```
namespace System.CommandLine
{
    public sealed partial class ArgumentArity
    {
        public ArgumentArity(int minimumNumberOfValues, int maximumNumberOfValues){}
        public static ArgumentArity ExactlyOne { get{} }
        public int MaximumNumberOfValues { get{} }
        public int MinimumNumberOfValues { get{} }
        public static ArgumentArity OneOrMore { get{} }
        public static ArgumentArity Zero { get{} }
        public static ArgumentArity ZeroOrMore { get{} }
        public static ArgumentArity ZeroOrOne { get{} }
        public bool Equals(ArgumentArity other){}
        public override bool? Equals(object? obj){}
        public override int GetHashCode(){}
    }

    public static partial class ArgumentValidation
    {
        public static CliArgument<DirectoryInfo> AcceptExistingOnly(
            this CliArgument<DirectoryInfo> argument){}

        public static CliArgument<FileInfo> AcceptExistingOnly(
            this CliArgument<FileInfo> argument){}

        public static CliArgument<FileSystemInfo> AcceptExistingOnly(
            this CliArgument<FileSystemInfo> argument){}

        public static CliArgument<T> AcceptExistingOnly<T>(
            this CliArgument<T> argument) where T : IEnumerable<FileSystemInfo>{}
    }

    public abstract partial class CliArgument : CliSymbol
    {
        internal CliArgument(){}
        public ArgumentArity Arity { get{} set{} }

        public List<Func<Completions.CompletionContext,
            IEnumerable<Completions.CompletionItem>>> CompletionSources { get{} }

        public abstract bool HasDefaultValue { get{} }
        public string? HelpName { get{} set{} }
        public List<Action<Parsing.ArgumentResult>> Validators { get{} }
        public abstract Type ValueType { get{} }

        public override IEnumerable<Completions.CompletionItem> GetCompletions(
            Completions.CompletionContext context){}

        public object? GetDefaultValue(){}
        public override string ToString(){}
    }

    public partial class CliArgument<T> : CliArgument
    {
        public CliArgument(string name){}
        public Func<Parsing.ArgumentResult, T?>? CustomParser { get{} set{} }
        public Func<Parsing.ArgumentResult, T>? DefaultValueFactory { get{} set{} }
        public override bool HasDefaultValue { get{} }
        public override Type ValueType { get{} }
        public void AcceptLegalFileNamesOnly(){}
        public void AcceptLegalFilePathsOnly(){}
        public void AcceptOnlyFromAmong(params string[] values){}
    }
```

```
public partial class CliCommand : CliSymbol
{
    public CliCommand(string name, string? description = null){}
    public Invocation.CliAction? Action { get{} set{} }
    public ICollection<string> Aliases { get{} }
    public IList<CliArgument> Arguments { get{} }
    public IEnumerable<CliSymbol> Children { get{} }
    public IList<CliOption> Options { get{} }
    public IList<CliCommand> Subcommands { get{} }
    public bool TreatUnmatchedTokensAsErrors { get{} set{} }
    public List<Action<Parsing.CommandResult>> Validators { get{} }
    public void Add(CliArgument argument){}
    public void Add(CliCommand command){}
    public void Add(CliOption option){}

    public override IEnumerable<Completions.CompletionItem> GetCompletions(
        Completions.CompletionContext context){}

    public IEnumerator<CliSymbol> GetEnumerator(){}
    public ParseResult Parse(IReadOnlyList<string> args, CliConfiguration? configuration = null){}
    public ParseResult Parse(string commandLine, CliConfiguration? configuration = null){}
    public void SetAction(Action<ParseResult> action){}
    public void SetAction(Func<ParseResult, int> action){}
    public void SetAction(Func<ParseResult, CancellationToken, Task<int>> action){}
    public void SetAction(Func<ParseResult, CancellationToken, Task> action){}
}


public partial class CliConfiguration
{
    public CliConfiguration(CliCommand rootCommand){}
    public bool EnableDefaultExceptionHandler { get{} set{} }
    public bool EnablePosixBundling { get{} set{} }
    public TextWriter Error { get{} set{} }
    public TextWriter Output { get{} set{} }
    public TimeSpan? ProcessTerminationTimeout { get{} set{} }
    public Parsing.TryReplaceToken? ResponseFileTokenReplacer { get{} set{} }
    public CliCommand RootCommand { get{} }
    public int Invoke(string commandLine){}
    public int Invoke(string[] args){}
    public Task<int> InvokeAsync(string commandLine, CancellationToken cancellationToken = null){}
    public Task<int> InvokeAsync(string[] args, CancellationToken cancellationToken = null){}
    public ParseResult Parse(IReadOnlyList<string> args){}
    public ParseResult Parse(string commandLine){}
    public void ThrowIfInvalid(){}
}

public partial class CliConfigurationException
{
    public CliConfigurationException(string message){}
}

public partial class CliDirective : CliSymbol
{
    public CliDirective(string name){}
    public virtual Invocation.CliAction? Action { get{} set{} }

    public override IEnumerable<Completions.CompletionItem> GetCompletions(
        Completions.CompletionContext context){}
}
```

```
public abstract partial class CliOption : CliSymbol
{
    internal CliOption(){}
    public virtual Invocation.CliAction? Action { get{} set{} }
    public ICollection<string> Aliases { get{} }
    public bool AllowMultipleArgumentsPerToken { get{} set{} }
    public ArgumentArity Arity { get{} set{} }

    public List<Func<Completions.CompletionContext,
        IEnumerable<Completions.CompletionItem>>> CompletionSources { get{} }

    public bool HasDefaultValue { get{} }
    public string? HelpName { get{} set{} }
    public bool Recursive { get{} set{} }
    public bool Required { get{} set{} }
    public List<Action<Parsing.OptionResult>> Validators { get{} }
    public abstract Type ValueType { get{} }

    public override IEnumerable<Completions.CompletionItem> GetCompletions(
        Completions.CompletionContext context){}
}

public partial class CliOption<T> : CliOption
{
    public CliOption(string name, params string[] aliases){}
    public Func<Parsing.ArgumentResult, T?>? CustomParser { get{} set{} }
    public Func<Parsing.ArgumentResult, T>? DefaultValueFactory { get{} set{} }
    public override Type ValueType { get{} }
    public void AcceptLegalFileNamesOnly(){}
    public void AcceptLegalFilePathsOnly(){}
    public void AcceptOnlyFromAmong(params string[] values){}
}

public partial class CliRootCommand : CliCommand
{
    public CliRootCommand(string description = "") : base(default(string)!, default(string?)){}
    public IList<CliDirective> Directives { get{} }
    public static string ExecutableName { get{} }
    public static string ExecutablePath { get{} }
    public void Add(CliDirective directive){}
}

public abstract partial class CliSymbol
{
    internal CliSymbol(){}
    public string? Description { get{} set{} }
    public bool Hidden { get{} set{} }
    public string Name { get{} }
    public IEnumerable<CliSymbol> Parents { get{} }

    public abstract IEnumerable<Completions.CompletionItem> GetCompletions(
        Completions.CompletionContext context){}

    public override string ToString(){}
}

public static partial class CompletionSourceExtensions
{
    public static void Add(this List<Func<Completions.CompletionContext,
        IEnumerable<Completions.CompletionItem>>> completionSources,
        Func<Completions.CompletionContext, IEnumerable<string>> completionsDelegate){}

    public static void Add(this List<Func<Completions.CompletionContext,
        IEnumerable<Completions.CompletionItem>>> completionSources,
        params string[] completions){}
}
```

```
    public sealed partial class DiagramDirective : CliDirective
    {
        public DiagramDirective() : base(default(string)!){}
        public override Invocation.CliAction? Action { get{} set{} }
        public int? ParseErrorReturnValue { get{} set{} }
    }

    public sealed partial class EnvironmentVariablesDirective : CliDirective
    {
        public EnvironmentVariablesDirective() : base(default(string)!){}
        public override Invocation.CliAction? Action { get{} set{} }
    }

    public static partial class OptionValidation
    {
        public static CliOption<DirectoryInfo> AcceptExistingOnly(this CliOption<DirectoryInfo> option){}
        public static CliOption<FileInfo> AcceptExistingOnly(this CliOption<FileInfo> option){}
        public static CliOption<FileSystemInfo> AcceptExistingOnly(this CliOption<FileSystemInfo> option){}
        public static CliOption<T> AcceptExistingOnly<T>(this CliOption<T> option)
            where T : IEnumerable<FileSystemInfo>{}
    }

    public sealed partial class ParseResult
    {
        internal ParseResult(){}
        public Invocation.CliAction? Action { get{} }
        public Parsing.CommandResult CommandResult { get{} }
        public CliConfiguration Configuration { get{} }
        public IReadOnlyList<Parsing.ParseError> Errors { get{} }
        public Parsing.CommandResult RootCommandResult { get{} }
        public IReadOnlyList<Parsing.CliToken> Tokens { get{} }
        public IReadOnlyList<string> UnmatchedTokens { get{} }
        public Completions.CompletionContext GetCompletionContext(){}
        public IEnumerable<Completions.CompletionItem> GetCompletions(int? position = null){}
        public Parsing.ArgumentResult? GetResult(CliArgument argument){}
        public Parsing.CommandResult? GetResult(CliCommand command){}
        public Parsing.DirectiveResult? GetResult(CliDirective directive){}
        public Parsing.OptionResult? GetResult(CliOption option){}
        public Parsing.SymbolResult? GetResult(CliSymbol symbol){}
        public T? GetValue<T>(CliArgument<T> argument){}
        public T? GetValue<T>(CliOption<T> option){}
        public T? GetValue<T>(string name){}
        public int Invoke(){}
        public Task<int> InvokeAsync(CancellationToken cancellationToken = null){}
        public override string ToString(){}
    }

    public sealed partial class VersionOption : CliOption<bool>
    {
        public VersionOption() : base(default(string)!, default(string[])!){}
        public VersionOption(string name, params string[] aliases) :
            base(default(string)!, default(string[])!){}

        public override Invocation.CliAction? Action { get{} set{} }
    }
}

namespace System.CommandLine.Completions
{
    public partial class CompletionContext
    {
        internal CompletionContext(){}
        public static CompletionContext Empty { get{} }
        public ParseResult ParseResult { get{} }
        public string WordToComplete { get{} }
        protected static string GetWordToComplete(ParseResult parseResult, int? position = null){}
    }
    public partial class CompletionItem
    {
        public CompletionItem(string label, string kind = "Value", string? sortText = null,
            string? insertText = null, string? documentation = null, string? detail = null){}

        public string? Detail { get{} }
        public string? Documentation { get{} set{} }
        public string? InsertText { get{} }
        public string? Kind { get{} }
        public string Label { get{} }
        public string SortText { get{} }
        public bool? Equals(CompletionItem? other){}
        public override bool? Equals(object? obj){}
        public override int? GetHashCode(){}
        public override string ToString(){}
    }

    public sealed partial class SuggestDirective : CliDirective
    {
        public SuggestDirective() : base(default(string)!){}
        public override Invocation.CliAction? Action { get{} set{} }
    }

    public partial class TextCompletionContext : CompletionContext
    {
        internal TextCompletionContext(){}
        public string CommandLineText { get{} }
        public int CursorPosition { get{} }
        public TextCompletionContext AtCursorPosition(int position){}
    }
}

namespace System.CommandLine.Help
{
    public sealed partial class HelpAction : Invocation.SynchronousCliAction
    {
        public HelpBuilder Builder { get{} set{} }
        public override int Invoke(ParseResult parseResult){}
    }

    public partial class HelpBuilder
    {
        public HelpBuilder(int maxWidth = int.MaxValue){}
        public int MaxWidth { get{} }
        public void CustomizeLayout(Func<HelpContext, IEnumerable<Func<HelpContext, bool>>> getLayout){}
        public void CustomizeSymbol(CliSymbol symbol,
            Func<HelpContext, string?>? firstColumnText = null,
            Func<HelpContext, string?>? secondColumnText = null,
            Func<HelpContext, string?>? defaultValue = null){}

        public void CustomizeSymbol(CliSymbol symbol,
            string? firstColumnText = null,
            string? secondColumnText = null,
            string? defaultValue = null){}

        public TwoColumnHelpRow GetTwoColumnRow(CliSymbol symbol, HelpContext context){}
        public void Write(CliCommand command, TextWriter writer){}
        public virtual void Write(HelpContext context){}
        public void WriteColumns(IReadOnlyList<TwoColumnHelpRow> items, HelpContext context){}
        public static partial class Default
        {
            public static Func<HelpContext, bool> AdditionalArgumentsSection(){}
            public static Func<HelpContext, bool> CommandArgumentsSection(){}
            public static Func<HelpContext, bool> CommandUsageSection(){}
            public static string GetArgumentDefaultValue(CliArgument argument){}
            public static string GetArgumentDescription(CliArgument argument){}
            public static string GetArgumentUsageLabel(CliArgument argument){}
            public static string GetCommandUsageLabel(CliCommand symbol){}
            public static IEnumerable<Func<HelpContext, bool>> GetLayout(){}
            public static string GetOptionUsageLabel(CliOption symbol){}
            public static Func<HelpContext, bool> OptionsSection(){}
            public static Func<HelpContext, bool> SubcommandsSection(){}
            public static Func<HelpContext, bool> SynopsisSection(){}
        }
    }

    public partial class HelpContext
    {
        public HelpContext(HelpBuilder helpBuilder, CliCommand command, TextWriter output,
            ParseResult? parseResult = null){}

        public CliCommand Command { get{} }
        public HelpBuilder HelpBuilder { get{} }
        public TextWriter Output { get{} }
        public ParseResult ParseResult { get{} }
    }

    public sealed partial class HelpOption : CliOption<bool>
    {
        public HelpOption() : base(default(string)!, default(string[])!){}

        public HelpOption(string name, params string[] aliases) :
            base(default(string)!, default(string[])!){}

        public override Invocation.CliAction? Action { get{} set{} }
    }

    public partial class TwoColumnHelpRow
    {
        public TwoColumnHelpRow(string firstColumnText, string secondColumnText){}
        public string FirstColumnText { get{} }
        public string SecondColumnText { get{} }
        public bool? Equals(TwoColumnHelpRow? other){}
        public override bool? Equals(object? obj){}
        public override int GetHashCode(){}
    }
}

namespace System.CommandLine.Invocation
{
    public abstract partial class AsynchronousCliAction : CliAction
    {
        protected AsynchronousCliAction(){}

        public abstract Task<int> InvokeAsync(ParseResult parseResult,
            CancellationToken cancellationToken = null){}
    }

    public abstract partial class CliAction
    {
        internal CliAction(){}
        public bool Terminating { get{} protected init{} }
    }
    public sealed partial class ParseErrorAction : SynchronousCliAction
    {
        public bool ShowHelp { get{} set{} }
        public bool ShowTypoCorrections { get{} set{} }
        public override int Invoke(ParseResult parseResult){}
    }

    public abstract partial class SynchronousCliAction : CliAction
    {
        protected SynchronousCliAction(){}
        public abstract int Invoke(ParseResult parseResult){}
    }
}

namespace System.CommandLine.Parsing
{
    public sealed partial class ArgumentResult : SymbolResult
    {
        internal ArgumentResult(){}
        public CliArgument Argument { get{} }
        public override void AddError(string errorMessage){}
        public T GetValueOrDefault<T>(){}
        public void OnlyTake(int numberOfTokens){}
        public override string ToString(){}
    }

    public static partial class CliParser
    {
        public static ParseResult Parse(CliCommand command, IReadOnlyList<string> args,
            CliConfiguration? configuration = null){}

        public static ParseResult Parse(CliCommand command, string commandLine,
            CliConfiguration? configuration = null){}

        public static IEnumerable<string> SplitCommandLine(string commandLine){}
    }

    public sealed partial class CliToken
    {
        public CliToken(string? value, CliTokenType type, CliSymbol symbol){}
        public CliTokenType? Type { get{} }
        public string Value { get{} }
        public bool? Equals(CliToken? other){}
        public override bool? Equals(object? obj){}
        public override int? GetHashCode(){}
        public static bool? operator ==(CliToken? left, CliToken? right){}
        public static bool? operator !=(CliToken? left, CliToken? right){}
        public override string ToString(){}
    }

    public sealed partial class CliTokenType
    {
        internal CliTokenType(){}
        public const CliTokenType Argument = 0;
        public const CliTokenType Command = 1;
        public const CliTokenType Directive = 4;
        public const CliTokenType DoubleDash = 3;
        public const CliTokenType Option = 2;
    }

    public sealed partial class CommandResult : SymbolResult
    {
        internal CommandResult(){}
        public IEnumerable<SymbolResult> Children { get{} }
        public CliCommand Command { get{} }
        public CliToken IdentifierToken { get{} }
        public override string ToString(){}
    }
    public sealed partial class DirectiveResult : SymbolResult
    {
        internal DirectiveResult(){}
        public CliDirective Directive { get{} }
        public CliToken Token { get{} }
        public IReadOnlyList<string> Values { get{} }
    }

    public sealed partial class OptionResult : SymbolResult
    {
        internal OptionResult(){}
        public CliToken? IdentifierToken { get{} }
        public int IdentifierTokenCount { get{} }
        public bool Implicit { get{} }
        public CliOption Option { get{} }
        public T GetValueOrDefault<T>(){}
        public override string ToString(){}
    }

    public sealed partial class ParseError
    {
        internal ParseError(){}
        public string Message { get{} }
        public SymbolResult? SymbolResult { get{} }
        public override string ToString(){}
    }

    public abstract partial class SymbolResult
    {
        internal SymbolResult(){}
        public IEnumerable<ParseError> Errors { get{} }
        public SymbolResult? Parent { get{} }
        public IReadOnlyList<CliToken> Tokens { get{} }
        public virtual void AddError(string errorMessage){}
        public ArgumentResult? GetResult(CliArgument argument){}
        public CommandResult? GetResult(CliCommand command){}
        public DirectiveResult? GetResult(CliDirective directive){}
        public OptionResult? GetResult(CliOption option){}
        public T? GetValue<T>(CliArgument<T> argument){}
        public T? GetValue<T>(CliOption<T> option){}
    }

    public sealed partial class TryReplaceToken
    {
        public TryReplaceToken(object @object, IntPtr method){}

        public virtual System.IAsyncResult BeginInvoke(string tokenToReplace,
            scoped out IReadOnlyList<string>? replacementTokens, scoped out string? errorMessage,
            System.AsyncCallback callback, object @object){}

        public virtual bool EndInvoke(scoped out IReadOnlyList<string>? replacementTokens,
            scoped out string? errorMessage, System.IAsyncResult result){}

        public virtual bool Invoke(string tokenToReplace, scoped out IReadOnlyList<string>? replacementTokens,
            scoped out string? errorMessage){}
    }
}
```