Name: ____________________________________

USC loginid (e.g., ttrojan):__________________________

CS 455 Midterm Exam 1
Fall 2012 [Bono]

Oct. 3, 2012

There are 4 problems on the exam, with 50 points total available. There are 6 pages to the exam, including this one; make sure you have all of them. There is also a one-page code handout that accompanies the exam. If you need additional space to write any answers, you may use the backs of exam pages (just direct us to look there).

Note: if you give multiple answers for a problem, we will only grade the first one. Avoid this issue by labeling and circling your final answers and crossing out any other answers you changed your mind about (though it's fine if you show your work).

Put your name and USC loginid at the top of the exam. Please read over the whole test before beginning. Good luck!

	value	**score**
Problem 1	11 pts.	
Problem 2a	12 pts.	
Problem 2b	2 pts.	
Problem 3	10 pts.	
Problem 4	15 pts.	
TOTAL	50 pts.	

Problem 1 [11 pts. total]

Consider the following program:

```
public class Prob1 {
  public static void foo(int i, int[] vals) {
    i++;
    vals[i]++;
    System.out.println(i + " "
                          + Arrays.toString(vals));
  }
  public static void main(String[] args) {
    int j = 1;
    int[] arr = new int[3];
    foo(j, arr);
    System.out.println(j + " "
                          + Arrays.toString(arr));
    j--;
    foo(j, arr);
    System.out.println(j + " "
                          + Arrays.toString(arr));
  }
}
```

Output of Program

Part A. In the space below, draw a box-and-pointer diagram (a.k.a., memory diagram) showing all variables, objects, arrays, and their state as they have changed during the code sequence. You may identify the data in the first call to `foo` as follows: i_1, $vals_1$, and for the second call: i_2, $vals_2$. When a value gets updated, cross it out and show the new value next to the old one rather than drawing it again: e.g., you will only have one box labeled `arr` and one box labeled `j`.

Part B. In the box to the right of the code above, show the output of the program.

Problem 2 [14 pts. total]

Part A [12]. Complete the implementation of the class `Duration`, which represents some amount of time in hours and minutes, such as 3 hours and 25 minutes (not a time of day). Such a class could be a useful building-block in an applications such as a timer, stopwatch, or scheduling programs (e.g., to schedule your DVR or flights for an airline).

The class allows us to create a duration expressed using a number of hours and minutes or just total number of minutes, it also allows us to increase a duration by some amount of time, to add two durations together, and to convert a duration to a String form (for printing or appending to larger messages). The detailed specification for each method appears in its method comments below.

Furthermore, **you are restricted to use the internal representation for `Duration` shown in the starter code for the class: internally we will store a duration as its total number of minutes**. This representation actually would be one an implementer would likely use in a more fleshed out version of such a class because makes some operations easier to implement.

The class interface and space for your answer starts here and continues onto the next page.

```
/**
   Duration represents some amount of time in hours and minutes.
*/
public class Duration {

   // Problem restriction:
   //     do not change the private instance variable definitions

   private int totMinutes;  // representation invariant: totMinutes >= 0

   /**
      Creates duration of the given hours and minutes
      PRE: hours >=0 and minutes >= 0
   */
   public Duration (int hours, int minutes) {







   }

   /**
     Creates a duration given as a total number of minutes.
     PRE: minutes >= 0
   */
   public Duration (int totalMinutes) {







   }
```

Problem 2 (cont.)

```
   /**
      Increase the current duration by the amount of time
         given by hours and minutes.
      PRE: hours >=0 and minutes >= 0
   */
   public void increase(int hours, int minutes) {








   }

   /**
      Returns the sum of "this" duration and duration "b"
      This method is not a mutator (does not modify "this").
      Note the return type is a Duration object.
      HINT: Java rules allow this method to access private data of other
            Duration objects besides "this": for example, b's private data.
   */
   public Duration add(Duration b) {








   }

   /**
      Returns a String form of the duration using the format shown by
        example here: "8hrs 32min"
   */
   public String toString() {








   }
}
```

Part B [2]. Are `Duration` objects immutable? If so why, if not, why not?

Problem 3 [10 pts.]

Implement a new private method for our `Names` class called `isValidNames`, which returns true if and only if the internals of this `Names` object is in a valid state, that is, it satisfies its representation invariant. Such a method is useful to check that the code in your other `Names` methods works correctly.

We're using the same `Names` class and representation we discussed in lecture, and that is reviewed below:

```
/**
   Stores a list of unique names in alphabetical order.  Allows
   look-up, insert, and removal of elements in the list.
*/
public class Names {
   private String[] namesArr;     // the capacity is namesArr.length
   private int numNames;          // the number of names in the array
   /*
     Representation invariant:
     -- 0 <= numNames <= namesArr.length
     -- when numNames > 0 the names are stored in namesArr[0] through
                                                 namesArr[numNames-1]
     -- the names stored are unique
     -- the names are in alphabetical order
   */

   private boolean isValidNames()
```

Problem 4 [15 pts. total]

Implement the static method `mode`, which returns the value that occurs most often in an array of integers. You may assume the array has at least one element, and all of the values in the array are in the range 0 - 100. If there is more than one value that occurs most often, return the smallest such value (See Ex2 below). Here are some examples:

	`arr:`	return value of **`mode(arr):`**	
Ex1:	`99 86 99 99 95 86`	`99`	
Ex2:	`23 15 15 74 23`	`15`	*(two 15's and two 23's; chooses the smallest value)*
Ex3:	`76`	`76`	

For full credit, your answer must only traverse the data in `arr` once. Hint: you are allowed to use additional memory.

```
/**
   Returns the mode of the values in the array.
   If there is more than one mode, returns the smallest one.
   PRE: arr.length > 0
        and all the values in arr are in the range 0 - 100
*/
public static int mode(int[] arr) {
```