

# HW 04: Recursion

---

**Due** Mar 13, 2022 by 11:59pm      **Points** 1      **Submitting** a file upload      **File Types** zip  
**Available** Feb 23, 2022 at 12am - Mar 13, 2022 at 11:59pm



---

This assignment was locked Mar 13, 2022 at 11:59pm.

## Purpose:

The purpose of this assignment is to practice using recursion and backtracking to solve a puzzle.

## Preliminary Knowledge:

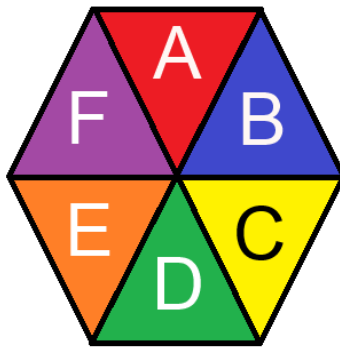
- You may need to review exception handling and file i/o in order to do this assignment. I found two lecture videos online that are pretty good and could be helpful for you.
- [Java Exception Handling](https://youtu.be/R86ObiKhMNc)  [\\_\(https://youtu.be/R86ObiKhMNc\)](https://youtu.be/R86ObiKhMNc)
- [Java File I/O](https://youtu.be/_jhCvy8_IGE)  [\\_\(https://youtu.be/\\_jhCvy8\\_IGE\)](https://youtu.be/_jhCvy8_IGE)

## Detailed Instructions:

Please create a new Eclipse project for this assignment. Inside the project please place all files into a package labeled **hw04**.

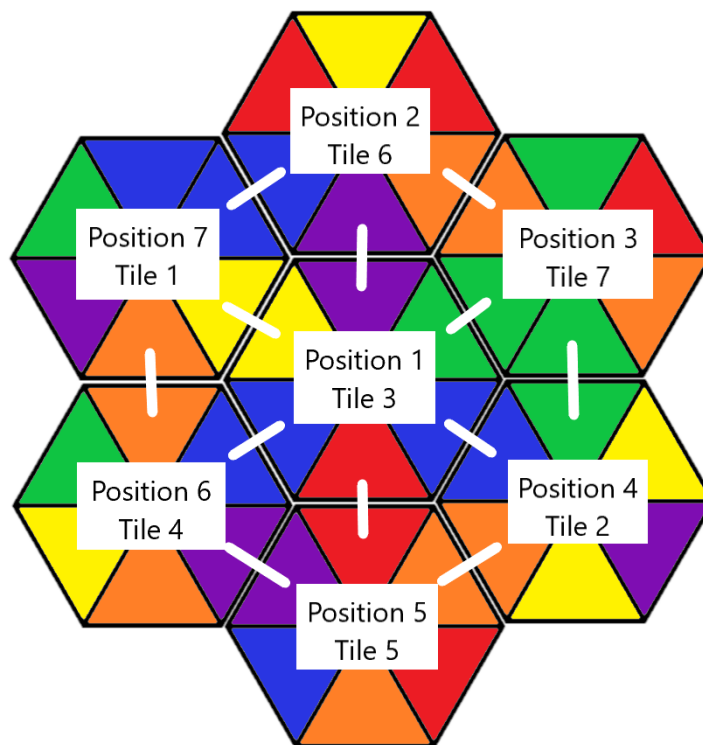
You are given **seven** hexagon tiles each with the following properties:

- Each tile has 6 colored segments labeled A ~ F. Pay no attention to the color of the letters. C is black simply to show up on the yellow background.
- Each segment has a random color chosen from the following: **Red**, **Blue**, **Yellow**, **Green**, **Orange**, **Purple**.
- Colors can be repeated on the same tile.
- The tiles themselves are also numbered 1 - 7 with tile 1 being the first tile in the input file. Tile 2 being the second and so on and so forth.



Your task is to place the 7 hexagon tiles on a board such that a tile is placed in the center of the board and the 6 remaining tiles are placed going clockwise around the first starting at the 12 o'clock position (above segment A of the center tile.) The trick is adjacent segments of the hexagons must have the same color in order for the placement to be valid. **Your program must use recursion to find all possible solutions.** To find a solution you **are required** to use recursion with backtracking to try all rotations of each tile in all 7 board positions until you have exhausted all possibilities (no solution), or all valid solution has been found.

The following presents what a solution might look like. Make note of how the adjacent (touching) sides are the same color indicated by the white lines.



NOTE: The numbers here on the tiles represent the initial tile numbering given by the input text file. So tile 1 is at the center (Position 1), tile 7 is at to the North (Position 2), and so on going clockwise around the center.

You will be given a text file containing information for the 7 tiles. All text files will have seven lines and each line will have the following format:

Tile #:

side\_A\_color,side\_B\_color,side\_C\_color,side\_D\_color,side\_E\_color,side\_F\_color

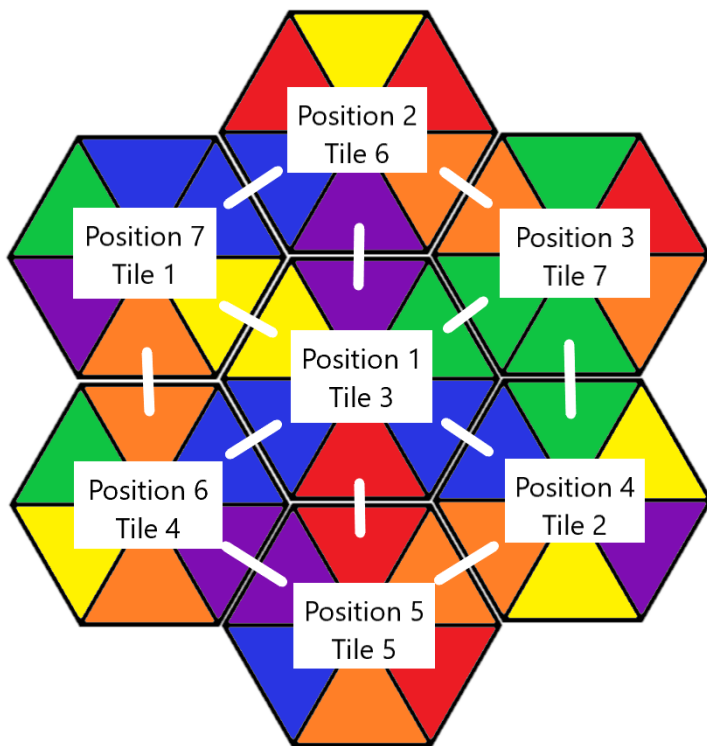
- Tiles will be numbered 1 ~ 7.
- The side colors will be represented by a single character:
  - R: Red
  - B: Blue
  - Y: Yellow
  - G: Green
  - P: Purple
  - O: Orange
- The colors will be separated by commas, with no spaces in between. This will make it easier to parse the color data.

The text file will be the input of the 7 tiles to your program. The program will read the text file and process it into a format that makes it easy to work with in your project. I suggest creating a Hexagon class and Hexagon objects to represent your tiles and any relevant information for each tile. The program will then find all solutions to the puzzle or no solution if none can be found. **The program must solve the puzzle recursively.** If a solution is found, then display the configuration of the hexagons on the console (see below for output format.) If no solution is found display a message saying that there is no solution for the given hexagon tiles.

NOTE: Given a set of 7 hexagons, there may be the following outcomes:

- No Solution
- 1 Solution
- More than 1 Solution.

You are **required** to find **all possible solutions** given a set of tiles. Your output will be a text representation of the tiles and their positions on the board. The following is an example of the solution shown in the picture below:



### Sample Output:

Solution #1-----

	SA	SB	SC	SD	SE	SF
Position 1: Tile #3:	P	G	B	R	B	Y
Position 2: Tile #6:	Y	R	O	P	B	R
Position 3: Tile #7:	G	R	O	G	G	O
Position 4: Tile #2:	G	Y	P	Y	O	B
Position 5: Tile #5:	R	O	R	O	B	P
Position 6: Tile #4:	O	B	P	O	Y	G
Position 7: Tile #1:	B	B	Y	O	P	G

-----

### Please note the following about the output:

- You shall number each solution.
- Each numbered Position is the position on the board where a tile can be placed. Each position number represents the following locations on the board:
  - Position 1: Center
  - Position 2: North
  - Position 3: North East
  - Position 4: South East
  - Position 5: South


- Position 6: South West
- Position 7: North West
- Each Tile number is the number of the tile as it was assigned by the data from the input file. NOTE: The tiles in the input file will be numbered sequentially from 1 to 7, but the tiles may not end up in that order in the solution.
- SA, SB, SC, SD, SE, SF mean Side A, Side B, Side C, Side D, Side E, Side F of each tile.
- The colors must be listed in the correct order (rotation) in your solution starting with Side A (AD) going clockwise around the tile.
  - SA is always the North segment of a tile.
  - SB is always the North East segment of a tile.
  - SC is always the South East segment of a tile.
  - SD is always the South segment of a tile.
  - SE is always the South West segment of a tile.
  - SF is always the North West segment of a tile.
- HINT: For Rotation, the labels SA, SB, SC, etc will stay the same, but it is the colors of those sides that will rotate or change when you "rotate" a tile.
- R, O, Y, G, B, P are the color values of each segment whose meaning is described above.

**NO CREDIT** will be given to programs which do not use **recursion** and **backtracking**.

### Additional Requirements:

- Your program **must use recursion and backtracking** to find all solutions. **NO CREDIT** will be given otherwise.
- Your program is **NOT** allowed to use **class variables / datafields** defined outside of your recursive method.. Any data that you need to pass to your recursive method must be passed through the parameters and any data you want to return from the method must be returned by the **return** statement. Again, **no static variables** may be used anywhere in your code. Basically the class that contains your recursive solving method can only use **local variables** throughout the class.
  - If you choose to use other classes to organize your code, those classes may have data fields, but the class with your recursive method may not.
- Divide your code into logical classes and methods. If you do everything in one class file, I will also give you **NO CREDIT**. Practice what you have learned about good Object Oriented Programming practices.
- Don't forget to include the header comment at the top of each Java file.
- **Document** all of your methods with comments helping to explain what is going on. I will be looking for this on each assignment moving forward.



### Test Input Files:

- [tileset1.txt \(https://calstatela.instructure.com/courses/73267/files/10191280/download?wrap=1\)](https://calstatela.instructure.com/courses/73267/files/10191280/download?wrap=1)   
([https://calstatela.instructure.com/courses/73267/files/10191280/download?download\\_frd=1](https://calstatela.instructure.com/courses/73267/files/10191280/download?download_frd=1))
  - This tile set has only 1 solutions.
  - Output for tileset1:

```

Solution #1-----
                SA    SB    SC    SD    SE    SF
Position 1: Tile #3:  P    G    B    R    B    Y
Position 2: Tile #6:  Y    R    O    P    B    R
Position 3: Tile #7:  G    R    O    G    G    O
Position 4: Tile #2:  G    Y    P    Y    O    B
Position 5: Tile #5:  R    O    R    O    B    P
Position 6: Tile #4:  O    B    P    O    Y    G
Position 7: Tile #1:  B    B    Y    O    P    G
-----

```

- [tileset2.txt \(https://calstatela.instructure.com/courses/73267/files/10191279/download?wrap=1\)](https://calstatela.instructure.com/courses/73267/files/10191279/download?wrap=1)   
([https://calstatela.instructure.com/courses/73267/files/10191279/download?download\\_frd=1](https://calstatela.instructure.com/courses/73267/files/10191279/download?download_frd=1))
  - This should have 235,146,240 solutions.
  - Don't try to render all of these or print all of these out, use this test to see if you can find the correct number of solutions.
  - If you have less than this your algorithm is incorrect.
  - If you have 6 times this number it means you are doing a rotation of the center tile, which is also incorrect.
- [tileset3.txt \(https://calstatela.instructure.com/courses/73267/files/10191278/download?wrap=1\)](https://calstatela.instructure.com/courses/73267/files/10191278/download?wrap=1)   
([https://calstatela.instructure.com/courses/73267/files/10191278/download?download\\_frd=1](https://calstatela.instructure.com/courses/73267/files/10191278/download?download_frd=1))
  - This should have 14 different solutions.
  - You are on your own for validating whether or not all 14 solutions are correct.
    - Welcome to being a programming and having to validate your answers ^\_^ !

## Preliminary Work / Thought Process Journal:

- For this project I would like you to keep a journal of all the work you did leading up to writing the actual code and also document any new thoughts or work on paper that you did while you were writing the code.
- This can include pseudo-code, hand-tracing's that you did on paper, diagrams, notes about what you spent your time working on, and so on...
- You **SHOULD NOT** write one line of code for this assignment until you have worked out all of the logic and you should work out this logic by hand.
- Please write everything on paper and take clear pictures of your "Work Journal" and turn the pictures in with your code.
  - **NOTE:** Please make sure your pictures are PNG or JPEG.

- Things to Think About:
  - How do you want to represent the hexagon tiles?
  - How do you simulate "rotating" a tile?
  - How are you going to keep track of the "board" and which tiles are in which positions?
  - How do you know when you have found a solution?
  - How do you know when a current partial solution will not lead to an actual solution?
  - When should you backtrack and what should you do upon backtracking?
  - How do you know when no solution can be found given then current tile set?
  - How do you know when you have found all solutions?
  - **NOTE:** These are but a few important things to think about and not the only things you should think about. All of these should stimulate your thought process for this assignment and allow you to think about how to approach the code. Again all of this should be documented on paper and turned in with your assignment.
- If you do not turn in a detailed "Work Journal" along with a working version of the assignment, **you will receive no credit.**

## Video Presentation Checklist:

- **Explain** how you represented your hexagons and rotations.
- **Explain** how you represented the board.
- **Explain** your recursive solution to the problem.

## Submission of Deliverables:

- You will need to turn in all **.java** files related to this assignment.
- **Work Journal:** Please take pictures of your **handwritten** journal and include the pictures in a folder in your **.zip** file.
- Please **.zip** your **src** folder from your Eclipse project and turn in the entire **.zip** file to Canvas by the due date and time.
  - Your src folder should include the hw04 package and all the .java files.
- **DOCUMENTATION:** Make sure that you document your code with JavaDoc comments.
- **VIDEO:** **See above.**