LinksPlatform's Platform.Exceptions Class Library

./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();
    }
}
```

./Platform.Exceptions/EnsureExtensions.cs

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

#pragma warning disable IDE0060 // Remove unused parameter
#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.Exceptions
{
    public static class EnsureAlwaysExtensions
    {
        #region Always

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void ArgumentNotNull<TArgument>(this EnsureAlwaysExtensionRoot root, TArgument argument, string argumentName, string message)
            where TArgument : class
        {
            if (argument == null)
            {
                throw new ArgumentNullException(argumentName, message);
            }
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void ArgumentNotNull<TArgument>(this EnsureAlwaysExtensionRoot root, TArgument argument, string argumentName) where TArgument : class =>
            ArgumentNotNull(root, argument, argumentName, $"Argument {argumentName} is null.");

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void ArgumentNotNull<TArgument>(this EnsureAlwaysExtensionRoot root, TArgument argument) where TArgument : class => ArgumentNotNull(root, argument, null);

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void ArgumentMeetsCriteria<TArgument>(this EnsureAlwaysExtensionRoot root, Predicate<TArgument> predicate, TArgument argument, string argumentName, string message)
        {
```

```
            if (!predicate(argument))
            {
                throw new ArgumentException(argumentName, message);
            }
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void ArgumentMeetsCriteria<TArgument>(this EnsureAlwaysExtensionRoot root,
        ↪   Predicate<TArgument> predicate, TArgument argument, string argumentName) =>
        ↪   ArgumentMeetsCriteria(root, predicate, argument, argumentName, $"Argument
        ↪   {argumentName} is does not meet criteria.");

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void ArgumentMeetsCriteria<TArgument>(this EnsureAlwaysExtensionRoot root,
        ↪   Predicate<TArgument> predicate, TArgument argument) => ArgumentMeetsCriteria(root,
        ↪   predicate, argument, null);

        #endregion

        #region OnDebug

        [Conditional("DEBUG")]
        public static void ArgumentNotNull<TArgument>(this EnsureOnDebugExtensionRoot root,
        ↪   TArgument argument, string argumentName, string message) where TArgument : class =>
        ↪   Ensure.Always.ArgumentNotNull(argument, argumentName, message);

        [Conditional("DEBUG")]
        public static void ArgumentNotNull<TArgument>(this EnsureOnDebugExtensionRoot root,
        ↪   TArgument argument, string argumentName) where TArgument : class =>
        ↪   Ensure.Always.ArgumentNotNull(argument, argumentName);

        [Conditional("DEBUG")]
        public static void ArgumentNotNull<TArgument>(this EnsureOnDebugExtensionRoot root,
        ↪   TArgument argument) where TArgument : class =>
        ↪   Ensure.Always.ArgumentNotNull(argument);

        [Conditional("DEBUG")]
        public static void ArgumentMeetsCriteria<TArgument>(this EnsureOnDebugExtensionRoot
        ↪   root, Predicate<TArgument> predicate, TArgument argument, string argumentName,
        ↪   string message) => Ensure.Always.ArgumentMeetsCriteria(predicate, argument,
        ↪   argumentName, message);

        [Conditional("DEBUG")]
        public static void ArgumentMeetsCriteria<TArgument>(this EnsureOnDebugExtensionRoot
        ↪   root, Predicate<TArgument> predicate, TArgument argument, string argumentName) =>
        ↪   Ensure.Always.ArgumentMeetsCriteria(predicate, argument, argumentName);

        [Conditional("DEBUG")]
        public static void ArgumentMeetsCriteria<TArgument>(this EnsureOnDebugExtensionRoot
        ↪   root, Predicate<TArgument> predicate, TArgument argument) =>
        ↪   Ensure.Always.ArgumentMeetsCriteria(predicate, argument);

        #endregion
    }
}
```

./Platform.Exceptions/ExceptionExtensions.cs

```
using System;
using System.Text;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.Exceptions
{
    public static class ExceptionExtensions
    {
        public static readonly string ExceptionContentsSeparator = "---";
        public static readonly string ExceptionStringBuildingFailed = "Unable to format
        ↪   exception.";

        public static void Ignore(this Exception exception) =>
        ↪   IgnoredExceptions.RaiseExceptionIgnoredEvent(exception);

        public static string ToStringWithAllInnerExceptions(this Exception exception)
        {
            try
            {
                var sb = new StringBuilder();
```

```
                BuildExceptionString(sb, exception, 0);
                return sb.ToString();
            }
            catch (Exception ex)
            {
                ex.Ignore();
                return ExceptionStringBuildingFailed;
            }
        }

        private static void BuildExceptionString(StringBuilder sb, Exception exception, int
        ↪ level)
        {
            sb.Append('\t', level);
            sb.Append("Exception message: ");
            sb.AppendLine(exception.Message);
            sb.Append('\t', level);
            sb.AppendLine(ExceptionContentsSeparator);
            if (exception.InnerException != null)
            {
                sb.Append('\t', level);
                sb.AppendLine("Inner Exception: ");
                BuildExceptionString(sb, exception.InnerException, level + 1);
            }
            sb.Append('\t', level);
            sb.AppendLine(ExceptionContentsSeparator);
            sb.Append('\t', level);
            sb.AppendLine(exception.StackTrace);
        }
    }
}
```

./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
    {
    }
}
```

./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
    {
    }
}
```

./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
    {
    }
}
```

./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);
            }
        }
    }
}
```

./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();
```

```
    }
}
```

./Platform.Exceptions/ThrowExtensions.cs
```
using System;
using System.Runtime.CompilerServices;
using Platform.Exceptions.ExtensionRoots;

#pragma warning disable IDE0060 // Remove unused parameter
#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.Exceptions
{
    public static class ThrowExtensions
    {
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void NotSupportedException(this ThrowExtensionRoot root) => throw new NotSupportedException();

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static T NotSupportedExceptionAndReturn<T>(this ThrowExtensionRoot root) => throw new NotSupportedException();

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void NotImplementedException(this ThrowExtensionRoot root) => throw new NotImplementedException();

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static T NotImplementedExceptionAndReturn<T>(this ThrowExtensionRoot root) => throw new NotImplementedException();
    }
}
```

./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"));
        }
    }
}
```

./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
        {
        }
    }
}
```

./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