Exam Number:

Points off	1	2	3	4	5	6	7	8	Total off	Net Score

CS 305j – Final – Spring 2010

Your Name_____

Your UTEID _____

Instructions:

1. Please turn off your cell phones.
2. There are 8 questions on this test.
3. You have 3 hours to complete the test.
4. You may not use a calculator.
5. When code is required, write Java code.
6. You may add helper methods if you wish.

1. Expressions. (10 points). For each Java expression in the left hand column, indicate its value in the right hand column. Be sure to show a constant of the appropriate type. For example, 7.0 rather than 7 for a `double`, `Strings` in double quotes, (For example `"dog"` instead of `dog`.), and `true` or `false` for `booleans`.

A. `2000 / 10 / 20 * 5 / 2` _____

B. `21 % 5 + 7 % 20` _____

C. `1 + 3 + "IT" + (3 + 5)` _____

D. `10 / 7 / 10 / 100` _____

E. `"SC" + 3 * 5` _____

F.
```
"STANFORD".charAt(2) == 'A'
     && "STANFORD".length() > 7
```
_____

G. `(int)(3.3 * 3 * 10)` _____

H. `(3 > 5) || (5 * 2 != 10) || (3 * 3 > 10 / 3)` _____

I. `(double)5 / 2` _____

J. `(10 != 5 * 2) && (10 == 5 * 2)` _____

2. **Program Logic** (10 points) Consider the following method. For each of the 5 points labeled by comments and each of the 4 assertions in the table write whether the assertion is ***always*** true, ***sometimes*** true, or ***never*** true at that point in the code. Abbreviate ***always*** with an A, ***sometimes*** with an S and ***never*** with an N. (Each answer is worth 0.5 points)

```
public static int find(int[] list, int tgt){
      int result = -1;
      if( list != null ){
            int low = 0;
            int high = list.length - 1;
            int mid = (low + high) / 2;
            // Point A
            while( result == -1 && low <= high ) {
                  mid = (low + high) / 2;
                  // Point B
                  if( list[mid] == tgt ) {
                        result = mid;
                        // Point C
                  }
                  else if( list[mid] < tgt ) {
                        low = mid + 1;
                  }
                  else {
                        high = mid - 1;
                  }
                  // Point D
            }
            // Point E
      }
      return result;
}
```

Abbreviate ***always*** with an A, ***sometimes*** with an S and ***never*** with an N.

	result == -1	low <= high	list[mid] == tgt	low == 0
Point A				
Point B				
Point C				
Point D				
Point E				

3. Simple methods. (10 points) Write a method to determine the overall price for an airline ticket. The method is given the base price for the ticket, a char indicating the day of the week for the ticket, a boolean indicating if the flight is non-stop, and a boolean indicating if the ticket is a first class ticket.

The char indicating the day of the week will be 'M' for Monday, 'T' for Tuesday, 'W' for Wednesday, 'H' for Thursday, 'F' for Friday, 'S' for Saturday, and 'U' for Sunday.

If the flight is nonstop the ticket price is 40% more than the base price.

If the fight takes place on Monday or Friday the ticket price is 30% more than the base price.

If the flight is non-stop and takes place on Monday or Friday then the ticket price is 100% more than the base price.

If the ticket is a first class ticket then the price is increase by 150%. This change in price is applied after any other changes in prices and is in addition to the increases due to the day of the week of the flight and whether the flight is non-stop or not. So for example if a ticket has a base price of $100, is on Sunday, is non-stop, and first class, then the ticket price goes up 40% because it is non-stop. This gives a price of $140. The ticket is also first class so it goes up an additional 150%. 150% of $140 is $210. So the final ticket price is $140 + $210 = $350.

All calculations are rounded down as in integer division.

The method header is:

```
public static int getTicketPrice(int basePrice, char day, boolean nonStop,
                                  boolean firstClass) {
```

Examples of method calls and results:

```
getTicketPrice(100, 'S', false, false) returns 100
getTicketPrice(100, 'S', false, true) returns 250
getTicketPrice(100, 'S', true, false) returns 140
getTicketPrice(100, 'S', true, true) returns 350
getTicketPrice(100, 'M', false, false) returns 130
getTicketPrice(100, 'M', false, true) returns 325
getTicketPrice(100, 'M', true, false) returns 200
getTicketPrice(100, 'M', true, true) returns 500
getTicketPrice(90, 'M', true, true) returns 450
```

Complete the following method on the next page.

```
public static int getTicketPrice(int basePrice, char day, boolean nonStop,
                                  boolean firstClass) {
```

Complete this method on the next page.

Complete this method on the next page.

Complete this method on the next page.

Complete this method on the next page.

```
/ I expect basePrice > 0, char will be 'M', 'T', 'W', 'H', 'F', 'S', or 'U'
public static int getTicketPrice(int basePrice, char day, boolean nonStop,
                                 boolean firstClass) {
```

4. Arrays (12 points) In class and on one of the assignments we used a frequency map, is an array that represents a count of how often characters occur in a file. For this question we will use a frequency map for the characters `'a'` through `'z'`. The frequency map will be an array of `int`s with a length of 26. Element 0 is the count of how many times `'a'` occurred, element 1 is the count of how many times `'b'` occurred and so forth.

Write a method that takes in an array of ints (the frequency map) as a parameter and returns a `String`. The String contains all the letters from the frequency map that occur less than 2% of the time based on the frequency map. So for example if the sum of all frequencies is 1000, any character that occurred fewer than 20 times would be added to the String that is returned.

Recall an `int` from `0` to `25` can be converted to the corresponding char `'a'` through `'z'` with the following expression if `i` is an int.

```
char ch = (char)(i + 'a');
```

Complete the following method:

```
// I expect frepMap.length = 26.
// return a String with letters than occur less than 2% of the time based
// on the frequency map. If no letters occur less than 2% of the time
// return an empty String.
public static String infrequentCharacters(int[] freqMap) {
```

5. ArrayLists. (13 points) Write a method that given an `ArrayList` of `Strings` and an array of `ints` that represents a frequency map, remove any `Strings` in the `ArrayList` that consist only of infrequent characters. Return those `Strings` in a new `ArrayList`.

Call method `infrequentCharacters` from question 4 to find the infrequent characters. Assume the method `infrequentCharacters` works regardless of what you wrote in question 4.

For example if the ArrayList contained these elements:

```
["cat", "dog", "XY", "X!!!", "xqz", "the"]
```

and the infrequent characters based on the frequency map were `"jkquvxyz"` then the original `ArrayList` would become this:

```
["cat", "dog", "X!!!", "the"]
```

and the returned `ArrayList` would be:

```
["XY", "xqz"]
```

You are to ignore case for the letters in the `Strings`. Thus `'X'` and `'x'` are considered equivalent in this question.

You may only use the following `String` methods on this question:

`public char` **`charAt`**`(int index)`
Returns the `char` value at the specified index.

`public int` **`length`**`()`
Returns the length of this string.

`public String` **`toLowerCase`**`()`
Returns a new String with all of the characters in the original `String` converted to lower case.

`public int` **`indexOf`**`(char ch)`
Returns the index within this string of the first occurrence of the specified character. Returns -1 if ch is not present in this String.

You may only use the following `ArrayList` methods on this question. For this question the data type `E` will be `String`.

`public int` **`size`**`()`
Returns the number of elements in this list.

`public boolean` **`add`**`(E e)`
Appends the specified element to the end of this list. (E will be `String` in this question.)

`public E` **`get`**`(int index)`
Returns the element at the specified position in this list.

`public E` **`remove`**`(int index)`
Removes the element at the specified position in this list. Shifts any subsequent elements to the left (subtracts one from their indices).

And you can use the `ArrayList` constructor. Recall the command to create a new, empty `ArrayList` of `Strings` is:

```
ArrayList<String> result = new ArrayList<String>();
```

Complete the following method:

```
// I expect frepMap.length = 26 and org != null
// Complete the method as described in the question.
public static ArrayList<String> removeWords(ArrayList<String> org,
                                            int[] freqMap) {
```

6. Implementing classes. (17 points) Write a complete `FlyTrap` class that implements the `Critter` interface from assignment 11.

- When created `FlyTraps`'s are a random color. 80% of the time the new `FlyTrap` is green (`Color.GREEN`) and 20% of the time it is red (`Color.RED`). The color is picked when the `FlyTrap` is created and never changes for a given `FlyTrap`.
- When created `FlyTrap`'s pick a direction to move. They have a 50% chance of picking north, a 25% chance of picking west, and a 25% chance of picking east.
- `FlyTrap`'s that are green always fight with `SCISSORS`. `FlyTrap`'s that are red randomly choose between `ROCK` and `PAPER` each time that fight with each having a 50% chance of being picked for the current fight.
- The `toString` method for `FlyTrap`'s always returns a letter corresponding to the weapon the `FlyTrap` used in the last fight. `S` for scissors, `P` for paper, and `R` for rock. If a `FlyTrap` has never been in a fight it returns an `F`.
- When moving `FlyTraps` have a 75% chance of standing still (not moving) and a 25% chance of moving their chosen direction.
- Create a single constructor with no parameters for the `FlyTrap` class.
- You may use either the `Math.random` method or the `Random` class to generate random numbers.

```
public interface Critter {
    // methods to be implemented
    public int fight(String opponent);
    public Color getColor();
    public int getMove(CritterInfo info);
    public String toString();

    // Definitions for NORTH, SOUTH, EAST, WEST, CENTER,
    // ROCK, PAPER, and SCISSORS
}
```

Complete your complete `FlyTrap` class below. Assume the `Color` and `Random` classes have been imported correctly.

```
// more room for the FlyTrap class on the next page.
```

```
// more room for the FlyTrap class if needed.
```

7. 2d arrays (18 points) You completed a program that allowed users to play the game of connect 4. There is a variation of connect 4, called variable gravity connect 4. In this version of the game after each player drops a checker the board is rotated 180 degrees. (So the top row is the now the bottom row and the bottom row is now the top row.) Obviously the checkers will fall towards the "new" bottom row. At first this does not have a big effect but as the game goes on it can.

Consider the following examples from a game:

red drops in column 4

```
Board before flip:
1 2 3 4 5 6 7
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .
. . . r . . .

Board after flip:
1 2 3 4 5 6 7
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .
. . . r . . .
```

black drops in column 4

```
Board before flip:
1 2 3 4 5 6 7
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .
. . . b . . .
. . . r . . .

Board after flip:
1 2 3 4 5 6 7
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .
. . . r . . .
. . . b . . .
```

red drops in column 5

```
Board before flip:
1 2 3 4 5 6 7
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .
. . . r . . .
. . . b r . .
```

```
Board after flip:
1 2 3 4 5 6 7
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .
. . . b . . .
. . . r r . .
```

... and later on:

Board at the start of red's turn

```
Current Board
1 2 3 4 5 6 7
. . . . . . .
. . . . . . .
. . . . . . r
. . . . . . r
. . . r r . b
. . . b b . b
```

red drops in column 6

```
Board before flip:
1 2 3 4 5 6 7
. . . . . . .
. . . . . . .
. . . . . . r
. . . . . . r
. . . r r . b
. . . b b r b
```

```
Board after flip:
1 2 3 4 5 6 7
. . . . . . .
. . . . . . .
. . . . . . b
. . . . . . b
. . . b b . r
. . . r r r r
```

This is a win for red on the bottom!

Complete a method that takes in a 2d array of chars that represents a connect 4 board and change the board so that it has been flipped and the checkers have moved as appropriate. All characters in the board will be periods, r's, or b's. The periods (`'.'`) represent open spaces.

Complete the method on the next page:

```
// I assume board is a rectangular matrix meaning each row has the same
// number of columns. There will be at least 1 row and at least one column
// per row.
// Flip the board as described in the question. Changes are made to board.
public static void flipBoard(char[][] board)
```

8. Simulation. (12 points) Write a method to carry out a biological simulation. The simulation involves 2 types of bacteria types A and B.

The method will be sent 2 parameters indicating the initial number of types A and B bacteria.

During each day of the simulation the following actions occur:

1. Each type B bacteria, one at a time, eats 3 type A bacteria. Any type B bacterium that does not eat 3 type A bacteria dies.

2. For every remaining 2 type A bacteria 1 more type A bacterium is created. Partial bacteria are not created.

3. For every 10 remaining type B bacteria 1 more type B bacterium is created. Partial bacteria are not created.

Write a method that has parameters indicating the starting number of each type of bacteria and prints out the number of each type of bacteria at the end of the day until there are zero type B bacteria left.

```
Day 0. Population of each type, A and B: 100 10
Day 1. Population of each type, A and B: 105 11
Day 2. Population of each type, A and B: 108 12
Day 3. Population of each type, A and B: 108 13
Day 4. Population of each type, A and B: 103 14
Day 5. Population of each type, A and B: 91 15
Day 6. Population of each type, A and B: 69 16
Day 7. Population of each type, A and B: 31 17
Day 8. Population of each type, A and B: 1 11
Day 9. Population of each type, A and B: 1 0
```

Note: On day 1, 30 type A bacteria are eaten leaving 70. Then 35 are created making the total type A at the end of the day equal to 105.

On day 2 there are 11 type B bacteria. There are enough type A bacteria for them all. Since partial bacteria cannot be created 1 new Type B bacteria is created, not 1.1.

On day 8 the Type B bacteria start to starve. There are only 31 type A bacteria, only enough for 10 type B bacteria to survive. 1 more type B bacteria is created after the original 10 eat.

Complete the following method on the next page:

```
// I expect typeA > 0 and typeB > 0
public static void runSim(int typeA, int typeB)
```

Complete this method on the next page.

Complete this method on the next page.

Complete this method on the next page.

Complete this method on the next page.

Complete this method on the next page.

```
// I expect typeA > 0 and typeB > 0
public static void runSim(int typeA, int typeB)
```