



University of Maryland College Park

Department of Computer Science

CMSC132 Spring 2025

Exam #1

FIRSTNAME, LASTNAME (PRINT IN UPPERCASE):

STUDENT ID (e.g. 123456789):

Instructions

- Please print your answers and use a pencil.
- This exam is a closed-book, closed-notes exam with a duration of 50 minutes and 100 total points.
- **Do not remove the exam's staple.** Removing it will interfere with the scanning process (even if you staple the exam again).
- Write your directory id (e.g., terps1, not UID) at the bottom of pages with **DirectoryId**.
- Provide answers in the rectangular areas.
- Do not remove any exam pages. Even if you don't use the extra pages for scratch work, return them with the rest of the exam.
- Your code must be efficient and as short as possible.
- If you continue a problem on the extra page(s) provided, make a note on the particular problem.
- For multiple choice questions you can assume only one answer is expected, unless stated otherwise.
- You don't need to use meaningful variable names; however, we expect good indentation.
- **You must write your name and id at this point (we will not wait for you after time is up).**
- You must stop writing once time is up.

Grader Use Only

#1	Problem #1 (Short Answer)	22	
#2	Problem #2 (Code 1)	25	
#3	Problem #3 (Code 2)	27	
#4	Problem #4 (Code 3)	26	
Total	Total	100	

Problem #1 (Short Answer)

1. (2 pts) Java differentiates overloaded methods by their _____
2. (2 pts) In Java, a subclass **must** call a constructor of its superclass explicitly using `super()`, or the program will fail to compile. True or False? _____
3. (2 pts) A subclass **inherits all** private fields and methods from its superclass, but it cannot access them directly. True or False? _____

Assume the following code for questions 4 to 7. No more than 3 sentence response, not counting output of code.

```
class Animal {
    void makeSound() {
        System.out.println("Animal makes a sound");
    }
}

class Dog extends Animal {
    void makeSound() {
        System.out.println("Dog barks");
    }
}

class Cat extends Animal {
    void makeSound() {
        System.out.println("Cat meows");
    }
}

public class Test {
    public static void main(String[] args) {
        Animal a1 = new Dog();
        Animal a2 = new Cat();
        Animal a3 = new Animal();

        a1.makeSound();
        a2.makeSound();
        a3.makeSound();
    }
}
```

4. (4 pts) What is the output of this program? Explain why.

5. (3 pts) Suppose we change the `makeSound()` method in `Animal` to be `final`. What will happen?_

6. (3 pts) If we add `@Override` above the `makeSound()` methods in `Dog` and `Cat`, does it change the output? Why or why not?

7. (3 pts) Consider the following `Wildlife` class in the same package as the given code:

```
public class Wildlife {
    public static void main(String[] args) {
        Animal a = new Dog();
        a.makeSound(); // Can this line be executed successfully?
    }
}
```

Can the `Wildlife` class call `makeSound()` on the `Dog` object? Why or why not?

8. (3 pts) In the JCF, the `Stack` class extends `Vector`. Explain how such a design can lead to misuse of the `Stack` ADT.

Problem #2 (Code 1)

```
import java.util.Stack;

public class PalindromeChecker {
    // Public method to check if a string is a palindrome using a stack
    public static boolean isPalindrome(String str) {
        Stack<Character> stack = new Stack<>();

        // Push all characters onto the stack
        for (char c : str.toCharArray()) {
            stack.push(c);
        }

        // Start the recursive palindrome check
        return isPalindromeRecursive(str, stack, 0);
    }

    // Recursive helper method
    private static boolean isPalindromeRecursive(String str, Stack<Character> stack, int index) {
        //YOU WILL CODE THIS
    }
}
```

Directory ID:

```

public static void main(String[] args) {
    // Test cases
    String[] testCases = {
        "racecar", // true
        "raceCar", // false (case-sensitive)
        "hello",   // false
        "madam",   // true
        "CS2 exam", // false
        "Never odd or even" // false (spaces are considered)
    };

    // Run palindrome checker for each test case
    for (String test : testCases) {
        System.out.println "\"" + test + "\" is palindrome? " + isPalindrome(test));
    }
}

```

Finish the code below. For this problem, a palindrome is a string that reads the same forwards and backwards, considering spaces and letter case. It must be recursive. If you use a loop, it is a zero. You may use **Stack** methods: **empty**, **peek**, **pop**, **push** (you will not need them all for the correct solution). From **String** you can use **charAt(int index)** and **length()**. Assume that the **str** argument is a valid String with one or more characters.

```

private static boolean isPalindromeRecursive(String str, Stack<Character> stack, int index)
{

```

Problem #3 (Code 2)

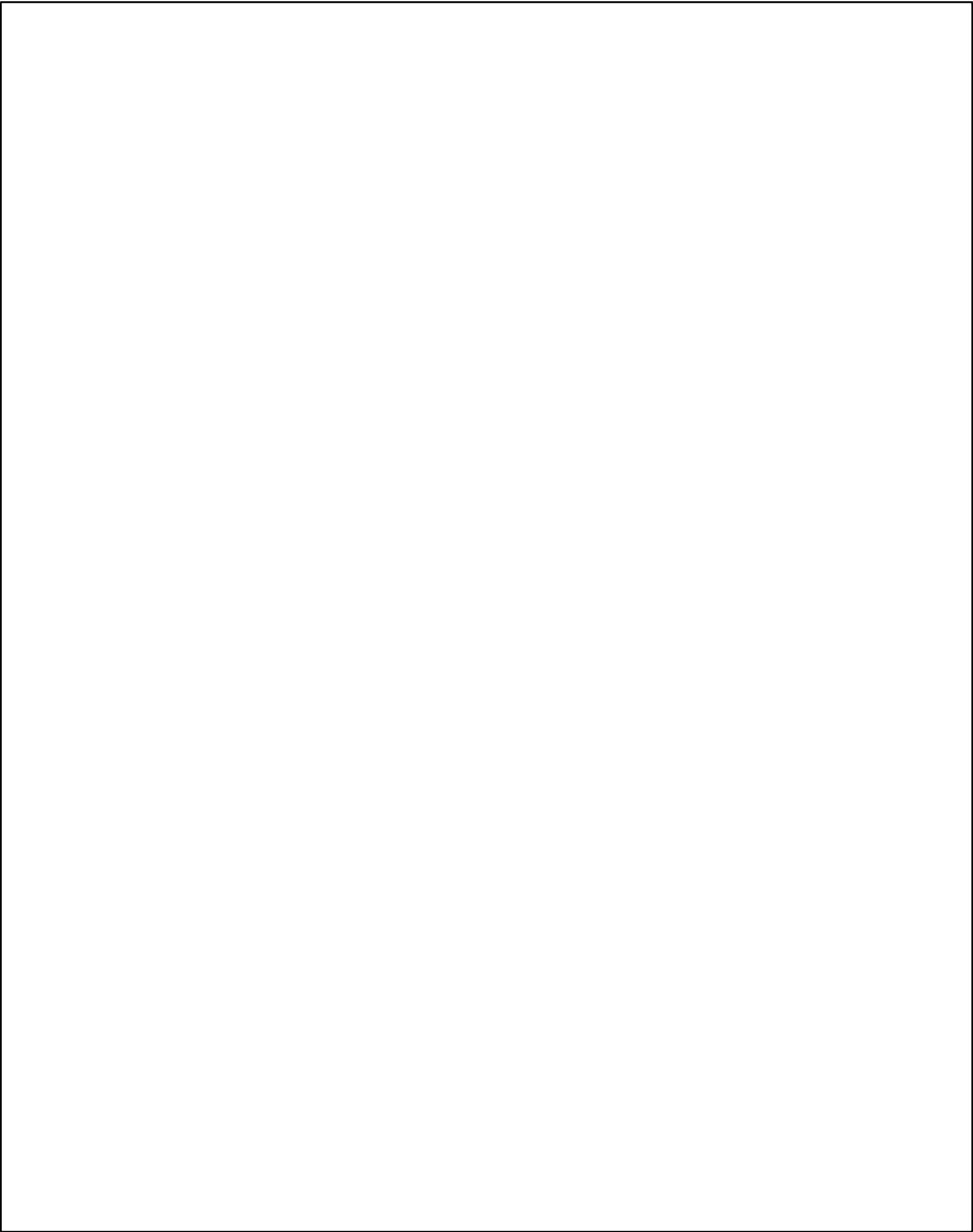
```
public class NullIndexer {
    public static void processArray(Object[] arr) {
        //YOU WILL CODE THIS
    }

    public static void main(String[] args) {
        Object[] arr = { "A", null, "B", null, "C", null, (short)51 };
        processArray(arr);
        Object[] arr1 = {null};
        processArray(arr1);
        Object[] arr2 = { "null", 5.7, new StringBuffer("cmsc") };
        processArray(arr2);

        // Print the modified array
        for (Object obj : arr) {
            System.out.print(obj + " "); // A 3 B 5 C 1 51
        }
        // Print the modified array
        System.out.println();
        for (Object obj : arr1) {
            System.out.print(obj + " "); // 0
        }
        // Print the modified array
        System.out.println();
        for (Object obj : arr2) {
            System.out.print(obj + " "); // null 5.7 cmsc
        }
    }
}
```

Finish the method below that processes an array of **Objects** in one pass. For each **null** in the array, set it to the index of the next **null**. The last **null** in the array should be set to the index of the first **null**. If an **Integer** is found in the array, throw an **IllegalArgumentException** with the message "**Array contains an integer**". Ensure the method works for arrays with multiple **nulls**, a single **null**, or no **nulls**. No library methods allowed other than constructing the exception. Using **length** field of the input array is allowed. Only one looping construct as you only get one pass from start to end of the array. Assume **arr** will be non-null with at least one element in it.

```
public static void processArray(Object[] arr) {
```



Problem #4 (Code 3)

```
import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;

public class QueueExample{

    public static void reverseFirstKElements(Queue<Integer> queue, int k) {
        //YOU WILL CODE THIS
    }

    public static void main(String[] args) {
        Queue<Integer> queue = new LinkedList<>();
        queue.add(1);
        queue.add(2);
        queue.add(3);
        queue.add(4);
        queue.add(5);
        queue.add(6);
        queue.add(7);
        queue.add(8);
        queue.add(9);
        queue.add(10);

        System.out.println("Original Queue: " + queue);
        reverseFirstKElements(queue, 5);
        System.out.println("Queue after reversing first 5 elements: " + queue);
    }
}
```

Original Queue: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] Queue after reversing first 5 elements: [5, 4, 3, 2, 1, 6, 7, 8, 9, 10]
--

Finish the method below that will reverse the first **k** elements in the queue. As for the **queue** object, you can use: **add**, **peek**, **remove**, **size**, and **isEmpty** methods. You can use the declared **Stack** to help you code. From **Stack** you can use: **empty**, **peek**, **push**, **pop**, and **size** methods. You do not need to use all these allowed method for a correct solution. No other library methods or additional data structures allowed (e.g. an array to hold the numbers). Local variables and looping/decision making constructs are allowed. The first part of the method that does validation and declares the **Stack** is given. Just finish the logic.

Directory ID:

```
public static void reverseFirstKElements(Queue<Integer> queue, int k) {  
    if (queue == null || queue.isEmpty() || k > queue.size() || k <= 0) {  
        throw new IllegalArgumentException("Invalid");  
    }  
  
    Stack<Integer> stack = new Stack<>();
```