1.1 ./Platform.Disposables/Disposable.cs

```
using System;

namespace Platform.Disposables
{
    /// <summary>
    /// <para>Represents disposable object that contains OnDispose event which is raised when the object itself is disposed.</para>
    /// <para>Представляет высвобождаемый объект, который содержит событие OnDispose, которое возникает при высвобождении самого объекта.</para>
    /// </summary>
    public class Disposable : DisposableBase
    {
        private static readonly Disposal _emptyDelegate = (manual, wasDisposed) => { };

        /// <summary>
        /// <para>Occurs when the object is being disposed.</para>
        /// <para>Возникает, когда объект высвобождается.</para>
        /// </summary>
        public event Disposal OnDispose;

        /// <summary>
        /// <para>Initializes a new instance of the <see cref="Disposable"/> object.</para>
        /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable"/>.</para>
        /// </summary>
        /// <param name="action"><para>The <see cref="Action"/> delegate.</para><para>Делегат <see cref="Action"/>.</para></param>
        public Disposable(Action action)
        {
            OnDispose = (manual, wasDisposed) =>
            {
                if (!wasDisposed)
                {
                    action();
                }
            };
        }

        /// <summary>
        /// <para>Initializes a new instance of the <see cref="Disposable"/> object.</para>
        /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable"/>.</para>
        /// </summary>
        /// <param name="disposal"><para>The <see cref="Disposal"/> delegate.</para><para>Делегат <see cref="Disposal"/>.</para></param>
        public Disposable(Disposal disposal) => OnDispose = disposal;

        /// <summary>
        /// <para>Initializes a new instance of the <see cref="Disposable"/> object.</para>
        /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable"/>.</para>
        /// </summary>
        public Disposable() => OnDispose = _emptyDelegate;

        /// <summary>
        /// <para>Creates a new <see cref="Disposable"/> object initialized with specified delegate <see cref="Action"/>.</para>
        /// <para>Создает новый объект <see cref="Disposable"/>, инициализированную с помощью указанного делегата <see cref="Action"/>.</para>
        /// </summary>
        /// <param name="action"><para>The <see cref="Action"/> delegate.</para><para>Делегат <see cref="Action"/>.</para></param>
        public static implicit operator Disposable(Action action) => new Disposable(action);

        /// <summary>
        /// <para>Creates a new <see cref="Disposable"/> object initialized with specified delegate <see cref="Disposal"/>.</para>
        /// <para>Создает новый объект <see cref="Disposable"/>, инициализированную с помощью указанного делегата <see cref="Disposal"/>.</para>
        /// </summary>
        /// <param name="disposal"><para>The <see cref="Disposal"/> delegate.</para><para>Делегат <see cref="Disposal"/>.</para></param>
        public static implicit operator Disposable(Disposal disposal) => new Disposable(disposal);

        /// <summary>
        /// <para>Disposes unmanaged resources.</para>
        /// <para>Высвобождает неуправляемые ресурсы.</para>
```

```
65          /// </summary>
66          /// <param name="manual">
67          /// <para>A value that determines whether the disposal was triggered manually (by the
            ↪  developer's code) or was executed automatically without an explicit indication from
            ↪  the developer.</para>
68          /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
            ↪  разработчика) или же выполнилось автоматически без явного указания со стороны
            ↪  разработчика.</para>
69          /// </param>
70          /// <param name="wasDisposed">
71          /// <para>A value that determines whether the object was released before calling this
            ↪  method.</para>
72          /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
73          /// </param>
74          protected override void Dispose(bool manual, bool wasDisposed) =>
            ↪  RaiseOnDisposeEvent(manual, wasDisposed);
75
76          /// <summary>
77          /// <para>Raises an unmanaged resource dispose event.</para>
78          /// <para>Генерирует событие высвобождения неуправляемых ресурсов.</para>
79          /// </summary>
80          /// <param name="manual">
81          /// <para>A value that determines whether the disposal was triggered manually (by the
            ↪  developer's code) or was executed automatically without an explicit indication from
            ↪  the developer.</para>
82          /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
            ↪  разработчика) или же выполнилось автоматически без явного указания со стороны
            ↪  разработчика.</para>
83          /// </param>
84          /// <param name="wasDisposed">
85          /// <para>A value that determines whether the object was released before calling this
            ↪  method.</para>
86          /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
87          /// </param>
88          protected void RaiseOnDisposeEvent(bool manual, bool wasDisposed) => OnDispose(manual,
            ↪  wasDisposed);
89
90          /// <summary>
91          /// <para>Attempts to dispose the specified object, as well as set the value of the
            ↪  variable containing this object to the default value.</para>
92          /// <para>Выполняет попытку высвободить указанный объект, а так же установить значение
            ↪  переменной содержащей этот объект в значение по умолчанию.</para>
93          /// </summary>
94          /// <typeparam name="T"><para>Type of the specified object.</para><para>Тип указанного
            ↪  объекта.</para></typeparam>
95          /// <param name="object"><para>The object to dispose.</para><para>Объект, который
            ↪  необходимо высвободить.</para></param>
96          /// <returns><para>A value that determines whether the attempt to release the specified
            ↪  object was successful.</para><para>Значение, определяющие удачно ли была выполнена
            ↪  попытка высвободить указанный объект.</para></returns>
97          public static bool TryDisposeAndResetToDefault<T>(ref T @object)
98          {
99              var result = @object.TryDispose();
100             if (result)
101             {
102                 @object = default;
103             }
104             return result;
105         }
106     }
107 }
```

1.2 ./Platform.Disposables/DisposableBase.cs

```
1   using System;
2   using System.Collections.Concurrent;
3   using System.Threading;
4   using Platform.Exceptions;
5
6   namespace Platform.Disposables
7   {
8       /// <summary>
9       /// <para>Provides a base implementation for IDisposable interface with the basic logic
            ↪  necessary to increase the likelihood of correct unmanaged resources release.</para>
10      /// <para>Предоставляет базовую реализацию для интерфейса IDisposable с основной логикой
            ↪  необходимой для повышения вероятности корректного высвобождения неуправляемых
            ↪  ресурсов.</para>
11      /// </summary>
```

```
12      public abstract class DisposableBase : IDisposable
13      {
14          private static readonly AppDomain _currentDomain = AppDomain.CurrentDomain;
15          private static readonly ConcurrentStack<WeakReference<DisposableBase>>
            ↪  _disposablesWeekReferencesStack = new
            ↪  ConcurrentStack<WeakReference<DisposableBase>>();
16
17          private volatile int _disposed;
18
19          /// <summary>
20          /// <para>Gets a value indicating whether the object was disposed.</para>
21          /// <para>Возвращает значение определяющее был ли высвобожден объект.</para>
22          /// </summary>
23          public bool IsDisposed => _disposed > 0;
24
25          /// <summary>
26          /// <para>Gets the name of an object or a unique string describing this object.</para>
27          /// <para>Возвращает имя объекта или уникальную строку описывающую этот объект.</para>
28          /// </summary>
29          protected virtual string ObjectName => GetType().Name;
30
31          /// <summary>
32          /// <para>Gets a value indicating whether multiple attempts to dispose this object are
            ↪  allowed.</para>
33          /// <para>Возвращает значение определяющие разрешено ли выполнять несколько попыток
            ↪  высвободить этот объект.</para>
34          /// </summary>
35          protected virtual bool AllowMultipleDisposeAttempts => false;
36
37          /// <summary>
38          /// <para>Gets a value indicating whether it is allowed to call this object disposal
            ↪  multiple times.</para>
39          /// <para>Возвращает значение определяющие разрешено ли несколько раз вызывать
            ↪  высвобождение этого объекта.</para>
40          /// </summary>
41          protected virtual bool AllowMultipleDisposeCalls => false;
42
43          static DisposableBase() => _currentDomain.ProcessExit += OnProcessExit;
44
45          /// <summary>
46          /// <para>Initializes a new instance of the <see cref="DisposableBase"/> class.</para>
47          /// <para>Инициализирует новый экземпляр класса <see cref="DisposableBase"/>.</para>
48          /// </summary>
49          protected DisposableBase()
50          {
51              _disposed = 0;
52              _disposablesWeekReferencesStack.Push(new WeakReference<DisposableBase>(this, false));
53          }
54
55          /// <summary>
56          /// <para>Performs any necessary final clean-up when a class instance is being collected
            ↪  by the garbage collector.</para>
57          /// <para>Выполняет любую необходимую окончательную очистку, когда сборщик мусора
            ↪  собирает экземпляр класса.</para>
58          /// </summary>
59          ~DisposableBase() => Destruct();
60
61          /// <summary>
62          /// <para>Disposes unmanaged resources.</para>
63          /// <para>Высвобождает неуправляемые ресурсы.</para>
64          /// </summary>
65          /// <param name="manual">
66          /// <para>A value that determines whether the disposal was triggered manually (by the
            ↪  developer's code) or was executed automatically without an explicit indication from
            ↪  the developer.</para>
67          /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
            ↪  разработчика) или же выполнилось автоматически без явного указания со стороны
            ↪  разработчика.</para>
68          /// </param>
69          /// <param name="wasDisposed">
70          /// <para>A value that determines whether the object was released before calling this
            ↪  method.</para>
71          /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
72          /// </param>
73          protected abstract void Dispose(bool manual, bool wasDisposed);
74
75          /// <summary>
```

```
76          /// <para>Performs application-defined tasks associated with freeing, releasing, or
            ↪  resetting unmanaged resources.</para>
77          /// <para>Выполняет определенные пользователем задачи, связанные с освобождением,
            ↪  высвобождением или сбросом неуправляемых ресурсов.</para>
78          /// </summary>
79          public void Dispose()
80          {
81              GC.SuppressFinalize(this);
82              Dispose(true);
83          }
84
85          /// <summary>
86          /// <para>Performs application-defined tasks associated with freeing, releasing, or
            ↪  resetting unmanaged resources without throwing any exceptions.</para>
87          /// <para>Выполняет определенные пользователем задачи, связанные с освобождением,
            ↪  высвобождением или сбросом неуправляемых ресурсов без выбрасывания исключений.</para>
88          /// </summary>
89          /// <remarks>
90          /// <para>Should be called only from classes destructors, or in case exceptions should
            ↪  be not thrown.</para>
91          /// <para>Должен вызываться только из деструкторов классов, или в случае, если
            ↪  исключения выбрасывать нельзя.</para>
92          /// </remarks>
93          public void Destruct()
94          {
95              try
96              {
97                  if (!IsDisposed)
98                  {
99                      Dispose(false);
100                 }
101             }
102             catch (Exception exception)
103             {
104                 exception.Ignore();
105             }
106         }
107
108         private void Dispose(bool manual)
109         {
110             var originalDisposedValue = Interlocked.CompareExchange(ref _disposed, 1, 0);
111             var wasDisposed = originalDisposedValue > 0;
112             if (wasDisposed && !AllowMultipleDisposeCalls && manual)
113             {
114                 Ensure.Always.NotDisposed(this, ObjectName, "Multiple dispose calls are not
                    ↪  allowed. Override AllowMultipleDisposeCalls property to modify behavior.");
115             }
116             if (AllowMultipleDisposeAttempts || !wasDisposed)
117             {
118                 Dispose(manual, wasDisposed);
119             }
120         }
121
122         private static void OnProcessExit(object sender, EventArgs e)
123         {
124             while (_disposablesWeekReferencesStack.TryPop(out WeakReference<DisposableBase>
                ↪  weakReference))
125             {
126                 if (weakReference.TryGetTarget(out DisposableBase disposable))
127                 {
128                     GC.SuppressFinalize(disposable);
129                     disposable.Destruct();
130                 }
131             }
132             UnsubscribeFromProcessExitedEventIfPossible();
133         }
134
135         private static void UnsubscribeFromProcessExitedEventIfPossible()
136         {
137             try
138             {
139                 if (_currentDomain != null)
140                 {
141                     _currentDomain.ProcessExit -= OnProcessExit;
142                 }
143                 else
144                 {
145                     AppDomain.CurrentDomain.ProcessExit -= OnProcessExit;
```

```
                }
            }
            catch (Exception exception)
            {
                exception.Ignore();
            }
        }
    }
}
```

1.3 ./Platform.Disposables/Disposable[TPrimary, TAuxiliary].cs

```
using System;

namespace Platform.Disposables
{
    /// <summary>
    /// <para>Represents disposable container that disposes two contained objects when the container itself is disposed.</para>
    /// <para>Представляет высвобождаемый контейнер, который высвобождает два содержащийся в нём объектов при высвобождении самого контейнера.</para>
    /// </summary>
    /// <typeparam name="TPrimary"><para>The primary object type.</para><para>Тип основного объекта.</para></typeparam>
    /// <typeparam name="TAuxiliary"><para>The auxiliary object type.</para><para>Тип вспомогательного объекта.</para></typeparam>
    public class Disposable<TPrimary, TAuxiliary> : Disposable<TPrimary>
    {
        /// <summary>
        /// <para>Gets the auxiliary object.</para>
        /// <para>Возвращает вспомогательный объект.</para>
        /// </summary>
        public TAuxiliary AuxiliaryObject { get; }

        /// <summary>
        /// <para>Initializes a new instance of the <see cref="Disposable{TPrimary, TAuxiliary}"/> object.</para>
        /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{TPrimary, TAuxiliary}"/>.</para>
        /// </summary>
        /// <param name="object"><para>The primary object.</para><para>Основной объект.</para></param>
        /// <param name="auxiliaryObject"><para>The auxiliary object.</para><para>Вспомогательный объект.</para></param>
        /// <param name="action"><para>The <see cref="Action{TPrimary, TAuxiliary}"/> delegate.</para><para>Делегат <see cref="Action{TPrimary, TAuxiliary}"/>.</para></param>
        public Disposable(TPrimary @object, TAuxiliary auxiliaryObject, Action<TPrimary, TAuxiliary> action)
            : base(@object)
        {
            AuxiliaryObject = auxiliaryObject;
            OnDispose += (manual, wasDisposed) =>
            {
                if (!wasDisposed)
                {
                    action(Object, AuxiliaryObject);
                }
            };
        }

        /// <summary>
        /// <para>Initializes a new instance of the <see cref="Disposable{TPrimary, TAuxiliary}"/> object.</para>
        /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{TPrimary, TAuxiliary}"/>.</para>
        /// </summary>
        /// <param name="object"><para>The primary object.</para><para>Основной объект.</para></param>
        /// <param name="auxiliaryObject"><para>The auxiliary object.</para><para>Вспомогательный объект.</para></param>
        /// <param name="action"><para>The <see cref="Action"/> delegate.</para><para>Делегат <see cref="Action"/>.</para></param>
        public Disposable(TPrimary @object, TAuxiliary auxiliaryObject, Action action) : base(@object, action) => AuxiliaryObject = auxiliaryObject;

        /// <summary>
```

```
49          /// <para>Initializes a new instance of the <see cref="Disposable{TPrimary,
            ↪  TAuxiliary}"/> object.</para>
50          /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{TPrimary,
            ↪  TAuxiliary}"/>.</para>
51          /// </summary>
52          /// <param name="object"><para>The primary object.</para><para>Основной
            ↪  объект.</para></param>
53          /// <param name="auxiliaryObject"><para>The auxiliary
            ↪  object.</para><para>Вспомогательный объект.</para></param>
54          /// <param name="disposal"><para>The <see cref="Disposal"/>
            ↪  delegate.</para><para>Делегат <see cref="Disposal"/>.</para></param>
55          public Disposable(TPrimary @object, TAuxiliary auxiliaryObject, Disposal disposal) :
            ↪  base(@object, disposal) => AuxiliaryObject = auxiliaryObject;
56
57          /// <summary>
58          /// <para>Initializes a new instance of the <see cref="Disposable{TPrimary,
            ↪  TAuxiliary}"/> object.</para>
59          /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{TPrimary,
            ↪  TAuxiliary}"/>.</para>
60          /// </summary>
61          /// <param name="object"><para>The primary object.</para><para>Основной
            ↪  объект.</para></param>
62          /// <param name="auxiliaryObject"><para>The auxiliary
            ↪  object.</para><para>Вспомогательный объект.</para></param>
63          public Disposable(TPrimary @object, TAuxiliary auxiliaryObject) : base(@object) =>
            ↪  AuxiliaryObject = auxiliaryObject;
64
65          /// <summary>
66          /// <para>Initializes a new instance of the <see cref="Disposable{TPrimary,
            ↪  TAuxiliary}"/> object.</para>
67          /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{TPrimary,
            ↪  TAuxiliary}"/>.</para>
68          /// </summary>
69          /// <param name="object"><para>The primary object.</para><para>Основной
            ↪  объект.</para></param>
70          public Disposable(TPrimary @object) : base(@object) { }
71
72          /// <summary>
73          /// <para>Creates a new <see cref="Disposable{TPrimary, TAuxiliary}"/> object
                initialized with <see cref="ValueTuple{TPrimary, TAuxiliary, TAction}.Item1"/> as
            ↪   <see cref="Disposable{TPrimary}.Object"/>, <see cref="ValueTuple{TPrimary,
            ↪   TAuxiliary, TAction}.Item2"/> as <see cref="Disposable{TPrimary,
            ↪   TAuxiliary}.AuxiliaryObject"/> and <see cref="ValueTuple{TPrimary, TAuxiliary,
            ↪   TAction}.Item3"/> as delegate <see cref="Action{TPrimary, TAuxiliary}"/>.</para>
74          /// <para>Создает новый объект <see cref="Disposable{TPrimary, TAuxiliary}"/>,
                инициализированную с помощью <see cref="ValueTuple{TPrimary, TAuxiliary,
            ↪   TAction}.Item1"/> как <see cref="Disposable{TPrimary}.Object"/>, <see
            ↪   cref="ValueTuple{TPrimary, TAuxiliary, TAction}.Item2"/> как <see
            ↪   cref="Disposable{TPrimary, TAuxiliary}.AuxiliaryObject"/> и <see
            ↪   cref="ValueTuple{TPrimary, TAuxiliary, TAction}.Item3"/> как делегат <see
            ↪   cref="Action{TPrimary, TAuxiliary}"/>.</para>
75          /// </summary>
76          /// <param name="tuple"><para>The tuple.</para><para>Кортеж.</para></param>
77          public static implicit operator Disposable<TPrimary, TAuxiliary>(ValueTuple<TPrimary,
            ↪  TAuxiliary, Action<TPrimary, TAuxiliary>> tuple) => new Disposable<TPrimary,
            ↪  TAuxiliary>(tuple.Item1, tuple.Item2, tuple.Item3);
78
79          /// <summary>
80          /// <para>Creates a new <see cref="Disposable{TPrimary, TAuxiliary}"/> object
                initialized with <see cref="ValueTuple{TPrimary, TAuxiliary, TAction}.Item1"/> as
            ↪   <see cref="Disposable{TPrimary}.Object"/>, <see cref="ValueTuple{TPrimary,
            ↪   TAuxiliary, TAction}.Item2"/> as <see cref="Disposable{TPrimary,
            ↪   TAuxiliary}.AuxiliaryObject"/> and <see cref="ValueTuple{TPrimary, TAuxiliary,
            ↪   TAction}.Item3"/> as delegate <see cref="Action"/>.</para>
81          /// <para>Создает новый объект <see cref="Disposable{TPrimary, TAuxiliary}"/>,
                инициализированную с помощью <see cref="ValueTuple{TPrimary, TAuxiliary,
            ↪   TAction}.Item1"/> как <see cref="Disposable{TPrimary}.Object"/>, <see
            ↪   cref="ValueTuple{TPrimary, TAuxiliary, TAction}.Item2"/> как <see
            ↪   cref="Disposable{TPrimary, TAuxiliary}.AuxiliaryObject"/> и <see
            ↪   cref="ValueTuple{TPrimary, TAuxiliary, TAction}.Item3"/> как делегат <see
            ↪   cref="Action"/>.</para>
82          /// </summary>
83          /// <param name="tuple"><para>The tuple.</para><para>Кортеж.</para></param>
```

```
84          public static implicit operator Disposable<TPrimary, TAuxiliary>(ValueTuple<TPrimary,
            ↪  TAuxiliary, Action> tuple) => new Disposable<TPrimary, TAuxiliary>(tuple.Item1,
            ↪  tuple.Item2, tuple.Item3);
85
86          /// <summary>
87          /// <para>Creates a new <see cref="Disposable{TPrimary, TAuxiliary}"/> object
                initialized with <see cref="ValueTuple{TPrimary, TAuxiliary, TDisposal}.Item1"/> as
            ↪  <see cref="Disposable{TPrimary}.Object"/>, <see cref="ValueTuple{TPrimary,
            ↪  TAuxiliary, TDisposal}.Item2"/> as <see cref="Disposable{TPrimary,
            ↪  TAuxiliary}.AuxiliaryObject"/> and <see cref="ValueTuple{TPrimary, TAuxiliary,
            ↪  TDisposal}.Item3"/> as delegate <see cref="Disposal"/>.</para>
88          /// <para>Создает новый объект <see cref="Disposable{TPrimary, TAuxiliary}"/>,
                инициализированную с помощью <see cref="ValueTuple{TPrimary, TAuxiliary,
            ↪  TDisposal}.Item1"/> как <see cref="Disposable{TPrimary}.Object"/>, <see
            ↪  cref="ValueTuple{TPrimary, TAuxiliary, TDisposal}.Item2"/> как <see
            ↪  cref="Disposable{TPrimary, TAuxiliary}.AuxiliaryObject"/> и <see
            ↪  cref="ValueTuple{TPrimary, TAuxiliary, TDisposal}.Item3"/> как делегат <see
            ↪  cref="Disposal"/>.</para>
89          /// </summary>
90          /// <param name="tuple"><para>The tuple.</para><para>Кортеж.</para></param>
91          public static implicit operator Disposable<TPrimary, TAuxiliary>(ValueTuple<TPrimary,
            ↪  TAuxiliary, Disposal> tuple) => new Disposable<TPrimary, TAuxiliary>(tuple.Item1,
            ↪  tuple.Item2, tuple.Item3);
92
93          /// <summary>
94          /// <para>Creates a new <see cref="Disposable{TPrimary, TAuxiliary}"/> object
                initialized with <see cref="ValueTuple{TPrimary, TAuxiliary}.Item1"/> as <see
            ↪  cref="Disposable{TPrimary}.Object"/> and <see cref="ValueTuple{TPrimary,
            ↪  TAuxiliary}.Item2"/> as <see cref="Disposable{TPrimary,
            ↪  TAuxiliary}.AuxiliaryObject"/>.</para>
95          /// <para>Создает новый объект <see cref="Disposable{TPrimary, TAuxiliary}"/>,
                инициализированную с помощью <see cref="ValueTuple{TPrimary, TAuxiliary}.Item1"/>
            ↪  как <see cref="Disposable{TPrimary}.Object"/> и <see cref="ValueTuple{TPrimary,
            ↪  TAuxiliary}.Item2"/> как <see cref="Disposable{TPrimary,
            ↪  TAuxiliary}.AuxiliaryObject"/>.</para>
96          /// </summary>
97          /// <param name="tuple"><para>The tuple.</para><para>Кортеж.</para></param>
98          public static implicit operator Disposable<TPrimary, TAuxiliary>(ValueTuple<TPrimary,
            ↪  TAuxiliary> tuple) => new Disposable<TPrimary, TAuxiliary>(tuple.Item1, tuple.Item2);
99
100         /// <summary>
101         /// <para>Creates a new copy of the primary object (<see
            ↪  cref="Disposable{TPrimary}.Object"/>).</para>
102         /// <para>Создаёт новую копию основного объекта (<see
            ↪  cref="Disposable{TPrimary}.Object"/>).</para>
103         /// </summary>
104         /// <param name="disposableContainer"><para>The disposable
            ↪  container.</para><para>Высвобождаемый контейнер.</para></param>
105         public static implicit operator TPrimary(Disposable<TPrimary, TAuxiliary>
            ↪  disposableContainer) => disposableContainer.Object;
106
107         /// <summary>
108         /// <para>Creates a new copy of the auxiliary object (<see
            ↪  cref="Disposable{TPrimary}.Object"/>).</para>
109         /// <para>Создаёт новую копию вспомогательного объекта (<see cref="Disposable{TPrimary,
            ↪  TAuxiliary}.AuxiliaryObject"/>).</para>
110         /// </summary>
111         /// <param name="disposableContainer"><para>The disposable
            ↪  container.</para><para>Высвобождаемый контейнер.</para></param>
112         public static implicit operator TAuxiliary(Disposable<TPrimary, TAuxiliary>
            ↪  disposableContainer) => disposableContainer.AuxiliaryObject;
113
114         /// <summary>
115         /// <para>Disposes unmanaged resources.</para>
116         /// <para>Высвобождает неуправляемые ресурсы.</para>
117         /// </summary>
118         /// <param name="manual">
119         /// <para>A value that determines whether the disposal was triggered manually (by the
            ↪  developer's code) or was executed automatically without an explicit indication from
            ↪  the developer.</para>
120         /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
            ↪  разработчика) или же выполнилось автоматически без явного указания со стороны
            ↪  разработчика.</para>
121         /// </param>
122         /// <param name="wasDisposed">
```

```
123         /// <para>A value that determines whether the object was released before calling this
            ↪  method.</para>
124         /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
125         /// </param>
126         protected override void Dispose(bool manual, bool wasDisposed)
127         {
128             RaiseOnDisposeEvent(manual, wasDisposed);
129             AuxiliaryObject.TryDispose();
130             Object.TryDispose();
131         }
132     }
133 }
```

1.4 ./Platform.Disposables/Disposable[T].cs

```
1  using System;
2
3  namespace Platform.Disposables
4  {
5      /// <summary>
6      /// <para>Represents disposable container that disposes contained object when the container
       ↪  itself is disposed.</para>
7      /// <para>Представляет высвобождаемый контейнер, который высвобождает содержащийся в нём
       ↪  объект при высвобождении самого контейнера.</para>
8      /// </summary>
9      public class Disposable<T> : Disposable
10     {
11         /// <summary>
12         /// <para>Gets the object.</para>
13         /// <para>Возвращает объект.</para>
14         /// </summary>
15         public T Object { get; }
16
17         /// <summary>
18         /// <para>Initializes a new instance of the <see cref="Disposable{T}"/> object.</para>
19         /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{T}"/>.</para>
20         /// </summary>
21         /// <param name="object"><para>The object.</para><para>Объект.</para></param>
22         /// <param name="action"><para>The <see cref="Action{T}"/> delegate.</para><para>Делегат
           ↪  <see cref="Action{T}"/>.</para></param>
23         public Disposable(T @object, Action<T> action)
24         {
25             Object = @object;
26             OnDispose += (manual, wasDisposed) =>
27             {
28                 if (!wasDisposed)
29                 {
30                     action(Object);
31                 }
32             };
33         }
34
35         /// <summary>
36         /// <para>Initializes a new instance of the <see cref="Disposable{T}"/> object.</para>
37         /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{T}"/>.</para>
38         /// </summary>
39         /// <param name="object"><para>The object.</para><para>Объект.</para></param>
40         /// <param name="action"><para>The <see cref="Action"/> delegate.</para><para>Делегат
           ↪  <see cref="Action"/>.</para></param>
41         public Disposable(T @object, Action action) : base(action) => Object = @object;
42
43         /// <summary>
44         /// <para>Initializes a new instance of the <see cref="Disposable{T}"/> object.</para>
45         /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{T}"/>.</para>
46         /// </summary>
47         /// <param name="object"><para>The object.</para><para>Объект.</para></param>
48         /// <param name="disposal"><para>The <see cref="Disposal"/>
           ↪  delegate.</para><para>Делегат <see cref="Disposal"/>.</para></param>
49         public Disposable(T @object, Disposal disposal) : base(disposal) => Object = @object;
50
51         /// <summary>
52         /// <para>Initializes a new instance of the <see cref="Disposable{T}"/> object.</para>
53         /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{T}"/>.</para>
54         /// </summary>
55         /// <param name="object"><para>The object.</para><para>Объект.</para></param>
56         public Disposable(T @object) => Object = @object;
57
58         /// <summary>
```

```
59          /// <para>Creates a new <see cref="Disposable{T}"/> object initialized with <see
                ↪ cref="ValueTuple{T, TDisposal}.Item1"/> as <see cref="Disposable{T}.Object"/> and
                ↪ <see cref="ValueTuple{T, TAction}.Item2"/> as delegate <see
                ↪ cref="Action{T}"/>.</para>
60          /// <para>Создает новый объект <see cref="Disposable{T}"/>, инициализированную с помощью
                ↪ <see cref="ValueTuple{T, TDisposal}.Item1"/> как <see cref="Disposable{T}.Object"/>
                ↪ и <see cref="ValueTuple{T, TAction}.Item2"/> как делегат <see
                ↪ cref="Action{T}"/>.</para>
61          /// </summary>
62          /// <param name="tuple"><para>The tuple.</para><para>Кортеж.</para></param>
63          public static implicit operator Disposable<T>(ValueTuple<T, Action<T>> tuple) => new
                ↪ Disposable<T>(tuple.Item1, tuple.Item2);
64
65          /// <summary>
66          /// <para>Creates a new <see cref="Disposable{T}"/> object initialized with <see
                ↪ cref="ValueTuple{T, TDisposal}.Item1"/> as <see cref="Disposable{T}.Object"/> and
                ↪ <see cref="ValueTuple{T, TAction}.Item2"/> as delegate <see cref="Action"/>.</para>
67          /// <para>Создает новый объект <see cref="Disposable{T}"/>, инициализированную с помощью
                ↪ <see cref="ValueTuple{T, TDisposal}.Item1"/> как <see cref="Disposable{T}.Object"/>
                ↪ и <see cref="ValueTuple{T, TAction}.Item2"/> как делегат <see cref="Action"/>.</para>
68          /// </summary>
69          /// <param name="tuple"><para>The tuple.</para><para>Кортеж.</para></param>
70          public static implicit operator Disposable<T>(ValueTuple<T, Action> tuple) => new
                ↪ Disposable<T>(tuple.Item1, tuple.Item2);
71
72          /// <summary>
73          /// <para>Creates a new <see cref="Disposable{T}"/> object initialized with <see
                ↪ cref="ValueTuple{T, TDisposal}.Item1"/> as <see cref="Disposable{T}.Object"/> and
                ↪ <see cref="ValueTuple{T, TDisposal}.Item2"/> as delegate <see
                ↪ cref="Disposal"/>.</para>
74          /// <para>Создает новый объект <see cref="Disposable{T}"/>, инициализированную с помощью
                ↪ <see cref="ValueTuple{T, TDisposal}.Item1"/> как <see cref="Disposable{T}.Object"/>
                ↪ и <see cref="ValueTuple{T, TDisposal}.Item2"/> как делегат <see
                ↪ cref="Disposal"/>.</para>
75          /// </summary>
76          /// <param name="tuple"><para>The tuple.</para><para>Кортеж.</para></param>
77          public static implicit operator Disposable<T>(ValueTuple<T, Disposal> tuple) => new
                ↪ Disposable<T>(tuple.Item1, tuple.Item2);
78
79          /// <summary>
80          /// <para>Creates a new <see cref="Disposable{T}"/> object initialized with specified
                ↪ object as <see cref="Disposable{T}.Object"/>.</para>
81          /// <para>Создает новый объект <see cref="Disposable{T}"/>, инициализированную с помощью
                ↪ указанного объекта как <see cref="Disposable{T}.Object"/>.</para>
82          /// </summary>
83          /// <param name="object"><para>The object.</para><para>Объект.</para></param>
84          public static implicit operator Disposable<T>(T @object) => new Disposable<T>(@object);
85
86          /// <summary>
87          /// <para>Creates a new copy of the primary object (<see
                ↪ cref="Disposable{T}.Object"/>).</para>
88          /// <para>Создаёт новую копию основного объекта (<see
                ↪ cref="Disposable{T}.Object"/>).</para>
89          /// </summary>
90          /// <param name="disposableContainer"><para>The disposable
                ↪ container.</para><para>Высвобождаемый контейнер.</para></param>
91          public static implicit operator T(Disposable<T> disposableContainer) =>
                ↪ disposableContainer.Object;
92
93          /// <summary>
94          /// <para>Disposes unmanaged resources.</para>
95          /// <para>Высвобождает неуправляемые ресурсы.</para>
96          /// </summary>
97          /// <param name="manual">
98          /// <para>A value that determines whether the disposal was triggered manually (by the
                ↪ developer's code) or was executed automatically without an explicit indication from
                ↪ the developer.</para>
99          /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
                ↪ разработчика) или же выполнилось автоматически без явного указания со стороны
                ↪ разработчика.</para>
100         /// </param>
101         /// <param name="wasDisposed">
102         /// <para>A value that determines whether the object was released before calling this
                ↪ method.</para>
103         /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
104         /// </param>
```

```
        protected override void Dispose(bool manual, bool wasDisposed)
        {
            base.Dispose(manual, wasDisposed);
            Object.TryDispose();
        }
    }
}
```

1.5 ./Platform.Disposables/Disposal.cs

```
namespace Platform.Disposables
{
    /// <summary>
    /// <para>Encapsulates a method that is used to dispose unmanaged resources.</para>
    /// <para>Инкапсулирует метод, который используется для высвобождения неуправляемых ресурсов.</para>
    /// </summary>
    /// <param name="manual">
    /// <para>A value that determines whether the disposal was triggered manually (by the developer's code) or was executed automatically without an explicit indication from the developer.</para>
    /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом разработчика) или же выполнилось автоматически без явного указания со стороны разработчика.</para>
    /// </param>
    /// <param name="wasDisposed">
    /// <para>A value that determines whether the object was released before calling this method.</para>
    /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
    /// </param>
    public delegate void Disposal(bool manual, bool wasDisposed);
}
```

1.6 ./Platform.Disposables/EnsureExtensions.cs

```
using System;
using System.Diagnostics;
using System.Runtime.CompilerServices;
using Platform.Exceptions;
using Platform.Exceptions.ExtensionRoots;

#pragma warning disable IDE0060 // Remove unused parameter

namespace Platform.Disposables
{
    /// <summary>
    /// <para>Provides a set of extension methods for <see cref="EnsureAlwaysExtensionRoot"/> and <see cref="EnsureOnDebugExtensionRoot"/> objects.</para>
    /// <para>Предоставляет набор методов расширения для объектов <see cref="EnsureAlwaysExtensionRoot"/> и <see cref="EnsureOnDebugExtensionRoot"/>.</para>
    /// </summary>
    public static class EnsureExtensions
    {
        #region Always

        /// <summary>
        /// <para>Ensures that an object that implements the <see cref="IDisposable"/> interface has not been released. This check is performed regardless of the build configuration.</para>
        /// <para>Гарантирует, что объект, реализующий интерфейс <see cref="IDisposable"/> не был высвобожден. Эта проверка выполняется внезависимости от конфигурации сборки.</para>
        /// </summary>
        /// <param name="root"><para>The extension root to which this method is bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
        /// <param name="disposable"><para>The object implementing the <see cref="IDisposable"/> interface.</para><para>Объект, реализующий интерфейс <see cref="IDisposable"/></para></param>
        /// <param name="objectName"><para>The name of object.</para><para>Имя объекта.</para></param>
        /// <param name="message"><para>The message of the thrown exception.</para><para>Сообщение выбрасываемого исключения.</para></param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void NotDisposed(this EnsureAlwaysExtensionRoot root, IDisposable disposable, string objectName, string message)
        {
            if (disposable.IsDisposed)
            {
                throw new ObjectDisposedException(objectName, message);
            }
```

```
34          }
35
36          /// <summary>
37          /// <para>Ensures that an object that implements the <see cref="IDisposable"/> interface
            ↪  has not been released. This check is performed regardless of the build
            ↪  configuration.</para>
38          /// <para>Гарантирует, что объект, реализующий интерфейс <see cref="IDisposable"/> не
            ↪  был высвобожден. Эта проверка выполняется внезависимости от конфигурации
            ↪  сборки.</para>
39          /// </summary>
40          /// <param name="root"><para>The extension root to which this method is
            ↪  bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
41          /// <param name="disposable"><para>The object implementing the <see cref="IDisposable"/>
            ↪  interface.</para><para>Объект, реализующий интерфейс <see
            ↪  cref="IDisposable"/></para></param>
42          /// <param name="objectName"><para>The name of object.</para><para>Имя
            ↪  объекта.</para></param>
43          [MethodImpl(MethodImplOptions.AggressiveInlining)]
44          public static void NotDisposed(this EnsureAlwaysExtensionRoot root, IDisposable
            ↪  disposable, string objectName) => NotDisposed(root, disposable, objectName, null);
45
46          /// <summary>
47          /// <para>Ensures that an object that implements the <see cref="IDisposable"/> interface
            ↪  has not been released. This check is performed regardless of the build
            ↪  configuration.</para>
48          /// <para>Гарантирует, что объект, реализующий интерфейс <see cref="IDisposable"/> не
            ↪  был высвобожден. Эта проверка выполняется внезависимости от конфигурации
            ↪  сборки.</para>
49          /// </summary>
50          /// <param name="root"><para>The extension root to which this method is
            ↪  bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
51          /// <param name="disposable"><para>The object implementing the <see cref="IDisposable"/>
            ↪  interface.</para><para>Объект, реализующий интерфейс <see
            ↪  cref="IDisposable"/></para></param>
52          [MethodImpl(MethodImplOptions.AggressiveInlining)]
53          public static void NotDisposed(this EnsureAlwaysExtensionRoot root, IDisposable
            ↪  disposable) => NotDisposed(root, disposable, null, null);
54
55          #endregion
56
57          #region OnDebug
58
59          /// <summary>
60          /// <para>Ensures that an object that implements the <see cref="IDisposable"/> interface
            ↪  has not been released. This check is performed only for DEBUG build
            ↪  configuration.</para>
61          /// <para>Гарантирует, что объект, реализующий интерфейс <see cref="IDisposable"/> не
            ↪  был высвобожден. Эта проверка выполняется только для конфигурации сборки
            ↪  DEBUG.</para>
62          /// </summary>
63          /// <param name="root"><para>The extension root to which this method is
            ↪  bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
64          /// <param name="disposable"><para>The object implementing the <see cref="IDisposable"/>
            ↪  interface.</para><para>Объект, реализующий интерфейс <see
            ↪  cref="IDisposable"/></para></param>
65          /// <param name="objectName"><para>The name of object.</para><para>Имя
            ↪  объекта.</para></param>
66          /// <param name="message"><para>The message of the thrown
            ↪  exception.</para><para>Сообщение выбрасываемого исключения.</para></param>
67          [Conditional("DEBUG")]
68          public static void NotDisposed(this EnsureOnDebugExtensionRoot root, IDisposable
            ↪  disposable, string objectName, string message) =>
            ↪  Ensure.Always.NotDisposed(disposable, objectName, message);
69
70          /// <summary>
71          /// <para>Ensures that an object that implements the <see cref="IDisposable"/> interface
            ↪  has not been released. This check is performed only for DEBUG build
            ↪  configuration.</para>
72          /// <para>Гарантирует, что объект, реализующий интерфейс <see cref="IDisposable"/> не
            ↪  был высвобожден. Эта проверка выполняется только для конфигурации сборки
            ↪  DEBUG.</para>
73          /// </summary>
74          /// <param name="root"><para>The extension root to which this method is
            ↪  bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
```

```
75          /// <param name="disposable"><para>The object implementing the <see cref="IDisposable"/>
            ↪  interface.</para><para>Объект, реализующий интерфейс <see
            ↪  cref="IDisposable"/></para></param>
76          /// <param name="objectName"><para>The name of object.</para><para>Имя
            ↪  объекта.</para></param>
77          [Conditional("DEBUG")]
78          public static void NotDisposed(this EnsureOnDebugExtensionRoot root, IDisposable
            ↪  disposable, string objectName) => Ensure.Always.NotDisposed(disposable, objectName,
            ↪  null);
79
80          /// <summary>
81          /// <para>Ensures that an object that implements the <see cref="IDisposable"/> interface
            ↪  has not been released. This check is performed only for DEBUG build
            ↪  configuration.</para>
82          /// <para>Гарантирует, что объект, реализующий интерфейс <see cref="IDisposable"/> не
            ↪  был высвобожден. Эта проверка выполняется только для конфигурации сборки
            ↪  DEBUG.</para>
83          /// </summary>
84          /// <param name="root"><para>The extension root to which this method is
            ↪  bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
85          /// <param name="disposable"><para>The object implementing the <see cref="IDisposable"/>
            ↪  interface.</para><para>Объект, реализующий интерфейс <see
            ↪  cref="IDisposable"/></para></param>
86          [Conditional("DEBUG")]
87          public static void NotDisposed(this EnsureOnDebugExtensionRoot root, IDisposable
            ↪  disposable) => Ensure.Always.NotDisposed(disposable, null, null);
88
89          #endregion
90      }
91  }
```

1.7 ./Platform.Disposables/GenericObjectExtensions.cs

```
1   using System;
2   using Platform.Exceptions;
3
4   namespace Platform.Disposables
5   {
6       /// <summary>
7       /// <para>Provides a set of static methods that help dispose a generic objects.</para>
8       /// <para>Предоставляет набор статических методов которые помогают высвободить универсальные
            ↪  объекты.</para>
9       /// </summary>
10      static public class GenericObjectExtensions
11      {
12          /// <summary>
13          /// <para>Attempts to dispose the specified object.</para>
14          /// <para>Выполняет попытку высвободить указанный объект.</para>
15          /// </summary>
16          /// <typeparam name="T"><para>Type of the specified object.</para><para>Тип указанного
            ↪  объекта.</para></typeparam>
17          /// <param name="object"><para>The object to dispose.</para><para>Объект, который
            ↪  необходимо высвободить.</para></param>
18          /// <returns><para>A value that determines whether the attempt to release the specified
            ↪  object was successful.</para><para>Значение, определяющие удачно ли была выполнена
            ↪  попытка высвободить указанный объект.</para></returns>
19          public static bool TryDispose<T>(this T @object)
20          {
21              try
22              {
23                  if (@object is DisposableBase disposableBase)
24                  {
25                      disposableBase.DisposeIfNotDisposed();
26                  }
27                  else if (@object is System.IDisposable disposable)
28                  {
29                      disposable.Dispose();
30                  }
31                  return true;
32              }
33              catch (Exception exception)
34              {
35                  exception.Ignore();
36              }
37              return false;
38          }
39
40          /// <summary>
```

```
41          /// <para>Attempts to dispose the specified object.</para>
42          /// <para>Выполняет попытку высвободить указанный объект.</para>
43          /// </summary>
44          /// <typeparam name="T"><para>Type of the specified object.</para><para>Тип указанного
            ↪ объекта.</para></typeparam>
45          /// <param name="object"><para>The object to dispose.</para><para>Объект, который
            ↪ необходимо высвободить.</para></param>
46          public static void DisposeIfPossible<T>(this T @object) => TryDispose(@object);
47      }
48  }
```

1.8 ./Platform.Disposables/IDisposable.cs

```
1   namespace Platform.Disposables
2   {
3       /// <summary>
4       /// <para>Представляет расширенный интерфейс IDisposable.</para>
5       /// <para>Represents an extended IDisposable interface.</para>
6       /// </summary>
7       public interface IDisposable : System.IDisposable
8       {
9           /// <summary>
10          /// <para>Gets a value indicating whether the object was disposed.</para>
11          /// <para>Возвращает значение определяющее был ли высвобожден объект.</para>
12          /// </summary>
13          bool IsDisposed { get; }
14
15          /// <summary>
16          /// <para>Performs application-defined tasks associated with freeing, releasing, or
            ↪ resetting unmanaged resources without throwing any exceptions.</para>
17          /// <para>Выполняет определенные пользователем задачи, связанные с освобождением,
            ↪ высвобождением или сбросом неуправляемых ресурсов без выбрасывания исключений.</para>
18          /// </summary>
19          /// <remarks>
20          /// <para>Should be called only from classes destructors, or in case exceptions should
            ↪ be not thrown.</para>
21          /// <para>Должен вызываться только из деструкторов классов, или в случае, если
            ↪ исключения выбрасывать нельзя.</para>
22          /// </remarks>
23          void Destruct();
24      }
25  }
```

1.9 ./Platform.Disposables/IDisposableExtensions.cs

```
1   namespace Platform.Disposables
2   {
3       /// <summary>
4       /// <para>Provides a set of extension methods for <see cref="IDisposable"/> objects.</para>
5       /// <para>Предоставляет набор методов расширения для объектов <see
        ↪ cref="IDisposable"/>.</para>
6       /// </summary>
7       public static class IDisposableExtensions
8       {
9           /// <summary>
10          /// <para>Attempts to dispose the specified object.</para>
11          /// <para>Выполняет попытку высвободить указанный объект.</para>
12          /// </summary>
13          /// <param name="disposable"><para>The object implementing the <see cref="IDisposable"/>
            ↪ interface.</para><para>Объект, реализующий интерфейс <see
            ↪ cref="IDisposable"/></para></param>
14          public static void DisposeIfNotDisposed(this IDisposable disposable)
15          {
16              if (!disposable.IsDisposed)
17              {
18                  disposable.Dispose();
19              }
20          }
21      }
22  }
```

1.10 ./Platform.Disposables.Tests/DisposableTests.cs

```
1   using System;
2   using System.Collections.Generic;
3   using System.Diagnostics;
4   using System.IO;
5   using System.Linq;
6   using System.Threading;
7   using Xunit;
```

```
8
9  namespace Platform.Disposables.Tests
10 {
11     public static class DisposableTests
12     {
13         [Fact]
14         public static void DisposalOrderTest()
15         {
16             var path = GetDisposalObjectTestProjectFilePath();
17             var logPath = Path.GetTempFileName();
18             var processStartInfo = new ProcessStartInfo
19             {
20                 FileName = "dotnet",
21                 Arguments = $"run -p \"{path}\" -f netcoreapp2.1 \"{logPath}\" false",
22                 UseShellExecute = false,
23                 //RedirectStandardOutput = true,
24                 CreateNoWindow = true
25             };
26             using (var process = Process.Start(processStartInfo))
27             {
28                 //string line = process.StandardOutput.ReadToEnd();
29                 process.WaitForExit();
30             }
31             var result = File.ReadAllText(logPath);
32             Assert.Equal("21", result);
33             File.Delete(logPath);
34         }
35
36         [Fact]
37         public static void DisposalAtProcessKillTest()
38         {
39             var path = GetDisposalObjectTestProjectFilePath();
40             var logPath = Path.GetTempFileName();
41             var processStartInfo = new ProcessStartInfo
42             {
43                 FileName = "dotnet",
44                 Arguments = $"run -p \"{path}\" -f netcoreapp2.1 \"{logPath}\" true",
45                 UseShellExecute = false,
46                 //RedirectStandardOutput = true,
47                 CreateNoWindow = true
48             };
49             using (var process = Process.Start(processStartInfo))
50             {
51                 //string line = process.StandardOutput.ReadToEnd();
52                 Thread.Sleep(1000);
53                 process.Kill();
54             }
55             var result = File.ReadAllText(logPath);
56             Assert.Equal("", result); // Currently, process termination will not release
                ↪ resources
57             File.Delete(logPath);
58         }
59
60         private static string GetDisposalObjectTestProjectFilePath()
61         {
62             const string currentProjectName = nameof(Platform) + "." + nameof(Disposables) + "."
                ↪ + nameof(Tests);
63             const string disposalOrderTestProjectName = currentProjectName + "." +
                ↪ nameof(DisposalOrderTest);
64             var currentDirectory = Environment.CurrentDirectory;
65             var pathParts = currentDirectory.Split(Path.DirectorySeparatorChar);
66             var newPathParts = new List<string>();
67             for (var i = 0; i < pathParts.Length; i++)
68             {
69                 if (string.Equals(pathParts[i], currentProjectName))
70                 {
71                     newPathParts.Add(disposalOrderTestProjectName);
72                     break;
73                 }
74                 else
75                 {
76                     newPathParts.Add(pathParts[i]);
77                 }
78             }
79             pathParts = newPathParts.ToArray();
80
81 #if NET471
82             var directory = string.Join(Path.DirectorySeparatorChar.ToString(),
                ↪ pathParts.ToArray());
```

```
#else
            var directory = Path.Combine(pathParts);
#endif
            var path = Path.Combine(directory, $"{disposalOrderTestProjectName}.csproj");
            if (!Path.IsPathRooted(path))
            {
                path = $"{Path.DirectorySeparatorChar}{path}";
            }
            return path;
        }
    }
}
```

1.11 ./Platform.Disposables.Tests/SystemTests.cs

```
using Xunit;

namespace Platform.Disposables.Tests
{
    /// <summary>
    /// <para>Contains tests for features of .NET Framework itself, that are required by current implementations.</para>
    /// <para>Содержит тесты для функций самого .NET Framework, которые требуются для текущих реализаций.</para>
    /// </summary>
    public static class SystemTests
    {
        [Fact]
        public static void UsingSupportsNullTest()
        {
            using (var disposable = null as IDisposable)
            {
                Assert.True(disposable == null);
            }
        }
    }
}
```

Index