Name: _____

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

Midterm Exam 2
CS 455, Spring 2013

April 4, 2013

There are 6 problems on the exam, with 60 points total available. There are 7 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	6 pts.	
Problem 2	6 pts.	
Problem 3	4 pts.	
Problem 4	14 pts.	
Problem 5	15 pts.	
Problem 6	15 pts.	
TOTAL	60 pts.	

Problem 1 [6 pts. total]

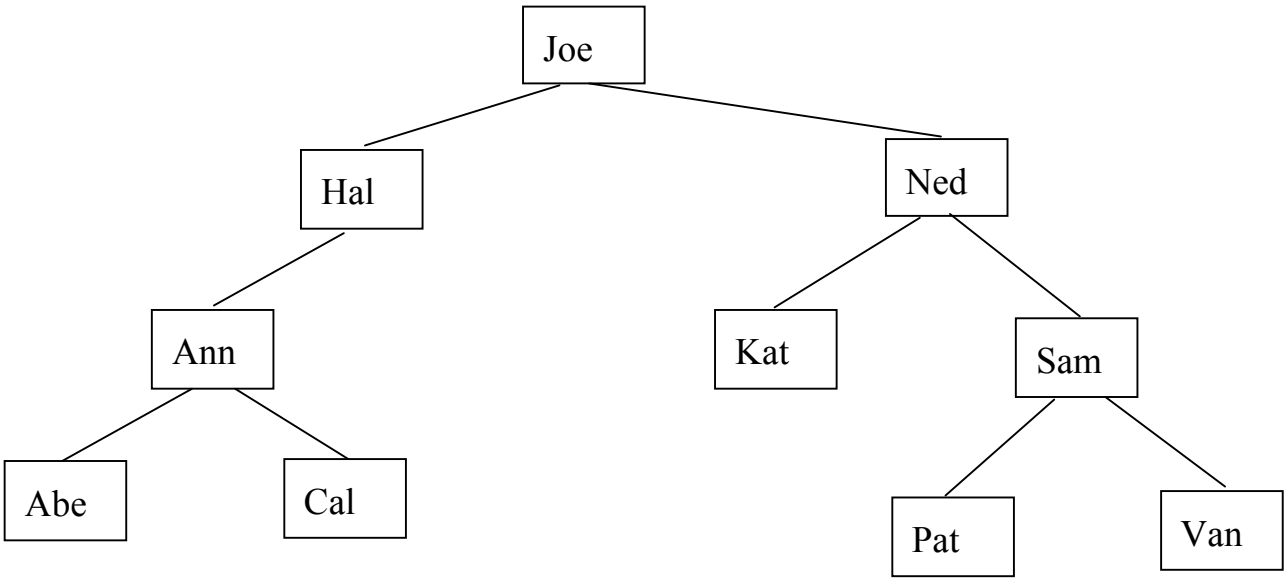
Consider an implementation for the Java `Set` interface we'll call `OrderedArraySet`, a class which uses an ordered array representation to store the elements of the set.

Part A (2). *(fill in the blanks)* With this representation we can do the `contains(`*`key`*`)` operation, in O(_____) time worst-case using the _____ algorithm.

Part B (4). Java does not provide an OrderedArraySet implementation for the `Set` interface. Why? Note: Your answer should include a discussion of specific Set operations (most of the Set interface is shown on the code handout). However, you shouldn't need all the space below – it's just the way the paging worked out.

Problem 2 [6 pts. total]

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



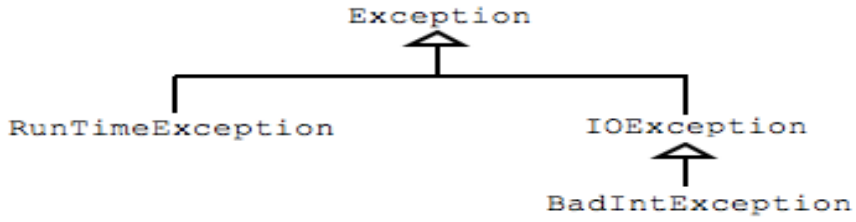
For each of the following lookups from the tree shown above, give the sequence of keys that the target key would have to be compared with to do the lookup.

Part A. lookup *Dan*

Part B. lookup *Lou*

Part C. lookup *Pat*

Problem 3 [4 points]



Consider the following Java program that uses exceptions (part of the exception class hierarchy shown above):

```
public class MyProg {
   public static void main(String[] args) {
      try {
         Scanner in = new Scanner(System.in);
         System.out.println("Please enter an integer: ");
         String line = in.nextLine();
         Scanner lineScanner = new Scanner(line);
         int val = getInt(lineScanner);
         if (val > 100) {
             throw new BadIntException("Value out of range");
         }
         System.out.println("Value entered was: " + val);
      }
      catch (BadIntException exc) {
         System.out.println("Bad Int Exception: " + exc.getMessage());
      }
   }
   public static int getInt(Scanner lineScanner) {
      int val;
      try {
         if (!lineScanner.hasNextInt()) {
            throw new BadIntException("Problem #1");
         }
         val = lineScanner.nextInt();
         if (lineScanner.hasNext()) {
            throw new BadIntException("Problem #2");
         }
      }
      catch (IOException exc) {
         System.out.println("IO ERROR: " + exc.getMessage());
         val = -1;
      }
      return val;
   }
}
```

Does `MyProg` compile? _____

If you answered "no": write in the minimal changes to it above so it would compile.

If you answered "yes": what exact output does the current code give if the user enters the following input at the prompt: **`a 200`**

Problem 4 [14 points]

Part A (10). Assume we make a call to the following method, `prob5Func`, with a stack with the following value (in this diagram the top of the stack is at the ***right*** side, and is labeled as such):

value of `s`: `1 6 7 0 2 4 5` ← top

Show the contents of each of the stacks listed at each of the points in the code, labeled A, B, and C. ***For each one clearly label where the top of your stack is in your diagram.***

```
public static Stack<Integer> prob5Func(Stack<Integer> s) {
   Stack<Integer> first = new Stack<Integer>();
   Stack<Integer> second = new Stack<Integer>();

   boolean found0 = false;
   while (!s.empty() && !found0) {
       int curr = s.peek();
       first.push(s.pop());
       if (curr == 0) {
          found0 = true;
       }
   }

   // A

   while (!s.empty()) {
      second.push(s.pop());
   }

   // B

   while (!first.isEmpty()) {
      second.push(first.pop());
   }

   // C

   return second;
}
```

at point **A**:

`s`:

`first`:

at point **B**:

`s`:

`second`:

at point **C**:

`first`:

`second`:

Part B (4). In general what does the function do with its argument, `s`? Assume it is called as follows:

```
s = prob5func(s);
```

Problem 5 [15 pts]

We can think of a Map as describing a function that maps from keys to values. Write the **Java** boolean function `isOneToOne` which tells if its `Map` argument is a one-to-one function, that is, whether no two *(key, value)* entries have the same *value* part. Here are some examples (maps shown as they would be by a call to `toString`):

`map`	**`isOneToOne(map)`**	
`{joe=3, sally=2, sam=3}`	*`false`*	*(both `joe` and `sam` have the value, `3`)*
`{joe=3, sally=2, sam=7}`	*`true`*	*(no two keys have the same value)*
`{}`	*`true`*	*(you may assume an empty map is one-to-one)*

To receive full credit your function must run in at most O(*n*) time, where *n* is the number of entries in the map (Hint: take advantage of the Java library).

```
public static boolean isOneToOne(Map<String, Integer> map) {
```

Problem 6 [15 points]

Write the method `interleave`, which takes two LinkedList objects and returns a new LinkedList where the values from the old lists are interleaved into the new list: that is, the new list starts with the first element from list1, then first element from list2, second element from list1, second element from list2, etc. If one of the lists is longer than the other, it puts all the leftover elements on the end of the resulting list. The original lists are unchanged by this method.

Examples:

list1	***list2***	*return value of* ***interleave(list1, list2)****:*
`(1 3 5 7 9 11 13)`	`(8 6 4 2)`	`(1 8 3 6 5 4 7 2 9 11 13)`
`(1 2)`	`(3 4 5)`	`(1 3 2 4 5)`
`()`	`(4 5 1)`	`(4 5 1)`

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