Assignment 4

William Shin (wshi593)
SOFTENG701: Advanced Software Engineering
Development Methods
Department of Software Engineering
University of Auckland
wshi593@aucklanduni.ac.nz

***Abstract*—This report outlines the impact of changing the user interface of Kalah which was developed in Assignment 3 from a board that is displayed horizontally to one that is displayed vertically. An assessment of the impact of the change case will be given, followed by a plan containing the minimal changes required to satisfy the new requirements. Then comments on each step of the plan and the whole plan overall as well as the challenges faced will be presented. The report concludes with a discussion on the accuracy of the changeability assessment proposed in Assignment 3.**

Keywords—Kalah, impact, change case, requirements, plan, changeability assessment

I. Introduction

The 4th assignment of SOFTENG701 introduced a new requirement to the implementation of Kalah in Java that was developed for assignment 3. In particular, the presentation of the Kalah board was to be changed from one that was horizontal to one that was vertical. As the Model View Controller (MVC) pattern was used, implementing this change did not require a significant amount of effort. Notably, a new view class, *VerticalAsciiKalahView*, was added, along with its utility class, *VerticalAsciiUtil*, and was initialized in the *Kalah* class. To revert to the old functionality, only one line of code needs to be changed involving initializing *AsciiKalahView* instead of *VerticalAsciiKalahView* in the *Kalah* class.

II. Impact Assessment

On a scale from 0.0 to 1.0, the impact of the change case in terms of functionality is approximately 0.2. This is because the core 'gameplay' of Kalah is insignificantly changed as only the presentation of the board is changed, rather than the game's mechanics or controls.

III. Planning

A plan, consisting of the minimal number of steps, was created before implementing the new requirement. The plan, excluding setting up the development environment to run a new test suite, consisted of the following steps:

1. Find the *AsciiKalahView* class.
2. Change the accessibility modifier of the *io* and *gameController* fields from *private* to *protected*.
3. Change the accessibility modifier of the *printPlayers()*, *printHorizontalBorder()* and *printDivider()* methods from *private* to *protected*.
4. Find the *AsciiUtil* class.
5. Change the accessibility modifier of all fields from *private* to *protected*.
6. Change the accessibility modifier of the *repeatString()*, *repeatChar()* and *rightAlignNumber()* methods from *private* to *protected*.
7. Add a new class called *VerticalAsciiUtil* that *extends AsciiUtil*.
8. Add the constructor for *VerticalAsciiUtil*, with the same parameters as *AsciiUtil* and call its *super* constructor with these parameters.
9. Override the *horizontalBorder()* and *divider()* methods and implement them to appropriately format and return *String* objects representing the top and bottom borders of the board and the divider between player stores and houses.
10. Add a new class called *VerticalAsciiKalahView* which *extends AsciiKalahView*.
11. Add an *AsciiUtil* field called asciiUtil.
12. Add the constructor for *VerticalAsciiKalahView* which takes the same parameters as *AsciiKalahView* and call its *super* constructor with these parameters. In the constructor, initialize the asciiUtil field with a new instance of *VerticalAsciiUtil* with the same parameters as found when initializing *AsciiUtil* in *AsciiKalahView*.
13. Override the *printPlayers()* method. In the method, call *printEvenPlayerStores()*, *printDivider()*, *printPitsForPlayers()*, *printDivider()* and *printOddPlayerStores()* in this order.
14. Add *printEvenPlayerStores()*. In this method, build a *Map* that maps even player numbers (integers) to the players' stores. Call *VerticalAsciiUtil#evenPlayerStores* with the asciiUtil field and pass the map as a parameter. Print the result of this method with the *io* field.
15. Similar to #14, add *printOddPlayerStores()* which builds a *Map* that maps odd player numbers (integers) to the players' stores. Call *VerticalAsciiUtil#oddPlayerStores* with the asciiUtil field and pass the map as a parameter. Print the result of this method with the *io* field.
16. Add *printPitsForPlayers()* which builds a *Map* that maps every player number (integers) to the players' houses (list of *Houses*). Call *VerticalAsciiUtil#pitsForPlayers* with the asciiUtil field and pass the map as a parameter. Print the result of this method with the *io* field.
17. Add *evenPlayerStores()* to *VerticalAsciiUtil* that takes in a *Map<Integer, Store>* as its parameters. Format the appropriate result, by looping through the player numbers and appending an appropriate *String* depending on the player number's parity.
18. Similar to #17, add *oddPlayerStores()* to *VerticalAsciiUtil* that takes in a *Map<Integer, Store>* as its parameters. Format the appropriate result, by

looping through the player numbers and appending an appropriate *String* depending on the player number's parity.

19. Add *pitsForPlayers()* to *VerticalAsciiUtil* that takes in a *Map<Integer, List<House>>* as its parameters. Format the appropriate result, by looping through the player numbers and appending an appropriate *String*.
20. In the Kalah class, initialize *VerticalAsciiKalahView* instead of *AsciiKalahView*.

IV. Execution

Following the plan and implementing the change took approximately two and a half hours. Most of the plan was followed as expected and without problems but minor challenges were faced due to a lack of clarity and misconceptions during the planning phase.

Notably, the following steps caused complications during development:

- Steps 7 and 10: The package in which *VerticalAsciiKalahView* and *VerticalAsciiUtil* should be added to was not specified. They were added to the 'ascii' package which also contains *AsciiKalahView* and *AsciiUtil*.
- Step 13: *printHorizontalBorder()* and *printDivider()* should also have been overridden to call *VerticalAsciiUtil*'s implementations of *horizontalBorder()* and *divider()* rather than those of *AsciiUtil*. This error occurred due to an incorrect judgement of how polymorphism would impact the programme.
- Steps 14-16: The *Map* implementation was not specified. At first a *TreeMap* was used and later, this was switched to a *HashMap* for improved performance.
- Steps 14-19: The total number of players was needed to be passed to the util methods so that the util methods would have access to the termination point when looping through player numbers.
- Step 18-19: The *pitsForPlayers()* method could not return a *String* containing line breaks (i.e. '\n'). Therefore, *printPitsForPlayers()* in *VerticalAsciiKalahView* was changed to loop through each house number and call *pitsForPlayers()*, renamed to *rowOfPits(),* in the *VerticalAsciiUtil* class. *rowOfPits()* then was only responsible for returning a string containing only 1 line. *rowOfPits()* also needed additional parameters such as, the number of players (*playerCount*), the row it was formatting (*houseIndex*) and the total number of houses (*houseCount*).

In addition, after the new feature was implemented, the code was refactored to improve maintainability and code readability. For example, a helper method and a new constant (*private static final* field) were introduced in *VerticalAsciiUtil*. The access modifier of two fields that were changed in step 5 were reverted to *private*.

V. Conclusion

Although there were a small number of complications during the execution of the plan, the initial Kalah design allowed for the new feature to be implemented relatively seamlessly. This is largely due to the use of the MVC pattern as well as conformance with the Open/Closed principle, as identified in the third assignment, which allowed for the view of the Kalah board to change through extension and not modification. Namely, only two classes were needed to be introduced, and changes to the existing implementation only involved changing the access modifiers of some methods and fields as well as initializing a *VerticalAsciiKalahView* object in the *Kalah* class. The relatively low amount of time taken to implement the feature, as well as near-seamless execution of the plan are evidence that the design of the original implementation was significantly changeable and was correctly identified in the third assignment with respect to the newly required functionality.

Acknowledgment

I would like to thank Dr. Ewan Tempero for lecturing the SOFTENG701 course and dedicating their time and energy to ensure that their students are actively engaged in learning about software development. I would also like to thank any teaching assistants who are involved in the marking of this assignment