Overview

In this assignment, you'll add add functionality to the TetrisBoard, as well as improving your Piece behavior. You should add to TetrisBoard a data structure for holding TetrisSquare objects which are on the board, rather than being part of a piece. This data structure should make the following operations as easy as possible:

- adding a TetrisSquare object at a specific x,y location
- checking to see if there is already a TetrisSquare at a specific x, y location
- checking an entire row of locations: (0, y), (1, y), (2, y), etc
- removing an entire row of squares, and moving every square above it down.

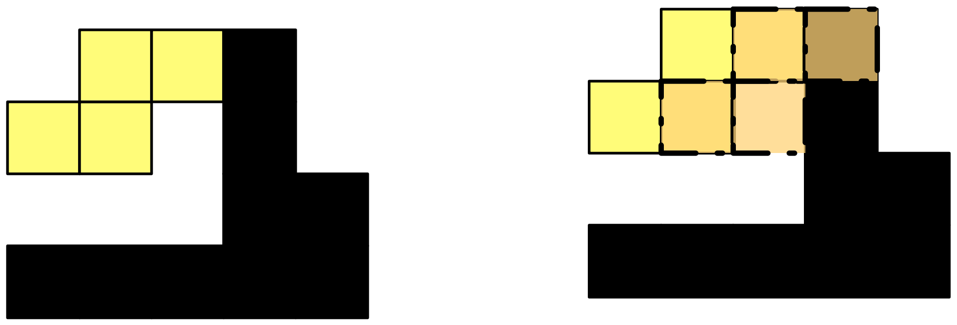
You will also need to update your tetris Piece to check for obstacles and the edge of the board before moving.

Details: Checking for obstacles

In the last checkpoint, you moved your pieces left and right as well as rotating them. Now we'll add a check for obstacles and a commit stage. Each of your existing methods should be separated into 3 stages:

- calculate the new locations for your TetrisSquares
- check these locations to see if they are free.
- if they are free, commit the changes in the TetrisSquare locations (and any associated data)

It is possible to implement a single checkSquares method and a single commitNewLocations method, each of which has parameters that specify the new square locations and are called from every move and rotate method. Otherwise you will need to repeat the checking and committing code in each of your existing methods.



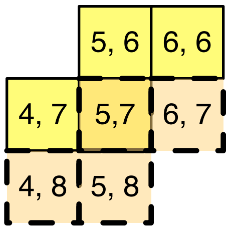
Z Piece move right blocked by existing squares

For now, you should implement the checks to be sure that the Piece doesn't move beyond the edge of the board, and add a placeholder that prints out "Add check for existing squares" where you would check for other squares on the board.

To check for the edges of the board, check for:
- x < 0
- x > TetrisBoard.X_DIM_SQUARES
- y < 0
- y > TetrisBoard.Y_DIM_SQUARES

Details: Moving Down and Generating new Pieces

Move down should be similar to move left and move right, although instead of changing the x-coordinate of the tetris piece, you should increment the y coordinate. You should perform the same checks that you do for move left, move right, rotate left and rotate right.



Move down for the z piece

Rather than being triggered by a key press event, move down should happen every time update() is called on your TetrisGame object. If the piece cannot move down, either because it is blocked by a square on the board or has reached the bottom of the screen, a more complex sequence of events should be triggered:

- The TetrisSquare objects are transferred from the current piece to the TetrisBoard, where they are stored.
- The board checks for completed rows and updates as needed
- A new piece is created, and starts at the normal piece starting location, causing the old piece object/data structure to be replaced with a new one.

The last of these 3 steps is the simplest: simply repeat the steps you took in your TetrisGame constructor to make a new piece.

Details: Transferring Pieces to the Board

You will now start to modify the TetrisBoard to manage Squares on the board. First, you will need to decide what data structure you will use to store TetrisSquare objects when they are on the board. The following will all work, and are perfectly acceptable for this assignment:

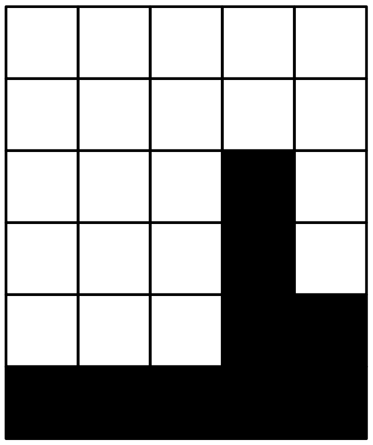
- a 2D array of TetrisSquares
- An ArrayList of ArrayLists of Tetris Squares
- A HashMap (or other map) with Point2D keys and TetrisSquare values

There are some other options, but these should be enough to get you started. Pick one, depending on what you feel most comfortable with.

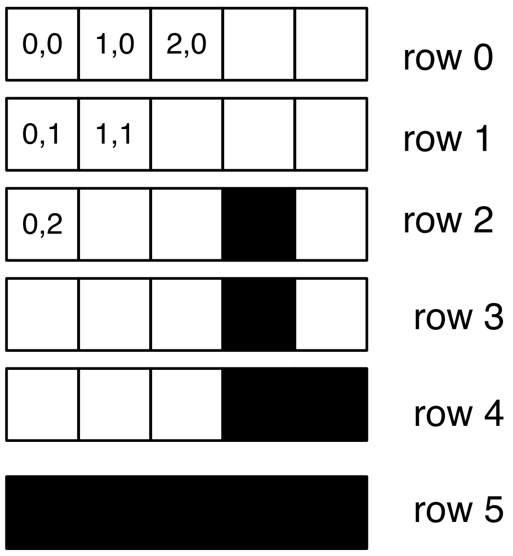
If you use the array or ArrayList as your data structure, you may want to swap the usual order of indices so that y comes first and x second, to make it easy to check or remove an entire row at locations (0, y), (1, y), (2, y), etc. This means that to get/set the square at (x, y), you would use:

squares[y][x] for the 2D array
squares.get(y).get(x) for the ArrayList of ArrayLists

Given this board:



These would be the rows: and you may want to set your data structure up so that square[y] or squares.get(y) refers to an entire row, and then squares[y][x] or squares.get(y).get(x) refers to a specific square.



Here is how each of these data structures would be declared (in your "field"/instance variable region for the TetrisBoard), assuming that y is the first coordinate and x is the second:

2Darray:
private TetrisSquare[][] squares = new TetrisSquare[Y_DIM_SQUARES][X_DIM_SQUARES];

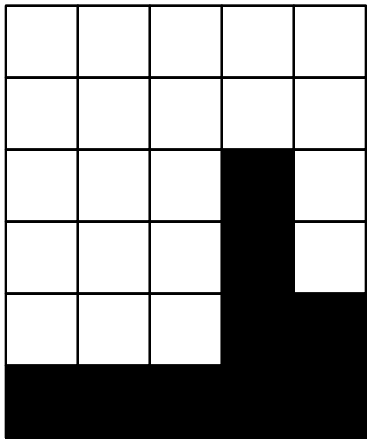
ArrayList of ArrayLists:
private ArrayList<ArrayList<TetrisSquare>> squares = new ArrayList<ArrayList<TetrisSquare>>();
you would then need a loop in your constructor to initialize each row as a new ArrayList<TetrisSquare>();

HashMap:
private HashMap<Point2D, TetrisSquare> squares = new HashMap<>();

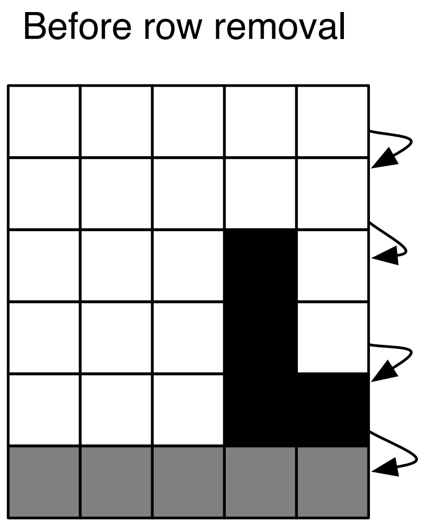
Now you need to implement two basic methods in TetrisBoard using your chosen data structure:
- a method that adds TetrisSquare object to the board, at the square's current x, y location. You may add one square at a time, or all 4 squares from one piece at the same time.
- a method that tests to see if there is already a TetrisSquare in a particular x, y location. You may want to check one location at a time, or you may check all 4 locations from the piece in one method call. Now you can implement the part of your "check move" method from the game/piece to check for existing squares

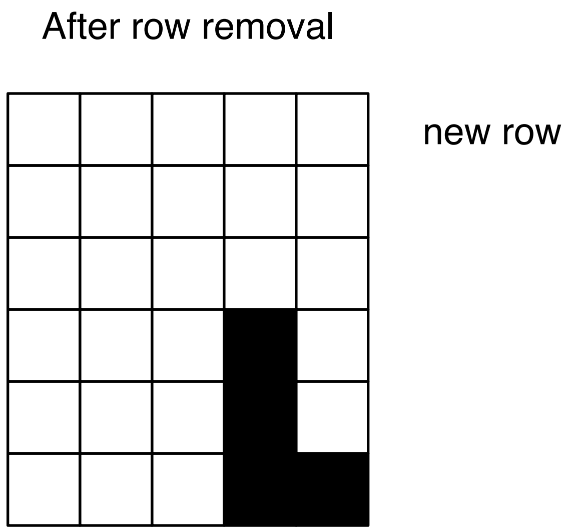
Details: Checking for completed rows and removing them

In the example above, there is a single completed row:



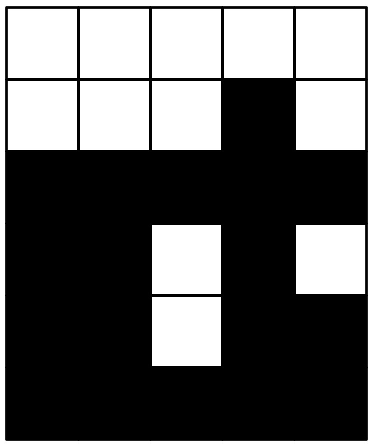
If we remove it, each row above it must move down, and a new empty row must be added at the top:



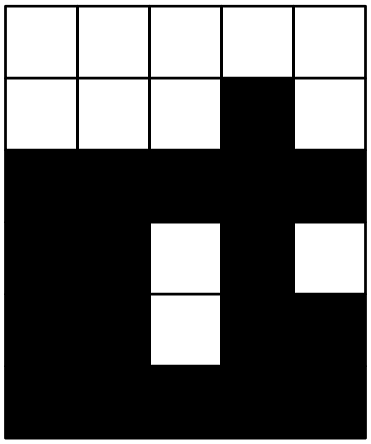


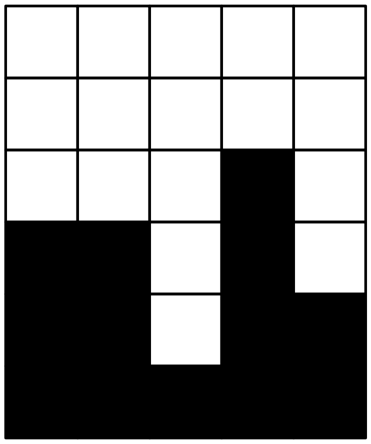
Each time you move a row down, you must update both your data structure and the location of any squares in that row, so that they show up correctly on the screen.

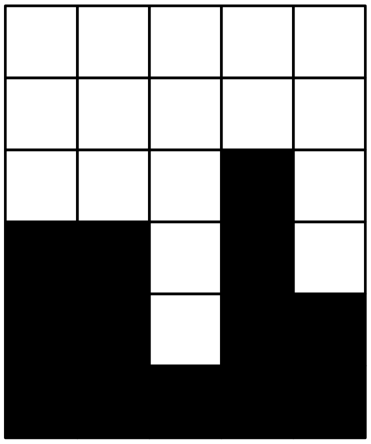
There will be cases where there are multiple rows that must be removed, and they may not be on the bottom:

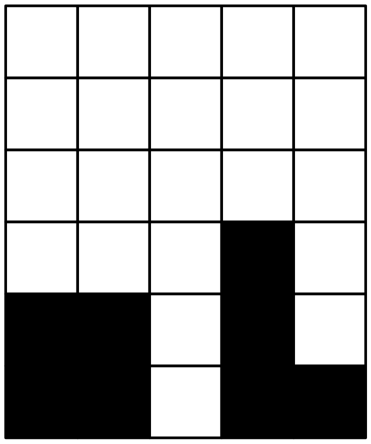


These cases are easiest to handle when you check rows starting at the top of the screen (y-coordinate 0) and work your way to the bottom, checking and removing rows as you go.


remove this row first (row 2)


new row
moved down
moved down


now remove this row


new row
moved down
moved down
moved down
moved down
moved down