**Name and Username:** Arjun Kejriwal ([akejriwal24@amherst.edu](mailto:akejriwal24@amherst.edu))

**Instructions for running the program. Please explain exactly how to do this, i.e., what program should I run:**

You will need to run the main method in the “MainFrameRun” class. This will open a window in which the GUI will consist of three different buttons, each of which allows you to play one of three games.

**What platform did you use to develop this?**

**What IDE on what operating system or the command line on what operating system?**

I used the IntelliJ IDE on the Microsoft Windows 10 operating system.

**What version of Java are you using? (8,9,10,11,12,13?)**

I am using the Java 15 SDK (If it doesn’t work with Java 15, please try using Java 14)

Puzzle 1 is **Kakurasu**

My program **does** correctly solve the puzzle(s).

My program **does not** read this type of puzzle from a file. If it does, how is the file specified?

Puzzle 2 is **Skyscraper**

My program **does** correctly solve the puzzle(s).

My program **does** read this type of puzzle from a file.

**If it does, how is the file specified?**

I set up the Skyscraper puzzle by creating a scanner that reads a “.dat” file. The file consists of 21 strings. The first string is to specify the size of the square grid. The remaining 20 strings are used to fill in the clues for the puzzle (at the front and end of each row and the front and end of each column).

Puzzle 3 is **Dosun-Fuwari**

My program **does** not correctly solve the puzzle(s).

My program **does not** read this type of puzzle from a file. If it does, how is the file specified?

**GUI:**

**How do we switch from puzzle to puzzle?**

Each puzzle’s individual GUI has a button named “Different Puzzle”. Clicking this will bring the user back to the home screen/GUI, from where the user can choose a different puzzle.

**How does the user make a move?**

For the Kakurasu puzzle, the user needs to press the cells in the grid to make its value count for the row sum and column sum. Pressing the cell again will deselect and bring it to the default configuration and appearance. For the Skyscraper puzzle, the user needs to first click on a cell in the grid. Next, a drop-down list appears that contains all the possible values that the user can choose for the cell. Clicking a particular option will make that value appear on the button. For the Dosun-Fuwari puzzle, the user needs to first click on a cell in the grid. Next, a drop-down list appears that contains all the possible shapes that the user can choose for the cell. Clicking a particular option will make that shape appear on the button.

**What commands can the user do?**

Other than choosing which of the three puzzles to solve from the main menu and filling in the cells in each puzzle’s grid, the user can go through the instructions/rules of a puzzle by clicking on the “Instructions” button on the puzzle’s interface. This will take the user to a new frame that displays the instructions and has a button named “Back” that will take the user back to the puzzle without resetting the puzzle. The user can also check his/her answer using the “Check Answer” button on the puzzle’s interface. The user can also make the program solve the puzzle itself and display the solution on a new frame; he/she can make the program do this by clicking the button “Show Solution” on the puzzle’s interface. The user can also switch from puzzle to puzzle (play a different puzzle) by clicking on the “Different Puzzle” button on the puzzle’s interface. Clicking this will bring the user back to the home screen/GUI, from where the user can choose a different puzzle. The program has another special functionality that allows the user to fill in the cells in the puzzle’s grid through a file, which can be done by clicking the “Fill from File” button on the puzzle’s interface. This is useful since some users might prefer typing in the values for each cell into a file rather than choosing values from a list. There is also a sixth button in the Skyscraper puzzle interface named “Start Over” that allows the user to clear each cell in the grid and start the puzzle again from the scratch. This feature is not provided in the Kakurasu and Dosun-Fuwari puzzles since each cell in both puzzles’ grids has an option for clearing the cell to make it default.

**Programming Issues:**

**Where is the solver (please tell me what method(s) in what class(es)?)**

The code for the solver is in the label method of the “PuzzleFrame” class, which is an abstract class that extends JFrame. Each of the three puzzle set-up classes (“KakurasuFrame”, “SkyscaperFrame”, and “DosunFuwariFrame”) extend the “PuzzleFrame” abstract class. The solver method uses recursion to solve the puzzle on its own. The solver method also uses several abstract methods, each of which are defined separately in each of the three puzzle set-up classes (“KakurasuFrame”, “SkyscaperFrame”, and “DosunFuwariFrame”).

**Briefly describe the classes in your program, explaining what each does. This should be brief. It is okay to describe related classes together:**

**ActionListener1** – This class creates an instance of a puzzle set-up class and displays it on a new frame (after closing the main menu frame). This class implements ActionListener and the “MainFrame” class creates instances of this class.

**ActionListener2** – This class creates an instance of a puzzle’s instruction class and displays it on a new frame (after closing the puzzle frame). The actionPerformed method in this class is called whenever the “Instructions” button is clicked on each puzzle’s interface. This class implements ActionListener.

**ActionListener3** – This class first closes the previous frame and opens up a new frame. Several classes create instances of this class. This class implements ActionListener.

**ButtonToMenuListener** – This class creates the drop-down list when each cell is clicked in the Skyscraper and Dosun-Fuwari grids. This class implements ActionListener.

**ClearListener** – This class clears the cells and sets them to default for the Skyscraper game. The actionPerformed method in this class is called whenever the “Start Over” button is clicked in the Skyscraper interface. This class implements ActionListener.

**ConstraintsInterface** – This interface specifies a single method called “constraintsSatisfied” and is implemented by all the constraint classes of all the three puzzles.

**DesignPanel** – This class creates the design at the bottom of the main menu GUI (The circle and two rectangles) using the “paintComponent” method. This class extends the JPanel class.

**DosunFuwariButtons, SkyscaperButtons, and KakurasuButtons** – These classes contain specific methods and instance variables specific to the cells in their respective puzzle grids. These classes extend the JButton class. (Please note that the “SkyscaperButtons” class doesn’t have any methods or variables. However, it was created to bring some uniformity to the program, since the other two puzzles have specific classes that extend JButton).

**DosunFuwariFrame, SkyscaperFrame, and KakurasuFrame** – These classes not only set-up their respective puzzles but also define the abstract methods part of the “PuzzleFrame” abstract class. All three of these classes extend the “PuzzleFrame” abstract class and implement ActionListener.

**DosunFuwariInstructions, SkyscaperInstructions, and KakurasuInstructions** – These classes display their respective puzzle’s instructions and also include a “Back” button to return to an unchanged puzzle. All three classes extend the JFrame class.

**DosunFuwariPositionChecker and DosunFuwariSingularChecker** – These are the constraint classes of the DosunFuwari puzzle that check the two different types of constraints of the puzzle – each area must have exactly one white circle and one black circle and white circles must be at the top of each area while black circles must be at the bottom of each area. These classes implement the ConstraintsInterface.

**FileAddListener** – This class fills in a puzzle’s grid by reading strings from a file (using a scanner). The actionPerformed method in this class is called whenever the “Fill from File” button is clicked on each puzzle’s interface. This class implements ActionListener.

**KakurasuRowChecker and KakurasuColumnChecker** - These are the constraint classes of the DosunFuwari puzzle that check the two different types of constraints of the puzzle – each row should sum to the value at the end of the row and each column should sum to the value at the end of the column. These classes implement the ConstraintsInterface.

**KakurasuListener** – This class changes the appearance of a cell in the Kakurasu puzzle grid depending on whether it is selected or not. The actionPerformed method of this class is called whenever a cell in the Kakurasu grid is clicked. This class implements ActionListener.

**MainFrame** – This class sets up the main menu frame that the user first sees when he runs the program. This class extends the JFrame class.

**MainFrameRun** – This class contains the main method for the entire program. This is the class that you have to run in order to solve the three types of puzzles. It sets up an instance of the MainFrame and displays it on a separate window.

**MenuItemListener** – This class sets the appearance of a cell on the Skyscraper and Dosun-Fuwari grids based on the user’s selection from the drop-down list. Instances of this class are created in the ButtonToMenuListener class. This class implements ActionListener.

**PuzzleFrame** – This is an abstract class that contains the solver methods, which is called “label”. It also contains several abstract methods that are defined by the three puzzle set-up classes that extend it. This class extends the JFrame class.

**SkysrcaperRowChecker, SkyscraperColumnChecker, SkyscraperRowRepeatChecker, and SkyscraperColumnRepeatChecker** - These are the constraint classes of the Skyscraper puzzle that check the two different types of constraints of the puzzle – no numbers should repeat in each row and column and the specific row and column rules must be satisfied. These classes implement the ConstraintsInterface.

**SolveListener** – This class calls the label solver method and displays the solution of the puzzle for the user. The actionPerformed method in this class is called by pressing the “Show Solution” button in each puzzle’s interface. This class implements ActionListener.

**Briefly describe how you would add another kind of puzzle to your program (this should include what classes, and if relevant, what classes they must extend or interfaces they must implement):**

If I were to add a new puzzle to my program, I would first create a button to access it on the main menu frame. Next, I would create a new set-up class for that puzzle which would extend the PuzzleFrame abstract class, implement ActionListener, and define all the abstract methods in it. I would create new classes for the buttons of the puzzle and for the instructions of the puzzle (these classes would extend JButton and JFrame respectively). I will also create different classes for the different constraints of the puzzle, and make each of these constraint classes implement ConstraintsInterface. I will also have to create a separate if condition body for this puzzle in most of the classes that implement ActionListener. I would probably set the puzzle up by reading from a file and create an appropriate layout.

**Tell me about anything special that you did that you would like me to notice. This should include anything major about the GUI that you did not describe above. It also should point out anything in the code that you think is particularly noteworthy. IF YOU DO NOT POINT IT OUT, I WILL PROBABLY NOT NOTICE IT!**

1. I used the paintComponent method to create a very interesting shape in the lowest panel on the main menu frame. It consists of a colorful circle that is formed through small circles on its perimeter. The panel also consists of a few round rectangles to add to the aesthetic look of the main menu frame.
2. Equipped with what I learned from the Exceptions lab, I used the try and catch structure for finding and reading “.dat” files in “SkyscaperFrame” and “FileAddListener”
3. I have made my program as abstract as possible – I only have a single solver method that works for all three puzzles, all of my classes that implement ActionListener are different, useful, and called several times, and my constraint classes make use of the benefits of instance variables and instance methods well to achieve abstraction.
4. To add to the aesthetic appeal of my program, I associated a single color to each of my three puzzles. Furthermore, I gave the main menu frame a colorful look to attract the user’s attention.
5. I used two new types of classes: JPopupMenu and JMenuItem
6. I also used the Graphics2D class in addition to the Graphics class so that I could call the setStroke method for changing the thickness of the outline of a shape.
7. I used the methods createHorizontalStrut and createVerticalStrut to space out the buttons in various frames.
8. I used the method JOptionPane.showMessageDialog to display a message as a dialog box as opposed to displaying the result on the same frame.

**SUBMIT EVERYTHING NEEDED TO COMPILE AND RUN YOUR PROGRAM. Be sure not to forget data files and image files. Do not submit anything that does not compile. Do not submit files that are not needed.**

**You will need to submit a zip file since the number of files you can submit is limited.**