Name: _____ Andrew Id: _____

15-121 Fall 2025 Assessment 1

Up to 60 minutes. No calculators, no notes, no books, and no computers. Show your work!

1. **Short Answer**.

   Answer each of the following in 1 sentence; longer answers will be marked incorrect.

   (a) (1 point) What is the fundamental difference between a compile-time error and a runtime error?

   (b) (1 point) When you create an array of integers in Java using the new keyword (for example, `int[] grades = new int[5];`), what are the elements of the array initialized to?

   (c) (1 point) In Java, what does a `String` variable actually store compared to a `char` variable?

   A. Both `String` and `char` variables store references to objects.

   B. A `String` variable stores a reference to a `String` object, while a `char` variable stores a single char value directly.

   C. Both `String` and `char` variables store raw character arrays.

   (d) (2 points) Assuming each of the below classes is in an appropriately named file, does the following code compile? If not, why?

```
1  public class BankAccount {
2      private double balance;
3
4      public BankAccount(double b) {
5          this.balance = b;
6      }
7  }
8
9  public class ATM {
10     public static void main(String[] args) {
11         BankAccount acct = new BankAccount(100.0);
12         System.out.println(acct.balance);
13     }
14 }
15
```

(e) (2 points) Bender writes the following code to check a user's password but it never says access is granted, even when the correct password "1234" is entered. What is the bug and the fix?

```
1  import java.util.Scanner;
2
3  public class Login {
4      public static void main(String[] args) {
5
6          Scanner sc = new Scanner(System.in);
7          String input = sc.next();
8          String password = "1234";
9
10         if (input == password) {
11             System.out.println("Access granted!");
12         } else {
13             System.out.println("Access denied!");
14         }
15     }
16 }
```

2. (4 points) **Code Tracing:** Indicate what the following program prints. Place your answer (and nothing else) in the box under the code.

```
public class IncrementorExercise {
    public static void main(String[] args) {
        int a = 3;
        int b = 6;
        int c = 2;

        System.out.println(++a - b-- + c++ - --a + b++ + --c - ++b + a--);
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
    }
}
```

3. (6 points) **Code Tracing:** Indicate what the following program prints. Place your answer (and nothing else) in the box under the code. Don't miss the static!

```
public class CT {
    private String word;
    private int value;
    public static double ratio = 0;

    public CT(String w, int v) {
        this.word = w.trim();
        this.value = v;
        for (int i = 0; i < this.word.length(); i++) {
            if (this.word.charAt(i) == 'a') {
                this.value += i;
            }
        }
        ratio = (double) v / this.word.length();
        System.out.println("C:" + this.word + " " + this.value);
    }

    public int tweak(String s) {
        if (this.word.contains(s)) {
            this.word = this.word.substring(0, 2) + s;
        }
        System.out.println("D:" + this.word);
        return this.word.length();
    }

    public String toString() {
        return this.word + " " + this.value + " " + ratio;
    }

    public static void main(String[] args) {
        CT f = new CT(" data ", 5);
        CT s = new CT("cat", 6);
        System.out.println(f);
        System.out.println(s);
        f.tweak("x");
        s.tweak("a");
        System.out.println(f);
        System.out.println(s);
    }
}
```

4. (10 points) **Free Response:**

   Write a `Student` class with the following properties:

   - Each `Student` has two private instance variables:
     - `String name` (the student's name)
     - `int id` (the student's ID number)
   - The `Student` class provides the following methods:
     - A constructor that takes a `name` and `id`.
     - A `getName` method that returns the student `name`.
     - A `getID` method that returns the student `id`.
     - A `toString` method that returns a String in the format `"id:name"`.

```
public class Student {

    private String name;
    private int ID;


    public Student(String name, int id) {

        this.name = name;
        this.ID = id;
    }


    public String getName() {
        return name;
    }



    public int getID() {
        return ID;
    }




    @Override
    public String toString() {
        return  name + ":" + ID;
    }


}
```

5. **Free Response**

A stack is a data structure that stores items in a "last-in, first-out" (LIFO) manner. The last element that is pushed onto the end of the stack is the first one that will be popped off.

In this problem, you will write a variety of methods in a `StringStack` class. A `StringStack` is used to store and operate on a stack of Strings. The size of a `StringStack` is not fixed: if more Strings are pushed than it can currently hold, then the underlying array is resized to make room.

Consider the following code for the `StringStack` class:

```
public class StringStack {
    // This class only has two instance variables. You may not add any more.
    private String[] arr;
    private int size;

    public StringStack(int capacity) {
        this.arr = new String[capacity];
        this.size = 0;
    }

    public boolean push(String val) {
        // You will write this code
    }

    public String pop() {
        // You will write this code
    }


    public void reverse() {
        // You will write this code
    }


    public int indexOf(String val) {
        // You will write this code
    }


    public String removeAtIndex(int index) {
        // You will write this code
    }


    public boolean removeAll(String val) {
        // You will write this code
    }

}
```

(a) (4 points) Write the `push` method, as specified below.

```
/**
 * Adds val to the end of the stack.
 * If the array is full, then first resize it by making it twice as large.
 *
 * @param val The String to push
 * @return true (since the value is always pushed after resizing if needed)
 */
public boolean push(String val){
```

(b) (2 points) Write the `pop` method, as specified below.

```
/**
 * Removes and returns the last value.
 * If the stack is empty, it should return null.
 *
 * @return The last String, or null if the stack is empty
 */
public String pop() {
```

(c) (5 points) Write the `reverse` method, as specified below.

```
/**
 * Reverses the order of the elements in the stack without creating
 * a new array. For example, if the stack contains
 * ["cat", "dog", "fish"], then after calling reverse()
 * it should contain ["fish", "dog", "cat"] .
 */
public void reverse() {
```

(d) (3 points) Write the private helper method `indexOf`, as specified below.

```
/**
 * Returns the index of the first occurrence of the given String in the stack.
 * The beginning of the stack has index 0 and the end has index size-1.
 * If the String does not occur, return -1.
 *
 * Example: If the stack contains [cat, dog, fish],
 * then indexOf("dog") returns 1, and indexOf("bird") returns -1.
 *
 * @param val The String to search for
 * @return The index of the first occurrence, or -1 if not found
 */
public int indexOf(String val) {
```

(e) (5 points) Write the `removeAtIndex` method, as specified below.

```
/**
 * Removes and returns the element at the given index.
 * The beginning of the stack has index 0 and the end has index size-1.
 * After removing, all elements after that index should be shifted left.
 * If the index is invalid (less than 0 or greater than or equal to size),
 * return null and do nothing.
 *
 * Example: If the stack contains [cat, dog, fish] and removeAtIndex(1) is called,
 * then the method returns "dog" and the stack becomes [cat, fish].
 *
 * @param index The index of the element to remove
 * @return The removed String, or null if index is invalid
 */
public String removeAtIndex(int index) {
```

(f) (4 points) Write the `removeAll` method, as specified below. *Assume that your implementation for* `indexOf` and `removeAtIndex` works.

```
/**
 * Removes all occurrences of the given String from the stack.
 * The order of the remaining elements should be preserved.
 * Returns true if any elements were removed, and false otherwise.
 *
 * Example: If the stack contains [cat, dog, cat, fish] and removeAll("cat") is called,
 * then the method returns true and the stack becomes [dog, fish].
 *
 * @param val The String to remove
 * @return true if at least one occurrence was removed, false otherwise
 */
public boolean removeAll(String val) {
```