CommandParser

Boris Kapustík, Ricardo Bolemant

```
    private static readonly Type[] supportedTypes = ...;

    /// <summary> Takes an empty command into which parameters will be filled in.
    1 reference | Rikib1999, 34 days ago | 1 author, 1 change
    public static T Parse(string[] args, T commandInstance)
    {
        if (commandInstance == null) throw new NullReferenceException(nameof(commandInstance));
        if (!typeof(T).GetInterfaces().Contains(typeof(ICommandDefinition))) throw new MissingInterfaceException("Class " + typeof(T).Name + " does not implement ICommandDefinition interface.")
        if (args.Length < 1 || args is null) throw new ArgumentException("Command can not be null or empty.");

        ParseOptions(args[1..], commandInstance);
        ParseArguments(args, commandInstance);

        return commandInstance;
    }
```

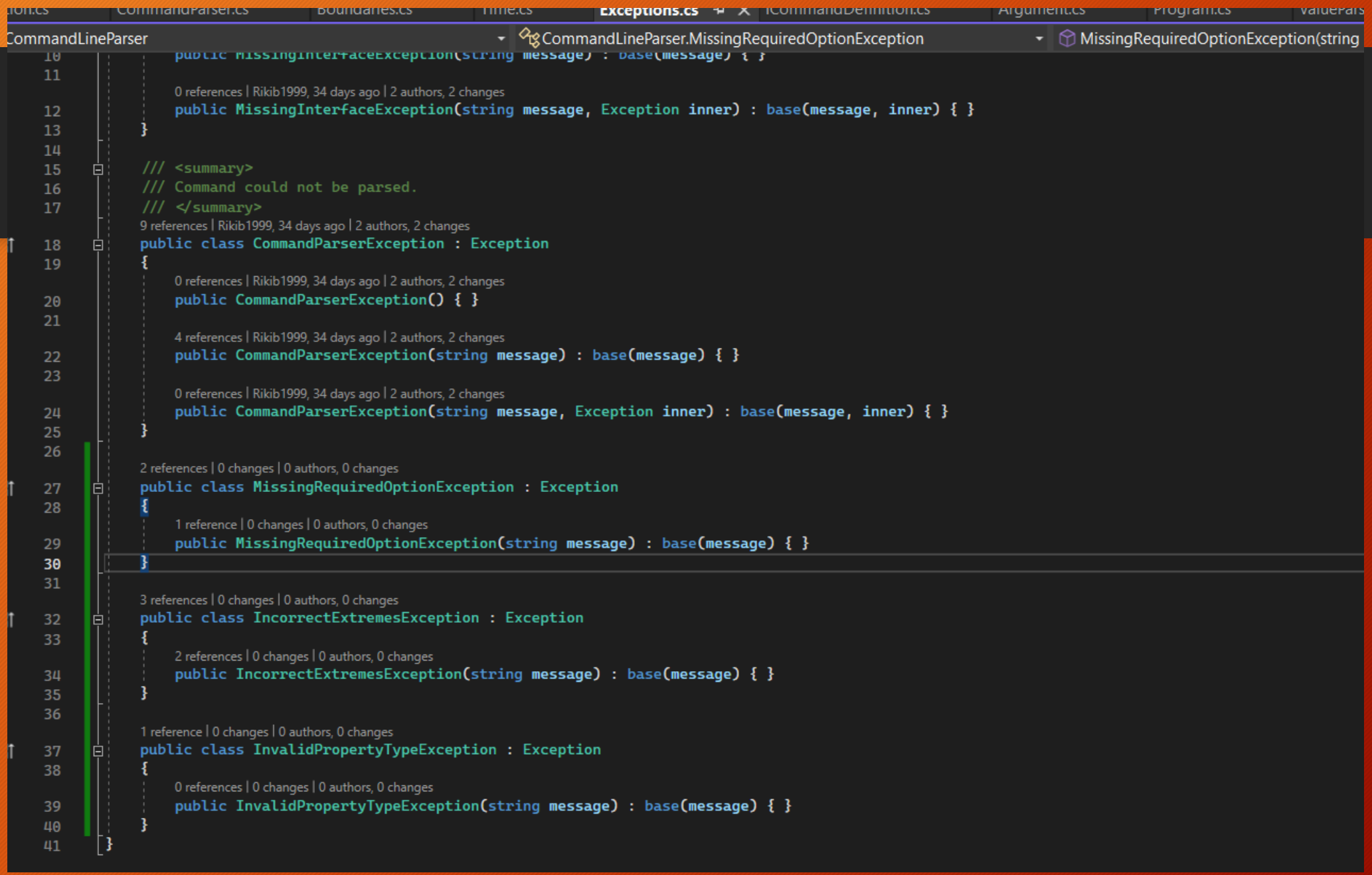
```
/// <summary>
/// Property attribute for defining command options.
/// </summary>
[AttributeUsage(AttributeTargets.Property, Inherited = false, AllowMultiple = false)]

public class Option : Attribute
{

    /// <param name="names"></param>
    /// <exception cref="ArgumentException"></exception>
    7 references | Ricardo Bolemant, 53 days ago | 1 author, 1 change
    public Option(params string[] names) ...

    #region positional arguments

    readonly string[] names;
    /// <summary>
    /// Name of the option with all its synonyms.
    /// </summary>

    public string[] Names { get { return names; } }

    #endregion

    #region named arguments


    public bool IsRequired { get; set; } = false;


    public string HelpText { get; set; } = "";

    /// <summary>
    /// Minimal numer of parameters, default is 0.
    /// </summary>
    4 references | Ricardo Bolemant, 53 days ago | 1 author, 1 change
    public int MinParameterCount { get; set; } = 0;

    /// <summary>
    /// Maximal numer of parameters, default is MaxValue.
    /// </summary>
    11 references | Ricardo Bolemant, 53 days ago | 1 author, 1 change
```

```
{
    3 references | kapustb, 34 days ago | 2 authors, 4 changes
    public class Time : ICommandDefinition
    {
        [Option(names: new string[] { "-f", "--format" }
            , HelpText = "Specify output format, possibly overriding the format specified in the environment variable TIME."
            , MinParameterCount = 1
            , MaxParameterCount = 1
        )]
        0 references | kapustb, 34 days ago | 1 author, 2 changes
        public string Format { get; set; }

        [Option(names: new string[] { "-p", "--portability" }
            , HelpText = "Use the portable output format."
            , MaxParameterCount = 0
        )]
        0 references | kapustb, 34 days ago | 1 author, 2 changes
        public object Portability { get; set; }

        [Option(names: new string[] { "-o", "--output" }
            , HelpText = "Do not send the results to stderr, but overwrite the specified file."
            , MinParameterCount = 1
            , MaxParameterCount = 1
        )]
        0 references | kapustb, 34 days ago | 1 author, 2 changes
        public string Output { get; set; }

        [Option(names: new string[] { "-a", "--append" }, HelpText = "(Used together with -o.) Do not overwrite but append."
            , MaxParameterCount = 0
            , Dependencies = new string[] { "-o" }
        )]
        0 references | kapustb, 34 days ago | 1 author, 2 changes
        public object Append { get; set; }

        [Option(names: new string[] { "-v", "--verbose" }
            , HelpText = "Give very verbose output about all the program knows about."
            , MaxParameterCount = 0
        )]
        0 references | kapustb, 34 days ago | 1 author, 3 changes
        public string Verbose { get; set; }

        [Argument(order: 0, IsRequired = true)]
        0 references | kapustb, 49 days ago | 1 author, 1 change
        public string Command { get; set; }
```

```
private static T ParseOptions(string [] command, T commandInstance)
{
    PropertyInfo[] properties = typeof(T).GetProperties();

    var propertyNames = properties
        .Select(x => x.GetCustomAttributes<Option>(false))
        .FirstOrDefault(defaultValue: null)
        .SelectMany(x => x.Names);

    foreach (PropertyInfo property in properties)
    {
        Option option = property.GetCustomAttributes<Option>(false).FirstOrDefault(defaultValue: null);

        if (option == null) continue;

        Type propType = property.PropertyType;

        CheckIsTypeSupported(option, propType);

        bool isEnummerable = CheckIsEnummerable(property, propType, option);

        Type internalType = isEnummerable ? propType.GenericTypeArguments[0] : propType;

        CheckNameValidity(option);

        var indexOfOptionInCommand = FindIndexOfOption(command, option);

        //If the property is missing in the command than it stays null

        CheckIsRequired(option, indexOfOptionInCommand);

        CheckExtremes(option, isEnummerable);

        int commandIndex = indexOfOptionInCommand;

        var valueOfProperty = Activator.CreateInstance(propType);
        valueOfProperty = TryAssignValue(valueOfProperty, isEnummerable, option, propType, command, indexOfOptionInCommand, ref commandIndex, propertyNames);

        Type boundariesType = typeof(Boundaries<>).MakeGenericType(internalType);
        var boundaries = Activator.CreateInstance(boundariesType);
```

```
1 reference | 0 changes | 0 authors, 0 changes
private static void CheckIsTypeSupported(Option option, Type propType)
{
    if (option.MaxParameterCount > 0)
    {
        bool isSupported = false;
        foreach (Type t in supportedTypes)
        {
            if (propType == t)
            {
                isSupported = true;
                break;
            }
        }

        if (!isSupported)
        {
            throw new CommandParserException("Option " + option.Names[0] + " can not be parsed. Type of option [" + propType.Name + "] is not supported.");
        }
    }
}
```

```
1 reference | 0 changes | 0 authors, 0 changes
private static bool CheckIsEnummerable(PropertyInfo property, Type propType, Option option)
{
    bool isEnumerable = property is IEnumerable && propType.IsGenericType;

    if (!isEnumerable && option.MaxParameterCount > 1)
    {
        throw new CommandParserException("Option " + option.Names[0] + " can not be parsed. Option supports multiple arguments but it is not a collection.");
    }
    return true;
}
```


Exceptions.cs
CommandLineParser
CommandLineParser.MissingRequiredOptionException
MissingRequiredOptionException(string
0 references | Rikib1999, 34 days ago | 2 authors, 2 changes
public MissingInterfaceException(string message, Exception inner) : base(message, inner) { }
}
/// <summary>
/// Command could not be parsed.
/// </summary>
9 references | Rikib1999, 34 days ago | 2 authors, 2 changes
public class CommandParserException : Exception
{
0 references | Rikib1999, 34 days ago | 2 authors, 2 changes
public CommandParserException() { }
4 references | Rikib1999, 34 days ago | 2 authors, 2 changes
public CommandParserException(string message) : base(message) { }
0 references | Rikib1999, 34 days ago | 2 authors, 2 changes
public CommandParserException(string message, Exception inner) : base(message, inner) { }
}
2 references | 0 changes | 0 authors, 0 changes
public class MissingRequiredOptionException : Exception
{
1 reference | 0 changes | 0 authors, 0 changes
public MissingRequiredOptionException(string message) : base(message) { }
}
3 references | 0 changes | 0 authors, 0 changes
public class IncorrectExtremesException : Exception
{
2 references | 0 changes | 0 authors, 0 changes
public IncorrectExtremesException(string message) : base(message) { }
}
1 reference | 0 changes | 0 authors, 0 changes
public class InvalidPropertyTypeException : Exception
{
0 references | 0 changes | 0 authors, 0 changes
public InvalidPropertyTypeException(string message) : base(message) { }
}
}

```
foreach (PropertyInfo property in properties)
{
    Option option = property.GetCustomAttributes<Option>(false).FirstOrDefault(defaultValue: null);

    if (option == null) continue;

    Type propType = property.PropertyType;

    CheckIsTypeSupported(option, propType);

    bool isEnummerable = CheckIsEnummerable(property, propType, option);

    Type internalType = isEnummerable ? propType.GenericTypeArguments[0] : propType;

    CheckNameValidity(option);

    var indexOfOptionInCommand = FindIndexOfOption(command, option);

    //If the property is missing in the command than it stays null

    CheckIsRequired(option, indexOfOptionInCommand);

    CheckExtremes(option, isEnummerable);

    int commandIndex = indexOfOptionInCommand;

    var valueOfProperty = Activator.CreateInstance(propType);
    valueOfProperty = TryAssignValue(valueOfProperty, isEnummerable, option, propType, command, indexOfOptionInCommand, ref commandIndex, propertyNames);

    Type boundariesType = typeof(Boundaries<>).MakeGenericType(internalType);
    var boundaries = Activator.CreateInstance(boundariesType);
    boundaries = property
        .GetCustomAttributes()
        .FirstOrDefault(x => x.GetType() == boundariesType.GetType());

    if (boundaries is not null)
    {
        CheckBoundaries(valueOfProperty, isEnummerable, option, propType, command, indexOfOptionInCommand, ref commandIndex, propertyNames);
    }
```

```
1 reference | 0 changes | 0 authors, 0 changes
private static void CheckNameValidity(Option option)
{
    foreach (string name in option.Names)
    {
        if (string.IsNullOrEmpty(name) || name.Length ≤ 1 || !name.StartsWith('-')) throw new CommandParserException(name + " is not a valid name for an option."
    }
}
```

```
1 reference | 0 changes | 0 authors, 0 changes
private static int FindIndexOfOption(string[] command, Option option)
{
    foreach (var name in option.Names)
    {
        int indexOfOption = Array.IndexOf(command, name);
        if (indexOfOption ≠ -1)
        {
            return indexOfOption;
        }
    }
    return -1;
}
```

```
1 reference | 0 changes | 0 authors, 0 changes
private static void CheckIsRequired(Option option, int indexOfOption)
{
    if (indexOfOption == -1 && option.IsRequired)
    {
        throw new MissingRequiredOptionException("Missing required option " + option.Names[0]);
    }
}
```

```
foreach (PropertyInfo property in properties)
{
    Option option = property.GetCustomAttributes<Option>(false).FirstOrDefault(defaultValue: null);

    if (option == null) continue;

    Type propType = property.PropertyType;

    CheckIsTypeSupported(option, propType);

    bool isEnummerable = CheckIsEnummerable(property, propType, option);

    Type internalType = isEnummerable ? propType.GenericTypeArguments[0] : propType;

    CheckNameValidity(option);

    var indexOfOptionInCommand = FindIndexOfOption(command, option);

    //If the property is missing in the command than it stays null

    CheckIsRequired(option, indexOfOptionInCommand);

    CheckExtremes(option, isEnummerable);

    int commandIndex = indexOfOptionInCommand;

    var valueOfProperty = Activator.CreateInstance(propType);
    valueOfProperty = TryAssignValue(valueOfProperty, isEnummerable, option, propType, command, indexOfOptionInCommand, ref commandIndex, propertyNames);

    Type boundariesType = typeof(Boundaries<>).MakeGenericType(internalType);
    var boundaries = Activator.CreateInstance(boundariesType);
    boundaries = property
        .GetCustomAttributes()
        .FirstOrDefault(x => x.GetType() == boundariesType.GetType());

    if (boundaries is not null)
    {
        CheckBoundaries(valueOfProperty, isEnummerable, option, propType, command, indexOfOptionInCommand, ref commandIndex, propertyNames);
    }
```

```
1 reference | 0 changes | 0 authors, 0 changes
private static void CheckExtremes(Option option, bool isEnummerable)
{
    if (option.MinParameterCount < 0 || option.MaxParameterCount < 0)
    {
        throw new IncorrectExtremesException("Min and MaxParameterCount cannot be negative");
    }

    if (!isEnummerable && option.MinParameterCount > 1)
    {
        throw new IncorrectExtremesException("Non enummerable type cannot have MinParameterCount greater than 1");
    }
}
```

```
1 reference | 0 changes | 0 authors, 0 changes
private static object? TryAssignValue(object? valueOfProperty, bool isEnummerable, Option option, Type propType, string[] command, int indexOfOptionInCommand, ref int commandIndex, IEnumerable<stri
{
    valueOfProperty = TryAssignPropertyValue(valueOfProperty, isEnummerable, option, propType, command, ref commandIndex);

    if (isEnummerable)
    {
        while (!IsDifferentOption(command[commandIndex], option, propertyNames))
        {
            //option arguments start at indexOfOption + 1
            commandIndex++;

            if (commandIndex ≥ command.Length)
            {
                break;
            }
            if (commandIndex ≥ option.MaxParameterCount + indexOfOptionInCommand + 1)
            {
                break;
            }
            //Add value to enummerable
        }
    }

    return valueOfProperty;
}
```

```
1 reference | 0 changes | 0 authors, 0 changes
private static object? TryAssignPropertyValue(object? valueOfProperty, bool isEnummerable, Option option, Type propType, string[] command, ref int indexOfOptionInCommand)
{
    if (!isEnummerable && option.MaxParameterCount == 1)
    {
        if (option.MaxParameterCount == 1)
        {
            var constructor = propType
                .GetConstructors()
                .FirstOrDefault(x =>
                    x.GetParameters().Length == 1
                    && x.GetParameters()[0].ParameterType == typeof(string)
                );

            if (constructor == null)
            {
                throw new InvalidPropertyTypeException("Specified property doesn't implement constructor which takes string as a single argument");
            }

            valueOfProperty = constructor?.Invoke(new object[] { command[indexOfOptionInCommand + 1] });
        }
    }
    return valueOfProperty;
}
```

```
foreach (PropertyInfo property in properties)
{
    Option option = property.GetCustomAttributes<Option>(false).FirstOrDefault(defaultValue: null);

    if (option == null) continue;

    Type propType = property.PropertyType;

    CheckIsTypeSupported(option, propType);

    bool isEnummerable = CheckIsEnummerable(property, propType, option);

    Type internalType = isEnummerable ? propType.GenericTypeArguments[0] : propType;

    CheckNameValidity(option);

    var indexOfOptionInCommand = FindIndexOfOption(command, option);

    //If the property is missing in the command than it stays null

    CheckIsRequired(option, indexOfOptionInCommand);

    CheckExtremes(option, isEnummerable);

    int commandIndex = indexOfOptionInCommand;

    var valueOfProperty = Activator.CreateInstance(propType);
    valueOfProperty = TryAssignValue(valueOfProperty, isEnummerable, option, propType, command, indexOfOptionInCommand, ref commandIndex, propertyNames);

    Type boundariesType = typeof(Boundaries<>).MakeGenericType(internalType);
    var boundaries = Activator.CreateInstance(boundariesType);
    boundaries = property
        .GetCustomAttributes()
        .FirstOrDefault(x => x.GetType() == boundariesType.GetType());

    if (boundaries is not null)
    {
        CheckBoundaries(valueOfProperty, isEnummerable, option, propType, command, indexOfOptionInCommand, ref commandIndex, propertyNames);
    }
```

What we need to implement

```
            CheckIsRequired(option, indexOfOptionInCommand);

            CheckExtremes(option, isEnummerable);

            int commandIndex = indexOfOptionInCommand;

            var valueOfProperty = Activator.CreateInstance(propType);
            valueOfProperty = TryAssignValue(valueOfProperty, isEnummerable, option, propType, command, indexOfOptionInCommand, ref commandIndex, propertyNames);

            Type boundariesType = typeof(Boundaries<>).MakeGenericType(internalType);
            var boundaries = Activator.CreateInstance(boundariesType);
            boundaries = property
                .GetCustomAttributes()
                .FirstOrDefault(x => x.GetType() == boundariesType.GetType());

            if (boundaries is not null)
            {
                CheckBoundaries(valueOfProperty, isEnummerable, option, propType, command, indexOfOptionInCommand, ref commandIndex, propertyNames);
            }

            //delete option arguments, so parsing plain arguments will be easier, just skipping - or --
        }

        return commandInstance;
    }

    1 reference | 0 changes | 0 authors, 0 changes
    private static T ParseArguments(string [] command, T commandInstance)...
}
```

Extendability options

- Custom Comparison methods
- Custom Constructors
- Custom parse methods