SOFTWARE TESTING COURSEWORK SPECIFICATION 2020/21

February 25, 2021

To be read in conjunction with the Description of the coursework.

1 Specifications for tasks 1 and 2

1.1 Overview

The command line option parser is developed to process command line options stored in an input string and output a set of value pairs containing information retrieved from the input string.
The user can first define a set of options:

- Option name `filename`, with a String value type.
- Option name `output`, with a shortcut `o` and a String value type.

After the options are defined, the parser is able to process the following example input of command line options:

```
--filename 1.txt -o 2.txt
```

In this example, the prefix `--` indicates using the name of the option while `-` indicates using the shortcut of the option.
The parser will get the following value pairs:

```
filename = "1.txt"
output = "2.txt"
```

1.2 Parser Initialisation

The parser is initialised using

```
Parser parser = new Parser();
```

There is no further specification regarding class initialisation.

1.3 Add options with a shortcut

```
void add(String option_name, String shortcut, int value_type)
```

This method is for adding a new option that has a shortcut.

Example:

```
parser.add( "output" , "o" , Parser.STRING);
parser.add( "optimise" , "O" , Parser.BOOLEAN);
```

Parameter list:

`String` *option_name* : name of the option in the command line to receive user-specified value.
`String` *short_cut* : a short cut of this option
`int` *value_type* : type of the expected value. Choices of the parameter include:

- `Parser.INTEGER`
- `Parser.BOOLEAN`
- `Parser.STRING`
- `Parser.CHAR`

Return value:

This method does not have a return value.

Specifications:

1. Adding an option with the same name as an existing option will override the option defined previously.
2. Name and shortcut of options should only contain digits, letters and underscores. Digits cannot appear as the first character. A runtime exception is thrown otherwise.
3. Option names and shortcuts are case-sensitive.
4. An option can have a shortcut that is the same as the name of another option. For example, the user can define an option whose name is `output` with a shortcut `o` and another option whose name is `o`. When assigning values to these options, `--output` and `-o` is used for the first option and `--o` is used for the second option.
5. Boolean options are assigned to true using any value except 0 and false (case insensitive), or without having a value. For example, the option `optimise` is set to true by `-O`, `--optimise`, `-O=true`, `-O=1` or `-O 100` while it is set to false by `-O=false`, `--optimise=0` or not using this option in the entire string.

1.4 Add options without a shortcut

```
void add(String option_name, int value_type)
```

This method is for adding a new option that does not have a shortcut.

Example:

```
parser.add( "output" , Parser.STRING);
parser.add( "optimise" , Parser.BOOLEAN);
```

Parameter list:

`String` *option_name* : name of the option in the command line to receive user-specified value.
`int` *value_type* : type of the expected value. Choices of the parameter include:

- `Parser.INTEGER`
- `Parser.BOOLEAN`
- `Parser.STRING`
- `Parser.CHAR`

Return value:

This method does not have a return value.

Specifications:

Same as specifications of the first add function.

1.5 Parse command line options

```
int parse(String command_line_options)
```

Parse the string containing command line options.

Example:

```
parser.parse( "--input␣1.txt␣--output=2.txt" );
parser.parse( "-O" );
```

Parameter list:

`String` *command_line_options* : command line option string.

Return value:

This method returns 0 if the parsing succeeds and other values if the input is not valid or the parser fails.

Specifications:

1. The prefix `--` is used for the full name of the option.
2. The prefix `-` is used for the shortcut version of the option.
3. Assigning a value to the option can either be using a `=` or a space. That is, `option=value` and `option value` are both valid.
4. The user can use quotation marks, single or double, optionally around the value. `option="value"` , `option='value'` and `option=value` are all valid and result in the same effect.
5. As quotation marks can be used to decorate values, they can still be part of the value if the user wants. If single quotation marks are used, double quotation marks in the value string are viewed as part of the string and vice versa. For example, `option='value="abc" '` is valid and the value of the option is `value="abc"`.
6. If the user assigns values to the option multiple times, the value taken by the option is from the last assignment.
7. The user does not need to provide values for every option. For the options that are not assigned a value using this function, a default value (described in Section 1.6) is stored.
8. This method can be used multiple times as shown in the example above. After these two function calls, three options are assigned to values and can be retrieved by the following methods.

1.6 Retrieve information

```
int getInteger(String option)
boolean getBoolean(String option)
String getString(String option)
char getChar(String option)
```

Retrieve the value of an option.

Example:

```
String output_file_name = parser.getString( "output" );
boolean optimise = parser.getBoolean( "O" );
```

Parameter list:

`String` *option* : the full name or shortcut of an option.

Return value:

These methods return the value of the corresponding option.

Specifications:

1. The order of search is full name of options first and then shortcut. For example, if `o` exists as a full name for an option and a shortcut for another option, this function returns the value of the first option.

2. If the option is not defined or not provided a value, a default value is used: 0 for `INTEGER`, `false` for `BOOLEAN`, an empty String `""` for `STRING` and `'\0'` for CHAR

2 Specifications for task 3

An option may also take a list of characters as its value. In order to retrieve value types that are character lists, please implement the following function as a method of the Parser class:

```
List<Character> getCharacterList(String option);
```

Example:

```
Parser parser = new Parser();
parser.add("list","l",Parser.STRING);
parser.parser("--list=e-ck.txt");
List l = parser.getCharacterList("list");
```

In this example, the returned list contains list values {'e','d','c', 'k', '.', 't', 'x', 't'}.

Parameter list:

`String` *option* : the full name or shortcut of an option.

Return value:

This method returns the list of all character values retrieved from the value of the option. It returns an empty list if the input is not valid or the parser fails.

Specifications:

1. The order of search is full name of options first and then shortcut. For example, if `o` exists as a full name for an option and a shortcut for another option, this function returns the value of the first option.

2. If the option is not provided a value, an empty list is returned.

3. The character list value may only contain letters, digits and full stops(.).

   ```
   parser.parse("--option=test123.txt");
   parser.getCharList("option");
   ```

   will return {'t', 'e', 's', 't', '1', '2', '3', '.', 't', 'x', 't'};

4. The value of the option is not case sensitive. For example, "Test123.txt" and "test123.txt" both will return {'t', 'e', 's', 't', '1', '2', '3', '.', 't', 'x', 't'};

5. If a value starts with a hyphen (-), the function will return an empty list. If a hyphen occurs elsewhere in the value, the function will remove the hyphen. For example,

```
parser.parse("--option=-test123.txt");
parser.getCharList("option"); // This returns an empty list


parser.parse("--option=test123-.txt");
// This returns  {'t','e','s','t','1','2','3','.','t','x','t'};
parser.getCharList("option");
```

6. A hyphen (-) indicates an inclusive range of char. For example, `"a-c"` represents a,b,c and `"c-a"` represents c,b,a.