I update my design a great deal from 4a to 4c. Admittedly, this is because my design for 4a was so terrible that there wasn't much that could do with my initial design. I pretty much did a complete overhaul, but I'll talk about the most significant changes here.

I eliminated the turn class from my initial design. I had that in at the beginning because I thought that the Carcassonne class wasn't supposed to be responsible for anything really besides for starting and ending the game. This turned out to be incorrect, so there was no reason to have a Turn object since it doesn't make sense as far as real world representation. All of the functionality that would go there was pretty much moved to either the Carcassonne class or the board class.

I added a TileStack class to represent the deck of tiles which have not been played yet. It made sense to have this be its own class because it needs to be initialized, shuffled, and drawn from which is enough to warrant not being a local object in the Carcassonne class in my opinion.

I changed my object model to have segments all be represented in one class, not with road, field, cloister, and city being subclasses of Segment. I describe this in detail in my updated Object Model Justification. I decided to have the Feature class be an abstract class with the different types of features extending it. This is also discussed in detail in the updated Object Model Justification.

In my original design I had the board be immutable. This can be seen in the interaction diagrams with the board class passing the updated board back to the turn class. I decided to make the board mutable in the end. The intent of this was to reduce the need to return the updated board object from functions that modify it. Since my

board contains a list of all the tiles in the board and all of the features it has to be updated every time any segment or board or tile is updated. I knew that this would occur frequently so I decided to make the board mutable so I could update it directly without returning the updated board from any function that updates it. In retrospect this may have been a mistake. Prof. Bloch said in lecture the other day that when possible we should make our data structures immutable to prevent unwanted tampering or effects. I think that my reason for making it mutable does not outweigh the risks for unwanted side effects that come with it.

I believe that my design is fairly robust. I tried my best to throw helpful exceptions in any situation where it is theoretically possible, even if it should not ever happen in the flow of a game. There are lots of double checks in places where a second check should never be necessary. I thought about efficiency in this design, but didn't end up over-optimizing. For example: my representation of a board as a 2-dimensional list instead of a map will for sure lead to reduced efficiency. I considered this and tested early on whether this loss of efficiency would mean anything in terms of performance. I saw that this was small-scale enough that it did not make gameplay any slower to have the representation as I had it. Therefore I felt like the better chance at functional correctness I had with my list representation was worth it over the efficiency optimizations to be had with a map representation. My board representation would likely become an issue if we played the game with hundreds of tiles, but as the rules state that there will only be 72 I didn't feel this would be an issue.