

Equals – HOL - Mixin

Hands on Labs

Copyright notice: ©2011 rubicon informationstechnologie gmbh all rights reserved

Table of Contents

[1 Goal 1](#_Toc286244738)

[2 Lab 1 – Uses Szenario 1](#_Toc286244739)

[2.1 Theory – Equals 1](#_Toc286244740)

[2.2 Exercise 1 – Equals 2](#_Toc286244741)

[2.2.1 Task 1 Implementation 2](#_Toc286244742)

[2.2.2 Questions / Excercises 3](#_Toc286244743)

[2.3 Theory - Reflection 3](#_Toc286244744)

[2.4 Exercise 2 – Equals with Base 3](#_Toc286244745)

[2.4.1 Task 1 Implementation 3](#_Toc286244746)

[2.4.2 Questions / Excercises 4](#_Toc286244747)

[2.5 Exercise – UtilityImplementation 4](#_Toc286244748)

[2.5.1 Task 1 Implementation 4](#_Toc286244749)

[2.5.2 Questions / Excercises 5](#_Toc286244750)

[3 Mixin Implementation 6](#_Toc286244751)

[3.1 Exercise – MixinImplementation 6](#_Toc286244752)

[3.1.1 Task 1 Implementation 6](#_Toc286244753)

[3.1.2 Questions / Excercises 7](#_Toc286244754)

[3.1.3 Task 2 Attribute Enhancement 7](#_Toc286244755)

[3.1.4 Task 3 Initialisation 7](#_Toc286244756)

[3.1.5 Questions / Excercises 8](#_Toc286244757)

[4 Lab Summary 9](#_Toc286244758)

# Goal

Goal of this HOL is to teach the fundamentals of re-mix.

Unlike the other HOLs to Mixins, this HOL wants to focus on the designer’s perspective. It wants to outline in which situations Mixins can be the best pattern to solve an issue and why.

During this HOL you will run into sections with questions. Answers to some questions might be straight forward. Other questions could be tricky. In case you are doing this exercise with other developers, we encourage you to talk about your ideas and your proposed solutions.

After this exercise you will be able to understand the advantages of re-mix and to compare them with other structural design patterns

Please have also a look on the further readings section in the end of this HOL. They provide some suggestions on how you might want to proceed to learn more.

In this HOL we will start with a problem to solve. We will try to use several methods to solve it before solving it with mixins.

Topics:

|  |
| --- |
| * Implementing Equals (POCO) * Implement Equals with Base classes * Solve Equal Szenarios with static methods of a CompareUtility class * Solve with Delegation * Mixins |

**Important:**

For this HOL Visual Studio 2010 must be installed on your working PC. Additionally, access to a SQL Server 2008 or higher is required.

We recommend installing the following two third party tools too:

* JetBrains Resharper (<http://www.jetbrains.com/resharper/>)
* RedGate Reflector (<http://reflector.red-gate.com/Download.aspx>)

Both tools will help you to understand the samples in this HOL better.

Go to <https://www.re-motion.org/builds/> to get the latest re-motion build. All samples were tested with version 1.13.95.

# Lab 1 – Uses Szenario

## Theory – Equals

It is essential that you understand the background to Equals. The following two articles provide a good instruction

* <http://msdn.microsoft.com/en-us/library/bsc2ak47.aspx>
* <http://www.codeproject.com/KB/dotnet/IEquatable.aspx>

When you start to work with the sample, you should know the difference between

* reference types and value types
* reference equality and bitwise equality

## Exercise 1 – Basic Equals Implementation

### Task 1 Implementation

1. Start Visual Studio 2010
2. Add a new project and select console application. For this HOL, we recommend the following settings:

Name: HOLApp

Location: C:\HOL\mixins

Solution name: HOLEquals

1. Add a file BasicImplementation.cs to the Domain project and use the following source code

namespace HOLApp.BasicImplementation

{

public class Address

{

public string Street;

public int StreetNumber;

public string City;

public string ZipCode;

public string Country;

public override bool Equals (object obj)

{

return Equals (obj as Address);

}

public bool Equals (Address other)

{

if (other == null)

return false;

if (this.GetType () != other.GetType ())

return false;

return Street == other.Street && StreetNumber == other.StreetNumber &&

City == other.City && ZipCode == other.ZipCode && Country == other.Country;

}

public override int GetHashCode ()

{

return Street.GetHashCode () ^ StreetNumber.GetHashCode () ^

City.GetHashCode () ^ ZipCode.GetHashCode () ^ Country.GetHashCode ();

}

}

}

1. Add the following source code to the console application.

using System;

namespace HOLApp

{

class Program

{

static void Main (string[] args)

{

HOLApp.BasicImplementation.Address address1 = new HOLApp.BasicImplementation.Address ();

HOLApp.BasicImplementation.Address address2 = new HOLApp.BasicImplementation.Address ();

Console.WriteLine ("Basic Implementation: Both instances have the same values: {0}", Equals (address1, address2));

Console.ReadKey();

}

}

}

1. Debug and have a look at the result.
2. Change the class declaration to

public class Address : IEquatable<Address>

1. Debug and have a look at the result.

### Questions / Excercises

* Can you describe what the GetHashCode implementation does? Technically I would be possible to implement Equals Functionality without overriding the GetHashCode() method. In which problems could you run into if you do that?
* In the class, we have five fields. In theory there could be far more and it can get inconvenient. Is there a way to solve that?
* Does it make sense to check the Equality of two types by comparing the values of their properties too?
* In Step 5, we have added the interface IEquatable<Address>. Can you explain, why this makes sense?
* True or false: If we implement other classes similar to Address (for example a class PhoneNumber with fields such as CountryCode, AreaCode, Number, Extension) and we have to implement a similar functionality again (in this case a method that compares values field per field), it would be a perfect example for a Boilerplate Code?
* Preparation for Exercise 2: If you want to apply the similar functionality to other classes, what can you do to avoid to implement the functionality to compare the values of the fields?
* **Expert Question:** DDD would describe Address as a Value Object (do not confuse this object with a value type). Should there be a different approach to determine Equality with Entities and Aggregates?

## Theory - Reflection

If you have no experience with reflection, it might be a good idea to get an overview before reading on.

## Exercise 2 – Equals with Base

In the follwo

### Task 1 Implementation

1. Add a new class BaseClassEquals and add the following code

using System;

using System.Linq;

using System.Reflection;

namespace HolApp.BaseClassEquals

{

public class EquatableByValues<T> : IEquatable<T>

where T : class

{

private static readonly FieldInfo[] s\_targetFields = typeof (T).GetFields (

BindingFlags.Instance | BindingFlags.Public | BindingFlags.NonPublic);

public bool Equals (T other)

{

if (other == null)

return false;

if (GetType () != other.GetType ())

return false;

for (int i = 0; i < s\_targetFields.Length; i++)

{

object thisFieldValue = s\_targetFields[i].GetValue (this);

object otherFieldValue = s\_targetFields[i].GetValue (other);

if (!Equals (thisFieldValue, otherFieldValue))

return false;

}

return true;

}

public new int GetHashCode ()

{

return s\_targetFields.Aggregate (0, (current, t) => current ^ t.GetValue (this).GetHashCode ());

}

}

}

1. Add to HolAppBaseClass namespace

public class Address : EquatableByValues<Address>

{

public string Street;

public string StreetNumber;

public string City;

public string ZipCode;

public string Country;

}

1. Add the following code to the implementation

HolApp.BaseClassEquals.Address address3 = new HolApp.BaseClassEquals.Address ();

HolApp.BaseClassEquals.Address address4 = new HolApp.BaseClassEquals.Address ();

Console.WriteLine ("Inheritance Implementation: Both instances have the same values: {0}", Equals (address3, address4));

1. Change the implementation of HolAppBaseClass.Address as follows

public class Address : EquatableByValues<Address>

{

public string City;

public string ZipCode;

public string Country;

}

public class StreetAddress : Address

{

public string Street;

public string StreetNumber;

}

1. Add

HolApp.BaseClassEquals.StreetAddress streetaddress1 = new HolApp.BaseClassEquals.StreetAddress ();

HolApp.BaseClassEquals.StreetAddress streetaddress2 = new HolApp.BaseClassEquals.StreetAddress ();

streetaddress1.Street = "Test";

streetaddress2.Street = "Test2";

Console.WriteLine ("(Value not as expected)Inheritance Implementation StreetAddress: Both instances have the same values: {0}", Equals (streetaddress1, streetaddress2));

### Questions / Excercises

* Problem 1: Look on public class Address : EquatableByValues<Address>. In which ways are you restricted? What if Address shall also derive from other classes? Look up “Multiple Inheritance in .NET” in case it is necessary
* Problem 2: Why does the StreetAddress compare fail? Hint: Think: Which interface does StreetAddress implement?

## Exercise – UtilityImplementation

### Task 1 Implementation

using System;

using System.Reflection;

namespace HOLApp.UtilityImplementation

{

public class ComparyUtility

{

public static bool FieldEquals (object a, object b)

{

if ((a == null) != (b == null))

return false;

if (a.GetType () != b.GetType ())

return false;

FieldInfo[] s\_targetFields = a.GetType().GetFields (BindingFlags.Instance | BindingFlags.Public | BindingFlags.NonPublic);

for (int i = 0; i < s\_targetFields.Length; i++)

{

object thisFieldValue = s\_targetFields[i].GetValue (a);

object otherFieldValue = s\_targetFields[i].GetValue (b);

if (!Equals (thisFieldValue, otherFieldValue))

return false;

}

return true;

}

}

public class Address : IEquatable<Address>

{

public string City;

public string ZipCode;

public string Country;

public override bool Equals (object obj)

{

return Equals (obj as Address);

}

public bool Equals (Address other)

{

return HOLApp.UtilityImplementation.ComparyUtility.FieldEquals(this, other);

}

public override int GetHashCode ()

{

throw new NotImplementedException();

}

}

public class StreetAddress : Address

{

public string Street;

public string StreetNumber;

}

}

HOLApp.UtilityImplementation.StreetAddress streetaddress3 = new HOLApp.UtilityImplementation.StreetAddress ();

HOLApp.UtilityImplementation.StreetAddress streetaddress4 = new HOLApp.UtilityImplementation.StreetAddress ();

streetaddress3.Street = "Test";

streetaddress4.Street = "Test2";

Console.WriteLine ("Utility Implementation StreetAddress: Both instances have the same values: {0}", Equals (streetaddress3, streetaddress4));

### Questions / Excercises

* How would you implement the Get Hashcode?

# Mixin Implementation

## Exercise – MixinImplementation

### Task 1 Implementation

1. Add References tot he proects

* remotion.dll
* remotion.interfaces.dll

**Please note:** A good practice might be to put all re-motion assemblies under the directory References below the solution file.

1. Add class MixinImplementation

using System;

using System.Linq;

using System.Reflection;

using Remotion.Mixins;

namespace HOLApp.MixinImplementation

{

public class EquatableByValuesMixin<T> : Mixin<T>, IEquatable<T>

where T : class

{

private static readonly FieldInfo[] s\_targetFields = typeof (T).GetFields (

BindingFlags.Instance | BindingFlags.Public | BindingFlags.NonPublic);

public bool Equals (T other)

{

if (other == null)

return false;

if (Target.GetType () != other.GetType ())

return false;

for (int i = 0; i < s\_targetFields.Length; i++)

{

object thisFieldValue = s\_targetFields[i].GetValue (Target);

object otherFieldValue = s\_targetFields[i].GetValue (other);

if (!Equals (thisFieldValue, otherFieldValue))

return false;

}

return true;

}

[OverrideTarget]

public new bool Equals (object other)

{

return ((IEquatable<T>)this).Equals (other as T);

}

[OverrideTarget]

public new int GetHashCode ()

{

return s\_targetFields.Aggregate (0, (current, t) => current ^ t.GetValue (Target).GetHashCode ());

}

}

public class Address

{

public string City;

public string ZipCode;

public string Country;

}

public class StreetAddress : Address

{

public string Street;

public string StreetNumber;

}

}

1. Add Code to the console app

HOLApp.MixinImplementation.StreetAddress mixedAddress1 = ObjectFactory.Create<HOLApp.MixinImplementation.StreetAddress> (ParamList.Empty);

HOLApp.MixinImplementation.StreetAddress mixedAddress2 = ObjectFactory.Create<HOLApp.MixinImplementation.StreetAddress> (ParamList.Empty);

Console.WriteLine ("Mixed Implementation StreetAddress: Both instances have the same values: {0}", Equals (mixedAddress1, mixedAddress2));

1. Enable Mixins by class decoraction

[Uses (typeof (EquatableByValuesMixin<Address>))]

public class Address

1. Build and test

### Questions / Excercises

* You can try to make Address Abstract.
* You can try to get rid of type Address in uses

### Task 2 Attribute Enhancement

1. Remove type abstract in uses
2. Add [BindToTargetType]
3. Now add

public class EquatableByValuesAttribute : UsesAttribute

{

public EquatableByValuesAttribute ()

: base (typeof (EquatableByValuesMixin<>))

{

}

}

And replace [Uses (typeof (EquatableByValuesMixin<>))]

With

[EquatableByValuesAttribute]

### Task 3 Initialisation

Replace Address

public abstract class Address

{

protected Address ()

{

}

protected Address (int zipCode, string city)

{

ZipCode = zipCode;

City = city;

}

protected Address (int zipCode, City city)

{

ZipCode = zipCode;

City = city.Name; // null-check missing

}

public int ZipCode;

public string City;

}

public class City

{

public string Name;

}

public class StreetAddress : Address

{

public StreetAddress (int zipCode, string city, string street, string streetNumber)

: base (zipCode, city)

{

Street = street;

StreetNumber = streetNumber;

}

public string Street;

public string StreetNumber;

}

public class POBoxAddress : Address

{

public POBoxAddress (int zipCode, string city, int poBox)

: base (zipCode, city)

{

POBox = poBox;

}

public int POBox;

}

Replace

StreetAddress mixedAddress1 = ObjectFactory.Create<StreetAddress> (ParamList.Create (1010, "Wien", "Stephansplatz", "1"));

StreetAddress mixedAddress2 = ObjectFactory.Create<StreetAddress> (ParamList.Create (1010, "Wien", "Stephansplatz", "1"), new EquatableByValuesMixinMock<StreetAddress> ());

Console.WriteLine ("fields of mixed address object have the same values: {0}", Equals (mixedAddress1, mixedAddress2));

StreetAddress mixedAddress3 = ObjectFactory.Create<StreetAddress> (ParamList.Create (1010, default (string), "Stephansplatz", "1"));

### Questions / Excercises

* You can try to make Address Abstract.
* You can try to get rid of type Address in uses

StreetAddress mixedAddress3 = ObjectFactory.Create<StreetAddress> (ParamList.Create (1010, default (string), "Stephansplatz", "1"));

* Add a constructor that contains class City

# Lab Summary

In this lab, you have successfully done the following

|  |
| --- |
| * You have created … * You have … * You have …. |

As you now are experienced in the main usage of mixins, you might want to dig deeper to become a real mixin expert. We suggest:

* Read “Head First Design Patterns” to be able to understand design techniques in total better. It helps you to put mixins in relation to other design topics
* Read the next re-motion HOL to understand another re-motion technology.
* On <http://en.wikipedia.org/wiki/Mixin> in further readings, there are many references to other mixin implementations, you might want to read them and compare them with the re-motion approach.
* Read the blogs on [www.re-motion.org](http://www.re-motion.org)
* For those who want to become real experts: Read “CLR via C#”, if you want to understand more in how code is generated and how mixins are generated. Take care: Reading the most essential parts book also with good background experience may take a lot of time. Plan this for an extended research time and prepare enough coffee.