Exam Number:

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

CS 305j – Final – Fall 2008

Your Name_____

Your UTEID _____

Circle you TA's name: Ann Alex

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. `10 / 3 + 2 * 5` _____

B. `12 % 10` _____

C. `1 + 2 + "UT"` _____

D. `1.0 + 3 / 5` _____

E. `3.0 / 2` _____

F. `100 / 5 / 2 / 5` _____

G. `"Long" + 5 + 5 / 2` _____

H. `(6 > 5) && (10 > 5) && (5 > 10)` _____

I. `(12 % 2 == 0) || (100 % 17 > 10)` _____

J. `!("Texas".charAt(1) == 'e')` _____

2. **Program Logic** (9 points) Consider the following method. For each of the 5 points labeled by comments and each of the 4 assertions in the table (except for the two marked `No answer required.`), 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 indexOfLast(int[] nums, int tgt){
      int result = -1;
      int i = 0;
      // Point A

      if( nums.length > 0 ){
            // Point B
            for(i = 0; i < nums.length; i++){

                  if( nums[i] > tgt ){
                        result = i;
                        // Point C
                  } // end of if

                  // Point D
            } // end of for

      } // end of if

      // Point E
      return result;
}
```

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

	`result == -1`	`i < nums.length`	`nums[i] > tgt`	`result == i`
`Point A`			`No answer required.`	
`Point B`				
`Point C`				
`Point D`				
`Point E`			`No answer required.`	

3. Boolean Logic. (11 points) Write a `static` method that returns whether it is appropriate to go for a bike ride or not based on weather conditions.

The parameters to the method are the temperature, the time of day, and whether it is raining or not.

Temperature will be an int between -100 and 130 inclusive.
Time of day will be an int between 0 and 23 inclusive. (This is based on a 24 hour clock.)
The status of rain will be expressed as a `boolean`.

Here are the conditions necessary for the method to return `true`.

- The time of day must be greater than or equal to 6 and less than or equal to 18.
- AND the temperature must be greater than 30 and less than 110, unless it is raining in which case the temperature must be greater than 70 and less than 115.

The method header is:

```
public static boolean okToRide(int temp, int timeOfDay, boolean isRaining){
```

Examples of expected results:

```
okToRide(80, 4, false) -> false  because of time of day.
okToRide(80, 12, false) -> true
okToRide(80, 6, true) -> true
okToRide(60, 12, false) -> true
okToRide(60, 12, true) -> false  because of status of rain and temperature.
okToRide(80, 12, true) -> true
okToRide(80, 20, false) -> false  because of time of day.
okToRide(20, 12, false) -> false  because of temperature.
okToRide(30, 12, false) -> false  because of temperature.
```

Complete the method below.

```
public static boolean okToRide(int temp, int timeOfDay, boolean isRaining){
```

4. Using Objects. (15 points) Complete a `static boolean` method that determines if an array of `Card` objects is *pretty*.

An array of Cards is pretty if:

- All of the cards are face cards. For this problem, cards whose rank is jack, queen, king, or ace are considered face cards.
- OR all the cards have a rank of 10 or less (not including aces) and are the same color. The suits hearts and diamonds are red and the suits spades and clubs are black.

Return `true` if the array sent to the method constitutes a pretty array of `Card` objects, `false` otherwise.
The method you are writing is not in the `Card` class.
Important: You may not use the ArrayList class or the static methods from the Arrays class when completing this method.

Here is the public interface of the `Card` class you may use for this problem:

```
public class Card implements Comparable{

    public static final int TWO = 0;
    public static final int THREE = 1;
    public static final int FOUR = 2;
    public static final int FIVE = 3;
    public static final int SIX = 4;
    public static final int SEVEN = 5;
    public static final int EIGHT = 6;
    public static final int NINE = 7;
    public static final int TEN = 8;
    public static final int JACK = 9;
    public static final int QUEEN = 10;
    public static final int KING = 11;
    public static final int ACE = 12;

    public static final int CLUBS = 0;
    public static final int DIAMONDS = 1;
    public static final int HEARTS = 2;
    public static final int SPADES = 3;

    // create a new card with the given rank and suit. Rank and suit must
    // be valid values based on the class constants.
    public Card(int suit, int rank)

    // return this Card's suit
    public int getSuit()

    // return this Card's rank
    public int getRank()

    // return an int < 0 if this Card is less than other
    // return an int == 0 if this Card is equal to other
    // return an int > 0 if this Card is greater than other
    // all results based on Cards' rank only. Suit does not affect results.
    public int compareTo(Card other)
}
```

```
// Assume hand.length > 0 and none of the elements of hand are null.
// Return true if the Cards in the array hand constitute a pretty array of
// Cards as described in the problem statement, false otherwise.
public static boolean isPretty(Card[] hand){
```

5. Implementing classes. (15 points) Write a complete `Badger` class that implements the `Critter` interface from assignment 11.

- When created `Badger`'s are a random color. 70% of the time a new `Badger` is black (`Color.BLACK`), 30% of the time it is gray (`Color.GRAY`), and 10% of the time it is white (`Color.WHITE`). The color is picked when the Badger is created and never changes for a given badger.
- The `toString` method for `Badger`'s returns a String that is the first letter of their color. So `"B"` for black, `"G"` for gray, and `"W"` for white.
- When `Badger`s fight they randomly pick between `Rock`, `Paper`, and `Scissors`. That same choice is used for 3 fights and then they pick a new random weapon (`Rock`, `Paper`, or `Scissors`). That choice is then used for the next 3 fights and so forth.
- `Badger`s always move North.
- Create a single constructor with no parameters for the `Badger` 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 `Badger` class below:

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

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

6. 2d arrays (15 points) Complete a `static` method that determines which 2 by 2 sub-array in a given 2d array of `int`s contains the largest sum.

- The method will be sent a 2d array of `int`s that is rectangular. (Same number of columns in every row.)
- Determine which 2 by 2 sub-array in the overall array has the maximum sum.
- Return the row and column number of the upper left cell of the 2 by 2 sub-array that has the maximum sum. The row and column indices of the maximum sub-array are returned as an array of `int`s of length 2.
- The first element in the returned array is the row index and the second is the column index of the upper left cell of the 2 by 2 sub-array that has the maximum sum. In the case of a tie choose the 2 by 2 sub-array that has the row closest to 0. If there are multiple sub-arrays with the smallest row choose the sub-array that has the column closest to 0.

As an example, consider if the following 2d array was passed to the method: (The top row shows the column indices and the left column shows the row indices.)

	0	1	2	3	4	<- column indices
0	1	9	9	9	5	
1	9	1	0	9	5	
2	5	9	2	9	7	

^
| row indices

There are eight 2 by 2 sub-arrays in the example. (Many of them overlap with each other.) Here is one:

	0	1	2	3	4
0	1	9	9	9	5
1	9	1	0	9	5
2	5	9	2	9	7

The sum of the elements in that 2 by 2 sub-array is (1 + 9 + 1 + 9) = 20.
The next sub-array one column to the right is shown next:

	0	1	2	3	4
0	1	9	9	9	5
1	9	1	0	9	5
2	5	9	2	9	7

The sum of the elements in that 2 by 2 sub-array is (9 + 9 + 1 + 0) = 19.
The following shows the bottom most and right most 2 by 2 sub-array in this example:

	0	1	2	3	4
0	1	9	9	9	5
1	9	1	0	9	5
2	5	9	2	9	7

The sum of the elements in that 2 by 2 sub-array is (9 + 5 + 9 + 7) = 30. That is the max sum of all the sub-arrays in the example so in this case the method would return an array equal to `{1, 3}`.

```
// Assume that data is a rectangular matrix and that there are at least 2 rows
// and 2 columns in data. (data.length >= 2 and data[0].length >= 2)
// Return an array of length two as described in the problem statement.
public static int[] locationOfMax2By2SubArray(int[][] data){
```

```
// Assume that data is a rectangular matrix and that there are at least 2 rows
// and 2 columns in data. (data.length >= 2 and data[0].length >= 2)
// Return an array of length two as described in the problem statement.
public static int[] locationOfMax2By2SubArray(int[][] data){
```

7. Arrays. (10 points) Complete a static method that reverses the specified portion of an array of `ints`. **Important: You may not use any other classes or methods from the Java standard library when completing this method.**

Note, you are making changes to the elements of the array passed as a parameter. You are not returning a new array.

Here is the method header for the method you are completing:

```
// Assume the following conditions will be true when the method is called:
//    0 <= start < data.length
//    start <= stop <= data.length
// Reverse the elements in the array from position start inclusive
// to position stop exclusive.
// If start equals stop the array is unchanged.
public static void reversePortion(int[] data, int start, int stop){
```

Here are some example method calls and the resulting array:

```
reversePortion( {10, 20, 30, 40, 50}, 0, 1} -> {10, 20, 30, 40, 50}
// in the above example reversing a portion with one element has
// no effect.

reversePortion( {10, 20, 30, 40, 50}, 0, 0} -> {10, 20, 30, 40, 50}

reversePortion( {10, 20, 30, 40, 50}, 0, 5} -> {50, 40, 30, 20, 10}

reversePortion( {10, 20, 30, 40, 50}, 0, 4} -> {40, 30, 20, 10, 50}

reversePortion( {10, 20, 30, 40, 50}, 1, 4} -> {10, 40, 30, 20, 50}

reversePortion( {10, 20, 30, 40, 50}, 2, 4} -> {10, 20, 40, 30, 50}

reversePortion( {50, 100}, 1, 1} -> {50, 100}

reversePortion( {50, 100}, 0, 1} -> {50, 100}

reversePortion( {50, 100}, 1, 2} -> {50, 100}

reversePortion( {50, 100}, 0, 2} -> {100, 50}
```

Complete the `reversePortion` method. Place you answer on the next page.

ANSWER ON NEXT PAGE

ANSWER ON NEXT PAGE

ANSWER ON NEXT PAGE

ANSWER ON NEXT PAGE

ANSWER ON NEXT PAGE

```
// Assume the following conditions will be true when the method is called:
//    0 <= start < data.length
//    start <= stop <= data.length
// Reverse the elements in the array from position start inclusive
// to position stop exclusive.
// If start equals stop the array is unchanged.
public static void reversePortion(int[] data, int start, int stop){
```

8. File Processing (15 points) Write a `static` method that has one parameter, a `Scanner` object that is already connected to a file.

- All elements of the file will be white space or the `Strings` `"one"`, `"two"`, `"three"`, or `"four"`.
- The strings represent the rolls of a four sided die.
- Each line in the file consists of two of the four possible strings and represents one round of a game.
- A player wins a round if the second roll is greater than the first. Here are some example rounds

```
one one -> lose
one two -> win
one four -> win
four three -> lose
three four -> win
```

- A player wins the overall game if they win **more than 50%** of the rounds. If the above examples were in a file there would be 5 rounds. To win the overall game a player would have to win 3 or more rounds. In the above example the player won the overall game.

Each line of the file will contain two of the possible results of the die. There will be at least one space between the results on a given line and there may be more white space before, in between, and after each result:

```
 one       two
two four
four four
     one        three
one   one
four       three
```

Write a method that processes the file the scanner is connected to. Print out the following information about the game the file represents:

- The number of rounds in the game.
- The number of rounds the player won.
- The percentage of rounds the player won. You **do not** have to round the percentages to any certain number of decimal places.
- If the player won or lost the game.

For the first example above the output would be as follows:

```
5 rounds total
3 rounds won
60.0% won
player won
```

For the second example above the output would be as follows:

```
6 rounds total
3 rounds won
50.0% won
player did not win
```

Complete the method on the next page:

```
// scan is connected to a file as described in the problem
// There will be at least 1 round in the game and thus at least one line
// in the file.
// Process the file and print out the required data.
public static void processGame(Scanner scan){
```