CSE 142 Sample Final Exam #3

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`, `String`s in `""` quotes).

Expression	Value
`(49 % 9 + 33 % 6) / (2 % 10)`	_____
`50 / 9 / 2.0 + 200 / 10 / (5.0 / 2)`	_____
`10 > 5 + 4 && !(33 % 2 == 0 || 34 % 2 != 0)`	_____
`1 + (2 + 3) + 4 + "5" + 6 + 7 + (8 + 9) * 10`	_____
`(20 < 30) == (10 % 2 != 0)`	_____

2. Array Mystery

Consider the following method:

```
public static void arrayMystery(int[] a) {
    for (int i = a.length - 2; i > 0; i--) {
        if (a[i + 1] <= a[i - 1]) {
            a[i]++;
        }
    }
}
```

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.

Original Contents of Array	Final Contents of Array
`int[] a1 = {42};` `arrayMystery(a1);`	_____
`int[] a2 = {1, 8, 3, 6};` `arrayMystery(a2);`	_____
`int[] a3 = {5, 5, 5, 5, 5};` `arrayMystery(a3);`	_____
`int[] a4 = {10, 7, 9, 6, 8, 5};` `arrayMystery(a4);`	_____
`int[] a5 = {1, 0, 1, 0, 0, 1, 0};` `arrayMystery(a5);`	_____

3. Inheritance Mystery

Assume that the following classes have been defined:

```
public class Vier extends Drei {
    public void method2() {
        super.method2();
        System.out.print("Vier 2   ");
    }

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


public class Zwei extends Eins {
    public void method2() {
        System.out.print("Zwei 2   ");
        method1();
    }
}
```

```
public class Drei extends Zwei {
    public void method1() {
        System.out.print("Drei 1   ");
    }

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

public class Eins {
    public String toString() {
        return "Eins";
    }

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

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

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

```
Eins[] elements = {new Zwei(), new Eins(), new Vier(), new Drei()};
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();
}
```

4. File Processing

Write a static method named `coinFlip` that accepts as its parameter a `Scanner` for an input file. Assume that the input file data represents results of sets of coin flips that are either heads (H) or tails (T) in either upper or lower case, separated by at least one space. Your method should consider each line to be a separate set of coin flips and should output to the console the number of heads and the percentage of heads in that line, rounded to the nearest tenth. If this percentage is more than 50%, you should print a "You win" message. For example, consider the following input file:

```
H T H H T
T t   t T h   H
   h
```

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

```
3 heads (60.0%)
You win!

2 heads (33.3%)

1 heads (100.0%)
You win!
```

The format of your output must exactly match that shown above. You may assume that the `Scanner` contains at least 1 line of input, that each line contains at least one token, and that no tokens other than h, H, t, or T will be in the lines.

5. Array Programming

Write a static method named `range` that takes an array of integers as a parameter and returns the range of values contained in the array. The range of an array is defined to be one more than the difference between its largest and smallest element. For example, if the largest element in the array is 15 and the smallest is 4, the range is 12. If the largest and smallest values are the same, the range is 1.

The following table shows some calls to your method and their results (the largest and smallest values are underlined):

Array	Returned Value
`int[] a1 = {8, 3, 5, 7, 2, 4};`	`range(a1)` returns `7`
`int[] a2 = {15, 22, 8, 19, 31};`	`range(a2)` returns `24`
`int[] a3 = {3, 10000000, 5, -29, 4};`	`range(a3)` returns `10000030`
`int[] a4 = {100, 5};`	`range(a4)` returns `96`
`int[] a5 = {32};`	`range(a5)` returns `1`

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

6. Array Programming

Write a static method named `zeroOut` that accepts two arrays of integers *a1* and *a2* as parameters and replaces any occurrences of *a2* in *a1* with zeroes. The sequence of elements in *a2* may appear anywhere in *a1* but must appear consecutively and in the same order. For example, if variables called `a1` and `a2` store the following values:

```
int[] a1 = {1, 2, 3, 4, 1, 2, 3, 4, 5};
int[] a2 = {2, 3, 4};
```

The call of `zeroOut(a1, a2)` should modify `a1`'s contents to be `{1, 0, 0, 0, 1, 0, 0, 0, 5}`.
Note that the pattern can occur many times, even consecutively. For the following two arrays `a3` and `a4`:

```
int[] a3 = {5, 5, 5, 18, 5, 42, 5, 5, 5, 5};
int[] a4 = {5, 5};
```

The call of `zeroOut(a3, a4)` should modify `a3`'s contents to be `{0, 0, 5, 18, 5, 42, 0, 0, 0, 0}`.

You may assume that both arrays passed to your method will have lengths of at least 1. If *a2* is not found in *a1*, or if *a1*'s length is shorter than *a2*'s, then *a1* is not modified by the call to your method. Please note that *a1*'s contents are being modified in place; you are not supposed to return a new array. Do not modify the contents of *a2*.

7. Critters

Write a class `Dragonfly` that extends the `Critter` class from the Critters assignment. Whenever a `Dragonfly` encounters food, it eats it. Eating affects the `Dragonfly`'s movement.
`Dragonfly` objects move in a N/E/S/E sequence, initially going east once between going north and south, but going east an additional time for each time the `Dragonfly` has eaten.

- If the `Dragonfly` has never eaten: `NORTH`, `EAST`, `SOUTH`, `EAST`, and repeat
- If the `Dragonfly` has eaten once: `NORTH`, `EAST`, **`EAST`**, `SOUTH`, `EAST`, **`EAST`**, and repeat.
- If the `Dragonfly` has eaten twice: `NORTH`, `EAST`, `EAST`, **`EAST`**, `SOUTH`, `EAST`, `EAST`, **`EAST`**, and repeat.
- ...

We will be somewhat flexible about what should happen if a `Dragonfly` eats in the middle of a movement sequence. Either it should take effect immediately, lengthening the rest of that sequence and all subsequent ones; or it can take effect at the beginning of the next overall N/E/S/E movement sequence, lengthening it and all subsequent sequences.

You may add anything needed (fields, constructors, etc.) to implement this behavior appropriately.

8. Classes and Objects

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

The `daysTillXmas` method returns how many days away the `Date` object is from Christmas, December 25, in the same year. For example:

- **Dec. 12** is **13** days away from Christmas.
- **Nov. 22** is **33** days away (the 8 remaining days in November, and 25 more to reach 12/25).
- **Sep. 3** is **113** days away (the 27 remaining days in September, the 61 days in October and November, and 25 more to reach 12/25).

Here is an example call of your method that would print 113:

```
Date d = new Date(9, 3);
System.out.println(d.daysTillXmas());   // 113
```

Here are more dates and the values that the method should return when called on `Date` objects representing them. 12/25 is 0 days away from itself, and days after 12/25 return a negative value.

- **Dec. 25** is **0** days away from Christmas.
- **Dec. 26** is **-1** days away from Christmas.
- **Dec. 31** is **-6** days away from Christmas.
- **Jan. 1** is **358** days away from Christmas.
- **Feb. 14** is **314** days away from Christmas.

Your method should not modify the `Date` object's state, such as by changing its `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

1. Expressions

Expression	Value
`(49 % 9 + 33 % 6) / (2 % 10)`	`3`
`50 / 9 / 2.0 + 200 / 10 / (5.0 / 2)`	`10.5`
`10 > 5 + 4 && !(33 % 2 == 0 || 34 % 2 != 0)`	`true`
`1 + (2 + 3) + 4 + "5" + 6 + 7 + (8 + 9) * 10`	`"10567170"`
`(20 < 30) == (10 % 2 != 0)`	`false`

2. Array Mystery

Expression	Final Contents of Array
`int[] a1 = {42};` `arrayMystery(a1);`	`[42]`
`int[] a2 = {1, 8, 3, 6};` `arrayMystery(a2);`	`[1, 8, 4, 6]`
`int[] a3 = {5, 5, 5, 5, 5};` `arrayMystery(a3);`	`[5, 6, 5, 6, 5]`
`int[] a4 = {10, 7, 9, 6, 8, 5};` `arrayMystery(a4);`	`[10, 8, 10, 7, 9, 5]`
`int[] a5 = {1, 0, 1, 0, 0, 1, 0};` `arrayMystery(a5);`	`[1, 1, 1, 1, 0, 2, 0]`

3. Inheritance Mystery

```
Eins
Eins 1
Zwei 2   Eins 1

Eins
Eins 1
Eins 2

Vier Drei
Drei 1
Zwei 2   Drei 1   Vier 2

Drei
Drei 1
Zwei 2   Drei 1
```

4. File Processing (three solutions shown)

```
public static void coinFlip(Scanner input) {
    while (input.hasNextLine()) {
        Scanner lineScan = new Scanner(input.nextLine().toUpperCase());
        int heads = 0;
        int total = 0;
        while (lineScan.hasNext()) {
            total++;
            if (lineScan.next().equals("H")) {
                heads++;
            }
        }

        double percent = 100.0 * heads / total;
        System.out.printf("%d heads (%.1f%%)\n", heads, percent);
        if (percent > 50) {
            System.out.println("You win!");
        }
        System.out.println();
    }
}

public static void coinFlip(Scanner input) {
    while (input.hasNextLine()) {
        String line = input.nextLine();
        Scanner lineScan = new Scanner(line);

        int heads = 0;
        int tails = 0;
        while (lineScan.hasNext()) {
            String flip = lineScan.next();
            if (flip.equalsIgnoreCase("H")) {
                heads++;
            } else {
                tails++;
            }
        }

        double percent = 100.0 * heads / (heads + tails);
        System.out.printf("%d heads (%.1f%%)\n", heads, percent);
        if (percent > 50) {
            System.out.println("You win!");
        }
        System.out.println();
    }
}

public static void coinFlip(Scanner input) {
    while (input.hasNextLine()) {
        String line = input.nextLine().toLowerCase();
        int heads = 0;
        int tails = 0;
        for (int i = 0; i < line.length(); i++) {
            if (line.charAt(i) == 'h') {
                heads++;
            } else if (line.charAt(i) == 't') {
                tails++;
            }
        }

        double percent = 100.0 * heads / (heads + tails);
        System.out.printf("%d heads (%.1f%%)\n", heads, percent);
        if (percent > 50) {
            System.out.println("You win!");
        }
        System.out.println();
    }
}
```

5. Array Programming (four solutions shown)

```
public static int range(int[] a) {
    int min = 0;
    int max = 0;
    for (int i = 0; i < a.length; i++) {
        if (i == 0 || a[i] < min) {
            min = a[i];
        }
        if (i == 0 || a[i] > max) {
            max = a[i];
        }
    }

    int valueRange = max - min + 1;
    return valueRange;
}

public static int range(int[] a) {
    int min = a[0];
    int max = a[0];
    for (int i = 1; i < a.length; i++) {
        min = Math.min(min, a[i]);
        max = Math.max(max, a[i]);
    }
    return max - min + 1;
}

public static int range(int[] a) {
    int[] copy = new int[a.length];
    for (int i = 0; i < a.length; i++) {
        copy[i] = a[i];
    }
    Arrays.sort(copy);
    return copy[copy.length - 1] - copy[0] + 1;
}

public static int range(int[] a) {
    int range = 1;
    for (int i = 0; i < a.length; i++) {
        for (int j = 0; j < a.length; j++) {
            int difference = Math.abs(a[i] - a[j]) + 1;
            if (difference > range) {
                range = difference;
            }
        }
    }

    return range;
}
```

6. Array Programming (three solutions shown)

```
public static void zeroOut(int[] a1, int[] a2) {
    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[j]) {
                count++;
            }
        }

        if (count == a2.length) {   // found it
            for (int j = 0; j < a2.length; j++) {
                a1[i + j] = 0;
            }
        }
    }
}
```

```
public static void zeroOut(int[] a1, int[] a2) {
    int i2 = 0;
    for (int i1 = 0; i1 < a1.length; i1++) {
        if (a1[i1] == a2[i2]) {
            i2++;
            if (i2 == a2.length) {   // found it
                for (int i = 0; i < a2.length; i++) {
                    a1[i1 - i] = 0;
                }
                i2 = 0;
            }
        } else {
            i2 = 0;
        }
    }
}
```

```
public static void zeroOut(int[] a1, int[] a2) {
    int i1 = 0;
    int i2 = 0;
    while (i1 < a1.length) {
        if (a1[i1] == a2[i2]) {
            i2++;
            if (i2 == a2.length) {   // found it
                while (i2 > 0) {
                    a1[i1 - i2 + 1] = 0;
                    i2--;
                }
            }
        } else {
            i2 = 0;
        }
        i1++;
    }
}
```

7. Critters (two solutions shown)

```
public class Dragonfly extends Critter {
    private int moves;
    private int right;
    private int maxRight;
    private boolean up;

    public Dragonfly() {
        moves = 0;
        right = 0;
        maxRight = 1;
        up = false;
    }

    public boolean eat() {
        maxRight++;
        return true;
    }

    public Direction getMove() {
        moves++;
        if (right > 0) {
            right--;
            return Direction.EAST;
        } else {
            right = maxRight;
            up = !up;
            if (up) {
                return Direction.NORTH;
            } else {
                return Direction.SOUTH;
            }
        }
    }
}
```

```
public class Dragonfly extends Critter {
    private int moves = 0;
    private int east = 1;

    public boolean eat() {
        east++;
        return true;
    }

    public Direction getMove() {
        moves++;
        if (moves > 2 * east + 2) {
            moves = 1;
        }
        if (moves == 1) {
            return Direction.NORTH;
        } else if (moves == east + 2) {
            return Direction.SOUTH;
        } else {
            return Direction.EAST;
        }
    }
}
```

8. Objects (three solutions shown)

```
public int daysTillXmas() {
    int days = 0;
    Date copy = new Date(month, day);
    while (copy.month < 12) {
        // get to December
        days += copy.daysInMonth();
        copy.month++;
    }
    return days + 25 - day;
}

public int daysTillXmas() {
    if (month == 12) {
        return 25 - day;
    } else {
        int days = 0;
        Date copy = new Date(month, day);
        while (copy.month != 12 || copy.day != 25) {
            // get to December
            copy.nextDay();
            days++;
        }
        return days;
    }
}

public int daysTillXmas() {   // dreadful, but full-credit, solution
    if (month == 12) {
        return 25 - day;
    } if (month == 11) {
        return 55 - day;
    } if (month == 10) {
        return 86 - day;
    } if (month == 9) {
        return 116 - day;
    } if (month == 8) {
        return 147 - day;
    } if (month == 7) {
        return 178 - day;
    } if (month == 6) {
        return 208 - day;
    } if (month == 5) {
        return 239 - day;
    } if (month == 4) {
        return 269 - day;
    } if (month == 3) {
        return 300 - day;
    } if (month == 2) {
        return 328 - day;
    } else {
        return 359 - day;
    }
}
```