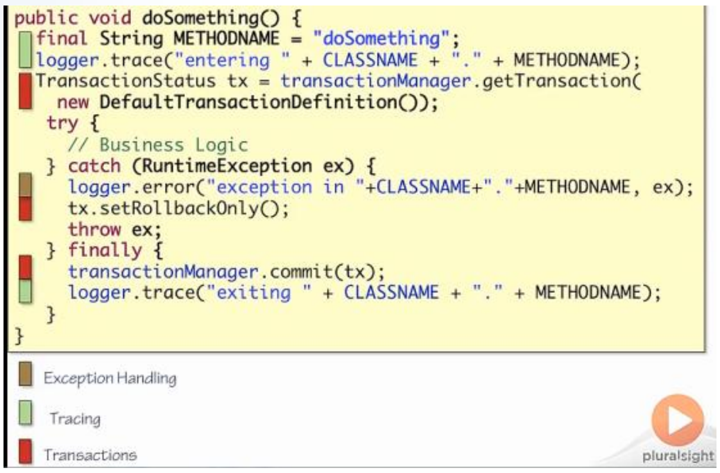
Spring AOP

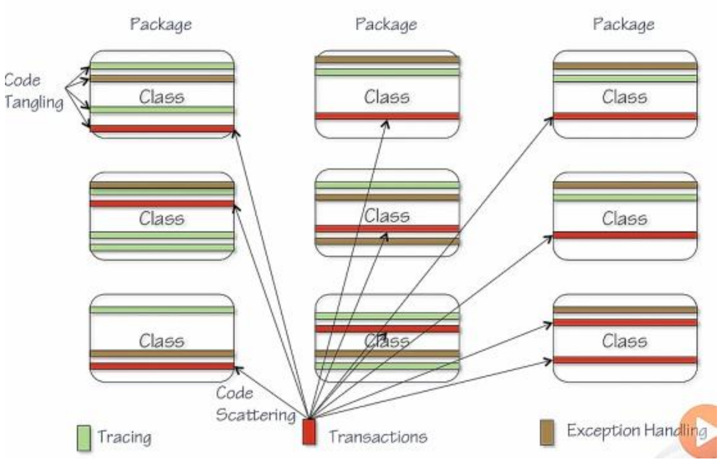
AOP Introduction

- Why AOP?
    - AOP is very important foundation of Spring and also of Java EE.
    - AOP is used to implement enterprise features like transactions, security.
    - AOP helps to where to start transactions and which methods to secure - therefore configurable middleware.
    - Simplify Code
- Simplify code using AOP
    - In a typical method without AOP we would need lots of boiler code for tracing and transaction handling, for instance in below method. The Business logic is still not present:

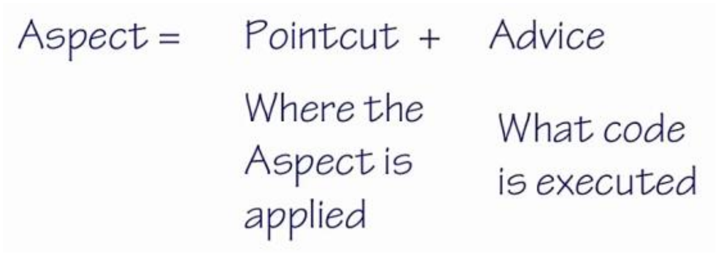


```
public void doSomething() {
  final String METHODNAME = "doSomething";
  logger.trace("entering " + CLASSNAME + "." + METHODNAME);
  TransactionStatus tx = transactionManager.getTransaction(
    new DefaultTransactionDefinition());
   try {
     // Business Logic
   } catch (RuntimeException ex) {
     logger.error("exception in "+CLASSNAME+"."+METHODNAME, ex);
     tx.setRollbackOnly();
     throw ex;
   } finally {
     transactionManager.commit(tx);
     logger.trace("exiting " + CLASSNAME + "." + METHODNAME);
   }
}
```

    - So when you write an aspect you remove all the boiler plate code and convert them to aspect; in this case we would have a Tracing Aspect, Exception Aspect, Transaction Aspect.
- How does AOP Works:
    - Suppose you have a system with lots of classes as shown and you are NOT using AOP than your exception handling code and transactions are spread:

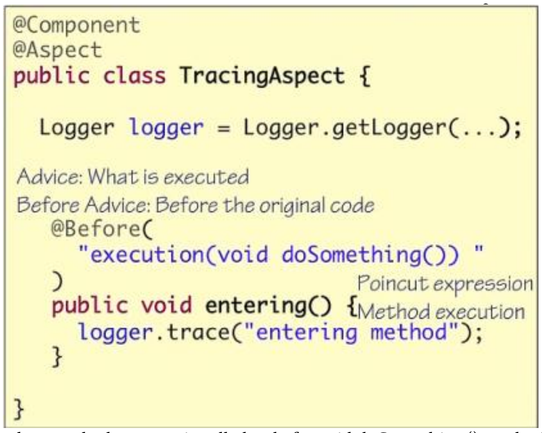


    - If you don't use AOP than there are problems like code tangling, code scattering. So what AOP does is that it will take all Tracing code and put it in a Tracing Aspect, and it will take all transactions and put it in Transactions Aspect and finally all exception handling is put in exception Aspect. This way there is a clear separation between business logic and additional aspect.
- Cross-Cutting Concerns:
    - Tracing, Exception Handling and Transactions are cross-cutting concerns.
    - A lot of classes must implement them, and they can't be implemented in one place if you use object oriented programming only.
    - Aspect oriented programming allows centralized implementation of cross-cutting concerns.
- Use Spring AOP or AspectJ to weave aspects into the applications.
- First Aspect Example:
    - What is an Aspect?
        - Aspect implements cross-cutting concern in one centralized space of the code base, otherwise cross-cutting concerns are shattered throughout the code.
        - Ultimate goal is to get rid of boiler plate code and focus on business logic.
        - Aspect = Pointcut + Advice



Aspect =	Pointcut +	Advice
	Where the Aspect is applied	What code is executed

    - Example Tracing:
        - Mark the class as @Component to indicate that it is a spring bean and @Aspect to indicate it is an Aspect. @Before is used to indicate that the method should be executed before the actual method, and the pointcut expression tells which method to apply for.



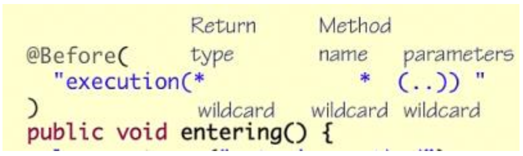
```
@Component
@Aspect
public class TracingAspect {

  Logger logger = Logger.getLogger(...);

    @Before(
      "execution(void doSomething()) "
    )
    public void entering() {
      logger.trace("entering method");
    }

}
```

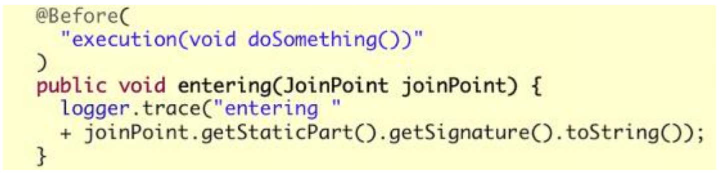
- In the above code the aspect is called only for void doSomething() method; which isn't very helpful. To call it for all metho ds use replace them with wild card characters:



```
@Before(
  "execution(*           *  (..)) "
)
public void entering() {
```

JoinPoint

- JoinPoints are use to findout which method was called, in the above implementation tracing indicates only that a method was called and doesn't tell which method was invoked to do this use JoinPoint. Join Point are Point in the control flow of a program.
- Advices can be presented with information about the join point. Here we adding JoinPoint to the entering method, this parameter will be filled by spring automatically for us:



```
@Before(
  "execution(void doSomething())"
)
public void entering(JoinPoint joinPoint) {
  logger.trace("entering "
  + joinPoint.getStaticPart().getSignature().toString());
}
```

Enable Aspects in Spring XML Configuration

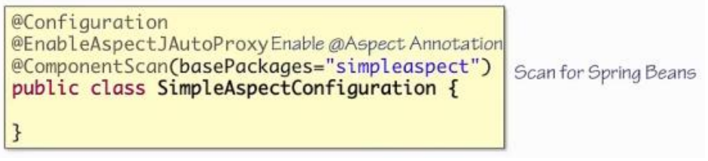
- To enable aspects in spring configuration xml file use `<context:component-scan base-package=""/>` this will turn any class that has spring component annotation marked into a spring bean. Our Aspects must be Spring beans. To enable @AspectJ support with XML based configuration use the `aop:aspectj-autoproxy` element:

```
<aop:aspectj-autoproxy />
<context:component-scan base-package="com.mujahed.aop.simpleaspect" />
```

- The other annotation will change the @Aspect marked class to Aspect.

Enable Aspects in Java Configuration

- To use Spring Java Configuration use should use `@ComponentScan(basePackages="")` this will scan for spring beans, you also need to add `@EnableAspectJAutoProxy` this will enable `@Aspect` annotation.



```
@Configuration
@EnableAspectJAutoProxy
@ComponentScan(basePackages="simpleaspect")
public class SimpleAspectConfiguration {

}
```

- asd