SUMMARY

I. Create the bundle 2

II. Create a program 2

A. Create an option 2

B. Use file options 3

C. Reuse common options 3

D. Program mechanics 3

III. Create the API 4

IV. Use the API 5

A. Parse commands 5

B. Window mode 5

C. XML files 6

V. Print data (do not use System.out) 6

VI. Analyze a program execution time 7

**CLIGUI DOCUMENTATION**

Cligui is a library used to run different programs using a console or a window.

# Create the bundle

Before you start coding, you should focus on the bundle file which will contain programs names and options names, an example can be found in "src/files/tests.properties".

Each line of a bundle file must contain a key and a text, you can use a tabulation to split them.

You can also add comments using the # letter a the start of a line.

When you have finished filling the bundle file, ensure you have placed it in a source directory (only internal bundles will be read by cligui).

# Create a program

You can create your own program by specializing the AbstractProgram class, an example can be found in "tests/data/BillProgram.java".

## Create an option

Firstly, you must create a class variable for each option you want in your program but you can only use primitive types and not null String types.

Then, you add the Parameter annotation above these variables with their respective option name (see keys in the bundle file you created earlier).

The Parameter annotation contains three other parameters :

> required : if used, the program option will be mandatory

> hidden : if used, the program option will be hidden from the program options list (note that the option will remain usable)

> enumeration : this parameter can be used on an integer option to point towards an enumeration class, this class will then be converted into a combo box in the window mode

## Use file options

In case you are using files to read or write data, you must define a String option and use one of the following annotations :

> InputFile / OutputFile : the file is an input file or an output file, the file extension is required in the window to target the desired files when browsing directories

> InternalFile : the file should contain a list and must be located in a source directory, its data will be converted into an enumeration in the window mode

## Reuse common options

Finally, there is another annotation called Delegate which can be used to reuse common options.

Sometimes, you may have to use the same options on different programs, which would be a redundancy problem.

With the Delegate annotation, you can create a class (not an AbstractProgram) with options and make the programs point towards this class to add the common options (see "tests/data/Product.java").

In other words, we use the Product options in BillProgram but we could also create another program pointing to the same Product options.

## Program mechanics

Now that you have defined all your program options, you can use them in the execute() method you had to implement when you specialized the AbstractProgram class.

When you will run your program, the execute() method will be called.

# Create the API

From this step, you should have a bundle file and a program ready to launch.

If so, your next goal is to create an API :

CLI\_api api = new CLI\_api("cligui.jar", "files/test.properties", "prog");

Some explanations about the API constructor :

1. The project name which will be used in GUI and XML files, it should be named after the JAR file you create when you export your project (do not forget the ".jar" extension")
2. The internal URL of the bundle file, which will mainly contain all options and programs descriptions, only bundles located in a source directory will be read and merged with cligui bundle
3. The name of the option used to choose a program, you can simply use "prog" as for MACSE

When you are done creating the API, you can add your program with the addProgram() method :

api.addProgram("bill", BillProgram.class);

The first parameter is the program name you will have to type in the commands line and the second parameter is the program class.

You now have an API containing your custom program.

# Use the API

The cligui logger is used to direct outputs in the console / windows and its level can be changed by doing the following :

CLI\_logger.getLogger().setLevel(...);

All messages with priorities higher or equals to the level you defined will be printed in the console.

You can disable logging using Level.OFF.

Note that the window mode only uses the INFO level and cannot be changed.

## Parse commands

The parse() method use a String or String array object (as you wish) and parse its elements to "execute()" the program.

For instance, you can type the following :

api.parse("-prog bill -id 4 -name productA01 -price 1.25");

Cligui will look for a program named "bill" and set the values you chose to the options :

id => 4

name => "productA01"

price => 1.25

Note that if you did something wrong, cligui will output an error and print the programs list or your program options list.

## Window mode

If you are done using the console mode, you can now test the window interface, a runnable class can be found at "tests/gui/WindowRunner.java".

As you see, you only need to create a new instance of CLI\_gui with an API to open the window mode :

new CLI\_window(api);

## XML files

The cligui API also allows you to create XML files compatible with Galaxy :

api.exportXML("projectVersion", "outputDirectory");

You just have to define your project version (which will be stored in the XML file) and the output directory which will contain the XML file.

Moreover, you can also add a filter to refrain some output files from being created (see SplitAlignment program in MACSE) by overriding the getXMLfilter() method.

# Print data (do not use System.out)

Cligui uses a logger to print data, so it is highly recommended you do not use System.out.

Instead, data printing can be achieved like this :

CLI\_logger.getLogger().info("text");

The CLI\_logger.getLogger() method will return a java Logger (see JRE documentation if needed).

As for exceptions, the following line is more convenient :

CLI\_logger.logError(Level.SEVERE, error);

In case you need to display some dynamic data, you can always use the bundle with the @ letter :

KEY MESSAGE

unexpectedError The file @ could not be read.

However, you will need to call CLI\_bundle.getPropertyDescription(KEY, DYNAMIC\_DATA).

Example :

CLI\_bundle.getPropertyDescription("unexpectedError", "sequences.fasta");

Result : "The file sequences.fasta could not be read."

You can add as much dynamic data as you wish, if you only define the key, then no @ letter will be updated.

# Analyze a program execution time

You can run a program and compute its execution time just by adding the "timeTest" option in the commands line.

The execution time will then be stored in a file "time.txt", you can even run several programs with a bash command, each time result will be appended to the others in the file.