Understand the Problem

We are trying to create a GUI that will allow the user to unscramble a puzzle. The user will be presented with a split screen- half the screen will have the puzzle pieces that have yet to be placed, and the other half will have the puzzle completed thus far. Additionally, a separate GUI that shows the picture that the user is trying to copy should also be implemented, using multithreading.

We can break this down into several steps:

1. Design classes to represent a puzzle piece and a puzzle
2. Design a class that creates the basic puzzle board GUI
3. Design a class inherited from the Thread class that will take care of the second GUI containing the picture that the user is trying to copy.
4. Create the functionality in the game class so that the user can unscramble the puzzle

Phase 1- PuzzlePiece and Puzzle Classes

Create a class that represents a puzzle piece.

The PuzzlePiece class should contain the following data:

pieceID- a string representing the unique pieceID of each puzzle. The pieceID contains a number that represents the row of the piece, and a letter that represents the column, in the unscrambled puzzle. Should be private.

puzzleName- a String representing the name of the puzzle to which this piece belongs. Should be private.

The PuzzlePiece class should contain the following methods:

Default Constructor

PuzzlePiece(String pieceID, String puzzleNam)- argument taking constructor

Accessors for both data members

Mutators for both data members- make sure to convert puzzleName to lowercase.

generatePieceIcon()- Instance method that generates an icon for the PuzzlePiece that is the calling object. Note that the puzzle piece names follow the following format: pieceID.gif, so it is quite simple to generate an icon based on the file name. The folder that the puzzle piece images is in is simply puzzleName in lower case letters. This method returns an Icon.

toString()- creates a string containing the data for the puzzle piece.

equals(PuzzlePiece compPiece)- tests that two puzzle pieces are equal. Returns a boolean.

The Puzzle class should contain the following data:

static final int MAX_ROWS- should be set to 10

static final int MAX_COLUMNS- should be set to 10

numRows- an int representing the number of rows in the puzzle. Should be private.

numColumns- an int representing the number of columns in a puzzle. Should be private

puzzleName- the name of the puzzle. Should be private.

PuzzlePiece[][] puzzleArray- This private data member represents the actual puzzle. It is a 2D array of PuzzlePieces, that has the dimensions of numRows and numColumns.

The Puzzle class should contain the following methods:

Default Constructor

Puzzle(int rowsNum, int columnsNum, String puzzleNam)- parameter-taking constructor.

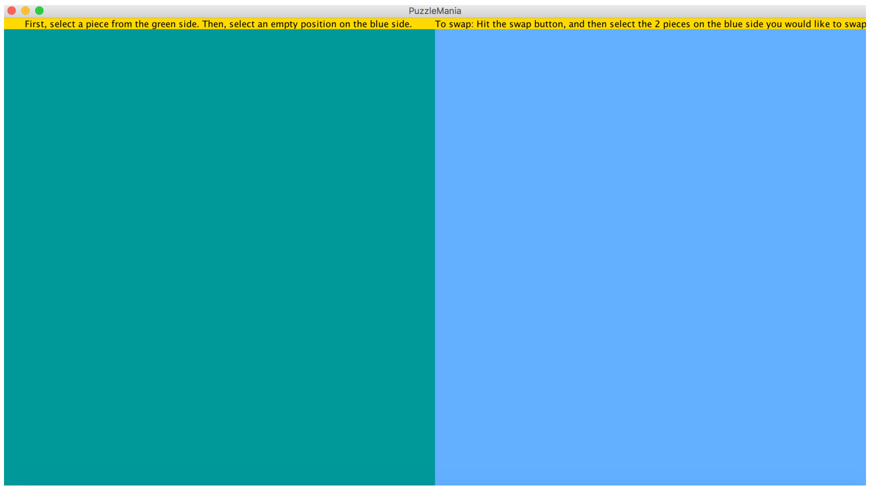
Accessors and mutators for numRows, numColumns, and PuzzleName- Make sure to convert puzzleName to lowercase.

fillWithDefault(): void- This method fills puzzleArray[][] with the default pieces. As mentioned above, the pieceID member of each puzzle piece contains the row number and column number.

Use a double loop and the private helper convertNumToAlpha() to fill the puzzleArray with the proper pieces to create a puzzle in order.
String convertNumToAlpha(int colNum)- This private helper takes a column number and converts it to its alphabet equivalent. Returns a StringUsed in fillWithDefault().
PuzzlePiece getPieceAt(int row, int col)- This method returns a copy of the puzzle piece at position PuzzleArray[row][col]. Should check that row and col are in range
setPieceAt(int row, int col, PuzzlePiece piece): boolean- This sets the piece at puzzleArray[row][col] to the piece that is passed as a parameter. If either row or col exceed numRows or numColumns, it returns false without setting the piece.
scramble(): void- This method scrambles the actual puzzle- not a copy of the puzzle. You will need a scrambling algorithm. One way is to loop through each position in puzzleArray[][]. At each loop pass, generate a random number between 0 and numRows, and a random number between 0 and numColumns. You can use rand.nextInt(). Switch the pieces at the loop location, and the random location generated.
equals(Puzzle compPuzzle): boolean- This method checks that 2 puzzles are exactly the same.
Phase 1 Testing: Make sure to thoroughly test the Puzzle and PuzzlePiece classes before moving forward! Instantiate PuzzlePieces and Puzzles using the various constructors, test the accessors and mutators, invoke any other methods. You may want to define a displayToConsole() method for testing to see the puzzleArray[][].

Phase 2- PuzzleBoard Class

This method creates a JFrame, so make sure it extends the JFrame class.
The GUI represents that puzzle board. It splits the board in 2. The piecesPanel and puzzlePanel will hold the buttons that still need to be placed, and the pieces that have been placed/empty locations. Keep this in mind when setting their layout.
The class should have the following public Panel members: **masterPuzzlePanel, piecesPanel, puzzlePanel.**
The only method is a constructor that takes parameters:
public PuzzleBoard(String title, int rows, int columns)
Should create a puzzle board that looks like this:



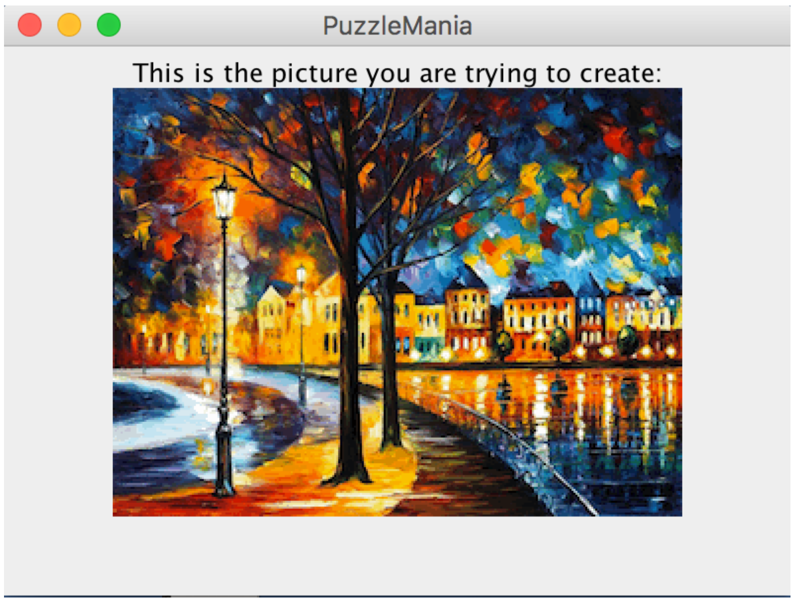
First, select a piece from the green side. Then, select an empty position on the blue side.

To swap: Hit the swap button, and then select the 2 pieces on the blue side you would like to swap

The buttons for the pieces to be placed (on the left), the swap button, and the buttons representing the filled-in puzzle(on the right) will be added in phase 4.

Phase 3- OriginalPicGUI Class

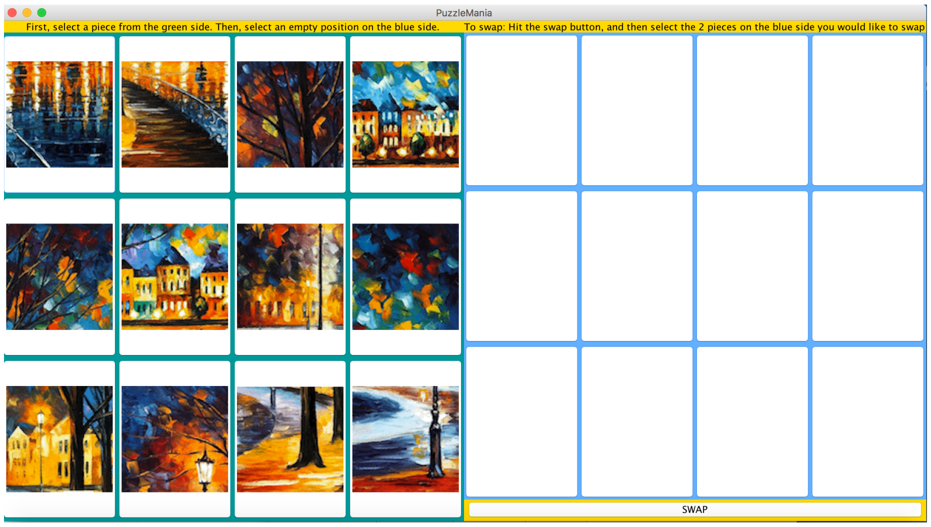
Make sure this method is inherited from the Thread class.
It has one data member: private String puzzleName.
It has 2 methods:
A constructor that takes a puzzle name as a parameter, and sets puzzleName.
The overriden run() method: This should create a GUI that has the original picture on it. It is entitled original.gif, and it is in the folder with the puzzle name in lower case letters.



Phase 4

This is how the game should proceed:
Each of the scrambled puzzle pieces should be placed on a button on the left side of the board.
The player needs to press a button with a piece on it on the left side of the board, and then press an empty location (also a button), on the right side of the board. The piece should then disappear from the left side, and appear on the right side.
Provide a way for the the player to swap 2 pieces that have been placed on the left side of the board. One way to do this is to have the player press a swap button, and then to select the 2 pieces that they want to swap.
When the puzzle is unscrambled, display a congratulations message.

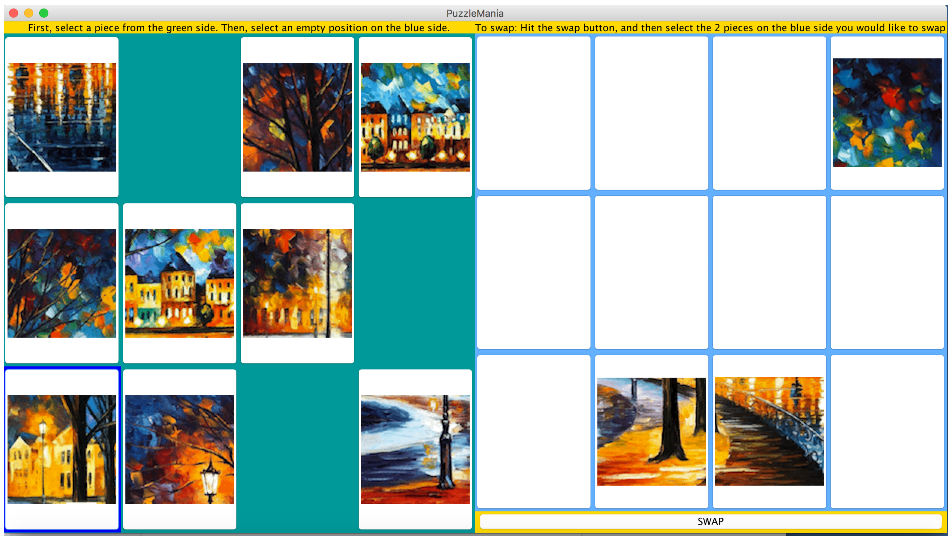
Board when user first sees it:


PuzzleMania

First, select a piece from the green side. Then, select an empty position on the blue side.

To swap: Hit the swap button, and then select the 2 pieces on the blue side you would like to swap

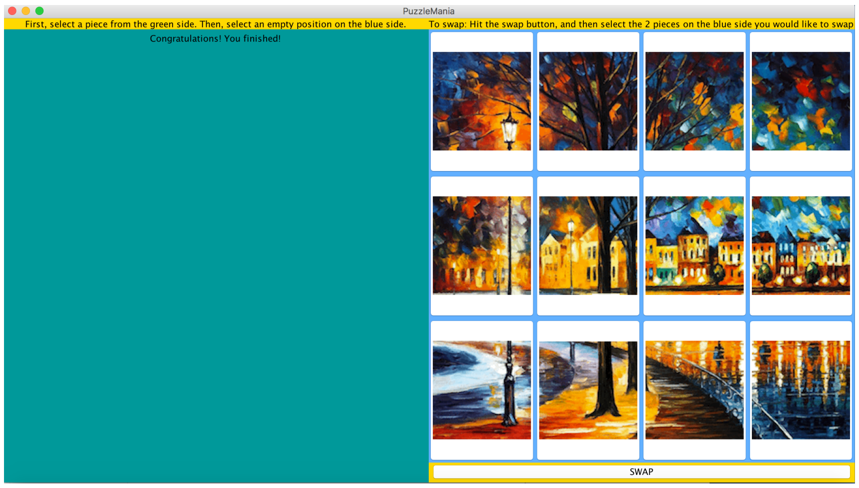
Board mid-play (clicked left-side pieces disappear, and they appear at the chosen location on right side):


PuzzleMania

First, select a piece from the green side. Then, select an empty position on the blue side.

To swap: Hit the swap button, and then select the 2 pieces on the blue side you would like to swap

After the puzzle is unscrambled:

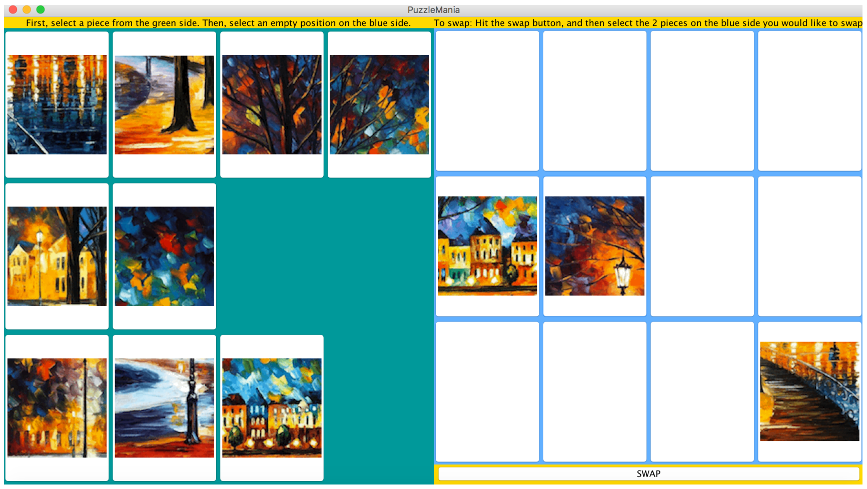

PuzzleMania

First, select a piece from the green side. Then, select an empty position on the blue side.

To swap: Hit the swap button, and then select the 2 pieces on the blue side you would like to swap

Congratulations! You finished!

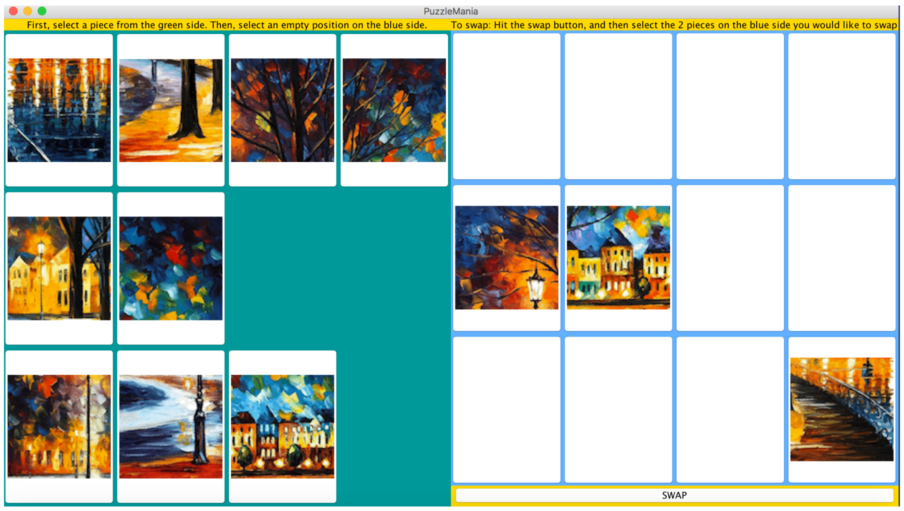
Before pressing “SWAP” button, and then selecting the piece with the house, and the piece with the light (on the right-hand side):



First, select a piece from the green side. Then, select an empty position on the blue side.

To swap: Hit the swap button, and then select the 2 pieces on the blue side you would like to swap

After pressing “SWAP” button, and then selecting the piece with the house, and the piece with the light (on the right-hand side):



First, select a piece from the green side. Then, select an empty position on the blue side.

To swap: Hit the swap button, and then select the 2 pieces on the blue side you would like to swap

Phase 4 testing: Thoroughly play your game to test it. Make mistakes, fix them, run your program through the works!