Computer Science 145

Lab 3 (10 points)

Due at 11:59 pm the night before your scheduled lab time.
Read all of the instructions. Late work will not be accepted.

Introduction

In this lab you will learn how to create and use 1D arrays, how to generate random numbers using java.util.Random, and how to create and use 2D arrays. You should have enough time to complete this lab during the lab session. In your Git repository for labs, create a directory named *lab3*, place your submission in this directory, and push it to the remote GitHub repository before the deadline. If you have questions, be sure to ask the TA. Ask questions often. Labs are your opportunity to get personalized help!

This lab is to be done on the Linux operating system. You must use Linux to complete this lab – basic knowledge of Linux is one of the course outcomes. If required, you can remotely access the campus Linux lab computers (see https://access.cs.wwu.edu/).

Lab Rules

You are encouraged to collaborate with your peers when completing the labs, but certain limitations apply. You may discuss the lab with a partner and work together to learn how to write Java code. However, the Java code you submit to GitHub must be your own work. You are required to write the code independently without assistance from others.

Part 1: Declaring and Using Arrays

Arrays enable us to declare a collection of values of the same data type and access them using an "index" variable that specifies the element we are interested in. Creating an array has the following syntax:

```
data_type[] array_name = new data_type[array_size];
```

For example, to create an array of 10 integers called `myNumbers`, we can use:

```
int[] myNumbers = new int[10];
```

The values within the array are at indexes 0 to `array_size-1`. In this case, `myNumbers[0]` is the first value, `myNumbers[1]` the second value, etc., and `myNumbers[9]` is the last value. Array indices less than 0 or greater than or equal to `array_size` are "out of bounds" and should not be used.

Steps for Part 1

Consider the following Java program. It declares an array of integers called "data" and uses for loops to initialize the array and print out the array.

```
import java.util.Random;

public class UsingArrayTest {
    public void test() {
        int[] data = new int[10];
        Random random = new Random();

        // Initializes array
        for (int i = 0; i < data.length; i++) {
            data[i] = random.nextInt() % 100;
        }

        // Print array
        for (int i = 0; i < data.length; i++) {
            System.out.printf("%d ", data[i]);
        }

        System.out.println();

        // Reverse array
    }
}
```

1. Open the IDE of your choice.
2. Create a new folder named "lab3".
3. Create a new class named `UsingArrayTest`, write the given code into your program, and run it to see what it prints (remember you will need to add a `main` method, create an instance of `UsingArrayTest`, and invoke its `test` method). You should see 10 random-looking numbers between -99 and 99. These are the 10 values you have stored in the `data` array.
4. You should have noticed that we used the literal value 10 as the size of the array. Edit your program and change the array size to 20, and rerun the program. Now you should see 20 random numbers.
5. Edit your program and copy the "print array" loop to run an extra time before the "initialize array" loop. Now run the program again. You should now see 20 zeros printed on one line, and 20 random values on the next line.
6. Arrays in Java always store their values in locations `[0..size-1]`. If you attempt to read the array beyond these bounds, the program will crash. To verify this, edit your program and change your "print array" loop so it goes from `[0..size+10]` or `[-10..size-1]`. Your program should crash with an "Array Index Out of Bounds" exception. We should never do this on purpose, so change your program back to the original loops to print values `[0..size-1]`.
7. One of the big advantages of arrays is that we can rearrange the data in the array for different purposes. For example, we can reverse the order of the data in the array by swapping the first value with the last value, the second value with the second-to-last value, etc. This can be accomplished with a `for` loop that swaps the values of `data[index]` with `data[size-1-index]`. Edit your program and implement the "reverse array" loop (shown below) at the end of your `test` method, and then put another "print loop" so we can see the result of the reverse. Run the program to see what happens.

```
// Reverse array
for (int i = 0; i < data.length; i++) {
    data[i] = data[data.length - 1 - i];
    data[data.length - 1 - i] = data[i];
}
```

8. There is a logic error in the reverse loop above. We cannot swap the values of two variables `A` and `B` with two statements `A = B;` and `B = A;`. Once we have executed the first statement, the original value of `A` is lost. To solve this, we need a temporary variable `T` and the following: `T = A; A = B; B = T;`. Use this logic to correct your reverse array loop and run your program. What do you see?

9. There is a second logic error in the reverse loop we gave you. As you noticed above, the reversed data ended up looking like the original data. This is because we ended up swapping each pair of array locations two times. Why? When `i=0`, we swap the first and last locations. Then when `i=array.length-1`, we swap the first and last locations again. To fix this, you simply need to stop the `for` loop when `i < data.length/2`. Make this change to your program and rerun. Now you should see the data in reverse order. Congratulations on a job well done!

10. Once you think your program is working correctly, move on to the next part.

Part 2: Using Arrays in Methods

In the previous part, we saw that Java arrays can be used to store and process a group of variables of the same data type. To declare an array parameter in a method, we must specify the data type and the name of the array. The special symbol `[]` is placed after the name to indicate that the parameter is an array. For example, we could declare the `process` function using the following:

```
public void process(double values[]) {
    ...
}
```

To call a method with an array parameter, we must give the name of the array we want to pass in, and the size of the array. For example, in the `main` method, we could send `myNumbers` into the `process` method using:

```
process(myNumbers);
```

Arrays in Java are unusual because they can only be passed by reference. Hence, any changes to the parameter `values` in the method above would really change the array `myNumbers` we declared in the calling method. The remainder of this lab will focus on using methods to process data in arrays.

Steps for Part 2

Consider the following Java program. It declares an array of integers called "data" and calls two methods to initialize the array and print out the array.

```
import java.util.Random;

public class ArraysInMethods {
    public void run() {
        int[] data = new int[10];

        initArray(data, 100);
        printArray(data);
    }

    public void initArray(int[] array, int range) {
        Random random = new Random();

        for (int i = 0; i < array.length; i++) {
            array[i] = random.nextInt() % range;
        }
    }

    public void printArray(int[] arrayToPrint) {
        for (int i = 0; i < arrayToPrint.length; i++) {
            System.out.printf("%d ", arrayToPrint[i]);
        }
        System.out.println();
    }
}
```

1. Create a new class named `ArraysInMethods`, write the given code into your program, and run it to see what it prints (remember to create an instance of `ArraysInMethods` and invoke its `run` method in the `main` method). You should see 10 random-looking numbers between -99 and 99. These are the 10 values you have stored in the `data` array.

2. If you look at the program, you will see that we have two methods `initArray` and `printArray` that take an integer array as a parameter. They contain `for` loops from the previous part of this lab to initialize and print the input array. These methods have been written so they can handle arrays of any size. To test this, edit your program and change the size of the `data` array to 20 and rerun the program. You should see 20 random values.

3. If an array contains data like test scores or product prices, it is quite natural to want to know the max value and/or the min value. Write the following code just at the end of the `run` method, and run the program. It should print the max value in your `data` array.

```
int max = data[0];

for (int i = 0; i < data.length; i++) {
    if (data[i] > max) {
        max = data[i];
    }
}

System.out.printf("max = %d%n", max);
```

4. The loop above is so handy, it would be nice to have a method to do this work for us. Look at the other two methods to see how we pass array parameters into a method, and create a new method called `findMax`. Move the code to find the max value (not print) from the `run` method into your new

`findMax` method. When you do this, you may notice that several variables need to be renamed. Fix any syntax errors you encounter. Modify the code so that your method returns the max value.

5. Once you get the `findMax` method completed, edit your `run` method to remove the max loop, and replace it with a call to your new `findMax` method. Be sure to pass parameters into the method in the same order as your method declaration. Otherwise, you will get all sorts of `type mismatch` errors. When you run your program, you should see the 20 random values and the max value found by your method.

6. Now that we know how to find the max value in an array using a method, let us use this as a template and create a new `findMin` method. Use your `findMax` method as a template. You will need to change all `max` variables to `min` and adjust the `if` statement accordingly. Fix any syntax errors you encounter.

7. To test `findMin`, edit your `run` method and add a call to `findMin` just below your call to `findMax` and a statement to print the returned value. When you run your program now, you should see 20 random values, the max value, and the min value.

8. Once you think your program is working correctly, move on to the next part.

Part 3: 2D Arrays

The whole purpose of an array is to store multiple values of the same data type together in the same portion of memory. There are many applications where it is helpful to visualize this data in two or more dimensions. For example, we typically access and process data in a spreadsheet using row and column coordinates. Similarly, operations to manipulate digital images are often done using row and column coordinates. In Java, creating a 2D array has the following syntax:

```
data_type[][] array_name = new data_type[num_rows][num_columns];
```

For example, to create a 480-row by 640-column array called `pixel`, use the following:

```
int[][] pixel = new int[480][640];
```

To access the data in a 2D array, we must specify the array name and a row value between `[0..num_rows-1]` and a column value between `[0..num_columns-1]`. Because we now have two array indices, it is quite common to use two nested loops to process a 2D image. For example, the following code will initialize the `pixel` array:

```
for (int r = 0; r < 480; r++) {
    for (int c = 0; c < 640; c++) {
        pixel[r][c] = 42;
    }
}
```

Any attempt to access a 2D array using a row or column index outside the `[0..num_rows-1]` or `[0..num_columns-1]` range will cause an exception to occur at runtime.

Steps for Part 3

Consider the following Java program. It uses a 2D String array "`board`" and a variety of methods to implement a very simple "tic-tac-toe" game. The program reads a sequence of x's and o's from the user, displays them on the tic-tac-toe board, and then checks to see if x or o has three in a row anywhere. If so, the program prints who the winner is.

```
import java.util.Scanner;

public class TicTacToe {
    private String[][] board;

    public TicTacToe() {
        board = new String[3][3];
    }

    public void run() {
        final String INVALID_BOARD_MESSAGE = "Sorry, you can only enter x's and o's.";

        initBoard();
        printBoard();

        if (!isBoardValid()) {
            System.out.println(INVALID_BOARD_MESSAGE);
            return;
        }

        determineWinner();
    }

    private void initBoard() {
        final String INIT_BOARD_MESSAGE = "Enter x's and o's on board (L-R, T-B): ";

        System.out.print(INIT_BOARD_MESSAGE);

        Scanner input = new Scanner(System.in);

        for (int r = 0; r < 3; r++) {
            for (int c = 0; c < 3; c++) {
                board[r][c] = input.next();
            }
        }
    }

    private void printBoard() {
        final String BOARD_BORDER = "\n+---+---+---+\n";
        final String ROW_START = "| ";
        final String ROW_DIVIDER = "%s | ";

        System.out.print(BOARD_BORDER);

        for (int r = 0; r < 3; r++) {
            System.out.print(ROW_START);
            for (int c = 0; c < 3; c++) {
                System.out.printf(ROW_DIVIDER, board[r][c]);
            }
            System.out.print("\n" + BOARD_BORDER);
        }
    }
```

```
    private boolean isBoardValid() {
        // Check board contains only x's and o's
        // To Be Implemented
        return true;
    }

    private void determineWinner() {
        final String WINNER_MESSAGE = "Congratulations %s is the winner!\n";
        final String NO_WINNER_MESSAGE = "Sorry, no one wins.";

        String winner = null;

        // Check first diagonal to see who wins
        if (board[0][0].equals(board[1][1]) &&
            board[1][1].equals(board[2][2])) {
            winner = board[0][0];
        }

        // Check second diagonal to see who wins
        // To Be Implemented

        // Check rows to see who wins
        for (int r = 0; r < board.length; r++) {
            if ((board[r][0].equals(board[r][1])) &&
                (board[r][1].equals(board[r][2]))) {
                winner = board[r][0];
            }
        }

        // Check columns to see who wins
        // To Be Implemented

        // Print winner
        if (winner != null) {
            System.out.printf(WINNER_MESSAGE, winner);
        } else {
            System.out.println(NO_WINNER_MESSAGE);
        }
    }
}
```

1. Create a new class named `TicTacToe` and write the given code into your program. Also, create an instance of `TicTacToe` and invoke its `run` method in the `main` method.

2. Run your program and type in `terrible!` one character at a time (followed by enter). You will see the 3 by 3 tic-tac-toe board displayed with the characters above going left to right (L-R) and top to bottom (T-B) in the array. The program says there is no winner, but it really should say that the board was invalid because we did not enter x's and o's.

3. Modify your program and implement the `isBoardValid` method. Replace the "To Be Implemented" comment with the correct code to check all nine array locations to make sure the user entered in only x's and o's (lower case). If they type in any other character, you should return `false`. Otherwise, return `true`. When you have this code finished, test it with `terrible!` to verify correctness.

4. Now run the program and type in `xoo oxo xox` to initialize the tic-tac-toe array. The program should tell you that x is the winner. If you check the code, you will see that there is a section of code in `determineWinner` that checks that array locations `[0][0]`, `[1][1]`, and `[2][2]` on the first diagonal all have the same value. Because the board has an x in all three of these locations, the program declares x to be the winner.

5. Modify your program to add some code in `determineWinner` to check the second diagonal direction on the tic-tac-toe board to see if there is a winner. When you think you have it right, run the program and type in `oxo xoo oxx`. You should see that o is the winner this time.

6. Run your program again and type in `oxo xox xxx`. You should see that x is the winner. In this case, the program found three values on one row that were equal to x. Take a look at the `for` loop in `determineWinner` for checking rows, and you will see that we loop over rows 0, 1, 2, and then check that all three values match each other.

7. You have probably noticed that the code to check columns for the winner is missing. Modify the `determineWinner` method to add the necessary code to check all three columns. Make up some test data to verify that this is working. Once you have this going, you will have a simple tic-tac-toe game that will let you check who wins/loses after the board is all filled in. There is clearly much more needed to implement an interactive game, but we will leave that for another time.

8. Once you think your program is working correctly, look over the grading rubric and submission instructions to submit your lab.

Submission

Once you have finished all of the above, add, commit and push the following files to your course repository main branch:

- `UsingArrayTest.java`
- `ArraysInMethods.java`
- `TicTacToe.java`

Note that these files should be directly within a `lab3` subdirectory of your repository (spelling, spacing, and capitalization matter).

Rubric

Criteria	Points
Arrays are correctly initialized and manipulated, including reversing.	3
Methods `findMax` and `findMin` are correctly implemented and functional.	3
Tic-tac-toe functionality is complete, including valid input checks and winner determination.	3
Programs follow good programming style and are submitted correctly.	1
Total	**10**

Acknowledgments

Thanks are owed to Wesley Deneke for the lab on which this lab is modeled.