# Overview

The following document contains the documentation for the fifth iteration of the *ObjectDraw* application. This includes a high level class diagram of the current system, low level diagrams of any new or updated classes, the architectural model of the system, collaboration diagrams for new features, and a final discussion of design decisions made.

The main objectives for this iteration were as follows:

* Implement the eraser tool such that if clicked while an object is selected, said object is removed from the canvas and the change persists
* Implement the de-selection of objects if any of the tools are clicked on so the user can go back into drawing mode immediately
* If time permits – allow the filled state of Two End Shapes (Rectangles, Ovals and lines – not really applicable to lines) to be toggled by first selecting the object and then toggling the filled checkbox

How we handled the first two objectives will be discussed at the end of the document. To handle the toggling of the filled state we followed a procedure similar to when we implemented the ability to change the color of objects on the fly, but because only TwoEndShape descendants have the filled variable, there was a little speed bump. We solved this by first checking if the selected object was an instance of the TwoEndShape class and if it was we allowed the toggling, for any other shapes nothing is actually done when the user toggles the filled checkbox.

# Iteration 4

## Class Diagram - High Level

***\* Note: The only change from the previous iteration is the addition of the EraserTool class***



## Class Diagram - Low Level

Those classes whose instance variables and/or methods were updated are detailed below:



## Collaboration Diagrams

***\*Note: No significant feature or implementation changes occurred between iteration 4 and the iteration 5 so no collaboration diagrams have been drawn up.***

## Architectural Model

The current architecture follows an MVC architectural pattern where controllers listen for events fired from the various views. Depending on the action fired, the controller which was listening for and caught the event, updates the model of the particular shape object and notifies the view of those changes.

***\*Note: No changes were made to the architecture from iteration 3.***



## Use Cases

|  |  |  |
| --- | --- | --- |
| Name: | **Erase Object** | |
| Actors: | User (initiates) | |
| Pre-Conditions: | User has selected an object on the canvas with the *Select Tool* | |
| Flow of Events: | **Actor Action** | **System Response** |
| 1) User clicks the *Erase Tool* button in the menu or the toolbar | The object is removed from the list of objects and the canvas is refreshed |
| Post-Conditions: | The list of objects in the Drawing Canvas View has removed the previously selected object and the object is no longer on the canvas | |
| Quality Requirements: | Must persist the object that was removed | |

|  |  |  |
| --- | --- | --- |
| Name: | **Deselect Object** | |
| Actors: | User (initiates) | |
| Pre-Conditions: | None | |
| Flow of Events: | **Actor Action** | **System Response** |
| 1) User clicks any tool on the toolbar or menu besides the *Erase Tool* | Any previously selected objects selected using the *Select Tool* have been released and it is reflected on the canvas |
| Post-Conditions: | None of the objects on the canvas may be selected. | |
| Quality Requirements: | None. | |

|  |  |  |
| --- | --- | --- |
| Name: | **Change fill state of TwoEndShape Objects** | |
| Actors: | User (initiates) | |
| Pre-Conditions: | User has selected an object on the canvas with the *Select Tool* that is an instance of the *TwoEndShape* class (Rectangle, Oval or Line – not applicable for the line though). | |
| Flow of Events: | **Actor Action** | **System Response** |
| 1) User clicks the checkbox that toggles the filled state of an object | The filled state of the selected object toggles (ie – if it was previously filled it is now not filled) |
| Post-Conditions: | The selected shape’s fill state has been changed and it is reflected on the canvas | |
| Quality Requirements: | The changed fill state must persist. | |

## Discussion

We’ll start the discussion of how the de-selection of objects was implemented because that is the simplest. This was done by simply calling the method in the DrawingCanvasView class that removes any selection (removeSelection()) when the actionPerformed() function was called in the ToolController class. This class is called whenever a user changes the current tool from either the menu or the toolbar. In this way we were able to deselect an object whenever the user changed the tool.

To implement the EraserTool we again went back to the actionPerformed() method in the ToolController class. We added a conditional check to see if the currently selected tool was the SelectTool, if the tool the user was trying to select was the EraserTool and if there was an object selected. If all of these checks passed we called the eraseObject() function of the EraserTool class. This function simply removes the item from the list of objects on the canvas and refreshes the canvas.

The simplicity of these tasks has led us to not create any collaboration diagrams for this iteration. We felt they were better explained with discussion and the use case diagrams above because the code was not complex and ‘interesting’.