**Introduction to Object-Oriented Programming.**

**Project assignment: Swedish-style crossword**

**Author: Andrei Kedich**

*Remark: the provided folder (archive) includes src folder from the existing crossword project, so the path to the txt file is created via concatenation with “src/”. For correct implementation, the new project should be created and the src folder should be extracted to the newly created folder. In case we want to run our program, for example from the command prompt, we should remove the string concatenation and leave only the name of the file (in Reading class). Path to the src folder in my system: C:\Users\kedic\eclipse-workspace\Crossword\src*

The final version of the crossword project includes the following packages and classes (the total number of classes is 8). Consequently, their functionality will be discussed step by step.

**1. package importdata**

**Class Reading**

The class that implements the reading and Import of all information from provided txt file. It includes several private methods (as temporary steps for derivation of data) and public methods (for free access from all other packages and classes). Such public methods: getRows() and getCols(). These two are used for further correct data import and construction of matrix for JButtons, Separation of columns and rows allow construction of non-square fields for a crossword from varying input data. Other public methods are readField() and readingLines(), which are responsible for reading information about different cells and associated text instructions. The first method returns a two-dimensional array of strings that will be used to construct the main field for the game and the second one returns an array for text instructions, where each line is a new string.

**2. package gui**

**Class CheckField**

This class creates a panel for the representation of cells for answer-input below the main playing field. It includes the n-number of JButtons (in GridLayout) based on the number of letters in the answer and a separate JButton to check the answer. To set the letter for the JButton we are using here the Keyboard object (with one input variable) and the public setButton method, which is accessed from the Keyboard listener. In addition, all letters are stored in the letter String array. We have here the action listener for the check button also, that access the method answerChecker() from CheckAnswer class. And the method getButtonLetter() transforms our array of strings into one array for further comparison in other classes.

**Class Field**

This class was written for the creation of the main playing field. It accesses the Reading class methods by the creation of Reading objects. Through the loop iterations, we are going across and down the 2d-array with cell-identification symbols (X, O, and with tips) and by a set of conditions assign buttons in a 2d-array of JButtons. Moreover, through the loop, we set the visual parameters for each button and link it with the action listener, that opens the keyboard. We are using here two types of action listeners: to create different Keyboard objects and access to different methods in the Keyboard class. We create a Keyboard object with two input variables for normal white and gray cells (variables represent the position in the cell-matrix), and the special Keyboard that in addition takes for input clue letter, and further access the Keyboard method keyboardFieldClue(). To conclude, this class includes protected methods to set the letter for the button field and the method to export the final JPanel.

**Class FinalView**

The next class is responsible for the creation of the final composition. We are creating here the object of each other class: List, Field, and CheckField. After we are importing their final panels via returnPanel() method in each class and all further work is focused on the composition of these panels into one JFrame within method run(). Also, this method includes the method getF(), which is used in Keyboard and CheckAnswer classes to locate the pop-upping JDialog in relation to the main frame.

**Class Keyboard**

This class is the most complicated one. It consists of 2 inner classes of Listeners, 3 constructors, and 3 methods with 10 private variables. All the methods are associated with the creation of different keyboards as JDialogs. The first constructor takes for input 2 variables, they reflect the position of the certain button that was clicked (for white and gray squares in the main field). After these variables are set the method keyboardField is used from the Field class, this method creates and returns a keyboard JDiolog and associated with the specific listener that sets the button in the field (setButton() from Field class) and closes JDialog. The next piece of the class works similarly, but also accepts the third variable, the clue letter. For the creation of the next keyboard we are using a supplementary method getRandomLetter(), that allows us to generate Random letters from the alphabet 2-dimensional String. That method is accessed from the method keyboardFieldClue, where also is being checked that 4 generated letters in addition to the clue are not identical. Another part of the method is similar. The final constructor accepts only one variable – the position of the letter in the answer field array of buttons. The associated method with this constructor is identical, the only difference is in Action Listeners, the letter should be assigned to the variable in CheckField class (access static setButton() method). Moreover, we access here the gerF() method with the creation of FinalView object for positioning our pop-up JDialog with the keyboard in the center of the frame.

**Class List**

The next class is responsible for the presentation of the text field with crossword instructions to the left of the main field. The main idea behind the class is to create a JPanel from the text lines in the provided txt file. Firstly we access method readingLines() from the Reading class and through further manipulations setting up styles and position for the instructions in the JTextPane (colors, bold or plain, spaces between rows). Final method exports the created JTextPanel: returnPanel().

**3. package checker**

**Class CheckAnswer**

The next package contains only one class whose purpose is checking the correctness of the user-provided answer when the button Check is clicked. It includes one method answerChecker that compares the inputted answer (by calling the method getButtonLetter() from CheckField class with the correct solution (by calling method getAnswerName() from OpenCrossword class). In return, it provides the JDialog with information about the correctness of the provided solution. For accessing the methods from other classes objects of CheckField, OpenCrossword, and FinalView (to access getF() to locate to JDialog relative to frame) are created. The object of this class and method answerChecker() is called from the CheckField Action Listener.

**4. package Opener**

**Class OpenCrossword**

It is the class that contains the configuration variables: the source to our txt file and the solution. The class includes the public getter method for freely accessing these variables from other classes, while exact variables are private. Finally, it contains the main method, where the object of Final View is created and method run() is called to create a graphic visualization of the game. The class was created to simplify the set of input parameters and run the application.

**Advantages, downsides, and limitations**

The main advantage is that this project is highly robust and is capable of puzzle creation from the txt file with the same data structure. It allows the creation of a non-square field, as it operates with rows and columns separately. In addition, the creation of the main field in the gui is dependent on the number of rows and columns, so even with the enlargement or reduction of the puzzle, JButtons will have the square form. The next advantage is the ease of changing input parameters, there is a special class only for setting the file name and correct solution, with further running the crossword. All logically differing parts of code are put in different classes: Keyboard (only creation of different Keyboards), FinalView (only the composition of panels), etc. Notwithstanding, the project has downsides. Probably, it was better to detach listener to other classes and to make them more powerful to reduce the number of intermediate methods, that allows changing the button letter. In addition, that could help us to get rid of several static variables and methods, that I usually use here to have free access to them. This approach should be enhanced in terms of object-oriented programming. Another disadvantage is the very massive Keyboard class, where parts of code are repeated. In the better solution, we should get rid of it, probably, the better idea here is to use the inheritance approach with one basic keyboard as a parent and more difficult child ones, that extend or limit the functionality. Finally, the application was checked on another platform (Mac) and several issues with the visualizations appeared: JButtons are not filled as on Windows. These difficulties were resolved by setting Metal Look And Feel that is supported by Windows and Mac either, and makes the graphic application identical for both operation systems.

During the completion of the project, several aspects have been revealed as time-consuming. Firstly, the operation of setting the button letter from the keyboard before the modifications the letter was assigned to the next clicked button. The problem was resolved by the creation of the JButton 2-dimensional array and storing all the buttons with the associated letters there. Another difficulty was associated with a further change of answer field after clicking the check button (it was the only one-time possibility to check your answer). The problem was solved by the method creation apart from the constructor to check the provided solution and return a JDialog.

Qr code

Description automatically generated

Figure 1. The final view of crossword application

Diagram

Description automatically generated

Figure 2. Simplified diagram of methods that accessed from different classes (only public and protected methods are included)