CSCI 235 Spring 2020

Lab 12 **Apr. 28**

Connecting to a VNC display

You will use a lab account for a VNC display. The lab account already started a VNC session, so you don't need to do Step 3 or 4 of Lab 7's Task #1. Two steps that you need to do are:

1. Create a tunnel from your machine to a VNC display (Step 5 of Lab 7's Task #1).
    a. A cslab machine number and a display number will be provided to you.
    b. For example, if the given machine is **cslab10**, and the display number is **12**, the command for creating a tunnel to the display will be:
        - Mac Terminal:
          `ssh -L 5901:localhost:5912 your_loginID@cslab10.wheaton.edu`
        - Windows PuTTY:
          Host Name: `cslab10.wheaton.edu`
          For Connection, SSH, Tunnels: `5901` for the "Source port" and `localhost:5912` for the "Destination".
    c. The CS system will prompt for your (ID and) password as usual.

2. Start a VNC viewer on your machine (Step 6 of Lab 7's Task #1).
    a. The given display's VNC password will be provided. You need to use this password to connect to the display.

Tips:

- If you don't see the menu bar at the top of the CS desktop window, right-click the desktop screen to get Applications menu.
- When you are done with the desktop, close the desktop and connection by doing:
    - For TightVNC, use the X button at the top corner of the window.
    - For Mac, in Screen Sharing menu at the top, select Connection, and select Close.

===

Documentation and styling of your code will be part of your grade.

Create a new directory and cd to it. Then copy starting files from the course directory to your current directory.

```
$ cp /cslab/class/csci235/labs/lab12/* .
```

You should have two files, `FifteenPuzzle.java` and `PuzzlePiece.java`.

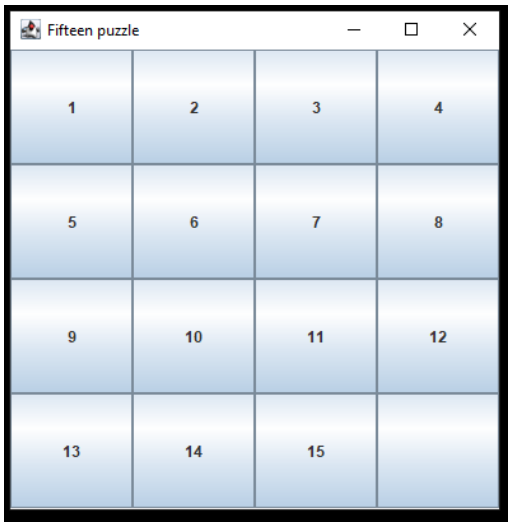
Fifteen Puzzle

A *Fifteen Puzzle* consists in a grid of square, same-sized tiles, with all but one of the grid positions filled. The object of the game is to rearrange the tiles by sliding the tiles one at a time until a certain configuration is reached. You will write a program that simulates a fifteen puzzle.

Open `FifteenPuzzle.java`, and inspect the code. Before moving to the next paragraph, **answer to the following questions** (Your answers will not be collected, but DO NOT skip the questions!):

1. How many buttons will be generated from the first nested for loop?
2. What is the action listener class? What is the type of `pieces`?
3. What arguments are passed to the action listener's constructor? How does the variable `counter` work?
4. How many listener objects are generated?

5. What class has the method `addNeighbor`? What is the argument to the method? What is its type?
6. See the second nested for loop, and answer to the following questions:
    a. When *i* is 0, and *j* is 0, what pieces are linked to `pieces[0][0]`?
    b. When *i* is 1, and *j* is 0, what pieces are linked to `pieces[1][0]`?
    c. When *i* is 2, and *j* is 2, what pieces are linked to `pieces[2][2]`?
    d. What is the maximum number of neighbors? What is the minimum number of neighbors? Why does this difference exist on the grid?
    e. See the below figure, which is the window generated by the complete FifteenPuzzle program.
        i. What does this second nested loop do?
        ii. The provided starting file needs to be edited for this complete window.



The code basically adds sixteen buttons to the window, numbered from 1 through 15 and then 0. Compile and run FifteenPuzzle, and see that the buttons are put right on top of each other: thus, you see only one of them. **Update the code so that you can see all of the 16 buttons arranged in a square** as seen in the figure.

- Hint: Resize the window, and use a layout manager.
- You will not see the empty button when running the current version of FifteenPuzzle. Instead, you will see a button with 0. The work for presenting the empty button should be done by the action listener class.

Each button stands for a tile or block in the puzzle. Fifteen of them will have numbers, and the 16th (numbered 0) will be blank. We won't actually move the buttons around; instead, we will change the numbers on the buttons to give the appearance of moving tiles. When a button is clicked, the following should happen:

- If it is blank, nothing should happen.
- If it is not blank and none of its neighbors is blank (neighbors meaning immediately above, below, left, or right; diagonals do not count), nothing should happen.
- If one of its neighbors is the blank one, then the blank neighbor should change to have the clicked button's number, and the clicked button should turn blank.

Task #1: Each button needs to have an action listener to perform one of the above behaviors. We are using an action listener class called `PuzzlePiece`, and your task is to complete this class.

- Open the file, and see the four instance variables. Why does each PuzzlePiece object need these four instance variables? **Document each variable** based on your understanding on the code.

Task #2: Complete the following methods next:

- `PuzzlePiece(JButton, int)`: This constructor tells the action listener what button it is attached to and the number in the order the buttons were made (1 to 15, with the last one numbered 0). It also takes care of setting the text on the corresponding button. You'll need to fix its body so that the button for 0 shows as blank. In addition, create or initialize the instance variables.
    - After completing the constructor, run the program. You should see the empty button instead of the button with 0.

- `addNeighbor(PuzzlePiece)`: When a button is clicked, it affects its neighbors. The action listener is not going to have direct access to the neighboring buttons, but it should have **references to the action listeners attached to those buttons** (You can perform some actions on the neighboring buttons through those references). How can we store this information for later reuse? Write the method body.
    - Hint: Find the code of FifteenPuzzle that calls `addNeighbor`. Whenever this method is called, the reference delivered through the formal parameter, `neighbor` should be stored in the array, `neighbors`. In addition, the instance variable, `numOfNeighbors` should be updated accordingly.

- `actionPerformed(ActionEvent)`: This method should implement the actions described above.
    - Hint: Determine if one of the neighbors is blank, and then make any necessary changes. Recall that the information about the neighbors has been stored in the array. Each element of the array (a neighbor) is in the type of PuzzePiece, so your code can directly access to the neighbor's instance variables (the four variables).
    - **Important:** When a PuzzlePiece's instance variable, `number` is updated, the corresponding text on the button should be also updated, and vice versa.
    - If you want, you can add a few extra helper methods that can be used by this method.

Run the FifteenPuzzle program, and see if it works. **Make sure to understand how all the programming components are related and work together in the program.**

Important: After you email your files to the TA, delete your lab folder for the next team. Your display will be available during the scheduled lab hours only.

What to submit

- Email `FifteenPuzzle.java` and `PuzzlePiece.java` to your TA:
    - 8:30 am session: to Drew at drew.smith@my.wheaton.edu
    - 1:15 pm session: to Brian at brian.drown@my.wheaton.edu
    - **In the subject line**, put "CSCI 235 Lab 12 (*the first initial of your first name and your last name*; *the first initial of the partner's first name and the last name*)"
        - For example, CSCI 235 Lab 12 (HKim; JSmith)