Not much is updated from my previous rationale, the designs have remained generally based on delegation. use of observer patterns, use of interfaces, etc. The use of observer patterns are greatly evident in the GUI code, for buttons and other action listeners, especially the textfield input in the main game window. I also used threads, approx. 2 times (to initialize the Game object, due to it's possibly slow creation of the Dictionary object, and to run a getMoves() checker thread).

I have special tiles that implement the SpecialTile interface which stores their locations, icons and a function that calls the defining property of each special tile. The letter and blank tiles implement Tile and store location, icon, character (either a letter, or "_" for blank tile), and score value (0- 10). After much planning, I decided not to unite special tiles and letter tiles under one interface "Tile" because of the vastly different purposes and accessibility of the two types of tiles. Letter tiles are important to word-validation algorithms and displaying in the GUI and are randomly given to players. Special tiles are purchased, hidden and don't have game value unless a tile is placed on top of it. So I made them separate groups of classes.

I made a "Square" class to replace the idea of using one or more two-dimensional arrays that each store separate things such as score modifiers or letter tiles in play. A Square class can be represented in a 2D array in Board.java, and each Square can keep track of everything on that square. It would have an owner for each SpecialTile to help the GUI display them correctly. Another implementation I'm considering is only having the active player's class be in charge of keeping track of which SpecialTiles it has used, and only display it's own.