Final Exam Key
Summer 2019

1. 44,22 [44, 77] 0
   44,22 [44, 77] 55
   22,22 77

2. [1, 2, 3]
   [2, 2, 3, 4]
   [6, 3, 3, 5, 7]
   [-1, 1, 3, 0, -3]
   [7, 5, 3, 0, 6, 12]

3. Seal Bluth
   Bluth 2
   Bluth 1
   Bluth 2

   Bluth
   Gob 1
   Gob 2

   Bluth
   Bluth 2
   Bluth 1
   Bluth 2

   Buster
   Buster 2
   Bluth 1
   Buster 2

4. One sample solution shown below:

```
public static void redact(Scanner input) {
   while (input.hasNext()) {
      String next = input.next();
      if (next.equals("CLASSIFIED")) {
         int count = input.nextInt();
         for (int i = 0; i < count; i++) {
            System.out.println("[redacted]");
            input.next();
         }
      } else {
         System.out.println(next);
      }
   }
}
```

5. One sample solution shown below:

```
public static void underline(Scanner input) {
   while (input.hasNextLine()) {
      String text = input.nextLine();
      if (!text.startsWith(".")) {
         System.out.println(text);
      } else {
         System.out.println(text.substring(1));
         for (int i = 1; i <= text.length() - 1; i++) {
            System.out.print("-");
         }
         System.out.println();
      }
   }
}
```

6. One sample solution shown below:

```
public static int minGap(int[] list) {
   if (list.length < 2) {
      return 0;
   } else {
      int min = list[1] - list[0];
      for (int i = 2; i < list.length; i++) {
         int gap = list[i] - list[i - 1];
         if (gap < min) {
            min = gap;
         }
      }
      return min;
   }
}
```

7. Three possible solutions shown below:

```
   // one loop, going forward, needs i++/i--
   public static void cloneOddsRemoveEvens(ArrayList<Integer> numbers) {
      for (int i = 0; i < numbers.size(); i++) {
         int n = numbers.get(i);
         if (n % 2 == 0) {
            numbers.remove(i);
            i--;
         } else {
            numbers.add(i, n);
            i++;
         }
      }
   }
   // one loop, going backwards, doesn't need i++/i--
   public static void cloneOddsRemoveEvens(ArrayList<Integer> numbers) {
      for (int i = numbers.size() - 1; i >= 0; i--) {
         int n = numbers.get(i);
         if (n % 2 == 0) {
            numbers.remove(i);
         } else {
            numbers.add(i, n);
         }
      }
   }
```

```
// two loops, going forward, need i++/i--, no if required in second loop
// because remove is done first
public static void cloneOddsRemoveEvens(ArrayList<Integer> numbers) {
   for (int i = 0; i < numbers.size(); i++) {
      if (numbers.get(i) % 2 == 0) {
         numbers.remove(i);
         i--;
      }
   }
   for (int i = 0; i < numbers.size(); i += 2) {
      numbers.add(i + 1, numbers.get(i));
   }
}
```

8. One possible solution shown below:

```
public class Orca extends Critter {
   int count;

   public Orca() {
      count = 0;
   }

   public Action getMove(CritterInfo info) {
      if (count % 6 == 4 || count % 6 == 5) {
         count++;
         return Action.LEFT;
      } else if (info.getFront() != Neighbor.EMPTY) {
         return Action.INFECT;
      } else {
         count++;
         return Action.HOP;
      }
   }

   public String toString() {
      if (count % 6 < 4) {
         return "M";
      } else {
         return "T";
      }
   }
}
```

9. Two possible solutions shown below:

```
// loop bounds is half the length and uses 2 * i
public static int[] expand(int[] data) {
   int len = 0;
   for (int i = 0; i < data.length / 2; i++) {
      len = len + data[2 * i];
   }
   int[] result = new int[len];
   // keeps tracks of index which increments by one for each value
   int index = 0;
   for (int i = 0; i < data.length / 2; i++) {
      int times = data[2 * i];
      int n = data[2 * i + 1];
      for (int j = 0; j < times; j++) {
         result[index] = n;
         index++;
      }
   }
   return result;
}

// Uses i+=2 to look at every other value
public static int[] expand(int[] data) {
   int len = 0;
   for (int i = 0; i < data.length; i += 2) {
      len += data[i];
   }
   int[] result = new int[len];
   // keeps track of the beginning index of each "chunk"
   int index = 0;
   for (int i = 0; i < data.length; i += 2) {
      int times = data[i];
      int n = data[i + 1];
      for (int j = 0; j < times; j++) {
         result[j + index] = n;
      }
      index += times;
   }
   return result;
}
```

10. Two possible solutions shown below:

```
// nested loop approach
public static int[] findIndexes(int n, int[] data) {
   int[] result = new int[n];
   for (int i = 0; i < result.length; i++) {
      int index = -1;
      for (int j = 0; j < data.length; j++) {
         if (data[j] == i && index == -1) {
            index = j;
         }
      }
      result[i] = index;
   }
   return result;
}

// single loop over data setting values in result
public static int[] findIndexes(int n, int[] data) {
   int[] result = new int[n];
   for (int i = 0; i < result.length; i++) {
      result[i] = -1;
   }
   for (int i = 0; i < data.length; i++) {
      int next = data[i];
      if (next >= 0 && next < n && result[next] == -1) {
         result[next] = i;
      }
   }
   return result;
}
```