ECE25100 Object Oriented Programming
Lab 5: FOR Loops and Arrays

Description:
The purpose of this lab is to help you understand how to use arrays and FOR loops.

To get credit for the lab, you need to demonstrate to the instructor/lab assistant that you have completed all the requirements, and sign your name on the grade sheet.

QUESTION 1: Recall that arrays can store multiple types of objects (or primitives) by using a single variable which can be used to access any of the data elements. Copy/paste the code below into a file called **ArrayTestProgram1.java**.

```
public class ArrayTestProgram1 {
    public static void main(String args[]) {
        int    total = 0;
        int[]  numbers;

        System.out.println("The array looks like this: " +
       numbers);
       System.out.println("It has " + numbers.length + " elements in
       it");
       System.out.println("The 5th element in it is: " + numbers[4]);
    }
}
```

Compile the code. Notice that it complains, stating that variable numbers might not have been initialized. Of course, you can easily fix this by changing **int**[] numbers; to **int**[] numbers = **null**; Do this, and then re-compile and run the code.

Now you will notice that the array displays as **null** and that there is a NullPointerException on line 7 (line number may differ from yours). Do you know why? We are using **numbers.length** which attempts to access the **length** attribute of the **numbers** array. But the **numbers** variable is currently empty ... there is no array there. You cannot access attributes of (nor send messages to) **null**. So, you need to create an array.

You can try changing **int**[] numbers = **null**; to one of the lines below. Only 1 will work ... do you know which one ?

```
int[] numbers = new Array[];
int[] numbers = new Array[0];
int[] numbers = new Integer[];
int[] numbers = new Integer[0];
int[] numbers = new int[];
int[] numbers = new int[0];
```

Once you figure out which line will compile, run the code. You will notice that the code generates an ArrayIndexOutOfBoundsException when we attempt to ask for the 5th element. Do you understand why ? It is simple. The array has been created, but it is an empty array of size zero. In fact, we need to create the array with a reasonable size limit, such as 12. Change the dimension of the array from **0** to **12**, then recompile and run the code.

Now the code will run and it tells us that the array has 12 elements in it and that the 5th element is **0**. In fact, all of the 12 elements are **0** because when we create a new array, the values are all set to zero by default. We should place some actual numbers in the array. Insert the following code (copy/paste it) just after the line that creates the array:

```
numbers[0] = 1;
numbers[1] = 45;
numbers[2] = 23;
numbers[3] = 87;
numbers[4] = 89;
numbers[5] = 213;
```

Now recompile and run the code. You will notice that element 5 is now **89** (notice that it is not **213** because array indexing always starts at 0 instead of 1). Interestingly, we only placed 6 numbers in the array, but it is telling us that the length is 12. Do you know why ? The 12 represents the size "limit" of the array, and in fact, the array does contain 12 actual integers... half of them are zero in this case. So then, how can we tell how many numbers we actually placed in the array? We would have to keep track of this number on our own using another variable as a counter.

Of course, if we already know the integers that we want to use, we can hard-code them into the array. Replace all of these lines of code:

```
int[]  numbers = new int[12];
numbers[0] = 1;
numbers[1] = 45;
numbers[2] = 23;
numbers[3] = 87;
numbers[4] = 89;
numbers[5] = 213;
```

with this single line:

```
int[]  numbers = {1, 45, 23, 87, 89, 213, 54, 11, 76, 98, 23, 5};
```

Then, recompile the code. There should be no change in the output. Notice that the length of the array will equal the number of numbers that you supplied.

Replace the 3 **System.out.println** lines of code by some code that will compute and displays the total sum of the integers in the **numbers** array. Use a FOR loop. The total should be 725.

QUESTION 2: Now copy/paste the following into a file called **ArrayTestProgram2.java**. In a similar manner as you did above, complete the code so that it displays **true** if the majority of booleans in the **values** array are **true** and displays **false** otherwise. Make sure that your code works for any **boolean** array. A **count** variable is provided. You can use it by adding 1 to the count when a **true** is encountered and subtracting 1 otherwise. Do you know how to use the final **count** to determine the answer?

```
public class ArrayTestProgram2 {
    public static void main(String args[]) {
        int        count = 0;
        boolean[]  values = {true, false, false, true, true, true,
false};
    }
}
```

QUESTION 3:

STEP 1: Download the **Team** and **League** classes. Also download a test program (i.e., **LeagueTestProgram.java**) that you can use to test your methods. Compile all 3 files.
Now write a method called **teamMostPlayed** in the **League** class which returns a **Team** object representing the team that has played the most games. Compile your code. Add the following code to the **LeagueTestProgram** to test your code:

```
// Find the team that played the most
System.out.println("\n\nThe team that played the most is " +
aLeague.teamMostPlayed());
```

Run the test.

STEP 2: Write a method called **undefeatedTeams** in the **League** class that returns an array of all undefeated teams (i.e., actual **Team** objects) in the league. A team is *undefeated* if it has never lost a game. Since arrays cannot grow, you may have to first count all of the undefeated teams and then make an appropriate-sized array and fill it up. Compile your code. Add the following code to the **LeagueTestProgram** to test your code:

```
// Find the teams that are undefeated
Team[] undefeated = aLeague.undefeatedTeams();
System.out.println("\nThere are " + undefeated.length + " undefeated
teams as follows:");
if (undefeated.length > 0) {
    for (int i=0; i<undefeated.length; i++) {
        System.out.println("  " + undefeated[i]);
     }
}
```

Run the test code.

STEP 3: Now assume that we want to determine whether one **Team** has played another one. Currently, we cannot do this without changing the **Team** and/or **League** class definitions. Do you know why?

In the **League** class, add the following attribute (i.e., instance variable):

```java
private int[][] gamesPlayed;
```

This 2-dimensional array will represent a table showing which teams played each other in the **League**.
In the **League**'s constructor, set **gamesPlayed** to be **new int[8][8];** We will assume that each **Team** in the **League** has an "ID" which is its index in the **teams** array. Add the following method to the **League** class which will return the index of a specific team in the league (i.e., its ID):

```java
public int indexOf(Team t) {
    for (int i=0; i<teams.length; i++) {
        if (teams[i] == t)
            return i;
    }
    return -1;
}
```

Also, add the following method which will display a table of all the games that were played in the **League** up to this point:

```java
//This method prints out a table showing which teams played which
other teams
public void showTable() {
    System.out.println("    0   1   2   3   4   5   6   7 ");
    System.out.println("   -------------------------------");
    for (int row=0; row<8; row++) {
        System.out.print(row + " | ");
        for (int col=0; col<8; col++) {
            System.out.print(gamesPlayed[row][col] + " | ");
        }
        System.out.println("\n   -------------------------------");
    }
    for (int i=0; i<8; i++)
        System.out.println(i + " = " + teams[i].getName());
}
```

Add the line **aLeague.showTable();** to the **LeagueTestProgram** in order to test it. Run your code and make sure that the output of this table looks as follows:

```
    0   1   2   3   4   5   6   7
   -------------------------------
0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
   -------------------------------
1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
   -------------------------------
2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
   -------------------------------
3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
   -------------------------------
4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
   -------------------------------
5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
   -------------------------------
```

```
6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
   -------------------------------
7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
   -------------------------------
0 = Chicago Bulls
1 = Detroit Pistons
2 = New Jersey Nets
3 = Toronto Raptors
4 = New York Knicks
5 = Atlanta Hawks
6 = Boston Celtics
7 = Indiana Pacers
```

Append some code to the bottom of the following two methods so that the table is updated after each game is played:

```
public void recordWinAndLoss(Team winner, Team loser) { ... }
public void recordTie(Team teamA, Team teamB) { ... }
```

Every time a team with index X plays another team with index **Y**, you should increase the value of the table at row X, column **Y** as well as the value at row **Y**, column X. So, each game played should increase exactly two values in the array. Once you complete this code, compile it and then run the test again. You should see the following values in the table:

```
    0   1   2   3   4   5   6   7
   -------------------------------
0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 2 |
   -------------------------------
1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
   -------------------------------
2 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
   -------------------------------
3 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
   -------------------------------
4 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
   -------------------------------
5 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
   -------------------------------
6 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
   -------------------------------
7 | 2 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
   -------------------------------
```

STEP 4: Now write a method called **havePlayedTogether(String teamA, String teamB)** in the **League** class that returns a **boolean** indicating whether or not the teams with the given names have played together or not. You should make use of the **gamesPlayed** array that you just filled in. Once completed, append the following code to the **LeagueTestProgram** and compile/run it:

```
System.out.println(aLeague.havePlayedTogether("Chicago Bulls", "Boston
Celtics"));
System.out.println(aLeague.havePlayedTogether("New York Knicks",
"Detroit Pistons"));
System.out.println(aLeague.havePlayedTogether("Detroit Pistons",
"Atlanta Hawks"));
System.out.println(aLeague.havePlayedTogether("Chicago Bulls", "New
York Knicks"));
System.out.println(aLeague.havePlayedTogether("Atlanta Hawks", "New
York Knicks"));
System.out.println(aLeague.havePlayedTogether("Indiana Pacers",
"Boston Celtics"));
System.out.println(aLeague.havePlayedTogether("Toronto Raptors", "New
Jersey Nets"));
System.out.println(aLeague.havePlayedTogether("New Jersey Nets", "New
York Knicks"));
System.out.println(aLeague.havePlayedTogether("Boston Celtics",
"Detroit Pistons"));
System.out.println(aLeague.havePlayedTogether("Detroit Pistons", "New
Jersey Nets"));
System.out.println(aLeague.havePlayedTogether("I Love 2017", "Frank"));
System.out.println(aLeague.havePlayedTogether("Chicago Bulls",
"Chicago Bulls"));
```

The resulting output should be **true** and **false** values as following:

false
false
false
true
false
true
true
false
false
false
false
false