**CST-361 Design Patterns in Java**

**Student** **Name**: Michael Weaver

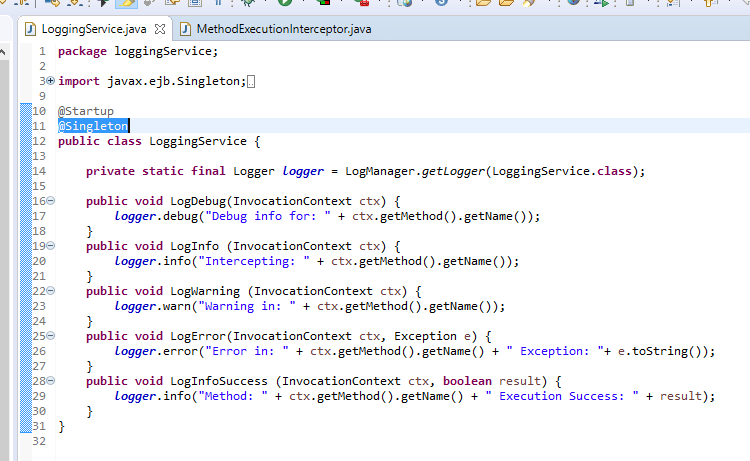
**Project** **Title**: ACTIVITY 7 C: INTEGRATION LAYER PATTERNS

**Date**: 02/10/2019

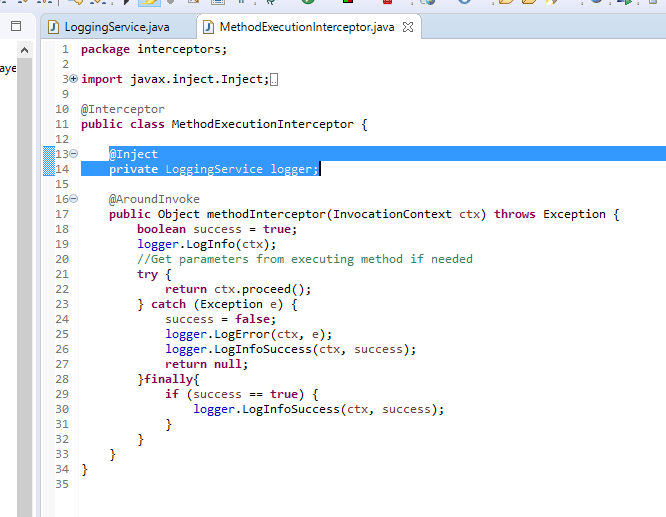
**GitHub URL**: <https://github.com/battousairurik/CST361_Design_Patterns_In_Java>

SCREENSHOTS

SINGLETON LOGGER



SINGLETON INJECTION



DESIGN PATTERN

For this activity the Singleton pattern was used. Put simply, The singleton pattern is a design pattern that restricts the instantiation of a class to one object. For this assignment it would be possible to implement the Object Pool design pattern, which is a version of the singleton but applied to multiple objects. With that design pattern I could have created a logger, a single Album DAO, a single User DAO, and placed them all inside the Object Pool. Effectively any object that I would only typically need one instance of, I could treat it as a singleton and save it to the object pool.

DECORATOR DESIGN PATTERN

The Decorator pattern is described as follows:

*“The decorator pattern is one of the structural patterns described in the book of GoF . Its purpose is to wrap a target object so that you can dynamically add new responsibilities at run time.”*

Effectively, each decorator dynamically adds to or changes existing properties of a base object. A few points about the decorator pattern are:

* *To enable the container to identify this class as a decorator, you must annotate it with @Decorator. The delegate injection point (the instance that will be decorated) is annotated with @Delegate and must be an injected field, an initializer method parameter, or a bean constructor method parameter.*
* *Finally, you must declare the decorator in bean.xml .*

All classes that implement the same interface as the one implemented by the delegate injection point are decorated, but only if those decorators are declared in bean.xml . This has two major implications:

* *Decorators can be enabled and disabled at deployment time by editing the bean.xml file.*
* *A decorator is automatically applied to classes that implement the same interface.*

The decorator pattern is used mainly when you need to adjust an existing object dynamically at runtime.

For this project, the decorator could be used to add songs to the album before it is saved to the database. This way there would always be a single object saved and not an object with a list of objects tied to it.

DECORATOR UML

