Brandeis University COSI 12B, Spring 2017 PA2

Due date: Sunday, Feb 5, 2017, at 11:59pm

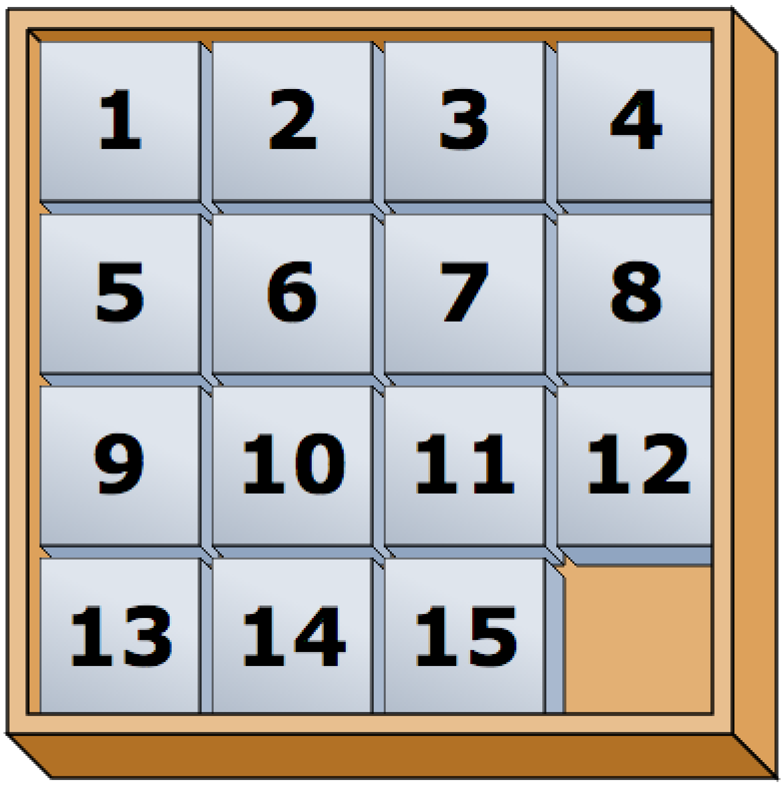
Learning objectives

In this assignment you will practice how to interact with an existing object. The assignment involves:

1. Understanding and using someone else's code.
2. Interaction with an object, using it's methods.
3. Implementation of additional functionality for an object.
4. Writing a human-computer interactive program.

Background information

The Game of 15, or 15-puzzle, involves 15 tiles that can slide around in a 4x4 box.



The image above shows a 15-puzzle in a solved position. The numbers are sorted in ascending order from left to right and then from top to bottom. A puzzle is only considered "solved" when it is in this exact state.

Consider the following board, representing an unsolved 15-puzzle:

C1	**C2**	**C3**	**C4**
1	2	3	4
5	6	7	8
9	10	11	12
13	14		15

Solving this puzzle is fairly simple, as only one tile is misplaced. In order to solve this puzzle, we simply slide the "15" tile to the left. We call this *moving left*. Similar to *moving left,* we define *moving right, moving down* and *moving up.*

In most cases, solving a puzzle will require multiple moves. For example, consider the following board:

C1	**C2**	**C3**	**C4**
1	2	3	4
5	6		7
9	10	11	8
13	14	15	12

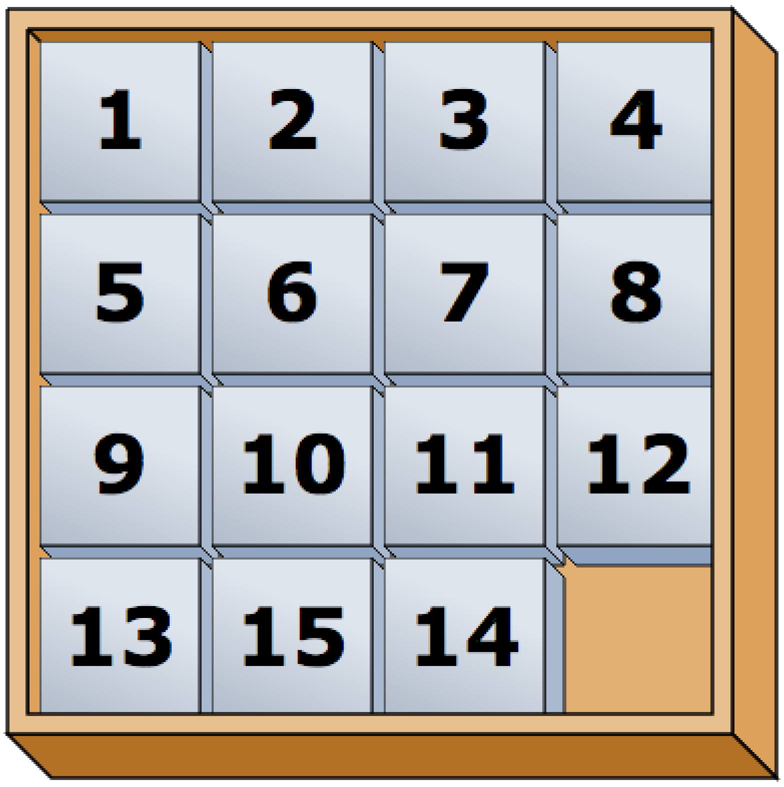
In order to solve the puzzle above, we can move the "7" tile to left, then move the "8" tile upwards, then move the "12" tile upwards.

Notice, that in general, we don't need to specify which tile is being moved, because there may be only a single tile that can be moved in a specific direction. Therefore, for the last example, the solution is: *move left, move up, move up.*

In the Game of Fifteen, there are four possible moves:

- `up`: slide a tile upwards into the blank space. This move cannot be performed if the blank spot is in the bottom row.
- `right`: slide a tile to the right into the blank space. This move cannot be performed if the blank spot is in the leftmost column.
- `down`: slide a tile downward into the blank space. This move cannot be performed if the blank spot is in the topmost row.
- `left`: slide a tile to the left into the blank space. This move cannot be performed if the blank spot is in the rightmost column.

One interesting property of the 15-Puzzle is that not all puzzle states can be solved. For example, consider the puzzle below:



No matter how many valid moves you make, you'll never be able to get the puzzle into a solved state. For a proof, see the Wikipedia page.

Your task

In this assignment, you'll be required to implement two "front ends" for the 15-puzzle. In the first part, you will implement an interactive program that allows a human player to play the game. In the second part, a computer will be the player.

We have provided you with a class called `GameOf15`, located in `GameOf15.java`, which contains all the code needed to store boards, generate solvable games, conduct slides, and test to see if the puzzle is solved. You may look at the code, but **you must not modify it**. The goal of this assignment is to give you experience working with objects provided by others.

The `GameOf15` class has several methods you will need:

- `GameOf15::moveUp()` will slide the piece below the blank space upwards.
- `GameOf15::moveLeft()` will slide the piece to the right of the blank space to the left.
- `GameOf15::moveDown()` will slide the piece above the blank space downwards.
- `GameOf15::moveRight()` will slide the piece to the left of the blank space to the right.
    - `moveUp`, `moveLeft`, `moveDown` and `moveRight` methods return *true* if the move was actually made. If the move cannot be performed, (for example, `moveDown`, when the blank piece is in the first row), these methods don't change the board and return *false*.
- `GameOf15::hasWon()` returns true if the puzzle has been solved, and false otherwise.
- `GameOf15::getValue(int row, int col)` will return the tile at the given row/column index of the board. The blank space is represented by zero.

- `GameOf15::toString()` will return a string representation of the board that you can display to the user.

Part 1: The Human Player

You should start by implementing the `HumanPlayer::play` method, located in `HumanPlayer.java`. In this method, you should create a new `Scanner` and continuously execute the following steps:

1. Print out the current game board, which can be done by executing `System.out.println(game)`. This takes advantage of the provided `toString()` method in the `GameOf15` class.
2. Prompt the user to enter an input with the following string: *"Enter a move: (l)eft, (u)p, (r)ight, (d)own, or (exit): "*.
   Valid inputs are one of the following, followed by the return key:
   - `l` - for left
   - `u` - for up
   - `r` - for right
   - `d` - for down
   - `exit` - for finishing the game
   - If the user enters invalid input, you should print the message: "Invalid input. Try again.", disregard the invalid input and prompt the user again.
     If the user enters `exit`, you shouldn't print anything and finish the game.
3. Execute the move indicated by the user.If the move cannot be performed, print *"No move was made. Try again."* and continue from the first step.
   If the game has been solved, print out the board, then print "You win!", and then exit the loop.
   Otherwise, continue to step 1.

Here's an example session:

```
1   2   3
5   6   7   4
9   10  11  8
13  14  15  12

Enter a move: (l)eft, (u)p, (r)ight, (d)own, or (exit): u
1   2   3   4
5   6   7
9   10  11  8
13  14  15  12

Enter a move: (l)eft, (u)p, (r)ight, (d)own, or (exit): l
No move was made. Try again.
1   2   3   4
5   6   7
9   10  11  8
13  14  15  12

Enter a move: (l)eft, (u)p, (r)ight, (d)own, or (exit): x
Invalid input. Try again.
1   2   3   4
5   6   7
9   10  11  8
```

```
13  14  15  12

Enter a move: (l)eft, (u)p, (r)ight, (d)own, or (exit): u
1   2   3   4
5   6   7   8
9   10  11
13  14  15  12

Enter a move: (l)eft, (u)p, (r)ight, (d)own, or (exit): u
1   2   3   4
5   6   7   8
9   10  11  12
13  14  15

You win!
```

Important notice: All your printouts (the *prompt*, *invalid input*, *no move was made* and *you win* messages) should be exactly as specified. As you can see from the example, the only blank lines are the ones after printing the board. This blank lines are part of the `GameOf15::toString()` method. You shouldn't print any other blank lines. Your code will be tested against expected printouts, so, make sure you print them correctly. Sample printouts will be provided in the test files for your convenience.

You should be able to run your code and play! You should also pass all of the tests in `HumanPlayerTest.java`.

Part 2: The Computer Player

For the second part of the assignment, you'll implement a simple method to solve a **relaxed version of the puzzle.** In the `ComputerPlayer::solveRelaxed` method, write code that moves the blank space from wherever it is to the bottom right corner and prints out the needed moves as a sequence of newline-separated `u`, `l`, `r`, and `d`s.

If the `solveRelaxed` method is given the following board:

C1	C2	C3	C4
1	2	3	4
5	6	7	8
10	13	11	12
9		14	15

Then the output of your code should be:

```
l
l
```

Performing two "left" moves will cause the blank space to be in the bottom right, which, for this assignment, will be considered acceptable. Please, **notice**, that although the blank space “moves” twice to the right, the corresponding *moves,* as defined before, are 2 *left moves*. In addition to printing out the needed moves, you should also perform the moves on the board using the appropriate methods and return the board.

You can run the provided `main` method to test your code on a random board. After you complete this task, you should pass the tests in `ComputerPlayerTest.java`.

Extra credit

Make your computer solve the puzzle! Implement `ComputerPlayer::solveReal` method to solve the entire puzzle. **Hint:** no solvable board requires more than 81 moves to solve. There are a few more hints in the comments above the method as well.

Testing

Like in the first assignment, we are providing you some **non-exhaustive** unit tests. You should test your code significantly more on your own. You are not required to make any changes to "HumanPlayerTest.java" and “ComputerPlayerTest.java” files, but you are highly recommended to do so and add your own tests. Pay attention to edge cases and invalid inputs.

Style guidelines

Please, look at the course website
http://cosi12b-s2017.s3-website-us-west-2.amazonaws.com/content/resources/10_programming_style.md/and follow the guidelines.

Submission

- You shouldn’t change the names of any provided file, class or method. (Obviously, you can add your own methods, if needed).
- You should export the Eclipse project as a zip file and name it with the following scheme:
  "\ + "\" + "_PA02" + ".zip".
  For example, Albert Einstein would submit: aeinstein_PA02.zip
- For late policy check the syllabus.

Grading

You will be graded on:

- 70% - The correctness of the implementation. Your code will be tested against our tests and your output match exactly the expected output. Programs that do not compile will not receive points for this part.
- 30% - your code will be reviewed by the TAs and graded on your 2 on 1 meetings. Your source code should follow the style guidelines as described above.