# HW 05: Linked Lists

**Due** Apr 10, 2022 by 11:59pm **Points** 1 **Submitting** a file upload **File Types** zip
**Available** until Apr 10, 2022 at 11:59pm

This assignment was locked Apr 10, 2022 at 11:59pm.

## CS-2013 Programming with Data Structures

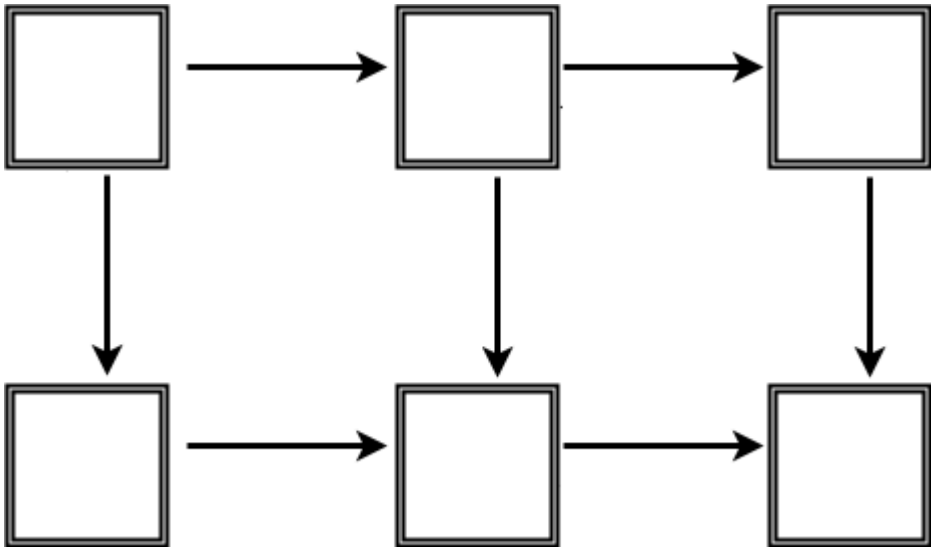
## Homework 05: Linked Lists

## Purpose:

The purpose of this assignment is to practice your knowledge of how a linked data structure is implemented. This is a more advanced version of a linked list, and will help to give you practice in how to deal with data structures connected through references. This assignment is very important to help you understand the fundamentals of Linked Lists. At the end of this assignment, you should be able to fully understand how any variation of a Linked List is implemented.

## Detailed Instructions:

For this project you will be designing a new data structure called an **`Array2D`**. This is a linked type data structure version of a 2D array. The **`Array2D`** can support any combination of dimensions (n x n, n x m, or m x n, where n > m or n < m) This data structure has a singly linked system which allows you to traverse the array in multiple directions in addition to allowing you to retrieve specific rows and columns. The following diagram gives a rough visual idea of this data structure. NOTE: Your Array2D MUST work with any number of nodes, the picture only shows 6 as an idea. You may not hardcode only six nodes...



## Package Declaration and General Setup:

- Create a new project in Eclipse for this assignment.
- Create a package in your new project called **hw05**.
- Do not forget to include the header comment, as well as comments throughout your code documenting your work.

## Array2DNode<E> Class:

- Implement this class according to the following requirements.
- This is a ***generic class*** and must be implemented accordingly.
- **Data Fields**:
  - **`item`**: store the data here. This data field is private and should have an appropriate getter and setter if necessary.
    - NOTE: You may make this datafield protected so that your other class can access it more easily.
  - **`nextCol`** and **`nextRow`**: these are your directional references to traverse your data structure. These references each point to the next node in sequence. If there is no node in a direction, the value of that reference should be **null**. These data fields should be protected in order to make accessing them from the Array2D class easier. These data fields should NOT have a getter or setter.
  - No other data fields are allowed.
- **Constructor**:
  - **`Array2DNode(E item)`**: Constructor which takes an item of any type and initializes the item data field.
  - No other constructors are allowed.

## Array2D<E> Class:

- Implement this class according to the following requirements.
- This is a ***generic class*** and must be implemented accordingly.
- **Data Fields** (All data fields are private):
  - **`rowSize`**: The number of rows.
  - **`colSize`**: The number of columns.
  - **`head`**:
    - This is an **`Array2DNode`** reference, which refers to the node at position (0,0) in the array.
    - Be very careful how you manipulate this reference as you do not want to accidentally delete any nodes in your data structure by moving this one around incorrectly.
    - Please make this data field **protected**.
  - **`rowTail`**:
    - This is a **`Array2DNode`** reference, which points to the node at the beginning of the last row.
    - This reference will help to make implementing some of the methods easier.
  - **`colTail`**:
    - This is a **`Array2DNode`** reference, which points to the node at the beginning of the last col.
    - This reference will help to make implementing some of the methods easier.
  - NOTE: Always remember to correctly update these references whenever adding or removing things from the array.
  - No other data fields are allowed.
- **Constructors:**
  - **`Array2D()`**:
    - Default constructor which has 0 rows and 0 cols.
    - head and tail references should start at **null.**
    - The Array2D at this point will have no nodes in it.
    - **NOTE:** If the user adds a row or column when there are none, then you would add **one node** to start increasing the row and cols by 1. (So after adding one node, the count for numRows and numCols becomes 1. After that follow the normal case for adding a new row or column.
      - Also note, the user could choose to add a new row OR new column when the data structure is empty. In either case the outcome is the same, you add 1 node, and set **both** numRows and numCols to be 1.
      - This is a special case which you must account for in your implementation.
  - **`Array2D(E[][] values)`**:
    - Constructor which takes a normal 2D array as an argument.
    - This is the ONLY place you can use a 2D array and this 2D array is only used to initialize the values of your Array2D object.
    - The parameter 2D array must remain as a local variable and is not allowed to be stored as a data field.
    - NOTE: You may pass the parameter 2D array to another helper function if necessary, but the 2D array cannot be stored as a data field in the `Array2D` Class.

        - For this constructor, you use the dimensions of the 2D array to create the linked structure, while at the same time copying the values from the `E[][]` array into your linked array.
    - No other constructors are allowed.
- **Public Functions:**
    - `addFirstCol(E ... values)`:
        - Adds a new column of nodes, populated with data from the **values** parameter, to the beginning of the data structure.
        - All the nodes in the new column must be connected to each other vertically, as well as to the existing first column horizontally.
        - You must verify that the number of values passed into the method parameter, matches the current number of **rows**. If not, then you must throw an `IllegalArgumentException` with an appropriate error message.
        - Don't forget to increase the **numCols** by 1 at the end of this method.
    - `addFirstRow(E ... values)`:
        - Adds a new row of nodes, populated with data from the **values** parameter, to the beginning of the data structure.
        - All the nodes in the new row must be connected to each other horizontally, as well as to the existing first row vertically.
        - You must verify that the number of values passed into the method parameter, matches the current number of **columns**. If not, then you must throw an `IllegalArgumentException` with an appropriate error message.
        - Don't forget to increase the **numRows** by 1 at the end of this method.
    - `addLastCol(E ... values)`:
        - Adds a new column to the end of the list.
        - This method has the same requirements as `addFirstCol()`, but instead of adding to the beginning, you are adding to the end.
    - `addLastRow(E ... values)`:
        - Adds a new row to the end of the list.
        - This method has the same requirements as `addFirstRow()`, but instead of adding to the beginning, you are adding to the end.
    - `insertCol(int index, E ... values)`:
        - Inserts a column of values at the given index.
        - This method should not replace an existing column, but perform an insertion where you make room for the new column. The parameter index will be the insertion point and this index refers to the column index of where you should do the insertion.
        - The new column must contain the given values. The nodes of the new column should be connected to each other vertically as well as horizontally to the columns on either side of it.
        - Remember to consider the three special cases: when you insert at index 0, insert anywhere in the middle, or insert at the end.

- You must verify that the number of values passed into the **values** parameter, matches the current number of **rows**. If not, then you must throw an **`IllegalArgumentException`** with an appropriate error message.
- You must also verify that the **index** is within bounds of the column indexes. If it is out of bounds, then you must throw an **`IndexOutOfBoundsException.`**
- Remember to increase the number of cols by 1.

- **`insertRow(int index, E ... values)`**:
  - Inserts a row of values at the given index.
  - This method should not replace an existing row, but perform an insertion where you make room for the new row. The parameter index will be the insertion point and this index refers to the row index of where you should do the insertion.
  - The new row must contain the given values. The nodes of the new row should be connected to each other horizontally as well as vertically to the rows above and below it.
  - Remember to consider the three special cases: when you insert at index 0, insert anywhere in the middle, or insert at the end.
  - You must verify that the number of values passed into the **values** parameter, matches the current number of **columns**. If not, then you must throw an **`IllegalArgumentException`** with an appropriate error message.
  - You must also verify that the **index** is within bounds of the row indexes. If it is out of bounds, then you must throw an **`IndexOutOfBoundsException.`**
  - Remember to increase the number or columns by 1.
- **`deleteFirstCol()`**:
  - This method takes no parameters and returns nothing.
  - Remove the first column of nodes from the Array.
  - Make sure that you don't have any "dangling references" where something is pointing to a node that is supposed to be removed.
  - Don't forget to decrease the colSize by 1.
- **`deleteFirstRow()`**:
  - This method takes no parameters and returns nothing.
  - Remove the first row of nodes from the Array.
  - Make sure that you don't have any "dangling references" where something is pointing to a node that is supposed to be removed.
  - Don't forget to decrease the rowSize by 1.
- **`deleteLastCol()`**:
  - This method takes no parameters and returns nothing.
  - Remove the last row of nodes from the Array.
  - Make sure that you don't have any "dangling references" where something is pointing to a node that is supposed to be removed.
  - Don't forget to decrease the colSize by 1.
- **`deleteLastRow()`**:

- This method takes no parameters and returns nothing.
- Remove the last row of nodes from the Array.
- Make sure that you don't have any "dangling references" where something is pointing to a node that is supposed to be removed.
- Don't forget to decrease the rowSize by 1.

- **`deleteCol(int index)`**:
  - This method returns nothing.
  - Remove the column of nodes from the Array at the given index.
  - Make sure that you don't have any "dangling references" where something is pointing to a node that is supposed to be removed.
  - Don't forget to decrease the colSize by 1.
  - Make sure to validate index and throw an **`IndexOutOfBoundsException`** if necessary.
  - Consider the three cases when you are deleting from the beginning, middle, or end of the structure.
- **`deleteRow(int index)`**:
  - This method returns nothing.
  - Remove the row of nodes from the Array at the given index.
  - Make sure that you don't have any "dangling references" where something is pointing to a node that is supposed to be removed.
  - Don't forget to decrease the rowSize by 1.
  - Make sure to validate index and throw an **`IndexOutOfBoundsException`** if necessary.
  - Consider the three cases when you are deleting from the beginning, middle, or end of the structure.
- **`get(int rowIndex, int colIndex)`**:
  - Returns the **item** at the given (rowIndex, colIndex).
  - NOTE: You must return the item stored in the `Array2DNode`, **NOT** the `Array2DNode` object.
  - You must validate that **`rowIndex`** and **`colIndex`** are within bounds. Throw an **`IndexOutOfBoundsException`** if necessary.
- **`getCol(int colIndex)`**:
  - Returns a generic ArrayList<E> which holds the values from the requested column.
  - NOTE: This is not an ArrayList of nodes, but of the values in the nodes.
  - You must validate that colIndex is within bounds. Throw an **`IndexOutOfBoundsException`** with an appropriate error message if necessary.
- **`getRow(int rowIndex)`**:
  - Returns a generic **`ArrayList<E>`** which holds the values from the requested row.
  - NOTE: This is not an **`ArrayList`** of nodes, but of the values in the nodes.
  - You must validate that **`rowIndex`** is within bounds. Throw an **`IndexOutOfBoundsException`** with an appropriate error message if necessary.
- **`set(int rowIndex, int colIndex, E item)`:**
  - Assigns the given item to the `Array2DNode` at the given **`rowIndex`** and **`colIndex`**.

        - This method will replace the any previous value stored at that location with the new value.
        - You must validate that **`rowIndex`** and **`colIndex`** are within bounds. Throw an **`IndexOutOfBoundsException`** if necessary.
    - **`colSize()`**:
        - Returns the number of columns.
    - **`rowSize()`**:
        - Returns the number of rows.
    - **`toString():`**
        - Override the Java toString() method to return a string representation of your table, where the table is traversed row by row.
    - **`toStringColByCol():`**
        - Add a method to return a string representation of your table, where the table is traversed col by col.
    - No other public functions are allowed.
- **Private Functions:**
    - You may implement any other private functions you feel are necessary to help you implement the above constructors / public functions.
    - If you are unsure whether a private function will violate the requirements for this assignment, be sure to ask me.

## Array2DTester Class:

- This class is where you should do al of your testing and demonstrate that all functionality of your project works. Try to design it using Unit Testing techniques introduced in class.
- Again, this class should properly demonstrate that all functionality of your data structure works.

## Additional Requirements:

- This data structure is zero-indexed (meaning that the first **`rowIndex`** or **`colIndex`** is always zero).
- Your data structure must be designed with generics.
- Your data structure must include the required functionality and names cannot be changed.
    - If your submission does not compile with my test program, you will receive no credit.
- You are **NOT** allowed to use any other data structures in the implementation of your **`Array2D`**. This includes but is not limited to arrays, 2D arrays, arraylists, and so on. The point is to create a new data structure. If you are unsure of something feel free to ask. NOTE: The only place you may use a 2D Array is in the methods where it is specifically required. You will receive no credit for the assignment for not following this requirement.
    - NOTE: This does not apply to the parts of the program where I specifically ask you to use another data structure.
- You must use reference hopping to traverse your data structure as much as possible.
    - (This means having a current reference which can move through the structure.)
    - You may use the counting traversal method where appropriate.

- **HINT:** You may need multiple reference variables to help you keep track of where you are in the data structure, especially when adding or removing rows and columns.

# Submission of Deliverables:

- You will need to turn in all .java files related to this assignment.
- Please **zip** your **src** folder from your Eclipse project and turn in the entire **.zip** file to Canvas by the due date and time.
- **DOCUMENTATION:**
  - I would like you to document (with proper **JAVADOC comments**) all of your .java files for this project.
    - Please follow the example for documentation on Canvas.
    - If your documentation is too sparse or incomplete, it will count against your grade for this assignment.
  - The header comment should include your name, cin, course number, section numbers, in the author tag.
- **VIDEO EXPLANATION:**
  - For your video, please do the following:
    - Explain in your own words how you understand this data structure.
    - Explain your **insertRow()** and **insertCol()** methods.
      - Be sure to discuss the case when you insert at the beginning, end, or middle.
    - Explain your **deleteRow()** and **deleteCol()** methods.
      - Be sure to discuss the case when you delete from the beginning, end, or middle.
    - Explain your process for testing your code.
  - Please paste the public link to your video as a comment on Canvas.
    - Once the video is recorded and uploaded to Canvas Studio, there should be an option on the video to create a public link. This is the link that will allow me to watch the video.
    - No other link will work.

# Criteria for Success:

In order to receive credit for this assignment, your program must do all of the following successfully.

- Be able to compile and run from beginning to end with no runtime errors (program crashing).
- Be able to demonstrate that all requirements were implemented successfully.
- Be able to demonstrate that all operations (methods/functions) were implemented and the results of each operation are correct.

- Use the requirements of each class as a checklist.  If any one item fails to perform as required, your program will most likely not receive any credit.

**Good Luck! and Have Fun!**