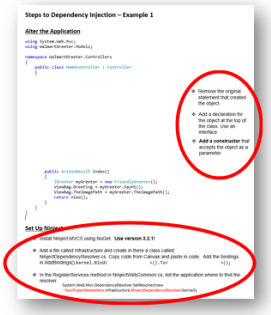
Dependency Injection – Part 1

Six Steps →

Learn them well!



Example: The Walmart Greeter

"Walmart Greeter" is an abstract concept that can be filled with actual people who perform the role of greeter. Greet the guests (Say hi)

```
public class FriendlyGreeter
{
     public string SayHi()
     {
        return "Welcome to Walmart! How are you doing today?";
     }
 }

 public class HappyGreeter
 {
     public string SayHi()
     {
         return "Happy to see you. Have a great day!";
     }
 }
```

What is an interface?

An interface is like a class but it has no method implementations – just declarations.

```
public interface IGreeter
{
    string SayHi();
}
```

No object can be based on this class. There must be other classes that have the same method names and actually include implementations.

Implementations →

```
public class FriendlyGreeter
  {
       public string SayHi()
       {
              return "Welcome to Walmart! How are you doing today?";
       }
   }

   public class HappyGreeter
   {
       public string SayHi()
       {
              return "Happy to see you. Have a great day!";
       }
   }
```

Building Loosely Coupled Components

Important feature of the MVC pattern: **Separation of Concerns**

We want the components used in an application to have as **few interdependencies** as possible. In an ideal situation, each component ***knows nothing about any other component*** and only deals with other areas of the application through **abstract interfaces**.

This is known as **loose coupling**, and it makes testing and modifying applications easier.

Dependency Injection helps us accomplish this.

Dependency Injection (DI)

A statement like the following within our application would establish a dependency on HappyGreeter.

```
HappyGreeter myGreeter = new HappyGreeter();
```

That's not good if we want **loose coupling**.

So what would we do if we want to break the dependency?

We should **take out any mention of HappyGreeter** within our application. Instead, we should create an interface (which is an abstract representation of HappyGreeter) and **refer to it only**.

But how will our application ever know which actual implementation we want it to use? We will *find a way* to **inject the dependencies** at runtime, hence the term dependency injection.

How can a dependency be injected into our application? We will use **Constructor Injection**.

What is a constructor?

A constructor is a method with no return type, with the same name as the class. The statements within a constructor execute when the class is instantiated.

```
public class MyClass
{
 public MyClass()
 {
     //Statements here
 }
}
```

```
public class MyClass
{
 public MyClass(Some parameter)
 {
     //Statements here
 }
}
```

We can set up a parameter here that will allow us to send in the dependency we want to use.

Using a Dependency Injection Container

How are we going to **pass in a parameter** if we don't want to write any code to do it? We will create a **dependency injection container** that acts as a **broker** between the dependencies and the application itself.

The DI container will list the **set of interfaces** that our application uses along with the **specific implementations** we would like to be used.

Our application **only deals with the interfaces** and when it comes time to instantiate an object, we say,

"Hey DI container, go check to see which implementation I should use and use that one. "

We will use **Ninject** for our DI containers.

http://www.ninject.org/

Download and upzip the WalmartGreeterNoDI project from Canvas. Open the project in Visual Studio.

This project works fine but it does not use dependency injection.

```
public class FriendlyGreeter
{
    1 reference
    public string SayHi()
    {
        return "Welcome to Walmart!  How are you doing today?";
    }
    1 reference
    public string TheImagePath()
    {
        return "Images\\FriendlyGuy.jpg";
    }
}
```

```
public class HappyGreeter
{
    0 references
    public string SayHi()
    {
        return "Happy to see you. Have a great day!";
    }
    0 references
    public string TheImagePath()
    {
        return "Images\\HappyGuy.jpg";
    }
}
```

```
public class HomeController : Controller
{
    // GET: Home
    0 references
    public ActionResult Index()
    {
        FriendlyGreeter myGreeter = new FriendlyGreeter();
        ViewBag.Greeting = myGreeter.SayHi();
        ViewBag.TheImagePath = myGreeter.TheImagePath();
        return View();
    }
}
```

```
Index View

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body style="background-color: #0026ff; color:white;">
    <h1>Walmart</h1>
    <div>
        <img src="~/@ViewBag.TheImagePath" />
        <h1>    @ViewBag.Greeting </h1>
    </div>
</body>
</html>
```

Now let's alter this project so that it incorporates **dependency injection**.

We will write out on paper what changes need to be made. Watch the video and then write in your changes.

Steps to Dependency Injection – Example 1

Alter the Application

```
using System.Web.Mvc;
using WalmartGreeter.Models;

namespace WalmartGreeter.Controllers
{
    public class HomeController : Controller
    {
```

- Remove the original statement that created the object.
- Add a declaration for the object at the top of the class. Use an interface.
- **Add a constructor** that accepts the object as a parameter.

```
        public ActionResult Index()
        {
            IGreeter myGreeter = new FriendlyGreeter();
            ViewBag.Greeting = myGreeter.SayHi();
            ViewBag.TheImagePath = myGreeter.TheImagePath();
            return View();
        }
    }
}
```

There are three more steps but we'll talk about them later.

Again, watch the video and then write in your changes.

Steps to Dependency Injection – Example 2

Alter the Application

```
using System.Web.Mvc;
using EssentialTools.Models;
namespace EssentialTools.Controllers
{
    public class HomeController : Controller
    {
        private Product[] products = {
            new Product {Name = "Kayak", Category = "Watersports", Price = 275M},
            new Product {Name = "Lifejacket", Category = "Watersports", Price = 48.95M},
            new Product {Name = "Soccer ball", Category = "Soccer", Price = 19.50M},
            new Product {Name = "Corner flag", Category = "Soccer", Price = 34.95M}
        };
```

- Remove the original statement that created the object.
- Add a declaration for the object at the top of the class. Use an interface.
- **Add a constructor** that accepts the object as a parameter.

```
        public ActionResult Index()
        {
            IValueCalculator calc = new LinqValueCalculator();
            ShoppingCart cart = new ShoppingCart(calc) { Products = products };
            decimal totalValue = cart.CalculateProductTotal();
            return View(totalValue);
        }
    }
}
```

There are three more steps but we'll talk about them later.

Steps to Dependency Injection – Example 3

Alter the Application

```
using System.Web.Mvc;
using Lab6.Models;

namespace Lab6.Controllers
{
    public class HomeController : Controller
    {
        StoreItem item = new StoreItem { Name = "Sofa", Category = "Furniture",
                         Price = 935.95M, Discountable = true };

        public ActionResult Index()
        {
            IPriceCalculator calc = new BlackFridaySalePrice();
            if (item.Discountable == true)
            {
                decimal totalValue = calc.CalcSalesPrice(item.Price);
                return View(totalValue);
            }
            else
            {
                return View(item.Price);
            }
        }
    }
}
```

- Remove the original statement that created the object.
- Add a declaration for the object at the top of the class. Use an interface.
- **Add a constructor** that accepts the object as a parameter.

There are three more steps but we'll talk about them later.

Now that we know how to alter the code to prepare for dependency injection, let's learn how to complete the remaining steps in **Dependency Injection – Part 2**.