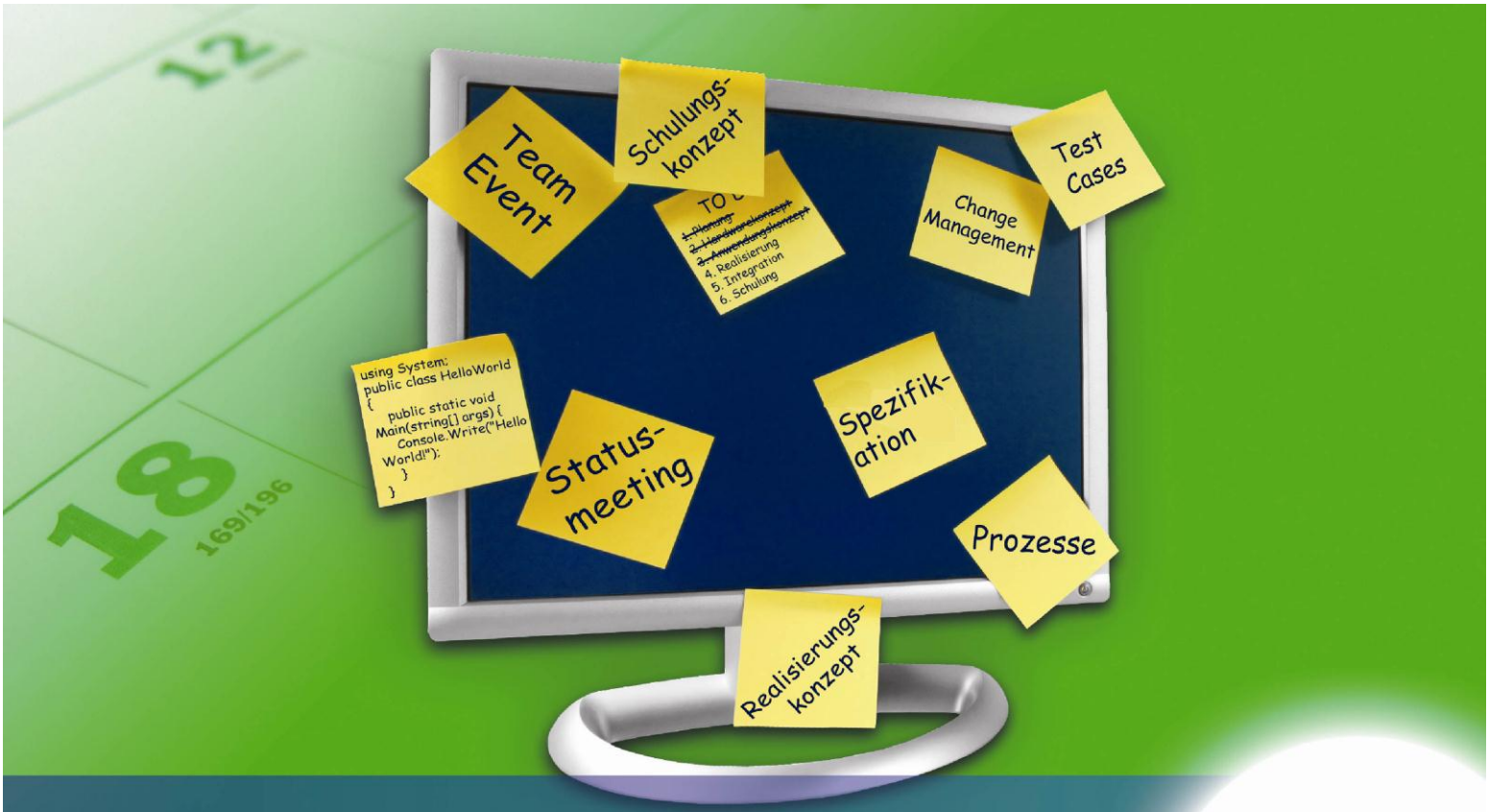

Level 100 – mixins

Hands on Labs

rubicon
informationstechnologie gmbh

Inhaltsverzeichnis

1 Lab 1: First steps with mixins

1.1 Goal

Estimated Time: **30 Minuten**

Goal of this HOL is to outline the advantages of mixins by programming examples. All exercises are created to be solved step by step. At the end of each exercise, you should be able to finish it with an executable result.

At the end of an exercise, in some cases you will find a section with questions. We suggest that you answer the questions as good as possible. Some answers to some questions might be straight forward, if you are an experienced developer. Other questions could be tricky. In case, you are doing this exercise with other software developers, we encourage you to talk about your ideas and your proposed solutions.

After this exercise, you are able to understand the advantages of mixins. You will be able to decide, when a mixin is appropriate and when not and you are ready to implement mixins in common scenarios.

Please have also a look on the further readings section in the end of this HOL, if you want to dig deeper

Topics covered:

- Basics: What are mixins
- Developing your first mixin
- Understanding the difference between “use” and “extends” implementations
- Parameters

Important:

For this HOL you must have Visual Studio 2010 installed on your working PC. We recommend installing additional third party tools such as Reshaper or Reflector too. This will help you to understand some samples better and you are able to dig deeper.

If you have no idea how to get re-motion, in the appendix Getting re-motion there is a guide how to get a version of re-motion. All samples were tested with version 1.13.87.

1.2 Theory – Why mixins

Temporary: read \\rubicon\projekte\remotion\Dokumentation\Mixins\Mixins.pptx

1.3 Exercise 1 – ten minutes mixin

1. Start Visual Studio 2010
2. Add a new project (Menu: File – New project) and select a class library. For this HOL, recommend the following settings:

   Name: Domain

   Location: D:\HOL\mixins (use C: if you have no data partition D:)

   Solution name: HOLMixin

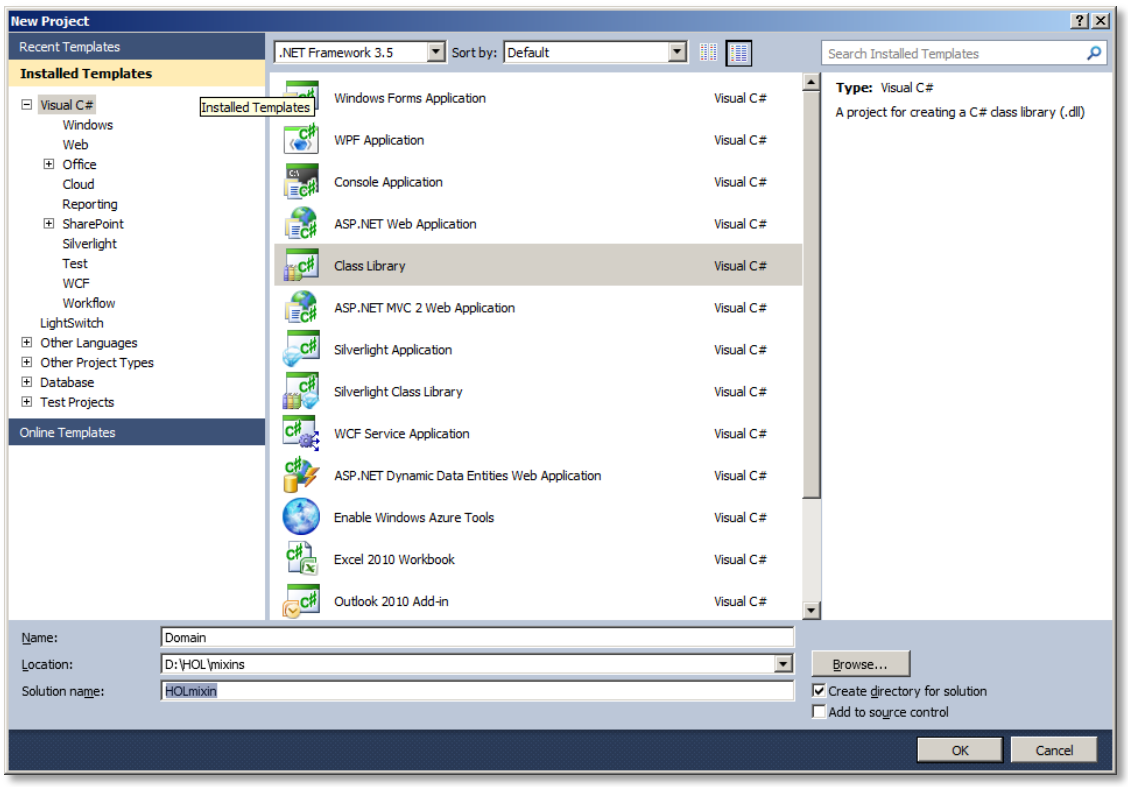

Illustration 1: New class library

3. Create a new console library (In the Solution Explorer, open the context menu by selecting the very first tree element (solution name) and click on add new project.

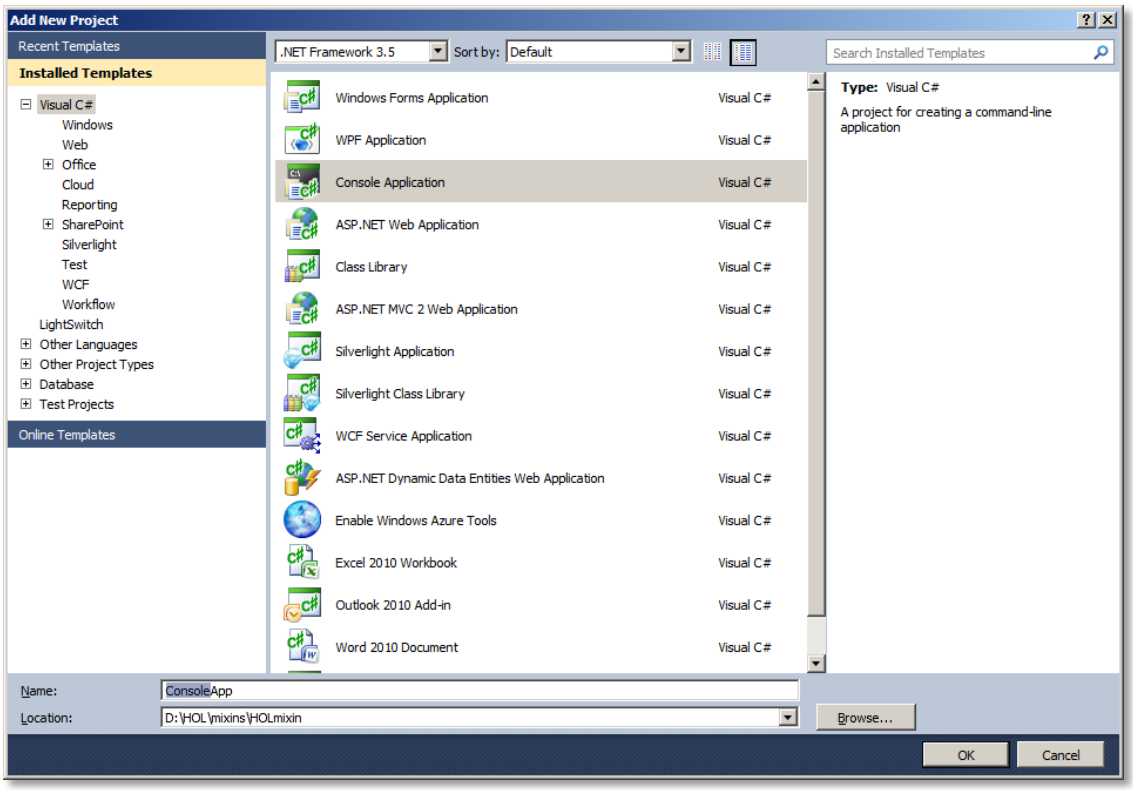

Illustration 2 New Console Application

4. Add the following references

- remotion.dll
- remotion.interfaces.dll

You might want to read the section “Adding references” in case you are not sure where to put the binary assemblies.

5. Add a file person.cs the domain and use the following source code

```
using System;

namespace Domain
{
  public class Person
  {
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public DateTime BirthDay { get; set; }
  }
}
```

6. Add a file EmployeeMixin.cs the domain and use the following source code

```
using System;
using Remotion.Mixins;

namespace Domain
{
  public interface IEmployeeMixin
  {
    int Salary { get; set; }
    DateTime? HireDate { get; set; }
  }

  [Extends(typeof(Person))]
  public class EmployeeMixin : IEmployeeMixin
  {
    public int Salary { get; set;  }
    public DateTime? HireDate { get; set;}
    public EmployeeMixin()
    {
      // set default values
      Salary = 1000;
      HireDate = DateTime.Now;
    }
  }
}
```

7. Add the following source code to the Console Application.

```
using Domain;
using Remotion.Reflection;
using Remotion.Mixins;

namespace ConsoleApp
{
  class Program
  {
    static void Main(string[] args)
    {
      var employee =
(IEmployeeMixin)ObjectFactory.Create<Person>(ParamList.Empty);
      ((Person) employee).FirstName = "Max";
      ((Person)employee).LastName = "Mustermann";
      System.Console.WriteLine("Fullname: {0}", ((Person)employee).FirstName
+ " " + ((Person)employee).LastName);
      System.Console.WriteLine("Salary: {0}", employee.Salary);

      System.Console.ReadKey();
    }
  }
}
```

8. Debug and have a look on the result.

Within a very short time, you have learned that is very easy to implement a basic mixin within your solution. In the upcoming tasks, we want to dig deeper and explore some scenarios.

1.4 Questions

- In the first sample, we implemented an employee mixin for person. Think about this implementation. If being an employee in a company is a context, which more contexts could be in the real world?
- You will probably find a lot of different contexts. How would you implement this with pure OOP? What are disadvantages and advantages over mixins?
- In the first sample, the employee mixin extends the person class. Does it make sense to put this mixin on the top of different class? If not, try to give a reason.
- After thinking about the answer to the question before. It seems that person is “a fixed constant” and this class will be extended by mixins. Can you think of a scenario, where the mixin is a bundle of functionality that can be added to more than just one class?

1.5 Free Exercise

Please add two mixins for different contexts to our sample.

In the first context, the person is bank customer. Add at least account number and balance as properties for the mixins.

In the second context, we want to manage a family. We should add composition in a mixin. As a family, we have a list of children containing 0 or more entries. Also add a spouse (which might be of course null too), a father and a mother.

It is not required to add father and mother as required parameters. But it might be a good idea to think, if it makes sense to add required parameters

1.6 Exercise 2 – Parameter

In our free exercise, we have added more mixins.

Now our test application should be expanded to. For each new context, we would have to add the properties of the base class person such as FirstName after its creation via the object factory. We will agree that this is complicated and it is preferable to supply a parameter to create the object.

1. Please add a copy constructor to tje person class. We also have to define a default constructor

```
public Person()
{
}

public Person(Person person)
{
  FirstName = person.FirstName;
  LastName = person.LastName;
  BirthDay = person.BirthDay;
}
```

2. Please replace

```
var employee = (IFamilyMember)ObjectFactory.Create<Person>(ParamList.Empty);
```

with

```
      Person p = new Person();
      p.FirstName = "Max";
      p.LastName = "Mustermann";

      var employee =
(IEmployeeMixin)ObjectFactory.Create<Person>(ParamList.Create(p));
```

1.6.1 Questions

- In the first example, we have not specified a default constructor. When we add now a copy constructor without adding a default constructor too, we run into compiler error. Why?
- In our sample objects are created via a Factory Design Pattern. Can you explain the advantages of using Factories to instantiate classes via constructors within a few sentences? How would you try to convince a fellow developer to use factories instead of construction
- If you use intellisense to look for possible parameters for ParamList.Create(), you will find that there are a lot of possibilities to choose. Can you explain the reason?

1.7 Exercise 3 – A use case for „use“

Our first sample, is a “extends” scenario. We have a fixed base class that can be extended with one or more mixins. There is also another scenario, where we have “a fixed mixin” that can be added to various classes. This scenario is called “use scenario”. In the following example we implement a mixin that will generate a string that summarizes all public properties of the class and their current values.

This mixin could be added to our sample. As we might want to add information to the console such as:

```
      System.Console.WriteLine("Fullname: {0}", ((Person)employee).FirstName
+ " " + ((Person)employee).LastName);
```

The idea is to generate a string with all public properties in one method (we will call it **PropertyValues** of a mixin. This mixin is then used by various classes.

```
   System.Console.WriteLine(((IPropertyValue)employee).PropertyValues);
```

1. We add a new Mixin. In the Property PropertyValues we read all properties and

```
public interface IPropertyValue
  {
    string PropertyValues { get; }
  }

  public class PropertyValueStringChainMixin<T> : Mixin<T>, IPropertyValue
where T : class
  {
    public string PropertyValues
    {
      get
      {
        StringBuilder stringBuilder = new StringBuilder();

        foreach (PropertyInfo propertyInfo in This.GetType().GetProperties())
        {
          stringBuilder.Append(propertyInfo.Name);
          stringBuilder.Append(": ");
          stringBuilder.Append(propertyInfo.GetValue(This, null));
          stringBuilder.Append("\r\n");
        }
        return stringBuilder.ToString();
      }
    }
  }
```

2. We also add the attribute over the class declaration of Person.

```
[Uses(typeof(PropertyValueStringChainMixin<Person>))]
```

1.7.1 Questions

- We would need not to add a mixin to this. We could also add this method to the class Person too. What are the advantages by using mixins?
- Could you get the same advantages of mixins also with extension methods in this scenario?
- You might want to try out different parameters in the GetProperties method, such as

```
GetProperties(BindingFlags.Public|BindingFlags.NonPublic|BindingFlags.Instanc
e))
```

What happens`?

  - If you have tried it out, you will run into a StackOverflowExpection. Adding if (!propertyInfo.Name == "PropertyValues") after the for clause does not solve the problem. Why?
  - If you have solved the problem via if (!propertyInfo.Name.Contains("PropertyValues")) or something similar, you will see that the Salary is aways 1000. Why?

- Can you explain what the following constructor does in detail? What is <T>?

```
public class PropertyValueStringChainMixin<T> : Mixin<T>, IPropertyValue
where T : class
```

- Do you know what the keysword where means?

2 Lab Summary

In this lab, you have successfully done the following

- You created a mixin sample, where you extended a class person
- You used a mixin within the person sample to collect property information
- You added parameterlist to the construction.

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

- 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.
- 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

3 Common Practices

3.1 Getting re-motion

For all examples, the re-motion binaries are required. You can either build them or download them via the community page (https://www.re-motion.org/builds/)

If you are not sure where to put the binaries, please read Adding references

If you want to build re-motion, you have to get a version from the following subversion repository: https://svn.re-motion.org/svn/Remotion/

In most cases, the trunk build is sufficient. If you do not want to work with a trunk build, you might want to get the build with that this HOL was tested: https://svn.re-motion.org/svn/Remotion/tags/1.13.87

Please read „How to build.txt“(found in the base directory of the repository) before your build.

3.2 Adding references

There are several ways to add binaries in folder hierarchy in a project. We propose the following way for all re-motion HOLs:

Put them in a subfolder of your solution file alongside the subfolders for your projects:

- ./SolutionItems/References

Depending on the amount of different assemblies and product, you want to use on your project you might want to add subdirectories for each product too.

This has the advantage that you can also add other stuff such as Tools, Scripts, etc to your SolutionItems directory.