

LinksPlatform's Platform.Exceptions Class Library

1.1 ./Platform.Exceptions/Ensure.cs

```
1 using Platform.Exceptions.ExtensionRoots;
2
3 namespace Platform.Exceptions
4 {
5     /// <summary>
6     /// <para>Contains two extensible classes instances that can be supplemented with static
7     ///     ↪ helper methods by using the extension mechanism. These methods ensure the contract
8     ///     ↪ compliance.</para>
9     /// <para>Содержит два экземпляра расширяемых класса, которые можно дополнять статическими
10    ///     ↪ вспомогательными методами путём использования механизма расширений. Эти методы
11    ///     ↪ занимаются гарантированием соответствия контракту.</para>
12    /// </summary>
13    public static class Ensure
14    {
15        /// <summary>
16        /// <para>Gets an instance of the extension root class that contains helper methods to
17        ///     ↪ guarantee compliance with the contract.</para>
18        /// <para>Возвращает экземпляр класса корня-расширения, который содержит вспомогательные
19        ///     ↪ методы для гарантирования соответствия контракту.</para>
20        /// </summary>
21        public static readonly EnsureAlwaysExtensionRoot Always = new
22        ↪ EnsureAlwaysExtensionRoot();
23
24        /// <summary>
25        /// <para>Gets an instance of the extension root class that contains helper methods to
26        ///     ↪ guarantee compliance with the contract, but are executed only during
27        ///     ↪ debugging.</para>
28        /// <para>Возвращает экземпляр класса корня-расширения, который содержит вспомогательные
29        ///     ↪ методы для гарантирования соответствия контракту, но выполняются только во время
30        ///     ↪ отладки.</para>
31        /// </summary>
32        public static readonly EnsureOnDebugExtensionRoot OnDebug = new
33        ↪ EnsureOnDebugExtensionRoot();
34    }
35 }
```

1.2 ./Platform.Exceptions/EnsureExtensions.cs

```
1 using System;
2 using System.Diagnostics;
3 using System.Runtime.CompilerServices;
4 using Platform.Exceptions.ExtensionRoots;
5
6 #pragma warning disable IDE0060 // Remove unused parameter
7
8 namespace Platform.Exceptions
9 {
10    /// <summary>
11    /// <para>Provides a set of extension methods for <see cref="EnsureAlwaysExtensionRoot"/>
12    ///     ↪ and <see cref="EnsureOnDebugExtensionRoot"/> objects.</para>
13    /// <para>Предоставляет набор методов расширения для объектов <see
14    ///     ↪ cref="EnsureAlwaysExtensionRoot"/> и <see cref="EnsureOnDebugExtensionRoot"/>.</para>
15    /// </summary>
16    public static class EnsureExtensions
17    {
18        #region Always
19
20        /// <summary>
21        /// <para>Ensures that argument is not null. This check is performed regardless of the
22        ///     ↪ build configuration.</para>
23        /// <para>Гарантирует, что аргумент не нулевой. Эта проверка выполняется независимо
24        ///     ↪ от конфигурации сборки.</para>
25        /// </summary>
26        /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
27        ///     ↪ аргумента.</para></typeparam>
28        /// <param name="root"><para>The extension root to which this method is
29        ///     ↪ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
30        /// <param name="argument"><para>The argument.</para><para>Аргумент.</para></param>
31        /// <param name="argumentName"><para>The argument's name.</para><para>Имя
32        ///     ↪ аргумента.</para></param>
33        /// <param name="message"><para>The message of the thrown
34        ///     ↪ exception.</para><para>Сообщение выбрасываемого исключения.</para></param>
35        [MethodImpl(MethodImplOptions.AggressiveInlining)]
36        public static void ArgumentNotNull<TArgument>(this EnsureAlwaysExtensionRoot root,
37        ↪ TArgument argument, string argumentName, string message)
38        where TArgument : class
39    {
40    }
41
42    #region OnDebug
43
44    /// <summary>
45    /// <para>Ensures that argument is not null. This check is performed only during debugging.
46    ///     ↪ This method is marked with <code>ConditionalAttribute</code> and will be removed by the
47    ///     ↪ compiler if the <code>DEBUG</code> symbol is not defined.</para>
48    /// <para>Гарантирует, что аргумент не нулевой. Эта проверка выполняется только во время
49    ///     ↪ отладки. Этот метод помечен атрибутом <code>ConditionalAttribute</code> и будет
50    ///     ↪ удален компилятором, если символ <code>DEBUG</code> не определен.</para>
51    /// </summary>
52    /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
53    ///     ↪ аргумента.</para></typeparam>
54    /// <param name="root"><para>The extension root to which this method is
55    ///     ↪ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
56    /// <param name="argument"><para>The argument.</para><para>Аргумент.</para></param>
57    /// <param name="argumentName"><para>The argument's name.</para><para>Имя
58    ///     ↪ аргумента.</para></param>
59    /// <param name="message"><para>The message of the thrown
60    ///     ↪ exception.</para><para>Сообщение выбрасываемого исключения.</para></param>
61    [Conditional("DEBUG")]
62    [MethodImpl(MethodImplOptions.AggressiveInlining)]
63    public static void ArgumentNotNull<TArgument>(this EnsureOnDebugExtensionRoot root,
64    ↪ TArgument argument, string argumentName, string message)
65    where TArgument : class
66    {
67    }
68
69    #endregion
70
71    #region OnDebugOnly
72
73    /// <summary>
74    /// <para>Ensures that argument is not null. This check is performed only during debugging.
75    ///     ↪ This method is marked with <code>ConditionalAttribute</code> and will be removed by the
76    ///     ↪ compiler if the <code>DEBUG</code> symbol is not defined.</para>
77    /// <para>Гарантирует, что аргумент не нулевой. Эта проверка выполняется только во время
78    ///     ↪ отладки. Этот метод помечен атрибутом <code>ConditionalAttribute</code> и будет
79    ///     ↪ удален компилятором, если символ <code>DEBUG</code> не определен.</para>
80    /// </summary>
81    /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
82    ///     ↪ аргумента.</para></typeparam>
83    /// <param name="root"><para>The extension root to which this method is
84    ///     ↪ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
85    /// <param name="argument"><para>The argument.</para><para>Аргумент.</para></param>
86    /// <param name="argumentName"><para>The argument's name.</para><para>Имя
87    ///     ↪ аргумента.</para></param>
88    /// <param name="message"><para>The message of the thrown
89    ///     ↪ exception.</para><para>Сообщение выбрасываемого исключения.</para></param>
90    [Conditional("DEBUG")]
91    [MethodImpl(MethodImplOptions.AggressiveInlining)]
92    public static void ArgumentNotNull<TArgument>(this EnsureOnDebugOnlyExtensionRoot root,
93    ↪ TArgument argument, string argumentName, string message)
94    where TArgument : class
95    {
96    }
97
98    #endregion
99 }
```

```

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
39   → build configuration.</para>
40 /// <para>Гарантирует, что аргумент не нулевой. Эта проверка выполняется независимо
41   → от конфигурации сборки.</para>
42 /// </summary>
43 /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
44   → аргумента.</para></typeparam>
45 /// <param name="root"><para>The extension root to which this method is
46   → bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
47 /// <param name="argument"><para>The argument.</para><para>Аргумент.</para></param>
48 /// <param name="argumentName"><para>The argument's name.</para><para>Имя
49   → аргумента.</para></param>
50 [MethodImpl(MethodImplOptions.AggressiveInlining)]
51 public static void ArgumentNotNull<TArgument>(this EnsureAlwaysExtensionRoot root,
52   → TArgument argument, string argumentName) where TArgument : class =>
53   → ArgumentNotNull(root, argument, argumentName, $"Argument {argumentName} is null.");
54
55 /// <summary>
56 /// <para>Ensures that argument is not null. This check is performed regardless of the
57   → build configuration.</para>
58 /// <para>Гарантирует, что аргумент не нулевой. Эта проверка выполняется независимо
59   → от конфигурации сборки.</para>
60 /// </summary>
61 /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
62   → аргумента.</para></typeparam>
63 /// <param name="root"><para>The extension root to which this method is
64   → bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
65 /// <param name="argument"><para>The argument.</para><para>Аргумент.</para></param>
66 [MethodImpl(MethodImplOptions.AggressiveInlining)]
67 public static void ArgumentNotNull<TArgument>(this EnsureAlwaysExtensionRoot root,
68   → TArgument argument) where TArgument : class => ArgumentNotNull(root, argument, null);
69
70 /// <summary>
71 /// <para>Ensures that the argument meets the criteria. This check is performed
72   → regardless of the build configuration.</para>
73 /// <para>Гарантирует, что аргумент соответствует критерию. Эта проверка выполняется
74   → независимо от конфигурации сборки.</para>
75 /// </summary>
76 /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
77   → аргумента.</para></typeparam>
78 /// <param name="root"><para>The extension root to which this method is
79   → bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
80 /// <param name="argument"><para>The argument.</para><para>Аргумент.</para></param>
81 /// <param name="predicate"><para>A predicate that determines whether the argument meets
82   → a criterion.</para><para>Предикат определяющий, соответствует ли аргумент
83   → критерию.</para></param>
84 /// <param name="argumentName"><para>The argument's name.</para><para>Имя
85   → аргумента.</para></param>
86 /// <param name="message"><para>The message of the thrown
87   → exception.</para><para>Сообщение выбрасываемого исключения.</para></param>
88 [MethodImpl(MethodImplOptions.AggressiveInlining)]
89 public static void ArgumentMeetsCriteria<TArgument>(this EnsureAlwaysExtensionRoot root,
90   → TArgument argument, Predicate<TArgument> predicate, string argumentName, string
91   → message)
92 {
93     if (!predicate(argument))
94     {
95         throw new ArgumentException(message, argumentName);
96     }
97 }
98
99 /// <summary>
100 /// <para>Ensures that the argument meets the criteria. This check is performed
101   → regardless of the build configuration.</para>
102 /// <para>Гарантирует, что аргумент соответствует критерию. Эта проверка выполняется
103   → независимо от конфигурации сборки.</para>
104 /// </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
    → независимо 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
129   ↳ build configuration.</para>
130 /// <para>Гарантирует, что аргумент не нулевой. Эта проверка выполняется только для
131   ↳ конфигурации сборки DEBUG.</para>
132 /// </summary>
133 /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
134   ↳ аргумента.</para></typeparam>
135 /// <param name="root"><para>The extension root to which this method is
136   ↳ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
137 /// <param name="argument"><para>The argument.</para><para>Аргумент.</para></param>
138 [Conditional("DEBUG")]
139 public static void ArgumentNotNull<TArgument>(this EnsureOnDebugExtensionRoot root,
140   ↳ TArgument argument) where TArgument : class =>
141   ↳ Ensure.Always.ArgumentNotNull(argument);
142
143 /// <summary>
144 /// <para>Ensures that the argument meets the criteria. This check is performed only for
145   ↳ DEBUG build configuration.</para>
146 /// <para>Гарантирует, что аргумент соответствует критерию. Эта проверка выполняется
147   ↳ только для конфигурации сборки DEBUG.</para>
148 /// </summary>
149 /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
150   ↳ аргумента.</para></typeparam>
151 /// <param name="root"><para>The extension root to which this method is
152   ↳ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
153 /// <param name="argument"><para>The argument.</para><para>Аргумент.</para></param>
154 /// <param name="predicate"><para>A predicate that determines whether the argument meets
155   ↳ a criterion.</para><para>Предикат определяющий, соответствует ли аргумент
156   ↳ критерию.</para></param>
157 /// <param name="argumentName"><para>The argument's name.</para><para>Имя
158   ↳ аргумента.</para></param>
159 /// <param name="message"><para>The message of the thrown
160   ↳ exception.</para><para>Сообщение выбрасываемого исключения.</para></param>
161 [Conditional("DEBUG")]
162 public static void ArgumentMeetsCriteria<TArgument>(this EnsureOnDebugExtensionRoot
163   ↳ root, TArgument argument, Predicate<TArgument> predicate, string argumentName,
164   ↳ string message) => Ensure.Always.ArgumentMeetsCriteria(argument, predicate,
165   ↳ argumentName, message);
166
167 /// <summary>
168 /// <para>Ensures that the argument meets the criteria. This check is performed only for
169   ↳ DEBUG build configuration.</para>
170 /// <para>Гарантирует, что аргумент соответствует критерию. Эта проверка выполняется
171   ↳ только для конфигурации сборки DEBUG.</para>
172 /// </summary>
173 /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
174   ↳ аргумента.</para></typeparam>
175 /// <param name="root"><para>The extension root to which this method is
176   ↳ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
177 /// <param name="argument"><para>The argument.</para><para>Аргумент.</para></param>
178 /// <param name="predicate"><para>A predicate that determines whether the argument meets
179   ↳ a criterion.</para><para>Предикат определяющий, соответствует ли аргумент
180   ↳ критерию.</para></param>
181 /// <param name="argumentName"><para>The argument's name.</para><para>Имя
182   ↳ аргумента.</para></param>
183 [Conditional("DEBUG")]
184 public static void ArgumentMeetsCriteria<TArgument>(this EnsureOnDebugExtensionRoot
185   ↳ root, TArgument argument, Predicate<TArgument> predicate, string argumentName) =>
186   ↳ Ensure.Always.ArgumentMeetsCriteria(argument, predicate, argumentName);
187
188 /// <summary>
189 /// <para>Ensures that the argument meets the criteria. This check is performed only for
190   ↳ DEBUG build configuration.</para>
191 /// <para>Гарантирует, что аргумент соответствует критерию. Эта проверка выполняется
192   ↳ только для конфигурации сборки DEBUG.</para>
193 /// </summary>
194 /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
195   ↳ аргумента.</para></typeparam>
196 /// <param name="root"><para>The extension root to which this method is
197   ↳ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
198 /// <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 #endregion
173 }
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 представлено 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)

```

```

46         {
47             ex.Ignore();
48             return ExceptionStringBuilderFailed;
49         }
50     }
51
52     private static void BuildExceptionString(this StringBuilder sb, Exception exception, int
    ↪ level)
53     {
54         sb.Indent(level);
55         sb.AppendLine(exception.Message);
56         sb.Indent(level);
57         sb.AppendLine(ExceptionContentsSeparator);
58         if (exception.InnerException != null)
59         {
60             sb.Indent(level);
61             sb.AppendLine("Inner exception: ");
62             sb.BuildExceptionString(exception.InnerException, level + 1);
63         }
64         sb.Indent(level);
65         sb.AppendLine(ExceptionContentsSeparator);
66         sb.Indent(level);
67         sb.AppendLine(exception.StackTrace);
68     }
69
70     private static void Indent(this StringBuilder sb, int level)
71     {
72         sb.Append('\t', level);
73     }
74 }
75 }

```

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

```

1 namespace Platform.Exceptions.ExtensionRoots
2 {
3     /// <summary>
4     /// <para>Represents the extension root class for Ensure.Always.</para>
5     /// <para>Представляет класс корень-расширения для Ensure.Always.</para>
6     /// </summary>
7     public class EnsureAlwaysExtensionRoot
8     {
9     }
10 }

```

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

```

1 namespace Platform.Exceptions.ExtensionRoots
2 {
3     /// <summary>
4     /// <para>Represents the extension root class for Ensure.OnDebug.</para>
5     /// <para>Представляет класс корень-расширения для Ensure.OnDebug.</para>
6     /// </summary>
7     public class EnsureOnDebugExtensionRoot
8     {
9     }
10 }

```

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

```

1 namespace Platform.Exceptions.ExtensionRoots
2 {
3     /// <summary>
4     /// <para>Represents the extension root class for Throw.A.</para>
5     /// <para>Представляет класс корень-расширения для Throw.A.</para>
6     /// </summary>
7     public class ThrowExtensionRoot
8     {
9     }
10 }

```

1.7 ./Platform.Exceptions/IgnoredExceptions.cs

```

1 using System;
2 using System.Collections.Concurrent;
3 using System.Collections.Generic;
4
5 namespace Platform.Exceptions
6 {
7     /// <summary>
8     /// <para>Contains a mechanism for notifying about the occurrence of ignored exceptions, as
    ↪ wellas a mechanism for their collection.</para>

```

```

9      /// <para>Содержит механизм уведомления о возникновении игнорируемых исключений, а так же
    ↳ механизм их сбора.</para>
10     /// </summary>
11     public static class IgnoredExceptions
12     {
13         private static readonly ConcurrentBag<Exception> _exceptionsBag = new
            ↳ ConcurrentBag<Exception>();
14
15         /// <summary>
16         /// <para>An event that is raised every time an exception has been ignored.</para>
17         /// <para>Событие, которое генерируется каждый раз, когда исключение было
            ↳ проигнорировано.</para>
18         /// </summary>
19         public static event EventHandler<Exception> ExceptionIgnored = OnExceptionIgnored;
20
21         /// <summary>
22         /// <para>Gets an immutable collection with all collected exceptions that were
            ↳ ignored.</para>
23         /// <para>Возвращает неизменяемую коллекцию со всеми собранными исключениями которые
            ↳ были проигнорированы.</para>
24         /// </summary>
25         public static IReadOnlyCollection<Exception> CollectedExceptions => _exceptionsBag;
26
27         /// <summary>
28         /// <para>Gets or sets a value that determines whether to collect ignored exceptions
            ↳ into CollectedExceptions.</para>
29         /// <para>Возвращает или устанавливает значение, определяющие нужно ли собирать
            ↳ игнорируемые исключения в CollectedExceptions.</para>
30         /// </summary>
31         public static bool CollectExceptions { get; set; }
32
33         /// <summary>
34         /// <para>Raises an exception ignored event.</para>
35         /// <para>Генерирует событие игнорирования исключения.</para>
36         /// </summary>
37         /// <param name="exception"><para>The ignored exception.</para><para>Игнорируемое
            ↳ исключение.</para></param>
38         /// <remarks>
39         /// <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>
40         /// <para>Рекомендуется вызывать этот метод в тех случаях, когда у вас есть catch блок,
            ↳ но вы ничего не делаете в нём с исключением.</para>
41         /// </remarks>
42         public static void RaiseExceptionIgnoredEvent(Exception exception) =>
            ↳ ExceptionIgnored.Invoke(null, exception);
43
44         private static void OnExceptionIgnored(object sender, Exception exception)
45         {
46             if (CollectExceptions)
47             {
48                 _exceptionsBag.Add(exception);
49             }
50         }
51     }
52 }

```

1.8 ./Platform.Exceptions/Throw.cs

```

1     using Platform.Exceptions.ExtensionRoots;
2
3     namespace Platform.Exceptions
4     {
5         /// <summary>
6         /// <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>
7         /// <para>Содержит экземпляр расширяемого класса, который можно дополнять статическими
            ↳ вспомогательными методами путём использования механизма расширений. Эти методы
            ↳ занимаются выбрасыванием исключений.</para>
8         /// </summary>
9         public static class Throw
10        {
11            /// <summary>
12            /// <para>Gets an instance of the extension root class that contains helper methods for
                ↳ throwing exceptions.</para>
13            /// <para>Возвращает экземпляр класса корня-расширения, который содержит вспомогательные
                ↳ методы для выбрасывания исключений.</para>
14            /// </summary>
15            public static readonly ThrowExtensionRoot A = new ThrowExtensionRoot();

```

```
16     }
17 }
```

1.9 ./Platform.Exceptions/ThrowExtensions.cs

```
1 using System;
2 using System.Runtime.CompilerServices;
3 using Platform.Exceptions.ExtensionRoots;
4
5 #pragma warning disable IDE0060 // Remove unused parameter
6
7 namespace Platform.Exceptions
8 {
9     /// <summary>
10     /// <para>Provides a set of extension methods for <see cref="ThrowExtensionRoot"/>
11     ///   ↳ objects.</para>
12     /// <para>Предоставляет набор методов расширения для объектов <see
13     ///   ↳ cref="ThrowExtensionRoot"/>.</para>
14     /// </summary>
15     public static class ThrowExtensions
16     {
17         /// <summary>
18         /// <para>Throws a new <see cref="System.NotSupportedException"/>.</para>
19         /// <para>Выбрасывает новое <see cref="System.NotSupportedException"/>.</para>
20         /// </summary>
21         /// <param name="root"><para>The extension root to which this method is
22         ///   ↳ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
23         [MethodImpl(MethodImplOptions.AggressiveInlining)]
24         public static void NotSupportedException(this ThrowExtensionRoot root) => throw new
25         ///   ↳ NotSupportedException();
26
27         /// <summary>
28         /// <para>Throws a new <see cref="System.NotSupportedException"/>, while returning a
29         ///   ↳ value of <typeparamref name="TReturn"/> type.</para>
30         /// <para>Выбрасывает новое <see cref="System.NotSupportedException"/>, возвращая при
31         ///   ↳ этом значение типа <typeparamref name="TReturn"/>.</para>
32         /// </summary>
33         /// <typeparam name="TReturn"><para>The type of returned value.</para><para>Тип
34         ///   ↳ возвращаемого значения.</para></typeparam>
35         /// <param name="root"><para>The extension root to which this method is
36         ///   ↳ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
37         /// <returns><para>A value of <typeparamref name="TReturn"/> type.</para><para>Значение
38         ///   ↳ типа <typeparamref name="TReturn"/>.</para></returns>
39         [MethodImpl(MethodImplOptions.AggressiveInlining)]
40         public static TReturn NotSupportedExceptionAndReturn<TReturn>(this ThrowExtensionRoot
41         ///   ↳ root) => throw new NotSupportedException();
42
43         /// <summary>
44         /// <para>Throws a new <see cref="System.NotImplementedException"/>.</para>
45         /// <para>Выбрасывает новое <see cref="System.NotImplementedException"/>.</para>
46         /// </summary>
47         /// <param name="root"><para>The extension root to which this method is
48         ///   ↳ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
49         [MethodImpl(MethodImplOptions.AggressiveInlining)]
50         public static void NotImplementedException(this ThrowExtensionRoot root) => throw new
51         ///   ↳ NotImplementedException();
52
53         /// <summary>
54         /// <para>Throws a new <see cref="System.NotImplementedException"/>, while returning a
55         ///   ↳ value of <typeparamref name="TReturn"/> type.</para>
56         /// <para>Выбрасывает новое <see cref="System.NotImplementedException"/>, возвращая при
57         ///   ↳ этом значение типа <typeparamref name="TReturn"/>.</para>
58         /// </summary>
59         /// <typeparam name="TReturn"><para>The type of returned value.</para><para>Тип
60         ///   ↳ возвращаемого значения.</para></typeparam>
61         /// <param name="root"><para>The extension root to which this method is
62         ///   ↳ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
63         /// <returns><para>A value of <typeparamref name="TReturn"/> type.</para><para>Значение
64         ///   ↳ типа <typeparamref name="TReturn"/>.</para></returns>
65         [MethodImpl(MethodImplOptions.AggressiveInlining)]
66         public static TReturn NotImplementedExceptionAndReturn<TReturn>(this ThrowExtensionRoot
67         ///   ↳ root) => throw new NotImplementedException();
68     }
69 }
```

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

```
1 using System;
2 using Xunit;
```



```

3
4 namespace Platform.Exceptions.Tests
5 {
6     public static class EnsuranceTests
7     {
8         [Fact]
9         public static void ArgumentNotNullEnsuranceTest()
10        {
11            // Should throw an exception (even if in neighbour "Ignore" namespace it was
12            // ↳ overridden, but here this namespace is not used)
13            Assert.Throws<ArgumentNullException>(() =>
14                ↳ Ensure.Always.ArgumentNotNull<object>(null, "object"));
15        }
16    }
17 }

```

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

```

1 using System.Diagnostics;
2 using Platform.Exceptions.ExtensionRoots;
3
4 namespace Platform.Exceptions.Tests.Ignore
5 {
6     public static class EnsureAlwaysExtensions
7     {
8         [Conditional("DEBUG")]
9         public static void ArgumentNotNull<TArgument>(this EnsureAlwaysExtensionRoot root,
10            ↳ TArgument argument, string argumentName)
11         where TArgument : class
12         {
13         }
14     }
15 }

```

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

```

1 using Xunit;
2
3 namespace Platform.Exceptions.Tests.Ignore
4 {
5     public static class IgnoredEnsuranceTests
6     {
7         [Fact]
8         public static void EnsuranceIgnoredTest()
9         {
10            // Should not throw an exception (because logic is overridden in
11            // ↳ EnsureAlwaysExtensions that is located within the same namespace)
12            // ↳ And even should be optimized out at RELEASE (because method is now marked
13            // ↳ conditional DEBUG)
14            // This can be useful in performance critical situations there even an check for
15            // ↳ exception is hurting performance enough
16            Ensure.Always.ArgumentNotNull<object>(null, "object");
17        }
18    }
19 }

```

Index

- ./Platform.Exceptions.Tests/EnsuranceTests.cs, 8
- ./Platform.Exceptions.Tests/Ignore/EnsureAlwaysExtensions.cs, 9
- ./Platform.Exceptions.Tests/Ignore/IgnoredEnsuranceTests.cs, 9
- ./Platform.Exceptions/Ensure.cs, 1
- ./Platform.Exceptions/EnsureExtensions.cs, 1
- ./Platform.Exceptions/ExceptionExtensions.cs, 5
- ./Platform.Exceptions/ExtensionRoots/EnsureAlwaysExtensionRoot.cs, 6
- ./Platform.Exceptions/ExtensionRoots/EnsureOnDebugExtensionRoot.cs, 6
- ./Platform.Exceptions/ExtensionRoots/ThrowExtensionRoot.cs, 6
- ./Platform.Exceptions/IgnoredExceptions.cs, 6
- ./Platform.Exceptions/Throw.cs, 7
- ./Platform.Exceptions/ThrowExtensions.cs, 8