|  |  |
| --- | --- |
|  | LAB | Dependency Injection with Unity |
|  | WORKSHOP | Design Patterns |
|  | Crystal Tenn  crtenn@microsoft.com |

# SETUP LAB ENVRIOMENT

## Prerequisites

To perform the tasks in this lab you need following:

* Visual Studio 2015 or 2017, any version

## Objectives

In this lab you will complete following tasks:

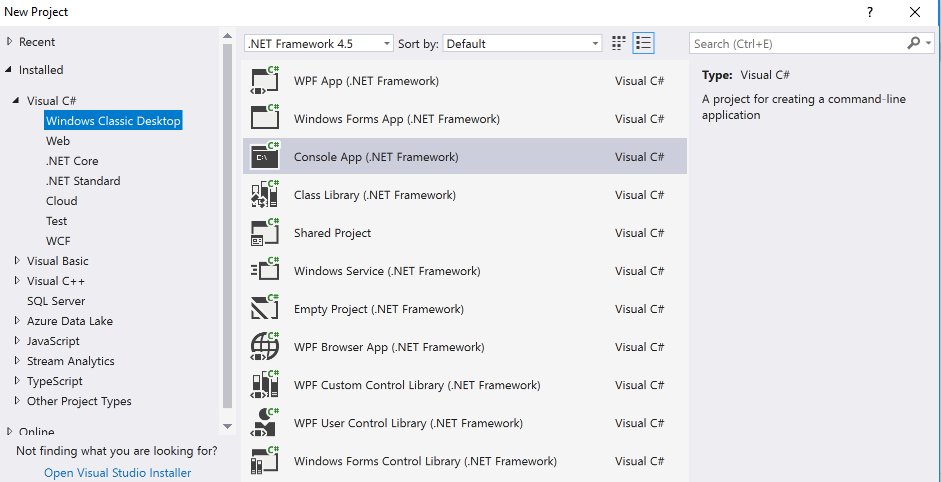
* Create a new console application utilizing Unity.

### Estimated Completion Time: 60 minutes.

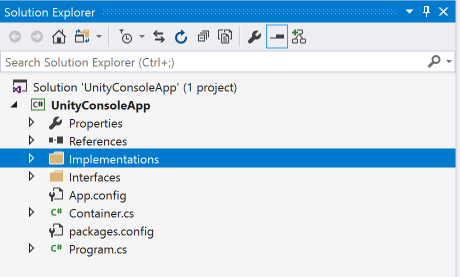
## Task: Create a Console Application

In this task you will create a new Console application.

1. Open Visual Studio.
2. Choose New Project. On the left select Windows Classic Desktop. Choose a Console App.



1. Name it UnityConsoleApp.
2. Add a folder to the project called Implementations.
3. Add a folder called Interfaces.
4. Add a new class under the main project called Container.cs.
5. Your project should look like this.



**Note**: We will start by creating 3 interfaces then implementing them. Then we will work on setting up a basic Unity container. We will do a mixture of classes to implement, some will have 1:1 ratio with an interface, some will have no interface, and some interfaces will have multiple classes implementing them. Unity is downloaded off NuGet.

1. Add an interface called IAdapter with the following code:

public interface IAdapter

{

string Voltage();

}

1. Add an interface called IRemote with the following code:

public interface IRemote

{

bool IsRefurb();

string ModelType();

string SerialNumber();

string Manufacturer();

}

1. Add an interface called ITelevision with the following code:

public interface ITelevision

{

IAdapter Adapter { get; set; }

IRemote Remote { get; set; }

string Manufacturer { get; set; }

string ModelNumber { get; set; }

void TurnOn();

void TurnOff();

}

1. Add a class called Adapter that implements IAdapter and has the following code:

public class Adapter : IAdapter

{

public string Voltage()

{

return "120V";

}

}

1. Add a class called Remote that implements IRemote and has the following code:

public class Remote : IRemote

{

public bool IsRefurb()

{

return false;

}

public string ModelType()

{

return "The Super Remote";

}

public string SerialNumber()

{

return "REMOTE1234654";

}

public string Manufacturer()

{

return "Samsung";

}

}

1. Add a class called Television that implements ITelevision and has the following code:

public class Television : ITelevision

{

public IAdapter Adapter { get; set; }

public IRemote Remote { get; set; }

public string Manufacturer { get; set; }

public string ModelNumber { get; set; }

public Television(IAdapter adapter, IRemote remote, string manufacturer, string modelNumber)

{

Adapter = adapter;

Remote = remote;

Manufacturer = manufacturer;

ModelNumber = modelNumber;

}

public void TurnOn()

{

Console.WriteLine("Bzzt. TV just turned on!");

}

public void TurnOff()

{

Console.WriteLine("Aw. TV just turned off!");

}

}

1. Add a class called TVStand that implements nothing and has the following code:

public class TVStand

{

public int MaxInchesOfTV { get; set; }

public TVStand(int maxInchesOfTv)

{

MaxInchesOfTV = maxInchesOfTv;

}

public int MaxInchesOfTVOnStand()

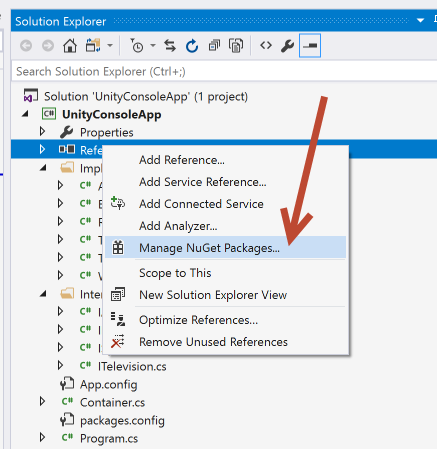
{

return MaxInchesOfTV;

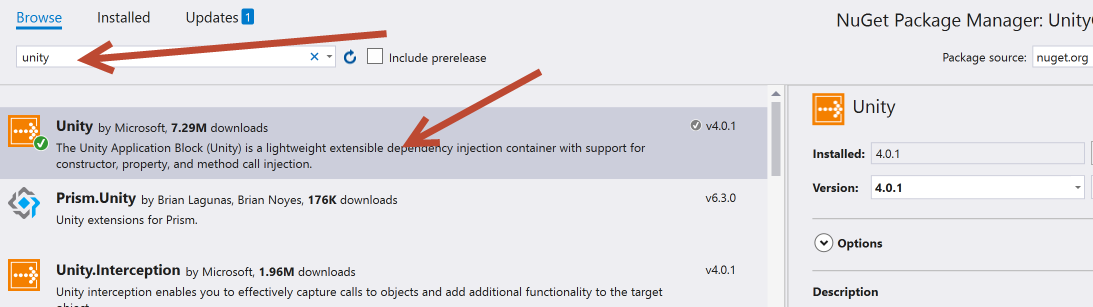
}

}

1. Right click on References for your project. Hit Manage NuGet packages.



1. Under Browse, search for Unity. Download Unity off NuGet.



1. Go to the Container.cs class. Add the following code to the class and the whole file should look like this:

using Microsoft.Practices.Unity;

namespace UnityConsoleApp

{

public class Container

{

public static void RegisterElements(IUnityContainer container)

{

TVStand tvStand = new TVStand(70);

container.RegisterInstance(tvStand);

container.RegisterType<IAdapter, Adapter>();

container.RegisterType<IRemote, Remote>();

var adapterType = typeof(IAdapter);

var remoteType = typeof(IRemote);

container.RegisterType<ITelevision, Television>(new InjectionConstructor(adapterType, remoteType, typeof(string), typeof(string)));

}

}

}

**Note**: This is an explanation of the Container.cs class. Please read the below and correlate/understand how it works with the code above that is found in your Container.cs class. They are highlighted above to make it easier to see.

* RegisterInstance – this is used to pass an instance of our Unity container to the RegisterElements method. Our TV stand did not have an interface associated with it so this one is treated differently.
* RegisterType – This is used to return a type of Battery any time the client asks us for type object IBattery. It maps your implemented class to your interface. Note later in this lab we can show how to automate finding these.
* InjectionConstructor – This will tell Unity the parameters Radio’s constructor requires. Remember Television was the only class that had a constructor that took in parameters. Note that the adapter and remote are variables set above as typeof(IAdapter) and typeof(IRemote) and that the two string parameters and just specified as the specific type.

1. Go to your Program.cs and add the following code:

static void Main(string[] args)

{

IUnityContainer container = new UnityContainer();

Container.RegisterElements(container);

IAdapter adapter = container.Resolve<IAdapter>();

Console.WriteLine(adapter.Voltage());

TVStand tvStand = container.Resolve<TVStand>();

Console.WriteLine(tvStand.MaxInchesOfTVOnStand());

IRemote remote = container.Resolve<IRemote>();

ITelevision television = container.Resolve<ITelevision>

(new ParameterOverride("adapter", adapter),

new ParameterOverride("remote", remote),

new ParameterOverride("manufacturer", "Visio"),

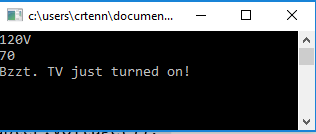
new ParameterOverride("modelNumber", "321asdf"));

television.TurnOn();

Console.ReadLine();

}

1. Run your code now and you should see this in your console:



1. Feel free to drop any breakpoints at this time and experiment with your code to understand how things are working with Unity.
2. When you are ready to continue, add a new interface called ISpeaker with the following code:

public interface ISpeaker

{

void Start();

}

1. Implement your interface with two classes now. The first class you can create as BoseSpeaker.cs and add the following code:

public class BoseSpeaker : ISpeaker

{

public void Start()

{

Console.Write("Sounds great!");

}

}

1. Then add your second class implementing the ISpeaker, call it WalmartSpeaker.cs and add the following code:

public class WalmartSpeaker : ISpeaker

{

public void Start()

{

Console.Write("Sounds awful and like it cost .99!");

}

}

1. Go back to your Unity Container.cs and add the following:

container.RegisterType<ISpeaker, WalmartSpeaker>("Cheap");

container.RegisterType<ISpeaker, BoseSpeaker>("Expensive");

1. Your full Container.cs file should look like this:

using Microsoft.Practices.Unity;

namespace UnityConsoleApp

{

public class Container

{

public static void RegisterElements(IUnityContainer container)

{

TVStand tvStand = new TVStand(70);

container.RegisterInstance(tvStand);

container.RegisterType<IAdapter, Adapter>();

container.RegisterType<IRemote, Remote>();

var adapterType = typeof(IAdapter);

var remoteType = typeof(IRemote);

container.RegisterType<ITelevision, Television>(new InjectionConstructor(adapterType, remoteType, typeof(string), typeof(string)));

container.RegisterType<ISpeaker, WalmartSpeaker>("Cheap");

container.RegisterType<ISpeaker, BoseSpeaker>("Expensive");

}

}

}

1. Add this to the end of your Main method in the Program.cs file:

ISpeaker cheapSpeaker = container.Resolve<ISpeaker>("Cheap");

ISpeaker priceySpeaker = container.Resolve<ISpeaker>("Expensive");

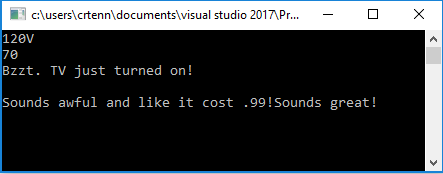
cheapSpeaker.Start();

priceySpeaker.Start();

Console.ReadLine();

**Note**: Think about the code we just added for Bose and Walmart speakers.. they both implemented ISpeaker. How did Unity know which speaker to get at which point? We used something called **Named Registration** where we gave each type a string name as we registered the mapping in the Container.cs class. Then when we called it by its “name” later in the Program.cs file it knew which one to get. Look at the above code highlighted portions. See how cheap and expensive are highlighted?

1. Run your program again. Hit enter after you see your text from before about the TV. You should see this:



1. Let’s figure out how to do automated name mappings now, also known as **registration by convention**. It relies on keeping to the same naming convention of: IMyNamedClass being associated with MyNamedClass, or IMouseBearCat with MouseBearCat. Now, every time IThing object is needed, it creates an object of Thing type and returns it. Replace the code in your Container.cs with the following:

public static void RegisterElements(IUnityContainer container)

{

container.RegisterTypes(

AllClasses.FromLoadedAssemblies(),

WithMappings.FromMatchingInterface,

WithName.Default,

WithLifetime.ContainerControlled);

{

**Note**: If you want to read about more specifics about Naming By Convention, please check the following MSDN link: <https://msdn.microsoft.com/en-us/library/dn507479%28v=pandp.30%29.aspx>

1. However, note if you try to run y our project after changing your Container.cs in the previous step it will crash on your TVStand class because it does not have an interface. Try running your program now! It would also crash on your Bose and Walmart speaker classes because the mapping is unclear. To deal with \*most\* situations + some exceptions change your class to look like the following (see how we added the exceptions back in and how they are highlighted):

using Microsoft.Practices.Unity;

namespace UnityConsoleApp

{

public class Container

{

public static void RegisterElements(IUnityContainer container)

{

container.RegisterTypes(

AllClasses.FromLoadedAssemblies(),

WithMappings.FromMatchingInterface,

WithName.Default,

WithLifetime.ContainerControlled);

TVStand tvStand = new TVStand(70);

container.RegisterInstance(tvStand);

//container.RegisterType<IAdapter, Adapter>();

//container.RegisterType<IRemote, Remote>();

//var adapterType = typeof(IAdapter);

//var remoteType = typeof(IRemote);

//container.RegisterType<ITelevision, Television>(new InjectionConstructor(adapterType, remoteType, typeof(string), typeof(string)));

container.RegisterType<ISpeaker, WalmartSpeaker>("Cheap");

container.RegisterType<ISpeaker, BoseSpeaker>("Expensive");

}

}

}

1. Let’s add one more scenario in and add another interface called IBox with the following code:

public interface IBox

{

string TypeOfMaterial();

}

1. Add the class implementing it called Box:

public class Box : IBox

{

private readonly string \_material;

public Box(string material)

{

\_material = material;

}

public string TypeOfMaterial()

{

return "Material used: " + \_material;

}

}

1. Add the following highlighted lines to the Container.cs class (note I also deleted the commented out lines as we do not need them anymore):

using Microsoft.Practices.Unity;

using UnityConsoleApp.Implementations;

namespace UnityConsoleApp

{

public class Container

{

public static void RegisterElements(IUnityContainer container)

{

container.RegisterTypes(

AllClasses.FromLoadedAssemblies(),

WithMappings.FromMatchingInterface,

WithName.Default,

WithLifetime.ContainerControlled);

TVStand tvStand = new TVStand(70);

container.RegisterInstance(tvStand);

container.RegisterType<ISpeaker, WalmartSpeaker>("Cheap");

container.RegisterType<ISpeaker, BoseSpeaker>("Expensive");

container.RegisterType<Box>(new InjectionConstructor("Cardboard"));

}

}

}

**Note**: We had to add an InjectionConstructor here because the constructor of the class took a parameter, compare how this is different from either the empty constructors or ones that took in other interfaces. Try commenting out the line containing this: container.RegisterType<Box>(new InjectionConstructor("Cardboard"));

And run your program again and see it will fail. It cannot be resolved on its own without this line. Also see that we registered the class implementation itself “Box” and not the IBox.

1. Add the highlighted to your Program.cs file:

using System;

using Microsoft.Practices.Unity;

using UnityConsoleApp.Interfaces;

namespace UnityConsoleApp

{

class Program

{

static void Main(string[] args)

{

IUnityContainer container = new UnityContainer();

Container.RegisterElements(container);

IAdapter adapter = container.Resolve<IAdapter>();

Console.WriteLine(adapter.Voltage());

TVStand tvStand = container.Resolve<TVStand>();

Console.WriteLine(tvStand.MaxInchesOfTVOnStand());

IRemote remote = container.Resolve<IRemote>();

ITelevision television = container.Resolve<ITelevision>

(new ParameterOverride("adapter", adapter),

new ParameterOverride("remote", remote),

new ParameterOverride("manufacturer", "Visio"),

new ParameterOverride("modelNumber", "321asdf"));

television.TurnOn();

Console.ReadLine();

ISpeaker cheapSpeaker = container.Resolve<ISpeaker>("Cheap");

ISpeaker priceySpeaker = container.Resolve<ISpeaker>("Expensive");

cheapSpeaker.Start();

priceySpeaker.Start();

Console.ReadLine();

IBox casing = container.Resolve<IBox>();

Console.WriteLine(casing.TypeOfMaterial());

Console.ReadLine();

}

}

}

1. Run your program and hit enter twice to get past two of the Console.ReadLines, your console should look like this:

