Exam 2 for Com S 228 Fall 2013 on October 31, 2013

Name:

University ID:

Lecture section (please circle one):

```
A. MWF 09:00am-09:50am    CARVER   0205    XQ
B. MWF 03:10pm-04:00pm    LAGOMAR  W0142   Pak
```

Recitation section (please circle one):

```
1. M 11:00am-11:50am    GILMAN 0611     Kyle, Brady
2. M 10:00am-10:50am    GILMAN 0312     Bryan, Caleb
3. M 4:10pm-5:00pm      GILMAN 0312     Curtis
4. T 11:00am-11:50am    GILMAN 1312     Chris, Brady
5. T 10:00am-10:50am    GILMAN 1810     Brandon
6. T 4:10pm-5:00pm      GILMAN 0312     Cameron, Jesse
```

There are 6 pages and 4 questions. Please start with easy questions. We will give partial credit for a partially correct solution. You are not allowed to use any books, notes, calculators, or electronic devices. The exam is 60 minutes long.

Question	Points	Your Score
1	25	
2	25	
3	25	
4	25	
Total	100	

1 (25 pts). Below is a version of non-recursive QuickSort different from those presented in lectures. It is slightly different from that in the practice.

```
public class QuickSortProblem
{
public static void quickSortNonRec(int[] arr)
{ if ( arr == null || arr.length == 0 )
    throw new IllegalArgumentException("Null or Zero");
  Stack<Integer> range = new Stack<Integer>();
  range.push(0);
  range.push(arr.length - 1);
  while ( ! range.isEmpty() )
  { int last = range.pop();
    int first = range.pop();
    if ( first >= last ) continue;
    int mid = partition(arr, first, last);
    range.push(first);
    range.push(mid-1);
    range.push(mid);
    range.push(last);
    // point E
  }
}
private static int partition(int[] arr, int first, int last)
{ int  pivot = arr[last];
  int  left = first;
  int  right = last;
  while ( true )
  { while ( arr[left] < pivot )
       left++;
    while ( arr[right] > pivot )
       right--;
    if ( left < right )
    {  int t = arr[left];
       arr[left] = arr[right];
       arr[right] = t;
       left++; right--;
    }
    else
       break;
  }
  return left;
}
}
```

Execute the above code on the following array of length 10:

65 61 25 5 76 91 5 52 98 40

(a) Show the contents of the array at the point `E` when the point is reached for the first time. Mark the partition of the array into two parts with a vertical line.

Array at the point `E`:

(b) Show the contents of the stack at the point `E` when the point is reached for the first time.

Stack with top on the left:

(c) Show the contents of the stack at the point `E` when the point is reached for the second time.

Stack with top on the left:

(d) Show the contents of the stack at the point `E` when the point is reached for the third time.

Stack with top on the left:

2 (25 pts). Write a public generic class named `Triple` that stores three objects of a generic type `E`. This `Triple` class has two public instance methods, `firstInOrder()` and `secondInOrder()`, to check if the three objects in the class are in decreasing order. The first method, `firstInOrder()`, uses the `compareTo()` method of the Java `Comparable` interface and the second method, `secondInOrder()`, uses the `compare()` method of the Java `Comparator` interface. See specific descriptions below.

(a) Write the heading of the `Triple` class with the type parameter `E` and declare the three private reference variables of type `E` named `one`, `two`, and `three`, respectively. The type `E` either inherits the `compareTo()` method of the Java `Comparable` interface from its supertype or implements the `compareTo()` method itself.

(b) Write a constructor that takes three parameters of the type `E` and saves them in the three private reference variables of `Triple`. If any parameter is a null pointer, throw a `NullPointerException()`.

(c) Write a public instance method named `firstInOrder()` that takes no parameter and returns a `boolean` value. The method uses `compareTo()` to check if `one`, `two`, and `three` are in decreasing order. If `one` is greater than `two` and `two` is greater than `three`, then the method returns `true`. Otherwise, it returns `false`.

(d) Write a public instance method named `secondInOrder()` that takes two objects `comp1` and `comp2` of `Comparator<Q>` with `Q` being a supertype of `E` and returns a `boolean` value. The method uses the `compare()` of `comp1` to check if `one` is greater than `two` and uses the `compare()` of `comp2` to check if `two` is greater than `three`. If so, then the method returns `true`. Otherwise, it returns `false`.

3 (25 pts). Below is an example of settting up a list object and creating/using its listIterator object in a section of code. Follow the instruction at the start of the section.

```
import java.util.LinkedList;
import java.util.ListIterator;

public class ListState
{
  public static void main(String[] args)
  {
    LinkedList<String> list = new LinkedList<String>();
    list.add("A");
    list.add("B");
    list.add("C");
    list.add("D");
    list.add("E");

    // Show Output and List & iterator state after each iterator
    // statement. Use (N) in the output column if there is no output.
    // Mark the logical cursor with the symbol |.
    LinkedList<String> copy1 = new LinkedList<String>(list);
    int num = copy1.size();
    ListIterator<String> iter1 = copy1.listIterator( num );
                                  // Output  List/iterator state

    // Show state before the loop:        (N)
    for ( int j = num-1; j > num / 2; j-- )
         iter1.previous();
    // Show state after the loop:         (N)
    System.out.println(iter1.previous());
    System.out.println(iter1.next());
    iter1.remove();
    System.out.println(iter1.previous());
    iter1.add("X");
    System.out.println( iter1.next() );
    System.out.println( iter1.next() );
    iter1.set("Y");
  }
}
```

4 (25 pts). Your task is to write a public method for a singly-linked list class. Assume that the class has two member variables `ListNode head` and `int numItems`, where `numItems` is the number of nodes in the list and `head` always refers to the first node in the list if the list is not empty and is `null` otherwise. The inner class `ListNode` has two member variables `public E data` and `public ListNode link`.

Write the public method `removeFirstDuplicateLastTwo()` that removes the first node if the list has at least three nodes and duplicates the last two nodes in the list if the list has at least two nodes and throws `NoSuchElementException` if the list has at most one node. For example, if the list contains two elements: A, B, then the new method turns the list into a new list with four elements: A, B, A, B; if the list contains three elements: A, B, C, then the new method turns the list into a new list with four elements: B, C, B, C.

The duplication is done by appending two new nodes to the end of the list and setting up their `data` fields properly. The method returns no object. You must not use any previously defined method in the class to write the `removeFirstDuplicateLastTwo()` method. Note that you do not have to insert your code into every space.

```
public void removeFirstDuplicateLastTwo()
{
  if (                                               )
     throw new NoSuchElementException();

  if (                                               )
  {

  }

  ListNode cur;
  for ( cur =        ;                         ;        )
  {


  }


  ListNode nLast = new ListNode();
  ListNode nSecLast = new ListNode();
  nLast.link =
  nSecLast.link =
  nLast.data =
  nSecLast.data =




  return;
}
```