Lab Partner 1 Name

Lab Partner 2 Name

CS 115 Fall 2019 Lab #9

Due: **Monday, November 18th, 5:00 PM**

Points: **20**

Instructions:

1. Use this document template to report your answers and create separate java files for your classes. Enter all lab partner names at the top of first page.

2. You don't need to finish your lab work during the corresponding lab session.

3. ZIP your Java files and lab report into a single file. Name the file as follows:

```
LastName_FirstName_CS115_Lab9_Report.zip
```

4. Submit the final document to Blackboard Assignments section before the due date. No late submissions will be accepted.

5. ALL lab partners need to submit a report, even if it is the same document.

Objectives:

1. (6 points) Demonstrate the ability to use while loops.

2. (7 points) Design, code and test a user-defined object containing an array.

3. (7 points) Demonstrate the ability to read a text file using loops and Java Scanner class.

Problem 1 [6 points]:

Write a program to play the game of craps (without the betting). In this game, two die are first rolled (**use the** `Die` **class provided below for that purpose**) to determine the roller's target. If you roll a sum of 7 on the target roll, you win. Otherwise, the roller

keeps rolling the two die to try to match the target sum before he rolls a sum of 7 (and craps out).

Java `Die` **class**:

```
public class Die {
    private int side;

    public Die()
    {
        setSide(1);
    }

    public Die(int newSide)
    {
        setSide(newSide);
    }

    public int getSide()
        { return side; }

    public void setSide(int newSide)
        { side=newSide; }

    public void roll()
    {
        side = (int)(Math.random()*6+1);
    }

    public String toString( )
      {
        return "Die=" + side;
      }
}
```

Sample Plays:

Sample plays:

```
Target Roll: 6 + 1 = 7
You won on the first roll!
Press any key to continue . . .

Target Roll: 2 + 3 = 5
You rolled: 1 + 2 = 3
You rolled: 4 + 6 = 10
You rolled: 3 + 3 = 6
You rolled: 2 + 1 = 3
You rolled: 4 + 4 = 8
You rolled: 6 + 2 = 8
You rolled: 1 + 2 = 3
You rolled: 6 + 6 = 12
You rolled: 5 + 6 = 11
You rolled: 3 + 5 = 8
You rolled: 2 + 5 = 7
CRAPS! You lost.
Press any key to continue . . .

Target Roll: 4 + 4 = 8
You rolled: 5 + 5 = 10
You rolled: 3 + 6 = 9
You rolled: 1 + 4 = 5
You rolled: 5 + 3 = 8
You rolled your target. You won!
Press any key to continue . . .

Target Roll: 4 + 5 = 9
You rolled: 4 + 4 = 8
You rolled: 5 + 3 = 8
You rolled: 2 + 1 = 3
```

```
You rolled: 6 + 2 = 8

You rolled: 6 + 5 = 11

You rolled: 5 + 4 = 9

You rolled your target. You won!

Press any key to continue . . .
```

Your task is to:

- Create a `Craps` class (containing your "main" program) from the description given above.
- Compile and run your program to see if it runs (no run-time errors),

Solution: `Craps` **class**

```
public class Craps {
    public static void main(String[] args) {

        Die die1 = new Die();
        Die die2 = new Die();
        Scanner input = new Scanner(System.in);
        die1.roll(); die2.roll();
        int side1=die1.getSide();
        int side2=die2.getSide();
        int target = side1+side2;
        int current=0;
        System.out.println("Target Roll: " + side1 +
            " + " + side2 + " = " + target );

        if (target==7)
            System.out.println("You won on the first roll!");
        else
        {
            while (current!=7 && current != target)
            {
                die1.roll(); die2.roll();
```

```
                side1=die1.getSide();
                side2=die2.getSide();
                current = side1+side2;
                System.out.println("You rolled: " + side1 +
                        " + " + side2 + " = " + current);
            }

            if (current==7)
                System.out.println("CRAPS! You lost.");
            else
                System.out.println("You rolled your target. You won!");
        }
    }
}
```

Problem 2 [7 points]:

Write a `DailySales` class to store a collection of a company's daily sales records for a single month (up to 31 days). **All daily sales are of integer data type**. You **do not have to worry about verifying the number of days actually in the month**. Please code a class with the following methods:

- `DailySales()` - default constructor,
- `DailySales (int daysInMonth)` - constructor,
- `public boolean addSales(int dayNumber, int sales)` - add "sales" to the current sales for "dayNumber". Return `true` of successful, else return `false` (if invalid sales amount or invalid dayNumber),
- `public int maxDay()` - return the day number with the maximum sales,
- `public int[] daysBelowGoal()` - return an array of day numbers that have less than 100 units sold,

Your tasks:

- Provide basic design of your program (define the fields/attributes and methods, include data type and valid ranges for attributes, and access control, arguments and return types for methods) in the box below **[1 out of 7 points]**:

Program design:

- Complete the Test Plan table below (the number of test cases is up to you, but should be fairly exhaustive) **[1 out of 7 points]**:

Test plan			
Test case	**Sample data**	**Expected result**	**Verified?**

- Code your program **[5 out of 7 points]**.

Solution

2. Design, code and test a user-defined object containing an array of objects.

2a. (1 point) Write a class design (define the attributes and methods, include data type and valid ranges for attributes, and access control, arguments and return types for methods) for the below problem.

attributes - dailysales (array of ints, initialize to zero, all postiive)

class constants - integers maxDays=31, dailyGoal=100

methods defined in lab writeup

2b. (1 points) Write a test table (with columns Test Case Reason, Sample Data, Expected Results) for the below problem.

test default constructor

test nondefault constructor (ok argument and bad argument)

test addSales (good day and good sales count)

test addSales (bad day and good sales count)

test addSales (good day and bad sales count)

test addSales (bad day and bad sales count)

test maxDay (unique max)

test maxDay (duplicate max)

test daysBelowGoal (no days below goal)

test daysBelowGoal (one day below goal)

test daysBelowGoal (all days below goal)

2c. (5 points) Code the class and test program (use your design and test table).

```
public class DailySales {
    private int [] data;
    private static final int MAX_SIZE=31, DAILY_GOAL=100;
    public DailySales() {
        data = new int[MAX_SIZE];
    }
    public DailySales(int daysInMonth) {
        if (daysInMonth>0)
            data = new int[daysInMonth];
        else data = new int[MAX_SIZE];
    }
    public boolean addSales(int dayNumber, int sales) {
        if (dayNumber<1 || dayNumber>data.length || sales<=0)
            return false;
        else {
```

```
            data[dayNumber-1]=data[dayNumber-1]+sales;
            return true;
        }
    }
    public int maxDay() {
        int maxSoFarDay=0;
        for(int i=1; i<data.length; i++)
            if (data[i]>data[maxSoFarDay]) maxSoFarDay=i;
        return maxSoFarDay+1;
    }
    public int[] daysBelowGoal() {
        int count=0;
        for (int i=0; i<data.length; i++)
            if (data[i]<DAILY_GOAL) count++;

        int [] belowGoalDays = new int[count];
        count=0;
        for (int i=0; i<data.length; i++)
            if (data[i]<DAILY_GOAL) {
                belowGoalDays[count]=i+1;
                count++;
            }
        return belowGoalDays;
    }
}
/* ALTERNATE METHOD USING EXTRA PRIVATE VARIABLE
    private int maxSoFarDay;    // add private
    // update addSales
    public boolean addSales(int dayNumber, int sales) {
```

```
            if (dayNumber<1 || dayNumber>data.length || sales<=0)
                return false;
            else {
                data[dayNumber-1]=data[dayNumber-1]+sales;
                if (data[dayNumber-1])>data[maxSoFarDay])
maxSoFarDay=dayNumber-1;
// ADD PREVIOUS LINE
                return true;
            }
        }
        public int maxDay() { return maxSoFarDay+1; }
*/
```

Problem 3 [7 points]:

Finding duplicate data in a sorted file is the first step to removing duplicates. Given an input file with each line representing a record of data and the first token (word) being the key that the file is sorted on, we want to load it and output the line number and record for any duplicate keys we encounter. Remember we are assuming the file is sorted by the key and we want to output to the screen the records (and line numbers) with duplicate keys.

Download `input1.txt` and `input2.txt` files from Blackboard (see Lab files section). Note: the input text files must be in the same directory as your program. Use `Scanner` class to load them.

Sample run

`Enter File Name: input1.txt`
`FileName:input1.txt`
`DUPLICATES`
`12 102380 CS US W 2.8 3.267 125`
`14 102395 PPCI US W 2.769 2.5 115`
`25 102567 PPCI US W 3.192 3.412 112`

```
35 102912 CS US Z 3.81 3.667 88
44 103087 CS US Z 2.956 2.688 90
76 103944 CS US W 3.134 3.294 134
77 103944 CS US W 3.698 3.7 94
86 104046 CS US W 2.863 3.133 65
88 104047 CS US W 3.523 3.524 77
89 104047 CS US O 3.825 3.824 49
91 104048 CS US W 3.071 3 94
92 104048 CS US W 3.114 3.111 44
93 104048 CS US W 3.375 3.6 71

Press any key to continue . . .
```

Your task is to

- create a `FindDuplicates` class with the following:
  - Declaration of an instance variables for the String filename
  - non-default Constructor - creates an object for user passed filename argument
  - Accessor methods return the value of each instance variable
  - Mutator methods that allows th user to set each instance variable (no validation required),
  - a "getDuplicates()" method that reads from the file (until end-of-file) using Scanner class, finds duplicate records based on the first token on each line (the key), and returns as a String the record number and entire duplicate record one to a line (see above Sample output)
  - toString() - returns a String message with the value of the instance variable
- Create a `FindDuplicatesApp` driver class / program to test your `FindDuplicates` class.
- Compile and run your program to see if it runs (no run-time errors).
-

Solution: `FindDuplicates` **class**

```
import java.util.Scanner;
import java.io.File;
import java.io.IOException;

public class FindDuplicates {
    private String fileName;

    public FindDuplicates(String f)
    {
        setFileName(f);
    }

    public String getFileName()
    {
        return fileName;
    }

    public void setFileName(String f)
    {
        fileName = f;
    }

    public String getDuplicates() throws IOException
    {
        int count=0;
        String prevKey="", key, record, duplicates="";
        File input = new File(fileName);
        Scanner in = new Scanner(input);
        while(in.hasNext())
        {
            key=in.next();
            record=in.nextLine();
            count++;
```

```
            if (key.equals(prevKey))
                duplicates = duplicates + count + " " + key + record +"\n";
            else
                prevKey=key;
        }

        return duplicates;
    }

    public String toString()
    {
        return "FileName:"+fileName;
    }
}
```

■

Solution: `FindDuplicatesApp` **class**

```
import java.io.IOException;
import java.util.Scanner;

public class FindDuplicatesApp {
    public static void main(String[] args) throws IOException
    {
        String fileName;
        Scanner scan = new Scanner( System.in );
        FindDuplicates a;

        System.out.print( "Enter File Name: ");
        fileName=scan.next();
        a = new FindDuplicates(fileName);
        System.out.println(a);
        System.out.println("DUPLICATES\n" + a.getDuplicates());
```

```
        }

}
```