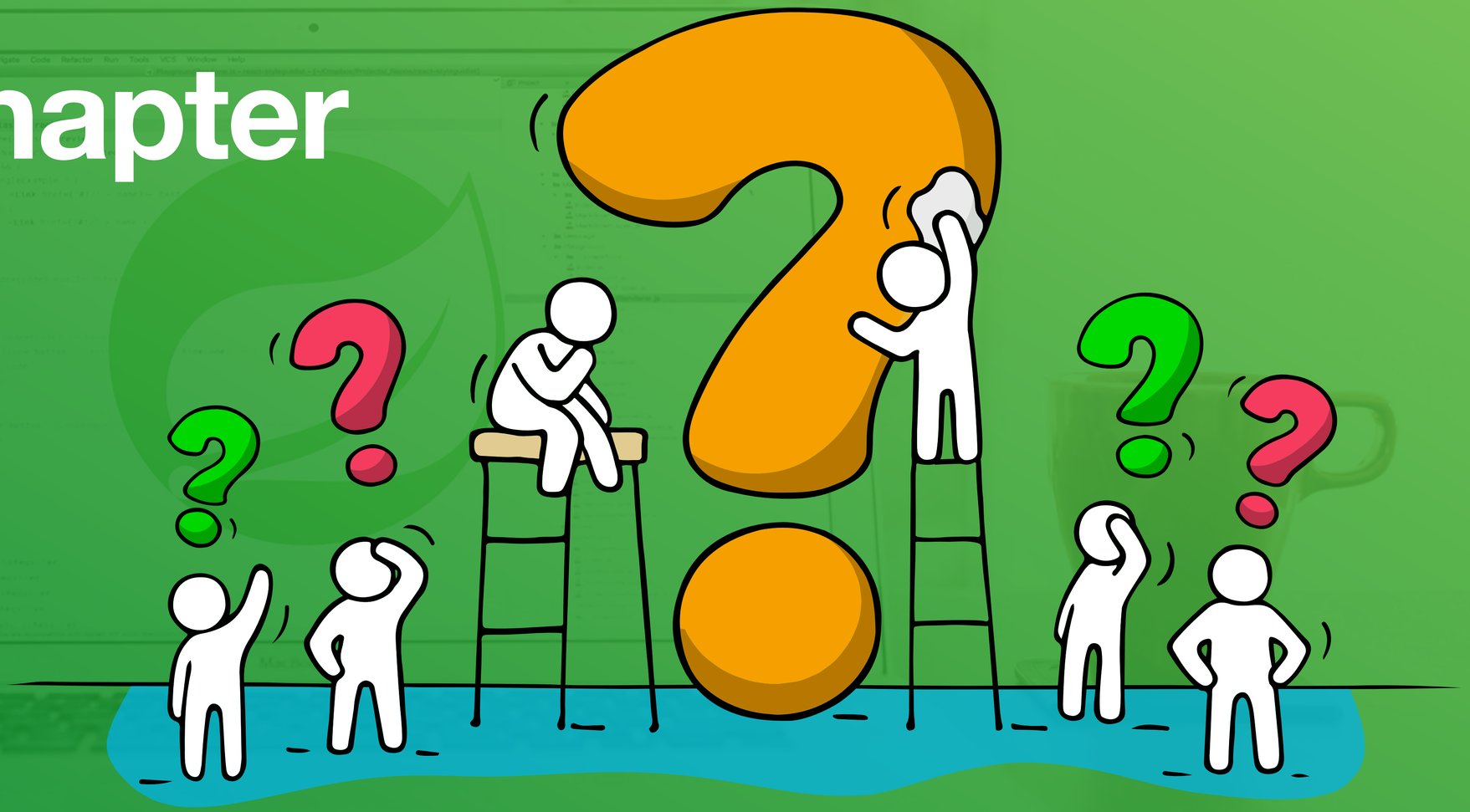
- The configuration metadata is represented in XML or properties files, Java annotations, or Java code.
  - Using XML and seldomly properties files for configuration metadata is the classical way of defining the application context.
  - Configuration using annotations became available with **Spring** 2.5.
  - And Java-based configuration is the programmatic way to configure **Spring** container which can be done by some specific annotations such as `@Configuration`, `@Bean`, `@Import`, etc.

Configuration Metadata - III

- XML-based configuration metadata configures beans as `<bean/>` elements inside a top-level `<beans/>` element.
- `@Component` and other annotations are used to designate beans.
- Java configuration typically uses `@Bean`-annotated methods within a `@Configuration` class.

End of Chapter

Time for Questions!



selsoft
build better, deliver faster

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