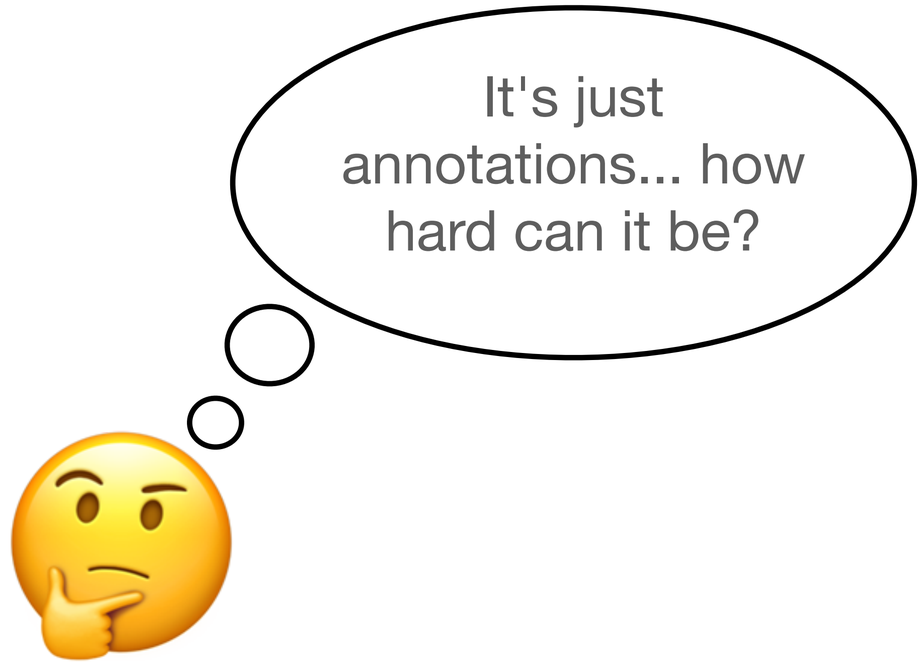
Spring Internals

Spring Boot 3 Workshop - Devnexus 2023 - Atlanta

Jonatan Ivanov & Phil Webb


It's just annotations... how hard can it be?

to the code...

@ComponentScan

@Component

Scan

@Component Scan

@Service @Repository

@Controller

@RestController

Scan

💩 `class HelloUtils`

🫘 `@Service`
`class HelloService`

🫘 `@RestController`
`class HelloController`

💩 `class MyOtherUtils`

Scan

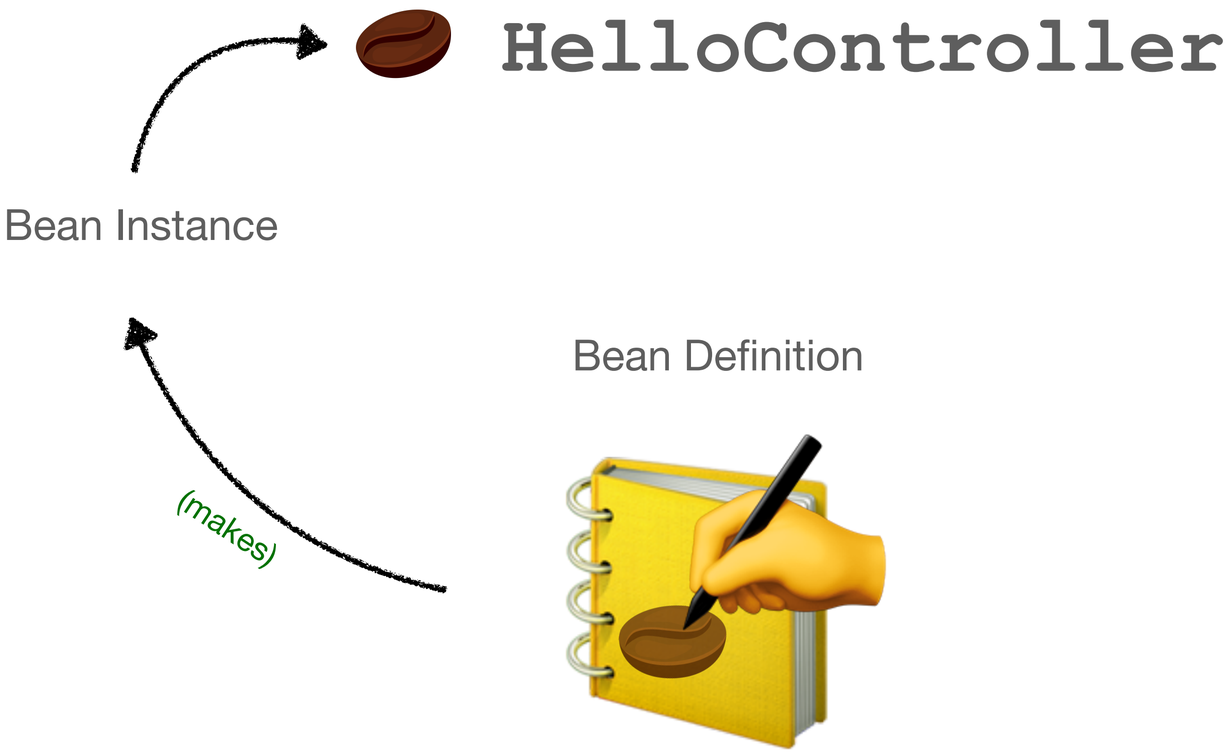
💩 `class HelloUtils`

🫘 `@Service`
`class HelloService`

🫘 `@RestController`
`class HelloController`

💩 `class MyOtherUtils`

```
@RestController
class HelloController
```

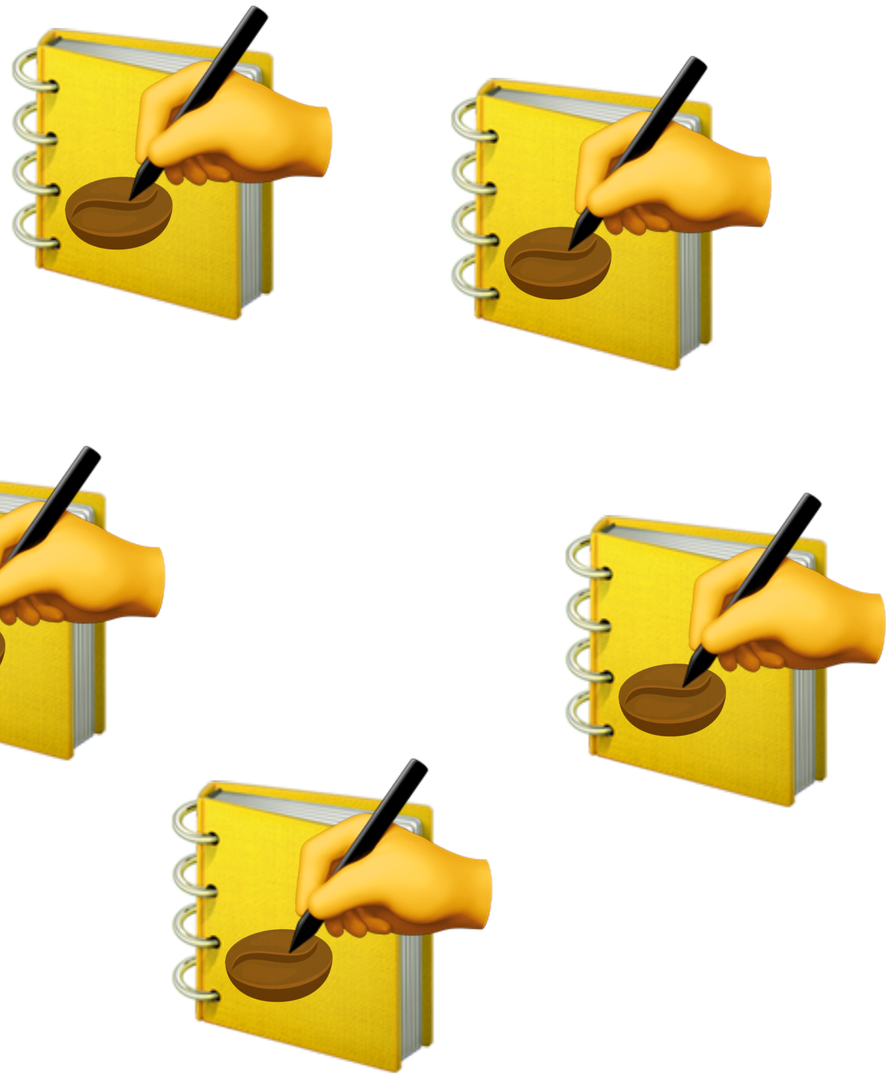

HelloController
Bean Instance
Bean Definition
(makes)

BeanDefinition

Bean Definition

- Lazy
- Primary
- Scope
- Type
- Factory Method Name
- Init Method Name
- Destroy Method Name

to the code...



BeanFactory

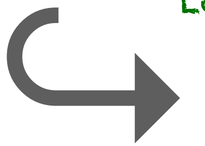
Let me get beans

BeanDefinitionRegistry

Let me register bean definitions

BeanFactory

Let me get beans



BeanDefinitionRegistry

Let me register bean definitions


BeanFactory
Let me get beans
BeanDefinitionRegistry
Let me register bean definitions
DefaultListableBeanFactory

DefaultListableBeanFactory

- The 💚 of Spring
- Manages Bean Definitions
- Manages Bean Instances
- Part of spring-beans.jar

to the code...

spring-beans.jar

BeanFactory

spring-context.jar

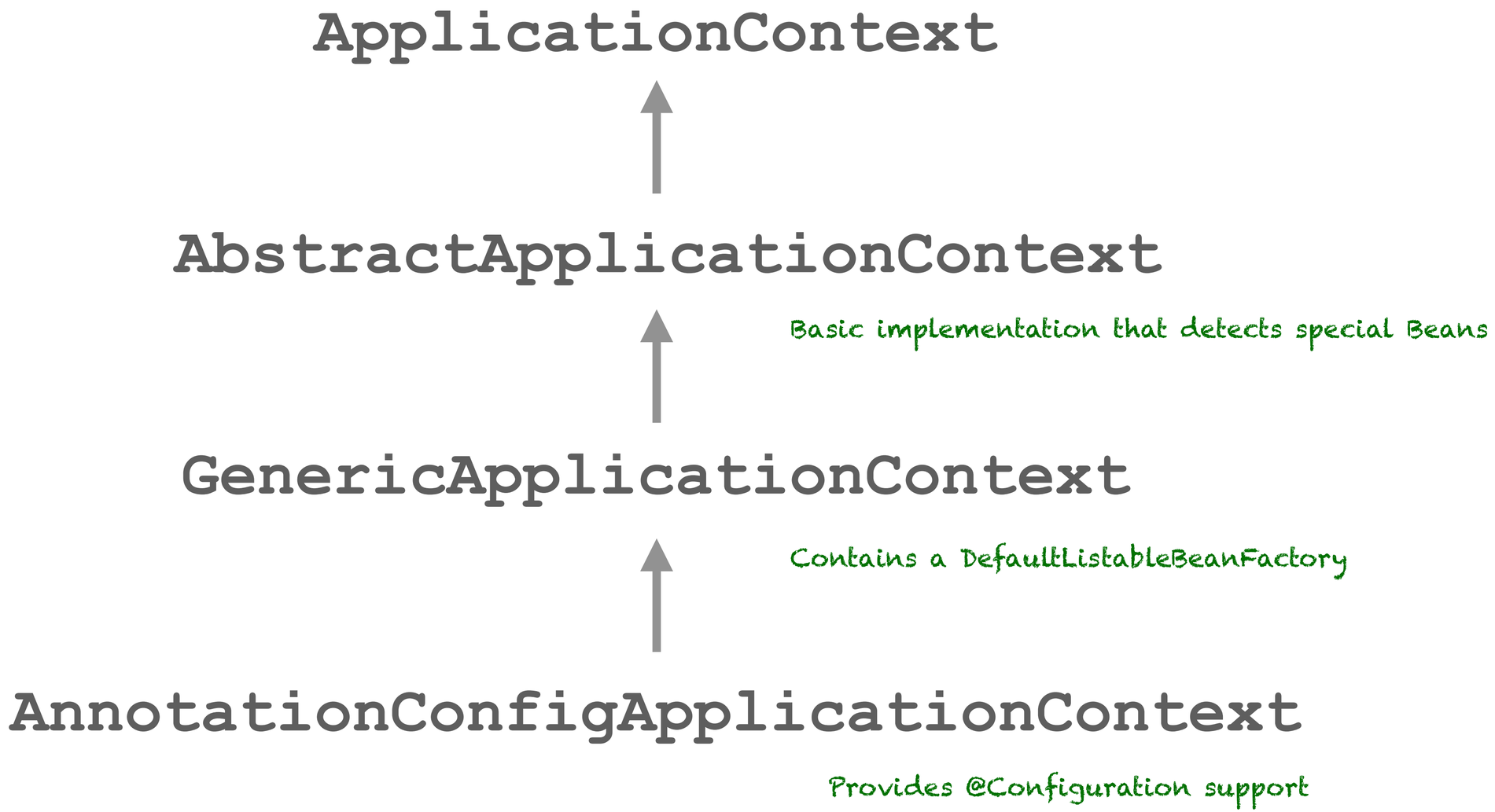
ApplicationContext

ApplicationEvent

Environment

@Configuration

MessageSource


ApplicationContext
AbstractApplicationContext
Basic implementation that detects special Beans
GenericApplicationContext
Contains a DefaultListableBeanFactory
AnnotationConfigApplicationContext
Provides @Configuration support

to the code...

AnnotationConfigApplicationContext

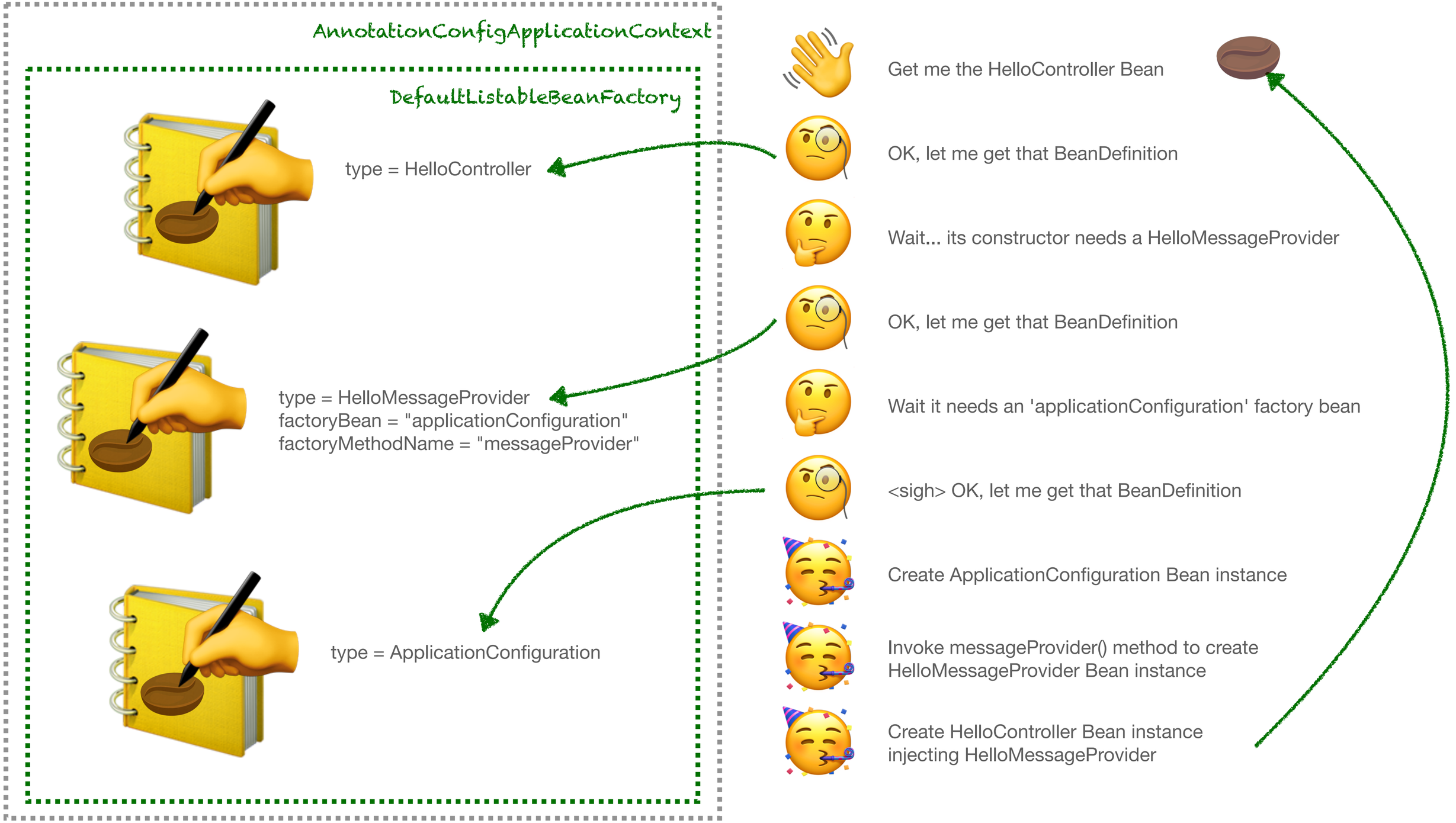
This is also a @Component

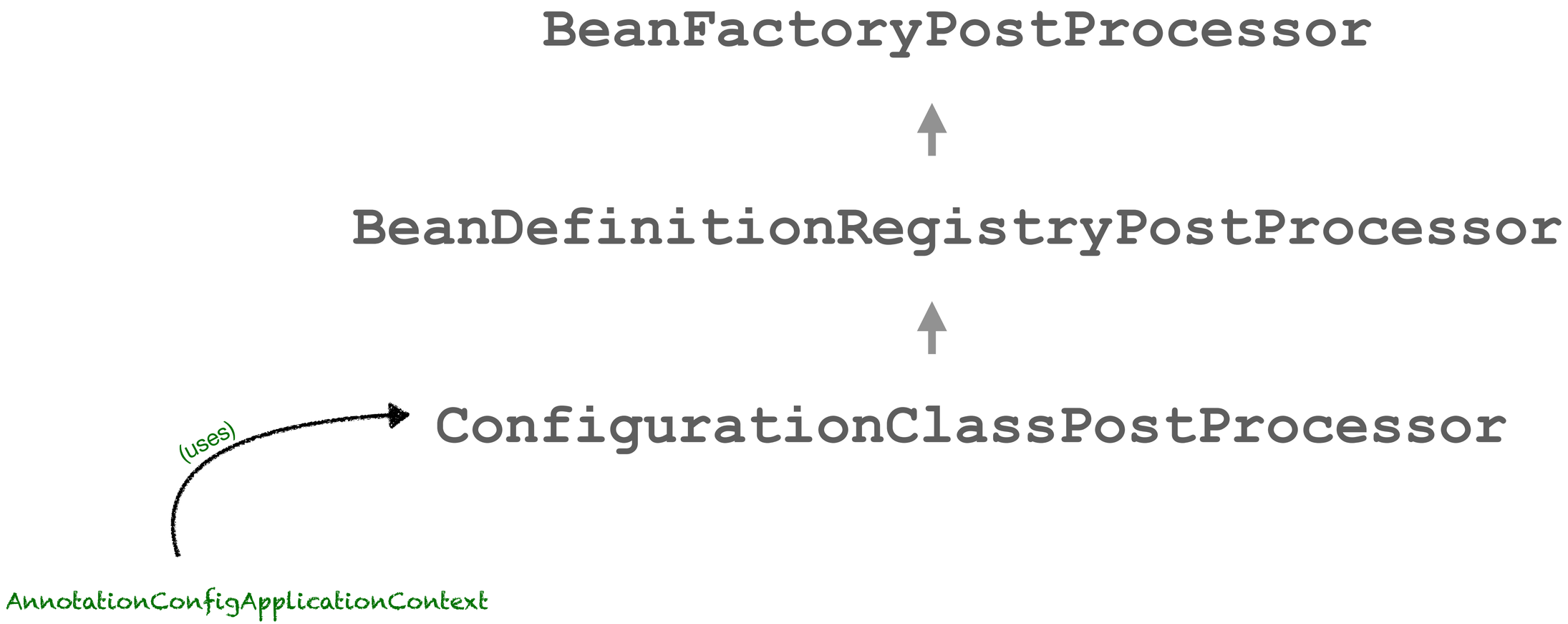
```
@Configuration
class ApplicationConfiguration {

    @Bean
    public HelloMessageProvider messageProvider() {
        return new HelloMessageProvider();
    }

}
```

Turns the method signature into a BeanDefinition

Invoked when the bean instance is needed

type = HelloMessageProvider
factoryBean = "applicationConfiguration"
factoryMethodName = "messageProvider"


AnnotationConfigApplicationContext
DefaultListableBeanFactory
type = HelloController
type = HelloMessageProvider
factoryBean = "applicationConfiguration"
factoryMethodName = "messageProvider"
type = ApplicationConfiguration
Get me the HelloController Bean
OK, let me get that BeanDefinition
Wait... its constructor needs a HelloMessageProvider
OK, let me get that BeanDefinition
Wait it needs an 'applicationConfiguration' factory bean
<sigh> OK, let me get that BeanDefinition
Create ApplicationConfiguration Bean instance
Invoke messageProvider() method to create HelloMessageProvider Bean instance
Create HelloController Bean instance injecting HelloMessageProvider


BeanFactoryPostProcessor
BeanDefinitionRegistryPostProcessor
ConfigurationClassPostProcessor
(uses)
AnnotationConfigApplicationContext

to the code...

GraalVM

`main`

`SpringApplication.run`

`@ComponentScan @Configuration @Bean`

Add reflection hints

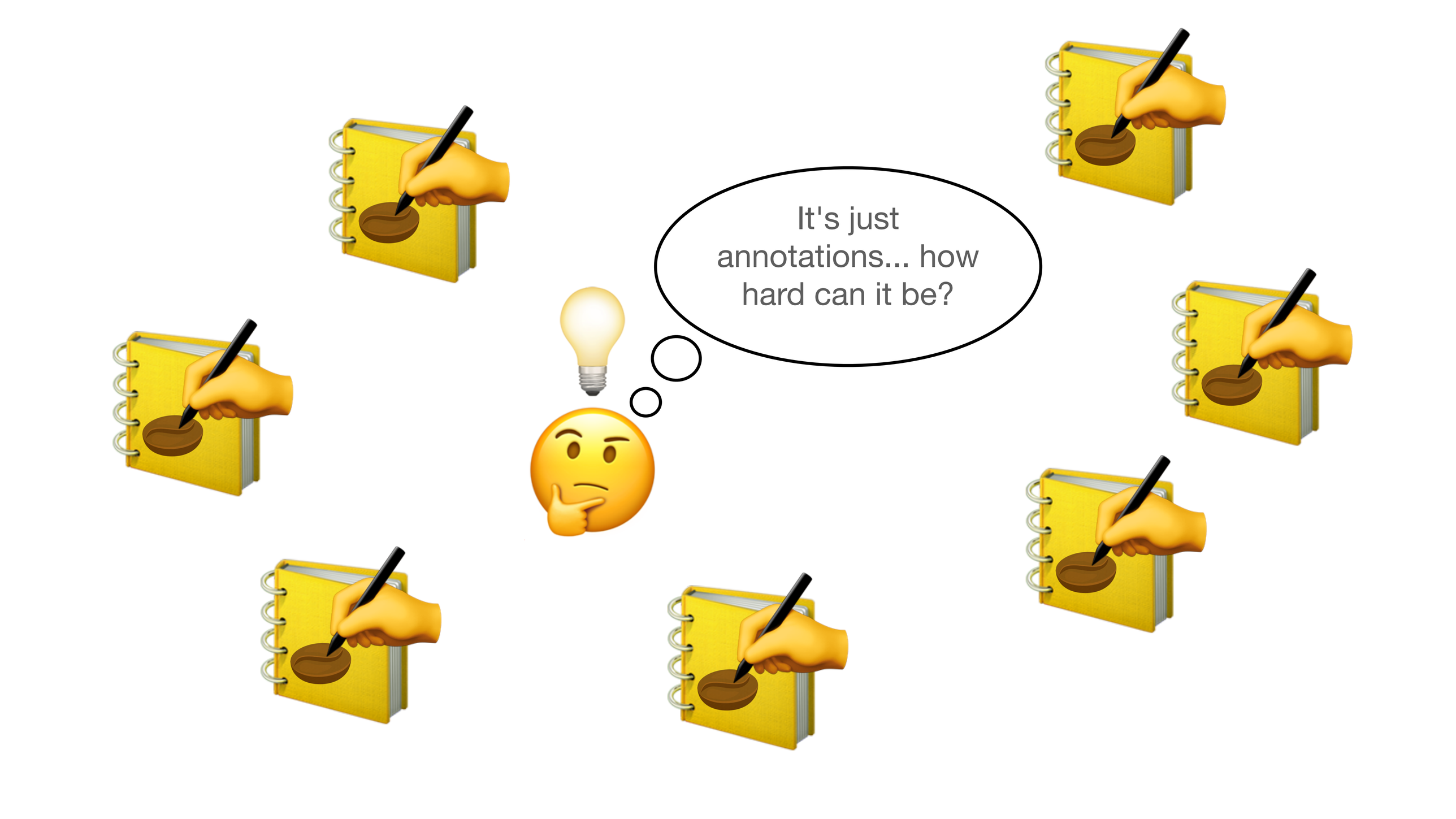
Write better code

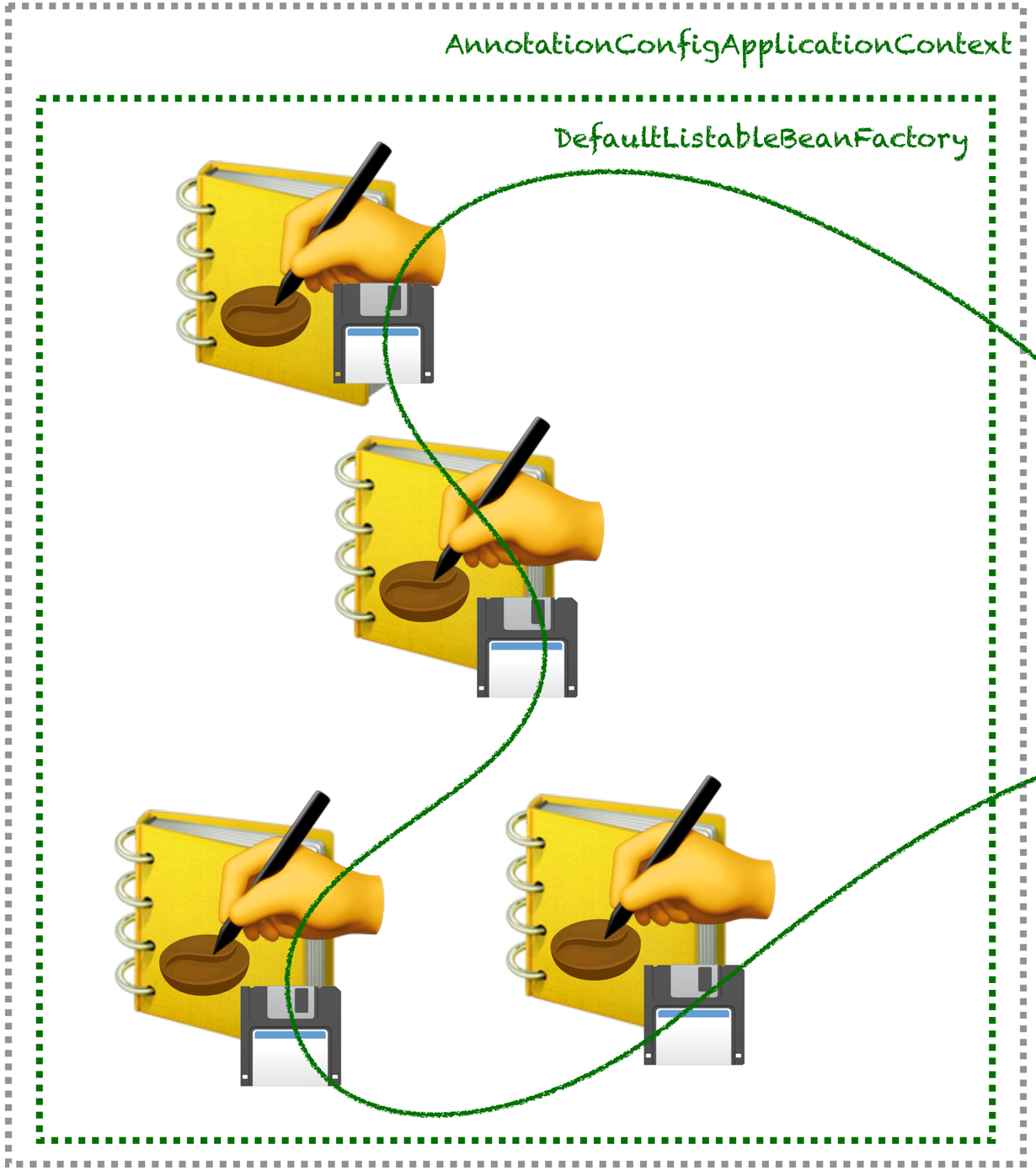
Make this...

```
@Configuration
class ApplicationConfiguration {

        @Bean
        public HelloMessageProvider messageProvider() {
                return new HelloMessageProvider();
        }

}
```

Into something more like this...

```
BeanDefinition bd = new RootBeanDefinition(HelloMessageProvider.class,
    HelloMessageProvider::new);
registry.registerBeanDefinition("helloMessageProvider", bd);
```


It's just annotations... how hard can it be?



Run the app so we have bean definitions

Stop before we create the bean instances

Create code that can recreate the bean definitions *

Let GraalVM do its thing

* There's a little more to it than that

to the code...

Checkpoint

Everyone understands more about Spring internals