



# University of Maryland College Park

## Department of Computer Science

### CMSC131 Spring 2024

#### 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 - 2pts each)	16
#2	Memory Map	24
#3	Coding	60
<b>Total</b>	<b>Total</b>	<b>100</b>

## Problem #1 (Short Answers – 2 pts each)

1. (2 pts) If a Java interface has a(n) \_\_\_\_\_ method, the class that implements the interface can use the code without re-writing the method definition.
2. (2 pts) A field in a Java interface will be public, static, and \_\_\_\_\_ .
3. (2 pts) A Stack is an example of a(n) \_\_\_\_\_ data type.

For 4 and 5, assume:

```
ArrayList <String> myList = new ArrayList<>();  
myList.add("Bob"); myList.add("Tom"); myList.add("Joe");
```

4. (2 pts) Use a for-each loop to print out all names. Your code should work even if there are more than 3 names. Use any library method(s) needed.

5. (2 pts) Use a traditional for loop to print out all names. Your code should work even if there are more than 3 names. Use any library method(s) needed.

Assume the following 4 files all in the same package. Use the code below for questions 6 to 8.

<pre>public interface Student {     public void takeExam(); }</pre>	<pre>public class StudentHistory implements Student{     public void takeExam() {         System.out.println("write the date");     } }</pre>
<pre>public class StudentCS implements Student{     public void takeExam() {         System.out.println("write code");     }     public void takeExam(int x) {         System.out.println("overload");     } }</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:

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**.

```
Student cs = new StudentCS();  
cs.takeExam(3);
```

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

<pre>public interface Student {     public void takeExam(); }</pre>	<pre>public class StudentHistory implements Student{     public void takeExam() {         System.out.println("write the date");     } }</pre>
<pre>public class StudentCS implements Student{     public void takeExam() {         System.out.println("write code");     }     public void takeExam(int x) {         System.out.println("overload");     } }</pre>	<pre>public class Driver {     public static void main(String[] args) {          /*HERE*/     } }</pre>

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

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**.

```
Student cs = new StudentHistory();
((StudentCS) cs).takeExam(3);
```

---

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

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**.

```
Student s = new StudentHistory();
s = new StudentCS();
s.takeExam();
```

---

**Directory id:**



## Problem #3 (Coding – 60 pts)

Complete the implementation of a class called **WordList** that represents a list of words. A **WordList** has a 1-D array (**strArray** instance variable). There is also a **character** static field that is counted twice when you code up the **compareTo**. You are given all the code of the **WordList** and only need to complete the following 3 methods: **getStringArray**, **getCharArray2D**, and **compareTo**. The only library methods you can use when you code up the three methods are: **length** field of an array, **length()** and **charAt()** from the **String** class, and **Integer.compareTo** method with 2 arguments. You will lose a significant number of points for using any other library methods. Using local primitive variables, creating arrays (both 1 and 2D), and using looping constructs of your choice are all allowed.

```
import java.util.Arrays;

public class WordList implements Comparable<WordList> {
    private static char character;
    private String[] strArray;

    public WordList(String[] strArray) {
        //makes a local copy for the class. You cannot use Arrays.copyOf in your code

        this.strArray = Arrays.copyOf(strArray, strArray.length);
    }

    public static void setCharacter(char character) {
        WordList.character = character;
    }

    public String[] getStringArray() {
        //YOUR CODE IS ANSWER TO P3 #1
    }

    public char[][] getCharArray2D() {
        //YOUR CODE IS ANSWER TO P3 #2
    }

    @Override
    public int compareTo(WordList other) {
        //YOUR CODE IS ANSWER TO P3 #3
    }

    @Override
    public String toString() {
        return "WordList{" +
            "character=" + character +
            ", strArray=" + Arrays.toString(strArray) +
            '}';
    }

    public static void main(String[] args) {
        String[] array1 = {"banana", "apple", "orange"}; //Your compareTo count should be 22
        String[] array2 = {"pear", "grape", "peach"}; // Your compareTo count should be 17

        WordList.setCharacter('a');

        WordList obj1 = new WordList(array1);
        WordList obj2 = new WordList(array2);

        System.out.println(obj1);
        System.out.println(obj2);

        System.out.println(obj1.compareTo(obj2));
        System.out.println(obj1.compareTo(obj1));
        System.out.println(obj2.compareTo(obj1));

        char[][] charArray2D = obj1.getCharArray2D();
        for (char[] row : charArray2D) {
            System.out.println(Arrays.toString(row));
        }
    }
}
```

### Output of the main method

```
WordList{character=a, strArray=[banana, apple, orange]}
WordList{character=a, strArray=[pear, grape, peach]}
1
0
-1
[b, a, n, a, n, a]
[a, p, p, l, e]
[o, r, a, n, g, e]
```

Directory id:

1. The purpose of the `getStringArray()` is to return a copy of the `strArray` instance field without causing a privacy leak. You decide if it should be a reference copy, a shallow copy, or a deep copy. We are looking for the most efficient implementation.

```
public String[] getStringArray() {
```

2. The `getCharArray2D()` will return a 2D array that has as many rows as `strArray` has string elements. Each row `i` will reference a 1D character array that has one character per element of the string located at index `i` of `strArray`. For simplicity, assume when this method is called that `strArray` is not an empty array, and that none of the elements of `strArray` are `null` or the empty string.

```
public char[][] getCharArray2D() {
```

3. To compare 2 **WordList** objects, count the number of characters in all of the strings referenced by the elements of **strArray** of each object (don't assume same length for each object's **strArray**). However, count instances of **character** twice in your count. Return 0 if both the current object and argument have an equal count, 1 if the current object has a greater count, and -1 if the current object has a smaller count. For simplicity, assume when this method is called that **strArray** of each object is not an empty array, and that none of the elements of **strArray** are **null** or the empty string. **Not required**, but you can write one private helper method to cut down on code duplication in **compareTo**. Write the private method definition after **compareTo** and call it in **compareTo**.

```
public int compareTo(WordList other) {
```