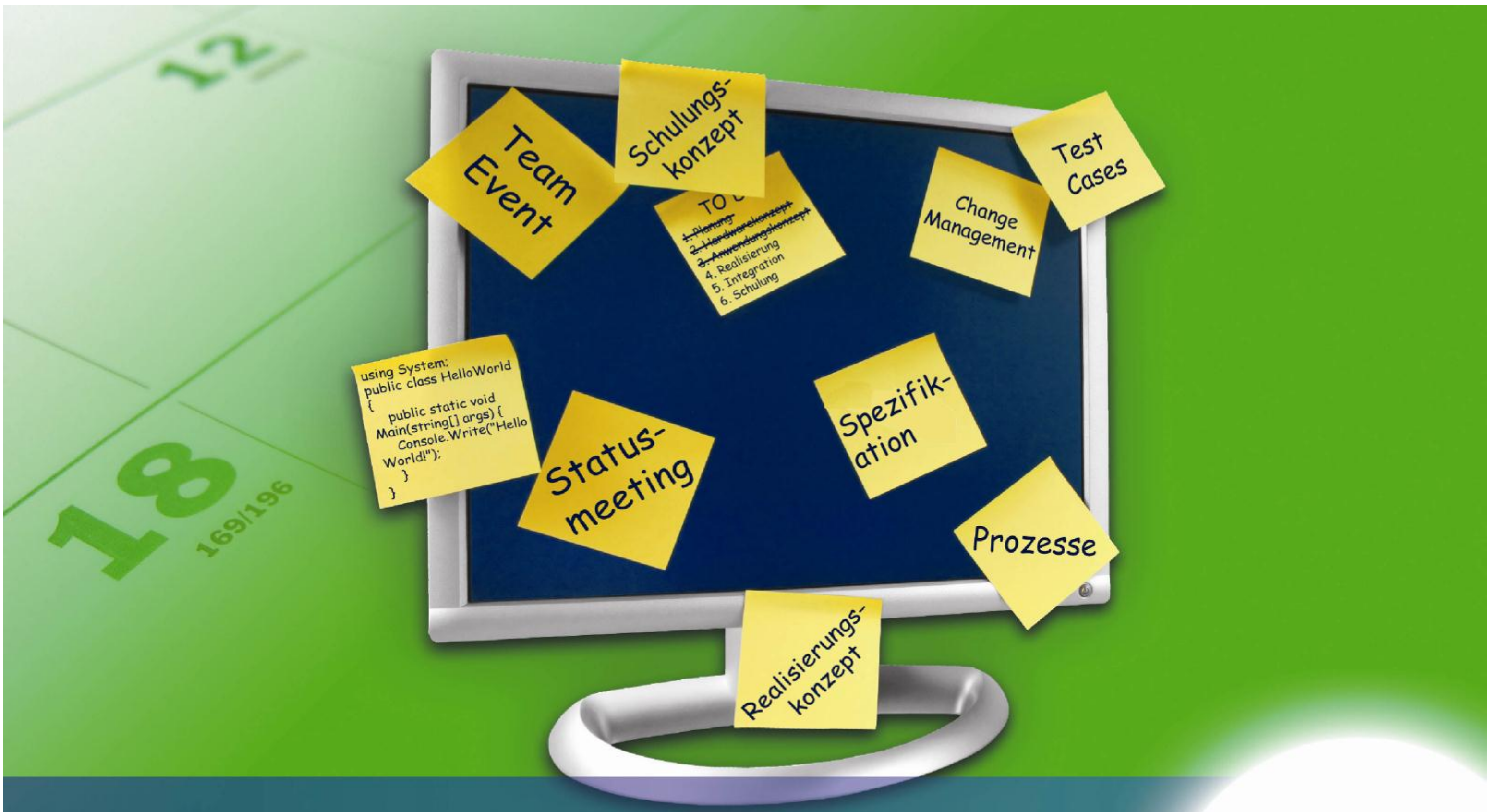

Equals – HOL - Mixin

Hands on Labs

rubicon
informationstechnologie gmbh

Copyright notice: ©2011 rubicon informationstechnologie gmbh all rights reserved

Table of Contents

1 Answers to 2.2.2 (Basic)

- **I would be possible to implement Equals without overriding the GetHashCode() method. In which problems could you run into if you do that?**

  A hash code is a numeric value that is used to identify an object during equality testing. It can also serve as an index for an object in a collection.

  So without a GetHashCode implementation the type cannot be added to HashTables,

- **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?**

  As shown in the later example, it is possible to use reflection to get all fields.

- **Does it make sense to check the equality of two types by comparing the values of their properties too?**

  Comparing properties can be troublesome as they could contain code that would change the type, when calling it. Consider LazyLoading.

```
private MyVal _myVal;
    public MyVal Test
    {
      get
      {
        if (_myVal == null)
        {
          _myVal = LazyLoader.LoadMyValue();
        }
        return _myVal;
      }
    }
```

- **Are there other “Standard Equals Implementations”?**

  Discussing this in full detail is out of scope.

  A good starting point for a discussion might be: An importer imports some data from csv files. Business rules require that with the generated objects a timestamp is stored when the object was imported. So an argument might be compare that there might be a standard implementation that compares all values except this timestamp or similar information which was added by an importer.

  Another approach would be to add algorithms that determine the likelihood that two entries are the same. For instance, you have an application, where more human users enter address data. They might enter for instance “Wall str.” and “Wall Street”.

- **In step 5, we have added the interface IEquatable<Address>. Can you explain why this makes sense?**

  If you add this type to a List<T>. Methods such as Contains, IndexOf, LastIndexOf, and Remove use an equality comparer for the list elements. The default equality comparer for type *T* is determined as follows. If type *T* implements the IEquatable<T> generic interface, then the equality comparer is the Equals(T) method of that interface; otherwise, the default equality comparer is Object.Equals(Object).

- **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?**

  True

- **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?**

  See the latter exercise

- **Expert Question: Using DDD terminology, Address is a perfect example for 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*?**

  Yes, but it is out of scope to discuss this in detail in this document.

2 Answers to 2.4.2 (Inheritance)

- **Problem 1: Look at** `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!**

  You cannot derive from another class as there is no multiple inheritance in .NET

- **Problem 2: Why does the StreetAddress equals does not return the expected values? Hint: Which interface does StreetAddress implement?**

  StreetAddress implements IEquatable<Address>, so only the fields of Address are used for the Equals method

- **We are using reflection in this sample. Do you think this can lead to performance issues?**

  Reflection is of course slower, but it is unlikely to run into serious performance problems.

3 Answers to 2.5.2 (Utility)

- **Statement: "In many projects there is a "UtilityHell". Every developer implements some generic functionality in static methods and puts it in a Utility Assembly. In worst cases, the developer does not even check if there is already an implementation available. Step by step this assembly gets stuffed and nobody knows if some implementations are still used and by whom". What is your opinion about this statement? Have you encountered something similar?**

Subjective answer: Many developers who have probably not too much experience with OOP, use static methods solve several issues, where they are blocked by a suboptimal design.

4 Answers to 2.5.4 (Utility)

- **In special cases, it might be necessary to store state information. A practical example would be a tolerance for minor syntactical differences. We might want to store that two objects would be non-equal, if they were case insensitive. How can this be implemented with this approach?**

  One possibility would be to change the parameter list of the delegate method and add a reference parameter.

5 Answers to 3.1.2 (Mixin)

- **Have a look on the source code and identify the parts that reference the re-motion assemblies. Look them up in the source code or inspect them via intellisense (pdb files required)**

  Mixin<T>: Acts as a base class for mixins that require a reference to their target object (<see cref="Target"/>).

  Target: Represents the target class.

OverrideTarget: Indicates that a mixin member overrides a virtual member of the mixin's target class.

- **Mixin<T> is required to reference the Target. If no “Target is needed”, can you work without this base class**

  Yes. The base class just adds the reference to the Target.

6 Answers to 3.1.4 (Mixin Attribute)

- **Use Reflector or re-motion Source Code to investigate** `BindToTargetType`

  BindToTargetType indicates that a generic parameter of a mixin should be bound to the mixin's target type (unless the generic parameter type is explicitly

7 Answers to 3.1.6 (Mixin Initialization)

- **ParamList.Create has a huge list of overloads. Can you explain why this might be necessary?**

  The Factory Methods cannot be dynamically created and must provide a huge number of possibilities.

8 Answers to 3.1.8 (Methods)

- **Can you explain what has changed in Task 3?**

  The first implementation implements the IEquatabley<T> implicitly. So there is a cast and the Equals method of the IEquatable is called. That’s why it would not work, if do not cast to (IEquatable<T> in the bool Equals(object other) method

  In the second imeplementation, the IEquatable Interface is not called implicitly

- **Does it make sense to make public book Equals(T other) virtual?**

  This depends on your subclass

9 Answers to 3.2

- **In which other uses cases it might make sense to use Mixins?**

  One example would be to implement IComparable. There are also many uses cases for the “Mixin Extens” scenario.

- **Imagine the following scenario:**

  You have several Entities as Domain Objects. Two of them are: Person and Document. In your business application, you need the following generic functionality: Revision Tracking, Change Tracking and Approval. The classic way to implement this is to implement a “BaseDomainObject class” that contains generic code for these three use cases.

  But not all Domain Object need everything. There is no approval for changes to a person required, whereas it is obligatory to Documents. Does it make sense to replace inheritance with Mixins, such as (remark: The ComposedObject is a pattern to include more interfaces in one base class)

```
  [Uses(typeof(TrackableMixin))]
  [Uses(typeof(RevisionableMixin))]
```

```
public class Person : ComposedObject<IPerson>, IPersonImpl

[Uses(typeof(TrackableMixin))]
[Uses(typeof(ApprovableMixin))]
[Uses(typeof(RevisionableMixin))]
public class Document : ComposedObject<IDocument>, IDocumentImpl
```

There are many advantages with this scenario.