CISC 322 Fall 2017
Group Project: Editor Framework

Version 2.0 2017-09-09 12:45 PM

DRAFT – will be revised in response to questions and/or comments

Table of Contents

1 Overview

Maintenance programming (modifying existing software) is a very common task in both industrial and academic software projects. In the group project (see handout) you incorporate your own work into an editor framework provided in the directory `/cas/student/cisc322/f2016/editor`.[1] This will require you to do a small amount of *design recovery*: learning the structure and functionality of an existing piece of software, so that you can modify it subject to its structural constraints.

1. Review the concept of Java interfaces and having classes implement interfaces.
2. Familiarize yourself with the AbstractFactory and FactoryMethod design patterns from the readings. It would be wise to start this in advance of when we cover the material in class, but you can start the project without doing so.
3. Read the Editor Framework overview at http://cs.queensu.ca/home/dalamb/java/docs/ca/queensu/cs/dal/edfmwk/package-summary.html
4. Read the brief directions on how to add your own editing classes to the framework at http://cs.queensu.ca/home/dalamb/java/docs/ca/queensu/cs/dal/txt/package-summary.html and familiarize yourself with the three interfaces (`Document`, `DocumentType`, and `Application`) mentioned in the brief text editor description.

You won't need to create a `.jar` file or a manifest (`.mf`) file until the last phase of the project.

It should not be necessary to learn the inner details of the framework itself, beyond how the existing `Text`*`XXX`* classes interact with it. However, if you wish to read further, the full documentation of the framework and several other programs (in varying stages of completion) is at http://cs.queensu.ca/home/dalamb/java/docs.

1 If there is time for me to write new editor code to help in your implentation, the new version will be in a 2017 folder.

Your final system must run correctly on CASlab Linux, regardless of where you develop it.

You need not retain the old functionality. However, retaining both requires a deeper understanding of the design patterns used, so is a better result. The following directions assume you're replacing the old functionality; if you want to retain .txt editing, you'll need to study the existing code more thoroughly.

2 Getting Started

First, verify that the example editor works in your preferred development environment:

1. Create a new subdirectory and copy the contents of the `editor` subdirectory into it.
2. If you are running directly on CASlab Linux with access to a GUI, run `make` to compile and run `TextEditor`, the main class.
3. If you are running in your own environment, you must accomplish the equivalent (running Java on the main class, which compiles all the classes it depends on, and then running the result with a particular `.jar` file in the classpath).
4. Use the editor menus to make read and minor changes to `sampleText.txt`, then save it to a new file.

3 Version Management

You will need to apply some form of version management to develop your system incrementally and make sure you can revert to previous versions if changes introduce a bug. If you already are famliar with such a system, use it. Otherwise apply the following very simple ancient strategy.

1. Create a subdirectory named `V0` and move all the text editor files into it.
2. Copy the files with names starting with `Text` back into your main directory, renaming then to start with whatever is appropriate for your file type.
3. Edit the contents to change the class names
4. Before any significant new changes, create a `V1`, `V2`, etc. subdirectory and copy all the current files into them.

4 Sketch of Necessary Changes

I may augment this section in response to questions and comments.

4.1 MainPanel.java

This sets up the initial window used before any files have been opened. You shouldn't need to change it.

4.2 TextType.java

1. Delete the contents of `actionPairs`; this is for menu actions that already exist, rather than new ones you create.
2. Delete the body of the **`try`** clause in `getStaticMenu` but leave the **`try/catch`**. This is where

you will add you own menu items.

3. Change the `extensions` string array to list just the appropriate extension(s) for your new filetype.[2]

4.3 TextContents.java

This is conceptually part of `TextDocument`, so you could chose to merge the two if you consider it appropriate. You will need to modify the various I/O methods to use ones appropriate to your new filetype.

4.4 TextDocument.java

Use an appropriate Swing component instead of a `JTextArea` as the window component.

4.5 TextEditor.Java

1. Change the static strings that identify the editor.
2. Change references to TextXXX to something appropriately-named for your new filetype.

4.6 Other Files

You should feel free to split up your new Java files however seems appropriate, as long as the `XXXType`, `XXXDocument`, and `XXXEditor` classes remain.

You will not need the `Action` modules, except possibly a superclass serving the same purpose as `TextAction`). This would be a good idea if you want to centralize code to deal with things like translating Swing-specific information into specific data model elements.

5 Advice

Don't try to understand every detail of the existing software; there is too much of it for the time you have available. One of the values of software architecture is providing an appropriate level of abstraction that lets you comprehend the overall system without having to delve into details.

2 Some packages such as the Java image library can read several formats with different extensions.