Discussion

I added two classes. ScoreWord: There are a pair consists of a square and a score, the score is responsible for the change of scored due to letter bonus, negative tiles, and etc., I do not want the score inside square to store that changed score. CurrentMove: Assigned most of the check, sort and modification work to the CurrentMove class. there are three instance variables, currentSquares is a list of squares, which are the squares the player placed at that round; scoreWordSquares is a List<List<ScoreWord>>, it stores all possible valid (or not) words, a List<ScoreWord> consists a word; newWords is a List<String> corresponding to the scoreWordSquares, newWords is responsible for checking valid from dictionary.

Then I decreased some of the methods from Board class. Because board should not be responsible for the current move action. So Game would have an instance of CurrentMove class, and CurrentMove would be responsible for all current move actions, including check isValid()'s helper methods, letter and word bonus calculation, and different special tiles' effect to the current squares. I don't need to tell each kind of bonus square, just first execute letterBonus() then wordBonus(). Also, I don't need to tell each kind of special tiles, just execute the effect() method. Inside the effect(), it would do a lot of things that I showed on the diagram. I added a method called checkFirstWordCol(game) inside CurrentMove class, to check if the first word is valid.

I deleted bonusFlag and changed to two integers, letterBonus and word Bonus. They are default as 1. They are multipliers that applies to all squares. If we call getLetterBonus() and getWordBonus() from CurrentMove class, they would automatically calculate the bonus points.

I added a method "isAdjacentToOldWords(Game game)" to the CurrentMove class, also added it to the object model. It is to check if the new tiles are adjacent to old words. For example, you cannot place a word far away which is not adjacent to the old words. Because I forgot to check that inside the isValid().

I added the limit of special squares a player can purchase, which is 10. Because it makes the gui easier to manage. I deleted the button attributes from game.java, because gui can handle that.

By these changes, I find my design having higher cohesion than my original design, because each class does what it is responses. Also, by doing this, the game has more control responsibility. Though the code reuse is not very satisfiable inside the CurrentMove class, the helper methods for validation are divided into two parts. But the division of labor is also achieve by this means. The implementation becomes easier and clearer.

This is a valid design program because it covers all aspects of the game with maximum code reuse. I achieved the goal of reuse by providing abstract classes for SpecialTiles. There is good division of labor, since I almost separate everything into different classes. Since the design is pretty clear, it achieved the goal of understanding and maintenance. With this design, I have made sure to have high cohesion, by providing a different class for each new step in the game. This also reduces the dependencies and lowers coupling. And object oriented design is achieved.

Additional Special Tile: ScoreSwitchTile
When the tile is triggered, the scores of the player who originally played this tile and the player who triggered the tile will be switched. The score switch occurs after the score has been updated for the current turn. A player triggering her own score-switch tile has no effect.

I actually did this function very simple. My design is to treat all special tiles the same and trigger the effect() method when appropriate.
Inside the effect() of ScoreSwitchTile, I first switch the score of the owner of the ScoreSwitchTile and the score of current player, regardless of the current round's score. And then I set the playerToAddScore to the owner of the ScoreSwitchTile (which should be currentPlayer if there is no ScoreSwitchTile).
Then inside the move() method in Game class, trigger the effect of all special tiles. Finally, instead of adding current round's score to currentPlayer, I add the current round's score to playerToAddScore. And playerToAddScore is currentPlayer when no ScoreSwitchTile is triggered, playerToAddScore is the owner if the ScoreSwitchTile is triggered.

PS:
The explanation to regain points from milestone A can be found partly in discussion and rationale and mostly in response.