1.1 ./csharp/Platform.Setters/SetterBase.cs

```
using System.Runtime.CompilerServices;
using Platform.Interfaces;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.Setters
{
    /// <summary>
    /// <para>Represents a base implementation for an setter that allows you to set a passed value as the result value.</para>
    /// <para>Представляет базовую реализацию для установщика, который позволяет установить переданное ему значение в качестве результирующего значения.</para>
    /// </summary>
    /// <typeparam name="TResult"><para>The type of result value.</para><para>Тип результирующего значения.</para></typeparam>
    /// <remarks>
    /// Must be class, not struct (in order to persist access to Result property value).
    /// </remarks>
    public abstract class SetterBase<TResult> : ISetter<TResult>
    {
        /// <summary>
        /// <para>Represents the result value.</para>
        /// <para>Представляет результирующие значение.</para>
        /// </summary>
        protected TResult _result;

        /// <summary>
        /// <para>Gets result value.</para>
        /// <para>Возвращает результирующее значение.</para>
        /// </summary>
        public TResult Result => _result;

        /// <summary>
        /// <para>Initializes a new instance of the SetterBase class.</para>
        /// <para>Инициализирует новый экземпляр класса SetterBase.</para>
        /// </summary>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected SetterBase() { }

        /// <summary>
        /// <para>Initializes a new instance of the SetterBase class using the passed-in value as the default result value.</para>
        /// <para>Инициализирует новый экземпляр класса SetterBase, используя переданное значение в качестве результирующего по умолчанию.</para>
        /// </summary>
        /// <param name="defaultValue"><para>The default result value.</para><para>Результирующее значение по умолчанию.</para></param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected SetterBase(TResult defaultValue) => _result = defaultValue;

        /// <summary>
        /// <para>Sets the passed value as the result.</para>
        /// <para>Устанавливает переданное значение в качестве результирующего.</para>
        /// </summary>
        /// <param name="value"><para>The result value.</para><para>Результирующее значение.</para></param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public void Set(TResult value) => _result = value;
    }
}
```

1.2 ./csharp/Platform.Setters/SetterExtensions.cs

```
using System.Collections.Generic;

namespace Platform.Setters
{
    public static class SetterExtensions
    {
        public static TDecision SetFirstFromNonNullListAndReturnTrue<TResult, TDecision>(this Setter<TResult, TDecision> setter, IList<TResult>? list)
        {
            if (list != null)
            {
                setter.Set(list[0]);
            }
            return setter.TrueValue;
```

```
14          }
15
16          public static TDecision SetFirstFromNonNullFirstListAndReturnTrue<TResult,
            ↪  TDecision>(this Setter<TResult, TDecision> setter, IList<TResult>? list1,
            ↪  IList<TResult>? list2)
17          {
18              if (list1 != null)
19              {
20                  setter.Set(list1[0]);
21              }
22              return setter.TrueValue;
23          }
24
25          public static TDecision SetSecondFromNonNullFirstListAndReturnTrue<TResult,
            ↪  TDecision>(this Setter<TResult, TDecision> setter, IList<TResult>? list1,
            ↪  IList<TResult>? list2)
26          {
27              if (list1 != null)
28              {
29                  setter.Set(list1[1]);
30              }
31              return setter.TrueValue;
32          }
33
34          public static TDecision SetThirdFromNonNullFirstListAndReturnTrue<TResult,
            ↪  TDecision>(this Setter<TResult, TDecision> setter, IList<TResult>? list1,
            ↪  IList<TResult>? list2)
35          {
36              if (list1 != null)
37              {
38                  setter.Set(list1[2]);
39              }
40              return setter.TrueValue;
41          }
42
43          public static TDecision SetFirstFromNonNullSecondListAndReturnTrue<TResult,
            ↪  TDecision>(this Setter<TResult, TDecision> setter, IList<TResult>? list1,
            ↪  IList<TResult>? list2)
44          {
45              if (list2 != null)
46              {
47                  setter.Set(list2[0]);
48              }
49              return setter.TrueValue;
50          }
51
52          public static TDecision SetSecondFromNonNullSecondListAndReturnTrue<TResult,
            ↪  TDecision>(this Setter<TResult, TDecision> setter, IList<TResult>? list1,
            ↪  IList<TResult>? list2)
53          {
54              if (list2 != null)
55              {
56                  setter.Set(list2[1]);
57              }
58              return setter.TrueValue;
59          }
60
61          public static TDecision SetThirdFromNonNullSecondListAndReturnTrue<TResult,
            ↪  TDecision>(this Setter<TResult, TDecision> setter, IList<TResult>? list1,
            ↪  IList<TResult>? list2)
62          {
63              if (list2 != null)
64              {
65                  setter.Set(list2[2]);
66              }
67              return setter.TrueValue;
68          }
69          public static TDecision SetFirstFromNonNullListAndReturnFalse<TResult, TDecision>(this
            ↪  Setter<TResult, TDecision> setter, IList<TResult>? list)
70          {
71              if (list != null)
72              {
73                  setter.Set(list[0]);
74              }
75              return setter.FalseValue;
76          }
77
```

```
78          public static TDecision SetFirstFromNonNullFirstListAndReturnFalse<TResult,
            ↪  TDecision>(this Setter<TResult, TDecision> setter, IList<TResult>? list1,
            ↪  IList<TResult>? list2)
79          {
80              if (list1 != null)
81              {
82                  setter.Set(list1[0]);
83              }
84              return setter.FalseValue;
85          }
86
87          public static TDecision SetSecondFromNonNullFirstListAndReturnFalse<TResult,
            ↪  TDecision>(this Setter<TResult, TDecision> setter, IList<TResult>? list1,
            ↪  IList<TResult>? list2)
88          {
89              if (list1 != null)
90              {
91                  setter.Set(list1[1]);
92              }
93              return setter.FalseValue;
94          }
95
96          public static TDecision SetThirdFromNonNullFirstListAndReturnFalse<TResult,
            ↪  TDecision>(this Setter<TResult, TDecision> setter, IList<TResult>? list1,
            ↪  IList<TResult>? list2)
97          {
98              if (list1 != null)
99              {
100                 setter.Set(list1[2]);
101             }
102             return setter.FalseValue;
103         }
104
105         public static TDecision SetFirstFromNonNullSecondListAndReturnFalse<TResult,
            ↪  TDecision>(this Setter<TResult, TDecision> setter, IList<TResult>? list1,
            ↪  IList<TResult>? list2)
106         {
107             if (list2 != null)
108             {
109                 setter.Set(list2[0]);
110             }
111             return setter.FalseValue;
112         }
113
114         public static TDecision SetSecondFromNonNullSecondListAndReturnFalse<TResult,
            ↪  TDecision>(this Setter<TResult, TDecision> setter, IList<TResult>? list1,
            ↪  IList<TResult>? list2)
115         {
116             if (list2 != null)
117             {
118                 setter.Set(list2[1]);
119             }
120             return setter.FalseValue;
121         }
122
123         public static TDecision SetThirdFromNonNullSecondListAndReturnFalse<TResult,
            ↪  TDecision>(this Setter<TResult, TDecision> setter, IList<TResult>? list1,
            ↪  IList<TResult>? list2)
124         {
125             if (list2 != null)
126             {
127                 setter.Set(list2[2]);
128             }
129             return setter.FalseValue;
130         }
131     }
132 }
```

1.3 ./csharp/Platform.Setters/Setter[TResult, TDecision].cs

```
1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Setters
7  {
8      /// <summary>
```

```
9       /// <para>Represents implementation for an setter that allows you to set a passed value as
            ↪ the result value. This setter implementation has additional methods that, simultaneously
            ↪ with setting the result value, return <typeparamref name="TDecision"/> values indicating
            ↪ true or false.</para>
10      /// <para>Представляет реализацию для установщика, который позволяет установить переданное
            ↪ ему значение в качестве результирующего значения. В этой реализации установщика есть
            ↪ дополнительные методы, которые одновременно с установкой результирующего значения
            ↪ возвращают значения типа <typeparamref name="TDecision"/>, обозначающие истину или
            ↪ ложь.</para>
11      /// </summary>
12      /// <typeparam name="TResult"><para>The type of result value.</para><para>Тип
            ↪ результирующего значения.</para></typeparam>
13      /// <typeparam name="TDecision"><para>The type of value which will be used to make the
            ↪ decision.</para><para>Тип значения на основе которого будет приниматься
            ↪ решение.</para></typeparam>
14      public class Setter<TResult, TDecision> : SetterBase<TResult>
15      {
16          public readonly TDecision TrueValue;
17          public readonly TDecision FalseValue;
18
19          /// <summary>
20          /// <para>Initializes a new instance of the Setter class using the passed-in value as
                ↪ the default result value.</para>
21          /// <para>Инициализирует новый экземпляр класса Setter, используя переданные значения
                ↪ trueValue, falseValue, defaultValue в качестве результирующего по умолчанию.</para>
22          /// </summary>
23          /// <param name="defaultValue"><para>The default result
                ↪ value.</para><para>Результирующее значение по умолчанию.</para></param>
24          [MethodImpl(MethodImplOptions.AggressiveInlining)]
25          public Setter(TDecision trueValue, TDecision falseValue, TResult defaultValue)
26              : base(defaultValue)
27          {
28              TrueValue = trueValue;
29              FalseValue = falseValue;
30          }
31
32          /// <summary>
33          /// <para>Gets result value.</para>
34          /// <para>Возвращает результирующее значение.</para>
35          /// </summary>
36          [MethodImpl(MethodImplOptions.AggressiveInlining)]
37          public Setter(TDecision trueValue, TDecision falseValue) : this(trueValue, falseValue,
                ↪ default) { }
38
39          /// <summary>
40          /// <para>Gets result value.</para>
41          /// <para>Возвращает результирующее значение.</para>
42          /// </summary>
43          [MethodImpl(MethodImplOptions.AggressiveInlining)]
44          public Setter(TResult defaultValue) : base(defaultValue) { }
45
46          /// <summary>
47          /// <para>Gets result value.</para>
48          /// <para>Возвращает результирующее значение.</para>
49          /// </summary>
50          [MethodImpl(MethodImplOptions.AggressiveInlining)]
51          public Setter() { }
52
53          /// <summary>
54          /// <para>Gets result value.</para>
55          /// <para>Возвращает результирующее значение.</para>
56          /// </summary>
57          [MethodImpl(MethodImplOptions.AggressiveInlining)]
58          public TDecision SetAndReturnTrue(TResult value)
59          {
60              _result = value;
61              return TrueValue;
62          }
63
64          /// <summary>
65          /// <para>Gets result value.</para>
66          /// <para>Возвращает результирующее значение.</para>
67          /// </summary>
68          [MethodImpl(MethodImplOptions.AggressiveInlining)]
69          public TDecision SetAndReturnFalse(TResult value)
70          {
71              _result = value;
72              return FalseValue;
```

```
        }

        /// <summary>
        /// <para>Gets result value.</para>
        /// <para>Возвращает результирующее значение.</para>
        /// </summary>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public TDecision SetFirstAndReturnTrue(IList<TResult>? list)
        {
            if (list != null)
            {
                _result = list[0];
            }
            return TrueValue;
        }

        /// <summary>
        /// <para>Gets result value.</para>
        /// <para>Возвращает результирующее значение.</para>
        /// </summary>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public TDecision SetFirstAndReturnFalse(IList<TResult>? list)
        {
            if (list != null)
            {
                _result = list[0];
            }
            return FalseValue;
        }
    }
}
```

1.4 ./csharp/Platform.Setters/Setter[TResult].cs

```
using System.Runtime.CompilerServices;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.Setters
{
    /// <summary>
    /// <para>
    /// Represents the setter.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="Setter{TResult, bool}"/>
    public class Setter<TResult> : Setter<TResult, bool>
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see cref="Setter"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="defaultValue">
        /// <para>A default value.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public Setter(TResult defaultValue) : base(true, false, defaultValue) { }

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="Setter"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public Setter() : base(true, false) { }
    }
}
```

1.5 ./csharp/Platform.Setters.Tests/SetterTests.cs

```
using Xunit;

namespace Platform.Setters.Tests
{
    public class SetterTests
```

```
    {
        [Fact]
        public void ParameterlessConstructedSetterTest()
        {
            Setter<int> setter = new Setter<int>();
            Assert.Equal(default, setter.Result);
        }

        [Fact]
        public void ConstructedWithDefaultValueSetterTest()
        {
            Setter<int> setter = new Setter<int>(9);
            Assert.Equal(9, setter.Result);
        }

        [Fact]
        public void MethodsWithBooleanReturnTypeTest()
        {
            Setter<int> setter = new Setter<int>();
            Assert.True(setter.SetAndReturnTrue(1));
            Assert.Equal(1, setter.Result);
            Assert.False(setter.SetAndReturnFalse(2));
            Assert.Equal(2, setter.Result);
            Assert.True(setter.SetFirstAndReturnTrue(new int[] { 3 }));
            Assert.Equal(3, setter.Result);
            Assert.False(setter.SetFirstAndReturnFalse(new int[] { 4 }));
            Assert.Equal(4, setter.Result);
        }

        [Fact]
        public void MethodsWithIntegerReturnTypeTest()
        {
            Setter<int, int> setter = new Setter<int, int>(1, 0);
            Assert.Equal(1, setter.SetAndReturnTrue(1));
            Assert.Equal(1, setter.Result);
            Assert.Equal(0, setter.SetAndReturnFalse(2));
            Assert.Equal(2, setter.Result);
            Assert.Equal(1, setter.SetFirstAndReturnTrue(new int[] { 3 }));
            Assert.Equal(3, setter.Result);
            Assert.Equal(0, setter.SetFirstAndReturnFalse(new int[] { 4 }));
            Assert.Equal(4, setter.Result);
        }
    }
}
```

Index