

CSE 142 Sample Final Exam #4

1. Expressions

For each expression in the left-hand column, indicate its value in the right-hand column.

Be sure to list a constant of appropriate type and capitalization.

e.g., 7 for an int, 7.0 for a double, "hello" for a String, true or false for a boolean.

<u>Expression</u>	<u>Value</u>
102 / 20 / 2.0 * 2 + 100 / 30 / 2	_____
27 % 4 + 6 % 6 + 5 % 20	_____
"\" + (3 + 3 * 3 + \" + 3 + 3 * 3 + \"")	_____
!(42 > 10 && (2 > 3 4 < 5))	_____
(2 == 3) == (4 == 5)	_____

2. Array Mystery

Consider the following method:

```
public static void arrayMystery(int[] a) {
    for (int i = a.length - 1; i >= 0; i--) {
        if (a[i] == a[a.length - i - 1]) {
            a[i]++;
            a[a.length - i - 1]++;
        }
    }
}
```

Indicate in the right-hand column what values would be stored in the array after the method `arrayMystery` executes if the integer array in the left-hand column is passed as a parameter to it.

<u>Original Contents of Array</u>	<u>Final Contents of Array</u>
<code>int[] a1 = {1, 8, 3, 8, 7};</code> <code>arrayMystery(a1);</code>	_____
<code>int[] a2 = {4, 0, 0, 4, 0, 0, 4, 0};</code> <code>arrayMystery(a2);</code>	_____
<code>int[] a3 = {9, 8, 7, 6, 4, 6, 2, 9, 9};</code> <code>arrayMystery(a3);</code>	_____
<code>int[] a4 = {42};</code> <code>arrayMystery(a4);</code>	_____
<code>int[] a5 = {5, 5, 5, 6, 5, 5, 5};</code> <code>arrayMystery(a5);</code>	_____

3. Inheritance Mystery

Assume that the following classes have been defined:

```
public class Brian extends Lois {
    public void b() {
        a();
        System.out.print("Brian b ");
    }

    public String toString() {
        return "Brian";
    }
}

public class Lois extends Meg {
    public void a() {
        System.out.print("Lois a ");
        super.a();
    }

    public void b() {
        System.out.print("Lois b ");
    }
}

public class Meg {
    public void a() {
        System.out.print("Meg a ");
    }

    public void b() {
        System.out.print("Meg b ");
    }

    public String toString() {
        return "Meg";
    }
}

public class Stewie extends Brian {
    public void a() {
        super.a();
        System.out.print("Stewie a ");
    }

    public String toString() {
        return super.toString() + " Stewie";
    }
}
```

Given the classes above, what output is produced by the following code?

```
Meg[] elements = {new Lois(), new Stewie(), new Meg(), new Brian()};
for (int i = 0; i < elements.length; i++) {
    elements[i].a();
    System.out.println();
    elements[i].b();
    System.out.println();
    System.out.println(elements[i]);
    System.out.println();
}
```

4. File Processing

Write a static method named `pizza` that accepts as its parameter a `Scanner` for an input file. Imagine that a college dorm room contains several boxes of leftover pizza. A complete pizza has 8 slices. The pizza boxes left in the room each contain either an entire pizza, half a pizza (4 slices), or a single slice. Your method's task is to figure out the fewest boxes needed to store all leftover pizza, if all the partial pizzas were consolidated together as much as possible.

Your `pizza` method will read lines from its input `Scanner` where each line of data represents the contents of the pizza boxes in one dorm room. These contents are written as "whole", "half", or "slice" in either upper or lower case, separated by at least one space. You should print to the console the number of pizza boxes necessary to store all the slices of pizza out of the total. You must use a whole number of boxes. For example, if there are 10 total slices of pizza in a dorm room, 2 pizza boxes are needed: one for the first whole pizza, and one for the final 2 slices. Note that some lines might be blank, meaning that the dorm room contains no pizzas; output for such a case is shown below.

For example, consider the following input file representing 5 dorm rooms (note that the fourth is blank):

```
slice half slice whole whole half half half
whole HALF  WhoLE half WHOLE  WhOLE Slice half  sLICE half
WHOLE slice WHOLE  SLICE whole SLICE  whole slice WHOLE  half

slice
```

For the input above, your method should produce the following output:

```
5 / 8 pizza boxes used
7 / 10 pizza boxes used
6 / 10 pizza boxes used
0 / 0 pizza boxes used
1 / 1 pizza boxes used
```

The format of your output must exactly match that shown above.

Assume the `Scanner` contains at least 1 line of input, and that no line will contain tokens other than whole/half/slice.

5. Array Programming

Write a static method named `sweep` that accepts an array of integers as a parameter and performs a single "sweep" over the array from lowest to highest index, comparing adjacent elements. If a pair of adjacent elements are out of order (if the element at the lower index has a greater value than the element at the higher index), your method should swap them. For example, in an array of 6 elements, your method first examines elements at indexes 0-1, then 1-2, then 2-3, 3-4, and 4-5, swapping if they are out of order. One side effect of this method is that the single element with the largest value will be moved into the final index of the array. (Repeated "sweeping" can be used to sort an array.)

If your method ends up swapping any elements, it should return `true` to indicate that the array was changed. Otherwise (if the array was already in ascending order before the sweep), it should return `false`.

The table below shows calls to your method and the array contents after the method is finished executing:

Array / Call	Contents of Array After Call	Value Returned
<code>int[] a1 = {1, 6, 2, 7, 3, 6, 4}; sweep(a1);</code>	{1, 2, 6, 3, 6, 4, 7}	true
<code>int[] a2 = {9, 4, 2, 1, 3, 12, 14, 6, 8}; sweep(a2);</code>	{4, 2, 1, 3, 9, 12, 6, 8, 14}	true
<code>int[] a3 = {3, 4, 8, 2, 1, 8, 8, 4, 12}; sweep(a3);</code>	{3, 4, 2, 1, 8, 8, 4, 8, 12}	true
<code>int[] a4 = {-1, -4, 17, 4, -1}; sweep(a4);</code>	{-4, -1, 4, -1, 17}	true
<code>int[] a5 = {2, 3, 5, 7, 11, 13}; sweep(a5);</code>	{2, 3, 5, 7, 11, 13}	false
<code>int[] a6 = {42}; sweep(a6);</code>	{42}	false

You may assume that the array contains at least one element (its length is at least 1). Do not make assumptions about the values of the elements in the array; they could be very large or small, etc.

6. Array Programming (10 points)

Write a static method named `hasMirrorTwice` that accepts two arrays of integers `a1` and `a2` as parameters and returns `true` if `a1` contains all the elements of `a2` in reverse order at least twice (and `false` otherwise). For example, if `a2` stores the elements `{1, 2, 3}` and `a1` stores the elements `{6, 3, 2, 1, 4, 1, 3, 2, 1, 5}`, your method would return `true`.

Assume that both arrays passed to your method will have a length of at least 1. This means that the shortest possible mirror will be of length 1, representing a single element (which is its own mirror). A sequence that is a palindrome (the same forwards as backwards) is considered its own mirror and should be included in your computations. For example, if `a1` is `{6, 1, 2, 1, 4, 1, 2, 1, 5}` and `a2` is `{1, 2, 1}`, your method should return `true`. The two occurrences of the mirror might overlap, as shown in the fourth sample call below.

The following table shows some calls to your method and their expected results:

Array	Returned Value
<code>int[] a1 = {6, 1, <u>2, 1</u>, 3, 1, 3, <u>2, 1</u>, 5};</code> <code>int[] a2 = {1, 2};</code>	<code>hasMirrorTwice(a1, a2)</code> returns <code>true</code>
<code>int[] a3 = {<u>5, 8, 4</u>, 18, 5, 42, 4, 8, 5, 5};</code> <code>int[] a4 = {4, 8, 5};</code>	<code>hasMirrorTwice(a3, a4)</code> returns <code>false</code>
<code>int[] a5 = {6, <u>3, 42</u>, 18, 12, 5, <u>3, 42</u>, <u>3, 42</u>};</code> <code>int[] a6 = {42, 3};</code>	<code>hasMirrorTwice(a5, a6)</code> returns <code>true</code>
<code>int[] a7 = {6, <u>1, 2, 4, 2, 1, 2, 4, 2, 1</u>, 5};</code> <code>int[] a8 = {1, 2, 4, 2, 1};</code>	<code>hasMirrorTwice(a7, a8)</code> returns <code>true</code>
<code>int[] a9 = {0, 0};</code> <code>int[] aa = {0};</code>	<code>hasMirrorTwice(a9, aa)</code> returns <code>true</code>
<code>int[] ab = {8, 9, 2, 1};</code> <code>int[] ac = {5, 7, 1, 2, 9, 8};</code>	<code>hasMirrorTwice(ab, ac)</code> returns <code>false</code>

Do not modify the contents of the arrays passed to your method as parameters.

7. Critters

Write a class `Crab` that extends the `Critter` class from the Critters assignment, along with its movement behavior. `Crab` objects move in the following repeating pattern:

- west 1 time, east 2 times
- west 3 times, east 4 times
- west 5 times, east 6 times
- west 7 times, east 8 times
- west 7 times, east 6 times
- west 5 times, east 4 times
- west 3 times, east 2 times
- (*entire pattern repeats*)

Write your complete `Crab` class below. All other aspects of `Crab` besides movement use the default `critter` behavior. You may add anything needed to your class (fields, constructors, etc.) to implement this behavior appropriately.

Note: You will receive a maximum of only 5 points if you "hard-code" the entire set of movements (by simply keeping a move counter and using an extremely long chain of `if` or `if/else` statements) rather than recognizing the crab's overall movement pattern.

8. Classes and Objects

Write a method named `bound` that will be placed inside the `Date` class from the Birthday/Date assignment, as shown at right.

Write a method named `bound` that will be placed inside the `Date` class. The `bound` method constrains a `Date` to within a given range of dates. It accepts two other `Date` objects `d1` and `d2` as parameters; `d1`'s date is guaranteed to represent a date that comes no later in the year than `d2`'s date.

The `bound` method makes sure that this `Date` object is between `d1`'s and `d2`'s dates, inclusive. If this `Date` object is not between those dates inclusive, it is adjusted to the nearest date in the acceptable range. The method returns a result of `true` if this `Date` was within the acceptable range, or `false` if it was shifted.

For example, given the following `Date` objects:

```
Date date1 = new Date(7, 12);
Date date2 = new Date(10, 31);
Date date3 = new Date(9, 19);
Date bound1 = new Date(8, 4);
Date bound2 = new Date(9, 26);
Date bound3 = new Date(12, 25);
```

The following calls to your method should adjust the given `Date` objects to represent the following dates and should return the following results:

call	date becomes	returns
<code>date1.bound(bound1, bound2)</code>	8/4	false
<code>date2.bound(bound1, bound2)</code>	9/26	false
<code>date3.bound(bound1, bound3)</code>	9/19	true
<code>date2.bound(bound3, bound3)</code>	12/25	false

```
public class Date {
    private int month;
    private int day;

    public Date(int m, int d)

    public int getDay()

    public int getMonth()

    public int daysInMonth()

    public void nextDay()

    // your method would go here
}
```

Solutions

1. Expressions

Expression

```
102 / 20 / 2.0 * 2 + 100 / 30 / 2
27 % 4 + 6 % 6 + 5 % 20
"\\" + (3 + 3 * 3 + "\" + 3 + 3 * 3 + "\\")
!(42 > 10 && (2 > 3 || 4 < 5))
(2 == 3) == (4 == 5)
```

Value

```
6.0
8
"\\"1239\\"
false
true
```

2. Array Mystery

Expression

```
int[] a1 = {1, 8, 3, 8, 7};
arrayMystery(a1);

int[] a2 = {4, 0, 0, 4, 0, 0, 4, 0};
arrayMystery(a2);

int[] a3 = {9, 8, 7, 6, 4, 6, 2, 9, 9};
arrayMystery(a3);

int[] a4 = {42};
arrayMystery(a4);

int[] a5 = {5, 5, 5, 6, 5, 5, 5};
arrayMystery(a5);
```

Final Contents of Array

```
[1, 10, 5, 10, 7]

[4, 0, 2, 4, 0, 2, 4, 0]

[11, 8, 7, 8, 6, 8, 2, 9, 11]

[44]

[7, 7, 7, 8, 7, 7, 7]
```

3. Inheritance Mystery

```
Lois a    Meg a
Lois b
Meg

Lois a    Meg a    Stewie a
Lois a    Meg a    Stewie a    Brian b
Brian Stewie

Meg a
Meg b
Meg

Lois a    Meg a
Lois a    Meg a    Brian b
Brian
```

4. File Processing (five solutions shown)

```
public static void pizza(Scanner input) {
    while (input.hasNextLine()) {
        String line = input.nextLine();
        Scanner lineScan = new Scanner(line.toLowerCase());
        double boxes = 0.0;
        int totalBoxes = 0;
        while (lineScan.hasNext()) {
            totalBoxes++;
            String box = lineScan.next();
            if (box.equals("whole")) {
                boxes++; // 1
            } else if (box.equals("half")) {
                boxes += 0.5; // 1/2
            } else { // "slice"
                boxes += 0.125; // 1/8
            }
        }

        System.out.println((int) Math.ceil(boxes) + " / " + totalBoxes +
            " pizza boxes used");
    }
}

public static void pizza(Scanner input) {
    while (input.hasNextLine()) {
        Scanner lineScan = new Scanner(input.nextLine().toLowerCase());
        int slices = 0;
        int totalBoxes = 0;
        while (lineScan.hasNext()) {
            totalBoxes++;
            String box = lineScan.next();
            if (box.equals("whole")) {
                slices += 8;
            } else if (box.equals("half")) {
                slices += 4;
            } else {
                slices++;
            }
        }

        // round up to an even box
        int boxes = slices / 8;
        if (slices % 8 != 0) {
            boxes++;
        }
        System.out.println(boxes + " / " + totalBoxes + " pizza boxes used");
    }
}

public static void pizza(Scanner input) {
    while (input.hasNextLine()) {
        Scanner lineScan = new Scanner(input.nextLine().toLowerCase());
        int slices = 0;
        int totalBoxes = 0;
        while (lineScan.hasNext()) {
            totalBoxes++;
            String box = lineScan.next();
            if (box.equals("whole")) {
                slices += 8;
            } else if (box.equals("half")) {
                slices += 4;
            } else {
                slices++;
            }
        }
    }
}
```



```

        // round up to an even box
        if (slices % 8 != 0) {
            slices += (8 - slices % 8);
        }
        System.out.println(slices / 8 + " / " + totalBoxes + " pizza boxes used");
    }
}

public static void pizza(Scanner input) {
    while (input.hasNextLine()) {
        Scanner lineScan = new Scanner(input.nextLine().toLowerCase());
        int slices = 0;
        int totalBoxes = 0;
        while (lineScan.hasNext()) {
            totalBoxes++;
            String box = lineScan.next();
            if (box.equals("whole")) {
                slices += 8;
            } else if (box.equals("half")) {
                slices += 4;
            } else {
                slices++;
            }
        }

        // round up to an even box
        System.out.println((int) Math.ceil(slices / 8.0) + " / " + totalBoxes +
            " pizza boxes used");
    }
}

public static void pizza(Scanner input) {
    while (input.hasNextLine()) {
        Scanner lineScan = new Scanner(input.nextLine().toLowerCase());
        int slices = 0;
        int halves = 0;
        int wholes = 0;
        int totalBoxes = 0;
        while (lineScan.hasNext()) {
            totalBoxes++;
            String box = lineScan.next();
            if (box.equals("whole")) {
                wholes++;
            } else if (box.equals("half")) {
                halves++;
            } else {
                slices++;
            }
        }

        int boxes = (int) Math.ceil((8*wholes + 4*halves + slices) / 8.0);
        System.out.println(boxes + " / " + totalBoxes + " pizza boxes used");
    }
}

```

5. Array Programming (three solutions shown)

```
public static boolean sweep(int[] a) {
    boolean changed = false;
    for (int i = 0; i < a.length - 1; i++) {
        if (a[i] > a[i + 1]) {
            int temp = a[i];
            a[i] = a[i + 1];
            a[i + 1] = temp;
            changed = true;
        }
    }
    return changed;
}
```

```
public static boolean sweep(int[] a) {
    int count = 0;
    for (int i = 0; i < a.length - 1; i++) {
        if (a[i] > a[i + 1]) {
            int temp = a[i];
            a[i] = a[i + 1];
            a[i + 1] = temp;
            count++;
        }
    }

    if (count > 0) {
        return true;
    } else {
        return false;
    }
}
```

```
public static boolean sweep(int[] a) {
    int count = 0;
    for (int i = 1; i < a.length; i++) {
        int temp1 = a[i];
        int temp2 = a[i - 1];

        if (a[i - 1] > a[i]) {
            a[i - 1] = temp1;
            a[i] = temp2;
            count++;
        }
    }

    if (count > 0) {
        return true;
    } else {
        return false;
    }
}
```

6. Array Programming (four solutions shown)

```
public static boolean hasMirrorTwice(int[] a1, int[] a2) {
    int mirrorCount = 0;
    for (int i = 0; i <= a1.length - a2.length; i++) {
        int count = 0;
        for (int j = 0; j < a2.length; j++) {
            if (a1[i + j] == a2[a2.length - 1 - j]) {
                count++;
            }
        }
        if (count >= a2.length) {
            mirrorCount++;
        }
    }

    return mirrorCount >= 2;
}
```

```
public static boolean hasMirrorTwice(int[] a1, int[] a2) {
    int mirrorCount = 0;
    for (int i = a2.length - 1; i < a1.length; i++) {
        int count = 0;
        for (int j = 0; j < a2.length; j++) {
            if (a1[i - j] == a2[j]) {
                count++;
            }
        }
        if (count >= a2.length) {
            mirrorCount++;
        }
    }

    return mirrorCount >= 2;
}
```

```
public static boolean hasMirrorTwice(int[] a1, int[] a2) {
    // copy/reverse a2 into a3
    int[] a3 = new int[a2.length];
    for (int i = 0; i < a2.length; i++) {
        a3[i] = a2[a2.length - 1 - i];
    }

    int mirrorCount = 0;
    for (int i = 0; i <= a1.length - a3.length; i++) {
        int count = 0;
        for (int j = 0; j < a3.length; j++) {
            if (a1[i + j] == a3[j]) {
                count++;
            }
        }
        if (count >= a3.length) {
            mirrorCount++;
        }
    }

    if (mirrorCount >= 2) {
        return true;
    } else {
        return false;
    }
}
```

```
public static boolean hasMirrorTwice(int[] a1, int[] a2) {
    int mirrorCount = 0;
    for (int i = a1.length - 1; i >= 0; i--) {
        int count = 0;
        for (int j = 0; j < a2.length; j++) {
            if (i + j < a1.length && a1[i + j] == a2[a2.length - 1 - j]) {
                count++;
            }
        }
        if (count >= a2.length) {
            mirrorCount++;
        }
    }
    return mirrorCount >= 2;
}
```

7. Critters (three solutions shown)

```
public class Crab extends Critter {
    private int max = 1;
    private int moves = 0;
    private boolean wider = true;

    public Direction getMove() {
        if (moves == max) {
            moves = 0;
            if (wider) {
                max++;
            } else {
                max--;
            }
            if (max == 8 || max == 1) {
                wider = !wider;
            }
        }

        moves++;
        if (max % 2 == 0) {
            return Direction.EAST;
        } else {
            return Direction.WEST;
        }
    }
}

public class Crab extends Critter {
    private int moves = 0;

    public Direction getMove() {
        moves++;
        if (moves == 64) { // 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 = 36
            moves = 1; // 36 + 7 + 6 + 5 + 4 + 3 + 2 + 1 = 64
        }

        int sum = 0;
        for (int i = 1; i <= 8; i++) {
            if (moves >= sum && moves <= sum + i) {
                if (i % 2 == 0) { return Direction.EAST; }
                else { return Direction.WEST; }
            }
            sum += i;
        }

        for (int i = 7; i >= 2; i--) {
            if (moves >= sum && moves <= sum + i) {
                if (i % 2 == 0) { return Direction.EAST; }
                else { return Direction.WEST; }
            }
            sum += i;
        }
        return Direction.CENTER; // it will never reach here
    }
}
```

```

public class Crab extends Critter {
    private static int[] phases = {1, 2, 3, 4, 5, 6, 7, 8, 7, 6, 5, 4, 3, 2, 1};
    private int moves = 0;

    public Direction getMove() {
        moves++;
        if (moves == 64) // 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 = 36
            moves = 1; // 36 + 7 + 6 + 5 + 4 + 3 + 2 + 1 = 64

        int sum = 0;
        for (int i = 0; i < phases.length; i++) {
            if (moves >= sum && moves <= sum + phases[i]) {
                if (phases[i] % 2 == 0) { return Direction.EAST; }
                else { return Direction.WEST; }
            }
            sum += phases[i];
        }
        return Direction.CENTER; // it will never reach here
    }
}

```

8. Objects (one solution shown)

```

public boolean bound(Date d1, Date d2) {
    if (month < d1.month || (month == d1.month && day < d1.day)) {
        month = d1.month;
        day = d1.day;
        return false;
    } else if (month > d2.month || (month == d2.month && day > d2.day)) {
        month = d2.month;
        day = d2.day;
        return false;
    } else {
        return true;
    }
}

```