CS 0445 Data Structures
Midterm-exam Practice Question SOLUTIONS

Before looking at these solutions, I strongly recommend that you give all of the questions a "good try".

Fill in the Blanks

1) Abstract Data Types (ADTs) can be thought of as an encapsulation of _____**data**_____ and _____**operations**_____.
2) The two primary techniques for building a new class in Java are through _____**composition**_____ (use components from previously defined classes) and _____**inheritance**_____ (extend a previously defined class).
3) If a method in a superclass is redefined in a subclass using the identical method signature, we say that we are _____**overriding**_____ the method
4) The two ADT Bag implementations that we discussed in detail had _____**an array (or dynamic array)** _____ and _____**a linked list**_____ as their underlying data implementations.
5) A program that executes $4N^2 + 100N + 10000$ operations for a problem of size N has a Big-O runtime of ___**$O(N^2)$**_____.
6) In the ArrayBag class the add() method has a worst case run-time of ____**O(1)**_____ and in the LinkedBag class the toArray() method has a worst case run-time of _____**O(N)**_____.
7) Any recursive method needs one or more _____**base cases**_____ and one or more _____**recursive cases**_____.
8) Any recursive method that is _____**tail recursive**_____ can easily be converted into a nonrecursive equivalent method.

True/False (explain why false answers are false)

1) Java primitive types can be stored in ArrayBags.
   FALSE – ArrayBags store only Objects (and subclasses thereof). If you want to store primitive values, you must use the wrapper classes (ex: Integer).

2) Assuming the BagInterface<T> interface and the LinkedBag<T> class as we discussed, the following statement is legal:
   ```
   BagInterface<String> L = new LinkedBag<String>();
   ```
   TRUE

3) Assuming the BagInterface<T> interface as we discussed, the following statement is legal:
   ```
   BagInterface<String> L = new BagInterface<String>();
   ```
   FALSE – We cannot make objects of interfaces.

4) Binary Search has a worst case run-time of O(N).
   FALSE – Binary Search has a worst case run-time of $O(\log_2 N)$.

5) The data in a **singly linked list** is **contiguous**.
FALSE – A fundamental property of linked list is that the data is NOT contiguous, but, rather, in arbitrary locations.

6) The method add(newEntry) is **more efficient** with an LinkedBag implementation than with an ArrayBag implementation.
FALSE. They both have a running-time of O(1).

7) Recursion is implemented utilizing **activation records** and the **run-time queue (RTQ).**
FALSE – it utilizes the run-time stack.

Short Answers

1) In class we discussed two variations for resizing a dynamic array list:
   1. Increase the size of the array by 1
   2. Double the size of the array

   Explain which of these is preferable and why. Be specific, using mathematical justification. You do not have to give all of the mathematical details for b) but you must give the general idea.

   It is preferable to double the size of the array rather than to increase it by one. If we increase the size by one, we must resize at each add(). Over a sequence of N adds, this will cause us to do 1 + 2 + 3 + … + N assignments, which is a total of O(N^2). Amortized over the N adds this gives us a time of O(N) per add. If we double the array size when we resize, during most adds we only have to do one assignment, and only occasionally have to resize (when N is 1 more than a power of 2). This evaluates to 2N-1 total assignments (why?), which gives us an amortized time of O(1) per add.

2) In implementing our LinkedBag class the text authors made the Node class an **inner class**. Explain what an inner class is and why the Node class was implemented in this way.

   An inner class is a Java class that is defined within another class. Its data and methods, even private ones, are accessible to the outer class. This is useful for our LinkedBag because it allows the LinkedBag methods to access the next and data fields of Node objects without requiring accessor and mutator methods, yet it does not allow a client to access them (since they are still private).

Traces

1) Give ALL output produced by the execution of the Java program below. To avoid ambiguity, clearly mark your output by **drawing a box around it.**

```
public class PracTrace1
{
       int data;

        public PracTrace1(int d)
          { data = d; }

        public void change(int newdata)
          { data = newdata;         }

        public String toString()
        {    return new String("Data: " + data);   }

        public static void main(String [] args)
        {
               PracTrace1 [] A1 = new PracTrace1[4];
              A1[0] = new PracTrace1(10);
              A1[1] = new PracTrace1(20);
              A1[2] = new PracTrace1(30);
              A1[3] = new PracTrace1(40);

              showData(A1);

              PracTrace1 [] A2 = A1;
              PracTrace1 [] A3 = new PracTrace1[A1.length];

              for (int i = 0; i < A1.length; i++)
                    A3[i] = A1[i];

              A2[1].change(25);
              A2[2] = new PracTrace1(35);
              A1[2].change(77);
              A3[0].change(15);
              A3[2].change(88);
              A3[3] = new PracTrace1(45);

              showData(A1); showData(A2); showData(A3);
       }

      public static void showData(PracTrace1 [] A)
      {
          for (int i = 0; i < A.length; i++)
             System.out.println(A[i]);
          System.out.println();
      }

}
```

OUTPUT

```
Data: 10
Data: 20
Data: 30
Data: 40

Data: 15
Data: 25
Data: 77
Data: 40

Data: 15
Data: 25
Data: 77
Data: 40

Data: 15
Data: 25
Data: 88
Data: 45
```

COMMENTS

Note that references A1 and A2 share the same object, so their output is identical. Even though A3 is a separate array, it shares objects with A1, so mutations on them will affect both arrays unless a new object is assigned to the location. Download and run this program and experiment with it to better understand the solution.

2) Give ALL output produced by the execution of the Java program below. To avoid ambiguity, clearly mark your output by **drawing a box around it.**

```
public class PracTrace
{
      static int [] A = {50, 40, 80, 60, 90};
      int workingSum = 0;

      public PracTrace()
      {
            workingSum = 0;
            int sum = recGetSum(A, 0);
            System.out.println("The final working sum is " + workingSum);
            System.out.println("The overall sum is " + sum);
      }

      public int recGetSum(int [] A, int loc)
      {
            if (loc < A.length)
            {
                  workingSum = workingSum + A[loc];
                  System.out.println("loc = " + loc + " working sum = " +
                                          workingSum);
                  int theSum = A[loc] + recGetSum(A, loc + 1);

                  workingSum = workingSum - A[loc];
                  System.out.println("loc = " + loc + " working sum = " +
                                          workingSum);
                  return theSum;
            }
            else
                  return 0;
      }

      public static void main(String [] args)
      {
            PracTrace P = new PracTrace();
      }
}
```

OUTPUT:	**COMMENTS:**
loc = 0 working sum = 50 loc = 1 working sum = 90 loc = 2 working sum = 170 loc = 3 working sum = 230 loc = 4 working sum = 320 loc = 4 working sum = 230 loc = 3 working sum = 170 loc = 2 working sum = 90 loc = 1 working sum = 50 loc = 0 working sum = 0 The final working sum is 0 The overall sum is 320	Note how the recursive calls terminate in the reverse order of their calls. Also note how the items are "removed" from the working sum as the calls terminate.

Coding

1) A method that is useful in the ArrayBag class is the method
`private int getIndexOf(T anEntry)`
This method iterates through the array and returns the index of the first location whose data matches **anEntry**. If **anEntry** is not found the method should return -1. Complete this method below.
Recall that the ArrayBag class has two instance variables:

```
private T [] bag;
private int numberOfEntries;
```

Answers vary. One is given below:

```
private int getIndexOf(T anEntry)
 {
        int where = -1;
        boolean found = false;
       for (int index = 0; !found && (index < numberOfEntries);
            index++)
        {
             if (anEntry.equals(bag[index]))
             {
                found = true;
                where = index;
             } // end if
        } // end for

        return where;
 } // end getIndexOf
```

2) A method that is useful in the LinkedBag class is the method

`private Node getReferenceTo(T anEntry)`

This method iterates through the list and returns a reference to the first Node whose data matches **`anEntry`**. If **`anEntry`** is not found the method should return null. Complete this method below. Recall some important declarations in the LinkedBag class:

```
private Node firstNode;
private int numberOfEntries;

 private class Node
 {
   private T data;
   private Node next;
   // methods omitted
 }
```

Answers vary. One is given below:

```
 private Node getReferenceTo(T anEntry)
 {
   boolean found = false;
   Node currentNode = firstNode;

         while (!found && (currentNode != null))
         {
               if (anEntry.equals(currentNode.data))
                     found = true;
               else
                     currentNode = currentNode.next;
         } // end while

         return currentNode;
 } // end getReferenceTo
```