Welcome to Multi-Dimensional Arrays Lab

Learning Outcomes

By the end of this lab:

- Be able to define chapter 8 terms
- Draw a picture of multi-dimensional arrays.
- Describe, trace, explain and write code with multi-dimensional arrays.
  - That passes arrays to methods, changes arrays, changes array references and returns arrays from methods.

Exercise A: Terms

Unit testing, test bench, test case/test vector, border cases/corner cases, array parameter

Exercise B: Trace and Explain

Example:
```java
char [][] strArr = new char[][]{{'X','O','O'},{'O','X',' '},{'X','O','X'}};
```

Draw a diagram showing memory for each to help explain what is happening.

1. double []nums = {8.0, 9.5, 10.1};

2. double [][] table = {{1.0,2.0,3.0},{4.0,5.0,6.0},{7.0,8.0,9.0}};

3. char [][] letters = new char[][]{{'a','b'},{'c','d'}};
   System.out.println( letters[1][1]);

4. char [][] letters = new char[][]{{'A','B'},{'C','D'}};
   letters[0][1] = letters[1][0];
   System.out.println( letters[0][1]);
5. char [][][] letters = new char[][]{{'A','B'},{'C','D'}};
   letters[0] = letters[1];
   System.out.println( letters[0][1]);

6. char [][] letters = new char[][]{{'A','B'},{'C','D'}};
   letters[1] = new char[]{'E','F'};
   System.out.println( letters[1][1]);

7. int [][] letters = new int[][]{{1,2},{3,4,5}};
    for ( int r = 0; r < 2; r++) {
        for ( int c = 0; c < 3; c++) {
            System.out.print( letters[r][c] + " ");
        }
    }

    //What is the error in this code? How would you fix it?

8. int [][] letters = new int[][]{{1,2},{3,4,5,6}};
    letters[0] = new int[]{7,8,9,10,11};
    letters[1] = letters[0];
    for ( int r = 0; r < letters.length; r++) {
        for ( int c = 0; c < letters[r].length; c++) {
            System.out.print( letters[r][c] + " ");
        }
    }
    System.out.println();

9. int [][] a = new int[5][];
    for(int i = 0; i < a.length; i++) {
        a[i] = new int[i+1];
        for(int j = 0; j < a[i].length; j++) {
            a[i][j] = j;
        }
    }

10. public static void methodA(boolean []items) {
        items[0] = true;
    }
    
    public static void main(String []args) {
        boolean []list = new boolean[5];
        methodA( list);
        System.out.println( list[0]);
    }

11. public static void methodB(boolean []items) {
        items = new boolean[2];
        items[1] = true;
        return;
```java
public static void main(String []args) {
    boolean []list = new boolean[5];
    methodB( list);
    System.out.println( list[4]);
}

Exercise C: More Scanner

A tutorial describing how Scanner works:

For each of the following test cases, step through the code to see how the code handles these situations.

**ScannerExample.java**

String name;
double weight = 0;

Scanner scnr = new Scanner(System.in);

System.out.print("Enter the name of a rock:");
name = scnr.nextLine();

boolean haveValidWeight = false;
do {
    System.out.print("Enter weight of rock:");
    if (scnr.hasNextDouble()) {
        weight = scnr.nextDouble();
        scnr.nextLine();
        if (weight > 0) {
            haveValidWeight = true;
        } else {
            System.out.println("Weight must be\n" + " greater than 0.");
        }
    } else {
        String invalid = scnr.nextLine();
        System.out.println("Invalid input: "
                           + invalid);
    }
} while (!haveValidWeight);

System.out.println("Name: " + name
                    + "\nWeight: " + weight);
scnr.close();

| Case 1: | Enter the name of a rock: iron ore  
Enter weight of rock: 121.3  
Name: iron ore  
Weight: 121.3 |
|---------|--------------------------------------------------|
| Case 2: | Enter the name of a rock: Quartzite  
Enter weight of rock: twenty one  
Invalid input: twenty one  
Enter weight of rock: 21  
Name: Quartzite  
Weight: 21.0 |
| Case 3: | Enter the name of a rock: Limestone  
Enter weight of rock: -0.4  
Weight must be greater than 0.  
Enter weight of rock: eighty five  
Invalid input: eighty five  
Enter weight of rock: 0  
Weight must be greater than 0.  
Enter weight of rock: 1  
Name: Limestone  
Weight: 1.0 |
Exercise D: Quilt Patterns

This exercise is working with two-dimensional arrays, using nested for loops, more practice with writing static methods and passing two-dimensional arrays to methods. Today we'll be making patterns for quilts using two-dimensional (2-D) arrays of characters, and we'll be organizing our code using static methods.

Getting Started

As usual, begin by starting a new project in Eclipse. Name your project QuiltLab and then download the file Quilt.java into your QuiltLab project.

Also recall, the blue boxes in Eclipse can be used to jump to the TODO steps.

Review of 2-D Arrays

For this lab, we'll be using 2-D arrays of characters. For example, you'll make an array of characters like this:

```java
char[][] myArray = { {'x', '.'}, {'x', '.'}, {'x', '.'}, {'x', '.'}, }
```

We'll assume our arrays to be rectangular, that is, every row in the array has the exact same number of columns. Recall, to access:

- the number of rows use: myArray.length
- the number of columns use: myArray[0].length

You've learned that Java uses "arrays of arrays" for multidimensional arrays. Below is a memory diagram to help you visualize this:
myArray references the vertically oriented array whose elements each hold on to a row, one of the horizontally oriented arrays. The code myArray.length is the length of the vertically oriented array, which corresponds to the number of rows in the 2-D array (4 rows above). The code myArray[0].length is the length of array that is row 0, and it corresponds to the number of columns in that row (5 columns above). It also represents the number of columns in every row since all rows are the same length in rectangular arrays. (It is possible for each row to have a different length so generally, myArray[r].length represents the number of columns in row r of myArray.)

Task 1: The Pattern Block

A patchwork quilt can be made by sewing together a number of rectangular blocks that are all the same size. Each block is made up of small pieces of different fabrics sewn together to form a pattern. We can represent a pattern block as a two-dimensional (2-D) array of characters with different characters representing small squares of different fabrics in that block. For example, the pattern block below represents 20 small squares (in 4 rows and 5 columns) of three different fabrics, one fabric represented by the character 'x', another by '+', and the last by '.':

\[
\begin{align*}
nm  \ldots  \\
nm+ \ldots  \\
nm++ \ldots  \\
nmxxxx  
\end{align*}
\]

In this lab you'll be creating quilt patterns from a single pattern block that is stored using a 2-D array of characters. The variable, named myBlock, in the main method, has been declared to hold the characters representing the pattern block. Add the code to make the 2-D array contain the pattern of characters as shown above.
Task 2: Displaying the Pattern Block

It would be useful to be able to see the contents of our pattern block. To do this we'll write a method so that we can use it repeatedly without adding redundancy in our code (a key benefit of using methods). We'll name the method, displayPattern and pass it a 2-D array of characters. We've already included the method header in the code you've downloaded. For this task, fill in the body of the displayPattern method so that it displays on the console window the characters in the array that is passed to the parameter variable myArray.

Verify that your method is working by adding a method call in the main method that calls displayPattern to display myBlock so that you can see what it stores.

**When you believe your method works correctly, show it to your Lab TA.**

Task 3: The Quilt Array

Next we will create a larger 2-D array of characters to represent an entire quilt pattern, which we'll fill with repeated pattern blocks in the next task. For this task, add code to the main method that allocates a 2-D array of characters for the variable, named myQuilt, so that it can hold three rows of pattern blocks with four blocks per row. Make your array large enough to hold all of the characters for the number of pattern blocks mentioned. (Hint: Calculate how many characters in total are needed per row and likewise for how many rows are needed in total, or tell your computer to calculate this for you.)

Next, use nested loops to fill the myQuilt array with with question mark mark characters (?), and then use your displayPattern method to display myQuilt to verify it has been filled with question marks. Your display method, if done correctly, should be able to display this array completely even though it is larger than the pattern block. Here's what the quilt pattern above would actually look like:

```
??????????????????????
??????????????????????
??????????????????????
??????????????????????
??????????????????????
??????????????????????
??????????????????????
??????????????????????
??????????????????????
```

We added lines ('|'s and '-'s) as separators in the output below so you can see the pattern blocks better. These are not included in the quilt patterns that your program displays.

```
???? | ????? | ????? | ?????
???? | ????? | ????? | ?????
???? | ????? | ????? | ?????
???? | ????? | ????? | ?????
----------------------------------------
```
When you have completed this task show your answer to your Lab TA.

Task 4: Creating the Quilt Pattern

The fourth task is to write the createQuilt method to fill in quilt with the correct number of blocks as shown below:

```
x.... | x.... | x.... | x....  
x+... | x+... | x+... | x+...  
x++.. | x++.. | x++.. | x++..  
xxxxx | xxxxx | xxxxx | xxxxx  
--------------------------------------------------
x.... | x.... | x.... | x....  
x+... | x+... | x+... | x+...  
x++.. | x++.. | x++.. | x++..  
xxxxx | xxxxx | xxxxx | xxxxx  
--------------------------------------------------
x.... | x.... | x.... | x....  
x+... | x+... | x+... | x+...  
x++.. | x++.. | x++.. | x++..  
xxxxx | xxxxx | xxxxx | xxxxx  
```

This example shows an array that represents a quilt pattern made of 12 pattern blocks (3 rows of pattern blocks with 4 in each row). To do this task, you are to use the method we've coded, named placeBlock. Your createQuilt method will call the placeBlock method to fill in the quilt parameter with the block. (Hint: this involves figuring out the coordinates of the upper left hand corner of each block you place and will be a lot easier if you sketch it out on paper first!)

Test your createQuilt method by calling it from your main method and also use your displayPattern method to display your results. Caution: When you call createQuilt make sure the first argument is the quilt array and the second is the block array or else you'll have a problem (try the wrong order and see what happens!). Recall that an argument is the information being passed into a method and is listed in the method call. A parameter is the variable where a method stores the value of its arguments and is listed in the method header. For things to work correctly, arguments in the method call must be in the order specified by the parameters in the method's header!

Show your code to your Lab TA.
Task 5: Flipping The Pattern Block

Switch so that your partner is in control of the keyboard

Next, write the method `createFlipped` (we've included its header at the bottom of the class) so that it creates and returns a 2-D array of characters that represents a flipped (upside-down) version of the block that is passed to the parameter variable, named `block`. For example, if `block` contains our example pattern block:

```
x....
x+. .
x++ .
xxxxx
```

then a call to `createFlipped` would create a new array and return it as the flipped version of the one passed in:

```
xxxxx
x++ .
x+ .
x. .
```

(Hint: Note that the last row of the main block becomes the first row of the flipped block, the second-last row of the main block becomes the second row of the flipped block, and so on. In other words, the block is only flipped on the horizontal axis.)

To see whether your `createFlipped` method is correct, modify your `createQuilt` method to call `createFlipped`, then pass the result produced by the `createFlipped` method to `placeBlock`, which should produce a quilt where all of the blocks are now flipped upside-down.

**After you've completed this task, show your Lab TA your output.**

Task 6: A Quilt with Alternating Unflipped and Flipped Blocks

In this last task we'll make a more interesting quilt by alternately flipping the pattern block:

```
x....xxxxxx....xxxxx
x+..x++..x+..x++..
x++..x+..x++..x+...
xxxxxx....xxxxxx....
xxxxxx....xxxxxx....
x++..x+..x++..x+...
x+..x++..x+..x++..
x....xxxxxx....xxxxx
x....xxxxxx....xxxxx
x+..x++..x+..x++..
x++..x+..x++..x+...
xxxxxx....xxxxxx....
```

```java
// Flipping The Pattern Block

public class FlippingPatternBlock {
    public static void main(String[] args) {
        String block = "x....
        x+. .
        x++ .
        xxxx"
        String flippedBlock = createFlipped(block);
        System.out.println("Original:
        " + block);
        System.out.println("Flipped:
        " + flippedBlock);
    }

    public static String createFlipped(String originalBlock) {
        // Implementation...
    }
}
```
Pattern with Blocks Separated by |'s and -'s:

```
|x....|xxxxx|x....|xxxxx|
|x+..|x+++|x+..|x++..|
|x++..|x+..|x+++|x+...
```

Finally, modify the createQuilt method so that it alternates unflipped and flipped blocks as shown in the initial quilt diagram above. You'll be using your createFlipped method with the placeBlock method that we've provided.

(Hint: Think about using one or more boolean variables to keep track of whether the next block should be flipped or unflipped.)

**Show your Lab TA your final result.**

**Task 7:**

If time remains in the lab, try these tasks. **Let your Lab TA know if you finish either of these tasks.**

1. A quick and fun task, is to change the pattern block that you use to create the quilt. This could create a new, perhaps more interesting, quilt. You can use the new block given below, or create your own.

   ```java
   char[][] myBlock2 = {
   {'x','.','.','.','.'},
   {'x','x','.','.','.'},
   {'x','x','x','x','.'},
   {'x','x','x','x','x'}
   };
   ```

   Note that you aren't restricted to using just 'x' and '.' in your block. Here's another block you can try (sometimes known as the "flying geese" block):

   ```java
   char[][] myBlock3 = {
   {'-','-','-','-','-','-','-','-','-','-','-','-','-','-','|'},
   {'|','|','|','|','|','|','|','|','|','|','|','|','|','|'},
   {'|','a','a','a','a','a','a','a','a','a','a','a','a','a','|'},
   {'|','|','|','|','|','|','|','|','|','|','|','|','|','|'},
   {'|','r','r','r','r','r','r','r','r','r','r','r','r','r','|'},
   };
   ```

   ```java
   ```
Make sure that you increase the size of the myQuilt array to accommodate this larger pattern block.

2. Another way to make a quilt from a square pattern block is to rotate the main block by 90 degrees each time you move across a row and also each time you move down a column. For example, if our main block is:

```
#...  
#==.  
###
```

then the quilt we’d get using rotated blocks and creating a quilt with 12 blocks in 3 rows and 4 columns is:

```
#... | #### | #### | ...#  
#==. | #==. | .==# | .==#  
###  | #==. | .==# | .==#  
##  | #... | .==# | .==#  
```

Note that you’ll need to do extra rotations before beginning rows 2 and 3 to ensure that the pattern is rotated each time you move down a column.

Write a new Quilt class method, named createRotated, that is passed a pattern block and a parameter that is the number of degrees to rotate (which should be 0, 90, 180 or 270). It should return a 2-D array of characters that represents the pattern block rotated by that many degrees clockwise.

Now modify your createQuilt method to create the quilt using rotated blocks instead of flipped blocks. You will also need to modify the arrays declared in the main method so that they work with the new square pattern block.
Additional Learning Materials

When you have mastered everything in this lab, then you are welcome to learn from additional learning resources available on the web and beyond this course:
https://cs200-www.cs.wisc.edu/wp/learn-to-program-resources/

*Note: Due to programs and zyBooks being individual work, it is Not appropriate to work on them during the Team Lab.*

Lab designed by Jim Williams, Marc Renault, Quilt Patterns project was created by other Faculty Associates.