The interfaces were reasonably flexible except for the IViewWorksheet interface (in their model package) which was generic. Therefore, their SimpleWorksheet class that implements that interface was tightly coupled to their implementation of the Cell class. Therefore, to work around this, we had to ask for their implementation of their Cell class and construct their Cell class with our implementation of Cell to adapt our Cell implementation to their Cell implementation.

While looking at your code, we realized that there were some problems with the design. When testing the infinite scroll, there was visible lag when scrolling up and down. Your Infinite scrolling also does not work unless you scroll with your mouse pad. You are physically unable to have infinite scroll from just using the scroll bar. Also, when we loaded our Triangular number text-files, only column A appears. This must mean that the default max column and rows values are hardcoded into their Worksheet class which should not be the case. Additionally, since the default max column and rows were not defined in the EditableView class, then we were unable to scroll horizontally or add any new columns, making you unable to edit any cells right of Column A. Therefore, an easy fix is that those values should be defaultly defined inside your GUI view classes instead.

Improvement in your java documentation is also recommended. For classes such as ValueVisitor, there needs to be a more clear explanation of what methods return. There is also some methods that leaks too much information from the model, like the method getAllCells() in the IViewWorksheet interface which pretty much returns the current HashMap field object. Re-implementing the method in our adapter class proved to be redundant in the end too, as we never needed to use that method at all. Overall, however, the javadocs were nicely explained and self-intuitive.

Other things that we noticed were redundancy in your setView method. The controller constructor should already be setting the view so that you wouldn’t need to call setView after instantiating your controller in the main class. Alternatively, instead of your current implementation, there could be a function called runController, for example, that handles both the rendering of the view and setting of the view so you wouldn’t need to call both methods separately in the main class.

Overall however, everything else was pretty reasonable. Their view we didn’t need to touch as they included all their listeners in their view via lambda functions. Therefore, reimplementing their controller was self-explanatory. However, since the view is buggy in many aspects, especially scrolling where it was especially laggy depending on the number of cells on the screen and didn’t have proper infinite scrolling in general, we are not sure if all the features work. However, our model seems to work with theirs (for ex. automatic updating of cells) and checking for cycles. The only other recommendation I have is the delete feature. It should be using the delete key instead of having a seperate delete button, and also, not having the cell automatically evaluate a cell once a new one is created (via infinite scrolling) or deleting an existing cell with their delete button. It would probably be easier to use JTables instead of JPanels, as it especially gets extra buggy/laggy when running their implementation that they created on MAC OS to Windows.