# Object Oriented Analysis and Design Project: Part 6

## 1 Summary

- Name: Yoshinari Fujinuma
- Github link: `https://github.com/akkikiki/csci5448_project`
- Title: Machine Learning (ML) Model Debugger
- Project Summary: A CUI tool that could visualize and interactwith a machine learning model (e.g., neural networks). The main objective of this tool is to let users save, load, train, and debug a trained model in an feasyc and nituitive way.

## 2 Features Implemented and Not Implemented

| ID | Requirement | Implemented |
|---|---|---|
| 1 | Plot and visualization of ML models | |
| 2 | Allow to be used by multiple users | ✓ |
| 3 | Model parameters should be easily tweakable | ✓ |
| 4 | A user cannot load another user's model | |
| 5 | A user can tweak the parameter of a trained model | ✓ |
| 6 | A user can delete a model | |
| 7 | A user cannot delete data | |
| 8 | A user can confirm whether a model's status (finished tranining or not) | |
| 9 | An admin can delete a user | |
| 10 | An admin can delete a model | |
| 11 | An admin can delete data | |

**Table 1:** Project Requirements. The features with ✓are implemented in this final project.

## 3 What Changed from the Initial Class Diagram

**More Classes** Since I did not have specific design pattern in mind when drawing the initial class diagram, I added more while implementing those.

**Separating out Model, View, and Controllers** Initially, I tried to implement everything into one class, but that soon screw up and violated the "separation of concerns" principle. Instead, I separate them out and created another class called "Driver" to communicate with each controllers.

## 4 Design Patterns

### 4.1 Memento

Figure 1 shows the class diagram for the Memento pattern I implemented. I choose Memento pattern to let the users save the current state of tweaked parameters. Furthermore, users would like to go back to the state specified rather than doing undos one-by-one.

I imeplemented this in the “classifier_caretaker.py”, “classifier_memento.py”, and “classifier_originator.py”
.

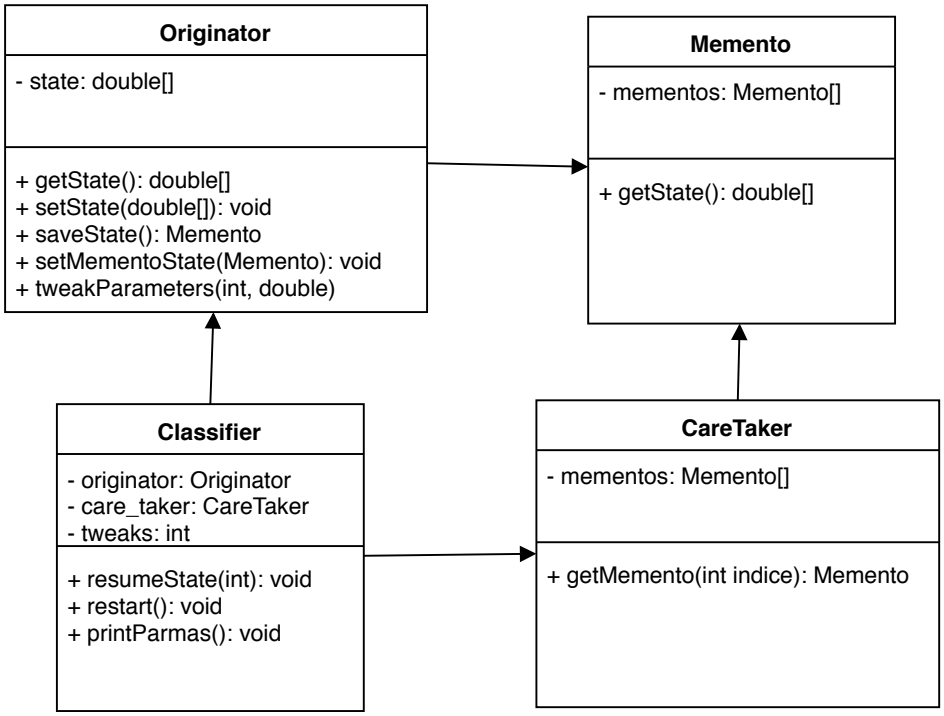


**Figure 1:** Class diagram for Memento pattern.

### 4.2 Factory

Figure 2 shows the class diagram for the Factory pattern I implemented. I choose Factory pattern because I wanted the same function across all corpus file types. Therefore, I thought factory design pattern is most suitable for it.

I imeplemented this pattern in “corpus.py”.

### 4.3 State

Figure 3 shows the class diagram for the State pattern I implemented. I choose State pattern to implement the “Undo” function to go back one menu transition. In the future, there will be more than one level hierarchy in the menu transition.

I imeplemented this pattern in “menu.py”.

## 5 What I have Learned

Classes are more modular than I thought before implementing each design pattern.

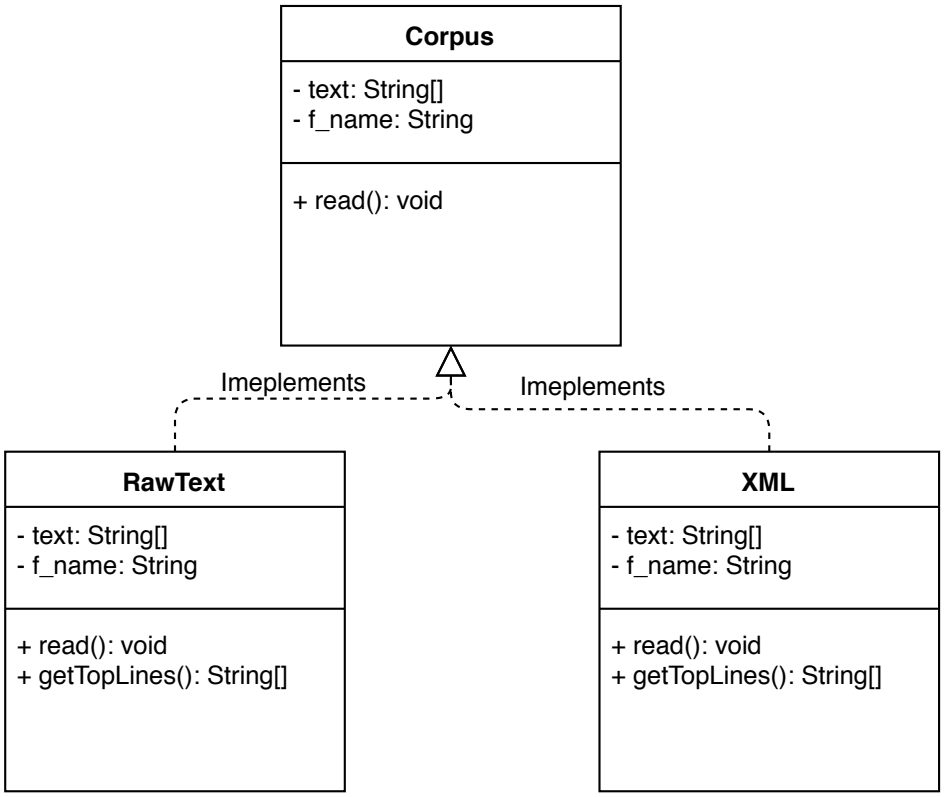


**Figure 2:** Class diagram for Factory pattern.

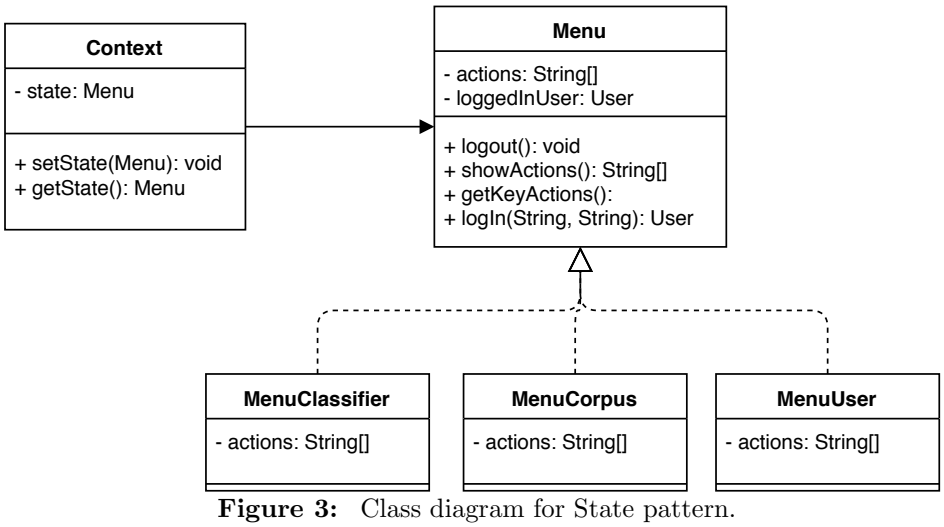


**Figure 3:** Class diagram for State pattern.