| GP 2 | Generative Programming | ST 17, Exercise 6 |
|---|---|---|

**Deadline: 23.06.2017, 09:40**

☒ **Gr. 1,** J. Karder, MSc.
☐ **Gr. 2,** P. Fleck, MSc.

**Name** Thomas Herzog **Effort in h** 4

**Points** _____ **Lecturer** _____

## 1. Dynamic Generation of XML Trees with the DLR (4 + 3 + 3 Points)

The Dynamic Language Runtime (DLR) API includes *DynamicObject*, a base class for specifying dynamic behavior at runtime.

a) Derive from this class and implement *XmlCreator*, a class that offers functionality for creating arbitrary XML trees at runtime. Realize this requirement by overriding *TryGetMember*, a method that is called each time a member of the respective class is accessed via a getter. For example, the following code in a C# program

```
String xml = XmlGenerator.Create()
             .Customer
               .BeginChildElements
                 .Name
                 .Phone
                 .Address.BeginChildElements
                   .Street
                   .City
                   .Zip
                 .EndChildElements
               .EndChildElements
               .Generate();
```

should lead to the generation of the following XML tree:

```
<Customer>
  <Name />
  <Phone />
  <Address>
    <Street />
    <City />
    <Zip />
  </Address>
</Customer>
```

Implement an auxiliary class *NodeGenerator* (also derived from *DynamicObject*) that is responsible for the correct handling of XML (child) node definitions.

b) Extend your XML creator so that the values of XML nodes can be defined at their creation. E.g., the following code shall lead to the definition of a XML tree with the given information:

```
String xml = XmlGenerator.Create()
             .Customer.BeginChildElements
               .Name("Franz")
               .Phone("033577021")
               .Address.BeginChildElements
                 .Street("Street 5")
                 .City("Linz")
                 .Zip("4020")
               .EndChildElements
             .EndChildElements
             .Generate();
```

c) Finally, extend your XML creator so that it also supports the generation of attributes. E.g., the following code

```
String xml = XmlGenerator.Create()
             .Customer.BeginChildElements
               .Name("Mayr", "FirstName", "Franz")
               .Phone("033577021", "CountryCode", "43")
               .Address.BeginChildElements
                 .Street("Street 5")
                 .City("Linz")
                 .Zip("4020")
               .EndChildElements
             .EndChildElements
             .Generate();
```

should lead to the generation of the following XML tree:

```
<Customer>
  <Name FirstName="Franz">Mayr</Name>
  <Phone CountryCode="43">033577021</Phone>
  <Address>
    <Street>Street 5</Street>
    <City>Linz</City>
    <Zip>4020</Zip>
  </Address>
</Customer>
```

## 2. Dynamic Generation of Proxies with the DLR (6 + 4 + 4 Points)

As we have already discussed the advantages and importance of the design pattern *Proxy* we will now implement a framework for generating proxies for arbitrary types at runtime using .NET and the Dynamic Language Runtime:

a) Use the .NET DLR for generating proxy objects at runtime. Implement the class *ProxyGenerator* that is derived from *DynamicMetaObject*, the base class for dynamic logic, and override the methods *BindGetMember*, *BindSetMember*, and *BindInvokeMember* for delegating the execution of called accesses and methods to the wrapped object.

b) Extend your implementation of dynamic proxies by implementing a mechanism for inducing arbitrary code before and after a call of the original object.

Again, use a simple interface *IInterception* that defines the methods *void Before()* and *void After()*, and extend the interface of the *Create* method of *ProxyGenerator* by a parameter of this type. The proxy object generated by the *ProxyGenerator* shall then call the methods *Before* and *After* each time a method is called and delegated to the encapsulated object.

c) Add around advice functionality to your implementation. *IInterception* should define another method *object Around(Delegate proceed, object[] args)*. Implementations of this method can call the *proceed* delegate to propagate method execution and have access to the method arguments in *args*. *Before()* and *After()* shall only be called if method execution is propagated. E.g., the following *IInterception* implementation

```
public void Before() {
  Console.WriteLine("before");
}

public void After() {
  Console.WriteLine("after");
}
```

```
public object Around(Delegate proceed, object[] args) {
  Console.WriteLine("around (before)");
  var result = proceed.DynamicInvoke();
  Console.WriteLine("around (after)");
  return result;
}
```

should produce the following output

```
around (before)
before
after
around (after)
```

You don't have to take care of argument changes prior to the execution of *proceed*, i.e. *proceed* can always use the arguments that were passed by the caller. It should be possible to change the *result* after the *proceed* call though.

Test your implementation by implementing a class *LogInterception* that writes simple log messages to the console.

*Please note:* - Test all implementations extensively

- Take care about the proper structuring of your code as well as comments

# 1 *Dynamic Generation of XML Trees with the DLR*

Dieser Abschnitt behandelt die Aufgabenstellung *Dynamic Generation of XML Trees with the DLR*.

## 1.1 Lösungsidee

Es wird die Klasse *DynamicNode* als Ableitung der Klasse *DynamicObject* implementiert, welche die XML Zeichenkette für die aktuelle *Node* generiert und die Generierung ebenfalls an seine Kind Knoten weiter delegiert, die wiederum ihre XML String Repräsentation generieren. Die generierte XML Zeichenkette wird mit einem *StringBuilder* aufgebaut. Die Klasse erstellt für jeden neuen *Member* einen Knoten als Objekt der Klasse *DynacmicNode*. Bei einem Methodenaufruf wird ebenfalls ein Kind Knoten erstellt, wobei die Argumente als XML Attribute behandelt werden. Wenn mehr als 3 Argumente angegeben wurden wird eine Ausnahme ausgelöst.

Es wird die Klasse *XmlGenerator* implementiert, die auch eine Ableitung der Klasse *DynamicObject* ist und für den ersten *Member*, den *Rootknoten* als Objekt der Klasse *DynamicNode* erstellt.

## 1.2 Quelltexte

Dieser Abschnitt beinhaltet die implementierten Quelltexte und das Testprogramm.

Listing 1: XmlGenerator.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Dynamic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace XMLGenerator
9  {
10     /// <summary>
11     /// The class represneting the dynamic object for the xml generator
12     /// </summary>
13     public class XmlGenerator : DynamicObject
14     {
15         /// <summary>
16         /// The root node
17         /// </summary>
18         private readonly DynamicNode _node = new DynamicNode();
19
20         private XmlGenerator()
21         {
22             // Not alolowed to be instantiated from outside this class
23         }
24
25         /// <summary>
26         /// Factory method for creating the XML generator instance
27         /// </summary>
28         /// <returns></returns>
29         public static dynamic Create()
30         {
31             return new XmlGenerator();
32         }
33
34         /// <summary>
35         /// Create a new node for the member
36         /// </summary>
37         /// <param name="binder">the binder providing the member name</param>
```

```
38          /// <param name="result">out parameter holding the newly create node</param>
39          /// <returns>always true, because will always create a new node</returns>
40          public override bool TryGetMember(GetMemberBinder binder, out object result)
41          {
42              result = new DynamicNode(binder.Name, null, _node);
43
44              return true;
45          }
46      }
47  }
```

Listing 2: DynamicNode.cs

```
1   using System;
2   using System.CodeDom;
3   using System.Collections.Generic;
4   using System.Dynamic;
5   using System.Linq;
6   using System.Text;
7   using System.Threading.Tasks;
8
9   namespace XMLGenerator
10  {
11      /// <summary>
12      /// This class represents a dynamic node of the xml generator
13      /// </summary>
14      public class DynamicNode : DynamicObject
15      {
16          private readonly DynamicNode _parent;
17          private readonly string _name;
18          private readonly string[] _value;
19          private readonly List<DynamicNode> _children = new List<DynamicNode>();
20
21          public DynamicNode BeginChildElements => new DynamicNode(null, null, this);
22          public DynamicNode EndChildElements => _parent;
23
24          /// <summary>
25          /// Constructing a xml node.
26          /// </summary>
27          /// <param name="name">the tag name</param>
28          /// <param name="value">the tag attributes and values.</param>
29          /// <param name="parent">the parent if this is a child node</param>
30          public DynamicNode(string name = null, string[] value = null, DynamicNode parent = null)
31          {
32              _name = name;
33              _value = value ?? new string[0];
34              _parent = parent;
35              _parent?._children.Add(this);
36          }
37
38          /// <summary>
39          /// Generates the xml structure from this node on through all of its children
40          /// </summary>
41          /// <returns>the generated xml structure in form of an string</returns>
42          public string Generate()
43          {
44              var sb = new StringBuilder();
45
46              Generate(sb);
47
48              return sb.ToString();
49          }
```

```
50
51          /// <summary>
52          /// Creates a new node for the addressed member.
53          /// </summary>
54          /// <param name="binder">the binder which provides the member name</param>
55          /// <param name="result">the out parameter representing the newly created node</param>
56          /// <returns>always true because always a new node gets generated</returns>
57          public override bool TryGetMember(GetMemberBinder binder, out object result)
58          {
59              result = new DynamicNode(binder.Name, null, _parent);
60              return true;
61          }
62
63          /// <summary>
64          /// Creates a new node for the addressed method.
65          /// </summary>
66          /// <param name="binder">the binder providing the method name</param>
67          /// <param name="args">the arguments for the method which represent the value of the
    ↪   node</param>
68          /// <param name="result">the out parameter holding the newly create node</param>
69          /// <returns>always true because always a new node gets generated</returns>
70          public override bool TryInvokeMember(InvokeMemberBinder binder, object[] args, out object
    ↪   result)
71          {
72              if ((args != null) && (args.Length > 3))
73              {
74                  throw new ArgumentException(
75                      $"Member invocation with to many arguments found. maxCount=3,
    ↪   foundCount={args.Length}");
76              }
77              string[] attributes = args?.Select(arg => arg.ToString()).ToArray();
78
79              result = new DynamicNode(binder.Name, attributes, _parent);
80              return true;
81          }
82
83          /// <summary>
84          /// Private helper which generates the xml string for this and all of its child nodes.
85          /// </summary>
86          /// <param name="sb">the string builder to append the generated nodes on</param>
87          private void Generate(StringBuilder sb)
88          {
89              bool nameNotPresent = string.IsNullOrEmpty(_name);
90              bool valuePresent = _value.Any();
91              bool hasChildren = _children.Any();
92
93              // only if the node defines a tag name, which root node doesn't
94              if (!nameNotPresent)
95              {
96                  sb.Append($"<{_name}");
97                  // If more than 1 valie is present the tag has a attribute
98                  if (valuePresent && (_value.Length >= 2))
99                  {
100                     sb.Append($" {_value[1]}=\"");
101                     // if 3 values are present then the tag attribute has a value
102                     if (_value.Length == 3)
103                     {
104                         sb.Append(_value[2]);
105                     }
106                     sb.Append("\"");
107                 }
108                 // Empty tag only if no values and children are presnet
109                 if (!valuePresent  && !hasChildren)
```

```
110                 {
111                     sb.Append("/>").Append(Environment.NewLine);
112                 }
113                 else
114                 {
115                     sb.Append(">");
116                 }
117             }
118             // Add tag value if present
119             if (valuePresent)
120             {
121                 sb.Append(_value[0]);
122             }
123 
124             // Generate xml for child nodes
125             _children.ForEach(child => child.Generate(sb));
126 
127             // End tag if no empty tag or had children
128             if (!nameNotPresent && (valuePresent || hasChildren))
129             {
130                 sb.Append($"</{_name}>").Append(Environment.NewLine);
131             }
132         }
133     }
134 }
```

Listing 3: Program.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6 
7  namespace XMLGenerator
8  {
9      class Program
10     {
11         static void Main(string[] args)
12         {
13             Console.WriteLine("-----------------------------------------------");
14             Console.WriteLine("XMLGenerator tests");
15             Console.WriteLine("-----------------------------------------------");
16             Console.WriteLine("testExampleNoValuesAndNoAttributes:");
17             Console.WriteLine(testExampleNoValuesAndNoAttributes());
18             Console.WriteLine("-----------------------------------------------");
19             Console.WriteLine("testExampleValuesNoAttributes:");
20             Console.WriteLine(testExampleValuesNoAttributes());
21             Console.WriteLine("-----------------------------------------------");
22             Console.WriteLine("testExampleValuesAndAttributes:");
23             Console.WriteLine(testExampleValuesAndAttributes());
24             Console.WriteLine("-----------------------------------------------");
25         }
26 
27         private static string testExampleNoValuesAndNoAttributes()
28         {
29             return XmlGenerator.Create()
30                 .Customer
31                 .BeginChildElements
32                 .Name
33                 .Phone
34                 .Address.BeginChildElements
```

```
35                  .Street
36                  .City
37                  .Zip
38                  .EndChildElements
39                  .EndChildElements
40                  .Generate();
41          }
42
43          private static string testExampleValuesNoAttributes()
44          {
45              return XmlGenerator.Create()
46                  .Customer.BeginChildElements
47                  .Name("Franz")
48                  .Phone("033577021")
49                  .Address.BeginChildElements
50                  .Street("Street 5")
51                  .City("Linz")
52                  .Zip("4020")
53                  .EndChildElements
54                  .EndChildElements
55                  .Generate();
56          }
57
58          private static string testExampleValuesAndAttributes()
59          {
60              return XmlGenerator.Create()
61                  .Customer.BeginChildElements
62                  .Name("Mayr", "FirstName", "Franz")
63                  .Phone("033577021", "CountryCode", "43")
64                  .Address.BeginChildElements
65                  .Street("Street 5")
66                  .City("Linz")
67                  .Zip("4020")
68                  .EndChildElements
69                  .EndChildElements
70                  .Generate();
71          }
72      }
73  }
```

### 1.3 Tests

Dieser Abschnitt behandelt die Tests in Form von Ausgaben der *Logs*.

```
-----------------------------------------------
XMLGenerator tests
-----------------------------------------------
testExampleNoValuesAndNoAttributes:
<Customer><Name/>
<Phone/>
<Address><Street/>
<City/>
<Zip/>
</Address>
</Customer>

-----------------------------------------------
testExampleValuesNoAttributes:
<Customer><Name>Franz</Name>
<Phone>033577021</Phone>
<Address><Street>Street 5</Street>
<City>Linz</City>
<Zip>4020</Zip>
</Address>
</Customer>

-----------------------------------------------
testExampleValuesAndAttributes:
<Customer><Name FirstName="Franz">Mayr</Name>
<Phone CountryCode="43">033577021</Phone>
<Address><Street>Street 5</Street>
<City>Linz</City>
<Zip>4020</Zip>
</Address>
</Customer>

-----------------------------------------------
```

Abbildung 1: Test für die XML Generierung

# 2 *Dynamic Generation of Proxies with the DLR*

Dieser Abschnitt behandelt die Aufgabenstellung *Dynamic Generation of Proxies with the DLR*.

## 2.1 Lösungsidee

Es wird die Klasse *ProxyGenerator* implementiert, welche eine Ableitung der Klasse *DynamicMetaObject* ist. Diese Klasse wird die Aspekte für das dynamische Objekt einweben, aber nur wenn ein Objekt der Schnittstelle *IInterception* dieser Klasse übergeben wurde.

Es wird die Schnittstelle *IInterception* implementiert, welche einen *Interceptor* spezifiziert. Als Implementierung der Schnittstelle *IInterception* wird die Klasse *LogInterceptor* implementiert, die *Properties* vom Datentyp *bool* bereitstellt, mit denen das Ausführen der Methode und das Ersetzen des Resultats gesteuert werden kann. Ebenso wird ein *Property* vom Datentyp *object* bereitgestellt, mit dem das Resultat ersetzt werden soll, aber nur wenn der *bool Property* für das Ersetzen gesetzt wurde.

Es wird die Klasse *DynamicTest* implementiert, welche eine Ableitung der Klasse *IDynamicMetaObjectProvider* ist, wobei diese Implementierung das Metaobjekt erzeugt, das in ein Objekt der Klasse *ProxyGenerator* ist.

## 2.2 Quelltexte

Dieser Abschnitt beinhaltet die implementierten Quelltexte und das Testprogramm.

Listing 4: IInterception.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace ProxyGenerator
8  {
9      /// <summary>
10     /// This interface specifies an interceptor
11     /// </summary>
12     public interface IInterception
13     {
14         /// <summary>
15         /// Called before the method invocation
16         /// </summary>
17         void Before();
18
19         /// <summary>
20         /// Called after the method invocation
21         /// </summary>
22         void After();
23
24         /// <summary>
25         /// Around method which acn prevent method to be called
26         /// </summary>
27         /// <param name="proceed">the delegate representing the object the mthod was called
   ↪  on</param>
28         /// <param name="args">the mthod arguments</param>
29         /// <returns>the result, null if tis an void method</returns>
30         object Around(Delegate proceed, object[] args);
31     }
32 }
```

Listing 5: LogInterception.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ProxyGenerator
{
    /// <summary>
    /// The log interceptor implementation for IInterception
    /// </summary>
    public class LogInterception : IInterception
    {
        public bool ProceedEnabled { get; set; }
        public bool EnforceResultEnabled { get; set; }
        public object EnforcedResult { get; set; }

        public void Before()
        {
            Console.WriteLine("Before called");
        }

        public void After()
        {
            Console.WriteLine("After called");
        }

        public object Around(Delegate proceed, object[] args)
        {
            object result = null;
            Console.WriteLine("around (before)");

            if (ProceedEnabled)
            {
                Console.WriteLine("around (procceding)");
                result = proceed.DynamicInvoke();
                if (EnforceResultEnabled)
                {
                    Console.WriteLine("around (enforced result)");
                    result = EnforcedResult;
                }
                else
                {
                    Console.WriteLine("around (result not enforced)");
                }
                Console.WriteLine("around (proceeded)");
            }
            else
            {
                Console.WriteLine("around (proceed disabled)");
            }

            Console.WriteLine("around (after)");

            return result;
        }
    }
}
```

Listing 6: ProxyGenerator.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Dynamic;
4  using System.Linq;
5  using System.Linq.Expressions;
6  using System.Text;
7  using System.Threading.Tasks;
8  using Microsoft.CSharp.RuntimeBinder;
9
10 namespace ProxyGenerator
11 {
12     /// <summary>
13     /// This class is the proxy generator.
14     /// </summary>
15     public class ProxyGenerator : DynamicMetaObject
16     {
17         private readonly IInterception _interceptor;
18
19         /// <summary>
20         /// Constructs a proxy generator
21         /// </summary>
22         /// <param name="expression">the expression to proxy</param>
23         /// <param name="obj">the object the method gets called</param>
24         /// <param name="interceptor">the interceptor, can be null</param>
25         public ProxyGenerator(Expression expression, object obj, IInterception interceptor) :
   ↪  base(expression,
26             BindingRestrictions.Empty, obj)
27         {
28             _interceptor = interceptor;
29         }
30
31         public override DynamicMetaObject BindGetMember(GetMemberBinder binder)
32         {
33             return WeaveAspect(base.BindGetMember(binder));
34         }
35
36         public override DynamicMetaObject BindInvokeMember(InvokeMemberBinder binder,
   ↪  DynamicMetaObject[] args)
37         {
38             return WeaveAspect(base.BindInvokeMember(binder, args), args);
39         }
40
41         private DynamicMetaObject WeaveAspect(DynamicMetaObject obj)
42         {
43             return WeaveAspect(obj, new DynamicMetaObject[0]);
44         }
45
46         private DynamicMetaObject WeaveAspect(DynamicMetaObject origObj, DynamicMetaObject[] args)
47         {
48             // only if interception is provided
49             if (_interceptor != null)
50             {
51                 // original expressions
52                 var origExpr = origObj.Expression;
53                 var origVal = Expression.Parameter(origExpr.Type);
54
55                 // before/after expressions
56                 var beforeExpr = Expression.Call(Expression.Constant(_interceptor),
57                     typeof(IInterception).GetMethod(nameof(IInterception.Before)));
58                 var afterExpr = Expression.Call(Expression.Constant(_interceptor),
59                     typeof(IInterception).GetMethod(nameof(IInterception.After)));
```

```
60
61                  // new execution block expression
62                  var block = Expression.Block(
63                      new[] {origVal},
64                      beforeExpr,
65                      Expression.Assign(origVal, origExpr),
66                      afterExpr,
67                      origVal
68                  );
69
70                  // proceed expression
71                  var proceedExpressions = Expression.Lambda(Expression.Block(new
    ↪  ParameterExpression[] { }, block));
72
73                  // around expression
74                  var aroundExpr = Expression.Call(Expression.Constant(_interceptor),
75                      typeof(IInterception).GetMethod(nameof(IInterception.Around)),
76                      new List<Expression>
77                      {
78                          proceedExpressions,
79                          Expression.NewArrayInit(typeof(object),
80                              args.Select(x => Expression.Convert(x.Expression, typeof(object))))
81                      });
82
83                  return new DynamicMetaObject(aroundExpr, origObj.Restrictions);
84              }
85
86              return origObj;
87          }
88      }
89  }
```

Listing 7: DynamicTest.cs

```
1   using System;
2   using System.Collections.Generic;
3   using System.Dynamic;
4   using System.Linq;
5   using System.Linq.Expressions;
6   using System.Text;
7   using System.Threading.Tasks;
8
9   namespace ProxyGenerator
10  {
11      /// <summary>
12      /// A test implementation of a proxied dnymic meta object class
13      /// </summary>
14      public class DynamicTest : IDynamicMetaObjectProvider
15      {
16          private IInterception _interceptor;
17
18          public DynamicTest(IInterception interceptor = null)
19          {
20              _interceptor = interceptor;
21          }
22
23          public int DoSomething(int val, params string[] args)
24          {
25              Console.WriteLine($"Called: int DoSomething({val}, {string.Join(", ", args)})");
26              return val;
27          }
28
```

```
29          public DynamicMetaObject GetMetaObject(Expression parameter)
30          {
31              return new ProxyGenerator(parameter, this, _interceptor);
32          }
33      }
34  }
```

Listing 8: Program.cs

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace ProxyGenerator
8   {
9       /// <summary>
10      /// The main program for the tests
11      /// </summary>
12      class Program
13      {
14          /// <summary>
15          /// MAin holding the test code
16          /// </summary>
17          /// <param name="args">the arguments for the man method</param>
18          static void Main(string[] args)
19          {
20              var interceptor = new LogInterception();
21              dynamic obj = new DynamicTest();
22              dynamic interceptedObj = new DynamicTest(interceptor);
23
24              Console.WriteLine("-----------------------------------------");
25              Console.WriteLine($"Val: {1} / no interceptor");
26              AssertEquals(1, (int) obj.DoSomething(1, "firstArg", "secondArg", "thirdArg"));
27              Console.WriteLine("-----------------------------------------");
28
29              Console.WriteLine("-----------------------------------------");
30              Console.WriteLine($"All disabled");
31              interceptedObj.DoSomething(1, "firstArg", "secondArg", "thirdArg");
32              Console.WriteLine("-----------------------------------------");
33
34              Console.WriteLine("-----------------------------------------");
35              Console.WriteLine($"Val: {1} / proceed enabled");
36              interceptor.ProceedEnabled = true;
37              AssertEquals(1, (int) interceptedObj.DoSomething(1, "firstArg", "secondArg",
    ↪   "thirdArg"));
38              Console.WriteLine("-----------------------------------------");
39
40              Console.WriteLine("-----------------------------------------");
41              Console.WriteLine($"Val: {10} / proceed - enforce enabled");
42              interceptor.ProceedEnabled = true;
43              interceptor.EnforceResultEnabled = true;
44              interceptor.EnforcedResult = 10;
45              AssertEquals(10, (int) interceptedObj.DoSomething(11, "firstArg", "secondArg",
    ↪   "thirdArg"));
46              Console.WriteLine("-----------------------------------------");
47
48              Console.WriteLine("-----------------------------------------");
49              Console.WriteLine($"Val: 'I am the enforced result' / proceed - enforce enabled");
50              interceptor.ProceedEnabled = true;
51              interceptor.EnforceResultEnabled = true;
```

```
52              interceptor.EnforcedResult = "I am the enforced result";
53              AssertEquals(interceptor.EnforcedResult,
54                  (string) interceptedObj.DoSomething(1, "firstArg", "secondArg", "thirdArg"));
55              Console.WriteLine("-----------------------------------------");
56
57              Console.WriteLine("-----------------------------------------");
58              Console.WriteLine($"Val: 'null' / proceed - enforce enabled");
59              interceptor.ProceedEnabled = true;
60              interceptor.EnforceResultEnabled = true;
61              interceptor.EnforcedResult = null;
62              AssertEquals(interceptor.EnforcedResult,
63                  interceptedObj.DoSomething(1, "firstArg", "secondArg", "thirdArg"));
64              Console.WriteLine("-----------------------------------------");
65          }
66
67          /// <summary>
68          /// Asserts the results
69          /// </summary>
70          /// <param name="expected">the expected value</param>
71          /// <param name="actual">the actual value</param>
72          private static void AssertEquals(object expected, object actual)
73          {
74              if (expected == null && actual == null)
75              {
76                  Console.WriteLine($"Testing: null == null = True");
77              }
78              else if (expected != null && actual == null)
79              {
80                  Console.WriteLine($"Testing: {expected} == null = False");
81              }
82              else if (expected == null && actual != null)
83              {
84                  Console.WriteLine($"Testing: null == {actual} = False");
85              }
86              else
87              {
88                  Console.WriteLine($"Testing: {expected} == {actual} =
    ↪  {expected?.Equals(actual)}");
89              }
90          }
91      }
92  }
```

## 2.3 Tests

Dieser Abschnitt behandelt die Tests in Form von Ausgaben der *Logs*.

```
-----------------------------------------
Val: 1 / no interceptor
Called: int DoSomething(1, firstArg, secondArg, thirdArg)
Testing: 1 == 1 = True
-----------------------------------------
-----------------------------------------
All disabled
around (before)
around (proceed disabled)
around (after)
-----------------------------------------
-----------------------------------------
Val: 1 / proceed enabled
around (before)
around (procceding)
Before called
Called: int DoSomething(1, firstArg, secondArg, thirdArg)
After called
around (result not enforced)
around (proceeded)
around (after)
Testing: 1 == 1 = True
-----------------------------------------
-----------------------------------------
Val: 10 / proceed - enforce enabled
around (before)
around (procceding)
Before called
Called: int DoSomething(11, firstArg, secondArg, thirdArg)
After called
around (enforced result)
around (proceeded)
around (after)
Testing: 10 == 10 = True
-----------------------------------------
-----------------------------------------
Val: 'I am the enforced result' / proceed - enforce enabled
around (before)
around (procceding)
Before called
Called: int DoSomething(1, firstArg, secondArg, thirdArg)
After called
around (enforced result)
around (proceeded)
around (after)
Testing: I am the enforced result == I am the enforced result = True
-----------------------------------------
-----------------------------------------
Val: 'null' / proceed - enforce enabled
around (before)
around (procceding)
Before called
Called: int DoSomething(1, firstArg, secondArg, thirdArg)
After called
around (enforced result)
around (proceeded)
around (after)
Testing: null == null = True
-----------------------------------------
```

Abbildung 2: Test für die Proxy Generierung