**ByteBuddy Final Project – Annotation-based AOP framework using ByteBuddy**

Given the Cs124AOP project, you will have several java files that will require fill up. The basic structure and annotations are laid out, though you may opt to add to it as you see fit.

* Create an AspectManager class that loads classes marked with @Aspect via FastClasspathScanner in the aspects package. Each of the these Aspect files contains
  + One @Pointcut – specifies a list of Regex that represents the method name patterns for this Aspect, assume only one is provided
  + One @Targets - specifies a list of all the class name patterns that will be affected by this Aspect, assume only one is provided
    - NOTE: A given class can be the target of multiple aspects
  + One of the following advices: @Before, @After
    - @Before indicates code that will run before the original code
    - @After indicates code that will run after the original code,
    - assume only one is provided
* Create a ProxyMaker Factory class to create proxies that implements AOP on a specified Target class. The ProxyMaker method will be responsible for the creation of the object instance based on the following logic:
  + If the target class needs any AOP (i.e. is a Target for any of the aspects scanned) then return a proxy of the class with an interceptor (see below)
    - AspectManager should provide a boolean needsProxy(Class c) method to determine if a given Class needs a proxy or not.
  + Otherwise, just give a new instance of the unproxied class
* Create an AspectInterceptor class that will process all the @Before code that are applicable for a given method (if any), run the original code, then process all the @After code that are applicable for a given method (if any)
* AspectManager should provide a processBefore(Method method, Object[] args) and processAfter(Method method, Object[] args)
  + How you do this will be up to you
  + HINTS: you can either scan each time a method call occurs to detect a match or somehow remember so you won’t need to do a full scan each proxy call

Magis:

* Implement any applicable Desgin Patterns you see fit
* Allow the @Pointcut to specify a parameter list and return type and search based on these in addition to the name
  + NOTE: up to you how you wish to implement this
* Allow for processing of an @Around advice in addition to the required @Before and @After
  + Add an Object processAround(Method method, Object[] args) to AspectManager.java
  + NOTE: one of the design patterns that will be presented will give a potential answer
* Allow for the handling of stateful Aspects, i.e. Aspects that remember values from previous invocation of a SPECFIC proxy instance. Example, an Aspect that remembers how many times a method call was called on a specific object.