Michael Prigmore

UI and Domain Layer Separation Report

When done correctly, separating the UI and domain layers of a software application has many advantages. It allows developers to make changes to one layer without breaking the other (or the tests involved). These days software companies spend tremendous amounts of time and money updating and maintaining existing applications. This makes the separation of the UI and domain layer even more important. However, separating the UI and domain layers in a software program is a difficult task. A software implementation of something as simple as the dice game Skunk becomes very challenging to write when trying to separate the UI and domain layers.

In my linear algebra app, I strove to decouple these layers as much as possible. The presentation layer of my program needed to do a few things including:

1. Greet the user.
2. Allow the user to enter a matrix (number of rows, number of columns, and values) into the console.
3. Provide input validation.
4. Display matrices in the console.
5. Instantiate the calculator class (Reduced_Echelon_Calculator is the class name).

Summarized and simplified, the domain layer needed to:

1. Have the matrix as instance data.
2. Implement the algorithm to row reduce a matrix into Echelon Form.
3. Keep track of pivot positions.
4. Implement the algorithm to row reduce a matrix in Echelon Form into Reduced Echelon Form.
5. Allow the UI to print the calculated matrices.

What makes the separation difficult for me is trying to determine how much separation is necessary. There is a little bit of subjectivity concerning this matter. In my first iteration of the project, I decided to instantiate the matrix inside the UI and then pass it to the calculator. I felt that this would be appropriate since the UI is in charge of getting the matrix size and values from the user. However, when I finished the program and looked at the UML class diagram, I thought it would be better to have the calculator instantiate the matrix. I moved things around so that the UI instantiates the calculator and the calculator instantiates the matrix. Only a few of my unit tests were affected by the change with gave me confidence in the program's portability. Now the console requests values from the user and updates values in the matrix using the calculator instance it has. Decoupling could be taken even further. The UI could be set up to pass only numbers to the calculator and let the calculator handle the matrix completely. However, since the UI does have an instance of the calculator, it can access the matrix through getters and setters. It felt a little too extreme to not allow the UI to use these getters and setters.

I believe the program now has reasonable separation of the UI and domain layers. If the UI outputs were to be changed into another language, the program would still work as it should. All the heavy lifting of calculations is performed in the domain layer. Any changes (and I made many throughout the project) to the way matrices are transformed does not harm the UI layer.