Swedish-Style Crossword Puzzle

Introduction to Object Oriented Programming

KU Leuven
Mohammad Sadil Khan

July 4, 2022

Contents

1 Introduction **1**

2 Code Design **1**
2.1 Class Descriptions 1
2.2 Class Relationship 2

3 Changes from initial implementation **3**

4 Strengths and Weaknesses **3**

1 Introduction

In this project, I have redesigned the Swedish-Style Crossword Puzzle in Java. It's a single player puzzle game which contains a playing field made of square or rectangular grid of white and black squares and a panel containing clues. I used Eclipse IDE to develop the game.
In Section 2, I have explained how the game has been developed with relationship between different classes. The changes in this redesign have been explained in Section 3. Section 4 contains the final conclusion with strengths and weaknesses of the code and possible extension of the game.

2 Code Design

2.1 Class Descriptions

1. Class **Main:** It's the main class for initiating the game. The main method of `Main` creates a `FileChooserUI` object. It takes place inside `invokeLater`.
2. Package **ui.filechooserUI**
   (a) Class **FileChooserUI:** Class for creating a welcome UI and starting the game. The `Start` button opens `Config` class which stores the `file` properties. After storing the relevant texts, `utility.InfoExtractor.TxtExactor` method is called.
   (b) Class **Config:** Class for creating the properties which contains default file path and solution phrase.
   (c) Interface **FileChooserUIInterface:** Interface used by `FileChooserUI` class. This interface contains all the components(buttons,panels,etc) used by `FileChooserUI` to create the welcome interface.
3. Package **ui.boardUI**
   (a) Class **GameUI:** Class for creating game GUI. The `utility.InfoExtractor.TxtExactor` method calls this class after extracting necessary information from the txt file.
   (b) Interface **GameUIInterface:** The interface used by GameUI. It contains three methods for creating three panels (Clues Panel, Square Panel and Solution Panel).
   (c) Class **CluesPanel:** Class for creating the clues panel UI which is on the left side of the game. It's called from `createCluesPanel` method in `GameUI` class.
   (d) Class **SquarePanel:** Class for creating the square board for playing. It's called from `createSquarePanel` method in `GameUI` class. It uses the square field information passed from `utility.InfoExtractor.TxtExactor` and uses `utility.TextBoxInfo` class to represent the squares.
   (e) Class **SolutionPanel:** Class for creating the solution section at the bottom part. It consists of some gray squares (same number of gray squares in the square board) and a check button. All the squares are of `utility.TextBoxInfo` object. It's called from `createSolutionPanel` method in `GameUI` class.
4. Package **functionality.keyboard**
   (a) Class **PlainKeyboard:** Class for creating on-screen keyboard. The constructor of the class takes the parameters - position of the keyboard on the screen(x, y). The `initializeUI` class customizes the frame and uses a `HashMap` variable `charButtonMap` which creates a map of letters in azerty style and a button representing the letter. Only normal and gray squares use

this class. Some of the variables and the methods are of *protected* type because of subclass access.

(b) Class **RandomKeyboard:** Class for creating an on-screen keyboard with only 5 keys enabled (4 random + 1 solution keyboard). It inherits the class `PlainKeyboard`. The constructor takes - position of the keyboard on the screen(x, y) and the `enabledKey` array of 5 characters for buttons that are going to enabled only.

(c) Enum **KeyboardType:** Enum class for representing type of keyboards. There are four constants. There are two enum constants - `PLAIN` for refering to `PlainKeyboard` and `RANDOM` for referring to `RandomKeyboard`.

5. Package **functionality.textbox**

   (a) Class **TextBox:** Abstract class for creating `JTextField` part of the `utility.JTextField` class. Each square boxes in the board are the object of `utility.JTextField` .`TextBox` class has a *protected* method `createUI` which creates a simple `JTextField`. This class also contains two abstract method `afterEffects` for customization of `JTextField` and `getTextboxType` for returning the enum constant according to the type of squares.

   (b) Class **PlainTextbox:** Class for creating normal squares. It's inherited from `TextBox` class.

   (c) Class **BlackTextbox:** Class for creating black squares. It's inherited from `TextBox` class.

   (d) Class **BlueTextbox:** Class for creating blue squares. It's inherited from `TextBox` class.

   (e) Class **GrayTextbox:** Class for creating gray squares. It's inherited from `TextBox` class.

   (f) Class **TextBoxInfo:** Class for represeting the squares. It contains `JTextField` object and a `TextBoxType` constant.

   (g) Enum **TextBoxType:** Enum class for representing type of textbox. There are four constants - `REGULAR, BLACK, BLUE, GRAY`.

6. Package **functionality.listener**
   This package contains classes for actions related to buttons and squares.

   (a) Class **TextBoxListener:** Class for binding a square with a keyboard so that whenever a square is clicked, a keyboard appears on the screen and the previous keyboard (if it exists) gets disposed. The constructor takes the array of all the squares of `utility.TextBoxInfo` object as parameter and assign the specific keyboard for each square that is clicked. For this,it uses a variable `tbToKeyboardMap` to map constants of `TextBoxType` to `KeyboardType`.

   (b) Class **SolutionListener:** Class for checking the solution input in the gray squares at the bottom part. When the `check` button is clicked, it's invoked. It implements `ActionListener` inferface from `java.awt.event` package. It displays messages after evaluating the user input solution.

7. Package **utility**

   (a) Class **InfoExtractor:** Extracts information from txt file and then calls the `gameUI` class for creating the game. I used `FileReader` to read a txt file and since reading from `FileReader` may be inefficient, I wrapped `BufferedReader` over `FileReader`.

   (b) Class **Util:** Miscellaneous utility functions.

2.2 Class Relationship

Figure 1 details the relationship between different classes in different packages.

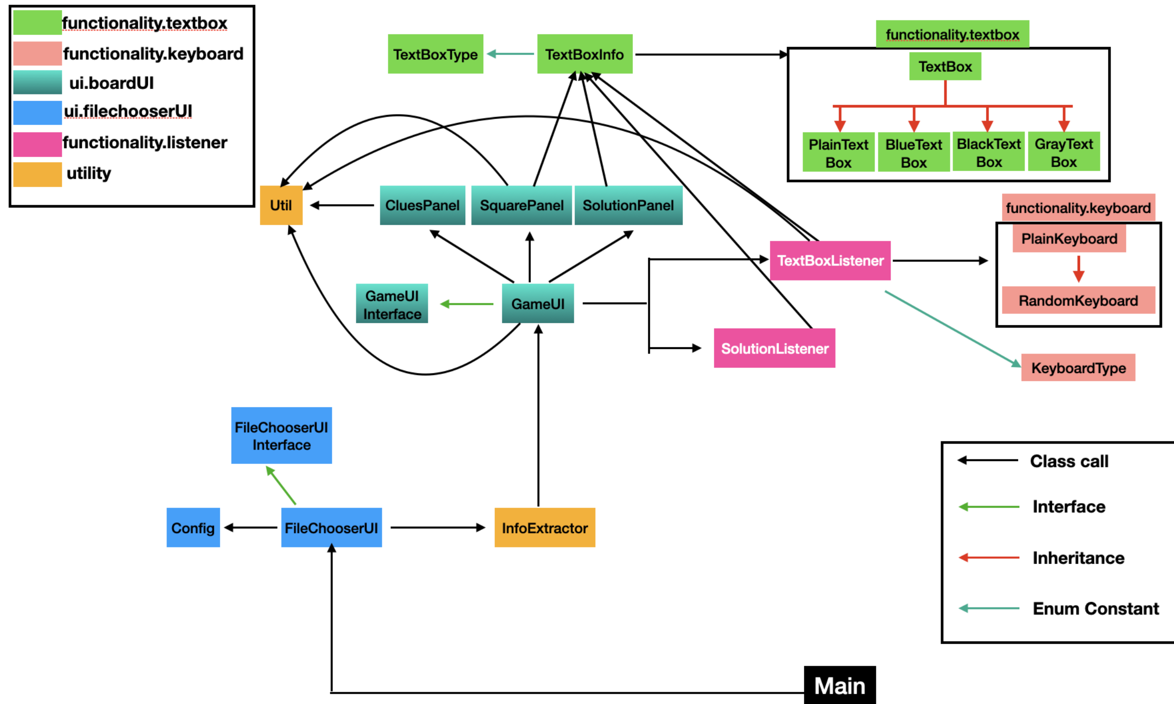


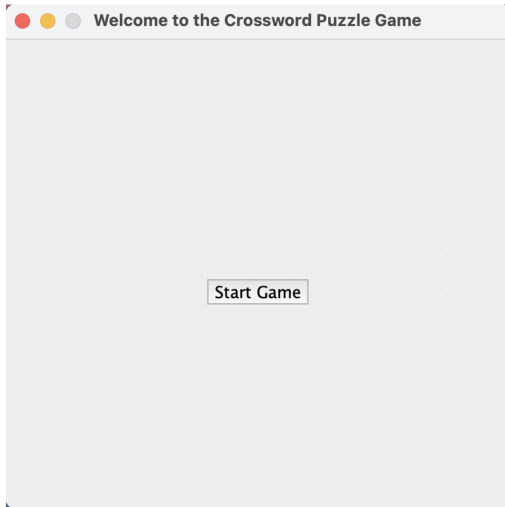
Figure 1: Class Relationship

3 Changes from initial implementation

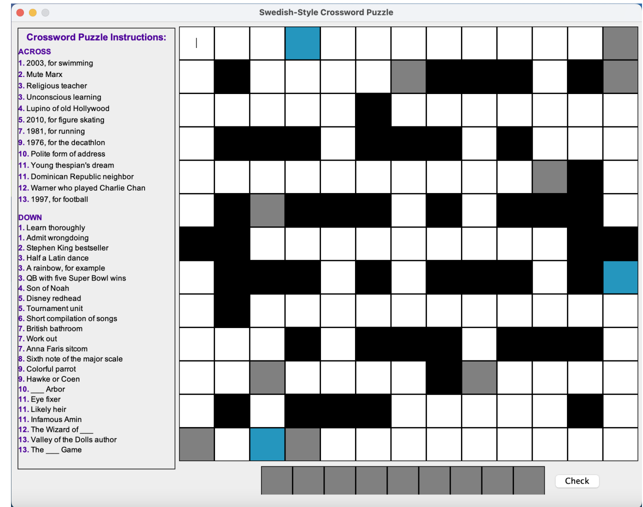
- **Use of Packages:** For this version, the codes are restructured into several packages depending upon the code task. Previous version had only 6 classes. The latest version contains 19 classes, 2 interfaces and 2 enum classes. Codes are stored into three packages - `functionality` (for actions,keyboard and squares), `ui` (for graphics related task) and `utility` (for utility functions). `functionality` package has three sub packages and `ui` has two subpackages.
- **Encapsulation:** For previous version, not a lot of effort was put to encapsulation. Several variables and methods had *public access*. For this version, I have put great effort into encapsulation and used *private* access as much as needed. Some variables and methods are *protected* for subclass access.
- **Inheritance:** Inheritance relationships are added for keyboards and squares. An abstract class is also used for squares which is inherited by all the type of squares. I didn't use any abstract class for keyboard since there are no difference between an abstract keyboard class and `PlainKeyboard` class. I have only used single inheritance although multiple inheritance is also possible for some cases. Polymorphism is also used for squares.
- **Simplified Txt Input:** I have simplified the txt input method for this version. I used java properties to store the txt file path and solution phrase and `FileChooserUI` gets access to this information by calling `Config` class.
- **Other:** Very few static methods and variables are used. InvokeLater is also used in `Main`. Codes are rewritten in a cleaner way. `actionPerformed` doesn't check the source of event now.

4 Strengths and Weaknesses

A lot of strengths of my code has already been mentioned is Section 3. The present version contains all the described characteristics of the game. It's divided into several organised packages for different types of functionalities and ui. Overall, I am very happy with the present version because of code simplicity



(a) Welcome Window



(b) Main Game Window

Figure 2: Screenshots of windows

and reusability. The classes can be further modified and extended to some other designs. The time and space complexity of the code is $O(NM)$, where N and M are the number of rows and columns respectively in the squareboard.
Some challenges during the retake of the project are

- I wanted to create an efficient keyboard which will disappear once any other box is clicked (not only when a button is clicked). At first, it would create another keyboard and I had to manually close multiple keyboards. I solved it in `TextBoxListener.java` by creating a `onscreenkeyboard` variable which keeps track of the current keyboard and once a new one is created the old one is removed.

- The hints for blue boxes were randomly generated everytime a user clicks on it. I fixed it by creating a `TextBoxInfo` class which represent a square box and added the set of keys to be enabled as a metadata information.

Although I am very much pleased with the current implementation, I think an area of improvement can be the structure of the code so that it can be modified easily. Also, the class dependency can be improved. `GameUI.java` should have been accessed from `Main.java` instead of `TxtExtractor.java`. In my present implementation, without complicating the code, I didn't find any easy way.
In conclusion, the second version of this code is very much focused on the correction of the shortcomings of the previous version with code restructure, simple GUI and reusable code. I am very happy that I implemented important OOP concepts like abstraction, inheritance, encapsulation and so on which were absent in the previous code. For future extension, dark mode and animation can be added with the option for users to save.