1.1 ./csharp/Platform.Disposables/Disposable.cs

```
using System;
using System.Runtime.CompilerServices;

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>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        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>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        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>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        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>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        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>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
```

```
66          public static implicit operator Disposable(Disposal disposal) => new
            ↪ Disposable(disposal);
67
68          /// <summary>
69          /// <para>Disposes unmanaged resources.</para>
70          /// <para>Высвобождает неуправляемые ресурсы.</para>
71          /// </summary>
72          /// <param name="manual">
73          /// <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
            ↪ a developer.</para>
74          /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
            ↪ разработчика) или же выполнилось автоматически без явного указания со стороны
            ↪ разработчика.</para>
75          /// </param>
76          /// <param name="wasDisposed">
77          /// <para>A value that determines whether the object was released before calling this
            ↪ method.</para>
78          /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
79          /// </param>
80          [MethodImpl(MethodImplOptions.AggressiveInlining)]
81          protected override void Dispose(bool manual, bool wasDisposed) =>
            ↪ RaiseOnDisposeEvent(manual, wasDisposed);
82
83          /// <summary>
84          /// <para>Raises an unmanaged resource dispose event.</para>
85          /// <para>Генерирует событие высвобождения неуправляемых ресурсов.</para>
86          /// </summary>
87          /// <param name="manual">
88          /// <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
            ↪ a developer.</para>
89          /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
            ↪ разработчика) или же выполнилось автоматически без явного указания со стороны
            ↪ разработчика.</para>
90          /// </param>
91          /// <param name="wasDisposed">
92          /// <para>A value that determines whether the object was released before calling this
            ↪ method.</para>
93          /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
94          /// </param>
95          [MethodImpl(MethodImplOptions.AggressiveInlining)]
96          protected void RaiseOnDisposeEvent(bool manual, bool wasDisposed) => OnDispose(manual,
            ↪ wasDisposed);
97
98          /// <summary>
99          /// <para>Attempts to dispose the specified object, as well as set the value of the
            ↪ variable containing this object to the default value.</para>
100         /// <para>Выполняет попытку высвободить указанный объект, а так же установить значение
            ↪ переменной содержащей этот объект в значение по умолчанию.</para>
101         /// </summary>
102         /// <typeparam name="T"><para>Type of the specified object.</para><para>Тип указанного
            ↪ объекта.</para></typeparam>
103         /// <param name="object"><para>The object to dispose.</para><para>Объект, который
            ↪ необходимо высвободить.</para></param>
104         /// <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>
105         [MethodImpl(MethodImplOptions.AggressiveInlining)]
106         public static bool TryDisposeAndResetToDefault<T>(ref T @object)
107         {
108             var result = @object.TryDispose();
109             if (result)
110             {
111                 @object = default;
112             }
113             return result;
114         }
115     }
116 }
```

1.2 ./csharp/Platform.Disposables/DisposableBase.cs

```
1  using System;
2  using System.Collections.Concurrent;
3  using System.Runtime.CompilerServices;
4  using System.Threading;
5  using Platform.Exceptions;
```

```
6
7   namespace Platform.Disposables
8   {
9       /// <summary>
10      /// <para>Represents a base implementation for <see
        ↪  cref="Platform.Disposables.IDisposable"/> interface with the basic logic necessary to
        ↪  increase the likelihood of correct unmanaged resources release.</para>
11      /// <para>Представляет базовую реализацию для интерфейса <see
        ↪  cref="Platform.Disposables.IDisposable"/> с основной логикой необходимой для повышения
        ↪  вероятности корректного высвобождения неуправляемых ресурсов.</para>
12      /// </summary>
13      public abstract class DisposableBase : IDisposable
14      {
15          private static readonly AppDomain _currentDomain = AppDomain.CurrentDomain;
16          private static readonly ConcurrentStack<WeakReference<DisposableBase>>
            ↪  _disposablesWeekReferencesStack = new
            ↪  ConcurrentStack<WeakReference<DisposableBase>>();
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
24          {
25              [MethodImpl(MethodImplOptions.AggressiveInlining)]
26              get => _disposed > 0;
27          }
28
29          /// <summary>
30          /// <para>Gets the name of an object or a unique string describing this object.</para>
31          /// <para>Возвращает имя объекта или уникальную строку описывающую этот объект.</para>
32          /// </summary>
33          protected virtual string ObjectName
34          {
35              [MethodImpl(MethodImplOptions.AggressiveInlining)]
36              get => GetType().Name;
37          }
38
39          /// <summary>
40          /// <para>Gets a value indicating whether multiple attempts to dispose this object are
            ↪  allowed.</para>
41          /// <para>Возвращает значение определяющие разрешено ли выполнять несколько попыток
            ↪  высвободить этот объект.</para>
42          /// </summary>
43          protected virtual bool AllowMultipleDisposeAttempts
44          {
45              [MethodImpl(MethodImplOptions.AggressiveInlining)]
46              get => false;
47          }
48
49          /// <summary>
50          /// <para>Gets a value indicating whether it is allowed to call this object disposal
            ↪  multiple times.</para>
51          /// <para>Возвращает значение определяющие разрешено ли несколько раз вызывать
            ↪  высвобождение этого объекта.</para>
52          /// </summary>
53          protected virtual bool AllowMultipleDisposeCalls
54          {
55              [MethodImpl(MethodImplOptions.AggressiveInlining)]
56              get => false;
57          }
58
59          /// <summary>
60          /// <para>
61          /// Initializes a new <see cref="DisposableBase"/> instance.
62          /// </para>
63          /// <para></para>
64          /// </summary>
65          [MethodImpl(MethodImplOptions.AggressiveInlining)]
66          static DisposableBase() => _currentDomain.ProcessExit += OnProcessExit;
67
68          /// <summary>
69          /// <para>Initializes a new instance of the <see cref="DisposableBase"/> class.</para>
70          /// <para>Инициализирует новый экземпляр класса <see cref="DisposableBase"/>.</para>
71          /// </summary>
72          [MethodImpl(MethodImplOptions.AggressiveInlining)]
73          protected DisposableBase()
```

```
74          {
75              _disposed = 0;
76              _disposablesWeekReferencesStack.Push(new WeakReference<DisposableBase>(this, false));
77          }
78
79          /// <summary>
80          /// <para>Performs any necessary final clean-up when a class instance is being collected
            ↪  by the garbage collector.</para>
81          /// <para>Выполняет любую необходимую окончательную очистку, когда сборщик мусора
            ↪  собирает экземпляр класса.</para>
82          /// </summary>
83          [MethodImpl(MethodImplOptions.AggressiveInlining)]
84          ~DisposableBase() => Destruct();
85
86          /// <summary>
87          /// <para>Disposes unmanaged resources.</para>
88          /// <para>Высвобождает неуправляемые ресурсы.</para>
89          /// </summary>
90          /// <param name="manual">
91          /// <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
            ↪  a developer.</para>
92          /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
            ↪  разработчика) или же выполнилось автоматически без явного указания со стороны
            ↪  разработчика.</para>
93          /// </param>
94          /// <param name="wasDisposed">
95          /// <para>A value that determines whether the object was released before calling this
            ↪  method.</para>
96          /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
97          /// </param>
98          [MethodImpl(MethodImplOptions.AggressiveInlining)]
99          protected abstract void Dispose(bool manual, bool wasDisposed);
100
101         /// <summary>
102         /// <para>Performs application-defined tasks associated with freeing, releasing, or
            ↪  resetting unmanaged resources.</para>
103         /// <para>Выполняет определенные пользователем задачи, связанные с освобождением,
            ↪  высвобождением или сбросом неуправляемых ресурсов.</para>
104         /// </summary>
105         [MethodImpl(MethodImplOptions.AggressiveInlining)]
106         public void Dispose()
107         {
108             Dispose(true);
109             GC.SuppressFinalize(this);
110         }
111
112         /// <summary>
113         /// <para>Performs application-defined tasks associated with freeing, releasing, or
            ↪  resetting unmanaged resources without throwing any exceptions.</para>
114         /// <para>Выполняет определенные пользователем задачи, связанные с освобождением,
            ↪  высвобождением или сбросом неуправляемых ресурсов без выбрасывания исключений.</para>
115         /// </summary>
116         /// <remarks>
117         /// <para>Should be called only from classes destructors, or in case exceptions should
            ↪  be not thrown.</para>
118         /// <para>Должен вызываться только из деструкторов классов, или в случае, если
            ↪  исключения выбрасывать нельзя.</para>
119         /// </remarks>
120         [MethodImpl(MethodImplOptions.AggressiveInlining)]
121         public void Destruct()
122         {
123             try
124             {
125                 if (!IsDisposed)
126                 {
127                     Dispose(false);
128                 }
129             }
130             catch (Exception exception)
131             {
132                 exception.Ignore();
133             }
134         }
135
136         /// <summary>
137         /// <para>Disposes unmanaged resources.</para>
```

```
        /// <para>Высвобождает неуправляемые ресурсы.</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 a developer.</para>
        /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом разработчика) или же выполнилось автоматически без явного указания со стороны разработчика.</para>
        /// </param>
        protected virtual void Dispose(bool manual)
        {
            var originalDisposedValue = Interlocked.CompareExchange(ref _disposed, 1, 0);
            var wasDisposed = originalDisposedValue > 0;
            if (wasDisposed && !AllowMultipleDisposeCalls && manual)
            {
                Ensure.Always.NotDisposed(this, ObjectName, "Multiple dispose calls are not allowed. Override AllowMultipleDisposeCalls property to modify behavior.");
            }
            if (AllowMultipleDisposeAttempts || !wasDisposed)
            {
                Dispose(manual, wasDisposed);
            }
        }
        private static void OnProcessExit(object sender, EventArgs e)
        {
            while (_disposablesWeekReferencesStack.TryPop(out WeakReference<DisposableBase> weakReference))
            {
                if (weakReference.TryGetTarget(out DisposableBase disposable))
                {
                    GC.SuppressFinalize(disposable);
                    disposable.Destruct();
                }
            }
            UnsubscribeFromProcessExitedEventIfPossible();
        }
        private static void UnsubscribeFromProcessExitedEventIfPossible()
        {
            try
            {
                if (_currentDomain != null)
                {
                    _currentDomain.ProcessExit -= OnProcessExit;
                }
                else
                {
                    AppDomain.CurrentDomain.ProcessExit -= OnProcessExit;
                }
            }
            catch (Exception exception)
            {
                exception.Ignore();
            }
        }
    }
}
```

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

```
using System;
using System.Runtime.CompilerServices;

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 содержащихся в нём объекта 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
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            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>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        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>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public Disposable(TPrimary @object, TAuxiliary auxiliaryObject, Action action) :
        ↪  base(@object, action) => AuxiliaryObject = 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="disposal"><para>The <see cref="Disposal"/>
        ↪  delegate.</para><para>Делегат <see cref="Disposal"/>.</para></param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public Disposable(TPrimary @object, TAuxiliary auxiliaryObject, Disposal disposal) :
        ↪  base(@object, disposal) => AuxiliaryObject = 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>
```

```
70          /// <param name="auxiliaryObject"><para>The auxiliary
            ↪  object.</para><para>Вспомогательный объект.</para></param>
71          [MethodImpl(MethodImplOptions.AggressiveInlining)]
72          public Disposable(TPrimary @object, TAuxiliary auxiliaryObject) : base(@object) =>
            ↪  AuxiliaryObject = auxiliaryObject;
73
74          /// <summary>
75          /// <para>Initializes a new instance of the <see cref="Disposable{TPrimary,
            ↪  TAuxiliary}"/> object.</para>
76          /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{TPrimary,
            ↪  TAuxiliary}"/>.</para>
77          /// </summary>
78          /// <param name="object"><para>The primary object.</para><para>Основной
            ↪  объект.</para></param>
79          [MethodImpl(MethodImplOptions.AggressiveInlining)]
80          public Disposable(TPrimary @object) : base(@object) { }
81
82          /// <summary>
83          /// <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>
84          /// <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>
85          /// </summary>
86          /// <param name="tuple"><para>The tuple.</para><para>Кортеж.</para></param>
87          [MethodImpl(MethodImplOptions.AggressiveInlining)]
88          public static implicit operator Disposable<TPrimary, TAuxiliary>(ValueTuple<TPrimary,
            ↪  TAuxiliary, Action<TPrimary, TAuxiliary>> tuple) => new Disposable<TPrimary,
            ↪  TAuxiliary>(tuple.Item1, tuple.Item2, tuple.Item3);
89
90          /// <summary>
91          /// <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>
92          /// <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>
93          /// </summary>
94          /// <param name="tuple"><para>The tuple.</para><para>Кортеж.</para></param>
95          [MethodImpl(MethodImplOptions.AggressiveInlining)]
96          public static implicit operator Disposable<TPrimary, TAuxiliary>(ValueTuple<TPrimary,
            ↪  TAuxiliary, Action> tuple) => new Disposable<TPrimary, TAuxiliary>(tuple.Item1,
            ↪  tuple.Item2, tuple.Item3);
97
98          /// <summary>
99          /// <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>
100         /// <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>
101         /// </summary>
102         /// <param name="tuple"><para>The tuple.</para><para>Кортеж.</para></param>
```

```
103         [MethodImpl(MethodImplOptions.AggressiveInlining)]
104         public static implicit operator Disposable<TPrimary, TAuxiliary>(ValueTuple<TPrimary,
            ↪  TAuxiliary, Disposal> tuple) => new Disposable<TPrimary, TAuxiliary>(tuple.Item1,
            ↪  tuple.Item2, tuple.Item3);
105
106         /// <summary>
107         /// <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>
108         /// <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>
109         /// </summary>
110         /// <param name="tuple"><para>The tuple.</para><para>Кортеж.</para></param>
111         [MethodImpl(MethodImplOptions.AggressiveInlining)]
112         public static implicit operator Disposable<TPrimary, TAuxiliary>(ValueTuple<TPrimary,
            ↪  TAuxiliary> tuple) => new Disposable<TPrimary, TAuxiliary>(tuple.Item1, tuple.Item2);
113
114         /// <summary>
115         /// <para>Creates a new copy of the primary object (<see
            ↪  cref="Disposable{TPrimary}.Object"/>).</para>
116         /// <para>Создаёт новую копию основного объекта (<see
            ↪  cref="Disposable{TPrimary}.Object"/>).</para>
117         /// </summary>
118         /// <param name="disposableContainer"><para>The disposable
            ↪  container.</para><para>Высвобождаемый контейнер.</para></param>
119         [MethodImpl(MethodImplOptions.AggressiveInlining)]
120         public static implicit operator TPrimary(Disposable<TPrimary, TAuxiliary>
            ↪  disposableContainer) => disposableContainer.Object;
121
122         /// <summary>
123         /// <para>Creates a new copy of the auxiliary object (<see
            ↪  cref="Disposable{TPrimary}.Object"/>).</para>
124         /// <para>Создаёт новую копию вспомогательного объекта (<see cref="Disposable{TPrimary,
            ↪  TAuxiliary}.AuxiliaryObject"/>).</para>
125         /// </summary>
126         /// <param name="disposableContainer"><para>The disposable
            ↪  container.</para><para>Высвобождаемый контейнер.</para></param>
127         [MethodImpl(MethodImplOptions.AggressiveInlining)]
128         public static implicit operator TAuxiliary(Disposable<TPrimary, TAuxiliary>
            ↪  disposableContainer) => disposableContainer.AuxiliaryObject;
129
130         /// <summary>
131         /// <para>Disposes unmanaged resources.</para>
132         /// <para>Высвобождает неуправляемые ресурсы.</para>
133         /// </summary>
134         /// <param name="manual">
135         /// <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
            ↪  a developer.</para>
136         /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
            ↪  разработчика) или же выполнилось автоматически без явного указания со стороны
            ↪  разработчика.</para>
137         /// </param>
138         /// <param name="wasDisposed">
139         /// <para>A value that determines whether the object was released before calling this
            ↪  method.</para>
140         /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
141         /// </param>
142         [MethodImpl(MethodImplOptions.AggressiveInlining)]
143         protected override void Dispose(bool manual, bool wasDisposed)
144         {
145             RaiseOnDisposeEvent(manual, wasDisposed);
146             AuxiliaryObject.TryDispose();
147             Object.TryDispose();
148         }
149     }
150 }
```

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

```
1  using System;
2  using System.Runtime.CompilerServices;
```

```
3
4  namespace Platform.Disposables
5  {
6      /// <summary>
7      /// <para>Represents disposable container that disposes contained object when the container
       ↪  itself is disposed.</para>
8      /// <para>Представляет высвобождаемый контейнер, который высвобождает содержащийся в нём
       ↪  объект при высвобождении самого контейнера.</para>
9      /// </summary>
10     public class Disposable<T> : Disposable
11     {
12         /// <summary>
13         /// <para>Gets the object.</para>
14         /// <para>Возвращает объект.</para>
15         /// </summary>
16         public T Object
17         {
18             [MethodImpl(MethodImplOptions.AggressiveInlining)]
19             get;
20         }
21
22         /// <summary>
23         /// <para>Initializes a new instance of the <see cref="Disposable{T}"/> object.</para>
24         /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{T}"/>.</para>
25         /// </summary>
26         /// <param name="object"><para>The object.</para><para>Объект.</para></param>
27         /// <param name="action"><para>The <see cref="Action{T}"/> delegate.</para><para>Делегат
           ↪  <see cref="Action{T}"/>.</para></param>
28         [MethodImpl(MethodImplOptions.AggressiveInlining)]
29         public Disposable(T @object, Action<T> action)
30         {
31             Object = @object;
32             OnDispose += (manual, wasDisposed) =>
33             {
34                 if (!wasDisposed)
35                 {
36                     action(Object);
37                 }
38             };
39         }
40
41         /// <summary>
42         /// <para>Initializes a new instance of the <see cref="Disposable{T}"/> object.</para>
43         /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{T}"/>.</para>
44         /// </summary>
45         /// <param name="object"><para>The object.</para><para>Объект.</para></param>
46         /// <param name="action"><para>The <see cref="Action"/> delegate.</para><para>Делегат
           ↪  <see cref="Action"/>.</para></param>
47         [MethodImpl(MethodImplOptions.AggressiveInlining)]
48         public Disposable(T @object, Action action) : base(action) => Object = @object;
49
50         /// <summary>
51         /// <para>Initializes a new instance of the <see cref="Disposable{T}"/> object.</para>
52         /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{T}"/>.</para>
53         /// </summary>
54         /// <param name="object"><para>The object.</para><para>Объект.</para></param>
55         /// <param name="disposal"><para>The <see cref="Disposal"/>
           ↪  delegate.</para><para>Делегат <see cref="Disposal"/>.</para></param>
56         [MethodImpl(MethodImplOptions.AggressiveInlining)]
57         public Disposable(T @object, Disposal disposal) : base(disposal) => Object = @object;
58
59         /// <summary>
60         /// <para>Initializes a new instance of the <see cref="Disposable{T}"/> object.</para>
61         /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{T}"/>.</para>
62         /// </summary>
63         /// <param name="object"><para>The object.</para><para>Объект.</para></param>
64         [MethodImpl(MethodImplOptions.AggressiveInlining)]
65         public Disposable(T @object) => Object = @object;
66
67         /// <summary>
68         /// <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>
69         /// <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>
```

```
70          /// </summary>
71          /// <param name="tuple"><para>The tuple.</para><para>Кортеж.</para></param>
72          [MethodImpl(MethodImplOptions.AggressiveInlining)]
73          public static implicit operator Disposable<T>(ValueTuple<T, Action<T>> tuple) => new
            ↪  Disposable<T>(tuple.Item1, tuple.Item2);
74
75          /// <summary>
76          /// <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>
77          /// <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>
78          /// </summary>
79          /// <param name="tuple"><para>The tuple.</para><para>Кортеж.</para></param>
80          [MethodImpl(MethodImplOptions.AggressiveInlining)]
81          public static implicit operator Disposable<T>(ValueTuple<T, Action> tuple) => new
            ↪  Disposable<T>(tuple.Item1, tuple.Item2);
82
83          /// <summary>
84          /// <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>
85          /// <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>
86          /// </summary>
87          /// <param name="tuple"><para>The tuple.</para><para>Кортеж.</para></param>
88          [MethodImpl(MethodImplOptions.AggressiveInlining)]
89          public static implicit operator Disposable<T>(ValueTuple<T, Disposal> tuple) => new
            ↪  Disposable<T>(tuple.Item1, tuple.Item2);
90
91          /// <summary>
92          /// <para>Creates a new <see cref="Disposable{T}"/> object initialized with specified
            ↪  object as <see cref="Disposable{T}.Object"/>.</para>
93          /// <para>Создает новый объект <see cref="Disposable{T}"/>, инициализированную с помощью
            ↪  указанного объекта как <see cref="Disposable{T}.Object"/>.</para>
94          /// </summary>
95          /// <param name="object"><para>The object.</para><para>Объект.</para></param>
96          [MethodImpl(MethodImplOptions.AggressiveInlining)]
97          public static implicit operator Disposable<T>(T @object) => new Disposable<T>(@object);
98
99          /// <summary>
100         /// <para>Creates a new copy of the primary object (<see
            ↪  cref="Disposable{T}.Object"/>).</para>
101         /// <para>Создаёт новую копию основного объекта (<see
            ↪  cref="Disposable{T}.Object"/>).</para>
102         /// </summary>
103         /// <param name="disposableContainer"><para>The disposable
            ↪  container.</para><para>Высвобождаемый контейнер.</para></param>
104         [MethodImpl(MethodImplOptions.AggressiveInlining)]
105         public static implicit operator T(Disposable<T> disposableContainer) =>
            ↪  disposableContainer.Object;
106
107         /// <summary>
108         /// <para>Disposes unmanaged resources.</para>
109         /// <para>Высвобождает неуправляемые ресурсы.</para>
110         /// </summary>
111         /// <param name="manual">
112         /// <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
            ↪  a developer.</para>
113         /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
            ↪  разработчика) или же выполнилось автоматически без явного указания со стороны
            ↪  разработчика.</para>
114         /// </param>
115         /// <param name="wasDisposed">
116         /// <para>A value that determines whether the object was released before calling this
            ↪  method.</para>
117         /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
118         /// </param>
119         [MethodImpl(MethodImplOptions.AggressiveInlining)]
120         protected override void Dispose(bool manual, bool wasDisposed)
121         {
```

```
122             base.Dispose(manual, wasDisposed);
123             Object.TryDispose();
124         }
125     }
126 }
```

1.5 ./csharp/Platform.Disposables/Disposal.cs

```
1  namespace Platform.Disposables
2  {
3      /// <summary>
4      /// <para>Encapsulates a method that is used to dispose unmanaged resources.</para>
5      /// <para>Инкапсулирует метод, который используется для высвобождения неуправляемых
       ↪  ресурсов.</para>
6      /// </summary>
7      /// <param name="manual">
8      /// <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 a
       ↪  developer.</para>
9      /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом разработчика)
       ↪  или же выполнилось автоматически без явного указания со стороны разработчика.</para>
10     /// </param>
11     /// <param name="wasDisposed">
12     /// <para>A value that determines whether the object was released before calling this
       ↪  method.</para>
13     /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
14     /// </param>
15     public delegate void Disposal(bool manual, bool wasDisposed);
16 }
```

1.6 ./csharp/Platform.Disposables/EnsureExtensions.cs

```
1  using System;
2  using System.Diagnostics;
3  using System.Runtime.CompilerServices;
4  using Platform.Exceptions;
5  using Platform.Exceptions.ExtensionRoots;
6
7  #pragma warning disable IDE0060 // Remove unused parameter
8
9  namespace Platform.Disposables
10 {
11     /// <summary>
12     /// <para>Provides a set of extension methods for <see
       ↪  cref="Platform.Exceptions.ExtensionRoots.EnsureAlwaysExtensionRoot"/> and <see
       ↪  cref="Platform.Exceptions.ExtensionRoots.EnsureOnDebugExtensionRoot"/> objects.</para>
13     /// <para>Предоставляет набор методов расширения для объектов <see
       ↪  cref="Platform.Exceptions.ExtensionRoots.EnsureAlwaysExtensionRoot"/> и <see
       ↪  cref="Platform.Exceptions.ExtensionRoots.EnsureOnDebugExtensionRoot"/>.</para>
14     /// </summary>
15     public static class EnsureExtensions
16     {
17         #region Always
18
19         /// <summary>
20         /// <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>
21         /// <para>Гарантирует, что объект, реализующий интерфейс <see cref="IDisposable"/> не
           ↪  был высвобожден. Эта проверка выполняется внезависимости от конфигурации
           ↪  сборки.</para>
22         /// </summary>
23         /// <param name="root"><para>The extension root to which this method is
           ↪  bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
24         /// <param name="disposable"><para>The object implementing the <see cref="IDisposable"/>
           ↪  interface.</para><para>Объект, реализующий интерфейс <see
           ↪  cref="IDisposable"/></para></param>
25         /// <param name="objectName"><para>The name of object.</para><para>Имя
           ↪  объекта.</para></param>
26         /// <param name="message"><para>The message of the thrown
           ↪  exception.</para><para>Сообщение выбрасываемого исключения.</para></param>
27         [MethodImpl(MethodImplOptions.AggressiveInlining)]
28         public static void NotDisposed(this EnsureAlwaysExtensionRoot root, IDisposable
           ↪  disposable, string objectName, string message)
29         {
30             if (disposable.IsDisposed)
31             {
32                 throw new ObjectDisposedException(objectName, message);
33             }
```

```
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 ./csharp/Platform.Disposables/GenericObjectExtensions.cs

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

```
41          /// <summary>
42          /// <para>Attempts to dispose the specified object.</para>
43          /// <para>Выполняет попытку высвободить указанный объект.</para>
44          /// </summary>
45          /// <typeparam name="T"><para>Type of the specified object.</para><para>Тип указанного
            ↪  объекта.</para></typeparam>
46          /// <param name="object"><para>The object to dispose.</para><para>Объект, который
            ↪  необходимо высвободить.</para></param>
47          [MethodImpl(MethodImplOptions.AggressiveInlining)]
48          public static void DisposeIfPossible<T>(this T @object) => TryDispose(@object);
49      }
50  }
```

1.8 ./csharp/Platform.Disposables/IDisposable.cs

```
1   using System.Runtime.CompilerServices;
2
3   namespace Platform.Disposables
4   {
5       /// <summary>
6       /// <para>Представляет расширенный интерфейс <see cref="System.IDisposable"/>.</para>
7       /// <para>Represents an extended <see cref="System.IDisposable"/> interface.</para>
8       /// </summary>
9       public interface IDisposable : System.IDisposable
10      {
11          /// <summary>
12          /// <para>Gets a value indicating whether the object was disposed.</para>
13          /// <para>Возвращает значение определяющее был ли высвобожден объект.</para>
14          /// </summary>
15          bool IsDisposed
16          {
17              [MethodImpl(MethodImplOptions.AggressiveInlining)]
18              get;
19          }
20
21          /// <summary>
22          /// <para>Performs application-defined tasks associated with freeing, releasing, or
            ↪  resetting unmanaged resources without throwing any exceptions.</para>
23          /// <para>Выполняет определенные пользователем задачи, связанные с освобождением,
            ↪  высвобождением или сбросом неуправляемых ресурсов без выбрасывания исключений.</para>
24          /// </summary>
25          /// <remarks>
26          /// <para>Should be called only from classes destructors, or in case exceptions should
            ↪  be not thrown.</para>
27          /// <para>Должен вызываться только из деструкторов классов, или в случае, если
            ↪  исключения выбрасывать нельзя.</para>
28          /// </remarks>
29          [MethodImpl(MethodImplOptions.AggressiveInlining)]
30          void Destruct();
31      }
32  }
```

1.9 ./csharp/Platform.Disposables/IDisposableExtensions.cs

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

```
        }
    }
}
```

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

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Threading;
using Xunit;

namespace Platform.Disposables.Tests
{
    public static class DisposableTests
    {
        [Fact]
        public static void DisposalOrderTest()
        {
            var logPath = Path.GetTempFileName();
            using (var process = Process.Start(CreateProcessStartInfo(logPath,
            ↪  waitForCancellation: false)))
            {
                process.WaitForExit();
            }
            var result = File.ReadAllText(logPath);
            Assert.Equal("21", result);
            File.Delete(logPath);
        }

        [Fact]
        public static void DisposalAtProcessKillTest()
        {
            var logPath = Path.GetTempFileName();
            using (var process = Process.Start(CreateProcessStartInfo(logPath,
            ↪  waitForCancellation: true)))
            {
                Thread.Sleep(1000);
                process.Kill();
            }
            var result = File.ReadAllText(logPath);
            Assert.Equal("", result); // Currently, process termination will not release
            ↪  resources
            File.Delete(logPath);
        }
        private static ProcessStartInfo CreateProcessStartInfo(string logPath, bool
        ↪  waitForCancellation)
        {
            var projectPath = GetDisposalObjectTestProjectFilePath();
            return new ProcessStartInfo
            {
                FileName = "dotnet",
                Arguments = $"run -p \"{projectPath}\" -f net5 \"{logPath}\"
                ↪  {waitForCancellation.ToString()}",
                UseShellExecute = false,
                CreateNoWindow = true
            };
        }
        private static string GetDisposalObjectTestProjectFilePath()
        {
            const string currentProjectName = nameof(Platform) + "." + nameof(Disposables) + "."
            ↪  + nameof(Tests);
            const string disposalOrderTestProjectName = currentProjectName + "." +
            ↪  nameof(DisposalOrderTest);
            var currentDirectory = Environment.CurrentDirectory;
            var pathParts = currentDirectory.Split(Path.DirectorySeparatorChar);
            var newPathParts = new List<string>();
            for (var i = 0; i < pathParts.Length; i++)
            {
                if (string.Equals(pathParts[i], currentProjectName))
                {
                    newPathParts.Add(disposalOrderTestProjectName);
                    break;
                }
                else
                {
                    newPathParts.Add(pathParts[i]);
                }
```

```
            }
            pathParts = newPathParts.ToArray();
#if NET472
            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 ./csharp/Platform.Disposables.Tests/SystemTests.cs

```
using Xunit;

namespace Platform.Disposables.Tests
{
    public static class SystemTests
    {
        [Fact]
        public static void UsingSupportsNullTest()
        {
            using var disposable = null as IDisposable;
            Assert.True(disposable == null);
        }
    }
}
```

Index