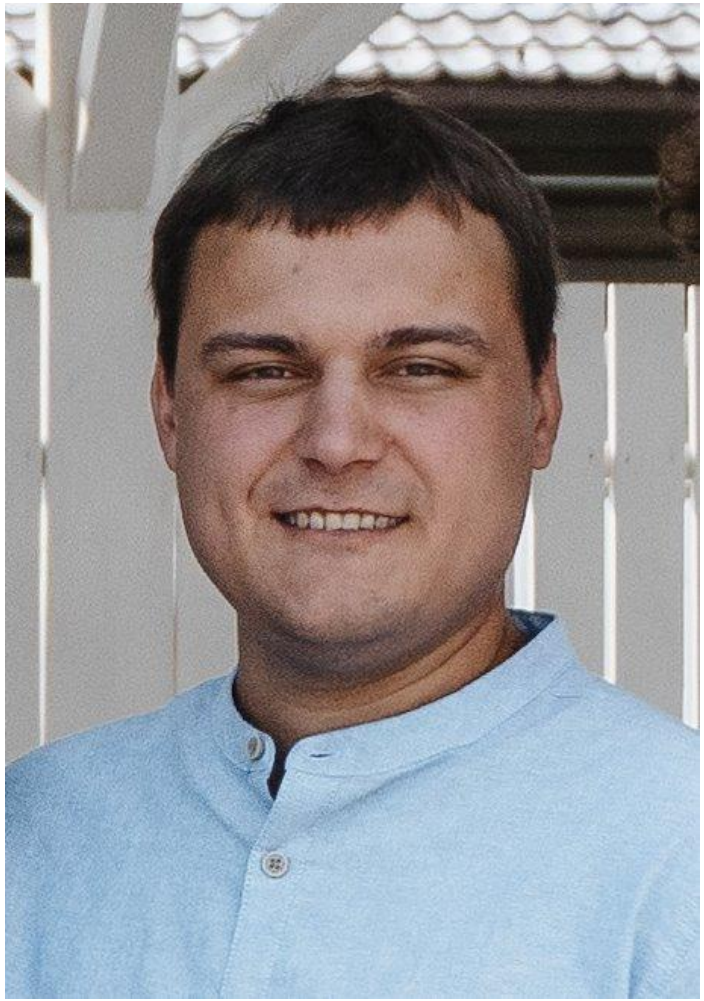
Beyond @SpringBootApplication: Создание собственных экосистем поверх Spring Boot

Кириллов Никита

Кириллов Никита

- Software Engineer
- Full-time Open Source Developer
- Axelix Project Core Maintainer



Контакты:

Email: kirilloffnikita1@gmail.com
Telegram: @kirilloffnikita
GitHub: NikitaKirilloff

О Чём Будет Доклад?

О Чём Будет Доклад?

- **Рассмотрим типичные челледжи при написнии своих решений поверх Spring Boot**

О Чём Будет Доклад?

- **Рассмотрим типичные челледжи при написнии своих решений поверх Spring Boot**

- **Пути решения этих проблем. Как это делает Spring Boot и как это делает Axelix**

О Чём Будет Доклад?

- **Рассмотрим типичные челледжи при написнии своих решений поверх Spring Boot**
- **Пути решения этих проблем. Как это делает Spring Boot и как это делает Axelix**
- **Как это применить Вам в Ваших проектах**

Axelix

Часть №0
Мотивация
Проблемы Создания Стартеров

Мотивация

В чём Выгода для слушателя?

Типичная Экосистема

Java 8
Spring Boot 2.1

Java 11
Spring Boot 2.3

Типичная Экосистема

Java 8
Spring Boot 2.1

Java 11
Spring Boot 2.3

Java 17
Spring Boot 3.1

Java 17
Spring Boot 3.5

Java 17
Spring Boot 3.3

Типичная Экосистема

Java 8
Spring Boot 2.1

Java 11
Spring Boot 2.3

Java 17
Spring Boot 3.1

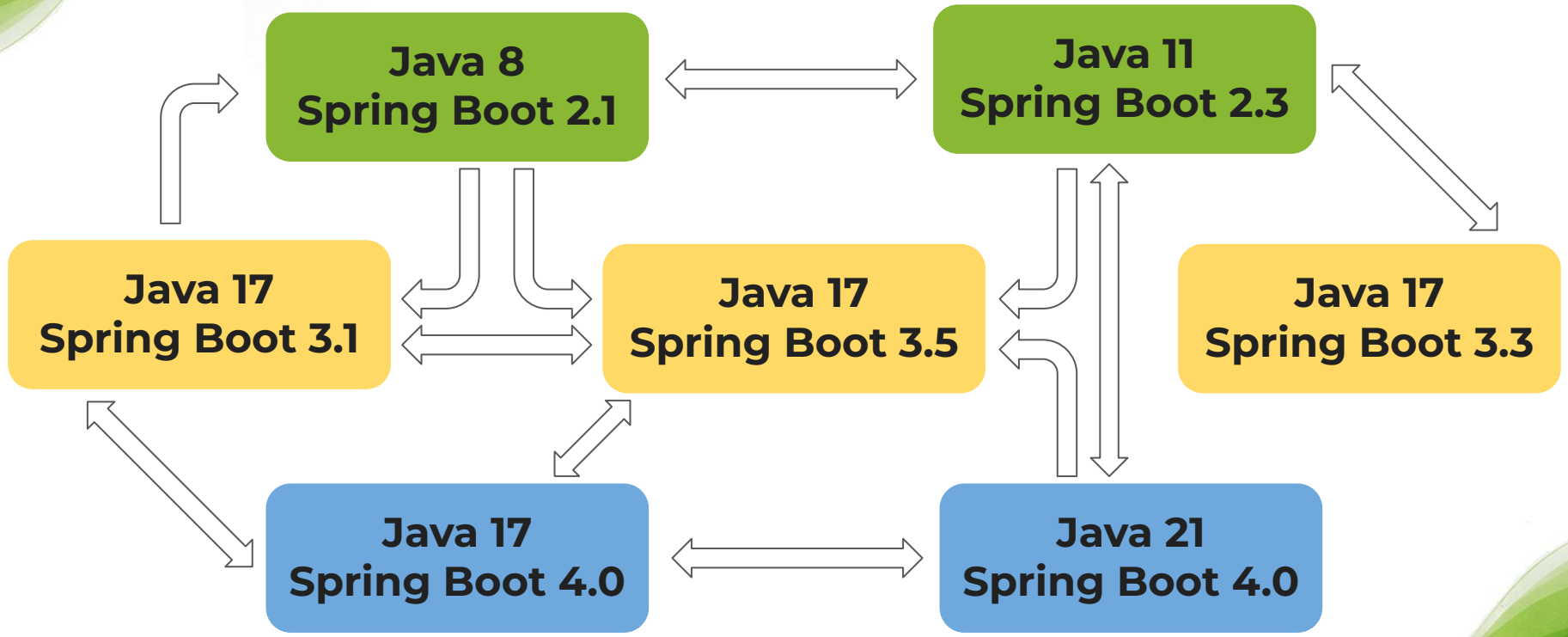
Java 17
Spring Boot 3.5

Java 17
Spring Boot 3.3

Java 17
Spring Boot 4.0

Java 21
Spring Boot 4.0

Типичная Экосистема



Что Мы Выносим в Стартеры?

Что Мы Выносим в Стартеры?

- **Security Фильтры (Jakarta’s FilterChain)**

Что Мы Выносим в Стартеры?

- **Security Фильтры (Jakarta’s FilterChain)**
- **Общий инструментарий для работы с БД (Аудит сущностей и т.д.)**

Что Мы Выносим в Стартеры?

- **Security Фильтры (Jakarta’s FilterChain)**
- **Общий инструментарий для работы с БД (Аудит сущностей и т.д.)**
- **Кастомный трейсинг и телеметрия**

Что Мы Выносим в Стартеры?

- **Security Фильтры (Jakarta's FilterChain)**
- **Общий инструментарий для работы с БД (Аудит сущностей и т.д.)**
- **Кастомный трейсинг и телеметрия**
- **Кастомные `@AutoConfiguration` бины**

Что Нам на Самом Деле Нужно?

Что Нам на Самом Деле Нужно?

- **Нам нужно какое-то единое платформенное решение**

Что Нам на Самом Деле Нужно?

- **Нам нужно какое-то единое платформенное решение**
- **Это решение должно работать на разных версиях Spring Boot**

Что Нам на Самом Деле Нужно?

- **Нам нужно какое-то единое платформенное решение**
- **Это решение должно работать на разных версиях Spring Boot**
- **Это решение должно поддерживать разные версии Java**

Представим Себе Кейс

```
@Service
public class PaymentService {


    public void processPayment(
                        Transaction transaction,
                        Customer customer,
                        String cardNumber
    ) {
        // ...
    }
}
```

Представим Себе Кейс

```
@Service
public class PaymentService {

    @NewSpan(name = "payment.processing")
    public void processPayment(
                        Transaction transaction,
                        Customer customer,
                        String cardNumber
    ) {
        // ...
    }
}
```

Представим Себе Кейс

```
@Service
public class PaymentService {

    @NewSpan(name = "payment.processing")
    public void processPayment(
                        Transaction transaction,
                        Customer customer,
                        String cardNumber

    ) {
        // ...
    }
}
```

Теги Span-а

Представим Себе Кейс

```
@Service
public class PaymentService {

    @NewSpan(name = “payment.processing”)
    public void processPayment(
                @SpanTag Transaction transaction,
                @SpanTag Customer customer,
                @SpanTag String cardNumber
    ) {
        // ...
    }
}
```

Представим Себе Кейс

```
@Service
public class PaymentService {

    @NewSpan(name = "payment.processing")
    public void processPayment(
                @SpanTag Transaction transaction,
                @SpanTag Customer customer,
                @SpanTag String cardNumber
    ) {
        // ...
    }
}
```

Представим Себе Кейс

```
@Service
public class PaymentService {

    @NewSpan(name = "payment.processing")
    public void processPayment(
            @FinancialTag Transaction transaction,
            @PiiTag       Customer customer,
            @SecretTag    String cardNumber
    ) {
        // ...
    }
}
```

Кейс. Возможные Типы Тегов

```
public enum DataSensitivity {

 PUBLIC(PublicSanitizer.class),              // ничего не скрываем
 PII(PiiSanitizer.class),                    // ФИО — маскируем фамилию и отчество
 FINANCIAL(FinancialSanitizer.class),        // только ID транзакции, остальное скрыть
 CREDENTIALS(CredentialsSanitizer.class), // пароли, токены, ключи, полная замена на ***
 SECRET(SecretSanitizer.class)               // ничего в спан не пишем

 ...
}
```

Кейс. Возможные Типы Тегов

```
public enum DataSensitivity {

  PUBLIC(PublicSanitizer.class),              // ничего не скрываем
  PII(PiiSanitizer.class),                    // ФИО — маскируем фамилию и отчество
  FINANCIAL(FinancialSanitizer.class),        // только ID транзакции, остальное скрыть
  CREDENTIALS(CredentialsSanitizer.class),    // пароли, токены, ключи, полная замена на ***
  SECRET(SecretSanitizer.class)               // ничего в спан не пишем

  ...
}
```

```
Span: payment.processing
  param.0 = { transactionId: "tx-123" }     // @Financial только ID
  param.1 = "Иван ***. ***."                // @PII маскируем фамилию и отчество
  param.2 = "[REDACTED]"                    // @Secret
```

Объявляем @Bean для Каждого Sanitzer

```
@AutoConfiguration
class SanitizerBeans {

    @Bean
    PublicSanitizer publicSanitizer() {
        return PublicSanitizer().create();
    }

    @Bean
    PiiSanitizer piiSanitizer() {
        return PiiSanitizer().create();
    }

    @Bean
    FinancialSanitizer financialSanitizer() {
        return FinancialSanitizer().create();
    }

    @Bean
    CredentialsSanitizer credentialsSanitizer() {
        return CredentialsSanitizer().create();
    }

    // ...
}
```

Объявляем @Bean для Каждого Sanitzer

```
@AutoConfiguration
class SanitizerBeans {

    @Bean
    PublicSanitizer publicSanitizer() {
        return PublicSanitizer().create();
    }

    @Bean
    PiiSanitizer piiSanitizer() {
        return PiiSanitizer().create();
    }

    @Bean
    FinancialSanitizer financialSanitizer() {
        return FinancialSanitizer().create();
    }

    @Bean
    CredentialsSanitizer credentialsSanitizer() {
        return CredentialsSanitizer().create();
    }

    // ...
}
```

В чём же Проблемы?

В чём же Проблемы?

<DRY>

Axelix

Часть №1
Создание Бинов
Программные Подходы

Способ Первый SingletonBeanRegistry

```
public interface SingletonBeanRegistry {
   ...




   ...
}
```

Способ Первый SingletonBeanRegistry

```
public interface SingletonBeanRegistry {
   ...

   /**
    * @param beanName the name of the bean
    * @param singletonObject the existing singleton object
    */
   void registerSingleton(String beanName, Object singletonObject);

   ...
}
```

Способ Первый SingletonBeanRegistry

```
public enum DataSensitivity {

    PUBLIC(PublicSanitizer.class),              // ничего не скрываем
    PII(PiiSanitizer.class),                    // ФИО — маскируем фамилию и отчество
    FINANCIAL(FinancialSanitizer.class),        // только ID транзакции, остальное скрыть
    CREDENTIALS(CredentialsSanitizer.class),    // пароли, токены — полная замена на ***
    SECRET(SecretSanitizer.class)               // ничего в спан не пишем

    ...
}
```

Способ Первый SingletonBeanRegistry

```
public class SanitizerSingletonRegistrar implements BeanFactoryPostProcessor {

    @Override
    public void postProcessBeanFactory(ConfigurableListableBeanFactory beanFactory) {




    }
}
```

Способ Первый SingletonBeanRegistry

```
public class SanitizerSingletonRegistrar implements BeanFactoryPostProcessor {

    @Override
    public void postProcessBeanFactory(ConfigurableListableBeanFactory beanFactory) {
        for (DataSensitivity sanitizer : DataSensitivity.values()) {




        }
    }
}
```

Способ Первый SingletonBeanRegistry

```
public class SanitizerSingletonRegistrar implements BeanFactoryPostProcessor {

    @Override
    public void postProcessBeanFactory(ConfigurableListableBeanFactory beanFactory) {
        for (DataSensitivity sanitizer : DataSensitivity.values()) {
            beanFactory.registerSingleton(
                    sanitizer.beanName(),
                    sanitizer.create()
            );
        }
    }
}
```

Способ Первый SingletonBeanRegistry

Плюсы:

Способ Первый SingletonBeanRegistry

Плюсы:

- **Максимально простой и быстрый способ регистрации бинов**

Способ Первый SingletonBeanRegistry

Плюсы:

- **Максимально простой и быстрый способ регистрации бинов**
- **Всё.....**

Способ Первый SingletonBeanRegistry

Плюсы:

- **Максимально простой и быстрый способ регистрации бинов**
- **Всё.....**

Минусы:

- **Scope всегда `Singleton`**

Способ Первый SingletonBeanRegistry

Плюсы:

- **Максимально простой и быстрый способ регистрации бинов**
- **Всё.....**

Минусы:

- **Scope всегда `Singleton`**
- **Нет управления lifecycle Spring Bean`s**

Способ Первый SingletonBeanRegistry

Плюсы:

- **Максимально простой и быстрый способ регистрации бинов**
- **Всё.....**

Минусы:

- **Scope всегда `Singleton`**
- **Нет управления lifecycle Spring Bean`s**
- **Ограниченная конфигурация для Spring Bean`s**

Способ Второй

Способ Второй ImportBeanDefinitionRegistrar

```
public interface ImportBeanDefinitionRegistrar {
   ...
   ...
}
```

Способ Второй ImportBeanDefinitionRegistrar

```
public interface ImportBeanDefinitionRegistrar {
   ...


   /**
    * @param importingClassMetadata annotation metadata of the importing class
    * @param registry current bean definition registry
    */
   default void registerBeanDefinitions(
      AnnotationMetadata importingClassMetadata,
      BeanDefinitionRegistry registry) {}

   ...
}
```

```
public class SanitizerBeanDefinitionRegistrar
        implements ImportBeanDefinitionRegistrar {

    @Override
    public void registerBeanDefinitions(AnnotationMetadata importingClassMetadata,
                                        BeanDefinitionRegistry registry) {









    }
}
```

```
public class SanitizerBeanDefinitionRegistrar
        implements ImportBeanDefinitionRegistrar {

    @Override
    public void registerBeanDefinitions(AnnotationMetadata importingClassMetadata,
                                        BeanDefinitionRegistry registry) {
        for (DataSensitivity sanitizer : DataSensitivity.values()) {









        }
    }
}
```

```
public class SanitizerBeanDefinitionRegistrar
        implements ImportBeanDefinitionRegistrar {

    @Override
    public void registerBeanDefinitions(AnnotationMetadata importingClassMetadata,
                                        BeanDefinitionRegistry registry) {
        for (DataSensitivity sanitizer : DataSensitivity.values()) {

            BeanDefinition beanDefinition = BeanDefinitionBuilder
                .rootBeanDefinition(sanitizer.sanitizerClass())
                .setScope(BeanDefinition.SCOPE_SINGLETON)
                .getBeanDefinition();

            registry.registerBeanDefinition(sanitizer.beanName(), beanDefinition);
        }
    }
}
```

```
public class SanitizerBeanDefinitionRegistrar
        implements ImportBeanDefinitionRegistrar {

    @Override
    public void registerBeanDefinitions(AnnotationMetadata importingClassMetadata,
                                        BeanDefinitionRegistry registry) {
        for (DataSensitivity sanitizer : DataSensitivity.values()) {

            BeanDefinition beanDefinition = BeanDefinitionBuilder
                .rootBeanDefinition(sanitizer.sanitizerClass())
                .setScope(BeanDefinition.SCOPE_SINGLETON)
                .setInitMethodName("init")
                .setDestroyMethodName("destroy")
                .setLazyInit(true)
                .setSynthetic(true)
                .setRole(ROLE_INFRASTRUCTURE)
                .setAutowireMode(AUTOWIRE_CONSTRUCTOR)
                .getBeanDefinition();

            registry.registerBeanDefinition(sanitizer.beanName(), beanDefinition);
        }
    }
}
```

Способ Второй ImportBeanDefinitionRegistrar

Плюсы:

Способ Второй ImportBeanDefinitionRegistrar

Плюсы:

- **Гибкость конфигурации для Spring Bean`s**

Способ Второй ImportBeanDefinitionRegistrar

Плюсы:

- **Гибкость конфигурации для Spring Bean`s**
- **Полное управление lifecycle для Spring Bean`s**

Способ Второй ImportBeanDefinitionRegistrar

Плюсы:

- **Гибкость конфигурации для Spring Bean`s**
- **Полное управление lifecycle для Spring Bean`s**
- **Стандарт для программной регистрации бинов в Spring**

Способ Второй ImportBeanDefinitionRegistrar

Плюсы:

- **Гибкость конфигурации для Spring Bean`s**
- **Полное управление lifecycle для Spring Bean`s**
- **Стандарт для программной регистрации бинов в Spring**

Минусы:

- **Низкоуровневый API, ручное построение `BeanDefinition`**

Способ Второй ImportBeanDefinitionRegistrar

Плюсы:

- **Гибкость конфигурации для Spring Bean`s**
- **Полное управление lifecycle для Spring Bean`s**
- **Стандарт для программной регистрации бинов в Spring**

Минусы:

- **Низкоуровневый API, ручное построение `BeanDefinition`**
- **Повышенная требовательность к компетенциям**

Особенности ImportBeanDefinitionRegistrar

- **Не является `BeanFactoryPostProcessor`-ом**

Особенности ImportBeanDefinitionRegistrar

- Не является `BeanFactoryPostProcessor`-ом
- Не является бином

Особенности ImportBeanDefinitionRegistrar

- Не является `BeanFactoryPostProcessor`-ом
- Не является бином
- Это “временный объект” на этапе парсинга `@Configuration` классов

“Условное” создание бина: на основе Env

```
public class SanitizerBeanDefinitionRegistrar
        implements ImportBeanDefinitionRegistrar {

    @Override
    public void registerBeanDefinitions(AnnotationMetadata importingClassMetadata,
                                        BeanDefinitionRegistry registry) {

        for (DataSensitivity sanitizer : DataSensitivity.values()) {


            BeanDefinition beanDefinition = BeanDefinitionBuilder
                .rootBeanDefinition(sanitizer.sanitizerClass())
                .getBeanDefinition();

            registry.registerBeanDefinition(sanitizer.beanName(), beanDefinition);
        }
    }


}
```

“Условное” создание бина: на основе Env

```
public class SanitizerBeanDefinitionRegistrar
        implements ImportBeanDefinitionRegistrar {

    @Autowired
    private Environment env;


    @Override
    public void registerBeanDefinitions(AnnotationMetadata importingClassMetadata,
                                        BeanDefinitionRegistry registry) {

        for (DataSensitivity sanitizer : DataSensitivity.values()) {
            // Если создание какой-либо бина "выключено" через properties
            if (sanitizerShouldBeSkipped(sanitizer)) continue;

            BeanDefinition beanDefinition = BeanDefinitionBuilder
                .rootBeanDefinition(sanitizer.sanitizerClass())
                .getBeanDefinition();

            registry.registerBeanDefinition(sanitizer.beanName(), beanDefinition);
        }
    }

    private boolean sanitizerShouldBeSkipped(DataSensitivity sanitizer) {
        List<String> sanitizersToSkip = env...
        return sanitizersToSkip.contains(sanitizer.beanName());
    }
}
```

“Условное” создание бина: на основе Env

```
public class SanitizerBeanDefinitionRegistrar
        implements ImportBeanDefinitionRegistrar {

    @Autowired
    private Environment env;



    @Override
    public void registerBeanDefinitions(AnnotationMetadata importingClassMetadata,
                                        BeanDefinitionRegistry registry) {

        for (DataSensitivity sanitizer : DataSensitivity.values()) {
            // Если создание какой-либо бина "выключено" через properties
            if (sanitizerShouldBeSkipped(sanitizer)) continue;

            BeanDefinition beanDefinition = BeanDefinitionBuilder
                .rootBeanDefinition(sanitizer.sanitizerClass())
                .getBeanDefinition();

            registry.registerBeanDefinition(sanitizer.beanName(), beanDefinition);
        }
    }

  private boolean sanitizerShouldBeSkipped(DataSensitivity sanitizer) {
      List<String> sanitizersToSkip = env.parse();
      return sanitizersToSkip.contains(sanitizer.beanName());
  }
}
```

“Условное” создание бина: на основе Env

```
public class SanitizerBeanDefinitionRegistrar
        implements ImportBeanDefinitionRegistrar {

    @Autowired
    private Environment env;



    @Override
    public void registerBeanDefinitions(AnnotationMetadata importingClassMetadata,
                                        BeanDefinitionRegistry registry) {

        for (DataSensitivity sanitizer : DataSensitivity.values()) {
            // Если создание какой-либо бина "выключено" через properties
            if (sanitizerShouldBeSkipped(sanitizer)) continue;

            BeanDefinition beanDefinition = BeanDefinitionBuilder
                .rootBeanDefinition(sanitizer.sanitizerClass())
                .getBeanDefinition();

            registry.registerBeanDefinition(sanitizer.beanName(), beanDefinition);
        }
    }

  private boolean sanitizerShouldBeSkipped(DataSensitivity sanitizer) {
      List<String> sanitizersToSkip = env...
      return sanitizersToSkip.contains(sanitizer.beanName());
  }
}
```

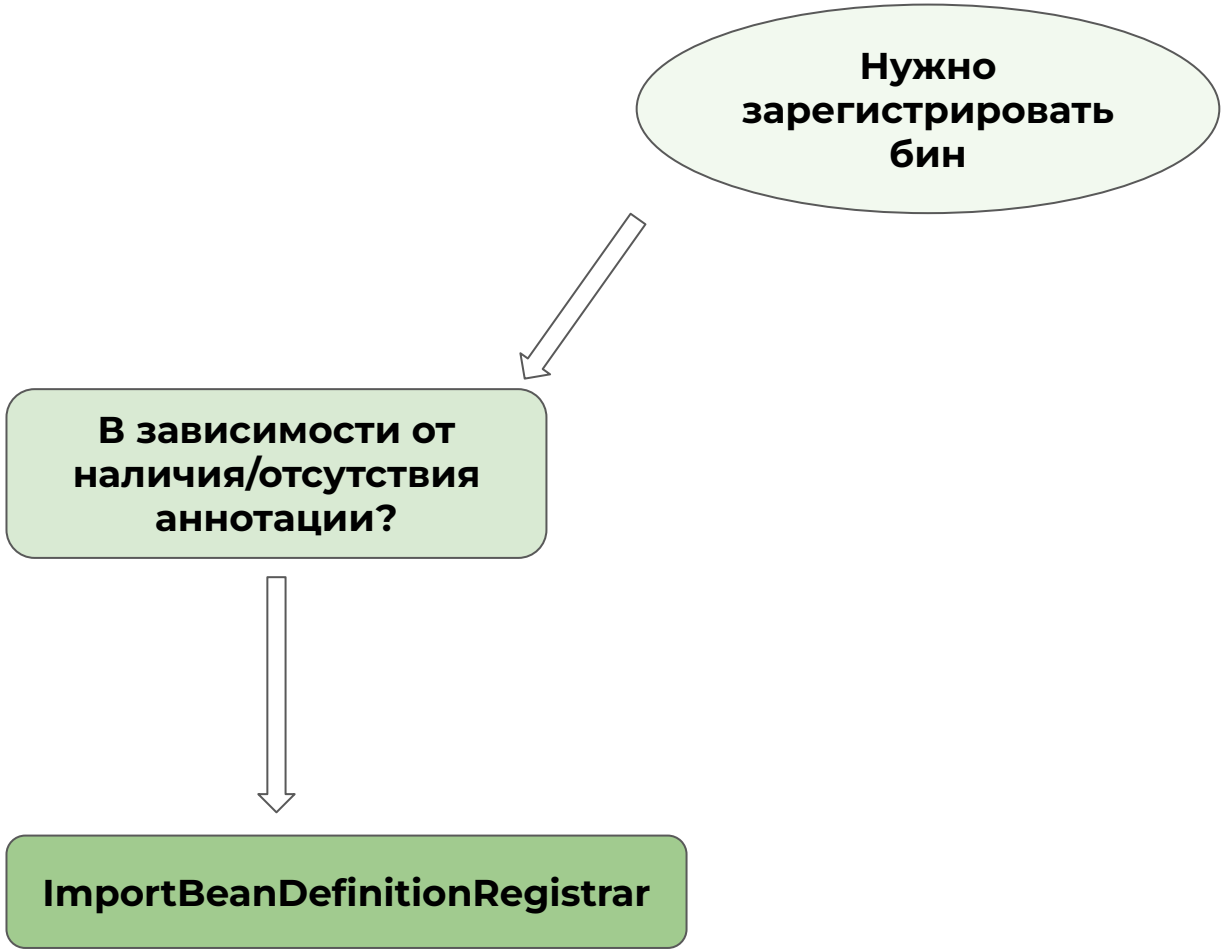
```
public class SanitizerBeanDefinitionRegistr
        implements ImportBeanDefinitionRegis

    @Autowired
    private Environment env;



    @Override
    public void registerBeanDefinitions (Anno
                                         Bean

        for (DataSensitivity sanitizer : DataSensitivity.values()) {
            // Если создание какой-либо бина "выключено" через properties
            if (sanitizerShouldBeSkipped (sanitizer)) continue;

            BeanDefinition beanDefinition = BeanDefinitionBuilder
                .rootBeanDefinition(sanitizer.sanitizerClass())
                .getBeanDefinition();

            registry.registerBeanDefinition (sanitizer.beanName(), beanDefinition);
        }
    }

  private boolean sanitizerShouldBeSkipped (DataSensitivity sanitizer) {
      List<String> sanitizersToSkip =  env...
      return sanitizersToSkip.contains(sanitizer.beanName());
  }
}
```

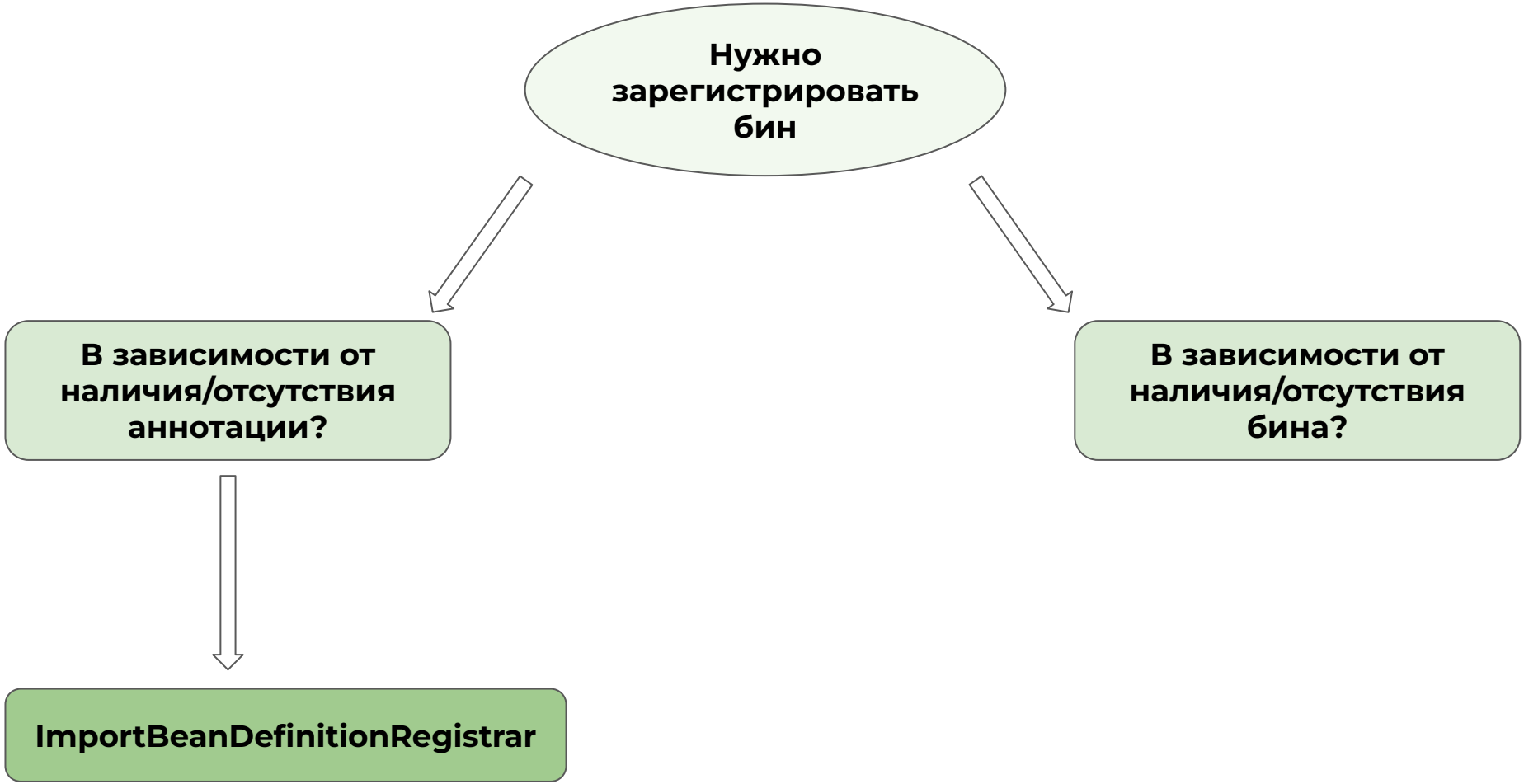
```
public class SanitizerBeanDefinitionRegistrar
        implements ImportBeanDefinitionRegistrar {

    @Autowired
    private Environment env;


    @Override
    public void registerBeanDefinitions(AnnotationMetadata importingClassMetadata,
                                        BeanDefinitionRegistry registry) {

        for (DataSensitivity sanitizer : DataSensitivity.values()) {
            // Если создание какой-либо бина "выключено" через properties
            if (sanitizerShouldBeSkipped(sanitizer)) continue;

            BeanDefinition beanDefinition = BeanDefinitionBuilder
                .rootBeanDefinition(sanitizer.sanitizerClass())
                .getBeanDefinition();

            registry.registerBeanDefinition(sanitizer.beanName(), beanDefinition);
        }
    }

  private boolean sanitizerShouldBeSkipped(DataSensitivity sanitizer) {
      List<String> sanitizersToSkip = env...
      return sanitizersToSkip.contains(sanitizer.beanName());
  }
}
```

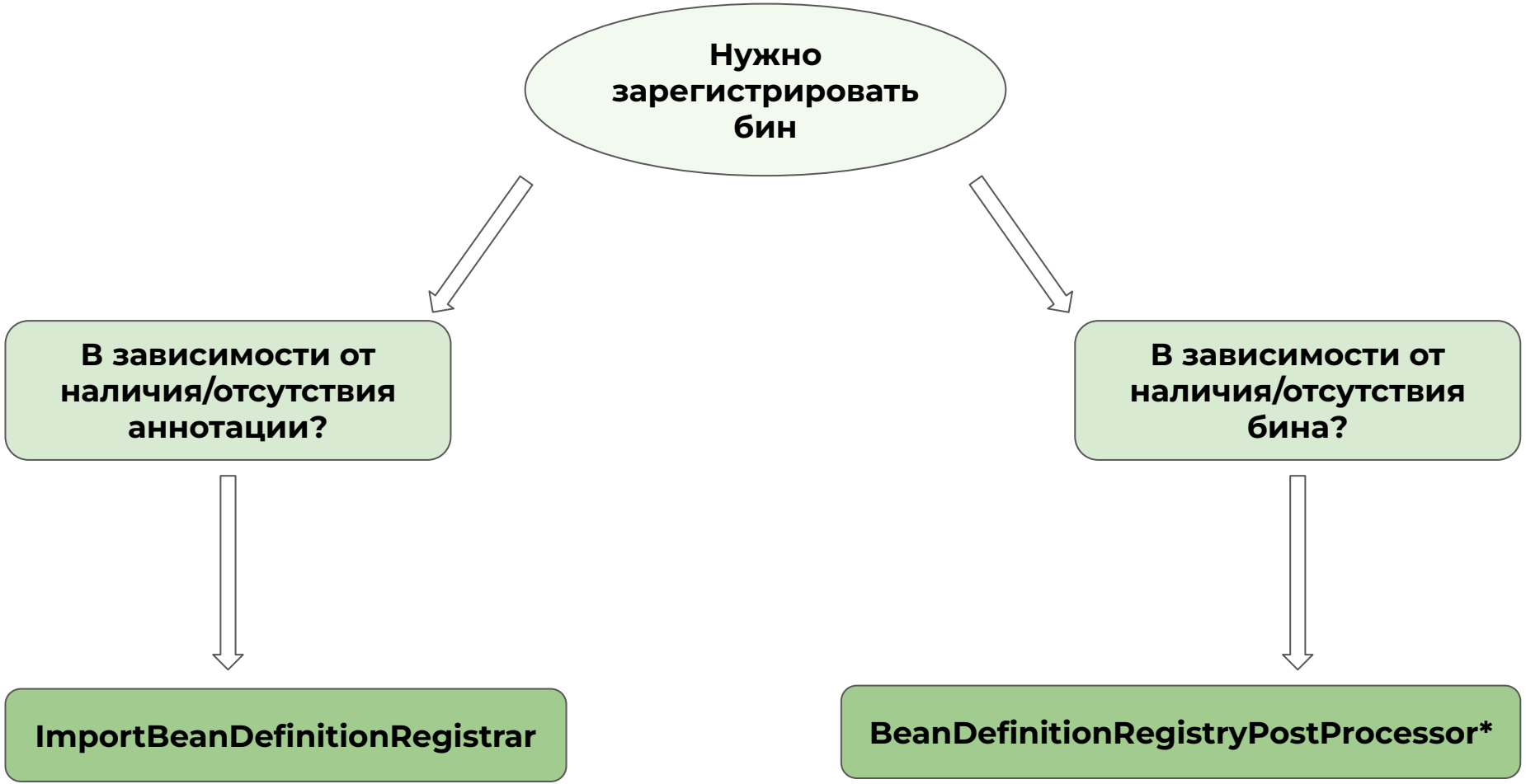
```
public class SanitizerBeanDefinitionRegistrar
        implements ImportBeanDefinitionRegistrar,  EnvironmentAware {

    private Environment env;

    @Override
    public void setEnvironment(Environment env) { this.env = env; }

    @Override
    public void registerBeanDefinitions (AnnotationMetadata importingClassMetadata ,
                                         BeanDefinitionRegistry registry ) {

        for (DataSensitivity sanitizer  : DataSensitivity.values()) {
            // Если создание какой-либо бина "выключено" через properties
            if (sanitizerShouldBeSkipped (sanitizer)) continue;

            BeanDefinition beanDefinition  = BeanDefinitionBuilder
                .rootBeanDefinition(sanitizer.sanitizerClass ())
                .getBeanDefinition ();

            registry.registerBeanDefinition (sanitizer.beanName (), beanDefinition );
        }
    }

  private boolean  sanitizerShouldBeSkipped (DataSensitivity  sanitizer) {
      List<String> sanitizersToSkip =  env...
      return sanitizersToSkip.contains(sanitizer.beanName());
  }
}
```

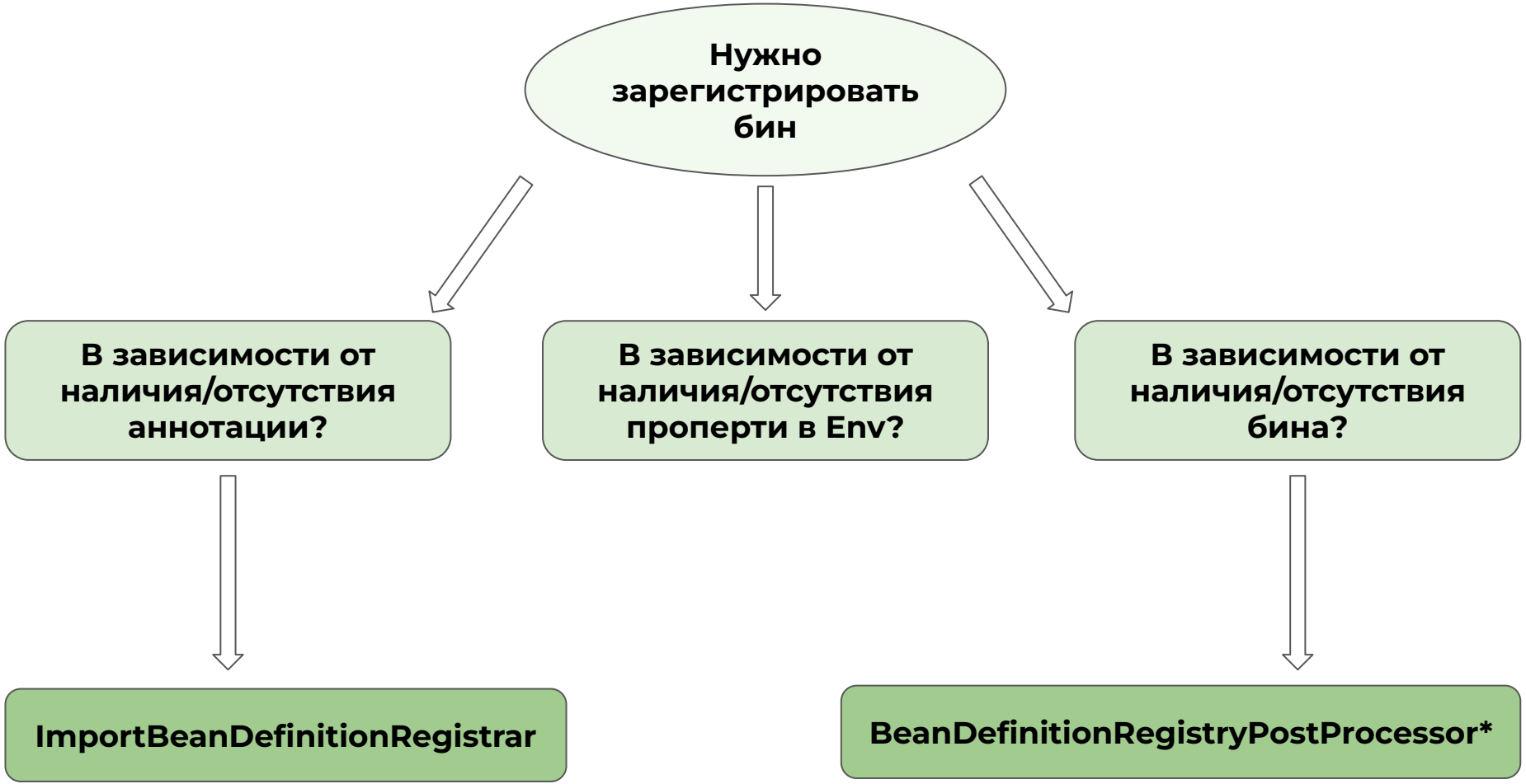

Нужно
зарегистрировать
бин

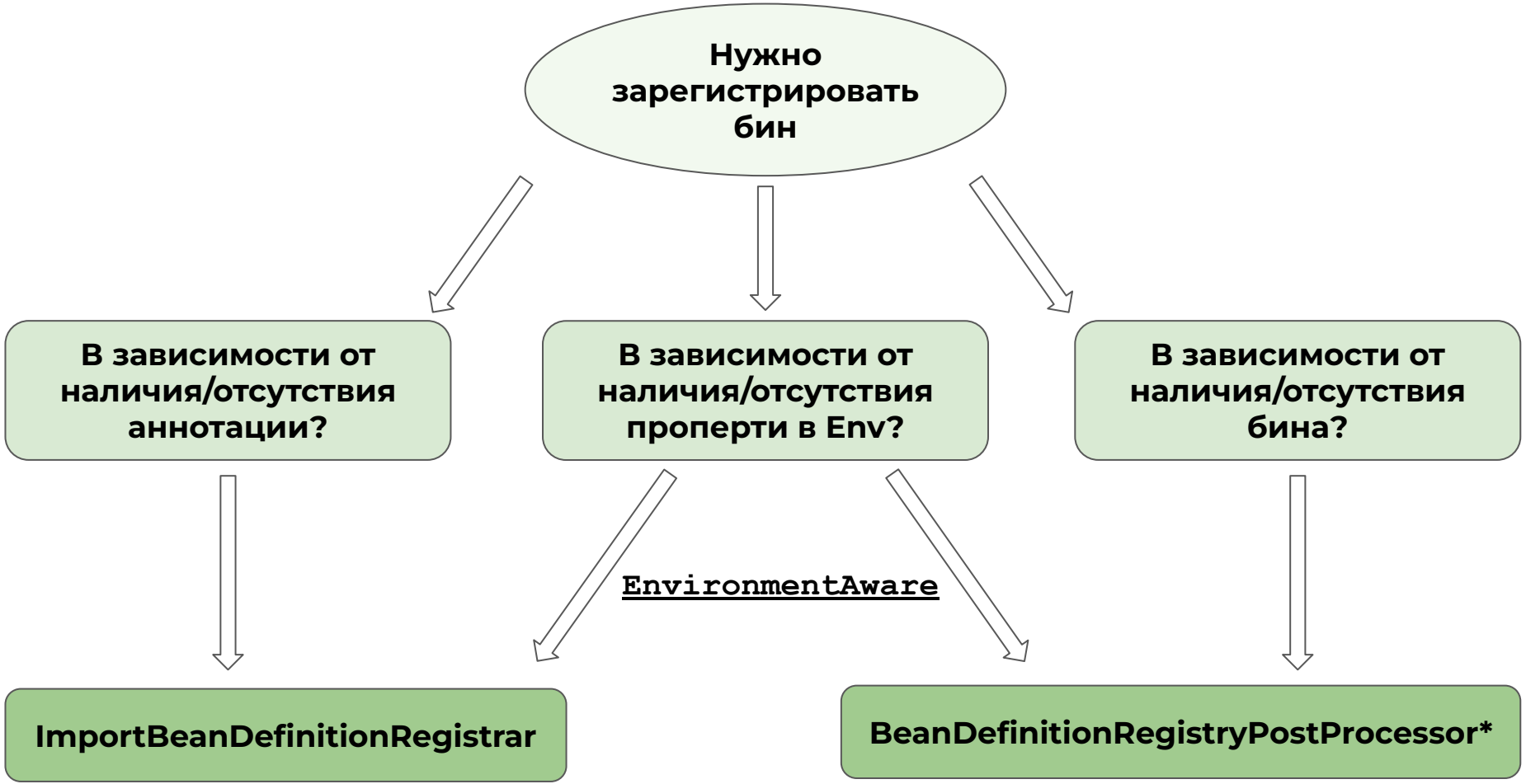

Нужно зарегистрировать бин
В зависимости от наличия/отсутствия аннотации?


Нужно зарегистрировать бин
В зависимости от наличия/отсутствия аннотации?
ImportBeanDefinitionRegistrar


Нужно зарегистрировать бин
В зависимости от наличия/отсутствия аннотации?
В зависимости от наличия/отсутствия бина?
ImportBeanDefinitionRegistrar


Нужно зарегистрировать бин
В зависимости от наличия/отсутствия аннотации?
В зависимости от наличия/отсутствия бина?
ImportBeanDefinitionRegistrar
BeanDefinitionRegistryPostProcessor*


Нужно зарегистрировать бин
В зависимости от наличия/отсутствия аннотации?
В зависимости от наличия/отсутствия проперти в Env?
В зависимости от наличия/отсутствия бина?
ImportBeanDefinitionRegistrar
BeanDefinitionRegistryPostProcessor*


Нужно зарегистрировать бин
В зависимости от наличия/отсутствия аннотации?
В зависимости от наличия/отсутствия проперти в Env?
В зависимости от наличия/отсутствия бина?
EnvironmentAware
ImportBeanDefinitionRegistrar
BeanDefinitionRegistryPostProcessor*

Способ Третий

Способ Третий BeanRegistrar

```java
/**
 * @author Sebastien Deleuze
 * @since 7.0
 */
@FunctionalInterface
public interface BeanRegistrar {

 /**
  * Register beans on the given {@link BeanRegistry} in a programmatic way.
  * @param registry the bean registry to operate on
  * @param env the environment that can be used to get the active profile or
  * some properties
  */
 void register(BeanRegistry registry, Environment env);
}
```

Способ Третий
BeanRegistrar

```
public class SanitizerBeanRegistrar implements BeanRegistrar {

  @Override
  public void register(BeanRegistry registry, Environment env) {









  }
}
```

Способ Третий
BeanRegistrar

```
public class SanitizerBeanRegistrar implements BeanRegistrar {

  @Override
  public void register(BeanRegistry registry, Environment env) {
    for (DataSensitivity sanitizer : DataSensitivity.values()) {









    }
  }
}
```

Способ Третий BeanRegistrar

```
public class SanitizerBeanRegistrar implements BeanRegistrar {

  @Override
  public void register(BeanRegistry registry, Environment env) {
    for (DataSensitivity sanitizer : DataSensitivity.values()) {

      registry.registerBean(sanitizer.beanName(), sanitizer.beanClass());




      );
    }
  }
}
```

Способ Третий BeanRegistrar

```
public class SanitizerBeanRegistrar implements BeanRegistrar {

  @Override
  public void register(BeanRegistry registry, Environment env) {
    for (DataSensitivity sanitizer : DataSensitivity.values()) {

      registry.registerBean(sanitizer.beanName(), sanitizer.beanClass(),
              spec -> spec
             .description("Sanitizer for " + sanitizer.name())
             .infrastructure()
             .lazyInit()
             .prototype()
             .primary()
      );
    }
  }
}
```

Способ Третий BeanRegistrar

Плюсы:

Способ Третий BeanRegistrar

Плюсы:

- **Полноценное управление lifecycle Spring Bean`s**

Способ Третий BeanRegistrar

Плюсы:

- **Полноценное управление lifecycle Spring Bean`s**
- **Простой доступ к Environment**

Способ Третий BeanRegistrar

Плюсы:

- **Полноценное управление lifecycle Spring Bean`s**
- **Простой доступ к Environment**
- **Kotlin DSL**

Способ Третий BeanRegistrar

Плюсы:

- **Полноценное управление lifecycle Spring Bean`s**
- **Простой доступ к Environment**
- **Kotlin DSL**
- **AOT-совместимость**

Способ Третий BeanRegistrar

Плюсы:

- **Полноценное управление lifecycle Spring Bean`s**
- **Простой доступ к Environment**
- **Kotlin DSL**
- **AOT-совместимость**

Минусы:

- **Зависимость от версии фреймворка, минимум Spring Framework 7 (Spring Boot 4)**

Способ Третий BeanRegistrar

Плюсы:

- **Полноценное управление lifecycle Spring Bean`s**
- **Простой доступ к Environment**
- **Kotlin DSL**
- **AOT-совместимость**

Минусы:

- **Зависимость от версии фреймворка, минимум Spring Framework 7 (Spring Boot 4)**
- ~~**Ограниченный функционал**~~ **?**

Способ Третий BeanRegistrar

- ~~Не подходит для создания DSL~~

Способ Третий BeanRegistrar

- ~~**Не подходит для создания DSL**~~

Евгений Зубенко, Максим Шестаков
— Мы писали DSL в Spring и грабли ловили

Способ Третий BeanRegistrar

- ~~**Не подходит для создания DSL**~~

Евгений Зубенко, Максим Шестаков
— Мы писали DSL в Spring и грабли ловили

issue

2015 -

Способ Третий BeanRegistrar

- ~~**Не подходит для создания DSL**~~

Евгений Зубенко, Максим Шестаков
— Мы писали DSL в Spring и грабли ловили

issue

2015 - 2025

BeanRegistrar

Способ Третий BeanRegistrar

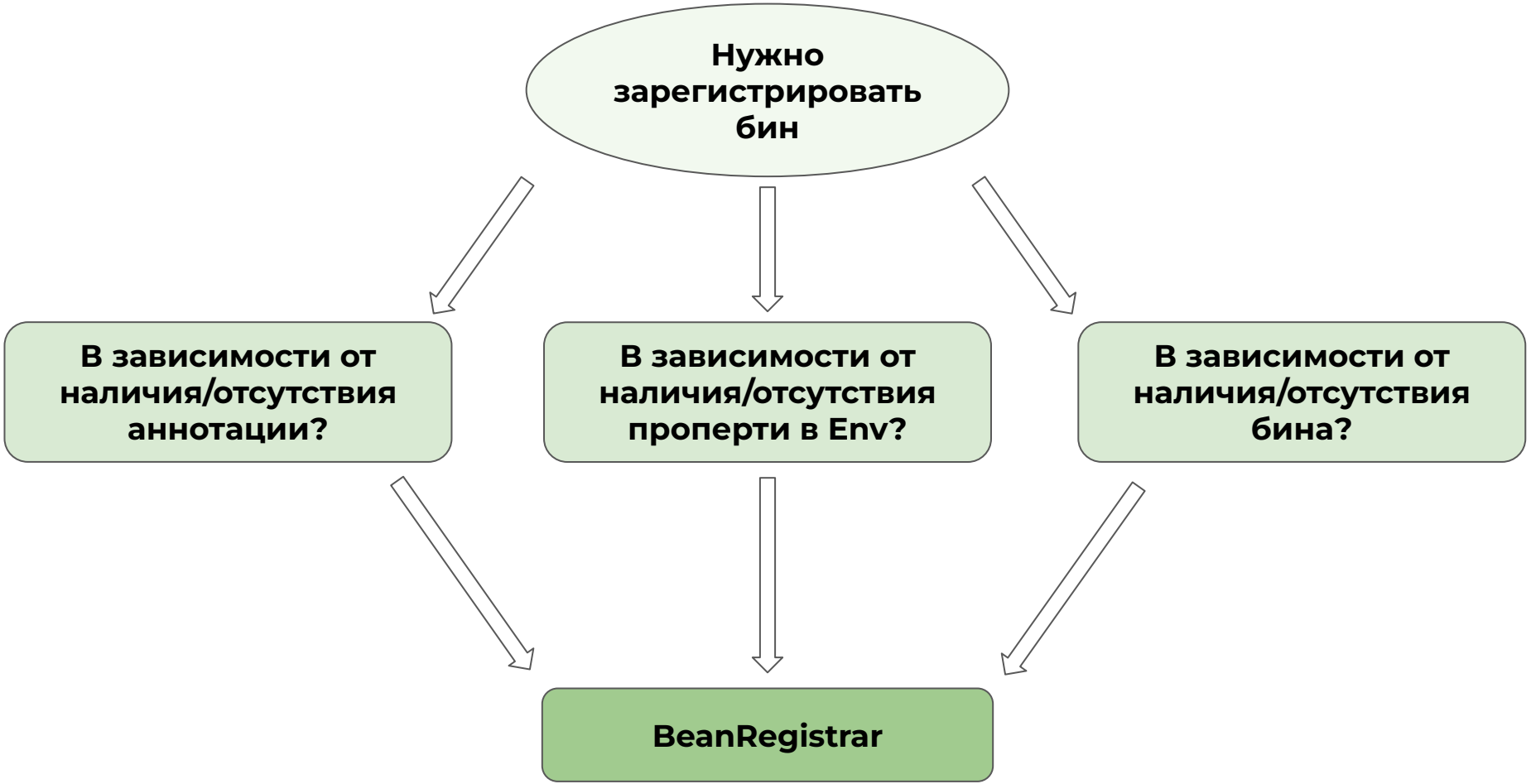
“Посмотреть, есть ли в контексте бины определённого типа, и если есть создать для каждого из них дополнительный бин”

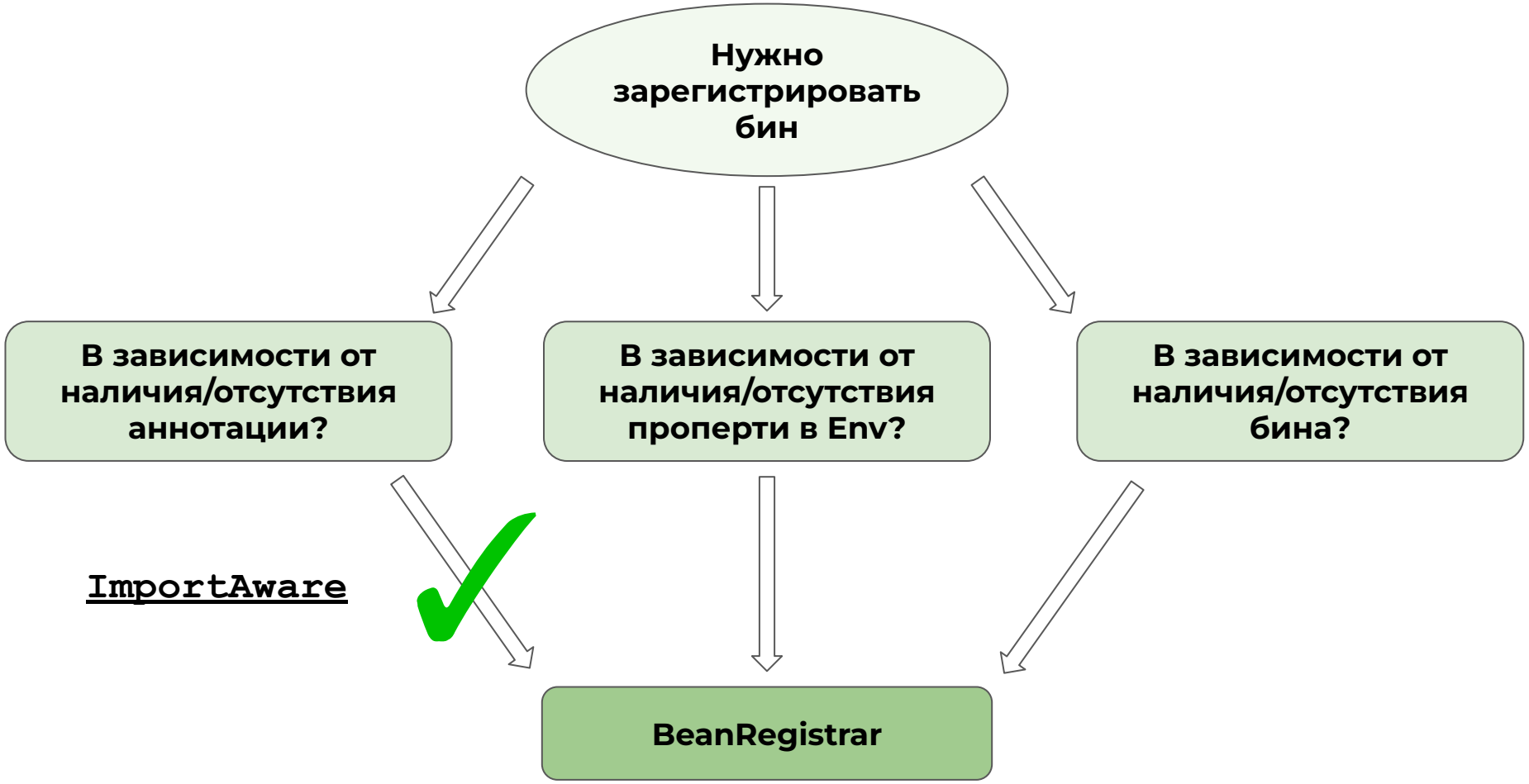
Способ Третий BeanRegistrar

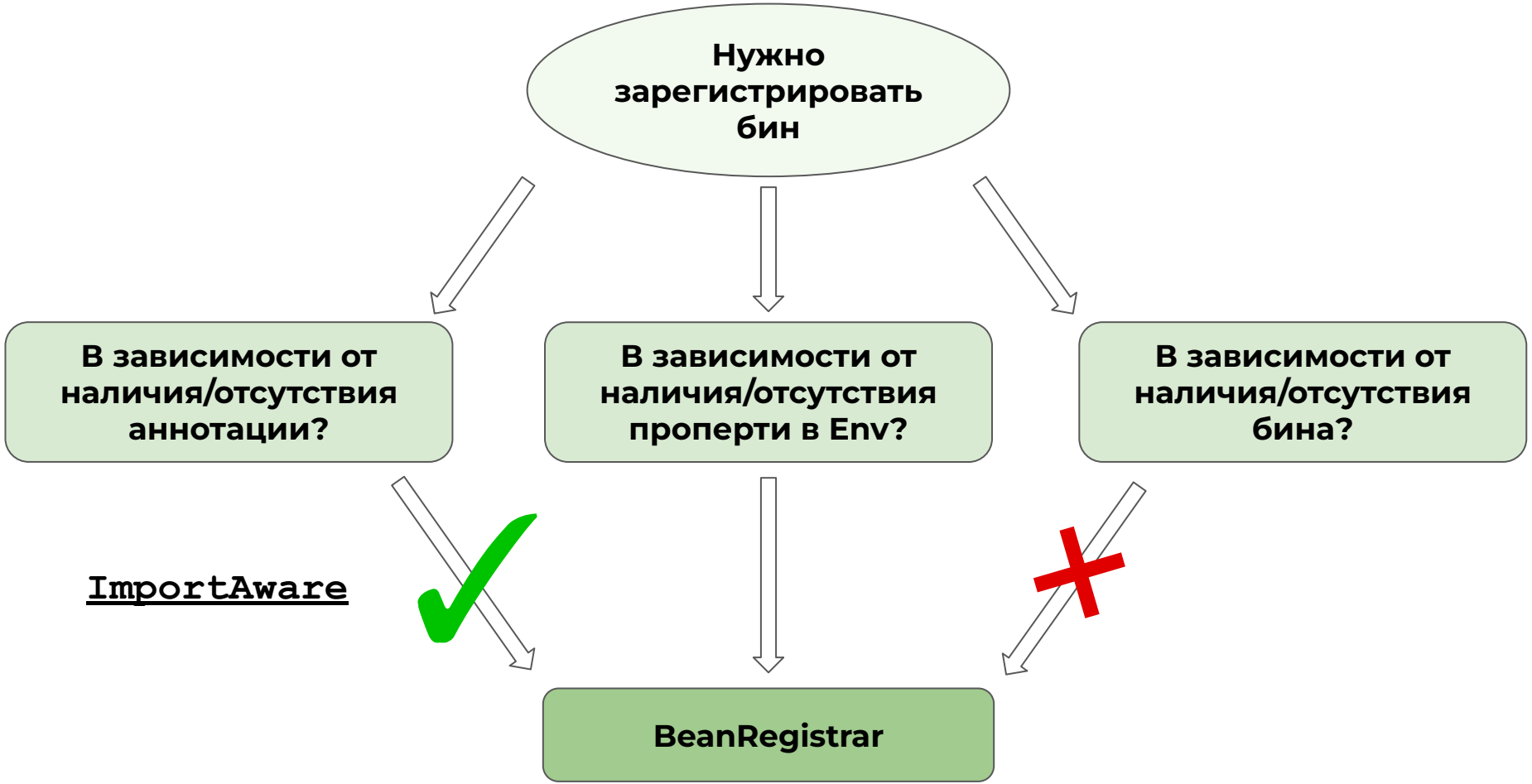
“Посмотреть, есть ли в контексте бины определённого типа, и если есть создать для каждого из них дополнительный бин”

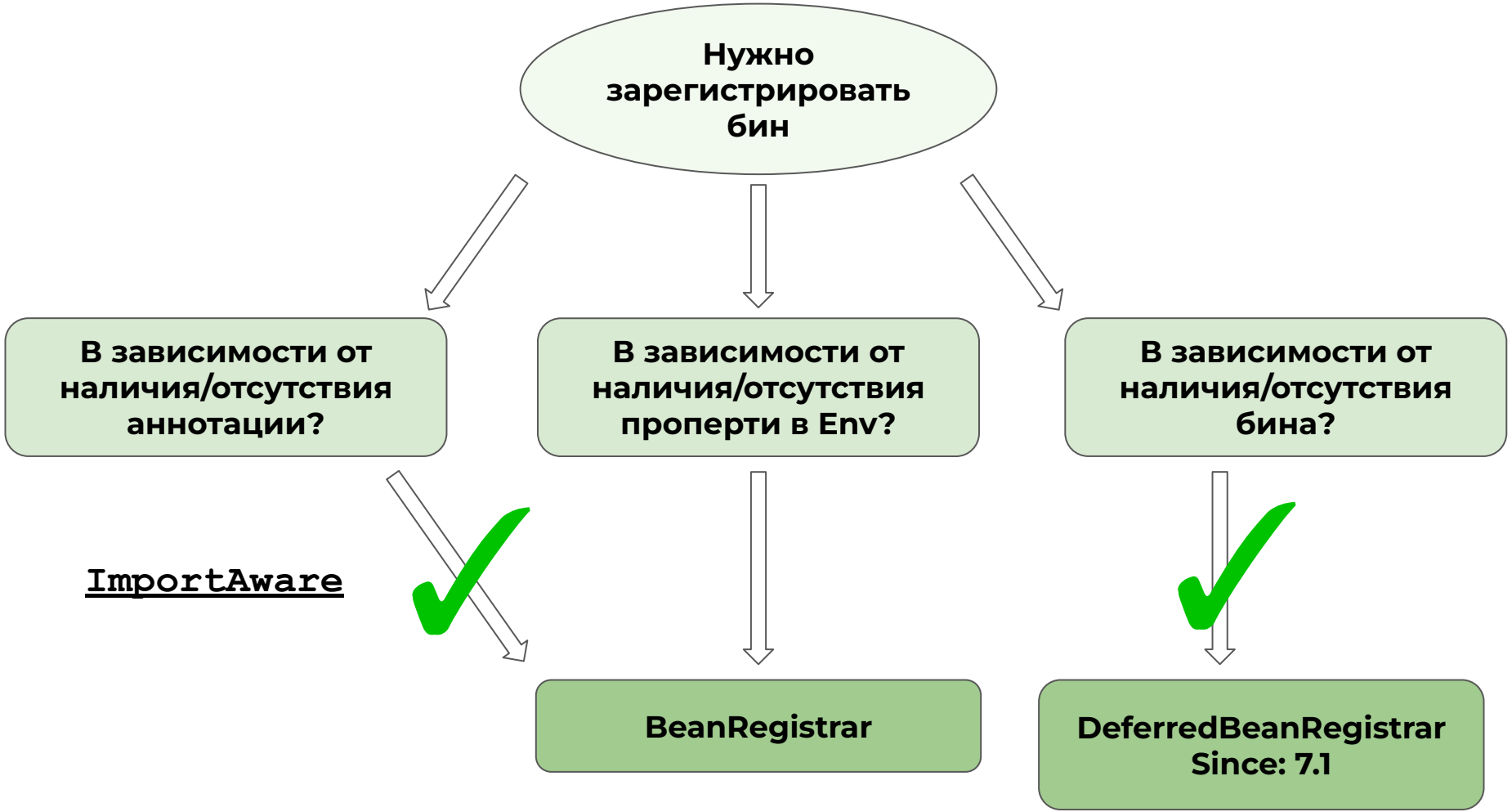
DeferredBeanRegistrar

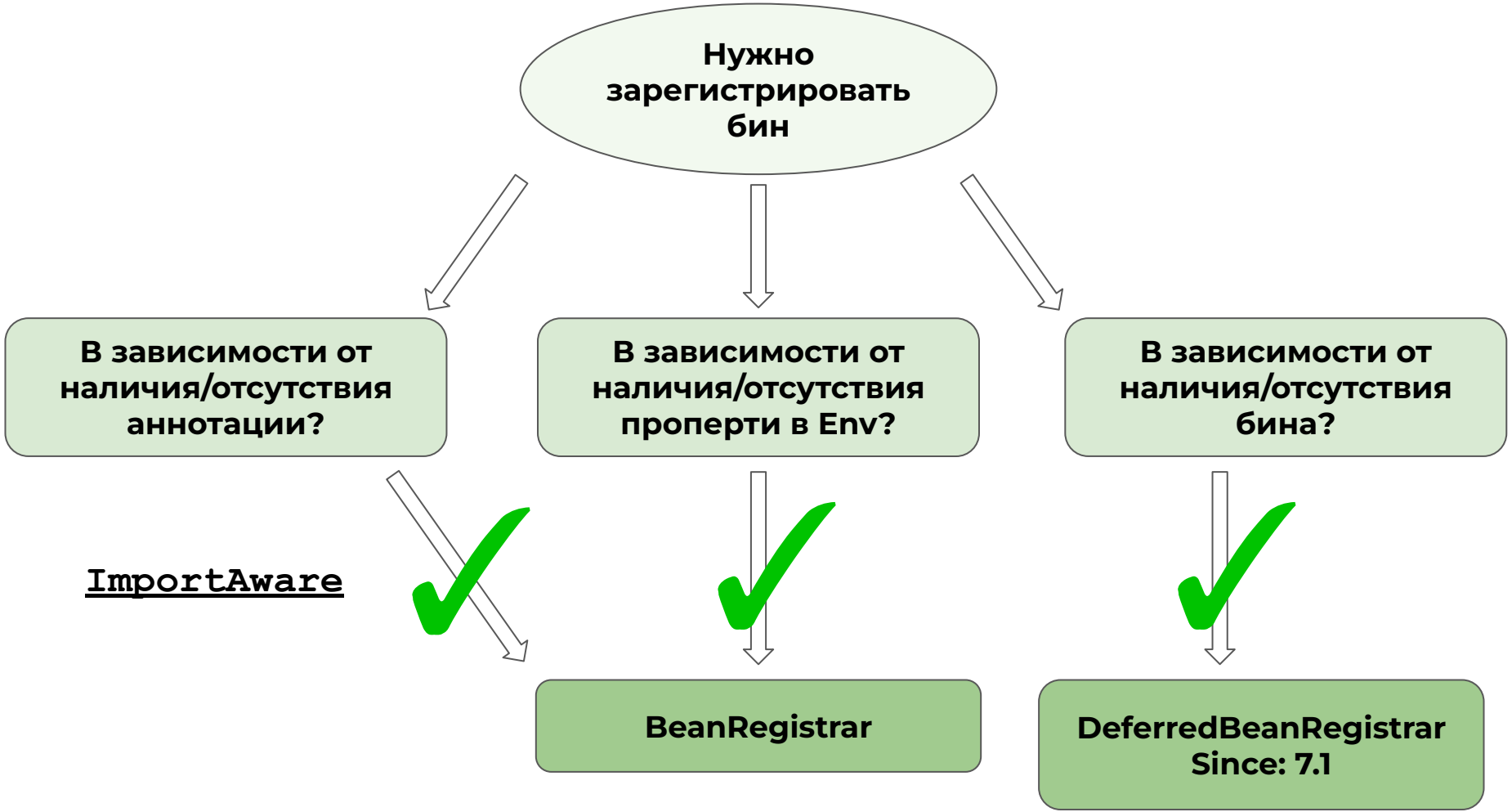
Since: 7.1


Нужно зарегистрировать бин
В зависимости от наличия/отсутствия аннотации?
В зависимости от наличия/отсутствия проперти в Env?
В зависимости от наличия/отсутствия бина?
BeanRegistrar


Нужно зарегистрировать бин
В зависимости от наличия/отсутствия аннотации?
В зависимости от наличия/отсутствия проперти в Env?
В зависимости от наличия/отсутствия бина?
ImportAware
BeanRegistrar


Нужно зарегистрировать бин
В зависимости от наличия/отсутствия аннотации?
В зависимости от наличия/отсутствия проперти в Env?
В зависимости от наличия/отсутствия бина?
ImportAware
BeanRegistrar


Нужно зарегистрировать бин
В зависимости от наличия/отсутствия аннотации?
В зависимости от наличия/отсутствия проперти в Env?
В зависимости от наличия/отсутствия бина?
ImportAware
BeanRegistrar
DeferredBeanRegistrar
Since: 7.1

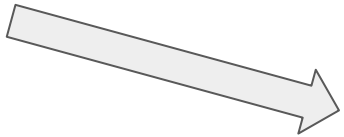

Нужно зарегистрировать бин
В зависимости от наличия/отсутствия аннотации?
В зависимости от наличия/отсутствия проперти в Env?
В зависимости от наличия/отсутствия бина?
ImportAware
BeanRegistrar
DeferredBeanRegistrar
Since: 7.1

Чем Является BeanRegistrar?

BeanRegistrar

Чем Является BeanRegistrar?

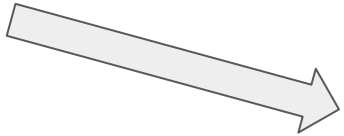
BeanRegistrar



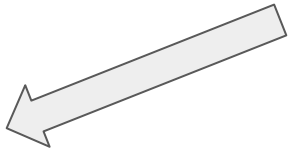
BeanRegistry
(BeanRegistryAdapter)
@since Spring 7.0

Чем Является BeanRegistrar?

BeanRegistrar



BeanRegistry
(BeanRegistryAdapter)
@since Spring 7.0



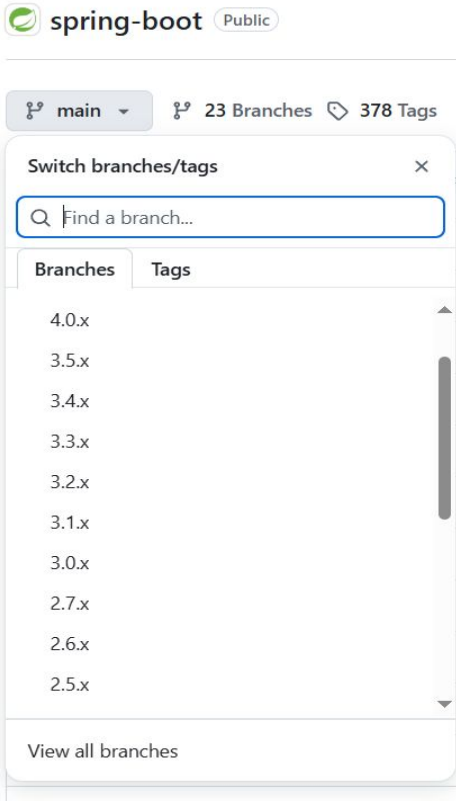
BeanDefinitionRegistry

ImportBeanDefinitionRegistrar ✓

Axelix

Часть №2
Проблема совместимости стартеров для major версий Spring Boot

Способ Первый: Отдельные модули Release Branching



Способ Первый: Отдельные модули Release Branching

Плюсы:

Способ Первый: Отдельные модули Release Branching

Плюсы:

- **Чистые зависимости**

Способ Первый: Отдельные модули Release Branching

Плюсы:

- **Чистые зависимости**
- **Знакомый и проверенный подход, именно так делает Spring Boot**

Способ Первый: Отдельные модули Release Branching

Плюсы:

- **Чистые зависимости**
- **Знакомый и проверенный подход, именно так делает Spring Boot**

Минусы:

- **Дублирование кода**

Способ Первый: Отдельные модули Release Branching

Плюсы:

- **Чистые зависимости**
- **Знакомый и проверенный подход, именно так делает Spring Boot**

Минусы:

- **Дублирование кода**
- **Масштабирование команды**

Способ Второй: Условия в коде Runtime Checks

Способ Второй: Условия в коде Runtime Checks

```
public class TracingInterceptor implements MethodInterceptor {

  private final Object tracer;

  public TracingInterceptor(Object tracer) { this.tracer = tracer; }

  @Override
  public Object invoke(MethodInvocation invocation) throws Throwable {
    try { return invocation.proceed(); }
    finally {
      Map<String, String> tags = buildTags(invocation.getArguments(), getAnn(invocation));

      if (isBrave()) { // Spring Boot 2 — Brave


      } else {          // Spring Boot 3 — Micrometer


      }
    }
  }

  private Map<String, String> buildTags(Object[] args, Annotation[][] ann) {
    Map<String, String> tags = new LinkedHashMap<>();
    for (int i = 0; i < args.length; i++) { tags.put("param." + i, sanitize(args[i], ann[i])); }
    return tags;
  }

  private boolean isBrave() {
   return ClassUtils.isPresent("brave.Tracer", null);
  }
```

Способ Второй: Условия в коде Runtime Checks

```
public class TracingInterceptor implements MethodInterceptor {

  private final Object tracer;

  public TracingInterceptor(Object tracer) { this.tracer = tracer; }

  @Override
  public Object invoke(MethodInvocation invocation) throws Throwable {
    try { return invocation.proceed(); }
    finally {
      Map<String, String> tags = buildTags(invocation.getArguments(), getAnn(invocation));

      if (isBrave()) { // Spring Boot 2 — Brave
        brave.Span span = ((brave.Tracer) tracer).currentSpan();
        tags.forEach(span::tag);
      } else {         // Spring Boot 3 — Micrometer
        io.micrometer.tracing.Span span = ((io.micrometer.tracing.Tracer) tracer).currentSpan();
        tags.forEach(span::tag);
      }
    }
  }

  private Map<String, String> buildTags(Object[] args, Annotation[][] ann) {
    Map<String, String> tags = new LinkedHashMap<>();
    for (int i = 0; i < args.length; i++) { tags.put("param." + i, sanitize(args[i], ann[i])); }
    return tags;
  }

  private boolean isBrave() {
    return ClassUtils.isPresent("brave.Tracer", null);
  }
```

Способ Второй: Условия в коде Runtime Checks

Плюсы:

Способ Второй: Условия в коде Runtime Checks

Плюсы:

- **Подходит для небольших библиотек**

Способ Второй: Условия в коде Runtime Checks

Плюсы:

- **Подходит для небольших библиотек**
- **Один модуль**

Способ Второй: Условия в коде Runtime Checks

Плюсы:

- **Подходит для небольших библиотек**
- **Один модуль**

Минусы:

- **Тяжело читаемый код**

Способ Второй: Условия в коде Runtime Checks

Плюсы:

- **Подходит для небольших библиотек**
- **Один модуль**

Минусы:

- **Тяжело читаемый код**
- **Сложно тестировать, плохо масштабируется**

Способ Второй: Условия в коде Runtime Checks

Плюсы:

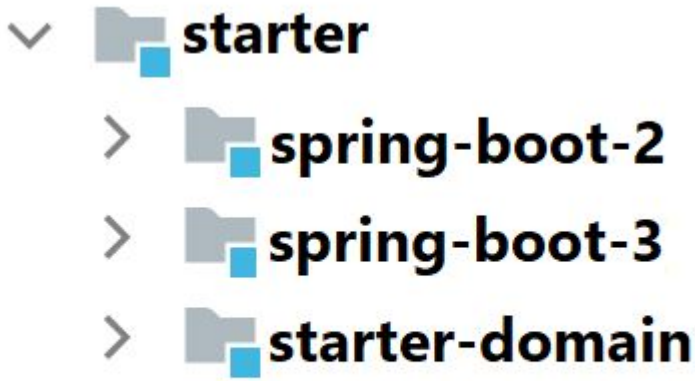
- **Подходит для небольших библиотек**
- **Один модуль**

Минусы:

- **Тяжело читаемый код**
- **Сложно тестировать, плохо масштабируется**
- **Jar Hell**

Способ Третий: Общий domain-модуль

Способ Третий: Общий domain-модуль



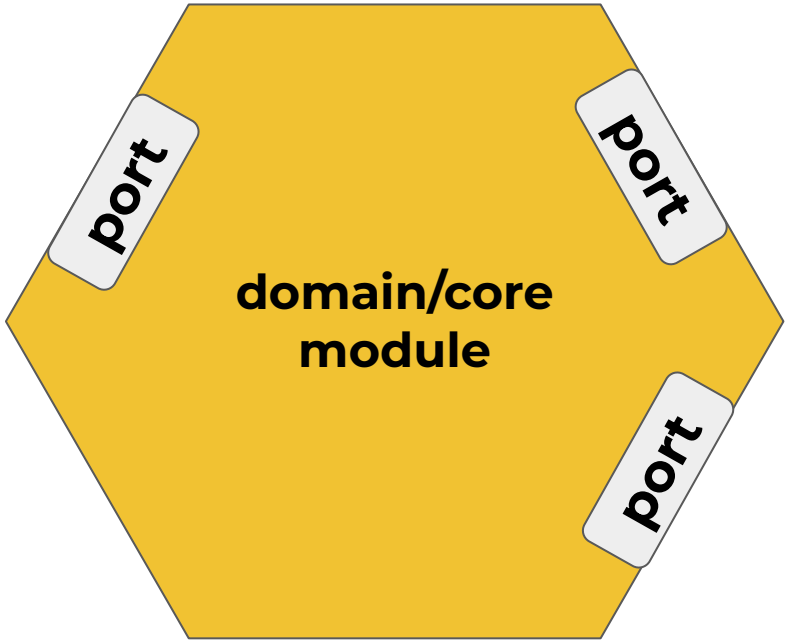

```java
// NO Spring, NO Brave, NO Micrometer imports
public class SanitizingProcessor {

    private final SanitizerRegistry sanitizers;

    public SanitizedOutcome process(Object[] args, Annotation[][] paramAnnotations) {
        Map<String, String> tags = new LinkedHashMap<>();
        for (int i = 0; i < args.length; i++) {
            String sanitized = sanitizers.sanitize(args[i], paramAnnotations[i]);
            tags.put("param." + i, sanitized);
        }
        return new SanitizedOutcome(tags);
    }

    public record SanitizedOutcome(Map<String, String> tags) {}
}
```


```
import brave.Tracer;                                    // ┐
import brave.Span;                                      // ┘ специфично для Spring Boot 2

public class TracingInterceptor implements MethodInterceptor {

    private final Tracer tracer;
    private final SanitizingProcessor sanitizingProcessor;

    @Override
    public Object invoke(MethodInvocation invocation) throws Throwable {
        try { return invocation.proceed(); }

        finally {
            SanitizedOutcome outcome = sanitizingProcessor.process(
                    invocation.getArguments(),
                    getParamAnnotations(invocation)
            );
            Span span = tracer.currentSpan();
            outcome.tags().forEach(span::tag);
        }
    }
}
```

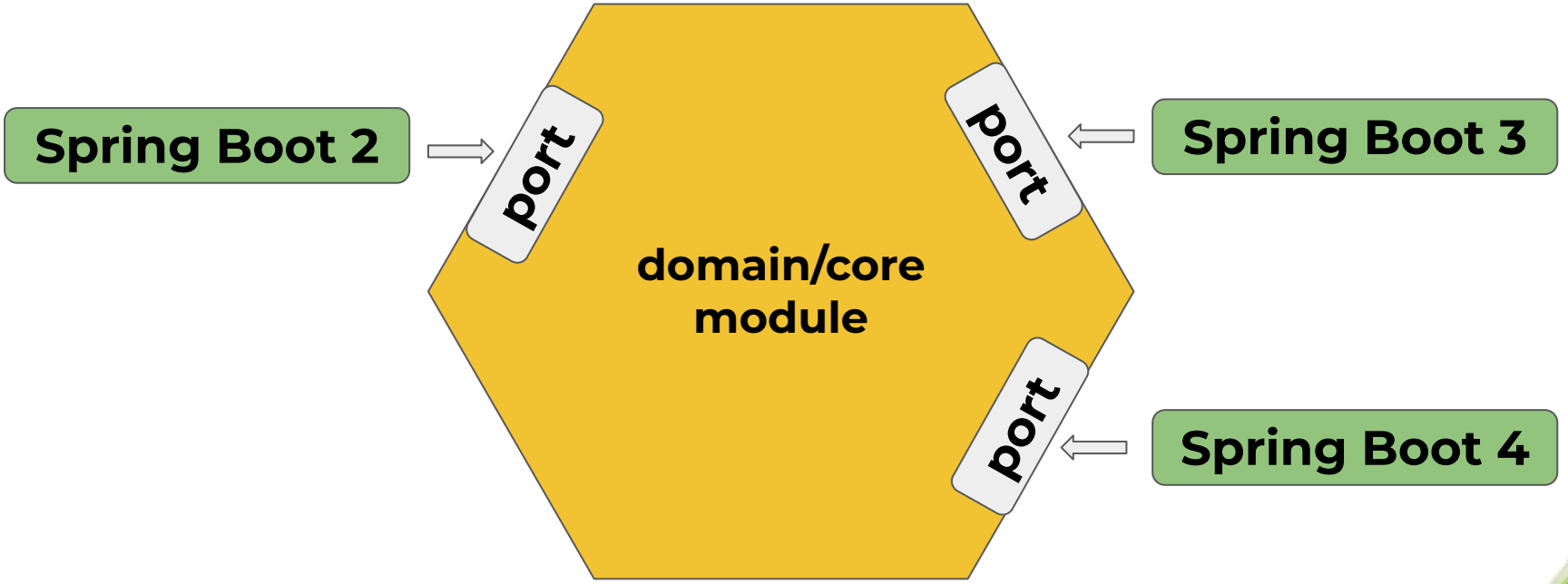
spring-boot-3

```
import io.micrometer.tracing.Tracer;          // ┐
import io.micrometer.tracing.Span;            // ┘ специфично для Spring Boot 3

public class TracingInterceptor implements MethodInterceptor {

    private final Tracer tracer;
    private final SanitizingProcessor sanitizingProcessor;

    @Override
    public Object invoke(MethodInvocation invocation) throws Throwable {
        try { return invocation.proceed(); }

        finally {
            SanitizedOutcome outcome = sanitizingProcessor.process(
                    invocation.getArguments(),
                    getParamAnnotations(invocation)
            );
            Span span = tracer.currentSpan();
            outcome.tags().forEach(span::tag);
        }
    }
}
```

Способ Третий:
Общий domain-модуль



Способ Третий: Общий domain-модуль



Способ Третий: Общий domain-модуль

Плюсы:

Способ Третий: Общий domain-модуль

Плюсы:

- **Чистые зависимости**

Способ Третий: Общий domain-модуль

Плюсы:

- **Чистые зависимости**
- **Легко тестировать**

Способ Третий: Общий domain-модуль

Плюсы:

- **Чистые зависимости**
- **Легко тестировать**
- **Масштабируется**

Способ Третий: Общий domain-модуль

Плюсы:

- **Чистые зависимости**
- **Легко тестировать**
- **Масштабируется**
- **Меньше дублирования кода**

Способ Третий: Общий domain-модуль

Плюсы:

- **Чистые зависимости**
- **Легко тестировать**
- **Масштабируется**
- **Меньше дублирования кода**

Минусы:

- **Усложнение структуры проекта и выше порог входа**

Общий domain-модуль 

Axelix

Часть №3
Тестирование @Autoconfiguration`s

Тестирование @Autoconfiguration`s

Почему это важно?

Тестирование @Autoconfiguration`s

Почему это важно?

Тестирование @Autoconfiguration`s

```
@AutoConfiguration
@ConditionalOnProperty(prefix = "extended.tracer", name = "enabled", havingValue = "true")
public class TracerAutoConfiguration {

    @Bean
    @ConditionalOnMissingBean
    public TracingInterceptor tracingInterceptor(
            Tracer tracer,
            SanitizingProcessor processor) {
        return new TracingInterceptor(tracer, processor);
    }

    @Bean
    @ConditionalOnMissingBean
    public SanitizingProcessor sanitizingProcessor(SanitizerRegistry sanitizers) {
        return new SanitizingProcessor(sanitizers);
    }
}
```

@SpringBootTest

Тестирование @Autoconfiguration`s С Использованием @SpringBootTest

Тестирование @Autoconfiguration`s С Использованием @SpringBootTest

```
@SpringBootTest(
        classes = TracerAutoConfiguration.class,
        properties = "extended.tracer.enabled=true")

public class WhenEnabledTest {









}
```

Тестирование @Autoconfiguration`s С Использованием @SpringBootTest

```
@SpringBootTest(
        classes = TracerAutoConfiguration.class,
        properties = "extended.tracer.enabled=true")

public class WhenEnabledTest {

    @Autowired
    private ApplicationContext context;

    @Test
    void shouldCreateAllBeans() {
     assertThat(context.containsBean("sanitizingProcessor")).isTrue();
     assertThat(context.containsBean("tracingInterceptor")).isTrue();
    }
}
```

Тестирование @Autoconfiguration`s С Использованием @SpringBootTest

```
public class TracerAutoConfigurationWithSpringBootTest {

    @SpringBootTest(
            classes = TracerAutoConfiguration.class,
            properties = "extended.tracer.enabled=true")
    @Nested
    public static class WhenEnabledTest {

        @Autowired
        private ApplicationContext context;

        @Test
        void shouldCreateAllBeans() {
         assertThat(context.containsBean("sanitizingProcessor")).isTrue();
         assertThat(context.containsBean("tracingInterceptor")).isTrue();
        }
    }

}
```

Тестирование @Autoconfiguration`s С Использованием @SpringBootTest

```
public class TracerAutoConfigurationWithSpringBootTest {

    @SpringBootTest(
            classes = TracerAutoConfiguration.class,
            properties = "extended.tracer.enabled=true")
    @Nested
    public static class WhenEnabledTest {

        @Autowired
        private ApplicationContext context;

        @Test
        void shouldCreateAllBeans() {
         assertThat(context.containsBean("sanitizingProcessor")).isTrue();
         assertThat(context.containsBean("tracingInterceptor")).isTrue();
        }
    }

    @SpringBootTest(
            classes = TracerAutoConfiguration.class,
            properties = "extended.tracer.enabled=false")
    @Nested
    public static class WhenDisabledTest {









    }
}
```

Тестирование @Autoconfiguration`s С Использованием @SpringBootTest

```
public class TracerAutoConfigurationWithSpringBootTest {

    @SpringBootTest(
            classes = TracerAutoConfiguration.class,
            properties = "extended.tracer.enabled=true")
    @Nested
    public static class WhenEnabledTest {

        @Autowired
        private ApplicationContext context;

        @Test
        void shouldCreateAllBeans() {
         assertThat(context.containsBean("sanitizingProcessor")).isTrue();
         assertThat(context.containsBean("tracingInterceptor")).isTrue();
        }
    }

    @SpringBootTest(
            classes = TracerAutoConfiguration.class,
            properties = "extended.tracer.enabled=false")
    @Nested
    public static class WhenDisabledTest {

        @Autowired
        private ApplicationContext context;

        @Test
        void shouldNotCreateAnyBeans() {
          assertThat(context.containsBean("sanitizingProcessor")).isFalse();
          assertThat(context.containsBean("tracingInterceptor")).isFalse();
        }
    }
}
```

Тестирование @Autoconfiguration`s С Использованием @SpringBootTest

Плюсы:

Тестирование @Autoconfiguration`s С Использованием @SpringBootTest

Плюсы:

- **Всем знакомая аннотация @SpringBootTest**

Тестирование @Autoconfiguration`s С Использованием @SpringBootTest

Плюсы:

- **Всем знакомая аннотация @SpringBootTest**
- **… “их нет”**

Тестирование @Autoconfiguration`s С Использованием @SpringBootTest

Плюсы:

- Всем знакомая аннотация @SpringBootTest
- … “их нет”

Минусы:

- Слишком тяжеловесно, замедляет сборку

Тестирование @Autoconfiguration`s С Использованием @SpringBootTest

Плюсы:

- **Всем знакомая аннотация @SpringBootTest**
- **… “их нет”**

Минусы:

- **Слишком тяжеловесно, замедляет сборку**
- **Неоптимальный подход**

Как правильно?

ApplicationContextRunner

Тестирование @Autoconfiguration`s С Использованием ApplicationContextRunner

Ключевые возможности

Тестирование @Autoconfiguration`s С Использованием ApplicationContextRunner

Ключевые возможности

- **Симуляция окружения**

Тестирование @Autoconfiguration`s С Использованием ApplicationContextRunner

Ключевые возможности

- **Симуляция окружения**
- **Условия загрузки (beans & classes)**

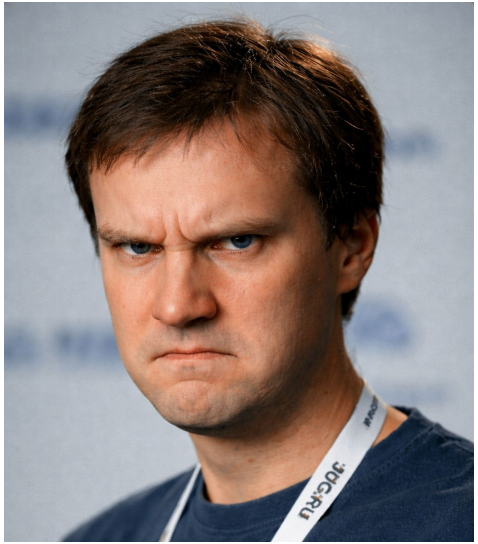
```
public class TracerAutoConfigurationTest {

    private final ApplicationContextRunner contextRunner = new ApplicationContextRunner()
            .withConfiguration(AutoConfigurations.of(TracerAutoConfiguration.class));









}
```

```
public class TracerAutoConfigurationTest {

    private final ApplicationContextRunner contextRunner = new ApplicationContextRunner()
            .withConfiguration(AutoConfigurations.of(TracerAutoConfiguration.class));

    @Test
    void whenEnabledShouldCreateBeans() {
        contextRunner
                .withPropertyValues("extended.tracer.enabled=true",)
                .withBean(Tracer.class, () -> mock(Tracer.class))
                .run(context -> {
                    assertThat(context).hasSingleBean(SanitizingProcessor.class);
                    assertThat(context).hasSingleBean(TracingInterceptor.class);
                });
    }

}
```

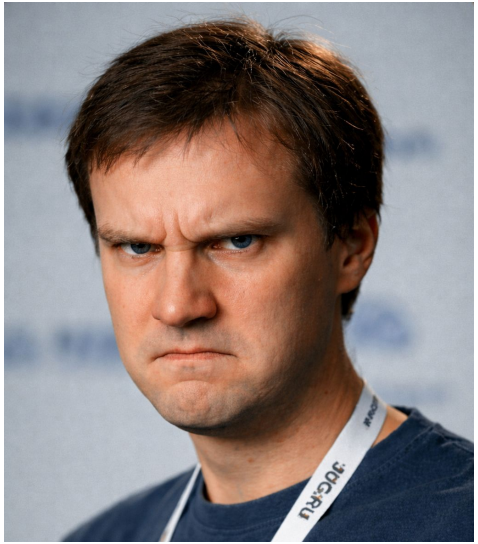
```
public class TracerAutoConfigurationTest {

    private final ApplicationContextRunner contextRunner = new ApplicationContextRunner()
            .withConfiguration(AutoConfigurations.of(TracerAutoConfiguration.class));

    @Test
    void whenEnabledShouldCreateBeans() {
        contextRunner
                .withPropertyValues("extended.tracer.enabled=true",)
                .withBean(Tracer.class, () -> mock(Tracer.class))
                .run(context -> {
                    assertThat(context).hasSingleBean(SanitizingProcessor.class);
                    assertThat(context).hasSingleBean(TracingInterceptor.class);
                });
    }

    @Test
    void whenDisabledShouldNotCreateBeans() {
        contextRunner
                .withPropertyValues("extended.tracer.enabled=false")
                .run(context -> {
                    assertThat(context).doesNotHaveBean(SanitizingProcessor.class);
                    assertThat(context).doesNotHaveBean(TracingInterceptor.class);
                });
    }
}
```

ApplicationContextRunner vs @SpringBootTest

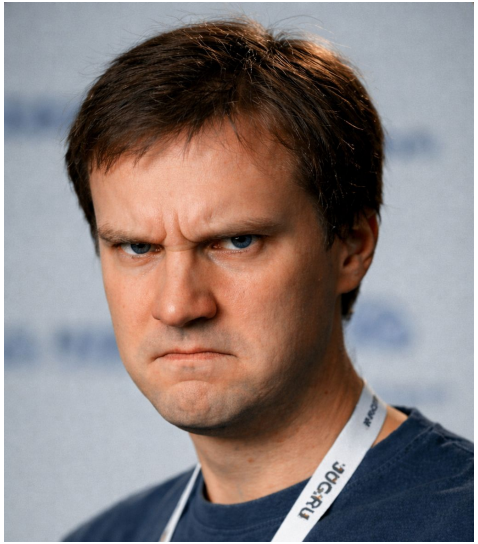
ApplicationContextRunner vs @SpringBootTest



ApplicationContextRunner vs @SpringBootTest



ApplicationContextRunner vs @SpringBootTest



ApplicationContextRunner vs @SpringBootTest

100 итераций:

ApplicationContextRunner vs @SpringBootTest

100 итераций:

- **@SpringBootTest ≈ 103 сек**

ApplicationContextRunner vs @SpringBootTest

100 итераций:

- **@SpringBootTest ≈ 103 сек**
- **ApplicationContextRunner ≈ 0,5 сек**

ApplicationContextRunner vs @SpringBootTest

100 итераций:

- **@SpringBootTest ≈ 103 сек**
- **ApplicationContextRunner ≈ 0,5 сек**

1000 итераций:

- **@SpringBootTest ≈ 1023 сек**

ApplicationContextRunner vs @SpringBootTest

100 итераций:

- **@SpringBootTest ≈ 103 сек**
- **ApplicationContextRunner ≈ 0,5 сек**

1000 итераций:

- **@SpringBootTest ≈ 1023 сек**
- **ApplicationContextRunner ≈ 2,1 сек**

ApplicationContextRunner vs @SpringBootTest

100 итераций:

- **@SpringBootTest ≈ 103 сек**
- **ApplicationContextRunner ≈ 0,5 сек**

1000 итераций:

- **@SpringBootTest ≈ 1023 сек**
- **ApplicationContextRunner ≈ 2,1 сек**

***@SpringBootTest отключено поднятие веб-сервера**

Почему такая разница во времени тестирования ?

Почему такая разница во времени тестирования ?

- **Тестирование с аннотацией @SpringBootTest**
  <u>без доп. настроек</u> - полное поднятие конекста

Почему такая разница во времени тестирования ?

- **Тестирование с аннотацией @SpringBootTest <u>без доп. настроек</u> - полное поднятие конекста**

- **@SpringBootTest нацелен на Cache Hit**

Почему такая разница во времени тестирования ?

- **Тестирование с аннотацией @SpringBootTest <u>без доп. настроек</u> - полное поднятие конекста**
- **@SpringBootTest нацелен на Cache Hit**
- **Тестирование @Autoconfiguration`s почти всегда Cache Miss**

Как Spring кеширует контексты в тестах?

Как Spring кеширует контексты в тестах?

```
public class MergedContextConfiguration implements Serializable {
      ...
   private final Class<?>[] classes;
   private final String[] activeProfiles;
   private final String[] propertySourceProperties;
   private final Set<ContextCustomizer> contextCustomizers
      ...
}
```

Как Spring кеширует контексты в тестах?

```
public class MergedContextConfiguration implements Serializable {
      ...
    private final Class<?>[] classes;
    private final String[] activeProfiles;
    private final String[] propertySourceProperties;
    private final Set<ContextCustomizer> contextCustomizers
      ...
}

public class DefaultContextCache implements ContextCache {

   /**
    * Map of context keys to Spring {@code ApplicationContext} instances.
    */
   private final Map<MergedContextConfiguration, ApplicationContext> contextMap =
     Collections.synchronizedMap(new LinkedHashMap<>(32, 0.75f, true));
   ...
}
```

Новое в кешировании ApplicationContext (Spring 7.0.x)

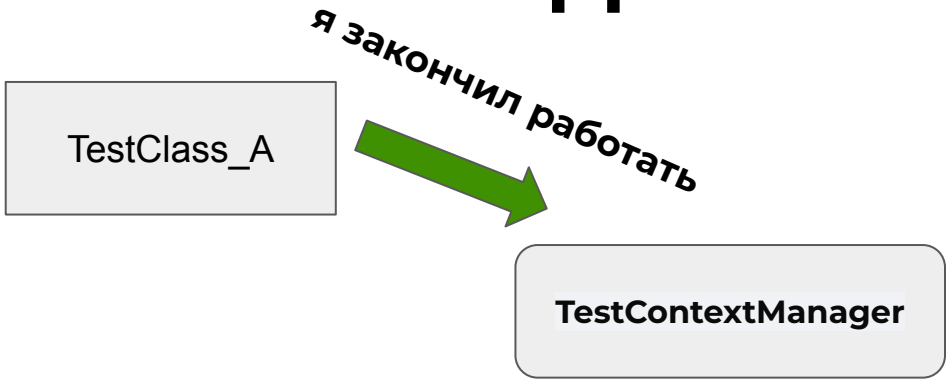
Новое в кешировании ApplicationContext (Spring 7.0.x)

```
/**
* Map of context keys to active test classes (i.e., test classes that are actively
* using the corresponding {@link ApplicationContext}).
* @since 7.0
*/
private final Map<MergedContextConfiguration, Set<Class<?>>> contextUsageMap =
      new ConcurrentHashMap<>(32);
```

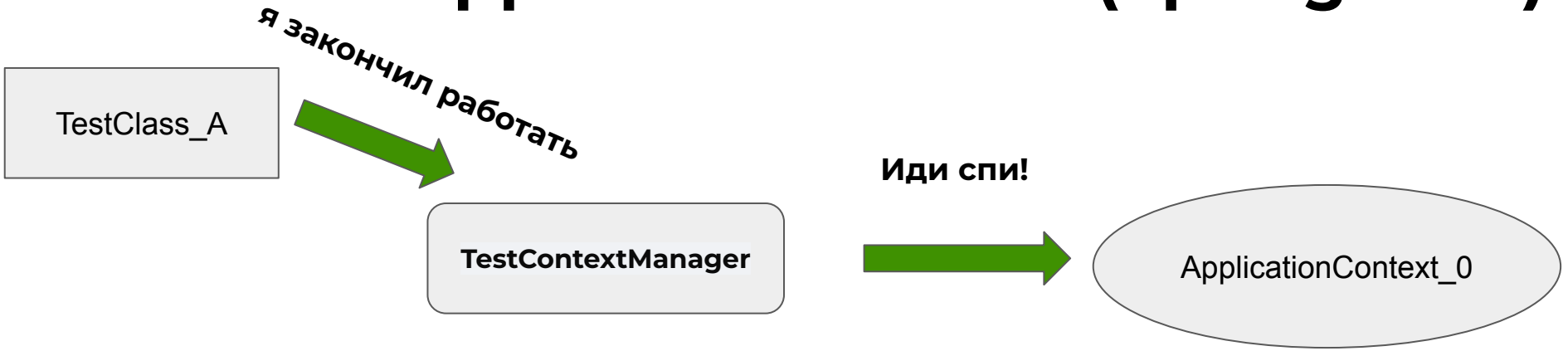
Новое в кешировании ApplicationContext (Spring 7.0.x)

```
/**
* Map of context keys to active test classes (i.e., test classes that are actively
* using the corresponding {@link ApplicationContext}).
* @since 7.0
*/
private final Map<MergedContextConfiguration, Set<Class<?>>> contextUsageMap =
      new ConcurrentHashMap<>(32);
```

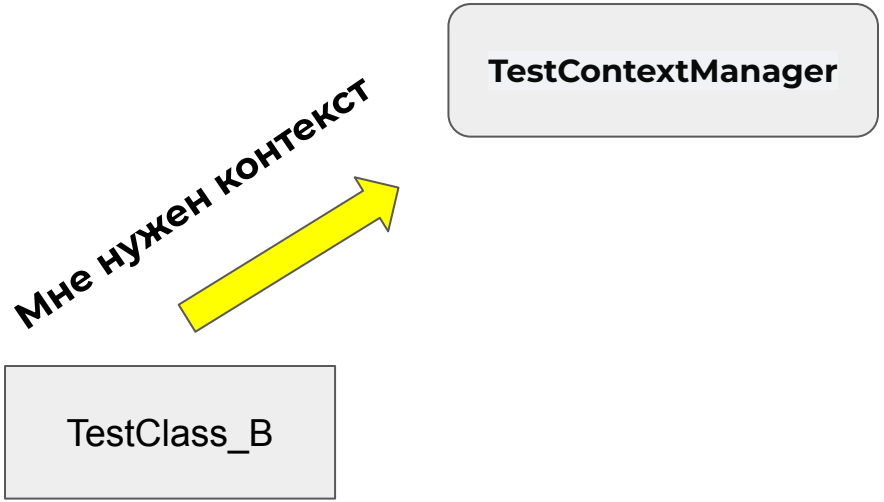
Новое в кешировании ApplicationContext (Spring 7.0.x)



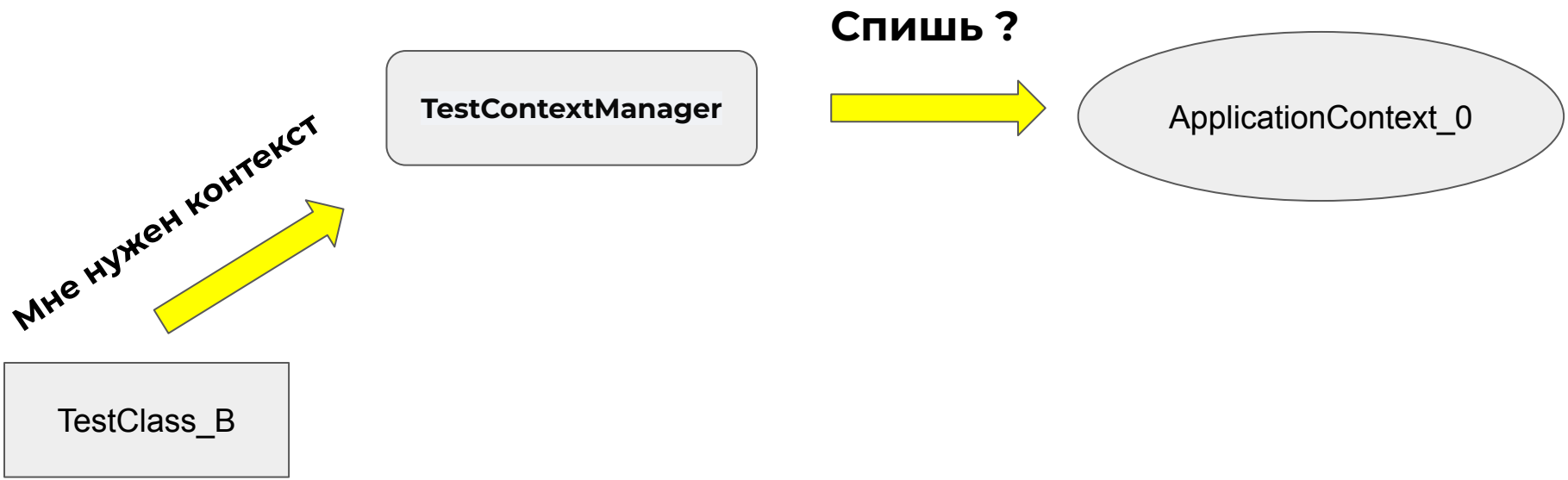
Новое в кешировании ApplicationContext (Spring 7.0.x)



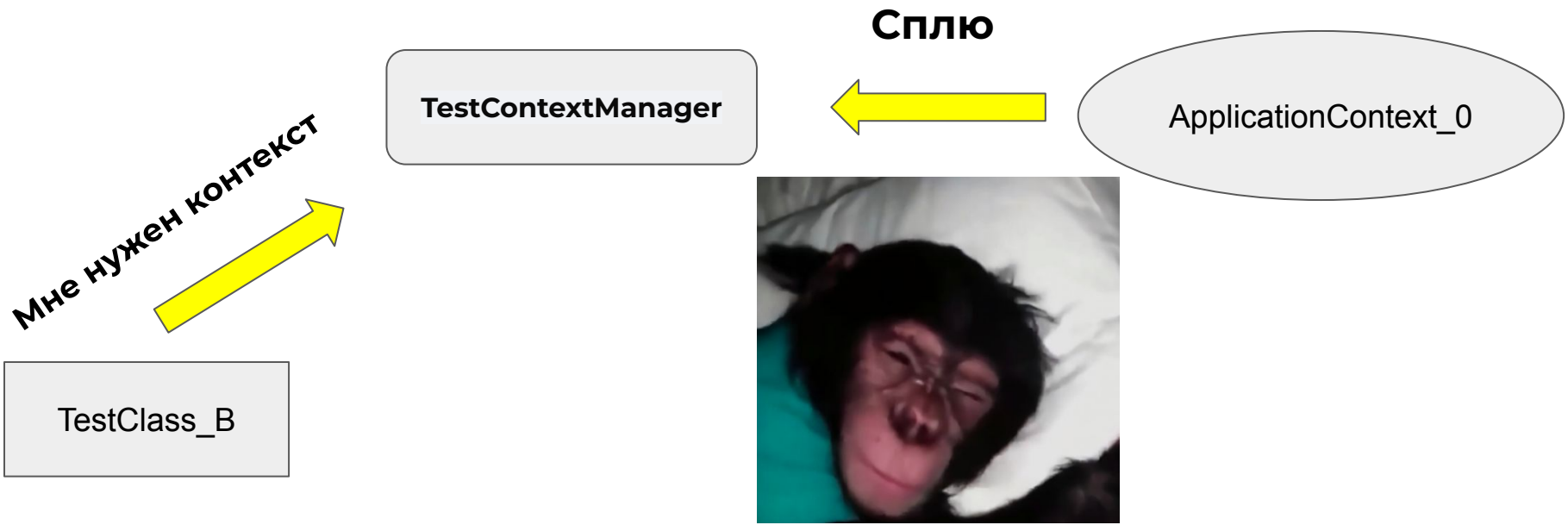
Новое в кешировании ApplicationContext (Spring 7.0.x)



Новое в кешировании ApplicationContext (Spring 7.0.x)

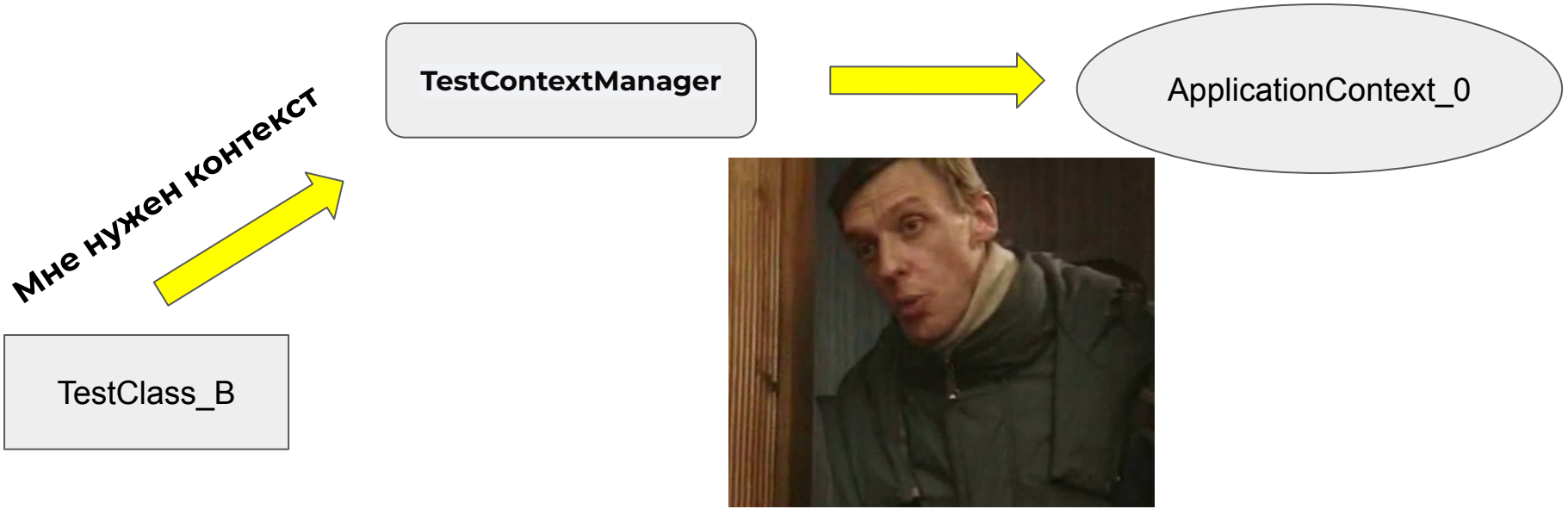


Новое в кешировании ApplicationContext (Spring 7.0.x)



Новое в кешировании ApplicationContext (Spring 7.0.x)

У нас Cache Hit по коням!



Новое в кешировании ApplicationContext (Spring 7.0.x)

```
/**
* Map of context keys to active test classes (i.e., test classes that are actively
* using the corresponding {@link ApplicationContext}).
* @since 7.0
*/
private final Map<MergedContextConfiguration, Set<Class<?>>> contextUsageMap =
      new ConcurrentHashMap<>(32);

/**
* Set of keys for contexts that are currently unused and are therefore
* candidates for pausing on context switch.
* @since 7.0.3
*/
private final Set<MergedContextConfiguration> unusedContexts = new LinkedHashSet<>(4);
```

ApplicationContextRunner

Axelix

Часть №4
Обеспечение кросс-версионной совместимости стартера с Java

Обеспечение кросс-версионной совместимости с Java

Почему об этом стоит задумываться?

Статья "ByteBuffer and the Dreaded NoSuchMethodError"

Статья "ByteBuffer and the Dreaded NoSuchMethodError"

1. Package - Java 11 package
   $ mvn package -Dmaven.compiler.source=1.8 -Dmaven.compiler.target=1.8

Статья "ByteBuffer and the Dreaded NoSuchMethodError"

1. Package - Java 11 package
   $ mvn package -Dmaven.compiler.source=1.8 -Dmaven.compiler.target=1.8

2. Run - Java 8
   $ java target/project-name.jar

Статья "ByteBuffer and the Dreaded NoSuchMethodError"

1. Package - Java 11 package

$ mvn package -Dmaven.compiler.source=1.8 -Dmaven.compiler.target=1.8

2. Run - Java 8

$ java target/project-name.jar

```
java.lang.NoSuchMethodError: java.nio.ByteBuffer.position(I)Ljava/nio/ByteBuffer;
  at io.debezium.connector.postgresql.connection.Lsn.valueOf(Lsn.java:86)
  at io.debezium.connector.postgresql.connection.PostgresConnection.tryParseLsn(PostgresConnection.java:270)
  at
io.debezium.connector.postgresql.connection.PostgresConnection.parseConfirmedFlushLsn(PostgresConnection.java:23
5)
  ...
```

Статья "ByteBuffer and the Dreaded NoSuchMethodError"

```
public abstract class Buffer {
  ...
  public final Buffer position(int newPosition) {
    if (newPosition > limit | newPosition < 0){
      throw createPositionException(newPosition);
    }

    if (mark > newPosition) {
      mark = -1;
    }

    position = newPosition;
    return this;
  }
  ...
}
```

Статья "ByteBuffer and the Dreaded NoSuchMethodError"

```
public abstract class Buffer {
  ...
  public final Buffer position(int newPosition) {
    if (newPosition > limit | newPosition < 0){
      throw createPositionException(newPosition);
    }

    if (mark > newPosition) {
      mark = -1;
    }

    position = newPosition;
    return this;
  }
  ...
}
```

Статья "ByteBuffer and the Dreaded NoSuchMethodError"

```
public abstract class  Buffer {
  ...
  public Buffer position(int newPosition) {
    if (newPosition > limit | newPosition < 0){
      throw createPositionException(newPosition);
    }

    if (mark > newPosition) {
      mark = -1;
    }

    position = newPosition;
    return this;
  }
  ...
}
```

```
public abstract class  ByteBuffer extends Buffer {
  ...

  // -- Covariant return type overrides
  /**
   * {@inheritDoc}
   * @sience 9
   */
  @Override
  public ByteBuffer position(int newPosition) {
   super.position(newPosition);
   return this;
  }

  ...
}
```

Статья "ByteBuffer and the Dreaded NoSuchMethodError"

~~final~~

```
public abstract class  Buffer {
  ...
  public Buffer position(int newPosition) {
    if (newPosition > limit | newPosition < 0){
      throw createPositionException(newPosition);
    }

    if (mark > newPosition) {
      mark = -1;
    }

    position = newPosition;
    return this;
  }
  ...
}
```

```
public abstract class  ByteBuffer extends Buffer {
  ...

  // -- Covariant return type overrides
  /**
   * {@inheritDoc}
   * @sience 9
   */
  @Override
  public ByteBuffer position(int newPosition) {
    super.position(newPosition);
    return this;
  }

  ...
}
```

"ByteBuffer and the Dreaded NoSuchMethodError"

Source Level Compatibility

"ByteBuffer and the Dreaded NoSuchMethodError"

Source Level Compatibility

Binary Level Compatibility

Обеспечение кросс-версионной совместимости с Java

Вы написали стартер и заявляете поддержку Java 17 - 21

Матричная сборка в CI (Matrix Build in CI)

Матричная сборка в CI (Matrix Build in CI)

```
name: CI
on:
  push: branches: [ "main" ]

jobs:
  test:
    name: "Java ${{ matrix.java  }}"
    runs-on: ubuntu-latest
    strategy:
      matrix:
        java [ 17, 18, 19, 20, 21 ]
```

Матричная сборка в CI (Matrix Build in CI)

```
name: CI
on:
  push: branches: [ "main" ]

jobs:
  test:
    name: "Java ${{ matrix.java  }}"
    runs-on: ubuntu-latest
    strategy:
      matrix:
        java [ 17, 18, 19, 20, 21 ]

    steps:
      - uses: actions/checkout@v4

      -  name: Set up JDK ${{ matrix.java }}
         uses: actions/setup-java@v4
         with:
           java-version: ${{ matrix.java }}
           distribution 'liberica'

      -  name: Run tests
         run: ./gradlew test
```

Матричная сборка в CI (Matrix Build in CI)

```
name: CI
on:
  push: branches: [ "main" ]

jobs:
  test:
    name: "${{ matrix.os }} | Java ${{ matrix.java }}"
    runs-on: ${{ matrix.os }}
    strategy:
     matrix:
        os: [ ubuntu-latest, windows-latest, macos-latest ]
        java [ 17, 18, 19, 20, 21 ]
    steps:
      - uses: actions/checkout@v4

      -  name: Set up JDK ${{ matrix.java }}
         uses: actions/setup-java@v4
         with:
           java-version: ${{ matrix.java }}
           distribution 'liberica'

      -  name: Run tests
         run: ./gradlew test
```

Выводы

Выводы

1. Не бойтесь программной регистрации бинов это стандартная практика в экосистеме Spring
   - **SingletonBeanRegistry**
   - **ImportBeanDefinitionRegistrar**
   - **BeanRegistrar**

Выводы

1. Не бойтесь программной регистрации бинов это стандартная практика в экосистеме Spring
   - **SingletonBeanRegistry**
   - **ImportBeanDefinitionRegistrar**
   - **BeanRegistrar**

2. **Общий core/domain** модуль оправдан при длительном жизненном цикле сервисов и огромном 'зоопарке' версий Spring например от версии 1.5 до 4.x в одной экосистеме.
   Иначе используем Отдельные модули/Release Branching

Выводы

1. Не бойтесь программной регистрации бинов это стандартная практика в экосистеме Spring
   - **SingletonBeanRegistry**
   - **ImportBeanDefinitionRegistrar**
   - **BeanRegistrar**

2. **Общий core/domain** модуль оправдан при длительном жизненном цикле сервисов и огромном 'зоопарке' версий Spring например от версии 1.5 до 4.x в одной экосистеме. **Иначе используем Отдельные модули/Release Branching**

3. Автоконфигурации можно и **нужно** тестировать, используем **ApplicaitonContextRunner**

Выводы

1. Не бойтесь программной регистрации бинов это стандартная практика в экосистеме Spring
   - **SingletonBeanRegistry**
   - **ImportBeanDefinitionRegistrar**
   - **BeanRegistrar**

2. **Общий core/domain** модуль оправдан при длительном жизненном цикле сервисов и огромном 'зоопарке' версий Spring например от версии 1.5 до 4.x в одной экосистеме. **Иначе используем Отдельные модули/Release Branching**

3. Автоконфигурации можно и **нужно** тестировать, используем **ApplicaitonContextRunner**

4. Для обеспечения кросс-версионной совместимости Java используем - **Matrix Build in CI**


Спасибо
за внимание!