CSE 142 Sample Final Exam #3

(based on Autumn 2007's final)

1. 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);`	_____

2. Reference Semantics Mystery

The following program produces 4 lines of output. Write the output below, as it would appear on the console.

```
import java.util.*;   // for Arrays class

public class ReferenceMystery {
    public static void main(String[] args) {
        int x = 0;
        int[] a = new int[4];

        x++;
        mystery(x, a);
        System.out.println(x + " " + Arrays.toString(a));

        x++;
        mystery(x, a);
        System.out.println(x + " " + Arrays.toString(a));
    }

    public static void mystery(int x, int[] a) {
        x++;
        a[x]++;
        System.out.println(x + " " + Arrays.toString(a));
    }
}
```

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. File Processing

Write a static method named `findFirstMatch` that accepts as its parameters a `Scanner` for an input file and an array of `String`s *keywords* representing a list of keywords in a search. Your method will read lines from its input `Scanner` and should return the line number of the first line in the file that contains one or more words from *keywords*. If none of the keywords are found in the file, your method should return a -1. The search should be case-insensitive, so if a keyword was "banana", the line "yummy baNAna split" would be considered a line that contains the keyword. Your method should also match **whole words** only, so if the only keyword in the array was "ball", the line "football game" would *not* be considered a match.

For example, consider the following input file saved in `sidewalk.txt`, consisting of 6 lines:

```
Let us leave this place where the smoke blows black
And the dark street winds and bends.
Past the pits where the asphalt flowers grow
We shall walk with a walk that is measured and slow,
And watch where the chalk-white arrows go
To the place where the sidewalk ends.
```

The following table shows some calls to your method and their expected results with the following `Scanner`:

```
Scanner input = new Scanner(new File("sidewalk.txt"));
```

Array	Call / Returned Value
`String[] k1 = {"place", "winds"};`	`findFirstMatch(input, k1)` returns `1`
`String[] k2 = {"dinosaur", "PITS", "pots"};`	`findFirstMatch(input, k2)` returns `3`
`String[] k3 = {"chalk", "row", "g", "ends"};`	`findFirstMatch(input, k3)` returns `-1`
`String[] k4 = {"to"};`	`findFirstMatch(input, k4)` returns `6`

You may assume that none of the words in the *keywords* array contain spaces, i.e. all keywords are single whole words, and the array contains at least one element. Do not modify the elements of the *keywords* array.

6. 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.

7. 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*.

8. 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.

9. Classes and Objects

Suppose that you are provided with a pre-written class `BankAccount` as described at right. (The headings are shown, but not the method bodies, to save space.) Assume that the fields, constructor, and methods shown are already implemented. You may refer to them or use them in solving this problem if necessary.

Write an instance method named **`transactionFee`** that will be placed inside the `BankAccount` class to become a part of each `BankAccount` object's behavior. The `transactionFee` method accepts a fee amount (a real number) as a parameter, and applies that fee to the user's past transactions. The fee is applied once for the first transaction, twice for the second transaction, three times for the third, and so on. These fees are subtracted out from the user's overall balance. If the user's balance is large enough to afford all of the fees with greater than $0.00 remaining, the method returns `true`. If the balance cannot afford all of the fees, the balance is left as 0.0 and the method returns `false`.

For example, given the following `BankAccount` object:

```
BankAccount savings = new BankAccount("Jimmy");
savings.deposit(10.00);
savings.deposit(50.00);
savings.deposit(10.00);
savings.deposit(70.00);
```

The account at that point has a balance of $140.00. If the following call were made:

```
savings.transactionFee(5.00)
```

Then the account would be deducted $5 + $10 + $15 + $20 for the four transactions, leaving a final balance of $90.00. The method would return `true`. If a second call were made,

```
savings.transactionFee(10.00)
```

Then the account would be deducted $10 + $20 + $30 + $40 for the four transactions, leaving a final balance of $0.00. The method would return `false`.

```
// A BankAccount keeps track of a
// user's money balance and ID,
// and counts how many transactions
// (deposits/withdrawals) are made.

public class BankAccount {
    private String id;
    private double balance;
    private int transactions;

    // Constructs a BankAccount
    // object with the given id, and
    // 0 balance and transactions.
    public BankAccount(String id)

    // returns the field values
    public double getBalance()
    public String getID()

    // Adds the amount to the balance
    // if it is between 0-500.
    // Also counts as 1 transaction.
    public void deposit(double amount)

    // Subtracts the amount from
    // the balance if the user has
    // enough money.
    // Also counts as 1 transaction.
    public void withdraw(double amount)


    // your method would go here

}
```

Solutions

1. Array Mystery

Call	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]`

2. Reference Semantics Mystery

```
2 [0, 0, 1, 0]
1 [0, 0, 1, 0]
3 [0, 0, 1, 1]
2 [0, 0, 1, 1]
```

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. File Processing (two solutions shown)

```
public static int findFirstMatch(Scanner input, String[] keywords) {
    int lineNum = 0;
    while (input.hasNextLine()) {
        String line = input.nextLine();
        Scanner lineScan = new Scanner(line);
        lineNum++;
        while (lineScan.hasNext()) {
            String word = lineScan.next();
            for (int i = 0; i < keywords.length; i++) {
                if (keywords[i].equalsIgnoreCase(word)) {
                    return lineNum;
                }
            }
        }
    }
    return -1;
}

public static int findFirstMatch(Scanner s, String[] k) {
    String[] sorted = Arrays.copyOf(k, k.length);
    for (int i = 0; i < sorted.length; i++) {
        sorted[i] = sorted[i].toLowerCase();
    }
    Arrays.sort(sorted);
    for (int line = 1; s.hasNextLine(); line++) {
        Scanner words = new Scanner(s.nextLine());
        while (words.hasNext()) {
            if (Arrays.binarySearch(sorted, words.next().toLowerCase()) >= 0) {
                return line;
            }
        }
    }
    return -1;
}
```

6. 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;
}
```

7. 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;
            }
            i += a2.length - 1; // skip over the parts that have been zeroed out
        }
    }
}
```

```
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++;
    }
}
```

8. 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;
        }
    }
}
```

9. Objects (two solutions shown)

```
public boolean transactionFee(double amount) {
    for (int i = 1; i <= transactions; i++) {
        balance -= amount * i;
    }

    if (balance > 0.0) {
        return true;
    } else {
        balance = 0.0;
        return false;
    }
}
```

```
public boolean transactionFee(double amount) {
    for (int i = 1; i <= transactions; i++) {
        balance = Math.max(0.0, balance - amount * i);
    }
    return balance > 0.0;
}
```