./**EnsureExtensions.cs**

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

#pragma warning disable IDE0060 // Remove unused parameter

namespace Platform.Ranges
{
    public static class EnsureExtensions
    {
        private const string DefaultMaximumShouldBeGreaterOrEqualToMinimumMessage = "Maximum should be greater or equal to minimum.";

        #region Always

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void MaximumArgumentIsGreaterOrEqualToMinimum<T>(this EnsureAlwaysExtensionRoot root, T minimum, T maximum) => MaximumArgumentIsGreaterOrEqualToMinimum(root, minimum, maximum, nameof(maximum));

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void MaximumArgumentIsGreaterOrEqualToMinimum<T>(this EnsureAlwaysExtensionRoot root, T minimum, T maximum, string argumentName) => MaximumArgumentIsGreaterOrEqualToMinimum(root, minimum, maximum, nameof(maximum), DefaultMaximumShouldBeGreaterOrEqualToMinimumMessage);

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void MaximumArgumentIsGreaterOrEqualToMinimum<T>(this EnsureAlwaysExtensionRoot root, T minimum, T maximum, string argumentName, string message)
        {
            string messageBuilder() => message;
            MaximumArgumentIsGreaterOrEqualToMinimum(root, minimum, maximum, argumentName, messageBuilder);
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void MaximumArgumentIsGreaterOrEqualToMinimum<T>(this EnsureAlwaysExtensionRoot root, T minimum, T maximum, string argumentName, Func<string> messageBuilder)
        {
            if (Comparer<T>.Default.Compare(maximum, minimum) < 0)
            {
                throw new ArgumentException(messageBuilder(), argumentName);
            }
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void ArgumentInRange<T>(this EnsureAlwaysExtensionRoot root, T argumentValue, T minimum, T maximum) => ArgumentInRange(root, argumentValue, new Range<T>(minimum, maximum), null);

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void ArgumentInRange<T>(this EnsureAlwaysExtensionRoot root, T argumentValue, T minimum, T maximum, string argumentName) => ArgumentInRange(root, argumentValue, new Range<T>(minimum, maximum), argumentName);

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void ArgumentInRange<T>(this EnsureAlwaysExtensionRoot root, T argumentValue, Range<T> range) => ArgumentInRange(root, argumentValue, range, null);

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void ArgumentInRange<T>(this EnsureAlwaysExtensionRoot root, T argumentValue, Range<T> range, string argumentName)
        {
            string messageBuilder() => $"Argument value [{argumentValue}] is out of range {range}.";
            ArgumentInRange(root, argumentValue, range, argumentName, messageBuilder);
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void ArgumentInRange<T>(this EnsureAlwaysExtensionRoot root, T argumentValue, Range<T> range, string argumentName, string message)
```

```
58          {
59              string messageBuilder() => message;
60              ArgumentInRange(root, argumentValue, range, argumentName, messageBuilder);
61          }
62
63          [MethodImpl(MethodImplOptions.AggressiveInlining)]
64          public static void ArgumentInRange<T>(this EnsureAlwaysExtensionRoot root, T
            ↪  argumentValue, Range<T> range, string argumentName, Func<string> messageBuilder)
65          {
66              if (!range.ContainsValue(argumentValue))
67              {
68                  throw new ArgumentOutOfRangeException(argumentName, argumentValue,
                    ↪  messageBuilder());
69              }
70          }
71
72          #endregion
73
74          #region OnDebug
75
76          [Conditional("DEBUG")]
77          public static void MaximumArgumentIsGreaterOrEqualToMinimum<T>(this
            ↪  EnsureOnDebugExtensionRoot root, T minimum, T maximum) =>
            ↪  Ensure.Always.MaximumArgumentIsGreaterOrEqualToMinimum(minimum, maximum, null);
78
79          [Conditional("DEBUG")]
80          public static void MaximumArgumentIsGreaterOrEqualToMinimum<T>(this
            ↪  EnsureOnDebugExtensionRoot root, T minimum, T maximum, string argumentName) =>
            ↪  Ensure.Always.MaximumArgumentIsGreaterOrEqualToMinimum(minimum, maximum,
            ↪  argumentName);
81
82          [Conditional("DEBUG")]
83          public static void MaximumArgumentIsGreaterOrEqualToMinimum<T>(this
            ↪  EnsureOnDebugExtensionRoot root, T minimum, T maximum, string argumentName, string
            ↪  message) => Ensure.Always.MaximumArgumentIsGreaterOrEqualToMinimum(minimum, maximum,
            ↪  argumentName, message);
84
85          [Conditional("DEBUG")]
86          public static void MaximumArgumentIsGreaterOrEqualToMinimum<T>(this
            ↪  EnsureOnDebugExtensionRoot root, T minimum, T maximum, string argumentName,
            ↪  Func<string> messageBuilder) =>
            ↪  Ensure.Always.MaximumArgumentIsGreaterOrEqualToMinimum(minimum, maximum,
            ↪  argumentName, messageBuilder);
87
88          [Conditional("DEBUG")]
89          public static void ArgumentInRange<T>(this EnsureOnDebugExtensionRoot root, T
            ↪  argumentValue, T minimum, T maximum) => Ensure.Always.ArgumentInRange(argumentValue,
            ↪  new Range<T>(minimum, maximum), null);
90
91          [Conditional("DEBUG")]
92          public static void ArgumentInRange<T>(this EnsureOnDebugExtensionRoot root, T
            ↪  argumentValue, T minimum, T maximum, string argumentName) =>
            ↪  Ensure.Always.ArgumentInRange(argumentValue, new Range<T>(minimum, maximum),
            ↪  argumentName);
93
94          [Conditional("DEBUG")]
95          public static void ArgumentInRange<T>(this EnsureOnDebugExtensionRoot root, T argument,
            ↪  Range<T> range) => Ensure.Always.ArgumentInRange(argument, range, null);
96
97          [Conditional("DEBUG")]
98          public static void ArgumentInRange<T>(this EnsureOnDebugExtensionRoot root, T argument,
            ↪  Range<T> range, string argumentName) => Ensure.Always.ArgumentInRange(argument,
            ↪  range, argumentName);
99
100         [Conditional("DEBUG")]
101         public static void ArgumentInRange<T>(this EnsureOnDebugExtensionRoot root, T argument,
            ↪  Range<T> range, string argumentName, string message) =>
            ↪  Ensure.Always.ArgumentInRange(argument, range, argumentName, message);
102
103         [Conditional("DEBUG")]
104         public static void ArgumentInRange<T>(this EnsureOnDebugExtensionRoot root, T argument,
            ↪  Range<T> range, string argumentName, Func<string> messageBuilder) =>
            ↪  Ensure.Always.ArgumentInRange(argument, range, argumentName, messageBuilder);
105
106         #endregion
107     }
108 }
```

./Range.cs

```
using System;
using System.Collections.Generic;
using Platform.Exceptions;

namespace Platform.Ranges
{
    /// <summary>
    /// Represents a range between minumum and maximum values.
    /// Представляет диапазон между минимальным и максимальным значениями.
    /// </summary>
    /// <remarks>
    /// Based on http://stackoverflow.com/questions/5343006/is-there-a-c-sharp-type-for-representing-an-integer-range
    /// </remarks>
    public struct Range<T> : IEquatable<Range<T>>
    {
        private static readonly Comparer<T> _comparer = Comparer<T>.Default;
        private static readonly EqualityComparer<T> _equalityComparer = EqualityComparer<T>.Default;

        /// <summary>
        /// Returns minimum value of the range.
        /// Возвращает минимальное значение диапазона.
        /// </summary>
        public readonly T Minimum;

        /// <summary>
        /// Returns maximum value of the range.
        /// Возвращает максимальное значение диапазона.
        /// </summary>
        public readonly T Maximum;

        /// <summary>
        /// Initializes a new instance of the Range class.
        /// Инициализирует новый экземпляр класса Range.
        /// </summary>
        /// <param name="minimumAndMaximum">Single value for both Minimum and Maximum fields. Одно значение для полей Minimum и Maximum.</param>
        public Range(T minimumAndMaximum)
        {
            Minimum = minimumAndMaximum;
            Maximum = minimumAndMaximum;
        }

        /// <summary>
        /// Initializes a new instance of the Range class.
        /// Инициализирует новый экземпляр класса Range.
        /// </summary>
        /// <param name="minimum">The minimum value of the range. Минимальное значение диапазона.</param>
        /// <param name="maximum">The maximum value of the range. Максимальное значение диапазона.</param>
        /// <exception cref="ArgumentException">Thrown when maximum is less than minimum.</exception>
        public Range(T minimum, T maximum)
        {
            Ensure.Always.MaximumArgumentIsGreaterOrEqualToMinimum(minimum, maximum, nameof(maximum));
            Minimum = minimum;
            Maximum = maximum;
        }

        /// <summary>
        /// Presents the Range in readable format.
        /// Представляет диапазон в удобном для чтения формате.
        /// </summary>
        /// <returns>String representation of the Range. Строковое представление диапазона.</returns>
        public override string ToString() => $"[{Minimum}, {Maximum}]";

        /// <summary>
        /// Determines if the provided value is inside the range.
        /// Определяет, находится ли указанное значение внутри диапазона.
        /// </summary>
        /// <param name="value">The value to test. Значение для проверки.</param>
        /// <returns>True if the value is inside Range, else false. True, если значение находится внутри диапазона, иначе false.</returns>
```

```
69          public bool ContainsValue(T value) => _comparer.Compare(Minimum, value) <= 0 &&
            ↪  _comparer.Compare(Maximum, value) >= 0;
70
71          /// <summary>
72          /// Determines if this Range is inside the bounds of another range.
73          /// Определяет, находится ли этот диапазон в пределах другого диапазона.
74          /// </summary>
75          /// <param name="range">The parent range to test on. Родительский диапазон для
            ↪  проверки.</param>
76          /// <returns>True if range is inclusive, else false. True, если диапазон включен, иначе
            ↪  false.</returns>
77          public bool IsInsideRange(Range<T> range) => range.ContainsRange(this);
78
79          /// <summary>
80          /// Determines if another range is inside the bounds of this range.
81          /// Определяет, находится ли другой диапазон внутри границ этого диапазона.
82          /// </summary>
83          /// <param name="range">The child range to test. Дочерний диапазон для проверки.</param>
84          /// <returns>True if range is inside, else false. True, если диапазон находится внутри,
            ↪  иначе false.</returns>
85          public bool ContainsRange(Range<T> range) => ContainsValue(range.Minimum) &&
            ↪  ContainsValue(range.Maximum);
86
87          /// <summary>
88          /// Indicates whether the current range is equal to another range.
89          /// Определяет, равен ли текущий диапазон другому диапазону.
90          /// </summary>
91          /// <param name="other">A range to compare with this range. Диапазон для сравнения с
            ↪  этим диапазоном.</param>
92          /// <returns>True if the current range is equal to the other range; otherwise, false.
            ↪  True, если текущий диапазон равен другому диапазону; иначе false.</returns>
93          public bool Equals(Range<T> other) => _equalityComparer.Equals(Minimum, other.Minimum)
            ↪  && _equalityComparer.Equals(Maximum, other.Maximum);
94      }
95  }
```

Index