University of Maryland College Park
Department of Computer Science
CMSC131 Fall 2025
Exam #2

FIRSTNAME, LASTNAME (PRINT IN UPPERCASE): KEY

STUDENT ID (e.g. 123456789):

YOU MAY LOSE POINTS IF EXAM IS MISSING YOUR FULL NAME IN ALL CAPS AND ID # ON FIRST PAGE (SEE ABOVE). ALSO, YOU MAY LOSE POINTS IF EXAM IS MISSING YOUR DIRECTORY ID ON BOTTOM OF ODD PAGES. DO THIS AS SOON AS TIME STARTS. WE WILL NOT WAIT AT THE END FOR YOU TO WRITE THIS INFORMATION DOWN.

Instructions

- Please print your answers and use a pencil.
- Do not remove the staple from the exam. Removing it will interfere with the Gradescope scanning process.
- To make sure Gradescope can recognize your exam, print your name, write your directory id at the bottom of pages with the text DirectoryId, provide answers in the rectangular areas provided, and do not remove any exam pages. Even if you use the provided extra pages for scratch work, they must be returned with the rest of the exam.
- This exam is a closed-book, closed-notes exam, with a duration of 50 minutes and 100 total points.
- Your code must be efficient.
- Multiple choice questions have only one answer unless indicated otherwise.
- You don't need to use meaningful variable names; however, we expect good indentation.

Grader Use Only

#1	Problem #1 (Short Answers - 2pts each)	16
#2	Write a method	20
#3	Write a class	64
Total	Total	100

Problem #1 (Short Answers – 2 pts each)

1. A **`static`** field belongs to the class rather than any individual object.

2. The **`this`** keyword can be used to call another constructor **in the same class**.

3. A method that has the **same name** as the class and **no return type** is called a **constructor**.

4. Which statement is **true**?
   a. A local **`int`** variable is automatically given a default value of 0.
   b. **A class can define multiple constructors as long as their parameter lists differ.**
   c. The keyword **`new`** can only be used when declaring an array.
   d. A **`continue`** statement can only be used in **`for`** loops.

5. Which statement is **false**?
   a. The **`this`** keyword can be used inside a static method to refer to the current object.
   b. A **`do-while`** loop executes its body at least once.
   c. The **`continue`** statement skips to the next iteration of a loop.
   d. A **`String`** in Java can be assigned the value **`null`**.

6. Will the following code compile? If yes, write the output. If not, explain why?

```
int[] arr = {5, 6, null};
System.out.println(arr[0] + arr[1]);
```

Will not compile because null is not a valid value to assign to a primitive int element.

7. Assume the following code fragment in a **`main`** method, what is the output (might be multiple lines)?

```
int total = 0;
for (int i = 10; i <= 15; i++) {
      total += (i % 2 == 0) ? i : 0;
      if (total > 13)
            break;
}
System.out.println(total);
```

```
22
```

8. Assume the following code fragment in a **`main`** method, what is the output (might be multiple lines)?

```
int n = 4;
try {
    n = n / (n - 4);
    System.out.println("Try");
} catch (ArithmeticException e) {
    System.out.println("Catch");
} finally {
    System.out.println("Finally");
}
```

```
Catch
Finally
```

Problem #2 (Write a method – 20 pts)

Complete the code for the method below. This method takes in an `int[]` parameter. You may **assume the array has at least one element**. The method should **swap each even index with the next odd index** in the original array. If the array has an odd length, leave the last element unchanged. The method should **return a copy of the array before it was modified**. You may **only use one `for` loop**. The only thing allowed from the Java library is the `length` field of arrays. You will lose significant points if you use any other library methods. Making a new array is fine. Assume **`import java.util.Arrays;`** needed for the **`main`** only.

```
public class ArrayQuestion {
    public static int[] evenOddSwap(int[] nums) {

        int[] copy = new int[nums.length];

        for (int i = 0; i < nums.length; i += 2) {
            // Save original values before swap
            copy[i] = nums[i];

            if (i + 1 < nums.length) {
                copy[i + 1] = nums[i + 1];

                // Swap in the original array
                int temp = nums[i];
                nums[i] = nums[i + 1];
                nums[i + 1] = temp;
            }
        }

        return copy;
    }
```

```
public static void main(String[] args) {
        int[] data1 = {10, 20, 30, 40, 50};
        int[] data2 = {5, 8, 2, 9, 1, 7};
        int[] return1 = evenOddSwap(data1);
        int[] return2 = evenOddSwap(data2);
        System.out.println(Arrays.toString(return1));//[10, 20, 30, 40, 50]
        System.out.println(Arrays.toString(data1));//[20, 10, 40, 30, 50]
        System.out.println(Arrays.toString(return2));//[5, 8, 2, 9, 1, 7]
        System.out.println(Arrays.toString(data2));//[8, 5, 9, 2, 7, 1]}}
```

Directory id:

Problem #3 (Write a class)

Complete the implementation of a class called `TempLog` that represents the daily high temperatures for a city. You **MAY NOT** add any methods other than the ones specified below (not even `private`). Also, you **MAY NOT** add any additional instance or static variables. Local variables inside your methods are allowed. You may only use the following library method as needed: `Math.abs(int a)`. Creating arrays, using `this`, and using the `length` field of arrays is always allowed.

```
import java.util.Arrays;

public class TempLog {
    private String city;
    private int[] temps;
    private static int threshold = 5; // degrees
    public TempLog(String city, int[] temps) {
            //answer to P1
    }
    private double average() {
            //answer to P2
    }

    public static TempLog higherAverage(TempLog t1, TempLog t2) {
          //answer to P3
    }

    public static void setThreshold(int newThreshold) {
           //answer to P4
    }

    public boolean equals(Object obj) {
        if (obj == this)
                return true;
        if (obj == null || getClass() != obj.getClass())
                return false;
          //answer to P5
    }


    public String toString() {
        return city + " " + Arrays.toString(temps);
    }

    public static void main(String[] args) {
        int[] temps1 = {70, 75, 80, 85};
        int[] temps2 = {68, 74, 82, 90};
        TempLog a = new TempLog("Terp City", temps1);
        TempLog b = new TempLog("Terpsville", temps2);

        System.out.println("Higher avg: " + TempLog.higherAverage(a, b));
        System.out.println("a equals b? " + a.equals(b));

        TempLog.setThreshold(1);
        System.out.println("a equals b (new threshold)? " + a.equals(b));
    }
}
```

Output

```
Higher avg: Terpsville [68, 74, 82, 90]
a equals b? true
a equals b (new threshold)? False
```

1. This constructor should assign the `city` parameter to the `city` field and make an **independent copy** of the `temps` array parameter. Changes to the array passed in after construction should **not** affect the `temps` field of the object. Assume neither parameter is null and that the array has at least two elements.

```
public TempLog(String city, int[] temps) {

        this.city = city;
        this.temps = new int[temps.length];
        for (int i = 0; i < temps.length; i++)
            this.temps[i] = temps[i];
    }
```

2. This private method should compute and return the average of the temperatures stored in the `temps` array. You must use a for-each loop to iterate through the array; no other looping construct is allowed. The array elements are `int`, but the method should return a `double`, so think carefully about how to handle the division to avoid integer truncation.

```
private double average() {

        int sum = 0;
        for (int t : temps)
            sum += t;
        return (double) sum / temps.length;
    }
```

3. This static method should take two `TempLog` objects as parameters and return the one with the higher average temperature. If the averages are equal, it should return `null`. You should call the `average()` method of each object to compute the averages in order to avoid duplicating code. Do not worry about having to use > or < with doubles for comparison — it is acceptable in this context.

```
public static TempLog higherAverage(TempLog t1, TempLog t2) {

        double avg1 = t1.average();
        double avg2 = t2.average();
        if (avg1 > avg2) return t1;
        else if (avg2 > avg1) return t2;
        else return null;
    }
```

4. Complete the code for the `setThreshold` method shown below. This method updates the `static threshold` field. If the value passed in as `newThreshold` is negative, the threshold should default back to 5; otherwise, it should be set to the provided value. You must use the ternary operator as the **only** decision-making construct in your implementation—no `if`, `else`, or other control statements are allowed.

```
public static void setThreshold(int newThreshold) {

        threshold = (newThreshold < 0) ? 5 : newThreshold;
    }
```

5. Complete the remainder of the `equals` method shown below. If the two temperature arrays differ in length, the method should immediately return `false`. Otherwise, compare the temperatures at corresponding positions in the two arrays and count how many days differ by no more than the current threshold value. If at least two days meet this condition, the method should return `true`; otherwise, it should return `false`. You may assume both `TempLog` objects have valid, non-null arrays with at least 2 elements.

```
public boolean equals(Object obj) {

        if (obj == this)
                return true;
        if (obj == null || getClass() != obj.getClass())
                return false;
         TempLog other = (TempLog) obj;

          // If array lengths differ, not equal
          if (this.temps.length != other.temps.length)
              return false;

          int closeDays = 0;
          for (int i = 0; i < temps.length; i++) {
              int diff = Math.abs(this.temps[i] - other.temps[i]);
              if (diff <= threshold)
                  closeDays++;
          }

          // Two or more close days → equal
          return closeDays >= 2;
    }
```

Directory id:

LAST PAGE – Use only if needed