Answer all of the following questions. READ EACH QUESTION CAREFULLY. Answer each question in the space provided on these pages. Budget your time so you spend enough on the longer programming questions at the end. There are a total of **60** points.

Keep your answers short and to the point. Good luck.

Question 1. (6 points) Here is a simple class definition.

```
class Question1 {
    public static final int XVII = 17;
    private String who;

    public void mangle(String what) {
        String where = “ here”;
        who = what + where;
    }

    public static void main(String [ ] args) {
        Confuse me = new Confuse(“instructor”);
        Confuse you = new Confuse(“student”);
    }
}
```

There are several identifiers declared in this class. Your job is to categorize them. After each of the following, please write down all of the identifiers that match the corresponding description.

(a) Instance variables – one copy is associated with each object (instance) of this class.

(b) Class variables – a single copy associated with the class itself.

(c) Local variables – variables that are local to a method.

(d) Instance methods – methods that are called by sending a message to an object (instance).

(e) Class methods – methods that are associated with the class itself.

Question 2. (8 points) Suppose we have a class that contains a method named printStuff, defined as follows.

```
class Test {
    …
    public void printStuff( ) {
        int i = 3;
        while (i > 0) {
            for (int j = 1; j <= i; j++) {
                int val = i+j;
                System.out.print(val + "  ");
            }
            System.out.println(" ");
            i --;
        }
    }
    …
}
```

What output does method printStuff( ) print on the console window when it is called?

Hint: You may find it helpful to hand-simulate the execution of the program and trace the values of the variables as the program executes. Use the blank space at the bottom of this page or on the opposite page to keep track of the values of variables if you wish.

Output:

Question 3. (3 points) The Othello game, like many typical applications, was an *event-driven* program with a graphical user interface. Give three specific examples of the kinds of events that a Java event-driven program might handle.

Question 4. (4 points) Java classes and interfaces have definitions that look quite similar. Both consist of a keyword (class or interface) followed by the class or interface name, followed by a list of the members of the class (methods and instance variables). What is the key difference between an interface and a class?

Question 5. (4 points) When we compared sequential and binary search, we observed that binary search was much faster than sequential search, particularly when searching large lists.

(a) What is the key difference between binary and sequential search that makes binary search so much faster?

(b) If binary search is so much faster, why don't we use it all the time? Why would we ever need to do a sequential search instead?

Question 6. (9 points) One of the examples given in lecture was class StringList, which implemented an ArrayList-like list of Strings. In this question, we'll use the version of the class that kept array of strings stored in sorted order (although that may not be particularly important in this particular case.

Complete the definition of method remove, which should remove the String at the specified position in the list. Your solution should not rely on any other methods in class StringList. You may, if you wish, define additional methods to use in your solution.

```
public class StringList {

    // instance variables
    private String[] strings;      // Items in this StringList are stored
    private int      size;         //   in strings[0..size-1].  Strings are
                                   //   sorted in ascending order, i.e.,
                                   //   strings[0] <= strings[1] <= ... strings[size-1]

    /**
     * Remove the string stored at position pos in this StringList.
     * @return String that was at position pos, provided pos selects an element
     *              in the list; if pos is out of bounds, return null.
     */
    public String remove(int pos) {




    }

}
```

Question 7. (9 points) For this question, write a method colSum that has one argument that is a 2-D array whose elements are doubles (floating-point numbers). Method colSum should return a 1-D array whose elements contain the sum of the values in the columns of the original array (i.e., the first element of the result should be the sum of the first column, etc.). For example, if the original array contained the following numbers

1.0	1.2	3.0	-1.5
0.5	2.0	1.0	5.0
2.5	-1.0	1.0	0.0

then the result returned from colSum should be a new array containing the numbers 4.0, 2.2, 5.0, and 3.5.

You may assume that the original array is rectangular, i.e., each row contains the same number of columns. You can also assume that the array has at least one row, and each row has at least one column.

```
// Return a 1-D array containing the sums of each of the columns in nums
double[ ] colSum(double[ ][ ] nums) {

    // allocate new array to hold result (you should fill in the correct size)

    double[ ] result = new double [ ________________________ ];

    // store answers in result













    // return final answer
    return result;
}
```

Question 8. (17 points) We'd like to add a bit of automated strategy to our Othello game. A new method nPointsAt(int row, int col) should evaluate a potential move and calculate the increase in the current piece's score if a move is made at that particular row and column on the board. If the current piece doesn't have a legal move at that location, then the result should be 0. Otherwise it should be 1 plus the number of pieces of the opposing color that would be flipped by making a move there.

For example, suppose the middle of the board contains the following squares:

```
.  B  .  B  .  .
W  W  W  B  ?  .
.  W  W  B  .  .
.  .  W  .  .  .
```

(B indicates black, W indicates white, and . indicates empty.) If function nPointsAt is asked to evaluate the square indicated by ? and it is white's turn, the result should be 3, since placing a white tile there will add one white tile, and flip the two black tiles immediately to the left and diagonally to the lower left for a total increase of three white pieces.

Implement your solution using two methods – one that returns the number of pieces flipped in a particular direction, assuming that there is a legal move in that direction, and a method nPointsAt that calculates the total possible score at a given location using the first method. Additional information about the class is provided on the last page of the exam. You may use any methods you find there in your solution.

```
// Direction changes - in each of 8 possible directions (0 to 7), deltaR[d] is the change in row
// # to move in that direction, and deltaC[d] is the change in column # in that direction.
private static final int[] deltaR = {-1, -1,  0, +1, +1, +1,  0, -1};
private static final int[] deltaC = { 0, +1, +1, +1,  0, -1, -1, -1};

// Return the number of pieces flipped in direction d if a move is made by the current player
// at the given row and column.  Assumption: the current player has a legal move in this
// direction at this row and column.
private int nFlippedInDirection(int d, int row, int col) {









}
```

```
// Return the total increase in points if the current player makes a move at the given row
// and column.  Return 0 if the current player does not have a legal move at that location.
private int nPointsAt(int row, int col) {









}
```

Reference information about class Board. Feel free to tear this page off if it is more convenient to do so.

```
public class Board {
   // public constants
   public static final int NROWSCOLS = 8;  // # rows/columns on the board
   public static final int EMPTY = 0;           // empty board square
   public static final int BLACK = 1;           // board square containing black piece
   public static final int WHITE = 2;           // board square containing white piece

   // private instance variables
   private int[ ][ ] board;            // game board
   private int numBlack;               // # of black pieces
   private int numWhite;               // # of white pieces
   private int currentColor;           // color currently moving

   //  Methods (Methods definitely not needed for this problem omitted)

   // Return true if row, col are on the board, otherwise false
   private boolean inBounds(int row, int col) { … }

   /** Return color of the pieces currently moving */
   public int getCurrentColor( ) { …}

   // Return color of other pieces not moving now
   private int getOther( ) { … }

   // Change current move to other color
   private void toggleMove( ) { … }

   // Update specified square to given state, & adjust piece totals
   private void setSquare(int row, int col, int state) { … }

   // Direction changes - in each of 8 possible directions (0 to 7),
   // deltaR[d] is the change in row number to move in that direction,
   // and deltaC[d] is the change in column number for that direction.
   private static final int[ ] deltaR = {-1, -1,  0, +1, +1, +1,  0, -1};
   private static final int[ ] deltaC = { 0, +1, +1, +1,  0, -1, -1, -1};

   // Return true if it current player (Board.WHITE or Board.BLACK)
   //  can make a move at the given row and column
   private boolean canMove(int row, int col) { … }

   // True if the current player can move starting at the given row and
   // column in direction d.  Precondition: startRow and startCol are  in bounds.
   private boolean canMoveInDirection(int d, int startRow, int startCol) {… }

}
```