CSE 142
Sample Final Exam #3

1. Expressions (5 points)

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 (e.g., `7.0` rather than `7` for a `double`, `String`s in quotes).

Expression	Value
`3 * 1.0 + 15 / 2`	_____
`17 % 10 + 1 / 5 + 0.5 * 4`	_____
`15 % 3 / 4 + 22.0 / 4 * (14 / 4)`	_____
`!(3 > 27) || 0.03 > 0.2 && 2 < 3 + 5`	_____
`12 - 5 + "2 - 1" + 13 / 2`	_____

2. Array Mystery (10 points)

Consider the following method:

```
public static void mystery(int[] array) {
    for (int i = 0; i < array.length - 1; i++) {
        if (array[i] < array[i + 1]) {
            array[i] = array[i + 1];
        }
    }
}
```

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`.

Original Contents of Array	Final Contents of Array
`int[] a1 = {2, 4};` `mystery(a1);`	_____
`int[] a2 = {1, 3, 6};` `mystery(a2);`	_____
`int[] a3 = {7, 2, 8, 4};` `mystery(a3);`	_____
`int[] a4 = {5, 2, 7, 2, 4};` `mystery(a4);`	_____
`int[] a5 = {2, 4, 6, 3, 7, 9};` `mystery(a5);`	_____

3. Inheritance Mystery (10 points)

Assume that the following classes have been defined:

```
public class Ice extends Fire {
    public void method1() {
        System.out.println("Ice 1");
    }
}

public class Rain extends Fire {
    public String toString() {
        return "Rain";
    }

    public void method1() {
        System.out.println("Rain 1");
    }
}
```

```
public class Fire {
    public String toString() {
        return "Fire";
    }

    public void method1() {
        System.out.println("Fire 1");
    }

    public void method2() {
        System.out.println("Fire 2");
    }
}

public class Snow extends Rain {
    public void method2() {
        System.out.println("Snow 2");
    }
}
```

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

```
Fire[] elements = {new Fire(), new Snow(), new Rain(), new Ice()};
for (int i = 0; i < elements.length; i++) {
    System.out.println(elements[i]);
    elements[i].method1();
    elements[i].method2();
    System.out.println();
}
```

4. File Processing (15 points)

Write a static method named `halfCaps` that accepts as its parameter a `Scanner` holding a sequence of words and outputs to the console the same sequence of words with alternating casing (lowercase, uppercase, lowercase, uppercase, etc). The first word, third word, fifth word, and all other "odd" words should be in lowercase letters, whereas the second word, fourth word, sixth word, and all other "even" words should be in uppercase letters. For example, suppose the `Scanner` contains the following words.

```
The QUick brown foX jumPED over the Sleepy student
```

For the purposes of this problem, we will use whitespace to separate words. You can assume that the sequence of words will not contain any numbers or punctuation and that each word will be separated by one space. For the input above, your method should produce the following output:

```
the QUICK brown FOX jumped OVER the SLEEPY student
```

Your output should separate each word by a single space. The output may end with a space if you like. Note that the `Scanner` may contain no words or may contain an even or odd number of words.

5. File Processing (10 points)

Write a static method named `countWords` that accepts as its parameter a `Scanner` for an input file, and that outputs to the console the total number of lines and words found in the file as well as the average number of words per line. For example, consider the following input file:

```
You must show: your Student ID card

to 1) a TA or 2) the instructor
before

leaving the room.
```

For the purposes of this problem, we will use whitespace to separate words. That means that some words might include punctuation, as in `"show:"` and `"1)"`. (This is the same definition that the `Scanner` uses for tokens.) For the input above, your method should produce the following output:

```
Total lines = 6
Total words = 19
Average words per line = 3.1666666666666665
```

The format of your output must exactly match that shown above. Notice that some input lines can be blank. Do not worry about rounding the average words per line. You may assume that the `Scanner` contains at least 1 line of input.

6. Array Programming (15 points)

Write a static method named `mode` that takes an array of integers as a parameter and that returns the value that occurs most frequently in the array. Assume that the integers in the array appear in sorted order. For example, if a variable called `list` stores the following values:

```
int[] list = {-3, 1, 4, 4, 4, 6, 7, 8, 8, 8, 8, 9, 11, 11, 11, 12, 14, 14};
```

Then the call of `mode(list)` should return `8` because 8 is the most frequently occurring value in the array, appearing four times.

If two or more values tie for the most occurrences, return the one with the lowest value. For example, if the array stores the following values, the call of `mode(list)` should return `2` despite the fact that there are also three 9s:

```
int[] list = {1, 2, 2, 2, 5, 7, 9, 9, 9};
```

If the array's elements are unique, every value occurs exactly once, so the first element value should be returned. You may assume that the array's length is at least 1. If the array contains only one element, that element's value is considered the mode.

7. Array Programming (10 points)

Write a static method named `contains` that accepts two arrays of integers *a1* and *a2* as parameters and that returns a `boolean` value indicating whether or not *a2*'s sequence of elements appears in *a1* (`true` for yes, `false` for no). 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 `list1` and `list2` store the following values:

```
int[] list1 = {1, 6, 2, 1, 4, 1, 2, 1, 8};
int[] list2 = {1, 2, 1};
```

Then the call of `contains(list1, list2)` should return `true` because `list2`'s sequence of values `{1, 2, 1}` is contained in `list1` starting at index 5. If `list2` had stored the values `{2, 1, 2}`, the call of `contains(list1, list2)` would return `false` because `list1` does not contain that sequence of values. Any two lists with identical elements are considered to contain each other, so a call such as `contains(list1, list1)` should return `true`.

You may assume that both arrays passed to your method will have lengths of at least 1. You may not use any `String`s to help you solve this problem, nor methods that produce `String`s such as `Arrays.toString`.

8. Critters (15 points)

Write the `getMove` method for the class `Shark` that implements the `Critter` interface from Homework 8. Instances of the `Shark` class should alternate between moving to the North and South as follows: first move 1 to the NORTH, then 2 to the SOUTH, then 3 to the NORTH, then 4 to the SOUTH, then 5 to the NORTH, then 6 to the SOUTH, and so on, each time moving one farther than previously. Use the constants for directions defined in the `Critter` interface, namely `NORTH`, `SOUTH`, `EAST`, `WEST`, and `CENTER`.

You may add anything needed (fields, other methods, constructors, etc.) to implement `getMove` appropriately.

```
import java.awt.*; // for Color

public class Shark implements Critter {
    // declare any necessary fields here









    // fight, getColor, toString methods omitted (you do not need to write them)

    public int getMove(CritterInfo info) {
        // complete the getMove method here
```

```
    }
}
```

9. Classes (10 points)

This question uses the `Date` class as specified at right (this is the same `Date` class as seen on your sample final exams).

Write a method named `addDays` that will be placed inside the `Date` class. The `addDays` method accepts an integer as a parameter and shifts the date represented by the `Date` object forward by that many days. You may assume the value passed is non-negative.

For example, if the following `Date` is declared in client code:

```
Date d = new Date(9, 19);
```

The following calls to the `addDays` method would modify the `Date` object's state as indicated in the comments. Remember that `Date` objects do not store the year; the date after December 31st is January 1st.

```
d.addDays(1);     // d now represents Sep. 20
d.addDays(1);     // d now represents Sep. 21
d.addDays(5);     // d now represents Sep. 26
d.addDays(10);    // d now represents Oct. 6
d.addDays(80);    // d now represents Dec. 25
d.addDays(10);    // d now represents Jan. 4
                  // (of the next year)
d.addDays(1000);  // d now represents Oct. 1
                  // (over two years later)
```

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

    public Date(int m, int d) {
        month = m;
        day = d;
    }

    public int getDay() {
        return day;
    }

    public int getMonth() {
        return month;
    }

    // returns the number of days
    // in the given month
    public int numDays(int m) {
        if (m == 2) {
            return 28;
        } else if (m == 4 ||
                   m == 6 ||
                   m == 9 ||
                   m == 11) {
            return 30;
        } else {
            return 31;
        }
    }

    // your method would go here
}
```

Solutions

1.

Expression	Value
`3 * 1.0 + 15 / 2`	`10.0`
`17 % 10 + 1 / 5 + 0.5 * 4`	`9.0`
`15 % 3 / 4 + 22.0 / 4 * (14 / 4)`	`16.5`
`!(3 > 27) || 0.03 > 0.2 && 2 < 3 + 5`	`true`
`12 - 5 + "2 - 1" + 13 / 2`	`"72 - 16"`

2.

Array	Final contents
`{2, 4}`	`{4, 4}`
`{1, 3, 6}`	`{3, 6, 6}`
`{7, 2, 8, 4}`	`{7, 8, 8, 4}`
`{5, 2, 7, 2, 4}`	`{5, 7, 7, 4, 4}`
`{2, 4, 6, 3, 7, 9}`	`{4, 6, 6, 7, 9, 9}`

3.

```
Fire
Fire 1
Fire 2

Rain
Rain 1
Snow 2

Rain
Rain 1
Fire 2

Fire
Ice 1
Fire 2
```

4. Three solutions are shown.

```
public static void halfCaps(Scanner input) {
    boolean odd = true;
    while (input.hasNext()) {
        String next = input.next();
        if (odd) {
            System.out.print(next.toLowerCase() + " ");
        } else {
            System.out.print(next.toUpperCase() + " ");
        }
        odd = !odd;
    }
}
```

```
public static void halfCaps(Scanner input) {
    int count = 0;
    while (input.hasNext()) {
        if (count % 2 == 0) {
            System.out.print(input.next().toLowerCase() + " ");
        } else {
            System.out.print(input.next().toUpperCase() + " ");
        }
        count++;
    }
}
```

```
public static void halfCaps(Scanner input) {
    while (input.hasNext()) {
        System.out.print(input.next().toLowerCase() + " ");
        if (input.hasNext()) {
            System.out.print(input.next().toUpperCase() + " ");
        }
    }
}
```

5.

```
public static void countWords(Scanner input) {
    int lineCount = 0;
    int wordCount = 0;

    while (input.hasNextLine()) {
        String line = input.nextLine();
        lineCount++;
        Scanner lineScan = new Scanner(line);
        while (lineScan.hasNext()) {
            String next = lineScan.next();
            wordCount++;
        }
    }

    double averageWords = (double) wordCount / lineCount;
    System.out.println("Total lines = " + lineCount);
    System.out.println("Total words = " + wordCount);
    System.out.println("Average words per line = " + averageWords);
}
```

6.

```
public static int mode(int[] a) {
    int count = 1;
    int maxCount = 1;
    int modeValue = a[0];

    for (int i = 0; i < a.length - 1; i++) {
        if (a[i] == a[i + 1]) {
            count++;
            if (count > maxCount) {
                modeValue = a[i];
                maxCount = count;
            }
        } else {
            count = 1;
        }
    }

    return modeValue;
}
```

7. Four solutions are shown.

```
public static boolean contains(int[] a1, int[] a2) {
    for (int i = 0; i <= a1.length - a2.length; i++) {
        boolean found = true;
        for (int j = 0; j < a2.length; j++) {
            if (a1[i + j] != a2[j]) {
                found = false;
            }
        }
        if (found) {
            return true;
        }
    }
    return false;
}

// varation of first solution that uses count instead of boolean
public static boolean contains(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)
            return true;
    }
    return false;
}
```

```
public static boolean contains(int[] a1, int[] a2) {
    int i1 = 0;
    int i2 = 0;
    while (i1 < a1.length && i2 < a2.length) {
        if (a1[i1] != a2[i2]) {  // doesn't match; start over
            i2 = 0;
        }
        if (a1[i1] == a2[i2]) {
            i2++;
        }
        i1++;
    }

    return i2 >= a2.length;
}

public static boolean contains(int[] a1, int[] a2) {
    for (int i = 0; i < a1.length; i++) {
        int j = 0;
        while (j < a2.length && i + j < a1.length && a1[i + j] == a2[j])
            j++;
        if (j == a2.length)
            return true;
    }
    return false;
}
```

8. Two solutions are shown.

```
public class Shark implements Critter {
    private int count;
    private int max;
    private int direction;

    public Shark() {
        count = 0;
        max = 1;
        direction = NORTH;
    }
    ...

    public int getMove(CritterInfo info) {
        count++;
        if (count > max) {
            count = 1;
            max++;
            if (direction == NORTH) {
                direction = SOUTH;
            } else {
                direction = NORTH;
            }
        }
        return direction;
    }
}
```

```
public class Shark implements Critter {
    private int count = 0;
    private int max = 1;
    ...

    public int getMove(CritterInfo info) {
        count++;
        if (count > max) {
            count = 1;
            max++;
        }
        if (max % 2 == 0) {
            return SOUTH;
        } else {
            return NORTH;
        }
    }
}
```

9. Three solutions are shown.

```
public void addDays(int days) {
    for (int i = 0; i < days; i++) {
        day++;
        if (day > numDays(month)) {
            day = 1;
            month++;
        }
        if (month > 12) {
            month = 1;
            day = 1;
        }
    }
}
```

```
public void addDays(int days) {
    while (day + days > numDays(month)) {
        days -= numDays(month);
        month++;
        if (month > 12) {
            month = 1;
        }
    }
    day += days;
}
```

```
public void addDays(int days) {
    day += days;
    while (day > 365) {
        day -= 365;
    }
    while (day > numDays(month)) {
        day -= numDays(month);
        month++;
        if (month > 12) {
            month = 1;
        }
    }
}
```