



# University of Maryland College Park

## Department of Computer Science

### CMSC131 Spring 2023

### Exam #3

FIRSTNAME, LASTNAME (PRINT IN UPPERCASE):

STUDENT ID (e.g. 123456789):

#### **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 – 3 pts each)	24
#2	Code 1	34
#3	Code 2	42
<b>Total</b>	<b>Total</b>	<b>100</b>

## **Problem #1 (Short Answers – 3 pts each)**

1. Given the code below:

```
ArrayList<String> aList = new ArrayList <String>();  
aList.add("Third");  
String [] strArray = {"Exam", null};
```

---

```
System.out.println(strArray [1]); // Third Exam
```

Finish the missing line that assigns to the second element of the array the concatenation of the first element of the ArrayList with a space and the first element of array. Your code **cannot** have the string literals “Third” or “Exam” in it.

2. What will display after the code below runs? \_\_\_\_\_

```
Stack<Integer> s = new Stack<>();  
s.push(5); s.pop();  
s.push(7); s.push(8);  
s.pop();  
System.out.println(s.pop()*2);
```

3. \_\_\_\_\_ allows you to focus on what code does and not how.
4. Circle the three headers below that would be valid overloads of the method with this header:

```
public static int e3(int x)
```

- a. public static void e3(int x)
- b. public static void e3(int x, int y)
- c. public static void e3(String s)
- d. public static int exam3(int x)
- e. public static void e3(double y)

Assume the following 4 files all in the same package. Use the code below for questions 5 to 8.

<pre>public interface Exam {     String whichExam(); }</pre>	<pre>public class Exam2 implements Exam{     public String whichExam() {         return "E2";     } }</pre>
<pre>public class Exam3 implements Exam{     public String whichExam() {         return "E3";     }     public int points(){         return 100;     } }</pre>	<pre>public class Driver {     public static void main(String[] args) {          /*HERE*/     } }</pre>

5. Assume the following code replaces the /\*HERE\*/ comment in the driver:

```
Exam e[] = {new Exam2(), new Exam3()} ;  
System.out.println(e[0].whichExam());
```

If the code above will not compile, write **NC**. If it will compile, but throw an exception, write **CE**. If it will compile and run, write the output.

**SAME EXACT CODE AS PAGE 1. JUST DUPLICATED ON THIS PAGE**

<pre>public interface Exam {     String whichExam(); }</pre>	<pre>public class Exam2 implements Exam{     public String whichExam() {         return "E2";     } }</pre>
<pre>public class Exam3 implements Exam{     public String whichExam() {         return "E3";     }     public int points(){         return 100;     } }</pre>	<pre>public class Driver {     public static void main(String[] args) {          /*HERE*/     } }</pre>

6. Assume the following code replaces the `/*HERE*/` comment in the driver:

```
Exam e1 = new Exam2();
System.out.println(((Exam3)e1).points());
```

If the code above will not compile, write **NC**. If it will compile, but throw an exception, write **CE**. If it will compile and run, **write the output**.

---

7. Assume the following code replaces the `/*HERE*/` comment in the driver:

```
Exam e2 = new Exam2();
System.out.println(((Exam2)e2).points());
```

If the code above will not compile, write **NC**. If it will compile, but throw an exception, write **CE**. If it will compile and run, **write the output**.

---

8. Assume the following code replaces the `/*HERE*/` comment in the driver:

```
Exam e3 = new Exam3();
System.out.println(((Exam3)e3).points());
```

If the code above will not compile, write **NC**. If it will compile, but throw an exception, write **CE**. If it will compile and run, **write the output**.

---

<b>Directory id:</b>
----------------------

## **Problem #2 (Code 1 – 34 pts )**

Complete the implementation of `TwoSums`. Assume that `input` is not `null`. If `key` is not in `input`, simply return `null`. If it is in `input`, return an `int` array with 2 elements. The first element will have the sum of all the integers to the left of the first instance of `key`, and the second element will have the sum of all the integers to the right of the first instance of `key`. The value of `key` is never included in the either sum even if it occurs more than once.

```
public static void main(String[] args) {  
  
    int [] a = {5,7,3,6,4,7,9,7,7,1,7,8};  
  
    System.out.println(Arrays.toString(TwoSums(a,7))); // [5, 31]  
    System.out.println(Arrays.toString(TwoSums(a,9))); // [32, 30]  
    System.out.println(Arrays.toString(TwoSums(a,1))); // [55, 15]  
    System.out.println(Arrays.toString(TwoSums(a,5))); // [0, 66]  
    System.out.println(Arrays.toString(TwoSums(a,8))); // [63, 0]  
    System.out.println(Arrays.toString(TwoSums(a,18))); // null  
    int [] b = {};  
    System.out.println(Arrays.toString(TwoSums(b,18))); // null  
    int [] c = {7,7,7};  
    System.out.println(Arrays.toString(TwoSums(c,18))); // null  
    System.out.println(Arrays.toString(TwoSums(c,7))); // [0, 0]  
}
```

```
public static int[] TwoSums(int [] input, int key) {
```

**//In case you need more room for Code 1**

**Directory id:**

### **Problem #3 (Code 2 – 42 pts )**

Complete the implementation of `makeArray`. Assume that `input` is not `null`. The method will return a 1-D array of `int` that contains integers from rows of `input` that have no `null` elements. The integers will appear in the returned array row after row. For example, in the driver you see that the integers from row 1 (index 0) and row 4 are in the answer, however the second row which is `null`, the third row which has a `null` element, and the 5<sup>th</sup> row which is empty are not included.

Although not required, you may use an `ArrayList` to help you keep track of information as you process `input`. However, if you do so, remember that you may only use the default constructor to make your `ArrayList`, the `get` method, the `add` method, and the `size` method. Again, your returned value is an array (not an `ArrayList`) of int (not `Integer`) with exactly the number of elements needed for the qualifying values.

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

    Integer [][] arr = new Integer[5][];
    arr[0] = new Integer []{5,7,12};
    arr[1] = null;
    arr[2] = new Integer []{0,null,8,15};
    arr[3] = new Integer []{3,4,5,6,8};
    arr[4] = new Integer []{};
    System.out.println(Arrays.toString(makeArray(arr)));

    Integer [][] arr1 = new Integer[3][];
    arr1[0] = null;
    arr1[1] = null;
    arr1[2] = new Integer []{0,null,8,15};

    System.out.println(Arrays.toString(makeArray(arr1)));

}
```

Output of main
[ 5, 7, 12, 3, 4, 5, 6, 8 ]
[ ]

```
public static int[] makeArray( Integer [][] input ) {
```

**Directory id:**

**EXTRA PAGE IN CASE YOU NEED IT FOR CODE 2**

**LAST PAGE**