

CSE 142 Sample Final Exam #1

1. Expressions

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

List a constant of appropriate type (e.g., 7.0 for a double, true or false for a boolean, Strings in " " quotes).

<u>Expression</u>	<u>Value</u>
4.5 / 3 / 2 + 1	_____
57 % 10 / 3 + 1.25 * 4	_____
5 * 6 / 4 % 3 - 23 / (14 % 6)	_____
(6.7 > 25) && 9 > 2 && !(8.2 < 5)	_____
8 - 4 + "17" + 7.5 * 2	_____

2. Array Mystery

Consider the following method:

```
public static void mystery(int[] array) {  
    for (int i = 0; i < array.length; i++) {  
        array[i] = i * array[i];  
    }  
}
```

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

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

3. Inheritance Mystery

Assume that the following classes have been defined:

```
public class A extends B {
    public void method2() {
        System.out.print("a 2 ");
        method1();
    }
}

public class B extends C {
    public String toString() {
        return "b";
    }

    public void method2() {
        System.out.print("b 2 ");
        super.method2();
    }
}

public class C {
    public String toString() {
        return "c";
    }

    public void method1() {
        System.out.print("c 1 ");
    }

    public void method2() {
        System.out.print("c 2 ");
    }
}

public class D extends B {
    public void method1() {
        System.out.print("d 1 ");
        method2();
    }
}
```

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

```
C[] elements = {new A(), new B(), new C(), new D()};
for (int i = 0; i < elements.length; i++) {
    System.out.println(elements[i]);
    elements[i].method1();
    System.out.println();
    elements[i].method2();
    System.out.println();
    System.out.println();
}
```


6. Array Programming

Write a static method `isAllEven` that takes an array of integers as a parameter and that returns a boolean value indicating whether or not all of the values are even numbers (`true` for yes, `false` for no). For example, if a variable called `list` stores the following values:

```
int[] list = {18, 0, 4, 204, 8, 4, 2, 18, 206, 1492, 42};
```

Then the call of `isAllEven(list)` should return `true` because each of these integers is an even number. If instead the list had stored these values:

```
int[] list = {2, 4, 6, 8, 10, 208, 16, 7, 92, 14};
```

Then the call should return `false` because, although most of these values are even, the value 7 is an odd number.

7. Array Programming

Write a static method named `isUnique` that takes an array of integers as a parameter and that returns a boolean value indicating whether or not the values in the array are unique (`true` for yes, `false` for no). The values in the list are considered unique if there is no pair of values that are equal. For example, if a variable called `list` stores the following values:

```
int[] list = {3, 8, 12, 2, 9, 17, 43, -8, 46, 203, 14, 97, 10, 4};
```

Then the call of `isUnique(list)` should return `true` because there are no duplicated values in this list. If instead the list stored these values:

```
int[] list = {4, 7, 2, 3, 9, 12, -47, -19, 308, 3, 74};
```

Then the call should return `false` because the value 3 appears twice in this list. Notice that given this definition, a list of 0 or 1 elements would be considered unique.

8. Critters

Write a class `Weasel` that extends the `Critter` class from the Critters assignment, including its `getMove` and `getColor` methods. `Weasel` objects first pick a random direction and go three steps in that direction, then stay in the same place for three moves, then pick a random direction and go three in that direction, then stay in the same place for three steps, and so on, alternating between moving in a random direction and staying still. Whenever a `Weasel` is moving, its color should be yellow (`Color.YELLOW`). As soon as it stops moving, its color should be cyan (`Color.CYAN`). When picking a random direction, use the directions defined in the `Critter` assignment, namely `Direction.NORTH`, `Direction.SOUTH`, `Direction.EAST`, and `Direction.WEST`. Each direction should be equally likely. To stay in one place, return `Direction.CENTER`.

You may add anything needed (fields, other methods) to implement the above behavior appropriately.

9. Classes and Objects

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

The `compareTo` method accepts another `Date` as a parameter and compares them to see which comes first in chronological order. It returns an integer with the following value:

- a negative integer (such as -1) if the date represented by this `Date` comes before that of the parameter
- 0 if the two `Date` objects represent the same month and day
- a positive integer (such as 1) if the date represented by this `Date` comes after that of the parameter

For example, if these `Date` objects are declared in client code:

```
Date sep19 = new Date(9, 19);
Date dec15 = new Date(12, 15);
Date temp = new Date(9, 19);
Date sep11 = new Date(9, 11);
```

The following boolean expressions produce true results.

```
sep19.compareTo(sep11) > 0
sep11.compareTo(sep19) < 0
temp.compareTo(sep19) == 0
dec15.compareTo(sep11) > 0
```

Your method should not modify the state of either `Date` object (such as by changing their day or month field values).

```
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

Expression	Value
4.5 / 3 / 2 + 1	1.75
57 % 10 / 3 + 1.25 * 4	7.0
5 * 6 / 4 % 3 - 23 / (14 % 6)	-10
(6.7 > 25) && 9 > 2 && !(8.2 < 5)	false
8 - 4 + "17" + 7.5 * 2	"41715.0"

Array	Value returned
{}	{}
{7}	{0}
{3, 2}	{0, 2}
{5, 4, 3}	{0, 4, 6}
{2, 4, 6, 8}	{0, 4, 12, 24}

```
4. public static void printStrings(Scanner input) {
    while (input.hasNextInt()) {
        int times = input.nextInt();
        String word = input.next();
        for (int i = 0; i < times; i++) {
            System.out.print(word);
        }
        System.out.println();
    }
}
```

```
5. public static void reverseLines(Scanner input) {
    while (input.hasNextLine()) {
        String text = input.nextLine();
        for (int i = text.length() - 1; i >= 0; i--) {
            System.out.print(text.charAt(i));
        }
        System.out.println();
    }
}
```

```
6. public static boolean isAllEven(int[] list) {
    for (int i = 0; i < list.length; i++) {
        if (list[i] % 2 != 0) {
            return false;
        }
    }
    return true;
}
```

```
7. public static boolean isUnique(int[] list) {
    for (int i = 0; i < list.length; i++) {
        for (int j = i + 1; j < list.length; j++) {
            if (list[i] == list[j]) {
                return false;
            }
        }
    }
    return true;
}
```

```
3. b
   c 1
   a 2   c 1

   b
   c 1
   b 2   c 2

   c
   c 1
   c 2

   b
   d 1   b 2   c 2
   b 2   c 2
```

8.

```
public class Weasel extends Critter {
    private Random rand;
    private int steps;
    private int direction;
    private boolean hiding;

    public Weasel() {
        rand = new Random();
        hiding = false;
        direction = rand.nextInt(4);
        steps = 0;
    }

    public Color getColor() {
        if (hiding) {
            return Color.CYAN;
        } else {
            return Color.YELLOW;
        }
    }

    public Direction getMove() {
        // Pick a new direction and re-set the steps counter
        if (steps == 3) {
            steps = 0;
            hiding = !hiding;
            direction = rand.nextInt(4);
        }

        steps++;
        if (hiding) {
            return Direction.CENTER;
        } else if (direction == 0) {
            return Direction.NORTH;
        } else if (direction == 1) {
            return Direction.SOUTH;
        } else if (direction == 2) {
            return Direction.EAST;
        } else { // direction == 3
            return Direction.WEST;
        }
    }
}
```

9. Two solutions are shown.

```
public int compareTo(Date other) {
    if (month < other.month || (month == other.month && day < other.day)) {
        return -1;
    } else if (month == other.month && day == other.day) {
        return 0;
    } else {
        return 1;
    }
}

public int compareTo(Date other) {
    if (month == other.month) {
        return day - other.day;
    } else {
        return month - other.month;
    }
}
```