Sample Solution

Question 1. (15 points) Arrays. For this problem, we'd like to add a method `nCopies(s)` to the `SortedStringList` class from assignment 1. Method `nCopies(s)` should return the number of copies of string `s` that are contained in the `SortedStringList`, or return 0 if `s` does not appear in the `SortedStringList`.

For full credit, your solution must work as follows:

- You **must** use method `bsearch` (binary search, specification repeated below) to find the location(s) in the list where string `s` is located, if it is present, and
- Your solution **may only** examine list entries that might contain the string `s`, i.e., you should use the information from `bsearch` to limit your search to the part of the array where copies of `s` might appear (for example, you should not search the entire array and count the number of occurrences of `s`), and
- You **may not** call any other methods of class `SortedStringList` or add any instance variables to the class.

Complete method `nCopies` in class `SortedStringList` on the next page.

```
public class SortedStringList {
   // instance variables
   private String[] items;   // strings in this list are
   private int size;         // stored in items[0..size-1]
                             // and the strings are sorted
                             // in non-decreasing order

   // Binary search method bsearch:
   // Search for value in items[lo..hi].  Return a value k
   // such that everything in items[lo..k] is <= value
   // according to compareTo and everything in items[k+1..hi]
   // is greater than value.  Either of these regions might be
   // empty, so if everything in items[lo..hi] is greater than
   // value, we return lo-1, and if everything is less than or
   // equal to value, then we return hi.
   // pre: items is sorted in ascending order and lo<=hi.

   private int bsearch(int lo, int hi, String value) {
      ...
      // implementation omitted
   }
```

(write your answer to this question on the next page)

Sample Solution

Question 1. (continued)

```
class SortedStringList (continued)

   /**
    * Return the number of times a string occurs in this list.
    * @param s the string to search for.
    * @return the number of copies of s in this list, or 0 if
    *         s does not occur in this list.
    */
   public int nCopies(String s) {

      int loc = bsearch(0,size-1,s);
      int nFound = 0;   // # copies found
      while (loc >= 0 && items[loc].equals(s)) {
         nFound++;
         loc—
      }
      return nFound;
   }
```

Question 2. (15 points) Recursive tracing. Consider the following method:

```
   public void mystery(int n) {
      if (n <= 0) {
         System.out.print('a');
      } else if (n%2 == 1) {
         System.out.print((char)('a'+n));
         mystery(n-1);
      } else {
         mystery(n-1);
         System.out.print((char)('a'+n));
      }
   }
```

For each of the following method calls, indicate the output that is produced.

(a) `mystery(0)` **a**
(b) `mystery(1)` **ba**
(c) `mystery(2)` **bac**
(d) `mystery(4)` **dbace**
(e) `mystery(7)` **hfdbaceg**

Sample Solution

Question 3. (15 points) More recursion. For this problem, complete the definition of method `mirrorString` below so that it prints the parameter string `s` both forward and backward on the same line as shown below. Notice in all cases that the final character in the string is only printed once (at the exact middle of the mirrored string) whereas all other characters are printed twice. If `mirrorString` is passed an empty string, it should not print anything. You may define additional private methods if they are helpful.

The output should appear as in the following examples:

Method call	Output produced
`mirrorString("")`	(i.e., none)
`mirrorString("a")`	`a`
`mirrorString("bo")`	`bob`
`mirrorString("How are you?")`	`How are you?uoy era woH`

You may assume that the `String` you are passed is not `null`. The `String` can contain any combination of characters and be of any length.

You **must** use recursion for this problem; you **may not** use any loops.

Some potentially useful `String` methods include:

```
charAt(int i) // returns the char at index i
length()      // returns the length of the String as an int
```

```
// print "mirrored" String s as describe above
public void mirrorString(String s) {
   mirrorStringHelp(s, 0);
}

// print mirror of string s starting at position index
private void mirrorStringHelp(String s, int index) {
   if (index < s.length()) {
      if (index == s.length() - 1)
         System.out.print(s.charAt(index));
      else {
         System.out.print(s.charAt(index));
         mirrorStringHelp(s, index + 1);
         System.out.print(s.charAt(index));
      }
   }
}
```

Here's another solution without a helper function.

```
public void mirrorString(String s) {
   if (s.length() > 0) {
      if (s.length() == 1) {
         System.out.print(s);
      } else {
         System.out.println(s.charAt(0));
         mirrorString(s.substring(1));
         System.out.println(s.charAt(0));
      }
   }
}
```

Sample Solution

Question 4. (20 points) Linked lists. For this problem, we have a class that holds a list of integers. The list is a linked list whose nodes are instances of the following class.

```
public class IntNode {
   public int item;      // data item in this node
   public IntNode next;  // next node in list, or null if none
}
```

Complete the definition of method `contains(n)` below so it returns true if `n` is found in the list and false if not. *In addition*, if the number `n` is found in the list, the `IntNode` containing that number should be moved to the *front* of the list (presumably so if we search for the same number again we will find it faster). Your solution **must** rearrange the existing nodes if a number is found. You **may not** create new nodes or change the `item` field in any of the nodes. You **may not** add any instance variables to the class or call any other methods. Don't worry about duplicate values in the list – if `n` appears more than once, return `true` and move one of the nodes containing `n` to the front of the list.

```
public class IntList {
   private IntNode front;   // first node in the list or
                            // null if list is empty
   /** Search for a value in this list
    *  @param n number to search for
    *  @return true if n is in the list, false if not
    */
   public boolean contains(int n) {
      // if n is found, also move the IntNode containing n to
      // the front of the list
      // base cases: empty list, or n found at front
      if (front == null) {
         return false;
      } else if (front.item == n) {
         return true;
      } else {
         // general case - if n is found, it is not first
         IntNode prev = front;
         while (prev.next != null) {
            if (prev.next.item == n) {
               // found it - move it to front & return true
               IntNode newFront = prev.next;
               prev.next = prev.next.next;
               newFront.next = front;
               front = newFront;
               return true;
            } else {
               // not here - advance to next node
               prev = prev.next;
            }
         }
         // not found
         return false;
      }
   }
```

Sample Solution

Question 5. (15 points) Stacks and Queues. In this problem, we start with a queue of characters that contains nested parentheses `(` and `)`, and brackets `[` and `]`. We would like to replace all the `[` and `]` characters in the queue with ordinary parentheses. A `[` should be replaced by a single `(`. However, a `]` is a "super right parenthesis". A single `]` matches the corresponding `[` that precedes it, *and* it also matches any unmatched simple left parentheses `(` that appear between the `]` and the previous matching `[`. In other words, a single `]` should be replaced by one or more `)`s – one to replace the `]` itself plus 0 or more to match all of the unclosed `(`s that appear between the `]` and the preceding, matching `[`. Here are a few examples:

Before	After
`((()(())))`	`((()(()))`
`[]`	`()`
`[(()`	`((()))`
`([()(()])`	`((()(()))))`

Complete the definition of method `expand` on the next page. The parameter to `expand` is a single queue of characters containing the original parentheses and brackets. When `expand` is done executing, the queue should contain a only parentheses, where all brackets have been replaced appropriately. You may assume that the input is properly formatted and does not contain excess or unbalanced characters – you do not need to worry about throwing exceptions for input errors.

Method `expand` contains a local variable that is a stack of characters. This stack is initially empty and can be used as you wish. You may declare additional simple variables, but you **may not** use any additional lists, stacks, queues, files, or other containers in your solution.

Recall:

- Operations on a stack: `top()` (return top without deleting), `pop()`, `push(item)`, `isEmpty()`.
- Operations on a queue: `enqueue(item)`, `dequeue()`, `size()`, `isEmpty()`.

Continued on the next page.

Sample Solution

Question 5. (cont.) Complete the definition of method `expand` below.

```
/**
 * Replace all []s in the queue with ()s, expanding ]s into
 * one or more )s, depending on the number of preceding
 * unmatched (s in the queue.
 * @param q the queue of characters
 */
public void expand(Queue<Character> q) {

   // local variables -- add more simple variables if needed
   Stack<Character> stk = new ArrayStack<Character>();

   // Strategy: Cycle through the queue.  When we see a ( or [,
   // put a ( in the queue, and push the ( or [ onto the stack.
   // When we see a ), put a ) in the queue and pop the stack
   // (which contains a ( on top, since the input is assumed to
   // be ok).  When we see a ], put )s in the queue and pop the
   // stack until we've popped the corresponding [

   int originalSize = q.size();
   for (int k = 0; k < originalSize; k++) {
      char ch = q.dequeue();
      if (ch == '(' || ch == '[') {
         q.enqueue('(');
         stk.push(ch);
      } else if (ch == ')') {
         q.enqueue(')');
         char x = stk.pop();
      } else { // ch == ']'
         do {
            q.enqueue(')');
            char ctop = stk.pop();
         } while (ctop != '[');
      }
   }
}
```

Some solutions kept a simple counter of the number of left parentheses and decremented that to determine how many right parentheses to add to the queue when a right bracket was encountered. That works as long as there are no unmatched left parentheses or brackets preceding the nearest left bracket; it does not work in general.

Sample Solution

Question 6. (20 points) Inheritance. Consider the following class definitions.

```
public class Pepsi extends Soda {
   public void m2() {
      System.out.println("Pepsi2");
      m1();
   }
}

public class Soda {
   public void m1() {
      System.out.println("Soda1");
   }
}

public class DietCoke extends Coke {
   public void m1() {
      System.out.println("DietCoke1");
   }
}

public class Coke extends Soda {
   public void m2() {
      System.out.println("Coke2");
      super.m1();
   }

   public void m3() {
      System.out.println("Coke3");
      m1();
   }
}
```

Now assume that the following variables have been declared and initialized.

```
Soda v1 = new Coke();
Soda v2 = new Pepsi();
Soda v3 = new DietCoke();
Object v4 = new Soda();
Coke v5 = new DietCoke();
Coke v6 = new Coke();
```

(continued next page – you may tear this page out for reference if it is convenient, but don't separate the next page from the rest of the test.)

Sample Solution

Question 6. (cont)

In the table below, indicate in the right-hand column the output produced by the statement in the left-hand column. If the statement produces more than one line of output, indicate the line breaks with slashes as in "a/b/c". If the statement causes an error, fill in the right-hand column with either the phrase "compile error" or "runtime error" to indicate when the error would be detected.

Statement	Output
`v1.m2();`	`compile error - no m2() in Soda`
`v2.m2();`	`compile error - no m2() in Soda`
`v3.m2();`	`compile error - no m2() in Soda`
`v4.m2();`	`compile error - no m2() in Object`
`v5.m3();`	`Coke3 / DietCoke1`
`v6.m3();`	`Coke3 / Soda1`
`((Object)v2).m1();`	`compile error - no m1() in Object`
`((Soda)v5).m1();`	`DietCoke1`
`((Pepsi)v4).m2();`	`runtime error - cast fails`
`((Soda)v4).m2();`	`compile error - no m2() in Soda`