Programming II (420-G20-HR)
Assignment 2 – Arrays

Date assigned:	Wed. March 17, 2021
Test Plan, Class Diagram due:	**Mon. March 22, 2021 in class**
Phase I due (CLI Connect-N):	**Thurs. April 1, 2021**
Phase II due (Frame Connect-N):	**Fri. April 16, 2021**

Learning Objectives

Upon successful completion of this assignment, the student will be able to:

1. Design test cases from requirements
2. Design a frame from requirements.
3. Design a class hierarchy from requirements.
4. Use two dimensional arrays.
5. Read from a text file.
6. Write to a text file.
7. Use various Swing UI methods

To be handed in:

1. A test plan and class diagram should be handed in for review during class on Mon. Mar 22. The test plan should be a numbered list of test scenarios that need to be tested. The files, the user interface, and the game logic all need to be considered in the test plan.
2. A Java project called ***username*_G20_A02_PhaseI** should be zipped and uploaded to Moodle on Thurs. April 1, 2021. This should contain the Connect-N game working through the command line interface (CLI), without the frame.
3. The final Java project (with frame) called ***username*_G20_A02_PhaseII** should be zipped and uploaded to **Moodle** on Fri. Apr. 16, 2021 and should include the following in addition to the Java **src** folder:
   a. ***username*_G20_A02_Test_Plans.docx** containing the updated test plans for the system.
   b. A self-assessment for the assignment. (Use the template from **Moodle**.)

Programming Style:

Marks will be deducted for poor style. The following criteria will be used:

- All Java source files must be formatted using the Eclipse formatter. (Right-click and select **Source→ Format**.),
- Your code must meet all the requirements of the Heritage College Computer Science Code Checklist.

Organization:

5 marks of the assignment mark will be for organization. The self-assessment must be completed and included in the assignment folder and the assignment must be:

- handed in to the correct location,
- be properly named
- be complete according to the assignment specifications

Marking Scheme:

		Out of	
Documentation	Test Case Plan – includes tests for the CLI, the Frame, the file and the game.	15	
	Class Diagram – all classes and methods included; correct use of access.	8	
Phase I	ConnectNInterface java code: board is displayed after each move; can enter a move and have it validated with an error message; can undo last move; can quit the game at any time;	32	← CLI
	ConnectNGame java code – code divided correctly between classes; constructors, accessors, mutators; two dimensional array used; read in board from file; write board to file; illegal move identified; undo last move;.	45	
Phase II	ConnectNFrame java code - all frame components work; can enter game size, players; can enter a move and have it validated with a pop-up error message; can undo last move; exit functionality, save functionality;	50	
Both Phases	Game works for N, and not just for Connect 4. Variable sized board and N value.	15	
	Correct execution against requirements; thoroughly tested against test cases.	20	
	Organization	5	
	Total	**190**	

Problem Specification

Connect Four is a classic two player board game. The players first choose a colour (either red or yellow) and then take turns dropping coloured checkers from the top into a seven-column, six row grid. The checkers fall straight down, occupying the next available space within the column. To win Connect Four you must be the first player to get four of your coloured checkers in a row either **horizontally, vertically or diagonally**.

You are going to build Connect N, which is a variant of Connect Four where the size of the board can be specified and the number of checkers required in a row to win can be specified by the player.

Phase 1 of your program will include the Connect N game being played from a command line interface. Phase 2 will include the Connect N game being played from a frame.

Phase 1 of your program must include the following functionality:

1. The program prompts the user for the name of player 1, who uses colour yellow, and player 2, who uses colour red.

2. The user has the choice of starting a new game, or continuing a game that is already in progress.

3. If the user chooses to start a new game, then the user must specify the size of the game board - how many rows and how many columns. A board does not have to have the same number of rows as columns, it can be rectangular. A board must have at least four and at most twelve rows and columns. The user must also specify the value of N – how many checkers in a row constitutes a win. N can be between three and eight.

Once the number of rows, columns and N is entered and validated, an empty game board is created.

4. If the user chooses to continue a game that is already in progress, then the in progress game is read in from a file called `currentGame.txt` and play resumes on that board. A sample `currentGame.txt` file is provided for a 6 by 7 game of Connect Four. The first line indicates the number of rows, the second line indicates the number of columns and the third line indicates the value of N. The fourth line indicates the name of player 1, the fifth line indicates the name of player 2 and the sixth line indicates which player has the next turn. A value of R indicates a red checker is in that square, Y indicates a yellow checker is in that square and E indicates an empty square.

```
6
7
4
Sandra
Marissa
1
R~E~E~E~E~E~E
Y~E~E~Y~E~E~E
R~E~E~Y~E~E~E
Y~R~Y~R~R~E~E
Y~R~Y~R~R~E~E
Y~Y~Y~R~R~E~E
```

rows
cols
N
Player 1
Player 2
whose turn is it
Game Board
Y - yellow (Player 1)
R - red (Player 2)
E - empty

5. The board should be displayed to the user. To play, the player enters a square number to place their checker in. For example, if the red player wants to place a red checker in row 1 column 6, then they should specify 1,6. It is up to you to decide how to prompt the user for the input. Once the move has been accepted, the program should display the updated board.

don't tell the user about row zero or column zero

6. Your program must validate input. The user should only be able to enter valid row and column values.

7. Your program should identify illegal moves. If the player enters a square that does not have a checker beneath it, then the user should be informed of the error and the move should not be made on the board

8. The player can only modify empty squares. Once there is a checker in a square (either because it was there to start with, or because of a move the player made), then that square cannot be changed.

9. Your program should allow the player to undo the last move. Only the last move can be undone. If the player undid their last move, and then tried to undo the move again before making another move, then this should not be allowed.

10. The player should be able to quit the game at any time.

11. The player should be able to save the game at any time. When save is selected, then the state of the game board should be written to the file `currentGame.txt`, overwriting any previous game stored in the file.

12. Your program should detect the first player to have N checkers in a row, either horizontally, vertically or diagonally, and indicate the player that won the game.

13. Your program should detect when the game is over due to all squares being occupied with a checker.

detect win or end

14. **You must use a two-dimensional array to store the game board.**

storage, server, logic

input, output, validation

15. You must have a **ConnectNGame** class and a **ConnectNInterface** class. The **ConnectNInterface** class will be your command line interface (CLI) for Phase 1.

Can farm out jobs to other classes

A sample run of the program follows:

```
Welcome to Heritage Connect-N.

Please enter N to start a new game or R to resume the game stored in currentGame.txt
➔  N
Enter the number of rows on the game board: 6
Enter the number of columns on the game board: 7
Enter the value for N, the number of checkers in a row for a win: 4
Enter the name of Player 1, yellow: Sandra
Enter the name of Player 2, red: Marissa
Type Q at any time to exit the game, S to save the game or U to undo your last move

        E   E   E   E   E   E   E
        E   E   E   E   E   E   E
        E   E   E   E   E   E   E
        E   E   E   E   E   E   E
        E   E   E   E   E   E   E
        E   E   E   E   E   E   E

Sandra, enter square number (row, column) of your move -> 1,3
        E   E   E   E   E   E   E
        E   E   E   E   E   E   E
        E   E   E   E   E   E   E
        E   E   E   E   E   E   E
        E   E   E   E   E   E   E
        E   E   Y   E   E   E   E
```

```
Marissa, enter square number (row, column) of your move -> 1,6
        E   E   E   E   E   E   E
        E   E   E   E   E   E   E
        E   E   E   E   E   E   E
        E   E   E   E   E   E   E
        E   E   E   E   E   E   E
        E   E   Y   E   E   R   E

Sandra, enter square number (row, column) of your move -> 2,3
        E   E   E   E   E   E   E
        E   E   E   E   E   E   E
        E   E   E   E   E   E   E
        E   E   E   E   E   E   E
        E   E   Y   E   E   E   E
        E   E   Y   E   E   R   E
```

Print on screen after each move

```
Marissa, enter square number (row, column) of your move -> 1,8
Invalid column number.  Please try again.

Marissa, enter square number (row, column) of your move -> 8,2
Invalid row number.  Please try again.

Marissa, enter square number (row, column) of your move -> 1,3
Invalid move.  That location already has a checker.  Please try again.

Marissa, enter square number (row, column) of your move -> 3,6
Invalid move.  You must place the checker on top of another checker.  Please try again.

Marissa, enter square number (row, column) of your move -> 1,5
        E   E   E   E   E   E   E
        E   E   E   E   E   E   E
        E   E   E   E   E   E   E
        E   E   E   E   E   E   E
        E   E   Y   E   E   E   E
        E   E   Y   E   R   R   E

Sandra, enter square number (row, column) of your move -> U
Marissa, your last move has been undone.
        E   E   E   E   E   E   E
        E   E   E   E   E   E   E
        E   E   E   E   E   E   E
        E   E   E   E   E   E   E
        E   E   Y   E   E   E   E
        E   E   Y   E   E   R   E
```

```
Marissa, enter square number (row, column) of your move -> 2,6
        E   E   E   E   E   E   E
        E   E   E   E   E   E   E
        E   E   E   E   E   E   E
        E   E   E   E   E   E   E
        E   E   Y   E   E   R   E
        E   E   Y   E   E   R   E

Sandra, enter square number (row, column) of your move -> S
Game state has been saved to the file.

Sandra, enter square number (row, column) of your move -> 3,3
        E   E   E   E   E   E   E
        E   E   E   E   E   E   E
        E   E   E   E   E   E   E
        E   E   Y   E   E   E   E
        E   E   Y   E   E   R   E
        E   E   Y   E   E   R   E

And so on…
```

Phase 2 of your program must include the following functionality:

1. The **ConnectNGame** class should not change in Phase 2. The game rules remain the same. You must have the same **ConnectNGame** class from Phase 1, and a **ConnectNFrame** class.

2. It is up to you to design your frame. You can have the user enter their move in a separate "move" text box, or you can have the user enter their move right in the square, or some other design.

3. When an error in input is detected (either because of invalid input or because of an illegal move), a pop-up window indicating the error message should be displayed.

4. When the game is won, a pop-up window indicating the winner should be displayed.

5. When the game is over without a winner, a pop-up window indicating the game is over should be displayed.

6. A drop down menu should exist that has the following options, organized in logical submenus:

   a) Undo last move
   b) Help – this should display a box that has instructions for the game.
   c) About – this should display a box with the title of your program, your name, year, and Heritage College

d) New Game – this should create an empty game board, after the user enters the number of rows, columns and value of N in addition to the two player names.

e) Restore Game – this should create a game board from the file `currentGame.txt` and resume play from the state of the game board.

f) Save Game – this should save the state of the game to the file `currentGame.txt`

g) Exit – this should quit the application without saving the state of the game.

Your tasks:

1. Create test plans for testing the **ConnectNGame** class, the **ConnectNInterface** class and the **ConnectNFrame** class. This should be a numbered list of the scenarios to be tested. The scenarios should describe in a sentence what to test and the expected result.
2. Design the system. Draw a class diagram that includes all classes involved, including your frame class.

Document

3. Get approval of your class diagram and test plans before you start coding.
4. Code the **ConnectNGame** and **ConnectNInterface** classes according to the specifications. Test as you go.
5. Completely test your program using the test plans you developed in the first step.
6. Hand in Phase 1.

Phase I

7. Design the frame. This can be a hand drawn design.
8. Code the **ConnectNFrame** class according to the specifications. Test as you go.
9. Completely test your program using the test plans you developed in the first step.
10. Document your code with comments; format your code.
11. Complete the self-assessment.

Phase II