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
    {
        /// <summary>
        /// <para>
        /// The was disposed.
        /// </para>
        /// <para></para>
        /// </summary>
        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>
```

```
68          /// <para>Создает новый объект <see cref="Disposable"/>, инициализированную с помощью
            ↪  указанного делегата <see cref="Disposal"/>.</para>
69          /// </summary>
70          /// <param name="disposal"><para>The <see cref="Disposal"/>
            ↪  delegate.</para><para>Делегат <see cref="Disposal"/>.</para></param>
71          [MethodImpl(MethodImplOptions.AggressiveInlining)]
72          public static implicit operator Disposable(Disposal disposal) => new
            ↪  Disposable(disposal);
73
74          /// <summary>
75          /// <para>Disposes unmanaged resources.</para>
76          /// <para>Высвобождает неуправляемые ресурсы.</para>
77          /// </summary>
78          /// <param name="manual">
79          /// <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>
80          /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
            ↪  разработчика) или же выполнилось автоматически без явного указания со стороны
            ↪  разработчика.</para>
81          /// </param>
82          /// <param name="wasDisposed">
83          /// <para>A value that determines whether the object was released before calling this
            ↪  method.</para>
84          /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
85          /// </param>
86          [MethodImpl(MethodImplOptions.AggressiveInlining)]
87          protected override void Dispose(bool manual, bool wasDisposed) =>
            ↪  RaiseOnDisposeEvent(manual, wasDisposed);
88
89          /// <summary>
90          /// <para>Raises an unmanaged resource dispose event.</para>
91          /// <para>Генерирует событие высвобождения неуправляемых ресурсов.</para>
92          /// </summary>
93          /// <param name="manual">
94          /// <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>
95          /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
            ↪  разработчика) или же выполнилось автоматически без явного указания со стороны
            ↪  разработчика.</para>
96          /// </param>
97          /// <param name="wasDisposed">
98          /// <para>A value that determines whether the object was released before calling this
            ↪  method.</para>
99          /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
100         /// </param>
101         [MethodImpl(MethodImplOptions.AggressiveInlining)]
102         protected void RaiseOnDisposeEvent(bool manual, bool wasDisposed) => OnDispose(manual,
            ↪  wasDisposed);
103
104         /// <summary>
105         /// <para>Attempts to dispose the specified object, as well as set the value of the
            ↪  variable containing this object to the default value.</para>
106         /// <para>Выполняет попытку высвободить указанный объект, а так же установить значение
            ↪  переменной содержащей этот объект в значение по умолчанию.</para>
107         /// </summary>
108         /// <typeparam name="T"><para>Type of the specified object.</para><para>Тип указанного
            ↪  объекта.</para></typeparam>
109         /// <param name="object"><para>The object to dispose.</para><para>Объект, который
            ↪  необходимо высвободить.</para></param>
110         /// <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>
111         [MethodImpl(MethodImplOptions.AggressiveInlining)]
112         public static bool TryDisposeAndResetToDefault<T>(ref T @object)
113         {
114             var result = @object.TryDispose();
115             if (result)
116             {
117                 @object = default;
118             }
119             return result;
120         }
121     }
122 }
```

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

```
using System;
using System.Collections.Concurrent;
using System.Runtime.CompilerServices;
using System.Threading;
using Platform.Exceptions;

namespace Platform.Disposables
{
    /// <summary>
    /// <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>
    /// <para>Представляет базовую реализацию для интерфейса <see cref="Platform.Disposables.IDisposable"/> с основной логикой необходимой для повышения вероятности корректного высвобождения неуправляемых ресурсов.</para>
    /// </summary>
    public abstract class DisposableBase : IDisposable
    {
        /// <summary>
        /// <para>
        /// The current domain.
        /// </para>
        /// <para></para>
        /// </summary>
        private static readonly AppDomain _currentDomain = AppDomain.CurrentDomain;
        /// <summary>
        /// <para>
        /// The disposable base.
        /// </para>
        /// <para></para>
        /// </summary>
        private static readonly ConcurrentStack<WeakReference<DisposableBase>> _disposablesWeekReferencesStack = new ConcurrentStack<WeakReference<DisposableBase>>();

        /// <summary>
        /// <para>
        /// The disposed.
        /// </para>
        /// <para></para>
        /// </summary>
        private volatile int _disposed;

        /// <summary>
        /// <para>Gets a value indicating whether the object was disposed.</para>
        /// <para>Возвращает значение определяющее был ли высвобожден объект.</para>
        /// </summary>
        public bool IsDisposed
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get => _disposed > 0;
        }

        /// <summary>
        /// <para>Gets the name of an object or a unique string describing this object.</para>
        /// <para>Возвращает имя объекта или уникальную строку описывающую этот объект.</para>
        /// </summary>
        protected virtual string ObjectName
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get => GetType().Name;
        }

        /// <summary>
        /// <para>Gets a value indicating whether multiple attempts to dispose this object are allowed.</para>
        /// <para>Возвращает значение определяющие разрешено ли выполнять несколько попыток высвободить этот объект.</para>
        /// </summary>
        protected virtual bool AllowMultipleDisposeAttempts
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get => false;
        }

        /// <summary>
        /// <para>Gets a value indicating whether it is allowed to call this object disposal multiple times.</para>
```

```
70          /// <para>Возвращает значение определяющие разрешено ли несколько раз вызывать
            ↪  высвобождение этого объекта.</para>
71          /// </summary>
72          protected virtual bool AllowMultipleDisposeCalls
73          {
74              [MethodImpl(MethodImplOptions.AggressiveInlining)]
75              get => false;
76          }
77
78          /// <summary>
79          /// <para>
80          /// Initializes a new <see cref="DisposableBase"/> instance.
81          /// </para>
82          /// <para></para>
83          /// </summary>
84          [MethodImpl(MethodImplOptions.AggressiveInlining)]
85          static DisposableBase() => _currentDomain.ProcessExit += OnProcessExit;
86
87          /// <summary>
88          /// <para>Initializes a new instance of the <see cref="DisposableBase"/> class.</para>
89          /// <para>Инициализирует новый экземпляр класса <see cref="DisposableBase"/>.</para>
90          /// </summary>
91          [MethodImpl(MethodImplOptions.AggressiveInlining)]
92          protected DisposableBase()
93          {
94              _disposed = 0;
95              _disposablesWeekReferencesStack.Push(new WeakReference<DisposableBase>(this, false));
96          }
97
98          /// <summary>
99          /// <para>Performs any necessary final clean-up when a class instance is being collected
            ↪  by the garbage collector.</para>
100         /// <para>Выполняет любую необходимую окончательную очистку, когда сборщик мусора
            ↪  собирает экземпляр класса.</para>
101         /// </summary>
102         [MethodImpl(MethodImplOptions.AggressiveInlining)]
103         ~DisposableBase() => Destruct();
104
105         /// <summary>
106         /// <para>Disposes unmanaged resources.</para>
107         /// <para>Высвобождает неуправляемые ресурсы.</para>
108         /// </summary>
109         /// <param name="manual">
110         /// <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>
111         /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
            ↪  разработчика) или же выполнилось автоматически без явного указания со стороны
            ↪  разработчика.</para>
112         /// </param>
113         /// <param name="wasDisposed">
114         /// <para>A value that determines whether the object was released before calling this
            ↪  method.</para>
115         /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
116         /// </param>
117         [MethodImpl(MethodImplOptions.AggressiveInlining)]
118         protected abstract void Dispose(bool manual, bool wasDisposed);
119
120         /// <summary>
121         /// <para>Performs application-defined tasks associated with freeing, releasing, or
            ↪  resetting unmanaged resources.</para>
122         /// <para>Выполняет определенные пользователем задачи, связанные с освобождением,
            ↪  высвобождением или сбросом неуправляемых ресурсов.</para>
123         /// </summary>
124         [MethodImpl(MethodImplOptions.AggressiveInlining)]
125         public void Dispose()
126         {
127             Dispose(true);
128             GC.SuppressFinalize(this);
129         }
130
131         /// <summary>
132         /// <para>Performs application-defined tasks associated with freeing, releasing, or
            ↪  resetting unmanaged resources without throwing any exceptions.</para>
133         /// <para>Выполняет определенные пользователем задачи, связанные с освобождением,
            ↪  высвобождением или сбросом неуправляемых ресурсов без выбрасывания исключений.</para>
134         /// </summary>
```

```
135         /// <remarks>
136         /// <para>Should be called only from classes destructors, or in case exceptions should
            ↪  be not thrown.</para>
137         /// <para>Должен вызываться только из деструкторов классов, или в случае, если
            ↪  исключения выбрасывать нельзя.</para>
138         /// </remarks>
139         [MethodImpl(MethodImplOptions.AggressiveInlining)]
140         public void Destruct()
141         {
142             try
143             {
144                 if (!IsDisposed)
145                 {
146                     Dispose(false);
147                 }
148             }
149             catch (Exception exception)
150             {
151                 exception.Ignore();
152             }
153         }
154 
155         /// <summary>
156         /// <para>Disposes unmanaged resources.</para>
157         /// <para>Высвобождает неуправляемые ресурсы.</para>
158         /// </summary>
159         /// <param name="manual">
160         /// <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>
161         /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
            ↪  разработчика) или же выполнилось автоматически без явного указания со стороны
            ↪  разработчика.</para>
162         /// </param>
163         protected virtual void Dispose(bool manual)
164         {
165             var originalDisposedValue = Interlocked.CompareExchange(ref _disposed, 1, 0);
166             var wasDisposed = originalDisposedValue > 0;
167             if (wasDisposed && !AllowMultipleDisposeCalls && manual)
168             {
169                 Ensure.Always.NotDisposed(this, ObjectName, "Multiple dispose calls are not
                    ↪  allowed. Override AllowMultipleDisposeCalls property to modify behavior.");
170             }
171             if (AllowMultipleDisposeAttempts || !wasDisposed)
172             {
173                 Dispose(manual, wasDisposed);
174             }
175         }
176 
177         /// <summary>
178         /// <para>
179         /// Ons the process exit using the specified sender.
180         /// </para>
181         /// <para></para>
182         /// </summary>
183         /// <param name="sender">
184         /// <para>The sender.</para>
185         /// <para></para>
186         /// </param>
187         /// <param name="e">
188         /// <para>The .</para>
189         /// <para></para>
190         /// </param>
191         private static void OnProcessExit(object sender, EventArgs e)
192         {
193             while (_disposablesWeekReferencesStack.TryPop(out WeakReference<DisposableBase>
                ↪  weakReference))
194             {
195                 if (weakReference.TryGetTarget(out DisposableBase disposable))
196                 {
197                     GC.SuppressFinalize(disposable);
198                     disposable.Destruct();
199                 }
200             }
201             UnsubscribeFromProcessExitedEventIfPossible();
202         }
203 
```

```
        /// <summary>
        /// <para>
        /// Unsubscribes the from process exited event if possible.
        /// </para>
        /// <para></para>
        /// </summary>
        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>
    /// <param name="auxiliaryObject"><para>The auxiliary
    ↪  object.</para><para>Вспомогательный объект.</para></param>
    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    public Disposable(TPrimary @object, TAuxiliary auxiliaryObject) : base(@object) =>
    ↪  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>
    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    public Disposable(TPrimary @object) : base(@object) { }

    /// <summary>
    /// <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>
    /// <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;
```

```
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="Platform.Exceptions.ExtensionRoots.EnsureAlwaysExtensionRoot"/> and <see cref="Platform.Exceptions.ExtensionRoots.EnsureOnDebugExtensionRoot"/> objects.</para>
    /// <para>Предоставляет набор методов расширения для объектов <see cref="Platform.Exceptions.ExtensionRoots.EnsureAlwaysExtensionRoot"/> и <see cref="Platform.Exceptions.ExtensionRoots.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);
            }
        }

        /// <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>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void NotDisposed(this EnsureAlwaysExtensionRoot root, IDisposable disposable, string objectName) => NotDisposed(root, disposable, objectName, null);

        /// <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>
```

```
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;
```

```
using Platform.Exceptions;

namespace Platform.Disposables
{
    /// <summary>
    /// <para>Provides a set of static methods that help dispose a generic objects.</para>
    /// <para>Предоставляет набор статических методов которые помогают высвободить универсальные объекты.</para>
    /// </summary>
    static public class GenericObjectExtensions
    {
        /// <summary>
        /// <para>Attempts to dispose the specified object.</para>
        /// <para>Выполняет попытку высвободить указанный объект.</para>
        /// </summary>
        /// <typeparam name="T"><para>Type of the specified object.</para><para>Тип указанного объекта.</para></typeparam>
        /// <param name="object"><para>The object to dispose.</para><para>Объект, который необходимо высвободить.</para></param>
        /// <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>
        public static bool TryDispose<T>(this T @object)
        {
            try
            {
                if (@object is DisposableBase disposableBase)
                {
                    disposableBase.DisposeIfNotDisposed();
                }
                else if (@object is System.IDisposable disposable)
                {
                    disposable.Dispose();
                }
                return true;
            }
            catch (Exception exception)
            {
                exception.Ignore();
            }
            return false;
        }

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

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

```
using System.Runtime.CompilerServices;

namespace Platform.Disposables
{
    /// <summary>
    /// <para>Представляет расширенный интерфейс <see cref="System.IDisposable"/>.</para>
    /// <para>Represents an extended <see cref="System.IDisposable"/> interface.</para>
    /// </summary>
    public interface IDisposable : System.IDisposable
    {
        /// <summary>
        /// <para>Gets a value indicating whether the object was disposed.</para>
        /// <para>Возвращает значение определяющее был ли высвобожден объект.</para>
        /// </summary>
        bool IsDisposed
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
        }

        /// <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              }
23          }
24      }
25  }
```

1.10 ./csharp/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      /// <summary>
12      /// <para>
13      /// Represents the disposable tests.
14      /// </para>
15      /// <para></para>
16      /// </summary>
17      public static class DisposableTests
18      {
19          /// <summary>
20          /// <para>
21          /// Tests that disposal order test.
22          /// </para>
23          /// <para></para>
24          /// </summary>
25          [Fact]
26          public static void DisposalOrderTest()
27          {
28              var logPath = Path.GetTempFileName();
29              using (var process = Process.Start(CreateProcessStartInfo(logPath,
                ↪  waitForCancellation: false)))
30              {
```

```
            process.WaitForExit();
        }
        var result = File.ReadAllText(logPath);
        Assert.Equal("21", result);
        File.Delete(logPath);
    }

    /// <summary>
    /// <para>
    /// Tests that disposal at process kill test.
    /// </para>
    /// <para></para>
    /// </summary>
    [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);
    }

    /// <summary>
    /// <para>
    /// Creates the process start info using the specified log path.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <param name="logPath">
    /// <para>The log path.</para>
    /// <para></para>
    /// </param>
    /// <param name="waitForCancellation">
    /// <para>The wait for cancellation.</para>
    /// <para></para>
    /// </param>
    /// <returns>
    /// <para>The process start info</para>
    /// <para></para>
    /// </returns>
    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
        };
    }

    /// <summary>
    /// <para>
    /// Gets the disposal object test project file path.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <returns>
    /// <para>The path.</para>
    /// <para></para>
    /// </returns>
    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
{
    /// <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
    {
        /// <summary>
        /// <para>
        /// Tests that using supports null test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public static void UsingSupportsNullTest()
        {
            using var disposable = null as IDisposable;
            Assert.True(disposable == null);
        }
    }
}
```

Index