Name: ____________________________________

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

Midterm Exam 2
CS 455, Fall 2013

Wednesday, November 6, 2013

There are 9 problems on the exam, with 58 points total available. There are 8 pages to the exam, including this one; make sure you have all of them. There is also a separate double-sided one-page code handout. 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 loginid at the top of the exam. Please read over the whole test before beginning. Good luck!

	value	**score**
Problem 1-3	10 pts.	
Problem 4	7 pts.	
Problem 5	6 pts.	
Problem 6	6 pts.	
Problem 7	6 pts.	
Problem 8	8 pts.	
Problem 9	15 pts.	
TOTAL	58 pts.	

Problem 1 [6 pts. total]

Part A. Name an abstract data type (ADT) we have discussed this semester that would be useful for implementing the BACK button on a Web browser.

Part B. In a few sentences describe how the ADT you named in part A would be used to help solve this problem (including what data is stored in the ADT and which operations would be used for doing what).

Problem 2 [2 pts.]

What is true about the first k elements in the array before pass k of insertion sort? (circle the answer that best describes what is known)

a. they are the k smallest values
b. they are the k smallest values in sorted order
c. they are k values in sorted order
d. none of the above

Problem 3 [2 pts.]

Suppose we have a hash table with String keys such that *key1* and *key2* hash to the same location. You may assume that we use the good hash function for Strings we discussed in lecture. Circle all that are guaranteed to be true about *key1* and *key2*:

a. *key1* and *key2* are equal
b. one of them is a substring of the other
c. they both start with the same letter of the alphabet
d. their map entries have the same value (i.e., first entry: [*key1*, *value1*] ; second entry: [*key2*, *value1*])
e. none of the above

Problem 4 [7 pts. total]

Here's the interface for the `Names` class we discussed in lecture earlier in the semester:

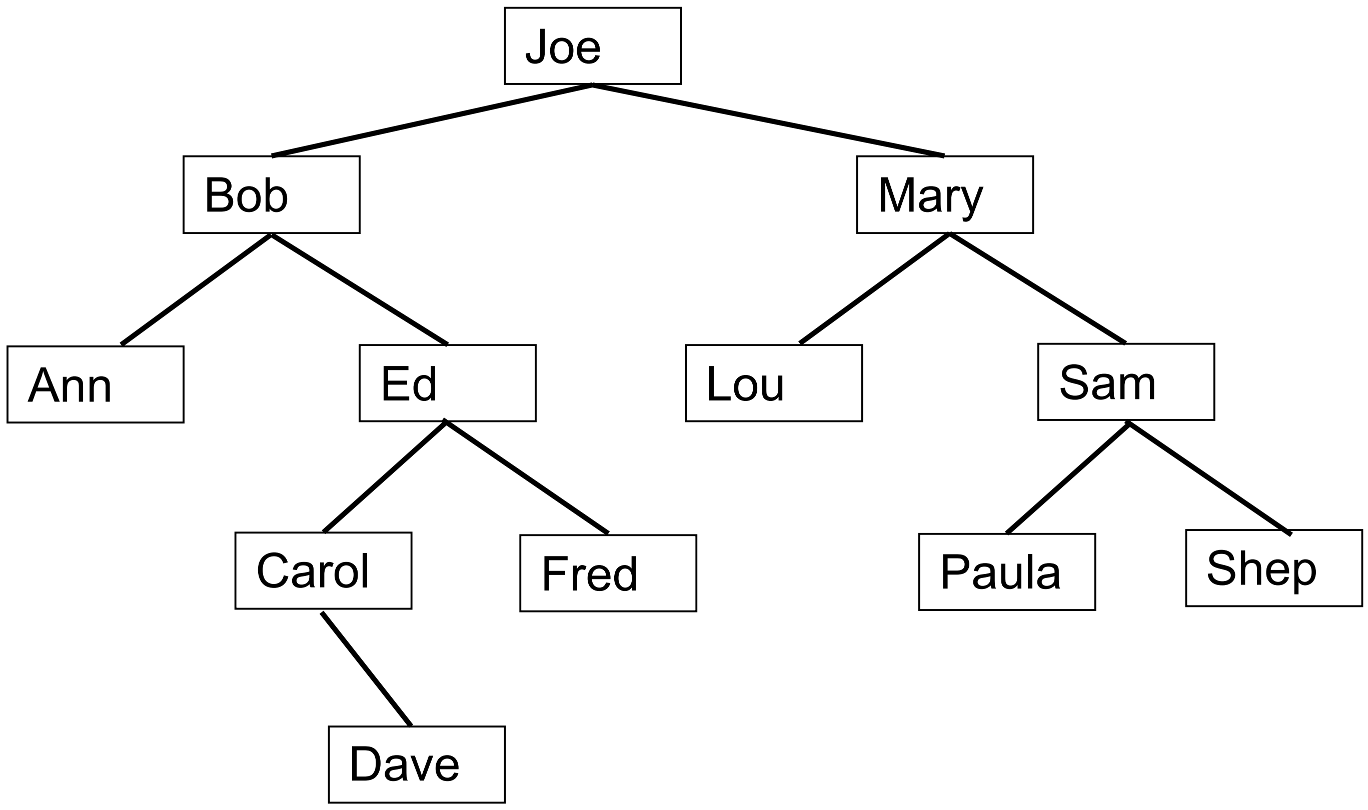
```
/**
   Stores a list of unique names in alphabetical order.  Allows
   look-up, insert, and removal of elements in the list.
*/
public class Names {

    /**
       Creates an empty names object
     */
    public Names() { . . . }

    /**
       Returns the number of names in the list
     */
    public int numNames() { . . . }


    /**
       Returns true iff target is present in names
     */
    public boolean lookup(String target) { . . . }


    /**
       Removes target from names, and returns true.
       If target wasn't present in names, returns false and no change
       made to names.
     */
    public boolean remove(String target) { . . . }

    /**
       Inserts newName into alphabetical names list.
       Returns false and no change is made to names if name is already
       present.
     */
    public boolean insert(String newName) { . . . }


    /**
       Prints the names in alpha order, one per line
     */
    public void printNames(){ . . . }

    . . .
}
```

Part A (2 pts). What specific Java class could we use to represent a Names object (i.e., type for the private data in Names) that we hadn't learned about when we first did this example that is a large improvement over our original representation?

Part B (5 pts). For each of the names operations listed above, next to its description write down the big-O time for doing that operation using the representation you mentioned in Part A. (All but one of them will have a better time complexity than our original Names representation; the one left will have the same time complexity as before, because we can't do any better than that.)

Problem 5 [6 pts. total]

Consider the following binary search tree, whose root is shown nearest the top of the page (i.e., it's not a sideways tree):



Part A. For this specific tree, what's the maximum number of keys we would have to compare a target key with to lookup that target? (i.e., worst case, although the answer will be a number, not a big-O expression)

Part B. For this specific tree, what's the minimum number of keys we could compare a target key with to lookup that target? (i.e., best case)

Part C. For this tree give an example of a target key we could look up *that's not present in the tree* that would entail comparing with the maximum number keys (i.e., your answer to part A) to search for it.

Problem 6 [6 points]

Consider the following incorrect Java program that does not compile, with an error related to exceptions. Note: it uses another class called `DataObject` – you do not need to understand anything about how `DataObject` works to solve this problem. (Note: see code handout for more information about the Scanner class.)

Fix the program and enhance it as follows: if the user gives a file name for a non-existent file, for example, `flkjm`, print the following message:

```
ERROR: File does not exist: flkjm
Exiting program.
```

and exit the program immediately. If the file does exist, the program will operate normally. Space is given below to make your modifications / additions right in the code below.

```
public class MyProg {

  public static DataObject readFile(String fileName) {

     DataObject data = new DataObject(); // create an empty data obejct

     File inFile = new File(filename);

     Scanner fileScanner = new Scanner(inFile);

     while (fileScanner.hasNext()) {

        // . . .  [ puts stuff in "data" ]

     }

     return data;

   }

   public static void main(String[] args) {

     Scanner in = new Scanner(System.in);

     System.out.print ("Please enter a file name: ");

     String fileName = in.next ();

     DataObject data = readfile(filename);

     data.process();

     data.printResults(System.out);

   }

}
```

Problem 7 [6 pts total]

Suppose we are using a `Map` to store information about Restaurants and their locations. It maps `Points` to `Restaurant` objects. A reminder of the `Point` class is shown on the code handout. A `Restaurant` object has a name and other attributes, and also has a mutator called `setName` to change the name of the restaurant.

You may assume the following code happens sometime before the code snippets given below:

```
Map<Point, Restaurant> restMap;
[. . . code to initialize and put values in restMap]
```

For each of the following code sequences, give an answer a, b, or c, where these letters stand for:

a. **the code has no effect on the Map (leaves it unchanged)**
b. **the code invalidates the Map**
c. **the code works as intended (intended effect for each described in the comments below)**

Part A. answer: _____

```
Restaurant rest = restMap.get(new Point(3, 7));
if (rest != null) {
  rest.setName("KFC");  // change the name of the restaurant at (3,7) to KFC
}
```

Part B. answer: _____

```
// [enhanced for loop below]
for (Map.Entry<Point, Restaurant> entry : restMap.entrySet()) {
   int xLoc = entry.getKey().getX();
   if (xLoc == 574) {     // adjust the location of some restaurants:
      entry.getKey().translate(3, 0);  // move to the right by 3
   }
}
```

Part C. answer: _____

```
Restaurant rest = restMap.get(new Point(10, 12));
if (rest != null) {
  rest = new Restaurant("McDonald's", . .);
       // change the restaurant at (10,12) to McDonald's
}
```

Problem 8 [8 pts]

Implement the **Java** method `sortByArea`, which sorts an `ArrayList` of rectangles so they are in increasing order by area. I.e., smaller rectangles will appear on the list before larger ones. *You must use the Java sort utility* (more info about that on the code handout), and include any additional code necessary to make it work (outside of the `sortByArea` method).

The code handout also has more about the `Rectangle` class from the Java library.

```
public static void sortByArea(ArrayList<Rectangle> boxes) {
```

Problem 9 [15 points]

Write the static `boolean` method `myEquals`, which takes two `LinkedList` objects and returns true iff the two lists have exactly the same values in the same order. Your method may not call the `LinkedList` `equals` method.

Examples:

list1	***list2***	*return value of* ***`equals(list1, list2)`***:
`(8 6 4 2)`	`(8 6 4 2)`	`true`
`(1 2)`	`(3 4 5)`	`false`
`()`	`()`	`true`
`(8 6 4 2)`	`(8 6 4 2 7)`	`false`
`(8 6 4 2)`	`(8 6 2)`	`false`
`(8 6 4 2)`	`(8 6 4 7)`	`false`

```
public static boolean myEquals(LinkedList<Integer> list1,
                               LinkedList<Integer> list2) {
```