1.1 ./Platform.Exceptions/Ensure.cs

```
using Platform.Exceptions.ExtensionRoots;

namespace Platform.Exceptions
{
    /// <summary>
    /// <para>Contains two extensible classes instances that can be supplemented with static helper methods by using the extension mechanism. These methods ensure the contract compliance.</para>
    /// <para>Содержит два экземпляра расширяемых класса, которые можно дополнять статическими вспомогательными методами путём использования механизма расширений. Эти методы занимаются гарантированием соответствия контракту.</para>
    /// </summary>
    public static class Ensure
    {
        /// <summary>
        /// <para>Gets an instance of the extension root class that contains helper methods to guarantee compliance with the contract.</para>
        /// <para>Возвращает экземпляр класса корня-расширения, который содержит вспомогательные методы для гарантирования соответствия контракту.</para>
        /// </summary>
        public static readonly EnsureAlwaysExtensionRoot Always = new EnsureAlwaysExtensionRoot();

        /// <summary>
        /// <para>Gets an instance of the extension root class that contains helper methods to guarantee compliance with the contract, but are executed only during debugging.</para>
        /// <para>Возвращает экземпляр класса корня-расширения, который содержит вспомогательные методы для гарантирования соответствия контракту, но выполняются только во время отладки.</para>
        /// </summary>
        public static readonly EnsureOnDebugExtensionRoot OnDebug = new EnsureOnDebugExtensionRoot();
    }
}
```

1.2 ./Platform.Exceptions/EnsureExtensions.cs

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

#pragma warning disable IDE0060 // Remove unused parameter

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

        /// <summary>
        /// <para>Ensures that argument is not null. This check is performed regardless of the build configuration.</para>
        /// <para>Гарантирует, что аргумент не нулевой. Эта проверка выполняется внезависимости от конфигурации сборки.</para>
        /// </summary>
        /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип аргумента.</para></typeparam>
        /// <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="argument"><para>The argument.</para><para>Аргумент.</para></param>
        /// <param name="argumentName"><para>The argument's name.</para><para>Имя аргумента.</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 ArgumentNotNull<TArgument>(this EnsureAlwaysExtensionRoot root, TArgument argument, string argumentName, string message)
            where TArgument : class
```

```
30          {
31              if (argument == null)
32              {
33                  throw new ArgumentNullException(argumentName, message);
34              }
35          }
36
37          /// <summary>
38          /// <para>Ensures that argument is not null. This check is performed regardless of the
            ↪  build configuration.</para>
39          /// <para>Гарантирует, что аргумент не нулевой. Эта проверка выполняется внезависимости
            ↪  от конфигурации сборки.</para>
40          /// </summary>
41          /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
            ↪  аргумента.</para></typeparam>
42          /// <param name="root"><para>The extension root to which this method is
            ↪  bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
43          /// <param name="argument"><para>The argument.</para><para>Аргумент.</para></param>
44          /// <param name="argumentName"><para>The argument's name.</para><para>Имя
            ↪  аргумента.</para></param>
45          [MethodImpl(MethodImplOptions.AggressiveInlining)]
46          public static void ArgumentNotNull<TArgument>(this EnsureAlwaysExtensionRoot root,
            ↪  TArgument argument, string argumentName) where TArgument : class =>
            ↪  ArgumentNotNull(root, argument, argumentName, $"Argument {argumentName} is null.");
47
48          /// <summary>
49          /// <para>Ensures that argument is not null. This check is performed regardless of the
            ↪  build configuration.</para>
50          /// <para>Гарантирует, что аргумент не нулевой. Эта проверка выполняется внезависимости
            ↪  от конфигурации сборки.</para>
51          /// </summary>
52          /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
            ↪  аргумента.</para></typeparam>
53          /// <param name="root"><para>The extension root to which this method is
            ↪  bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
54          /// <param name="argument"><para>The argument.</para><para>Аргумент.</para></param>
55          [MethodImpl(MethodImplOptions.AggressiveInlining)]
56          public static void ArgumentNotNull<TArgument>(this EnsureAlwaysExtensionRoot root,
            ↪  TArgument argument) where TArgument : class => ArgumentNotNull(root, argument, null);
57
58          /// <summary>
59          /// <para>Ensures that the argument meets the criteria. This check is performed
            ↪  regardless of the build configuration.</para>
60          /// <para>Гарантирует, что аргумент соответствует критерию. Эта проверка выполняется
            ↪  внезависимости от конфигурации сборки.</para>
61          /// </summary>
62          /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
            ↪  аргумента.</para></typeparam>
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="argument"><para>The argument.</para><para>Аргумент.</para></param>
65          /// <param name="predicate"><para>A predicate that determines whether the argument meets
            ↪  a criterion.</para><para>Предикат определяющий, соответствует ли аргумент
            ↪  критерию.</para></param>
66          /// <param name="argumentName"><para>The argument's name.</para><para>Имя
            ↪  аргумента.</para></param>
67          /// <param name="message"><para>The message of the thrown
            ↪  exception.</para><para>Сообщение выбрасываемого исключения.</para></param>
68          [MethodImpl(MethodImplOptions.AggressiveInlining)]
69          public static void ArgumentMeetsCriteria<TArgument>(this EnsureAlwaysExtensionRoot root,
            ↪  TArgument argument, Predicate<TArgument> predicate, string argumentName, string
            ↪  message)
70          {
71              if (!predicate(argument))
72              {
73                  throw new ArgumentException(message, argumentName);
74              }
75          }
76
77          /// <summary>
78          /// <para>Ensures that the argument meets the criteria. This check is performed
            ↪  regardless of the build configuration.</para>
79          /// <para>Гарантирует, что аргумент соответствует критерию. Эта проверка выполняется
            ↪  внезависимости от конфигурации сборки.</para>
80          /// </summary>
```

```
81          /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
            ↪  аргумента.</para></typeparam>
82          /// <param name="root"><para>The extension root to which this method is
            ↪  bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
83          /// <param name="argument"><para>The argument.</para><para>Аргумент.</para></param>
84          /// <param name="predicate"><para>A predicate that determines whether the argument meets
            ↪  a criterion.</para><para>Предикат определяющий, соответствует ли аргумент
            ↪  критерию.</para></param>
85          /// <param name="argumentName"><para>The argument's name.</para><para>Имя
            ↪  аргумента.</para></param>
86          [MethodImpl(MethodImplOptions.AggressiveInlining)]
87          public static void ArgumentMeetsCriteria<TArgument>(this EnsureAlwaysExtensionRoot root,
            ↪  TArgument argument, Predicate<TArgument> predicate, string argumentName) =>
            ↪  ArgumentMeetsCriteria(root, argument, predicate, argumentName, $"Argument
            ↪  {argumentName} is does not meet criteria.");
88
89          /// <summary>
90          /// <para>Ensures that the argument meets the criteria. This check is performed
            ↪  regardless of the build configuration.</para>
91          /// <para>Гарантирует, что аргумент соответствует критерию. Эта проверка выполняется
            ↪  внезависимости от конфигурации сборки.</para>
92          /// </summary>
93          /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
            ↪  аргумента.</para></typeparam>
94          /// <param name="root"><para>The extension root to which this method is
            ↪  bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
95          /// <param name="argument"><para>The argument.</para><para>Аргумент.</para></param>
96          /// <param name="predicate"><para>A predicate that determines whether the argument meets
            ↪  a criterion.</para><para>Предикат определяющий, соответствует ли аргумент
            ↪  критерию.</para></param>
97          [MethodImpl(MethodImplOptions.AggressiveInlining)]
98          public static void ArgumentMeetsCriteria<TArgument>(this EnsureAlwaysExtensionRoot root,
            ↪  TArgument argument, Predicate<TArgument> predicate) => ArgumentMeetsCriteria(root,
            ↪  argument, predicate, null);
99
100         #endregion
101
102         #region OnDebug
103
104         /// <summary>
105         /// <para>Ensures that argument is not null. This check is performed only for DEBUG
            ↪  build configuration.</para>
106         /// <para>Гарантирует, что аргумент не нулевой. Эта проверка выполняется только для
            ↪  конфигурации сборки DEBUG.</para>
107         /// </summary>
108         /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
            ↪  аргумента.</para></typeparam>
109         /// <param name="root"><para>The extension root to which this method is
            ↪  bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
110         /// <param name="argument"><para>The argument.</para><para>Аргумент.</para></param>
111         /// <param name="argumentName"><para>The argument's name.</para><para>Имя
            ↪  аргумента.</para></param>
112         /// <param name="message"><para>The message of the thrown
            ↪  exception.</para><para>Сообщение выбрасываемого исключения.</para></param>
113         [Conditional("DEBUG")]
114         public static void ArgumentNotNull<TArgument>(this EnsureOnDebugExtensionRoot root,
            ↪  TArgument argument, string argumentName, string message) where TArgument : class =>
            ↪  Ensure.Always.ArgumentNotNull(argument, argumentName, message);
115
116         /// <summary>
117         /// <para>Ensures that argument is not null. This check is performed only for DEBUG
            ↪  build configuration.</para>
118         /// <para>Гарантирует, что аргумент не нулевой. Эта проверка выполняется только для
            ↪  конфигурации сборки DEBUG.</para>
119         /// </summary>
120         /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
            ↪  аргумента.</para></typeparam>
121         /// <param name="root"><para>The extension root to which this method is
            ↪  bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
122         /// <param name="argument"><para>The argument.</para><para>Аргумент.</para></param>
123         /// <param name="argumentName"><para>The argument's name.</para><para>Имя
            ↪  аргумента.</para></param>
124         [Conditional("DEBUG")]
125         public static void ArgumentNotNull<TArgument>(this EnsureOnDebugExtensionRoot root,
            ↪  TArgument argument, string argumentName) where TArgument : class =>
            ↪  Ensure.Always.ArgumentNotNull(argument, argumentName);
```

```
126
127         /// <summary>
128         /// <para>Ensures that argument is not null. This check is performed only for DEBUG
            ↪  build configuration.</para>
129         /// <para>Гарантирует, что аргумент не нулевой. Эта проверка выполняется только для
            ↪  конфигурации сборки DEBUG.</para>
130         /// </summary>
131         /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
            ↪  аргумента.</para></typeparam>
132         /// <param name="root"><para>The extension root to which this method is
            ↪  bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
133         /// <param name="argument"><para>The argument.</para><para>Аргумент.</para></param>
134         [Conditional("DEBUG")]
135         public static void ArgumentNotNull<TArgument>(this EnsureOnDebugExtensionRoot root,
            ↪  TArgument argument) where TArgument : class =>
            ↪  Ensure.Always.ArgumentNotNull(argument);
136
137         /// <summary>
138         /// <para>Ensures that the argument meets the criteria. This check is performed only for
            ↪  DEBUG build configuration.</para>
139         /// <para>Гарантирует, что аргумент соответствует критерию. Эта проверка выполняется
            ↪  только для конфигурации сборки DEBUG.</para>
140         /// </summary>
141         /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
            ↪  аргумента.</para></typeparam>
142         /// <param name="root"><para>The extension root to which this method is
            ↪  bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
143         /// <param name="argument"><para>The argument.</para><para>Аргумент.</para></param>
144         /// <param name="predicate"><para>A predicate that determines whether the argument meets
            ↪  a criterion.</para><para>Предикат определяющий, соответствует ли аргумент
            ↪  критерию.</para></param>
145         /// <param name="argumentName"><para>The argument's name.</para><para>Имя
            ↪  аргумента.</para></param>
146         /// <param name="message"><para>The message of the thrown
            ↪  exception.</para><para>Сообщение выбрасываемого исключения.</para></param>
147         [Conditional("DEBUG")]
148         public static void ArgumentMeetsCriteria<TArgument>(this EnsureOnDebugExtensionRoot
            ↪  root, TArgument argument, Predicate<TArgument> predicate, string argumentName,
            ↪  string message) => Ensure.Always.ArgumentMeetsCriteria(argument, predicate,
            ↪  argumentName, message);
149
150         /// <summary>
151         /// <para>Ensures that the argument meets the criteria. This check is performed only for
            ↪  DEBUG build configuration.</para>
152         /// <para>Гарантирует, что аргумент соответствует критерию. Эта проверка выполняется
            ↪  только для конфигурации сборки DEBUG.</para>
153         /// </summary>
154         /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
            ↪  аргумента.</para></typeparam>
155         /// <param name="root"><para>The extension root to which this method is
            ↪  bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
156         /// <param name="argument"><para>The argument.</para><para>Аргумент.</para></param>
157         /// <param name="predicate"><para>A predicate that determines whether the argument meets
            ↪  a criterion.</para><para>Предикат определяющий, соответствует ли аргумент
            ↪  критерию.</para></param>
158         /// <param name="argumentName"><para>The argument's name.</para><para>Имя
            ↪  аргумента.</para></param>
159         [Conditional("DEBUG")]
160         public static void ArgumentMeetsCriteria<TArgument>(this EnsureOnDebugExtensionRoot
            ↪  root, TArgument argument, Predicate<TArgument> predicate, string argumentName) =>
            ↪  Ensure.Always.ArgumentMeetsCriteria(argument, predicate, argumentName);
161
162         /// <summary>
163         /// <para>Ensures that the argument meets the criteria. This check is performed only for
            ↪  DEBUG build configuration.</para>
164         /// <para>Гарантирует, что аргумент соответствует критерию. Эта проверка выполняется
            ↪  только для конфигурации сборки DEBUG.</para>
165         /// </summary>
166         /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
            ↪  аргумента.</para></typeparam>
167         /// <param name="root"><para>The extension root to which this method is
            ↪  bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
168         /// <param name="argument"><para>The argument.</para><para>Аргумент.</para></param>
```

```
169         /// <param name="predicate"><para>A predicate that determines whether the argument meets
            ↪  a criterion.</para><para>Предикат определяющий, соответствует ли аргумент
            ↪  критерию.</para></param>
170         [Conditional("DEBUG")]
171         public static void ArgumentMeetsCriteria<TArgument>(this EnsureOnDebugExtensionRoot
            ↪  root, TArgument argument, Predicate<TArgument> predicate) =>
            ↪  Ensure.Always.ArgumentMeetsCriteria(argument, predicate);
172
173         #endregion
174     }
175 }
```

1.3 ./Platform.Exceptions/ExceptionExtensions.cs

```
1  using System;
2  using System.Text;
3
4  namespace Platform.Exceptions
5  {
6      /// <summary>
7      /// <para>Provides a set of extension methods for <see cref="Exception"/> objects.</para>
8      /// <para>Предоставляет набор методов расширения для объектов <see cref="Exception"/>.</para>
9      /// </summary>
10     public static class ExceptionExtensions
11     {
12         /// <summary>
13         /// <para>Represents the separator used within the process of generating a
            ↪  representation string (<see cref="ToStringWithAllInnerExceptions(Exception)"/>) to
            ↪  separate different inner exceptions from each other. This field is constant.</para>
14         /// <para>Представляет разделитель, используемый внутри процесса формирования
            ↪  строки-представления (<see cref="ToStringWithAllInnerExceptions(Exception)"/>) для
            ↪  разделения различных внутренних исключений друг от друга. Это поле является
            ↪  константой.</para>
15         /// </summary>
16         public static readonly string ExceptionContentsSeparator = "---";
17
18         /// <summary>
19         /// <para>Represents a string returned from <see
            ↪  cref="ToStringWithAllInnerExceptions(Exception)"/> in the event of an unsuccessful
            ↪  attempt to format an exception. This field is a constant.</para>
20         /// <para>Представляет строку выдаваемую из <see
            ↪  cref="ToStringWithAllInnerExceptions(Exception)"/> в случае неудачной попытки
            ↪  форматирования исключения. Это поле является константой.</para>
21         /// </summary>
22         public static readonly string ExceptionStringBuildingFailed = "Unable to format
            ↪  exception.";
23
24         /// <summary>
25         /// <para>Ignores the exception, notifying the <see cref = "IgnoredExceptions" /> class
            ↪  about it.</para>
26         /// <para>Игнорирует исключение, уведомляя об этом класс <see
            ↪  cref="IgnoredExceptions"/>.</para>
27         /// </summary>
28         /// <param name="exception"><para></para><para></para></param>
29         public static void Ignore(this Exception exception) =>
            ↪  IgnoredExceptions.RaiseExceptionIgnoredEvent(exception);
30
31         /// <summary>
32         /// <para>Returns a string that represents the specified exception with all its inner
            ↪  exceptions.</para>
33         /// <para>Возвращает строку, которая представляет указанное исключение со всеми его
            ↪  внутренними исключениями.</para>
34         /// </summary>
35         /// <param name="exception"><para>The exception that will be represented as a
            ↪  string.</para><para>Исключение, которое будет представленно в виде
            ↪  строки.</para></param>
36         /// <returns><para>A string that represents the specified exception with all its inner
            ↪  exceptions.</para><para>Строку, которая представляет указанное исключение со всеми
            ↪  его внутренними исключениями.</para></returns>
37         public static string ToStringWithAllInnerExceptions(this Exception exception)
38         {
39             try
40             {
41                 var sb = new StringBuilder();
42                 sb.BuildExceptionString(exception, 0);
43                 return sb.ToString();
44             }
45             catch (Exception ex)
```

```
            {
                ex.Ignore();
                return ExceptionStringBuildingFailed;
            }
        }

        private static void BuildExceptionString(this StringBuilder sb, Exception exception, int
        ↪  level)
        {
            sb.Indent(level);
            sb.AppendLine(exception.Message);
            sb.Indent(level);
            sb.AppendLine(ExceptionContentsSeparator);
            if (exception.InnerException != null)
            {
                sb.Indent(level);
                sb.AppendLine("Inner exception: ");
                sb.BuildExceptionString(exception.InnerException, level + 1);
            }
            sb.Indent(level);
            sb.AppendLine(ExceptionContentsSeparator);
            sb.Indent(level);
            sb.AppendLine(exception.StackTrace);
        }

        private static void Indent(this StringBuilder sb, int level)
        {
            sb.Append('\t', level);
        }
    }
}
```

1.4 ./Platform.Exceptions/ExtensionRoots/EnsureAlwaysExtensionRoot.cs

```
namespace Platform.Exceptions.ExtensionRoots
{
    /// <summary>
    /// <para>Represents the extension root class for Ensure.Always.</para>
    /// <para>Представляет класс корень-расширения для Ensure.Always.</para>
    /// </summary>
    public class EnsureAlwaysExtensionRoot
    {
    }
}
```

1.5 ./Platform.Exceptions/ExtensionRoots/EnsureOnDebugExtensionRoot.cs

```
namespace Platform.Exceptions.ExtensionRoots
{
    /// <summary>
    /// <para>Represents the extension root class for Ensure.OnDebug.</para>
    /// <para>Представляет класс корень-расширения для Ensure.OnDebug.</para>
    /// </summary>
    public class EnsureOnDebugExtensionRoot
    {
    }
}
```

1.6 ./Platform.Exceptions/ExtensionRoots/ThrowExtensionRoot.cs

```
namespace Platform.Exceptions.ExtensionRoots
{
    /// <summary>
    /// <para>Represents the extension root class for Throw.A.</para>
    /// <para>Представляет класс корень-расширения для Throw.A.</para>
    /// </summary>
    public class ThrowExtensionRoot
    {
    }
}
```

1.7 ./Platform.Exceptions/IgnoredExceptions.cs

```
using System;
using System.Collections.Concurrent;
using System.Collections.Generic;

namespace Platform.Exceptions
{
    /// <summary>
    /// <para>Contains a mechanism for notifying about the occurrence of ignored exceptions, as
    ↪  well as a mechanism for their collection.</para>
```

```
    /// <para>Содержит механизм уведомления о возникновении игнорируемых исключений, а так же механизм их сбора.</para>
    /// </summary>
    public static class IgnoredExceptions
    {
        private static readonly ConcurrentBag<Exception> _exceptionsBag = new ConcurrentBag<Exception>();

        /// <summary>
        /// <para>An event that is raised every time an exception has been ignored.</para>
        /// <para>Событие, которое генерируется каждый раз, когда исключение было проигнорировано.</para>
        /// </summary>
        public static event EventHandler<Exception> ExceptionIgnored = OnExceptionIgnored;

        /// <summary>
        /// <para>Gets an immutable collection with all collected exceptions that were ignored.</para>
        /// <para>Возвращает неизменяемую коллекцию со всеми собранными исключениями которые были проигнорированы.</para>
        /// </summary>
        public static IReadOnlyCollection<Exception> CollectedExceptions => _exceptionsBag;

        /// <summary>
        /// <para>Gets or sets a value that determines whether to collect ignored exceptions into CollectedExceptions.</para>
        /// <para>Возвращает или устанавливает значение, определяющие нужно ли собирать игнорируемые исключения в CollectedExceptions.</para>
        /// </summary>
        public static bool CollectExceptions { get; set; }

        /// <summary>
        /// <para>Raises an exception ignored event.</para>
        /// <para>Генерирует событие игнорирования исключения.</para>
        /// </summary>
        /// <param name="exception"><para>The ignored exception.</para><para>Игнорируемое исключение.</para></param>
        /// <remarks>
        /// <para>It is recommended to call this method in cases where you have a catch block, but you do not do anything with exception in it.</para>
        /// <para>Рекомендуется вызывать этот метод в тех случаях, когда у вас есть catch блок, но вы ничего не делаете в нём с исключением.</para>
        /// </remarks>
        public static void RaiseExceptionIgnoredEvent(Exception exception) => ExceptionIgnored.Invoke(null, exception);

        private static void OnExceptionIgnored(object sender, Exception exception)
        {
            if (CollectExceptions)
            {
                _exceptionsBag.Add(exception);
            }
        }
    }
}
```

1.8 ./Platform.Exceptions/Throw.cs

```
using Platform.Exceptions.ExtensionRoots;

namespace Platform.Exceptions
{
    /// <summary>
    /// <para>Contains an instance of an extensible class that can be supplemented with static helper methods by using the extension mechanism. These methods throw exceptions.</para>
    /// <para>Содержит экземпляр расширяемого класса, который можно дополнять статическими вспомогательными методами путём использования механизма расширений. Эти методы занимаются выбрасыванием исключений.</para>
    /// </summary>
    public static class Throw
    {
        /// <summary>
        /// <para>Gets an instance of the extension root class that contains helper methods for throwing exceptions.</para>
        /// <para>Возвращает экземпляр класса корня-расширения, который содержит вспомогательные методы для выбрасывания исключений.</para>
        /// </summary>
        public static readonly ThrowExtensionRoot A = new ThrowExtensionRoot();
```

```
    }
}
```

1.9 ./Platform.Exceptions/ThrowExtensions.cs

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

#pragma warning disable IDE0060 // Remove unused parameter

namespace Platform.Exceptions
{
    /// <summary>
    /// <para>Provides a set of extension methods for <see cref="ThrowExtensionRoot"/> objects.</para>
    /// <para>Предоставляет набор методов расширения для объектов <see cref="ThrowExtensionRoot"/>.</para>
    /// </summary>
    public static class ThrowExtensions
    {
        /// <summary>
        /// <para>Throws a new <see cref="System.NotSupportedException"/>.</para>
        /// <para>Выбрасывает новое <see cref="System.NotSupportedException"/>.</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>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void NotSupportedException(this ThrowExtensionRoot root) => throw new NotSupportedException();

        /// <summary>
        /// <para>Throws a new <see cref="System.NotSupportedException"/>, while returning a value of <typeparamref name="TReturn"/> type.</para>
        /// <para>Выбрасывает новое <see cref="System.NotSupportedException"/>, вовращая при этом значение типа <typeparamref name="TReturn"/>.</para>
        /// </summary>
        /// <typeparam name="TReturn"><para>The type of returned value.</para><para>Тип возвращаемого значения.</para></typeparam>
        /// <param name="root"><para>The extension root to which this method is bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
        /// <returns><para>A value of <typeparamref name="TReturn"/> type.</para><para>Значение типа <typeparamref name="TReturn"/>.</para></returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static TReturn NotSupportedExceptionAndReturn<TReturn>(this ThrowExtensionRoot root) => throw new NotSupportedException();

        /// <summary>
        /// <para>Throws a new <see cref="System.NotImplementedException"/>.</para>
        /// <para>Выбрасывает новое <see cref="System.NotImplementedException"/>.</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>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void NotImplementedException(this ThrowExtensionRoot root) => throw new NotImplementedException();

        /// <summary>
        /// <para>Throws a new <see cref="System.NotImplementedException"/>, while returning a value of <typeparamref name="TReturn"/> type.</para>
        /// <para>Выбрасывает новое <see cref="System.NotImplementedException"/>, вовращая при этом значение типа <typeparamref name="TReturn"/>.</para>
        /// </summary>
        /// <typeparam name="TReturn"><para>The type of returned value.</para><para>Тип возвращаемого значения.</para></typeparam>
        /// <param name="root"><para>The extension root to which this method is bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
        /// <returns><para>A value of <typeparamref name="TReturn"/> type.</para><para>Значение типа <typeparamref name="TReturn"/>.</para></returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static TReturn NotImplementedExceptionAndReturn<TReturn>(this ThrowExtensionRoot root) => throw new NotImplementedException();
    }
}
```

1.10 ./Platform.Exceptions.Tests/EnsuranceTests.cs

```
using System;
using Xunit;
```

```

namespace Platform.Exceptions.Tests
{
    public static class EnsuranceTests
    {
        [Fact]
        public static void ArgumentNotNullEnsuranceTest()
        {
            // Should throw an exception (even if in neighbour "Ignore" namespace it was overridden, but here this namespace is not used)
            Assert.Throws<ArgumentNullException>(() => Ensure.Always.ArgumentNotNull<object>(null, "object"));
        }
    }
}
```

1.11 ./Platform.Exceptions.Tests/Ignore/EnsureAlwaysExtensions.cs

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

namespace Platform.Exceptions.Tests.Ignore
{
    public static class EnsureAlwaysExtensions
    {
        [Conditional("DEBUG")]
        public static void ArgumentNotNull<TArgument>(this EnsureAlwaysExtensionRoot root, TArgument argument, string argumentName)
            where TArgument : class
        {
        }
    }
}
```

1.12 ./Platform.Exceptions.Tests/Ignore/IgnoredEnsuranceTests.cs

```
using Xunit;

namespace Platform.Exceptions.Tests.Ignore
{
    public static class IgnoredEnsuranceTests
    {
        [Fact]
        public static void EnsuranceIgnoredTest()
        {
            // Should not throw an exception (because logic is overriden in EnsureAlwaysExtensions that is located within the same namespace)
            // And even should be optimized out at RELEASE (because method is now marked conditional DEBUG)
            // This can be useful in performance critical situations there even an check for exception is hurting performance enough
            Ensure.Always.ArgumentNotNull<object>(null, "object");
        }
    }
}
```

Index