This is copied from harry’s doc in the repo:

## Editor.java

* Moved four separate filter fields into an array list ‘filters’. This greatly improves extensibility and maintainability by making the list of filters enumerable, enabling the use of loops to perform operations. Also means that new filters can just be appended onto the ArrayList, instead of manually hard-coded as new fields.
* Compressed staircase of if statements into a for loop, operating on the ArrayList of filters.
* processCommand() changed to use reflection instead of many if/elseif to determine which command should be used. Commands now called using the Invoke() method. Increased extensibility as no longer need to add additional ‘elseif’ blocks.
* Refactored filter operations into a new ‘Image’ class, increasing cohesion.
* Differentiated between ‘Editor’ commands and ‘filter’ commands in order to denote which parameters are passed to method.invoke().
* Unnecessary public fields changed to be private to improve encapsulation and remove implicit coupling.
* Created new boolean ‘finished’ field to reduce code duplication from returning false from all commands except ‘quit’. Now only have to return true on the ‘quit’ command instead.
* Added ‘Undo’ command to replace the current working image from the most recent in that Image’s stack. Image snapshots pushed to the stack whenever a filter is applied.
* Added a HashMap field ‘ImageCache’ to store enable multiple images to be stored in memory and worked on concurrently. Added ‘put’ and ‘get’ commands to enable interaction with this memory space.
* Hard-coded printlns moved refactored to use internationalisation.

Image.java

* Created new Image class
* Added ArrayList field to store all currently applied filters. Increased cohesion and extendability.
* Moved Image names into Image class to increase cohesion.

Parser.java

• Moved individually-declared string fields (eg. word1, word2) into an ArrayList ‘words’. This makes the application more extendable - if the stakeholders wanted to add additional words for each command, no additional fields will need to be created.

Refactored hasNext checks into a while loop which runs while more words remain. Increases extendability, as number of possible words is no longer hard coded.

## Command.java

* Moved command words from individually-declared strings (e.g. secondWord, thirdWord) to an ArrayList of strings ‘words’. Command can now be accessed using their index within the array. This greatly increases maintainability and extensibility as more command words can be added if desired – without the need to add additional fields, getters or null checks.
* Refactored each command to use the ‘Command’ design pattern. Each command now its own containing class, which inherits from a superclass, Command.java. Commands then executed with a common execute() which accepts an editor instance as a param. Greatly increases extensibility as you can just add a new command subclass containing the logic and add it to a hashmap. Increases cohesion as all command logic is encapsulated in its own class, and reduces coupling between invoker (Editor) and class containing logic.
* Condensed the duplicated getters (e.g. getSecondWord, getThirdWord) and null checking methods into one function each, both accepting array an index parameter.
* Boolean checks (eg. hasSecondWord) moved to one method ‘hasWord’, using an index as a parameter. Improves extendability of the application if more possible parameters were to be added to a command.
* List of possible commands now saved as static final strings instead of using strings. Increases extendability as any new commands implemented just need to have their names added to this list.

Internationalisation

**How do they reduce coupling and increase cohesion?**

* The Editor class has the responsibility placed upon itself encapsulated within the method returnLanguageHashMap() obtain a hashmap of key-words it needs to perform its functionality. In this case, it accesses the I18N module to a obtain such a hashmap.
* The concept behind reducing coupling in this case is that in the unfortunate circumstance where the I18N module breaks, or suppose we need to swap out using the I18N class within the Editor class and use another means to obtain a key-words mapping, it can still be easily achieved by following the methodology employed inside the Editor class.

**How do they increase extensibility?**

* Words for different languages are segregated into different properties file, (E.g. there is one language file for the Arabic language, one for the Japanese language as well as one for the default english language.)

**How do they support the plans of your company?**

* Allows the company to extend its program to incorporate more modules by following the same approaches in the Editor class to reduce coupling, and reduce the impact of breaking changes that may be introduced in the I18N class.
* Also allows the company to extend its user base as the I18N module is capable of extending the number of languages it is capable of using in a simple manner.

Summary of General Additions

* A class called I18N has been introduced into the program to ensure that different modules of the program can get access to the localized words that it needs. The I18N class has been structured in such a way that its functionality is as self contained as possible. All features ranging from changing of languages to be used, words to access as well as checking what it the current language settings can all be done via the public static methods of the I18N class.
* Added cache to store the list of filters currently applied to an image (as a field of Image)
* Added support for an Undo command, to revert the image back to its previous state. Added a new Stack field to Image to contain these changes. Push and Pop used to add/revert from stack.

# Software Engineering Practices

* JavaDocs used to document all code written, making it more maintainable and extendable by other developers who are unfamiliar with the codebase.
* Static analysis report produced to show issues with current system - appendix 1
* Linter used to ensure coding standards are adhered to.
* Readme created within source control
* Version control - Github was used as source control. Using source control allowed us to contribute to the same final product. This prevented us from having to distribute many different copies - we all had the same source of truth. The additional features git provided us enabled us to create multiple branches, review each other’s work (via pull requests., and easily see changes between commits.
* **Bugs limitations in system?**

Breaking changes in the I18N module may break the program, as modules such as the Editor class depends on it to initialize its hashmap of key-word mappings. A possible solution is to have a local store of default english words in the form of a map of some sorts(e.g. Hashmap in this case) so that individual modules can default to this on error in the I18N module as well as send a error status report to the developer.