CSE 410 - Human Computer Interaction
The command line (project 1)

Overview
The command line is still a vital part of modern computing, used for scripting and automation, and powerful utility programs. What it provides in power, however, it often sacrifices usability by requiring that the users remember complex and arcane sequences of commands and options. There are features that can be introduced to help reduce the memory burden while retaining the power of the command line, and this lab will explore three of those features; help, completion, and history. We will do that in the context of a Java program that will generate blocks of "nonsense" text, which is useful when mocking up graphic design layouts - something you'll have cause to do later in the course!

Resources and Lab Setup
ANSI Terminal Support
If you are using a Windows environment to develop in, you will need to have an ANSI compliant terminal in order to use the tab completion libraries. The standard command prompt and powershell are not compliant, and don't support the necessary I/O for command history and tab completion. The best method is to download and install Cygwin, which provides a linux like shell that will interact with your file system.
If you are using a Mac or Linux OS, you should be good to go without further technology, as both support ANSI standard terminals.

Tab Completion Library
In order to facilitate the focus on designing the interface, you don't have to write the code that produces tab completion or command history - there is a good java library available to do so. It can be either compiled from source or (more simply) downloaded from my website. Version 3.5 of the jar file is what this activity has been tested with.

Nonsense Languages
Rather than actually "generating" text, you can use the approach of writing out predefined blocks of random text. At least two different types of random text are required for the lab; while you are welcome to find additional sources, Lorem Ipsum and Anguish Languish are two that are relatively easy to get samples of.

Java Version
This lab should be completed using Java 1.8

Requirements
The program that you write should operate in two different modalities - as a command line tool, which produces output that can be sent to other tools, and as an interactive session, where multiple commands can be run.

Command line mode
When run in command line mode (specified by the `--generate` option), your program should run one time, and produce the appropriate text. It should accept the following options to specify that output. It should accept any valid combination of command options - so, for example, a user should be able to specify

```
generator --generate --library lorem --count 5 --outfile out.txt
```

And have all of the included options apply. The syntax of the command should follow POSIX standards.

Option	Expected Behavior	Error Handling
`--help, -h`	Produce a help file, listing the various options and the way in which the program can be used. This may be the same as the man page format, or you may tweak it as appropriate	Help is only valid as a standalone option. If combined with other options, a helpful guidance message to that effect should be produced
`--version,-v`	This should display the version number of your program	Version is only valid as a standalone option. If combined with other options, a helpful guidance message to that effect should be produced
`--library, -l <library>`	This should specify the source of the nonsense language being generated. At least two sources are required, but you can support as many as you like	If the library option is not followed by a valid library name, then a guidance message listing the valid library names should be produced
`--outfile, -o <filename>`	This should specify an output file to be written to. If this parameter is not present, output is to system out.	If no file is specified, or if any file I/O issues are encountered, a guidance message listing the issue encountered and the way to fix it should be produced. If the file exists already, treat this as an issue.
`--mode, -m <mode>`	This should specify a mode of operation. Your program should produce either paragraphs, words, or bullet points. If no mode is specified, paragraph should be the default mode.	If an empty or incorrect mode is specified, a guidance message listing the valid modes and their meanings should be produced
`--count, -c <count>`	The number of words/paragraphs/bullets to be	If the count argument is either empty or invalid, a guidance

	produced. If this option is not used, 1 should be the default count for paragraphs, 3 for words, and 5 for bullets.	message to that effect should be produced. If your program limits the number of elements that can be produced (probably a good idea) then those limits should also be checked.
`--html, -h`	If this option is included, output should have HTML markup added in the form of <p> for paragraphs, <ul> and <li> for bullets, and <h1> for words.	
`--generate, -g`	Specifies command line mode. If this option is not included, the program should start in interactive mode.	

Additional Error Handling: if any incorrect parameters are included, a guidance message to that effect should be produced. If any error conditions are discovered in command line mode, the program should not generate text, it should only produce the requisite guidance message.

Interactive Mode
When run in interactive mode, your program should support the following command set

Command	Behavior
`generate <options>`	Generate text, as per the command line. All command line options should be allowed here as well, with the same expected behaviors, with the exception of defaults which may be changed for the session
`help`	As in the command line
`version`	As in the command line
`set <option> <value>`	Allow the user to specify a default for mode, count, library, html, and outfile. Those defaults should be honored when the generate command is run, unless overridden by an option in the generate command itself
`show <option>`	Display the current value that has been set. If no argument is given, show all values. If the value specified has not been set, give guidance to that effect.

`exit or quit`	Exits the program

Feedback should be produced for each command.

Additional Requirements
Along with the program, you should also submit a man file. Instructions on how to create and test man files. Additional help on man files.

Submission

An executable .jar file and a gzip file with your man page should be submitted via ublearns.

Scoring

Scoring for the lab will be as follows

- (9 Points) working application from jar file. If the program crashes at any point, a maximum of 3 points will be awarded. If the program produces unhandled exceptions, a maximum of 6 points will be awarded.
- (3 points) all commands reflected in help
- (3 points) all commands reflected in man page
- (3 points) all error handling works as specified
- (3 points) tab completion works for all valid command sequences
- (3 points) file option saves a file
- (3 points) HTML option creates well formatted HTML
- (3 points) multiple generation sets are supported
- (3 points) version information is delivered
- (3 points) Paragraph mode works as required
- (3 points) Word mode works as required
- (3 points) List mode works as required
- (3 points) Command line and interactive mode work as specified
- (3 points) man page included and loads correctly
- (3 points) Syntax conventions are followed