CSE 142, Spring 2009, Final Exam Sample Solutions

1. Array Simulation

```
Original Contents of Array          Final Contents of Array
--------------------------          -----------------------
{}                                  {}
{7}                                 {7}
{3, 1}                              {3, 3}
{5, 6, 8, 100}                      {5, 5, 10, 30}
{0, 2, 4, 0, 10}                    {0, 0, 0, 0, 0}
```

```
2. Inheritance

some sort of animal
motion: unknown

a crow
motion: flies

a sea creature with some tentacles
motion: swims

a sea creature with 8 tentacles
motion: swims

a sea creature with many tentacles
motion: swims
```

3. Parameters and References

```
Line 1: 4 [9, 0, 0, 14]

Line 2: 2 [9, 0, 0, 14]

Line 3: 8 [9, 9, 14, 14]

Line 4: 4 [9, 9, 14, 14]
```

4. Token-Based File Processing

```
    public static void printStatistics(Scanner s) {
        int intSum = 0;
        double doubleSum = 0.0;
        int totalTokens = 0;
        while (s.hasNext()) {
            if (s.hasNextInt()) {
                intSum = intSum + s.nextInt();
            } else if (s.hasNextDouble()) {
                doubleSum = doubleSum + s.nextDouble();
            } else {
                s.next();
            }
            totalTokens++;
        }
        System.out.println("int total: " + intSum);
        System.out.println("double total: " + doubleSum);
        System.out.println("total number of tokens: " + totalTokens);
    }
```

5. Line-Based File Processing

```
public static double netChange(Scanner input, int selectedAcctNumber) {
    double change = 0.0;
    while (input.hasNextLine()) {
        String line = input.nextLine();
        Scanner lineScan = new Scanner(line);
        String action = lineScan.next();
        if (action.equals("deposit")) {
            int acctNumber = lineScan.nextInt();
            if (acctNumber==selectedAcctNumber) {
                change += lineScan.nextDouble();
            }
        } else if (action.equals("withdraw")) {
            int acctNumber = lineScan.nextInt();
            if (acctNumber==selectedAcctNumber) {
                change -= lineScan.nextDouble();
            }
        }
    }
    return change;
}
```

6. Arrays

```
public static double[] averageWithNeighbors(double[] input) {
    double[] result = new double[input.length];
    result[0] = (input[0] + input[1]) / 2;
    for(int i=1; i < input.length-1; i++) {
      result[i] = (input[i-1] + input[i] + input[i+1]) / 3;
    }
    result[input.length-1] = (input[input.length-2] + input[input.length-1]) / 2;
    return result;
  }
```

Another solution that is worse style, but we don't grade style:

```
public static double[] averageWithNeighbors(double[] input) {
    double[] result = new double[input.length];
    for(int i=0; i < input.length; i++) {
      double a = 0.0;
      double b = input[i];
      double c = 0.0;
      int num = 1;
      if(i-1 >= 0) {
        a = input[i-1];
        num++;
      }
      if(i+1 < input.length) {
        c = input[i+1];
        num++;
      }
      result[i] = (a + b + c) / num;
    }
    return result;
  }
```

7. Arrays

```
   public static int[] interleave(int[] a, int[] b) {
        int[] c = new int[a.length + b.length];
        // in marches through a and b
        // out marches through c
        int in = 0;
        int out = 0;
        while (in<a.length || in<b.length) {
            if (in<a.length) {
                c[out] = a[in];
                out++;
            }
            if (in<b.length) {
                c[out] = b[in];
                out++;
            }
            in++;
        }
        return c;
    }
```

An alternate version with multiple loops

```
  public static int[] interleave(int[] a, int[] b) {
    int[] c = new int[a.length+b.length];
    for(int i=0; i < a.length && i < b.length; i++) {
      c[2*i] = a[i];
      c[2*i+1] = b[i];
    }
    // at most one of the next two loops will execute > 0 times
    // (putting them in ifs is perhaps clearer but unnecessary)
    for(int i=a.length; i < b.length; i++) {
      c[i+a.length] = b[i];
    }
    for(int i=b.length; i < a.length; i++) {
      c[i+b.length] = a[i];
    }
    return c;
  }

  // like the previous solution but simpler to compute
  public static int[] interleave(int [] a, int[] b) {
    int[] c = new int[a.length + b.length];
    int out = 0;
    for(int i=0; i < a.length && i < b.length; i++) {
      c[out] = a[i];
      out++;
      c[out] = b[i];
      out++;
    }
    // at most one of the next two loops will execute > 0 times
    // (putting them in ifs is perhaps clearer but unnecessary)
    for(int i=a.length; i < b.length; i++) {
      c[out] = b[i];
      out++;
    }
    for(int i=b.length; i < a.length; i++) {
      c[out] = a[i];
      out++;
    }
    return c;
  }
```

```
// This version copies one input array over before the other, ugly but works

  public static int[] interleave(int[] a, int[] b) {
    int[] c = new int[a.length + b.length];
    if(a.length < b.length) {
      for(int i=0; i < a.length; i++) {
        c[2*i] = a[i];
      }
      for(int i=0; i < b.length; i++) {
        if(i < a.length) {
          c[2*i+1] = b[i];
        } else {
          c[a.length + i] = b[i];
        }
      }
    } else {
      for(int i=0; i < a.length; i++) {
        if(i < b.length) {
          c[2*i] = a[i];
        } else {
          c[b.length+i] = a[i];
        }
      }
      for(int i=0; i < b.length; i++) {
        c[2*i+1] = b[i];
      }
    }
    return c;
  }
```

8. Programming

```
    public static void compress(ArrayList<String> words) {
        int index = 0;
        while (index<words.size()) {
            int i = index+1;
            int count = 1;
            while (i<words.size() && words.get(index).equals(words.get(i))) {
                words.remove(i);
                count++;
            }
            if (count>1) {
                words.set(index, count + "*" + words.get(index));
            }
            index++;
        }
    }

// this more complicated version finds how much to do first
    public static void compress(ArrayList<String> words) {
        int index = 0;
        while (index<words.size()) {
          String s = words.get(index);
          int count = 1;
          while(count + index < words.size() &&
                s.equals(words.get(count + index))) {
            count++;
          }
          for(int i=0; i < count; i++) {
            words.remove(index);
          }
          String newString = s;
          if(count>1) {
            s = count + "*" + s;
          }
          words.add(index,s);
          index++;
        }
    }

  // this way uses a second ArrayList to avoid shifting
  // (not what we intended, but an acceptable solution)
  public static void compress(ArrayList<String> words) {
    int index = 0;
    ArrayList<String> compressed = new ArrayList<String>();
    while(index < words.size()) {
      String s = words.get(index);
      int count = 1;
      while(index + count < words.size() && s.equals(words.get(index+count))) {
        count++;
      }
      String newS = s;
      if(count > 1) {
        s = count + "*" + s;
      }
      compressed.add(newS);
      index += count;
    }
    words.clear();
    for(int i=0; i < compressed.size(); i++) {
      words.add(compressed.get(i));
    }
  }
```