

Spring Tricks

Little more than a container

A little about me

- Javid Asgarov
 - asgarov1@gmail.com
 - <https://www.linkedin.com/in/asgarovjavid/>
- Java developer @Atos



What is on the menu

1. AOP with annotations
2. Spring Events
3. Implementing our custom AOP via BeanPostProcessors





Taking the cross cutting concerns out with Spring AOP

A method should only do one thing!





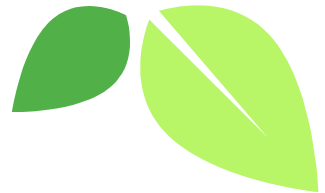
Overview of what is going to happen next

***Joinpoint** is a method we want to target*

1. Choose a **joinpoint**
2. Define a **pointcut** to be applied to that joinpoint
3. Define our **advice**

***Pointcut** is a predicate to match a joinpoints*

***Advice** is the actual code we want to be executed*



Add the Spring AOP dependency

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-aop</artifactId>
  </dependency>
</dependencies>
```

- Dependencies
 - > org.springframework.boot:spring-boot-starter:2.4.5
 - ▼ org.springframework.boot:spring-boot-starter-aop:2.4.5
 - org.springframework.boot:spring-boot-starter:2.4.5 (omitted for duplicate)
 - > org.springframework:spring-aop:5.3.6
 - org.aspectj:aspectjweaver:1.9.6

Create a marker Annotation

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Loggable {
}
```

Define your Aspect class

```
@Aspect
@Component
public class LoggingConfiguration {

    @Pointcut("@annotation(com.asgarov.presentation.aop.annotation.Loggable)")
    public void anyLoggableMethod(){}

    @Before("anyLoggableMethod()")
    public void logMethodCall(JoinPoint jp) {
        String methodName = jp.getSignature().toShortString();
        getLogger(jp).info("BEFORE: " + methodName);
    }

    @AfterReturning("anyLoggableMethod()")
    public void logMethodReturn(JoinPoint jp) {
        String methodName = jp.getSignature().toShortString();
        getLogger(jp).info("AFTER: " + methodName);
    }

    private Logger getLogger(JoinPoint jp) {
        return LoggerFactory.getLogger(jp.getTarget().getClass().getSimpleName());
    }
}
```

Aspect is the class where we define out pointcuts and advices

Advices in Spring:

- @Before
- @After
- @AfterReturning
- @AfterThrowing
- @Around

Create a service to use it

```
@Service
public class ImportantService {
    @Loggable
    public void veryImportantMethod() {
        System.out.println("===I AM BEING EXECUTED===");
    }
}
```

Enjoy your easy to log methods

```
@SpringBootApplication
public class PresentationApplication {

    @Autowired
    private ImportantService importantService;

    public static void main(String[] args) {
        SpringApplication.run(PresentationApplication.class, args);
    }

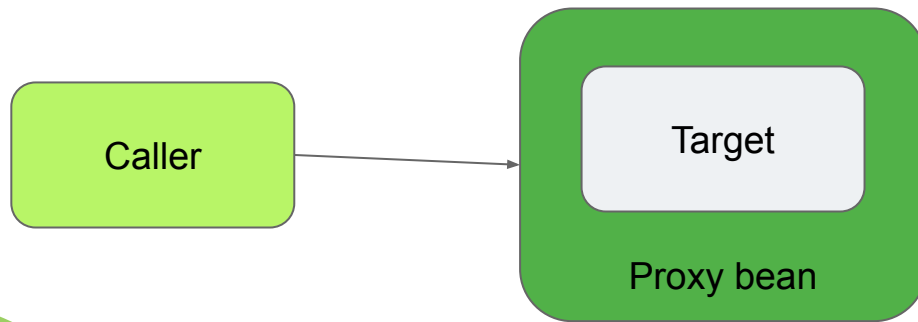
    @PostConstruct
    public void doImportantStuff() {
        importantService.veryImportantMethod();
    }
}
```

PresentationApplication x

```
↑ :: Spring Boot :: (v2.4.5)
↓
2021-05-06 21:41:45.015 INFO 8608 --- [main] c.a.p.PresentationApplication : Starting PresentationApplication using Java 11.0.
2021-05-06 21:41:45.020 INFO 8608 --- [main] c.a.p.PresentationApplication : No active profile set, falling back to default pr
2021-05-06 21:41:45.804 INFO 8608 --- [main] ImportantService : BEFORE: ImportantService.veryImportantMethod()
===I AM BEING EXECUTED===
2021-05-06 21:41:45.815 INFO 8608 --- [main] ImportantService : AFTER: ImportantService.veryImportantMethod()
```

Important to know:

Spring AOP works via proxies



Proxies are created via interfaces or subclassing.
This has 2 consequences:

1. AOP won't work with final classes or methods
2. In-class invocations won't work



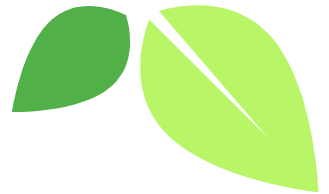
Spring Events





Overview

1. Create an event class
2. Define an event publisher
3. Define (possibly many) event listeners





We redefine Loggable to include value

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Loggable {
    String value() default "";
}
```

Create a new Aspect this time binding the annotation as a parameter

```
@Aspect
@Component
public class EventLoggingConfiguration {

    @Before("@annotation(loggable)")
    public void logMethodCall(JoinPoint jp, Loggable loggable) { getLogger(jp).info(loggable.value()); }

    private Logger getLogger(JoinPoint jp) {
        return LoggerFactory.getLogger(jp.getTarget().getClass().getSimpleName());
    }
}
```

Create an example Event for receiving payment

```
public class PaymentReceivedEvent extends ApplicationEvent {  
  
    private final String message;  
  
    public PaymentReceivedEvent(Object source, String message) {  
        super(source);  
        this.message = message;  
    }  
  
    public String getMessage() { return message; }  
}
```


Define an event publisher for it

```
@Component
public class PaymentReceivedPublisher {
    @Autowired
    private ApplicationEventPublisher publisher;

    @Loggable("Payment Received!!!")
    public void publishEvent(String message) {
        var event = new PaymentReceivedEvent(source: this, message);
        publisher.publishEvent(event);
    }
}
```

Last but not least an event listener:

```
@Component
public class MyEventListener implements ApplicationListener<PaymentReceivedEvent> {

    @Loggable("Got an event!!!")
    @Override
    public void onApplicationEvent(PaymentReceivedEvent event) {
        // do sth with the event
        System.out.println(event.getMessage());
    }
}
```

Run the code

```
@SpringBootApplication
public class EventApplication {

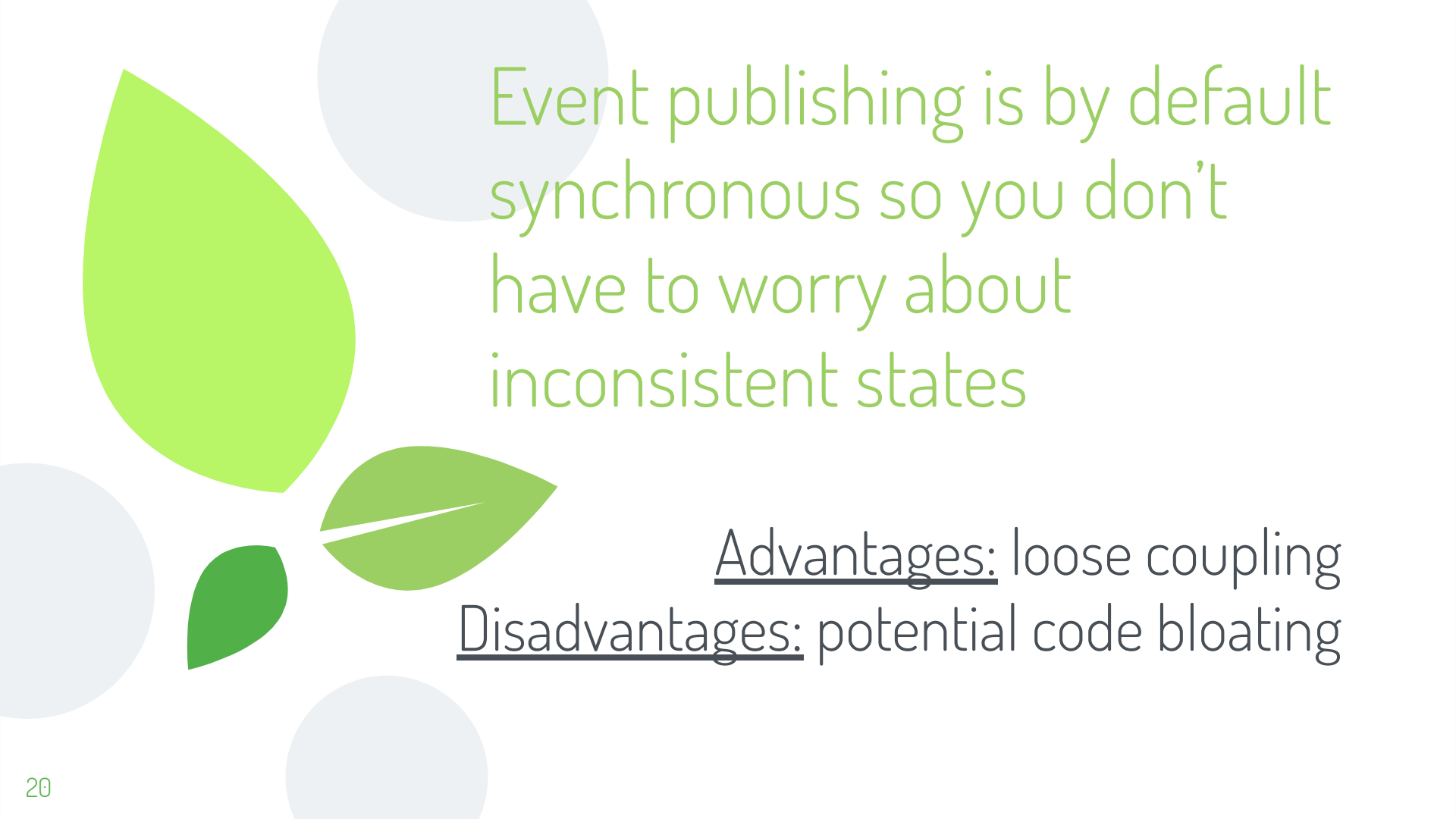
    @Autowired
    private PaymentReceivedPublisher publisher;

    @PostConstruct
    public void run() { publisher.publishEvent("Received payment from XYZ"); }

    public static void main(String[] args) { SpringApplication.run(EventApplication.class, args); }
}
```

EventApplication x

```
2021-05-13 10:06:44.819 INFO 12808 --- [main] com.asgarov.event.EventApplication : Starting EventApplication
2021-05-13 10:06:44.823 INFO 12808 --- [main] com.asgarov.event.EventApplication : No active profile set
2021-05-13 10:06:45.728 INFO 12808 --- [main] PaymentReceivedPublisher : Payment Received!!!
2021-05-13 10:06:45.751 INFO 12808 --- [main] MyEventListener : Got an event!!!
Received payment from XYZ
2021-05-13 10:06:45.845 INFO 12808 --- [main] com.asgarov.event.EventApplication : Started EventApplication
```



Event publishing is by default
synchronous so you don't
have to worry about
inconsistent states

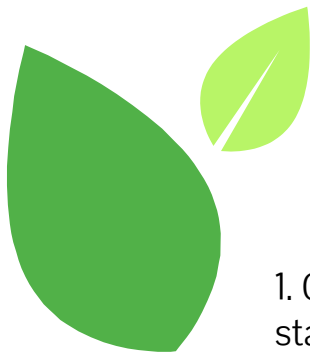
Advantages: loose coupling

Disadvantages: potential code bloating



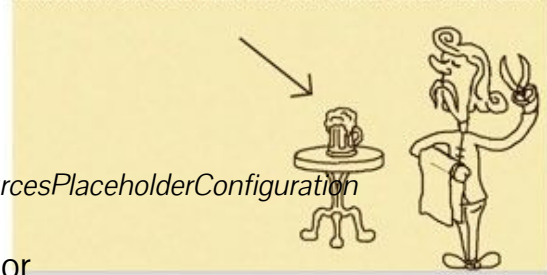
PostProcessors





Spring Bean Lifecycle

Rhababerbarabarbarbarbarenbartbarbiebier



1. Container has started



2. Bean definitions are loaded



3. BeanFactoryPostProcessor processes bean definitions
e.g. PropertySourcesPlaceholderConfiguration



4. Instantiation



5. Populating Properties



6. BeanNameAware's setName()



7. BeanFactoryAware's setBeanFactory()



8. ApplicationContextAware's setApplicationContext()



9. Pre-Initialization PostProcessor



10. Initializers (@PostConstruct, etc..)



11. Post-Initialization PostProcessor



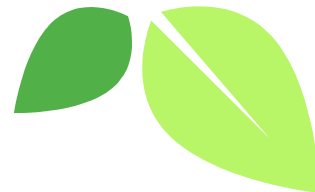
12. Bean is ready for use!



13. Destroy methods are called (@PreDestroy)



14. RIP Bean





Lets implement our own AOP via BeanPostProcessors

create new annotation:

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface PostProcessable {
    String value();
}
```

Create some simple class

```
@Component
public class Receipt {

    @PostProcessable("AOP is awesome")
    public int getAmount() { return new Random().nextInt( bound: 100); }
}
```


Define BeanPostProcessor

```
15  @Component
16  public class MyBeanPostProcessor implements BeanPostProcessor {
17      @Override
18  * public Object postProcessBeforeInitialization(Object bean, String beanName) throws BeansException {
19      if (hasAnnotatedMethods(bean, PostProcessable.class)) {
20          var enhancer = new Enhancer();
21          enhancer.setSuperclass(bean.getClass());
22          enhancer.setCallback(createCustomCallback());
23          return enhancer.create();
24      }
25      return bean;
26  }
27
28  @ private boolean hasAnnotatedMethods(Object bean, Class<? extends Annotation> clazz) {
29      return Arrays.stream(bean.getClass().getMethods())
30          .map(method -> method.getAnnotation(clazz))
31          .anyMatch(Objects::nonNull);
32  }
33
34  public MethodInterceptor createCustomCallback() {
35      return (bean, method, objects, methodProxy) -> {
36          var annotation : Optional<PostProcessable> = Optional.ofNullable(method.getAnnotation(PostProcessable.class));
37          if (annotation.isPresent()) {
38              LoggerFactory.getLogger(method.getDeclaringClass()).info("BEFORE: " + annotation.get().value());
39              return methodProxy.invokeSuper(bean, objects);
40          }
41          return methodProxy.invokeSuper(bean, objects);
42      };
43  }
44  }
45  }
```

Run it:

```
@SpringBootApplication
public class PostProcessorApplication {

    @Autowired
    private Receipt receipt;

    public static void main(String[] args) { SpringApplication.run(PostProcessorApplication.class, args); }

    @PostConstruct
    public void someMethod() {
        System.out.println(receipt.getAmount());
    }
}
```

PostProcessorApplication ×

```
2021-05-08 12:42:24.037 INFO 24536 --- [main] c.a.p.PostProcessorApplication : Starting PostProcessorAp
2021-05-08 12:42:24.041 INFO 24536 --- [main] c.a.p.PostProcessorApplication : No active profile set, f
2021-05-08 12:42:24.828 INFO 24536 --- [main] com.asgarov.postprocessor.Receipt : BEFORE: AOP is awesome
99
2021-05-08 12:42:24.899 INFO 24536 --- [main] c.a.p.PostProcessorApplication : Started PostProcessorApp
```



slides available at:
bit.ly/3hce34E

**Thanks for
listening!**

code available at:
github.com/asgarov1/springTricks

