LAB 2

COMP 354- Fall 23

Apekshaba Gohil apekshabagohil.it@gmail.com

Anam A. Shaikh anams403@gmail.com

Marzieh Adeli marzieh.adeli@gmail.com

Software Design Strategies e.g Text Editor with Smart Undo Capabilities

Architectural Strategies

In order to aim for a highly maintainable and scalable code, the three of the five SOLID principles of object-oriented design were implemented. (Janssen 2018)

Indeed as the subsequent section demonstrates, the single-responsibility principle (SRP) has been respected as every class of the application only tackles a single functionality or closely related functionalities of the overall program.

Moreover, the open–closed principle (OCP) has also been respected as code has been segregated in such a fashion that the implementation of the successives modifications don't require the rewriting of the other existing components.

Lastly, in order to achieve low coupling a keen attention has been accorded to the interface-segregation principle (ISP). For this purpose, the complex features have been subdivided into more easily manageable and understandable chunks without infringing on the territory of unnecessary complexity.

The topic that instigated the most decision within the team was the use of the appropriate data structure of the track of the changes/edits made by the user. Given the constraint of the arbitrary selection of an edit by the user and the possibility of performing various actions with the selected edit whether it's adding it to a group, undoing or deleting it, a unanimous decision of using the linked list was reached after weighing in the pros and cons of alternatives such as stack or multidimensional arrays. The latters were dropped as they required additional engineering to function adequately within the parameters of the editor. Thus, LinkedList emerged as the safest and wisest choice.

Architecture Level

System Architecture

Our team chose a top-down approach for the design of the system architecture. Classes and interfaces are designed with high cohesion and low coupling in mind. We achieved this by trying to make modules as independent as possible from other modules.

The following tables offer a quick overview of the classes and interfaces and the reasoning behind their existence.

Overview of the diagram **classes:**

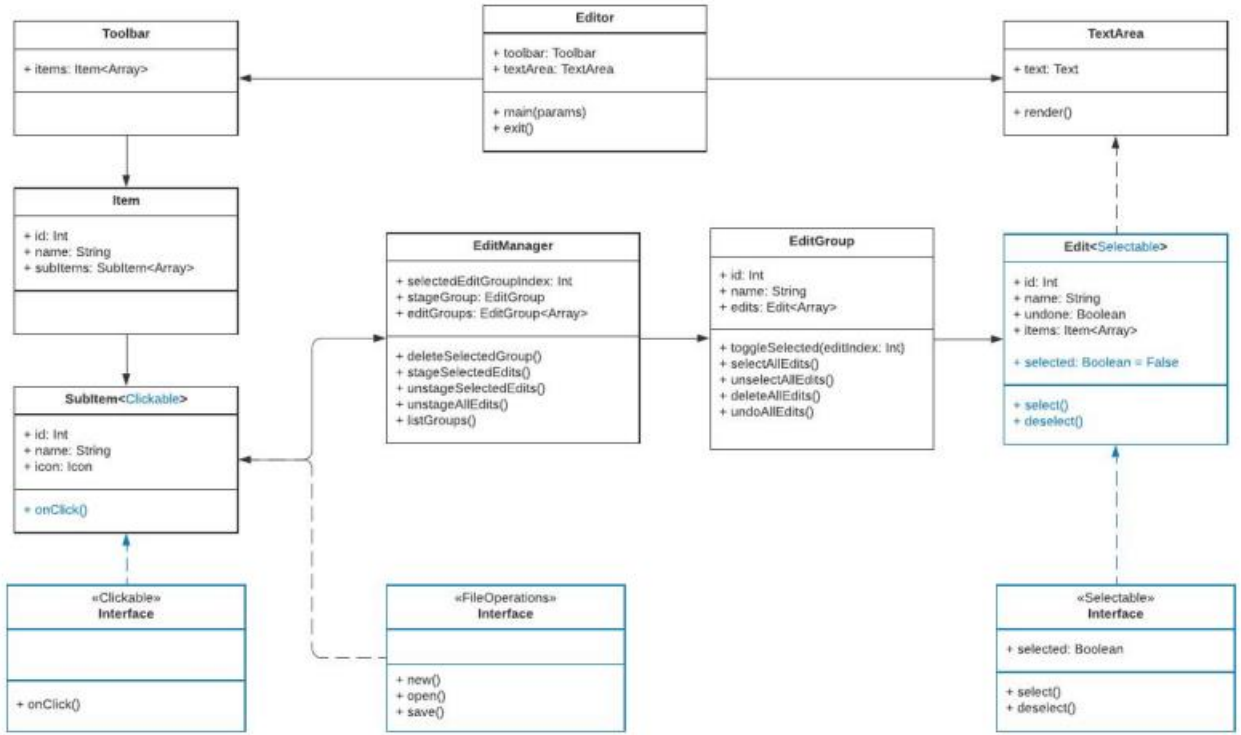
Editor	The main entrypoint of the application.
TextArea	The main textarea for editing.
Toolbar	The application toolbar. Contains multiple items.
Items	Items within the toolbar. Contains multiple subitems.
SubItems<Clickable**>**	SubItems within the toolbar items.
EditManager	Responsible for the general smart undo functionality.
EditGroup	Responsible for actions on multiple edits.
Edit<Selectable**>**	Responsible for identifying individual edits.

Overview of the diagram interfaces:

Clickable	OnClick() method enables clicking operation.
Selectable	Select(), Deselect() methods enable selection and deselection.
FileOperations	Save(), Open(), etc… operations used for file handling.

We chose to make interfaces part of our diagram because it's a core OOP concept that helps with modularity. For example since we know that FileOperations is an interface we can swap out that library at any time and as long as the new library implements FileOperations there will be no issues.

Architecture Level



Component Level e.g. of Editor Class

Notes: Have one Component Diagram as given in Tutorial example.

If you have two it would be considered as an add-on.

Editor

High cohesion:

- Components relating to running and terminating are grouped together to make it highly cohesive. The functions of main() and exit() in the program are mainly to run and stop the program. Editor executes methods from other components which are related with the functionality of running the Hanne Sergine editor.

Low coupling:

- Editor is linked to the Toolbar class as the toolbar is needed to enable users to navigate through the different features of the Hanne Sergine editor such as the File, Edit and Help tabs.

Definition:

- Editor is the main driver of the Hanne Sergine editor.

Responsibilities:

- The main responsibility of the Editor is to run the text editor program.

Composition:

- main (): It is the core of the program. It contains the instructions that make up the Hanne Sergine editor.
- exit(): This method terminates and closes the text editor completely.

Interaction:

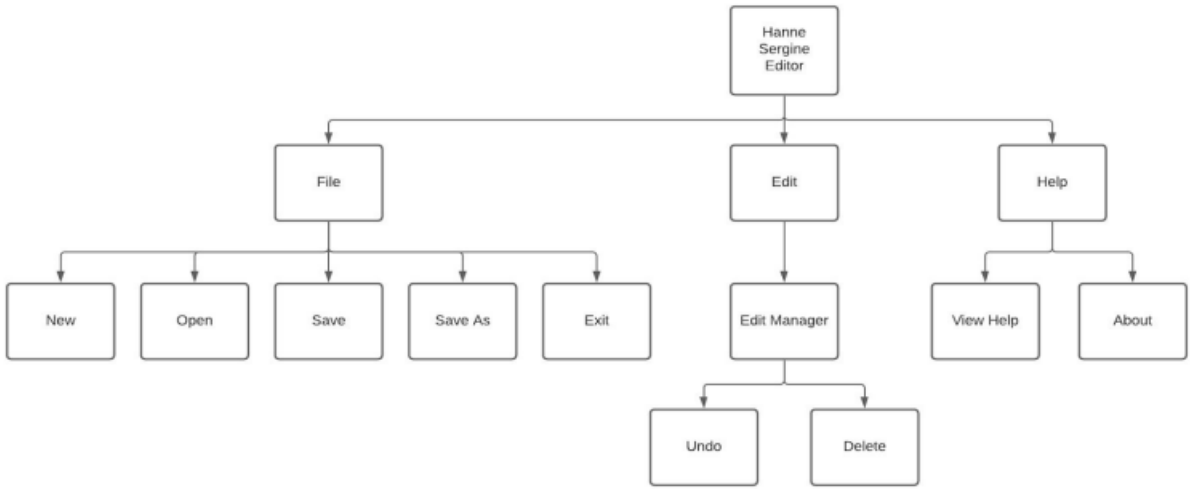
- It has a toolbar which consists of items and their subitems. These are the “File”, “Edit” and “Help” tabs seen in the UI section.
- It contains a Text area in which users can perform edits on their file.

Interface:

- Provides user interface and extended undo and redo capabilities that enable users to edit their file faster.

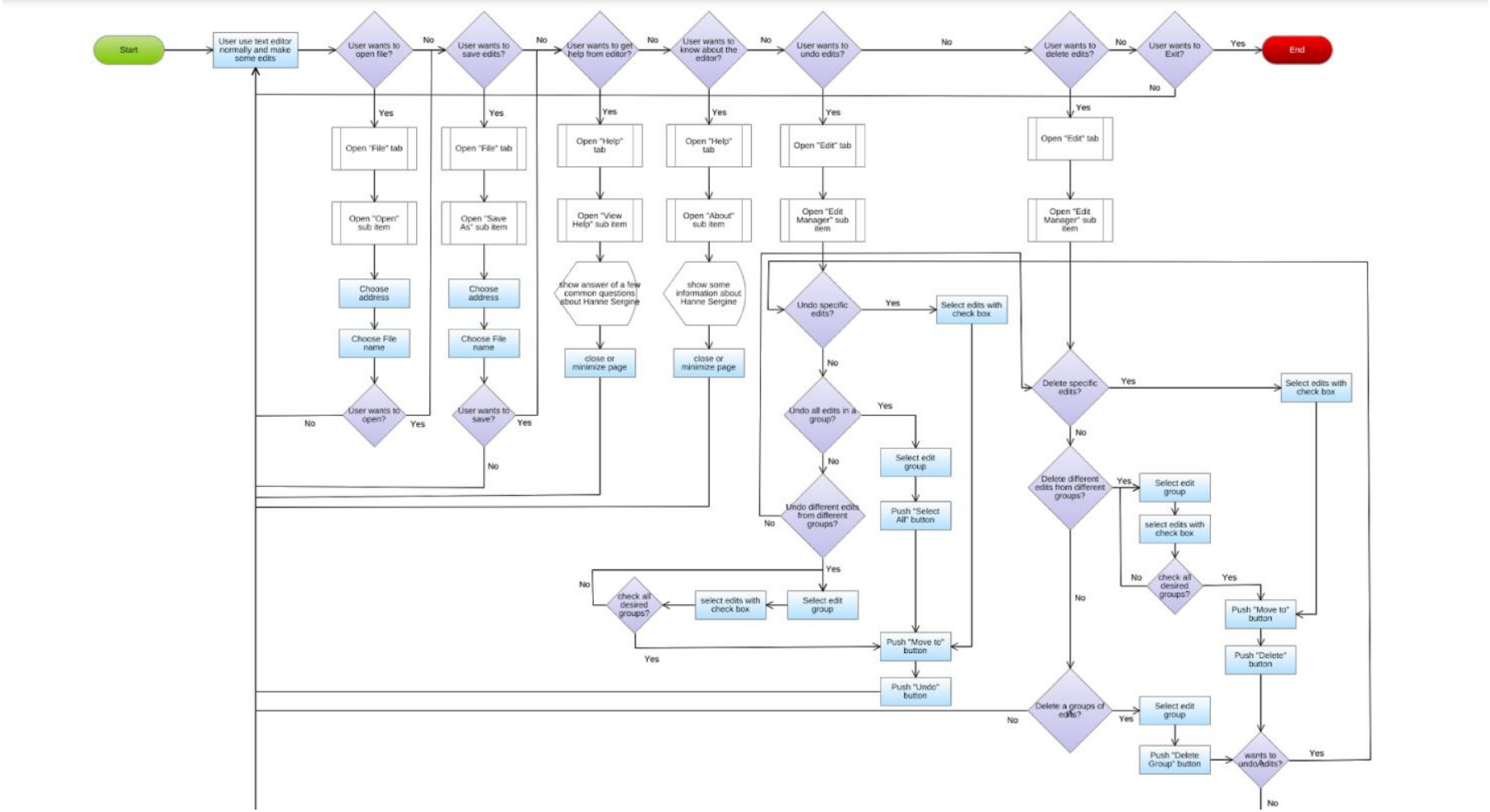
Interface Level e.g. Text Editor

1. Information Design



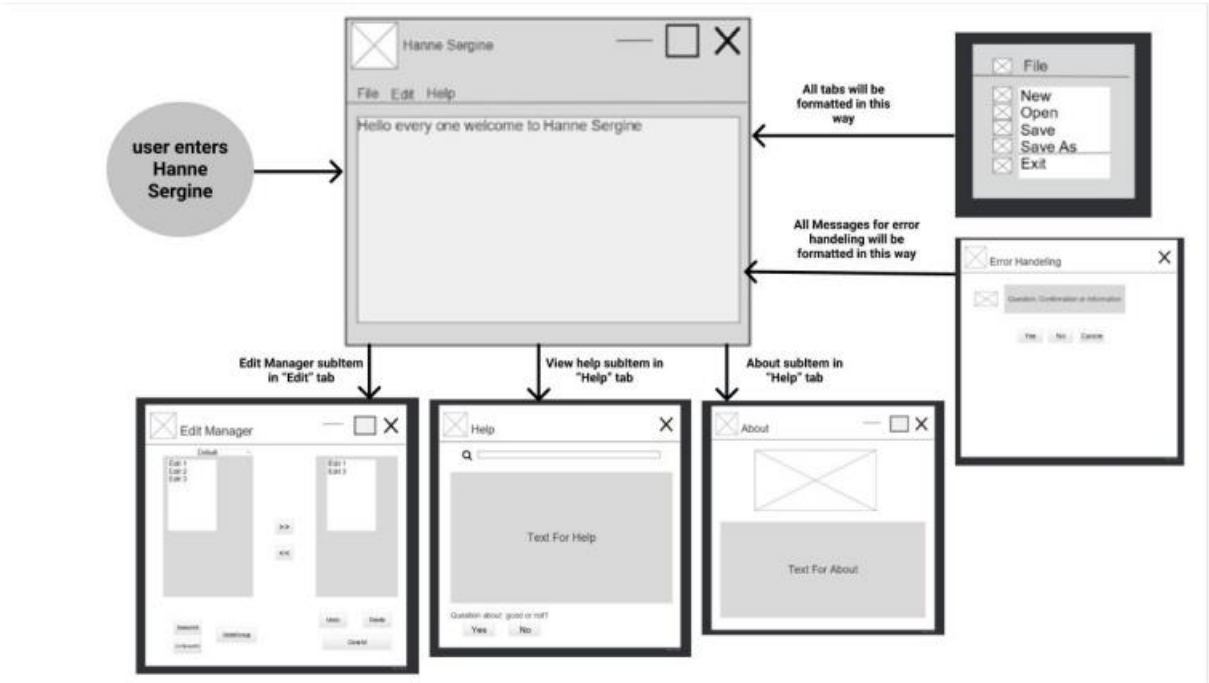
Interface Level e.g. Text Editor

2. Interaction Design



Interface Level e.g. Text Editor

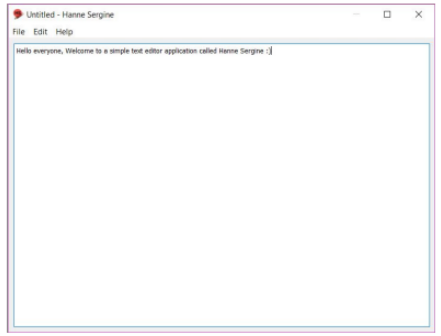
3. Visual Design



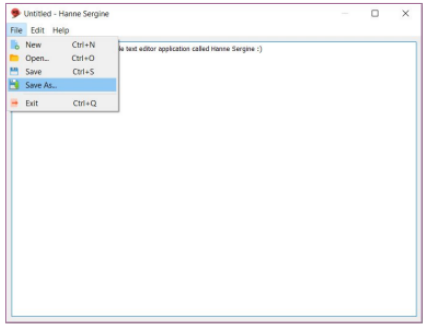
Interface Level e.g. Text Editor

4. User Experience Design

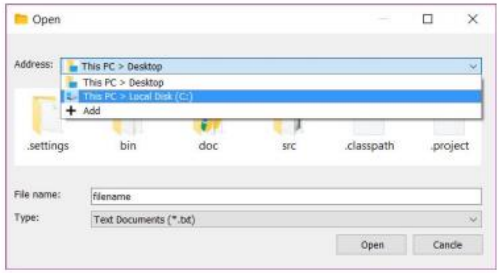
This editor includes three tabs called "File", "Edit" and "Help" at the top left of the page.



1- In the "File" tab, the user has five options: New, Open, Save, Sava As & Exit.

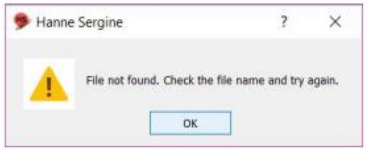


"Open" sub item:
When the user chooses the "Open" sub item, another page will be opened and the user can find and open the desired file according to the address, file name and its type.

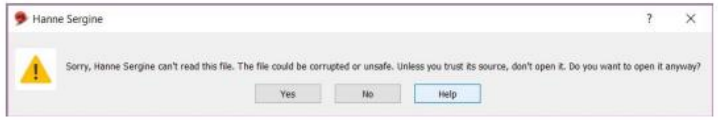


Also, In this option, there may be problems with opening the file, the list of which is as follows, and these errors must be managed.

- The user may enter the name of a file that does not exist in the system. In this case a message will come up and inform the user of this problem.



- The file may not be opened by the editor for any reason. In this case, a message will open and ask the user to decide for opening it,



Also in this case, the user can click on the help button and get information about why the file does not open.