UML Editor Specifications

Problem Statement

UML is an important tool for creating documents to visualize complex systems with graphical notation. Using UML, a user is able to grasp the basic understanding of various classes and utilities that a program brings to the table, without having to look at a single line of code. UML is widely used in the software engineering business to provide a quick and easy interpretation of a system at a glance. We've set out to create a UML editor for the purpose of aiding developers in creating the very documents that would help them in their design process.

System Personnel

Target users

Ideally, the audience of this program would be other software developers. A person with a background in programming and software engineering and a background in UML would be able to understand the data that is being portrayed in the diagram very easily with little learning curve.

System developers

Andrew
Bri
Don
Lukas
Nick

Operational Setting

Target Platforms

Linux, Mac, Windows

Software Environment

Java, as the entirety of the program is written in it, it is required to run it.

Useful Optional Software Environment

An image viewer would aid in reading the UML documents that are created with the editor itself. A file sharing service would be effective in sharing the documents created with this editor

Function Requirements

Functional Description

Overview

A GUI where the user can paint (to a default location), and afterwards drag, objects such as class boxes and UML relationships.

Feature List

Objects can be painted onto right panel
Objects can be moved by click and drag
Objects can be removed from the right panel
Class boxes can be populated with text
Relationships can be drawn between two objects, and are tethered to those objects
Diagrams can be printed
Diagrams can be saved
Actions may be undone
Actions may be redone after an undo command has been executed
Files may be opened that were saved by the program

User Interfaces

Overview

The left panel is a list of object buttons that the user can click. On the right pane is the space where the objects will be drawn and interacted with.

Menus

The normal window menus, file, and view menus are included on the menu bar. File has functionality for sub-menu options: New, Open, Save, Save As, Print, and Close. Edit has functionality for sub-menu options: Undo, Redo, Copy, and Paste.

Inspectors

No inspectors in this iteration.

Use Cases

Drawing a Class Box

- User clicks on “Class Box” button.
- The system recognizes the user wishes to paint a class box.
- The user then clicks a location on the right panel.
- The system paints the class box at the desired location from the top-left panel of the class box object.

Drawing a Comment

- User clicks on “Comment” button.
- The system recognizes the user wishes to paint a comment.
- The user then clicks a location on the right panel.
- The system paints the comment at the desired location from the top-left panel of the comment object.

Populating a Class Box

- *Precondition: A class box is painted on the right panel.*
- The user clicks in the desired field of the class box they wish to edit.
- The system responds by creating an editable text area in the class box.
- The user enters the desired text in the class box.
- The system responds by populating a the class box with desired text.

Deleting a Class Box

Precondition: a class box is painted on the right panel

- The user selects delete on the left panel.
- The system recognizes that the user wishes to delete a class box.
- The user selects the desired class box.
- The system unpaints the object and any associated relationships to said object.

Printing a Diagram

- The user selects the “File” menu in the top left-hand corner OR hits “Control” and “P” simultaneously.
- The system recognizes the user wants to print a diagram, and opens a page setup window.
- The user utilizes the fields provided to create an image appropriate to their needs.
- The system passes that information to a print page.
- The user selects the desired printer and number of pages, and hits okay.
- The system prints out an appropriate image within the desired parameters passed into the page setup window, with appropriate number of copies.

Dragging an Object

- *Precondition: At least 1 object is painted on right panel*
- User left clicks(and does not release) the object to be dragged.
- System recognizes that the user is selecting this object to be dragged.
- User drags object to desired location.
- System updates coordinates of the object as it is being dragged.

Drawing Generalization Relationship

- *Precondition: at least two class boxes are painted on the right panel*
- User clicks on the "Generalization" button.
- The user then selects two desired class boxes' gray bottom markers, starting with the initial class box, and following with the target class box.
- The system paints a Generalization between the two desired class boxes in the appropriate fashion.

Drawing Association Relationship

- *Precondition: at least two class boxes are painted on the right panel*
- User clicks on the "Association" button.
- The user then selects two desired class boxes' gray bottom markers, starting with the initial class box, and following with the target class box.
- The system paints an Association between the two desired class boxes in the appropriate fashion.

Drawing Dependency Relationship

- *Precondition: at least two class boxes are painted on the right panel*
- User clicks on the "Dependency" button.
- The user then selects two desired class boxes' gray bottom markers, starting with the initial class box, and following with the target class box.
- The system paints a dependency between the two desired class boxes in the appropriate fashion.

Drawing Aggregation Relationship

- *Precondition: at least two class boxes are painted on the right panel*
- User clicks on the "Aggregation" button.
- The user then selects two desired class boxes' gray bottom markers, starting with the initial class box, and following with the target class box.
- The system paints a aggregation between the two desired class boxes in the appropriate fashion.

__Drawing Composition Relationship__

- *Precondition: at least two class boxes are painted on the right panel*
- User clicks on the "Composition" button.
- The user then selects two desired class boxes' gray bottom markers, starting with the initial class box, and following with the target class box.
- The system paints a composition between the two desired class boxes in the appropriate fashion.

__Saving a File__

- The user selects the "File" menu in the top left hand corner, and selects "Save As", OR if it is a new document, select "Save" from the file menu. (Control + S saves the file as well.)
- The system recognizes the wish to save and opens a save explorer.
- The user selects the designated save location and filename, and selects okay.
- The system responds by saving the file with the selected name and in the selected location.

__Opening a File__

- *Precondition: A file has been previously saved by the program.*
- The user selects the "File" menu in the top left hand corner, and selects "Open" OR hits "Control" and "P".
- System system recognizes the user's wish to open a file and opens a file explorer.
- The user locates the desired file and selects "Open"
- The desired file is then opened in the corresponding window of the UML editor.

__Undoing an Action__

- *Precondition: An action has been taken in the program.*
- The user selects the "Edit" menu in the top left hand corner and selects "Undo", OR hits "Control" and "Z".
- The system recognizes the user wishes to undo the prior action, and returns the program to the state JUST before that undesired action was performed.

__Redoing an Action__

- *Precondition: An action has been undone that can be redone.*
- The user selects the "Edit" menu in the top left hand corner, and selects "Redo", OR hits "Control" and "Y".
- The system recognizes the user wishes to redo a previously undone action, and restores the program to a state JUST after the desired action was undone.

Non-Functional Requirements

Reliability

Reliability is something should always be a primary concern of good software engineers. To achieve a good level of reliability, with copious amounts of testing to back our claim, is a conscious effort to make sure all errors are caught and thrown exceptions for, with a prime example being using save as in a directory without proper credentials, the program catches the error without crashing and terminating the active window.

Performance

With performance being our primary goal, we want to make sure this will always be our front and foremost priority, with as little sacrifice to reliability and portability as possible. What we have done to achieve this is by organizing the code in such a way that the UMLView is what the user sees, and is completely separate and independent of UMLController, which handles all of the interactions with the GUI.

Usability

Within this current iteration, everything is properly labeled, and relatively self explanatory in the hands of an experienced user with a background in UML. Usability is not necessarily achieved in this current iteration because a few of the features do not have the expected functionality of their functional counterparts in traditional programs.

Portability

With many different systems running around in the world of software engineering and development, we aim to create a program that will run seamlessly on the three big operating systems that dominate today’s market (Mac, Windows and Linux). Utilizing Java, a very well known programming language, and a java environment to run in, the ability to run our software on any of the three big Operating Systems in use today will be a very straightforward and easy task.