BINUS UNIVERSITY
BINUS INTERNATIONAL

Assignment Cover Letter

(Individual Work)

Student Information :

Given Name	**: Aric**	**Student ID Number**	**: 2301890314**
Surname	**: Hernando**		
Course Code	**: COMP6510**	**Course Name**	**: Programming Languages**
Class	**: L2AC**	**Name of Lecturer**	**: Jude Joseph Lamug Martinez**
Major	**: Computer Science**		
Title of Assignment	**: Japanese Typing Game**		
Type of Assignment	**: Final Project**		
Due Date	**: 20 - 6 - 2020**	**Submission Date**	**: 20 - 6 - 2020**

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism / Cheating

Binus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept and consent to Binus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student :

Aric Hernando

TABLE OF CONTENTS

JAPANESE TYPING GAME

I. Project Specification

This program will be a typing game which will be based on the Japanese language. The typing game will be used for the learning Japanese, with the secondary objective of improving the user's typing speed in Japanese language.

The program should display a kanji, its meaning, and the typing prompt, which will be in alphabet. As it needs to help the user learn new words, the application needs to initialize a vocabulary based on a csv file. As it is a typing game, it will need to measure the user's typing speed, key per minute is typically used as words in Japanese have different length than the standard average of 5 letter per word.

As the csv file that was used didn't contain the romaji (romanized Japanese word), the program needs to be able to convert the given hiragana / katakana to romaji.

The program will be displayed using the java library swing. This library will be used to show and update all the things that is needed to display

By showing a vocabulary entry at a time, this program aims to improve the user's ability in reading and typing Japanese.

II. Solution Design

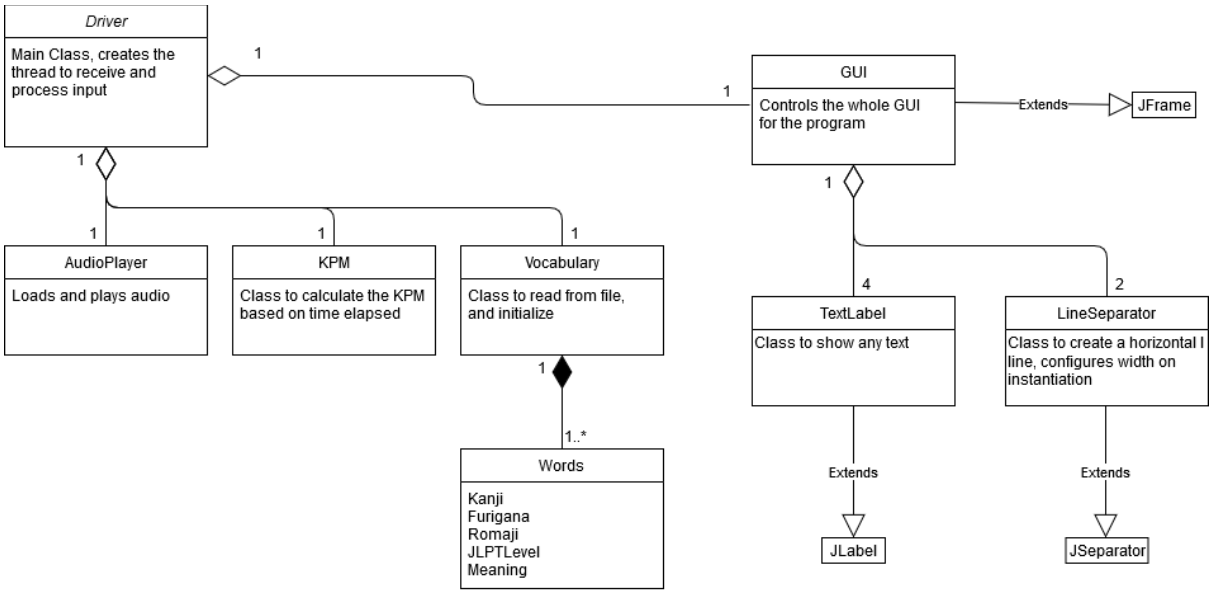
The program will start by initializing all of the objects that is needed for the program to run. Objects such as KPM, audio, vocabulary, and the GUI. The KPM will be used to measure the user's typing speed in keys per minute. It will measure this by dividing the total typed character by time elapsed and multiplying it by 60. The audio will be used to play sound effects.

The vocabulary is going to be used to store all the information regarding Japanese words. The information will first be initialized by reading from a csv file, containing the kanji, furigana (hiragana / katakana version of the word), and its meaning. The information will then be stored in an ArrayList of Word objects, which will contain all the information of a certain word.

After all of the preparation is done, the program will create the main user interface of the game.

The user interface will first present the user with simple instructions to play, after the player types 'type this to start', the game will start giving the player words to type. On the main file, it will pop a word from the vocabulary ArrayList, and then updates the furigana, kanji, meaning, and the romaji. The romaji is acquired through the use of a converter object, which will convert hiragana or katakana into an alphabet form. It will then format the color and updates the romaji display of the GUI. When the user has finished typing a word, it will then update the word in display.

IIIa. Class Diagram



IIIb. Code Explanation

- Word.java

Class to store the fields for a word.

```
public class Word {
    private String JLPTLevel;
    private String kanji;
    private String furi;
    private String meanings;
```

The rest of the class are just setters, getters, and the constructor.

- Vocabulary.java

This class will read from a csv, and then store all of the information from that csv in an ArrayList of Word objects

```
private ArrayList<Word> Words = new ArrayList<Word>();
```

The Word ArrayList

The constructor for the class will accept a directory as an argument, it will then create a BufferedReader object, which will take in a FileReader object, constructed using the directory that's been given.

```
Vocabulary(String dir){
        try {
            BufferedReader reader = new BufferedReader(new FileReader(dir));
```

It will then skip the first line, as that contains data that is not relevant in the vocabulary.

```
String currentLine = reader.readLine();
```

After that, the code will iterate through the whole file until it reaches end of file, in one iteration, the loop will assign several variables, and these variables will then be used to append a new Word to the Words ArrayList.

```
while((currentLine = reader.readLine()) != null) {
                // Splits the line into primitive array, and initialize severa
l variables
                // To be used later to create a new word object
                String arr[] = currentLine.split(",");
                String JLPTLvl = arr[0];
                String kanji = arr[1];
                String furi = arr[2];
                String meanings = "";

                // Iterates through the rest of the element in the 'arr' array
, inputting them into the meaning string
                for(int i = 3; i < arr.length; i++) {
                    if(i == 3) {
                        meanings += arr[i].replace("\"", "");

                    } else {
                        meanings += ", " + arr[i].replace("\"", "");
                    }
                }
                Words.add(new Word(JLPTLvl,kanji,furi,meanings));
            }
```

```
            reader.close();
```

While creating the meanings string, it will also remove all of the quotation mark from the string.

It will also catch an exception for when the file that is supposed to be read is not found.

```
catch(IOException e) {
            System.out.println("Error occured, file not found");
            System.exit(0);
        }
```

This class also has a function to return a single word at an index, while also removing it so from the ArrayList. This function will later be used to get new words that will be fed into the GUI.

```
public Word getWordAt(int index) {
        return Words.remove(index);
    }
```

The function size below this function will give back the total element of the ArrayList, this will later be used to generate the random index. By multiplying the Math.random function with the size of the ArrayList, you can generate an index between 0 and the total size of the Words ArrayList.

- KPM.java

```
public class KPM {
    private double timeElapsed;
    private int keyTyped;
    private double startTime;
```

This class KPM is the class that will be responsible for updating the key per minute based on these three variables. It will first do this by assigning the default values of these fields to 0.

```
public KPM() {
        this.startTime = 0;
        this.timeElapsed = 0L;
        this.keyTyped = 0;
    }
```

When the program starts its main loop, it will set a startTime for this class. The timeElapsed will then be calculated by subtracting timeElapsed by startTime, and then dividing it by a thousand, effectively calculating total time that has past since the main loop started, in second.

```
public void setStartTime(long startTime) {
        this.startTime = startTime;
    }
```

```
public void setTimeElapsed(double timeElapsed) {
        // Converts time from millisecond to second
        this.timeElapsed = ((double) (timeElapsed - startTime)) / 1000;
    }
```

The end result from all of these fields are the KPM, that can be obtained by dividing the total key that has been typed, and dividing it by total time. From that, we can multiply it by sixty to get the key per minute.

- AudioPlayer.java

This class will accept a directory string as it's constructor argument. It will then use this to create a clip object to load and to play audio.

```
public class AudioPlayer {
    void playAudio(String musicDir) {
        try {
            File musicPath = new File(musicDir);

            if(musicPath.exists()) {
                AudioInputStream audioInput = AudioSystem.getAudioInputStream(
musicPath);
                Clip clip = AudioSystem.getClip();
                clip.open(audioInput);
                clip.start();
            }
            else {
                System.out.println("File not found");
            }

        }
        catch(Exception ex) {
            System.out.println("Error occured");
        }
    }
```

```
}
```

- SyllabaryToRomaji.java

This class will start by creating a HashMap containing a key-value pair of Character and String. This will essentially be the alphabet for the transliteration of Japanese to English. This HashMap will be initialized on the constructor for this class.

```
public class SyllabaryToRomaji {
    HashMap<Character,String> syllabary = new HashMap<Character,String>();

    // Manual input of the hiragana set into the dictionary
    SyllabaryToRomaji(){…
```

The main function of the class is the convert function. This function will detect whether the kana of the word is comprised of katakana or hiragana. It does this by checking the first letter of the string as a number.

```
//  Accepts katakana or hiragana, automatically detects which one
    public String convert(String JPText) {
        // If katakana
        if(JPText.charAt(0) > '\u309F') {
            return convertKata(JPText);
        } else {
            return convertHira(JPText);
        }
    }
```

- convertHira.java

The function convertHira, will accept a hiragana string input. This function will use the syllabary HashMap, and then apply the transliteration rule from Japanese to English to create a romaji text.

```
public String convertHira(String hiragana){
        String newString = "";
        ArrayList<Character> specialSyll = new ArrayList<Character>();
```

```
        // Adds several special symbols to the list, these symbols can be pron
ounced differently based on the letters before it.
        specialSyll.add('っ');
        specialSyll.add('ょ');
        specialSyll.add('ゅ');
        specialSyll.add('ゃ');
        specialSyll.add('ぃ');
        specialSyll.add('ー');

        // Loops through the hiragana
        for(int i = 0; i < hiragana.length(); i++) {
            if(specialSyll.contains(hiragana.charAt(i))) {
                if(hiragana.charAt(i)  == 'っ') {
                    // Gets the first letter of the next char
                    newString += syllabary.get(hiragana.charAt(i + 1)).charAt(
0);
                }
                else if(hiragana.charAt(i)  == 'ー'){
                    newString += newString.charAt(newString.length() - 1);
                }
                else{
                    if(hiragana.charAt(i  - 1) == 'ち
' || hiragana.charAt(i  - 1) == 'し' || hiragana.charAt(i  - 1) == 'じ')
                        newString = newString.substring(0,newString.length() -
 1) + syllabary.get(hiragana.charAt(i)).charAt(1);
                    else
                        newString = newString.substring(0,newString.length() -
 1) + syllabary.get(hiragana.charAt(i));
                }
            }
            else
                newString += syllabary.get(hiragana.charAt(i));
        }
        return newString;
    }
```

- convertKata.java

  This function accepts a katakana string as an argument.

  This function will use the convertHira function. Since convertHira will translate from hiragana to romaji, the function convertKata will first convert the katakana to hiragana, and reuse the convertHira function.

```
public String convertKata(String kata){
        String newString = convertHira(convertKataToHira(kata));
        return newString;
```

```
    }
```

This function converts katakana to hiragana. It does this by subtracting the char by an integer, effectively offsetting it by that amount, and then append that letter to a new string.

```
public String convertKataToHira(String kata) {
        String hiragana = "";
        for(int i = 0; i < kata.length(); i++) {
            if(kata.charAt(i) == 'ー') {
                hiragana += kata.charAt(i);
            } else {
                hiragana += ((char)(kata.charAt(i) - 96));
            }
        }
        return hiragana;
    }
```

- TextLabel.java

This class exist for the purpose of showing any type of text on the GUI. The class will automatically configure the appearances of the text. This class is a subclass from the JLabel class.

```
public class TextLabel extends JLabel {
    TextLabel(){
        super();
        setFont(null);
        setForeground(null);
        setAlignmentX(CENTER_ALIGNMENT);
    }
    TextLabel(Color c,String s){
        super(s);
        setFont(null);
        setForeground(c);
        setAlignmentX(CENTER_ALIGNMENT);
    }
    TextLabel(Color c,String s, int align){
        super(s,align);
        setFont(null);
        setForeground(c);
        setAlignmentX(CENTER_ALIGNMENT);
    }
}
```

It will set the font to null so that rather than configuring it one by one, you can set the font in the container (JPanel) for all of the TextLabels.

- SeparatorLine.java

This is a class that extends from JSeparator. This class is made to create a divider line between the elements in the GUI. It configures it so that it's maximum size does not stretch beyond what it should display.

```
import java.awt.Dimension;
import javax.swing.JSeparator;
import javax.swing.SwingConstants;

public class SeparatorLine extends JSeparator{
    SeparatorLine(){
        super();
        setOrientation(SwingConstants.HORIZONTAL);
        Dimension d = getPreferredSize();
        d.width = getMaximumSize().width;
        setMaximumSize(d);
    }
}
```

- GUI.java

This is the class that will manage all of the GUI for the game. This class will first start by creating several TextLabel objects. This objects will be filled with text from a word.

The first time this class is created, these texts will be filled with simple instruction on how to use the program.

```
// Initializes all of the text field
    private TextLabel KPM = new TextLabel(Color.white,"Japanese Typing Game");
    private TextLabel furigana = new TextLabel(Color.white,"");
    private TextLabel kanji = new TextLabel(Color.white,"");
    private TextLabel meaning = new TextLabel(Color.white,"Press 'esc' to exit
");
```

```
    private TextLabel romajiDisplay = new TextLabel(new Color(100,100,100),rom
ajiText);
```

On the constructor, this class will add every one of the objects into the container JPanel. It will also use the SeparatorLine class to create the line between the UI elements.

```
GUI(String title) {
        setTitle(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JPanel container = new JPanel();

        BoxLayout bl = new BoxLayout(container, BoxLayout.Y_AXIS);
        container.setLayout(bl);
        setSize(1024,768);

        container.setBackground(new Color(32,32,32));
        container.setFont(new Font("serif", Font.PLAIN, 36));
        kanji.setFont(new Font("serif", Font.PLAIN, 64));
        romajiDisplay.setFont(new Font("serif", Font.PLAIN, 64));

        // Creates an invisible box to create padding
        // Box height is determined by its total height
        container.add(Box.createRigidArea(new Dimension(0,(int) (getHeight() *
 0.2))));

        container.add(KPM);
        container.add(new SeparatorLine());
        container.add(furigana);
        container.add(kanji);
        container.add(meaning);
        container.add(new SeparatorLine());

        container.add(Box.createRigidArea(new Dimension(0,(int) (getHeight() *
 0.05))));

        container.add(romajiDisplay);

        add(container);
        setResizable(false);
        setVisible(true);
    }
```

On the updateRomajiDisplay function, it will format the text so that the color of the first letter will be different than the rest. The function does this by using inline css.

```
public void updateRomajiDisplay() {
        // Checks if the text is less than one word so that it doesn't cause I
ndexOutOfBounds
        if(this.romajiText.length() <= 1) {
            romajiDisplay.setText("<html><p style ='color : white'>" + this.ro
majiText);
        } else {
            // Changes the color of the first letter, and leaves the rest as d
efault
            romajiDisplay.setText("<html>" +
                    "<span style ='color : white'>" + this.romajiText.charAt(0
)+"</span>" + this.romajiText.substring(1) +
                    "</html>");
            // Re-set the alignment as html will wrap to the left
            romajiDisplay.setHorizontalAlignment(SwingConstants.CENTER);
        }
    }
```

It would also reset the horizontal alignment since the html automatically wraps the text to the left.

The function setWord accepts a Word argument, and updates all of the text displayed in the GUI. It uses the SyllabaryToRomaji object to convert the word into romaji.

```
public void setWord(Word newWord) {
//      Changes the romajiText and display the newly changed text
        setRomaji(converter.convert(newWord.getFuri()));
        updateRomajiDisplay();
        setKanji(newWord.getKanji());
        setFuri(newWord.getFuri());
        setMeaning(newWord.getMeanings());
    }
```

- Driver.java

This is the main class. The main class will create a vocabulary, audio, KPM, and GUI object.

```
public class Driver{
    public static void main(String[] args){
```

```
        AudioPlayer audio = new AudioPlayer();
        KPM KPM = new KPM();

        // Initializes a list of word, with the directory as an argument
        Vocabulary vb = new Vocabulary("src/data.csv");

        // Creates the gui, and sets the title
        GUI ui = new GUI("Japanese Typing Game");
```

After that, it will add a new KeyListener to the GUI object. This key listener will create a new thread that will listen to the keyboard input of the user. Upon any keyboard touch, it will check if the input matches the first letter of the romajiText, when it does, shortens the romajiText by one letter. It also adds the typed key by one, later to be used for the KPM calculation. If the user presses the esc button, the program will terminate.

```
ui.addKeyListener(
            new KeyListener() {

            // A thread that will receive input
            boolean started = false;
            @Override
            public void keyPressed(KeyEvent key) {
                // Checks if the key typed is equal to the first character of
the romajiText
                if(Character.toLowerCase((char) key.getKeyCode()) == (int) ui.
getRomaji().charAt(0)) {
                    ui.setRomaji(ui.getRomaji().substring(1));
                    ui.updateRomajiDisplay();
                    if(started)  {
                        KPM.setKeyTyped(1);
                    }
                }
//  Terminates program if the user press escape
                else if(key.getKeyCode() == KeyEvent.VK_ESCAPE) {
                    System.exit(0);
                }
```

After the user has finished typing one word, it will then set the Boolean start to true, and start tracking the time and the keyTyped, effectively starting the calculation for the KPM. It will also play a jingle, after the start Boolean has been set true, it will then update the time and change the kpm every time the user has finished typing one word.

```
// If the word has been typed to completion, update the KPM and change the wor
d.
                if(ui.getRomaji().equals("")) {
                    audio.playAudio("src/Jingle.wav");
                    Word newWord = vb.getWordAt((int)(Math.random() * vb.size(
)));
                    ui.setWord(newWord);

                    if(!started) {
                        // Starts the time when the user has fully typed the "
type this to start"
                        KPM.setStartTime(System.currentTimeMillis());
                        ui.setKPM("0");
                        started = true;
                    } else {
                        // Time is added to the total time, which will be calc
ulated to get key per minute
                        KPM.setTimeElapsed(System.currentTimeMillis());
                        ui.setKPM(String.format("%.1f",KPM.getKPM()));
                    }
                }
```

IV. Source Code

Github link : https://github.com/Aric-prog/PLFinalProjectSem2

V. Working Program Evidence


KPM : 144.5
おてつだい
お手伝い
helper, assistant
otetsudai