**Summary – Spring AOP**

Aspect oriented programming provides an elegant solution for using cross cutting functionality. It does that by creating aspects which are special objects that can track activities in the main flow and add a cross cutting functionality besides it.

Special annotations

* @Aspect (annotates a class) – defines the aspect class
* @Before(<expression>) (annotates a method) – defines a method that will run before another method in a certain main flow.
* @After(<expression>) (annotates a method) – defines a method that will run after another method in a certain main flow.
* @AfterReturning(<expression>) (annotates a method) – defines a method that will run after another method in a certain main flow will end with no exceptions.
* @AfterThrowing(<expression>) (annotates a method) – defines a method that will run after another method in a certain main flow will throw an exception.
* @Around(<expression>) (annotates a method) – defines a method that will run before and after another method in a certain main flow.
* @Pointcut(<expression>) (annotates a method) – defines a void empty method that will represent a shortcut for above annotation to use. All the above annotations have the following types of expressions:
  + execution(public String <some method>()) – execution of some method
  + execution(public String <some object>.<some method>()) – execution of some method in a specific object
  + execution(\* <some object>.\*(..)) - execution of some method that the first wildcard matches any return value, the second matches any method name and the (..) pattern matches any number of parameters (zero or more).
  + within(<some object>) – execution of all methods in some object
  + within(<some package>..\*) – execution of all methods in a certain package
  + args(<some object name>) – (the method that’s annotated with this expression should get as a parameter the object and the object name). for example:

@Before("args(something)")

**public** **void** AdviceForStringArgumentsMethods(String something){}

execution of all methods that gets the same object as the parameter in the method.

a method in an Aspect that gets JoinPoint as a parameter means that the main flow method is populating the JoinPoint object. That means that we can manipulate with the method inside the Aspect method. When calling joinPoint.getTarget() we are getting the object that the main flow method is located in.