Quiz-2 Report

IF184301_OOP(E)

members of the group:

- Irsyad Fikriansyah Ramadhan (5025211149)
- Moh. Adib Syambudi (5025211017)
- Muhammad Zikri Ramadhan (5025211085)

[1] Prototypes and design [25 points]

- For PredictivePrototype, first we add the needed methods such as : wordToSignature, signatureToWords, and isValidWord (for checking if a string contains a non alphabet character).
- For wordToSignature, we traverse the string char by char and append to the new StringBuffer the corresponding signature number for each alphabet character. Then return the BufferString as a String
- For signatureToWords, we traverse the file “words” and change the read line into its signature number. If the read line signature number is equal to the signature parameter, we add it into the set then return the set if the already read all the lines in the file “words”. To shorten the time, we use the function isValidWord. If a word contains a non alphabetical character, we skip calling wordToSignature for that line of String.
- Why use StringBuffer instead of String? A string is immutable whereas StringBuffer is mutable, which means, every change is done to String, it is actually making a new string that is stored on another memory. therefore, it will be more cost-efficient for the memory if we use StringBuffer

  Source : https://www.youtube.com/watch?v=_drNcLWgUlE
- Why not store the dictionary in your Java program? First of all, it is not a good practice because it will be burdensome if we wanted to add/remove/edit the dictionary if it is stored in the code itself. Second, it will be hard to see the code giving the number of words that are there inside the dictionary. That is why it is stored on a separate file and if we need it, we can always access it from the code side.

[2] Storing and searching a dictionary [25 points]

- The second part of the task is searching and sorting the dictionary into an arraylist. List contains the objects which are filled by the pair of signature and word. First step we make this object namely WordSig. The field access of word and signature is private. Object builded by constructor with parameter word and signature. We also make a getter method for accessing the attribute because of the private keyword. The searching algorithm using binary search for a more efficient running process. This algorithm can be used only on sorted lists. Sorting list of objects using implementation comparable class for consequent this implementation, we will override compareto method as pivot of sorting based on numerical order of signature. This method returns the number with type BigInteger as error handling of constraint signature.
- The next step is build DictonaryListImpl class. This class consisting arraylist of object wordsig and some of methods include constructor. The constructor parameter is file, will be get from word.txt. Building constructor we exert try and catch keyword as error handling that will send message when file doesn't exist. First we create scanner object for input file that we received from file. Next, we iterate addition of list member until next line of input. Every of input stored in variable line. Line evaluated by isValidWord method, if it is false we go on next line. After this process, arraylist sort using collection sort method. Last step of creating our constructor is close the file.
- The important method from DictionaryListImp class is signatureToWord, convert the signature into word form. First we need two integer variable and set to store signature that we search using binary search collection. The integer variable intended to reserve the return of binarysearch method which return integer value index position of signature on list. This variable make to two form, first form for moving backward traversal and second one for moving forward traversal. Next process is iterate addition of list base on index with same value with parameter. We also make two iteration process as the purpose of two variable before. There is several method like getter of arraylist, isValidWord, wordTosignature and printlist. wordTosignature and isvalidword we build base on first section. The pritnlist method used for print all member of list which we get from file word.txt.

- Testing the program using maintestlist . This class contains an object dictionarylistimpl as prototype for test. Running process of this class is to print the result of the searching signature “4663”. Output : [hood, ione, ioof, good, hond, inne, gond, hone, hoof, gone, goof, home, gome]. Based on our output, we assume the program is running properly
- We answer the problem section 2 with Sigs2WordList. The first movement we build object like the testing class. Now we use user input from console applying scanner object. We also create error handling to control the input process. The input variable split into array of string and this array utilize for argument of signaturetoword. We loop this procces until last member of string input on signaturetoword. Testing use input “4663 43556 96753 69 6263 47” and output is
- 4663 : [hood, ione, ioof, good, hond, inne, gond, hone, hoof, gone, goof, home, gome]
- 43556 : [hello, gekko]
- 96753 : [world, yorke]
- 69 : [ow, nw, ox, mw, oy, ny, mx, oz, nz, my]
- 6263 : [name, mane, nane, mand, oboe, nand, mame]
- 47 : [ip, iq, hp, ir, hq, gp, hr, gq, is, hs, gr, gs]

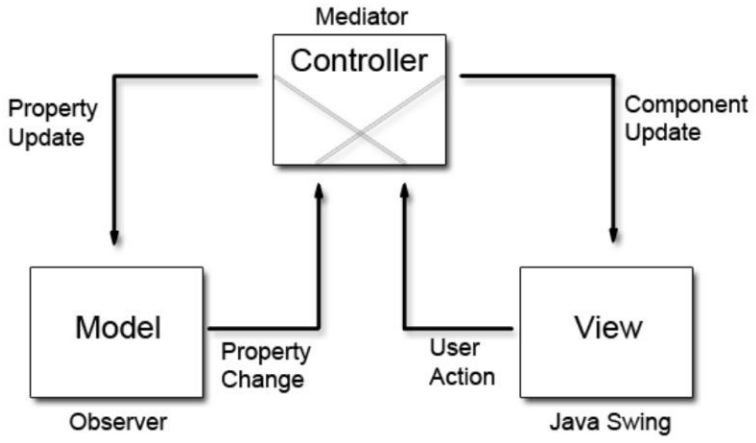
[3] More efficiency and prefix-matching [25 points]

- For the DictionaryMapImpl and DictionarytreeImpl, firstly we always check whether the word is a valid word or not using the isValidWord.
- Then for the DictionaryMapImpl, for the beginning, we read the words.txt file. Then using the wordToSignature method, we convert the word to its corresponding signature. To convert the word to its signature, we need to change all of its letters to lowercase. After that, we iterate all the characters in the word and change it to its corresponding number in the keypad. a, b, and c will be converted to 2, the letters d, e, and f to 3, and so on. Then we add the converted number to the new string. Then we store the words and the signature to the map that we already created before.
- For the signatureToWords method, we take the signature and we’ll return the string set which contains all the words with the same signature that we have stored in the Map before.
- For the DictionaryTreeImpl, first, we read the words.txt file. Then we also check if the word that we get is a valid word or not. Our tree contains a root and 8 children

nodes, each note for each number on the keypad, which is 2, 3, 4, and so on. To get the set of words from a signature, we go through the tree, starting from the root, and go through its children as we iterate each character in our signature string. Take an example as we are looking for the words for 4663. First, we go from the root to children number 4, then using iteration, we go to its children number 6, then its children number 6, and lastly 3. Then we take the set of strings that contains the words for 4663.

[4] Graphical user interface [25 points]

- Implementing *Model View Controller* (MVC) for this section
- The Model performs all the methods needed and that is it. It doesn't know the View exists. Inside the KeypadModel.java there are press, addChar, getCurrentWord, newWord, delChar, changeCurrentWord, and getResponse.
- The View only job is to display what the user sees. It performs no calculations, but instead passes information entered by the user to whomever needs it. Inside the KeypadView.java there are all the components for the GUI parts. Firstly we initiate a panel which contains all the components like JButton and JTextArea. Inside KeypadPanel.java also makes all the positioning for the button and the textArea. After making the panel variable, we make the JFrame. Then, making all the buttonListener methods for every button. Lastly, we make the setTextArea to update the JTextArea
- The Controller coordinates interactions between the View and Model



Job Descriptions and the Percentage (distribution) of Work :

- Irsyad Fikriansyah Ramadhan 50%
    - Section [1] Prototypes and design [25 points]
    - Section [4] Graphical user interface [25 points] :
        - making the whole code
    - Helping other members
- Moh. Adib Syambudi 25%
    - Section [2] Storing and searching a dictionary [25 points]
    - Section [4] Graphical user interface [25 points] :
        - Helping writing Report
    - Helping other members
- Muhammad Zikri Ramadhan 25%
    - Section [3] More efficiency and prefix-matching [25 points]
    - Section [4] Graphical user interface [25 points] :
        - Helping pitching ideas
    - Helping other members