Advanced Object-Oriented Programming (PROG 2200)

# Assignment 4 – Interfaces, Dependency Injection and IoC Containers

For this assignment, we will be revisiting your Assignment 3 work, which dealt with the Asynchronous Chat Application.

# Step 1. Refactor your application so that the Logging functionality is Loosely Coupled

Using the principle of Dependency Injection that we explored in class, refactor your code by implementing decoupling logging library and any object that it entails, you will rework your existing application by having your application implement an interface called **ILoggingService.**

1. The ILoggingService interface should define the following method

void Log(string message)

This means that you will likely have to rework your existing logging library a bit so that it adheres to the contract that is enforced by the **ILoggingService** interface.

1. The **ILoggingService** interface should live in its own class library project called ***Interfaces****.* (Don't do this...simply put the interface in your existing logger project)
2. Use **Constructor Injection** in order to introduce your logging functionality into your Chat library.

# Step 2. Implement an IoC container in your Form Application’s Main() Method

1. Implement your IoC container using the **Microsoft Unity Container**

( https://msdn.microsoft.com/en-us/library/ff647202.aspx ) as was done in the class demonstration.

1. Establish the mapping of your **ILoggingService** interface to resolve to your concrete Logging Class. Once complete, there should be no instantiating of your Logger class anywhere in your entire solution. In other words there will be no use of the new

Advanced Object-Oriented Programming (PROG 2200)

keyword in order to instantiate your Logger class.

1. In addition, research and implement one for the following three alternative IoC containers to do the exact same thing that you did with Unity:
   * Castle Windsor ( http://www.castleproject.org/projects/windsor/ )
   * StructureMap ( http://structuremap.github.io/documentation/ )
   * Ninject ( http://www.ninject.org/ )

Switch between IoC implementations by commenting and uncommenting code.

# Step 3. Implement a second logging class using an established .NET logging framework.

Your first logging class wrote to a simple text file. Implement a second logging mechanism by introducing a new project in your solution called [Your Name]\_Logger (Example:

MichaelCaines\_Logger. Don't do this...simply put your new logger class in your existing logger project.

You must choose to implement a logging framework from the list supplied at

http://www.dotnetlogging.com/. Chances are that most if not all of these frameworks are gettable from NuGet Package Manager. How you choose to use the logging framework to do your logging is up to you. Your choices will depend on the logging framework you choose.

# Step 4. Implement a third logging functionality using one of your classmate’s logging assemblies.

By building your alternative logging functionality in its own project, it will be housed in its own assembly (dll). And because yours and your classmates’ assemblies implement the same interface (ILoggingService), you should be able to plug in a classmates logging functionality and have it work in your application with very little configuration.

Here you’ll see the true power of Dependency Injection using Interfaces by noting how little work you’ll have to do in order to introduce alternative logging into your application that you didn’t write.

# Step 4a. Here are the in-class steps to test your classmate's logging assembly:

## Remove the reference to your own Logger Library in your Chat Application project. If you try and run the application now, you should see errors like: *"The type or namespace name 'ILoggingService' could not be found (are you missing a using directive or an assembly reference?)"*

## Copy the contents of another student's <Logger Library>/bin/Debug folder to your desktop. Make sure the project built successfully and all the files (\*.dll, \*.xml, etc.) are included. You do not need to see the other student's code to accomplish this.

## In your Chat Application project, select "Add Reference..."

## In the dialog that appears, select "Assemblies" from the left-hand pane. Click on "Browse..." and go to the folder on your desktop.

## Browse to the Assembly with your friend's project name. For example, if their project was called "Logger", you would select Logger.dll.

## Go to where your IoC container instantiates the ILoggingService. It should compile ILoggingService properly now. However, your own instance, (call it RonanConsoleLogger for example), no longer compiles.

## Change the instance of ILoggingService to be whatever your friend's class is. Make sure you add the correct "using" directive to your code. If your friend's instance of ILoggingService is called "Logger" then you would change it to that here. You can comment out your old code to accommodate this.

## If you have a separate Chat library project that references ILoggingService, also change its reference from your own project to your friend's Assembly.

## You should now be able to run the application with the alternative project's code.