./Ensure.cs

```
using Platform.Exceptions.ExtensionRoots;

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

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

./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
{
    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
    }
}
```

./ExceptionExtensions.cs

```
using System;
using System.Text;

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);
        }
    }
}
```

./ExtensionRoots/EnsureAlwaysExtensionRoot.cs

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

./ExtensionRoots/EnsureOnDebugExtensionRoot.cs

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

./ExtensionRoots/ThrowExtensionRoot.cs

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

./IgnoredExceptions.cs

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

namespace Platform.Exceptions
{
    /// <summary>
    /// Contains a mechanism for notifying about the occurrence of ignored exceptions, as well as a mechanism for their collection.
    /// Содержит механизм уведомления о возникновении игнорируемых исключений, а так же механизм их сбора.
    /// </summary>
    public static class IgnoredExceptions
    {
```

```
        private static readonly ConcurrentBag<Exception> _exceptionsBag = new
        ↪  ConcurrentBag<Exception>();

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

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

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

        /// <summary>
        /// Raises an exception ignored event.
        /// Генерирует событие игнорирования исключения.
        /// </summary>
        /// <param name="exception">The ignored exception. Игнорируемое исключение.</param>
        /// <remarks>
        /// 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.
        /// Рекомендуется вызывать этот метод в тех случаях, когда у вас есть catch блок, но вы
        ↪  ничего не делаете в нём с исключением.
        /// </remarks>
        public static void RaiseExceptionIgnoredEvent(Exception exception) =>
        ↪  ExceptionIgnored.Invoke(null, exception);

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

./Throw.cs

```
using Platform.Exceptions.ExtensionRoots;

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

./ThrowExtensions.cs

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

#pragma warning disable IDE0060 // Remove unused parameter
```

```
6
7  namespace Platform.Exceptions
8  {
9      public static class ThrowExtensions
10     {
11         [MethodImpl(MethodImplOptions.AggressiveInlining)]
12         public static void NotSupportedException(this ThrowExtensionRoot root) => throw new
               ↪  NotSupportedException();
13
14         [MethodImpl(MethodImplOptions.AggressiveInlining)]
15         public static T NotSupportedExceptionAndReturn<T>(this ThrowExtensionRoot root) => throw
               ↪  new NotSupportedException();
16
17         [MethodImpl(MethodImplOptions.AggressiveInlining)]
18         public static void NotImplementedException(this ThrowExtensionRoot root) => throw new
               ↪  NotImplementedException();
19
20         [MethodImpl(MethodImplOptions.AggressiveInlining)]
21         public static T NotImplementedExceptionAndReturn<T>(this ThrowExtensionRoot root) =>
               ↪  throw new NotImplementedException();
22     }
23 }
```

Index