System Test Plan
Input Space Partitioning

By: Abdulla Sakallah
Mohammed Zakiuddin

Table of Contents

1. System Under Test

Command line argument parser (CLI) is a free software project from Apache Commons Library. This library gives the user the ability to do things like add a new command under an option; it also handles parsing and storing the arguments. Depending on the arguments selected by the user, it also automatically modifies how the output should be presented to them.

1.1. Details

The project source code: https://commons.apache.org/proper/commons-cli/index.html

Number of the components for CLI

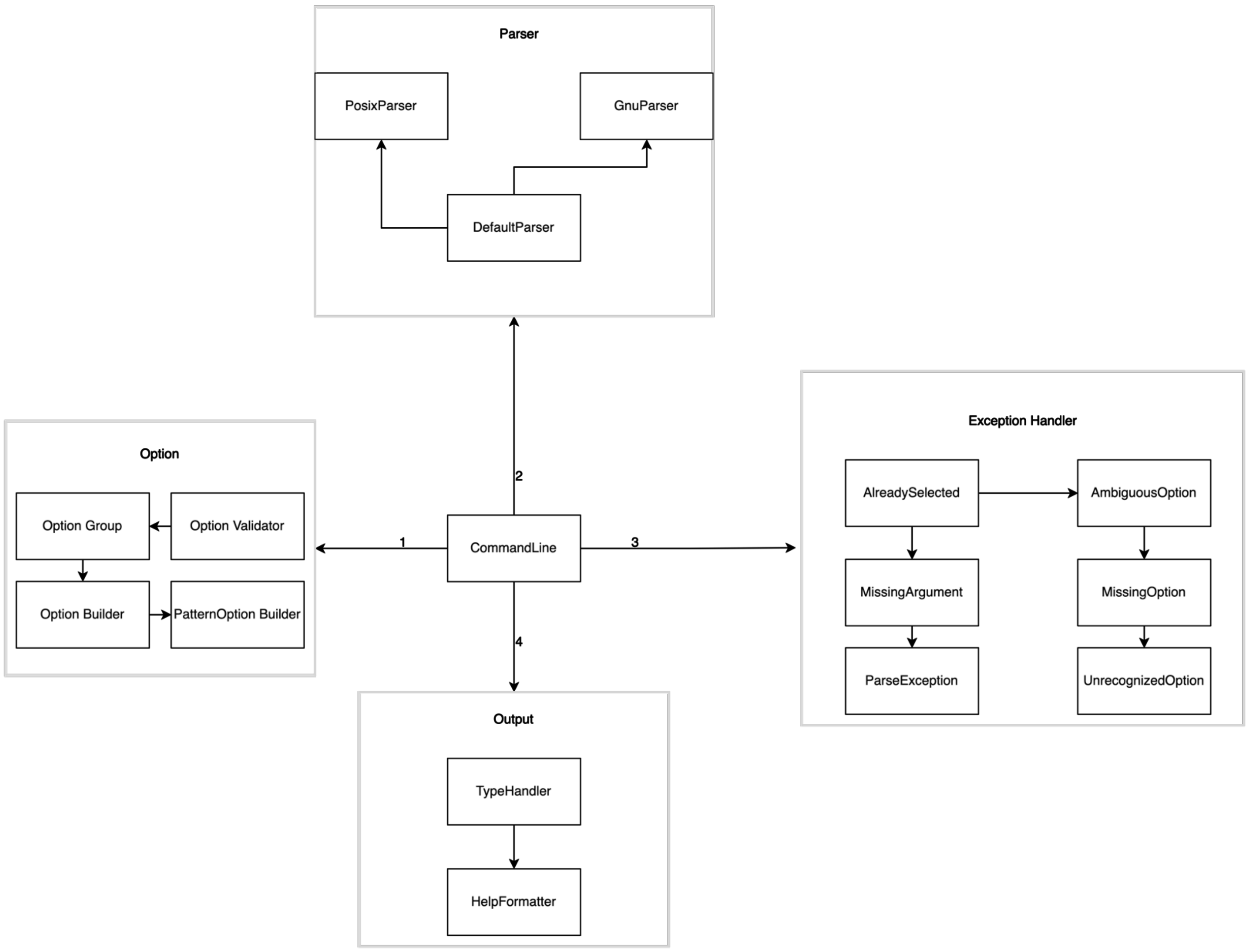
- Command Line Arguments Parser (CLI) :
  - 1 package
  - 21 class files & 1 interface

Lines of Code for each Component

- Util: 67 LOC
- UnrecognizedOptionException: 64 LOC
- TypeHandler: 209 LOC
- PosixParser: 244 LOC
- PatternOptionBuilder: 194 LOC
- Parser: 359 LOC
- ParseException: 38 LOC
- OptionValidator: 84 LOC
- Options: 337 LOC
- OptionGroup: 156 LOC
- OptionBuilder: 358 LOC
- Option: 898 LOC
- MissingOptionException: 86 LOC
- MissingArguementException: 62 LOC
- HelpFormatter: 933 LOC
- GnuParser: 90 LOC
- DefaultParser: 777 LOC
- CommandLineParser (Interface): 86 LOC
- CommandLine: 467 LOC
- BasicParser: 50 LOC
- AmbiguousOptionException: 83 LOC
- AlreadySelectedException: 83 LOC

Total LOC: 5,725

1.2. Software Architecture



2. Test Environment

The project under test is deployed under IntelliJ IDEA CE Version 2022.1.1. It uses JUnit 3, because we have faced dependency issues with JUnit 5. The Operating System that is used to run the tests is Windows 11, and the version of Java that is being used is 17.0.2.

3. Input Space Partitioning

3.1. The Input Domain

- Objects representing current state (class or integration testing)
- User level inputs (system testing)

Parameters to a method (unit testing)

- Command Line - getArgs()
- Default Parser – parse (Options options, String[] arguments, Boolean stopAtNonOption)
- Pattern Option Builder – ParsePattern(String pattern)
- Command Line – Option[] getOptions()
- Command Line – addOption(final Option opt)
- HelpFormatter – printHelp(final PrintWriter pw, final int width, final String cmdLineSyntax, final String header, final Options options, final int leftPad,final int descPad, final String footer)

Global Variables (unit testing)

- No global variables were found.

Data read from a file

- We have no data that is being read from a file.

3.2. Characteristics and Blocks

Command Line –
Getargs()

C1 – Functionality based (This characteristic was selected to check if the argument that is being tested is valid or not).
C2 & C3 – Interface based (Number of arguments is selected to check how many arguments are being passed) & (Length of the arguments was selected to check the length of the arguments that are being passed).

It satisfies the disjointness and completeness.

	Characteristic	B1	B2	B3
C1	If it is a valid argument	T	F	
C2	Number of arguments	0	1	>1
C3	Length of the arguments	0	1	>1

Example:
Number of arguments – cd, ls, touch (In number they are 3).

Length of the arguments – cd (2), ls (2), touch (5)
If all the test cases are valid then according to the pair wise coverage (PWS) – 3 * 3 = 9 valid test cases.

Default Parser –
Parse(Options options, String[] arguments, Boolean stopAtNonOption)

C1 – Interface based (This characteristic was selected to check if the string is valid to parse).
C2 – Functionality based (This characteristic was selected to check the number of arguments present inside a string).
C3 – Interface based (This characteristic was selected to check the arguments that can be parsed using the Non-Stop option).

It satisfies the disjointness and completeness.

	C	B1	B2	B3
C1	If the option to parse is a valid string	T	F	
C2	Arguments that can be present inside a string	0	1	>1
C3	Number of arguments that can be parsed using the Nonstop option	0	1	>1

If all the test cases are valid then according to the pair wise coverage (PWS) – 3 * 3 = 9 valid test cases.

Pattern Option Builder –
ParsePattern(String pattern)

C1 – Interface based (This characteristic was selected to check if the pattern was valid).
C2 –Interface based (This characteristic was selected to check if the pattern is parsed).
C3 – Functionality based (This characteristic was selected to check the length of the parsed pattern).

It satisfies the disjointness and completeness.

	C	B1	B2	B3
C1	If the pattern is valid	T	F	
C2	If the pattern is parsed	T	F	
C3	Length of the parsed pattern	0	1	>1

If all the test cases are valid then according to the pair wise coverage (PWS) – 3 * 2 = 6 valid test cases.

Command Line–
public getOptions()

C1 – Functionality based (This characteristic was selected to check if the list of options is empty).
C2 –Interface based (This characteristic was selected to check the number of options).
C3 – Interface based (This characteristic was selected to check the options that have a null descriptions).

It satisfies the disjointness and completeness.

	C	B1	B2	B3
C1	The list of options is empty	T	F	
C2	The number of options	0	1	>1
C3	Options will have null descriptions	T	F	

If all the test cases are valid then according to the pair wise coverage (PWS) – 3 * 2 = 6 valid test cases.

Command Line–
Protected void addOption(final Option opt)

C1 – Interface based (This characteristic was selected to check if the option is null or not).
C2 –Interface based (This characteristic was selected to check if the long option equals the short abbreviation for the short options).
C3 – Functionality based (This characteristic was selected to check if the resulting list of options).

It satisfies the disjointness and completeness.

	C	B1	B2	B3
C1	The option is null	T	F	
C2	Long option equals the short option.	T	F	
C3	Resulting list option length	0	1	>1

If all the test cases are valid then according to the pair wise coverage (PWS) – 3 * 2 = 6 valid test cases.

HelpFormatter–
public void printHelp(final PrintWriter pw, final int width, final String cmdLineSyntax, final String header, final Options options, final int leftPad,final int descPad, final String footer)

C1 – Interface based (This characteristic was selected if the cmdLineSyntax is empty or not).
C2 –Functionality based (This characteristic was selected to check if the number of characters of padding to be prefixed to each line can have).
C3 – Interface based (This characteristic was selected to check whether the left pad is greater than the desc pad).

C1, C2 & C3 are disjoint, and they don't satisfy the completeness.

	C	B1	B2	B3
C1	The cmdLineSyntax is empty	T	F	
C2	The number of characters of padding to be prefixed to each line can have	0	1	>1
C3	The left pad is greater than the desc pad	T	F	

If all the test cases are valid then according to the pair wise coverage (PWS) – 3 * 2 = 6 valid test cases.

4. Test Requirements

4.1. Pair-Wise Coverage Criteria

Command Line –
GetArgs() -

TR	A	B	C
1	A1	B1	C1
2	A1	B2	C2
3	A1	B3	C3
4	A2	B1	C2
5	A2	B2	C3
6	A2	B3	C1
7	A2	B1	C3
8	A2	B2	C2
9	A1	B3	C1

TR	A	B	C	Feasible
1	T	0	0	Yes
2	T	1	1	Yes
3	T	2	2	Yes
4	F	0	1	Yes
5	F	1	2	Yes
6	F	2	0	Yes
7	F	0	2	Yes
8	F	1	1	Yes
9	T	2	0	Yes

Default Parser –
Parse() -

TR	A	B	C
1	A1	B1	C1
2	A1	B2	C2
3	A1	B3	C3
4	A2	B1	C2
5	A2	B2	C3
6	A2	B3	C1
7	A2	B1	C3
8	A2	B2	C2
9	A1	B3	C1

TR	A	B	C	Feasible
1	T	0	0	Yes
2	T	1	1	Yes
3	T	2	2	Yes
4	F	0	1	Yes
5	F	1	2	Yes
6	F	2	0	Yes
7	F	0	2	Yes
8	F	1	1	Yes
9	T	2	0	Yes

Pattern Option Builder –
ParsePattern(String pattern) –

TR	C	A	B
1	C1	A1	B1
2	C1	A2	B2
3	C2	A1	B1
4	C2	A2	B2
5	C3	A1	B2
6	C3	A2	B1

TR	C	A	B	Feasible
1	0	T	T	Yes
2	0	F	F	Yes
3	1	T	T	Yes
4	1	F	F	Yes
5	2	T	F	Yes
6	2	F	T	Yes

Command Line -- getOptions()

A = {1,2,3}
B = {A.B}
C = {X,Y}

TR	A	B	C
1	1	A	X
2	1	B	Y
3	2	A	X
4	2	B	Y
5	3	A	Y
6	3	B	X

TR	A	B	C	Feasible
1	0	T	T	Yes
2	0	F	F	Yes
3	1	T	T	No, because the number of options is 1 while the list is said to be empty.
4	1	F	F	Yes
5	2	T	F	No, because the number of options can't be 2 when the list of options is empty.
6	2	F	T	Yes

HelpFormatter -- public void printHelp(final PrintWriter pw, final int width, final String cmdLineSyntax, final String header, final Options options, final int leftPad,final int descPad, final String footer)

A = {1,2,3}
B = {A.B}

C = {X,Y}

TR	A	B	C
1	1	A	X
2	1	B	Y
3	2	A	X
4	2	B	Y
5	3	A	Y
6	3	B	X

TR	A	B	C	Feasible
1	0	T	T	Yes
2	0	F	F	Yes
3	1	T	T	Yes
4	1	F	F	Yes
5	2	T	F	Yes
6	2	F	T	Yes

Command Line -- public addOption(Option opt)

A = {1,2,3}
B = {A.B}
C = {X,Y}

TR	A	B	C
1	1	A	X
2	1	B	Y
3	2	A	X
4	2	B	Y
5	3	A	Y
6	3	B	X

TR	A	B	C	Feasible
1	2	T	T	Yes
2	2	F	F	Yes
3	3	T	T	Yes
4	3	F	F	Yes
5	4	T	F	Yes
6	4	F	T	Yes

5. Test Results with Traceability

getArgs()

Test ID	Targeted TR	MUT	Input	Observed Output	Result
1	1	getArg	(T,0,0)	N/A	Pass
2	2	getArg	(T,1,1)	N/A	Pass
3	3	getArg	(T,2,2)	N/A	Pass
4	4	getArg	(F,0,1)	N/A	Pass
5	5	getArg	(F,1,2)	N/A	Pass
6	6	getArg	(F,2,0)	N/A	Pass
7	7	getArg	(F,0,2)	N/A	Pass
8	8	getArg	(F,1,1)	N/A	Pass
9	9	getArg	(T,2,0)	N/A	Pass

Parse()

Test ID	Targeted TR	MUT	Input	Observed Output	Result
1	1	Parse	(T,0,0)	N/A	Pass
2	2	Parse	(T,1,1)	N/A	Pass
3	3	Parse	(T,2,2)	N/A	Pass
4	4	Parse	(F,0,1)	N/A	Pass
5	5	Parse	(F,1,2)	N/A	Pass
6	6	Parse	(F,2,0)	N/A	Pass
7	7	Parse	(F,0,2)	N/A	Pass
8	8	Parse	(F,1,1)	N/A	Pass
9	9	Parse	(T,2,0)	N/A	Pass

addoption()

Test ID	Targeted TR	MUT	Input	Observed Output	Result
1	1	addOption	(2,T,T)	N/A	Pass
2	2	addOption	(2,F,F)	N/A	Pass
3	3	addOption	(3,T,T)	N/A	Pass
4	4	addOption	(3,F,F)	N/A	Pass
5	5	addOption	(4,T,F)	N/A	Pass
6	6	addOption	(4,F,T)	N/A	Pass

getOptions()

Test ID	Targeted TR	MUT	Input	Observed Output	Result
1	1	getOptions	(0,T,T)	N/A	Pass
2	2	getOptions	(0,F,F)	N/A	Pass
3	3	getOptions	(1,T,T)	N/A	Fail
4	4	getOptions	(1,F,F)	N/A	Pass
5	5	getOptions	(2,T,F)	N/A	Fail
6	6	getOptions	(2,F,T)	N/A	Pass

public void printHelp(final PrintWriter pw, final int width, final String cmdLineSyntax, final String header, final Options options, final int leftPad,final int descPad, final String footer)

Test ID	Targeted TR	MUT	Input	Observed Output	Result
1	1	printHelp	(0,T,T)	N/A	Pass
2	2	printHelp	(0,F,F)	N/A	Pass
3	3	printHelp	(1,T,T)	N/A	Pass
4	4	printHelp	(1,F,F)	N/A	Pass
5	5	printHelp	(2,T,F)	N/A	Pass
6	6	printHelp	(2,F,T)	N/A	Pass

ParsePattern(String pattern)

Test ID	Targeted TR	MUT	Input	Observed Output	Result
1	1	ParsePattern	(0,T,T)	N/A	Pass
2	2	ParsePattern	(0,F,F)	N/A	Pass
3	3	ParsePattern	(1,T,T)	N/A	Pass
4	4	ParsePattern	(1,F,F)	N/A	Pass
5	5	ParsePattern	(2,T,F)	N/A	Pass
6	6	ParsePattern	(2,F,T)	N/A	Pass