CS 145 Midterm Practice Problems/Solutions

Note that this is designed to give you a sample of the type of problems that may be on the exam, there will not be this many problems on the exam itself.

(1) **Array**

Consider the following method:

```
public static void mystery2(ArrayList list) {
    for (int i = list.size() – 1; i > 0; i--) {
        if (list.get(i) < list.get(i – 1)) {
            int element = list.get(i);
            list.remove(i);
            list.add(0, element);
        }
    }
    System.out.println(list);
}
```

Write the output produced by the method when passed each of the following ArrayLists:

[2, 6, 1, 8]

[1, 2, 6, 8]

[30, 20, 10, 60, 50, 40]

[10, 30, 40, 20, 60, 50]

[-4, 16, 9, 1, 64, 25, 36, 4, 49]

[-4, 1, 25, 4, 16, 9, 64, 36, 49]

(2) **List**

Write a method removeShorterStrings that takes an ArrayList of Strings as a parameter and that removes from each successive pair of values the shorter string in the pair. For example, suppose that an ArrayList called list contains the following values: {"four", "score", "and", "seven", "years", "ago"} In the first pair, "four" and "score", the shorter string is "four". In the second pair, "and" and "seven", the shorter string is "and". In the third pair, "years" and "ago", the shorter string is "ago". Therefore, the call: removeShorterStrings(list); should remove these shorter strings, leaving the list as follows: "score", "seven", "years". If there is a tie (both strings have the same length), your method should remove the first string in the pair. If there is an odd number of strings in the list, the final value should be kept in the list.

```
public static void removeShorterStrings(ArrayList<String> theList)
  {
      int lSize = theList.size() - 2;
      if (lSize % 2 == 1) lSize--;

      for (int x = lSize; x >= 0 ; x-=2)
      {
        if (theList.get(x).length() <= theList.get(x+1).length())
        {
          theList.remove(x);
        }
        else
        {
          theList.remove(x+1);
        }
      }
 }


// Note that this is just ONE way of doing it,  there are others.
```

(3) **Classes**

```
public class Bay extends Lake {
    public void method1() {
        System.out.println("Bay 1");
        super.method2();
    }

    public void method2() {
        System.out.println("Bay 2");
    }
}

public class Pond {
    public void method2() {
        System.out.println("Pond 2");
    }
}

public class Ocean extends Bay {
    public void method2() {
        System.out.println("Ocean 2");
    }
}

public class Lake extends Pond {
    public void method3() {
        System.out.println("Lake 3");
        method2();
    }
}
```

Suppose the following variables are defined:

```
Lake var1 = new Ocean();
Pond var2 = new Pond();
Pond var3 = new Lake();
Object var4 = new Bay();
Lake var5 = new Bay();
Bay var6 = new Ocean();
```

What would the output of the following method calls be?

```
var1.method2();
Ocean2

var2.method2();
Pond2

var3.method2();
Pond2

var4.method2();
COMPILER ERROR

var5.method2();
Bay 2

var6.method2();
Ocean 2


var1.method3();
Lake 3
Ocean 2

var2.method3();
COMPILER ERROR

var3.method3();
COMPILER ERROR

var4.method3();
COMPILER ERROR

var5.method3();
Lake 3
Bay 2

var6.method3();
Lake 3
Ocean 2
```

```
((Ocean) var5).method1();
RUNTIME ERROR

((Lake) var3).method3();
Lake 3
Pond 2

((Lake) var4).method1();
COMPILER ERROR

((Ocean) var1).method1();
Bay 1
Pond 2

((Bay) var4).method1();
Bay 1
Pond 2

((Lake) var2).method3();
RUNTIME ERROR


((Ocean) var5).method1();
RUNTIME ERROR

((Pond) var4).method2();
Bay 2

((Bay) var4).method1();
Bay 1
Pond 2

((Lake) var2).method3();
RUNTIME ERROR

((Ocean) var5).method1();
RUNTIME ERROR

((Pond) var4).method2();
Bay 2
```

(4) **Sets**

Write a method removeEvenLength that takes a Set of strings as a parameter and that removes all of the strings of even length from the set. For example, if your method is passed a set containing the following elements:

["foo", "buzz", "bar", "fork", "bort", "spoon", "!", "dude"]
Your method should modify the set to store the following elements (the order of the elements does not matter):

["foo", "bar", "spoon", "!"]

```
public static void removeEvenLength(Set<String> theSet)
{
   Set<String> temp = new TreeSet<String>();

   for (String x : theSet)
   {
     if (x.length() % 2 == 0)
     {
         temp.add(x);
     }
   }
   for (String x : temp)
   {
     theSet.remove(x);
   }

}

// Note that the code below DOES NOT WORK!!!!!!!!!!!!!!!!!!!!
public static void removeEvenLength(Set<String> theSet)
{
   for (String x : theSet)
   {
     if (x.length() % 2 == 0)
     {
         theSet.remove(x);
     }
   }
}
```

(5) **Collections Programming.**

Write a method countInAreaCode that accepts two parameters, a Map from names (strings) to phone numbers (strings) and an area code (as a string), and returns how many unique phone numbers in the map use that area code. For example, if a map m contains these pairs:

{Marty=206-685-2181, Rick=520-206-6126, Beekto=206-685-2181, Jenny=253-867-5309, Stuart=206-685-9138, DirecTV=800-494-4388, Bob=206-685-9138, Benson=206-616-1246, Hottline=900-520-2767}

The call of countInAreaCode(m, "206") should return 3, because there are 3 unique phone numbers that use the 206 area code: Marty/Beekto's number of "206-685-2181", Stuart/Bob's number of "206-685-9138", and Benson's number of "206-616-1246".

You may assume that the map passed is not null, that no key or value in it is null, that every phone number value string in the map will begin with a 3-digit numeric area code, and that the area code string passed will be a non-null numeric string exactly 3 characters in length. If the map is empty or contains no phone numbers with the given area code, your method should return 0.

You may create one collection of your choice as auxiliary storage to solve this problem. You can have as many simple variables as you like. You should not modify the contents of the map passed to your method.

```
public static int countInAreaCode(Map<String,String> theMap, String
theCode)
{
   Set<String> uniqueVersions = new TreeSet<String>();

   for (String x : theMap.keySet())
   {
      if(theMap.get(x).startsWith(theCode) )
      {
         uniqueVersions.add(theMap.get(x));
      }
   }

   return uniqueVersions.size();
}
```

(6) **Recursion**

Write a recursive method digitSum() that finds the sum of the digits of an integers. For example digitSum(1234) would return 10 because 1+2+3+4=10. You could do this with a for loop, but can you do it recursively.

```
public static int digitSum(int num)
{
  if (num < 0)  return digitSum(-1 * num);
  if (num < 10) return num;
  int x = num % 10;
  return x + digitSum(num / 10);
```

(7) **Recursive tracing**

Trace through the following recursive method given the outputs below:

```
public static int foo(int x, int y)
{

if (x <0 || y < 0) return -1;
if (y == 0) return 0;

return x + foo(y-1, x)
}

foo(5, 7);
foo(6, 1);
```

```
foo(5, 7) -> return 5 + foo(6, 5)
                      > 6 + foo(4,6)
                          > 4 + foo(5,4)
                              > 5 + foo(3,5)
                                  > 3 + foo(4,3)
                                      > 4 + foo(2,4)
                                          > 2 + foo(3,2)
                                              > 3 + foo(1,3)
                                                  > 1 + foo(2,1)
                                                      > 2 + foo(0,2)
                                                          > 0 + foo(1,0)
                                                            0
                                                        2
                                                    3
                                                6
                                            8
                                        12
                                    15
                                20
                            24
                        30
                    35

foo(6, 1);      6 + foo(0, 6)
                     0

                6
```

(8) **Recursion**

Write a recursive method moveToEnd that accepts a String s and a char c as parameters, and returns a new String similar to s but with all occurrences of c moved to the end of the string. The relative order of the other characters should be unchanged from their order in the original string s. If s does not contain c, it should be returned unmodified.

The following table shows calls to your method and their return values.

Call	Value Returned
moveToEnd("hello", 'l')	"heoll"
moveToEnd("hello", 'e')	"hlloe"
moveToEnd("hello there", 'e')	"hllo threee"
moveToEnd("hello there", 'q')	"hello there"
moveToEnd("HELLO there", 'e')	"HELLO three"
moveToEnd("", 'x')	""

```
public static String moveToEnd(String a, char b)
{
   if (a.length() <= 1) return a;
   if (a.charAt(0) == b) return moveToEnd(a.substring(1),b) + b;
   return a.charAt(0) + moveToEnd(a.substring(1),b);
}
```

(9) **RECURSIVE TRACE**

For each call to the following method, indicate what value is returned:

```
public int bar(int x, int y) {
    if (x < 0) {
        return -bar(-x, y);
    } else if (y < 0) {
        return -bar(x, -y);
    } else if (x == 0 && y == 0) {
        return 0;
    }

    return 100 * bar(x / 10, y / 10) + 10 * (x % 10) + y % 10;
} }
```

- bar (5, 7)
  - 100 * bar(0,0) + 10 * 5 + 7
  - 0
  - 100 * 0 + 57
  - 57
- bar (12, 9)
  - 100 * bar(1, 0) + 10 * 2 + 9
  - Bar(1,0) = 100 * bar(0,0) + 10 * 1 + 0
  - Bar(0,0) = 0
  - 100 * 0 + 10
  - 10
  - 100 * 10 + 10 * 2 + 9
  - 1000 + 29
  - 1029
- bar (-7, 4)
  - bar(7,4)
  - 100 * bar(0 , 0) + 10 * (7) + 4
  - -74
- bar (-23, -48)
  - bar(23,-47)
  - bar(23,47)
  - 100 * bar(2 , 4) + 10 * (3) + 8
  - 100 * bar(0 , 0) + 10 * (2) + 4
  - 2438
- bar (128, 343)
  - 100 * bar(12 , 34) + 10 * (8) + 3
  - 100 * bar(1 , 3) + 10 * (2) + 4
  - 100 * bar(0 , 0) + 10 * (1) + 3
  - 132483

10) **Lists, Maps, and Sets**

Write a method that accepts a List of Strings, and a Set of Characters and determines the number of times that each character is found in a word as a map. Do not count doubles.

For Example the set ('b','o') and the List ("book", "love", "born"). Would return a set that maps ['b' = 2 : 'o' = 3]

```
public static Map<Character,Integer> countChar(List<String> l,
                                               Set<Character> s)
  {
    Map<Character,Integer> myMap = new TreeMap<Character,Integer>();
    for (Character y : s)
    {
      myMap.put(y,0);
    }
    System.out.println(myMap);
    for (String x : l)
    {
      for (Character y : s)
      {
        if(x.indexOf(y) >= 0)
        {
          Integer temp =   myMap.get(y);
          myMap.put(y, temp+1);
        }
      }// For Character

    }// For String

    return myMap;
  }// countChar
```

11) **Sorting**

By hand showing each loop through the algorithm.

sort the array **[h, y, e, r, u, a, z, i]**

Using:

- Selection
- Insertion
- Bubble
- Merge Sort

Selection

[h, y, e, r, u, a, z, i]

[a, y, e, r, u, h, z, i]

[a, e, y, r, u, h, z, i]

[a, e, h, r, u, y, z, i]

[a, e, h, i, u, y, z, r]

[a, e, h, i, r, y, z, u]

[a, e, h, i, r, u, z, y]

[a, e, h, i, r, u, y, z]

Insertion

[h, y, e, r, u, a, z, i]

[h, y, e, r, u, a, z, i]

[h, y, e, r, u, a, z, i]

[e, h, y, r, u, a, z, i]

[e, h, r, y, u, a, z, i]

[e, h, r, u, y, a, z, i]

[a, e, h, r, u, y, z, i]

[a, e, h, r, u, y, z, i]

[a, e, h, i, r, u, y, z]

Bubble

[h, y, e, r, u, a, z, i]

[h, e, r, u, a, y, i, z]

[e, h, r, a, u, i, y, z]

[e, h, a, r, i, u, y, z]

[e, a, h, i, r, u, y, z]

[a, e, h, i, r, u, y, z]

[a, e, h, i, r, u, y, z]

Merge

[h, y, e, r, u, a, z, i]

[h, y, e, r] [u, a, z, i]

[h, y] [e, r] [u, a] [z, i]

[h] [y] [e] [r] [u] [a] [z] [i]

[h, y] [e, r] [a, u] [i, z]

[e, h, r, y] [a, i, u, z]

[a, e, h, i, r, u, y, z]

12) **Stacks and Queues**

Imagine that you have a queue called x and a stack called y. What would be the result and final values of each data structure given the following cdoe?

```
1   x.add(100);
2   x.add(200);
3   y.push(300);
4   y.push(400);
5   x.add(y.pop() );
6   y.add(x.remove() );
7   System.out.println(x.peek()  );
8   System.out.println(y.peek()  );
9   x.add(y.pop() +   y.pop());
10  x.add(500);
11  x.add(600);
12  y.push(700);
13  y.push(800);
14 System.out.println(x.remove()  );
15 System.out.println(y.pop()  );
16 x.add(y.peek() +   y.peek());
17 System.out.println(x.remove()  );
18 System.out.println(x.peek()  );
19 System.out.println(y.pop() );
```

Start x = { } y = { }
1 x = {100) y = {}
2 x = {100, 200} y = {}
3 x = {100, 200} y = {300}
4 x = {100, 200} y = {300, 400}
5 x = {100, 200, 400} y = {300}
6 x = {200, 400} y = {300, 100}
7 Print 200
8 Print 100
9 x = {200, 400, 400} y = { }
10 x = {200, 400, 400, 500} y = {}
11 x = {200, 400, 400, 500, 600} y = {}
12 x = {200, 400, 400, 500, 600} y = {700}
13 x = {200, 400, 400, 500, 600} y = {700, 800}
14 Print 200 x = {400, 400, 500, 600} y = {700, 800}
15 Print 800 x = {400, 400, 500, 600} y = {700}
16 x = {400, 400, 500, 600, 1400} y = {700}
17 Print 400 x = {400, 500, 600, 1400} y = {700}
18 Print 400 x = {400, 500, 600, 1400} y = {700}
19 Print 700 x = {400, 500, 600, 1400} y = {}

13) **Stacks and Queues**

Write a method collapse that takes a Stack of integers as a parameter and that collapses it by replacing each successive pair of integers with the sum of the pair. For example, suppose a stack stores this sequence of values:

bottom (7, 2, 8, 9, 4, 13, 7, 1, 9, 10) top

Assume that stack values appear from bottom to top. In other words, 7 is on the bottom, with 2 on top of it, with 8 on top of it, and so on, with 10 at the top of the stack.

The first pair should be collapsed into 9 (7 + 2), the second pair should be collapsed into 17 (8 + 9), the third pair should be collapsed into 17 (4 + 13) and so on to yield:

bottom (9, 17, 17, 8, 19) top

As before, stack values appear from bottom to top (with 9 on the bottom of the stack, 17 on top of it, etc). If the stack stores an odd number of elements, the final element is not collapsed. For example, the sequence:

bottom (1, 2, 3, 4, 5) top

would collapse into:

bottom (3, 7, 5) top

with the 5 at the top of the stack unchanged.

You are to use one queue as auxiliary storage to solve this problem. You may not use any other auxiliary data structures to solve this problem, although you can have as many simple variables as you like. You also may not solve the problem recursively.

In writing your method, assume that you are using the Stack and Queue interfaces and the Stack and LinkedList implementations discussed in lecture. As a result, values will be stored as Integer objects, not simple ints.

Your method should take a single parameter: the stack to collapse.

```
public void collapse(Stack<Integer> st)
      {
            Queue<Integer>  hold= new LinkedList<Integer>();
            int count = 0;
      // Move everything from the stack to the queue
      // and count how many.
            while(!st.empty() )
            {
                  hold.add(st.pop());
                  count++;
            }
      // if odd, then take the extra one and add it to the stack
            if (count % 2 == 1)
            {
             st.push(hold.remove() ) ;
            }
      // move all the sums to the stack
      // note that they will be in the wrong order
            while(!hold.isEmpty() )
            {
                  st.push(hold.remove() + hold. remove ()) ;
            }
      // Move everything to the queue, and then back to the stack
      // to reverse them.
            while(!st.isEmpty() )
            {
                  hold.add(st.pop());

            }
            while(!hold.empty() )
            {
                  st.push(hold.remove()) ;
            }


      }
```

S