* Spring Aop
  + What is AOP
    - Aspect Oriented Programming
    - Aspects
    - Join Points
    - Pointcuts
    - Advice
  + What is an aspect
    - Modularization of a Cross Cutting Concern ( it will solve one of these problems )
  + How does it relate to a cross cutting concern
    - It is a way for us to address CCC in java.
    - We only have to write code once and it will be applied in many locations
  + How?
    - By Building Proxy Classes ( a new class with aspect code that wraps the original class with its normal code ) behind the scenes
    - Spring can automatically inject the new Proxy into anywhere that needed the bean as a dependency
  + What are join point and point cuts
    - Join Point is any potential location you could apply code to
      * Mostly this is methods
      * But it could be constructors or fields as well
    - Point Cut is an expression that will match particular join points in your program.
  + What is the syntax for a point cut
    - <https://docs.jboss.org/jbossaop/docs/2.0.0.GA/docs/aspect-framework/reference/en/html/pointcuts.html>
    - Method
    - execution( FullyQualifiedClassPathForTheReturnType FullyQualifiedPathOrPAckagePath MethodName( Method Params) )
    - execution( com.io.spring.ResponseEntity com.revature.controllers.. \*(..))
      * Return type is ResponseEntity Object
      * It is any class in com.revature.controllers ( this is what the .. does)
      * \* matches any number of characters - all methods in the class
      * (..) for params matches every combination of parameters
    - Writing PointCuts is tricky and it is easy to target more or less than you were trying to
    - Its very hard to debug your pointcut expression
  + What is advice
    - This is the actual code to run to solve the CCC
    - Most of the time we have one piece of advice per aspect, but you could have more
  + Different kinds of advice
    - Use annotations to denote the kinds of advice
    - @Before - runs before the target JoinPOint
    - @After - runs after the target JoinPoint completes
    - @AfterReturning - runs only after the JoinPoint Returns a value
    - @AfterThrowing - runs only after the JoinPoint throws an exception
    - @Around - Runs before and after the JoinPoint
      * We need to pass in a special object called the ProceedingJoinPoint
      * We use that object to call the JoinPoints method
* Annotations
  + How to make an annotation in java
    - It is really hacky
    - We make a public @interface give it a name
    - That interface has methods, and the methods we declare act as getters/setters for fields of that method name.
  + What do they most look like
    - Interfaces
  + Our custom has 2 important annotations
    - @Retention for how long the javac should keep that annotation around for
    - @Target for what it is allowed to be applied to
* Controller Advice
  + How to handle an error and still send a response
    - By default, we just send the error object as the response
    - This is bad, because giving stack traces to end users can be dangerous
    - Is we build a special collection of ControllerAdvice which are methods that will run if a certain throwable occurs
    - Then we can send an appropriate response
  + How is this related to Aop
    - These pieces of Controller Advice spring builds aspects from and wraps up your controllers using these methods and afterthrowing advice
  + What kind of exceptions does spring always throw?
    - Spring always uses RunTimeExceptions, because you don’t have to try catch RunTimeExceptions when writing code.
    - This way all exceptions will get caught by the aspect layer that wraps the controllers